

# Decomposition of a 2D polygon into a minimal set of disjoint primitives

H. Lipson      M. Shpitalni

## Abstract

This paper presents a method based on artificial intelligence (AI) techniques for decomposing a two-dimensional polygon into a minimal set of disjoint primitives. Decomposition is accomplished using a search algorithm based on recursively splitting the given shape into smaller parts until a primitive is recognized. The method operates in a primitive-oriented environment, where each primitive type must provide three basic member functions, namely (a) a function for recognizing a shape as the primitive, (b) a function for suggesting new splitting curves for a given shape, and (c) a function for optimistically estimating the number of primitives required to tile a shape. Optimality in the number of decomposition units can be guaranteed subject to the domain spanned by function (b). The proposed method produces a hierarchical decomposition of the shape. Examples from a working implementation are shown.

## Introduction

The need for converting from Boundary representation (B-Rep) to Constructive Solid Geometry representation (CSG) has arisen in many geometric modelling and image processing applications and has motivated both practical and theoretical interest. Rigid three-dimensional homogeneous solids or two-dimensional shapes may be modelled as sets of compact, regular and semi-analytic points. Requicha [1] describes six families of informationally complete representation schemes. Of these, CSG and B-Rep are most widely used. While transformation from CSG to B-Rep is relatively well understood [2], the reverse is not. The need for B-Rep to CSG conversions arises mainly in solid modelling packages, where alternative representations allow application of various representation-specific technologies, more versatile user-interface options and more compact storage.

In general, there are three approaches to the problem of B-Rep to CSG conversion. The first approach converts the boundary to a set of half-spaces joined by Boolean algebra. Such algorithms have been proposed both for two-dimensional polygons [3] and for three-dimensional solids [4, 5]. The second approach describes the object's geometry as the difference between its convex hull and its concavities.

The process is repeated recursively until only convex components remain. These, in turn, are described as a union of half-spaces corresponding to their faces. Such algorithms are applied to two-dimensional shapes [6] and three-dimensional solids [7]. The third approach uses decomposition techniques [8,9]. These can be divided into partitioning methods and covering methods. Partitioning methods subdivide a shape into non-overlapping convex pieces, whereas covering techniques allow overlapping of the components and thus usually take advantage of collinear edges. Tiling problems in combinatorial geometry [10] are also related; however, they usually do not focus on minimal sets or multiple tile types (primitive shapes). Dynamic programming techniques have also been used [11].

The approach proposed in this paper can be classified as a partitioning decomposition method. A given two-dimensional polygon is partitioned into a finite set of non-overlapping primitives or subtracted primitives. The partitioning provides a general framework into which various primitives and target functions can be "plugged in". As output, the given polygon is tiled with a set of the given primitives, according to the given target function. The tiling approach provides a form of CSG decomposition of the given shape. Moreover, it can also be used in other applications, such as computing geometrical properties like the area of a polygon while allowing simple manual verification, two-dimensional geometrical disjoint feature recognition, and meshing.

In this paper, the basic problem and solution approach are outlined. Then, the algorithm structure and implementation are described in detail, and examples from a working implementation are provided.

## **Problem and solution approach**

The goal of the decomposition process is to break down a given convex or concave polygon into a minimal set of predefined primitives. The polygon is constructed from straight line segments arranged in a convex or concave contour. More complex geometries, such as multiple or nested contours, can be decomposed by treating each of their contours separately. The primitives themselves are also polygons with specific properties.

Decomposition is achieved by recursively splitting the given shape into smaller parts until a primitive is recognized, as illustrated in Figure 1. Note that although each polygon can be split in an infinite number of ways, the number of topologically different splits is limited. A split is considered topologically different from another split if it creates a different number or arrangement of segments or if the segments exhibit some geometrical regularity such as collinearity or parallelism.

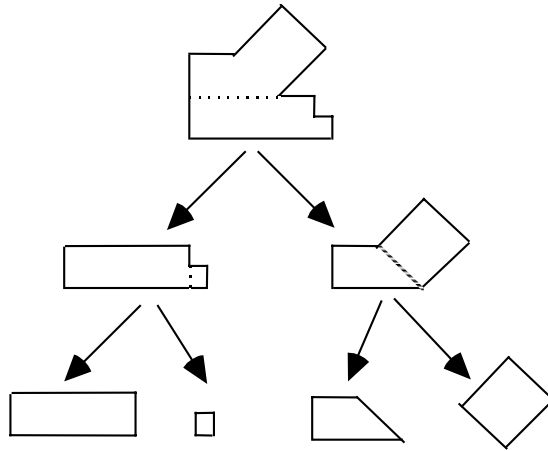


Figure 1. Recursive splitting down to the primitive level.

Figure 2 illustrates 16 different split possibilities for a rectangle. The last two are considered different only because they exhibit the parallelism regularity. The splitting problem can therefore be transformed from an infinitely continuous domain into a discrete problem space, making it a combinatorial optimization problem.

While the number of options still remains large, various well established AI search techniques can now be applied. For a survey of search techniques, refer to [12,13]. Relevant search methods are described briefly here.

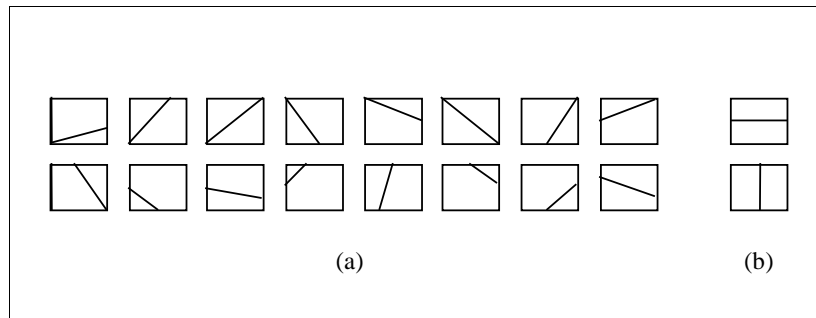


Figure 2. (a) 16 topologically different ways to split a rectangle.  
 (b) Two are geometrically (not topologically) special.

## Search Methods

Search is a universal problem-solving technique in artificial intelligence. It systematically explores alternative sequences of steps required for solving a problem. Each step leads from one state to the next in the problem-state domain. All that is required is a set of states, a set of legal operators, an initial state, and a goal criterion. For polygon decomposition, the initial state is the original polygon and the legal operator is splitting. Activating the splitting operator on the initial state produces several alternative new states, which, in turn, may be subject to further splitting. A final state in the decomposition problem is a state where all the "pieces" are identified as primitives, and the goal is to minimize the number of primitives. Figure 3 illustrates this process, from the initial state through generation of intermediate states to the final goal states.

### Brute force techniques

The most general search algorithms are termed *brute force* searches since they do not require any domain-specific knowledge. Thus, for brute force decomposition, two basic functions are required:

- A function to provide splitting operators (and execute them)
- A function to recognize and count primitives in a solution state.

The most important brute force search techniques are breadth-first, depth-first, iterative deepening and bi-directional searches. These techniques differ in time and space complexities, but they will eventually scan the complete problem domain before they can declare the optimal solution. Due to the large number of splitting operators possible in the polygon decomposition problem, however, a full scan is impractical.

### Heuristic Search

All brute force searches are inefficient because they are essentially blind searches. They use no knowledge about which nodes (decomposition states) should be expanded next. A heuristic search, in contrast, takes advantage of information available in the problem domain to differentiate between the likelihood of various states to lead to the goal state.

This information, often termed a *heuristic evaluation function*, estimates the distance between two states in the problem domain. The distance can usually be estimated without actually determining the path between the states and can therefore be evaluated at relatively low computational cost. In the polygon decomposition problem domain, a

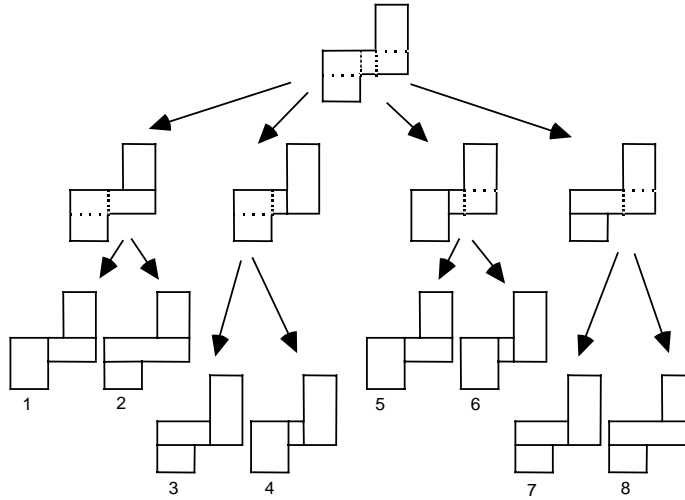


Figure 3. Decomposition states leading from the initial polygon to several decomposition alternatives.

heuristic evaluation function estimates the number of primitives required to tile a given shape without actually finding them.

### A\* Heuristic Search

One of the most established heuristic search algorithm is A\* (A-Star) [13]. In this algorithm, each state is assigned a value corresponding to its likelihood to lead to a solution. This value takes the form  $f(n) = g(n) + h(n)$ , where  $g(n)$  is the actual cost of the path leading from the initial state to node (state)  $n$  and  $h(n)$  estimates the lowest cost from node  $n$  to a final goal state. The search process always selects and expands the state with the lowest  $f(n)$ . In the polygon decomposition problem, new states are generated by splitting the basic state in various ways. The figure of merit associated with each state  $f(n)$  is the number of pieces already identified as primitives  $g(n)$  plus an estimation of the number of primitives required to tile the remaining pieces which have not as yet been identified as primitives  $h(n)$ .

A\* always finds an optimal path to a goal (lowest cost path) if the heuristic function  $h(n)$  does not overestimate the actual cost to the goal [14]. The first solution found is the optimal solution, provided that the cost function  $f(n)$  is *monotonic*, i.e., it is *consistent* by obeying the triangle inequality of metrics. This requirement ensures that A\* will expand paths in non-decreasing order of cost, and thus once A\* finds a path to a goal node, it is the lowest cost path.

To conclude, the three functions necessary to execute a heuristic search in the polygon decomposition problem domain are as follows:

- A function to estimate the number of primitives required to tile a polygon. This function must be monotonic and should never overestimate the real number.
- A function to provide splitting operators (and execute them).
- A function to recognize and count primitives in a solution state.

## Implementation

### Primitive member functions

To facilitate introduction of primitive-specific knowledge, "private" functions are associated individually with each primitive type. In an object-oriented sense, a primitive object must provide three *member functions*: functions for split-suggestions, functions for heuristic estimation, and functions for primitive identification.

### The split-suggestions function

The split-suggestions function proposes how to split the shape. The routine should suggest none, one or more split lines. Primitive-specific information is used to select the most promising possibilities. The following rules are observed:

1. *All split lines must start at a polygon vertex, thus reducing the number of segments in the split pieces.*
2. *Split lines that are collinear with other segments of the polygon boundary are preferable. This criterion increases the effectiveness of tiling primitives, since a primitive edge may match more than one edge of the polygon.*
3. *Split lines exhibiting some geometrical regularities which also exist in a primitive are preferable. For example, for a rectangular primitive, split lines creating  $90^\circ$  angles with one or more of the contour segments are preferable. For a trapezoid primitive, split lines parallel to one or more of the contour segments are preferable. A triangle has no preferences in this respect.*

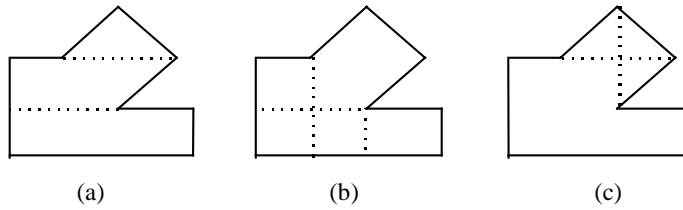


Figure 4. (a) Split lines preferred by collinearity. (b) Split lines preferred by 90° criterion. (c) Some split lines suggested by a triangle object.

Split line suggestions are "collected" from each of the primitive objects and are considered as operators to advance the current decomposition state in the decomposition problem domain. Figure 4 illustrates some split suggestions.

### Heuristic estimation function

The function guiding the heuristic search is the estimation function. This function forecasts the number of primitives necessary to tile a given polygon. It should not overestimate the actual number but should approach it from below as closely as possible. As the estimation approaches the real number, the number of nodes expanded by the heuristic function decreases.

If the  $N$ -sided polygon is already in the form of a primitive, the estimation function should return "1"; otherwise, at least "2" is necessary. Primitives used to tile a polygon must have at least one edge in common. Consequently, if a primitive with  $n$  sides is used as a tile, it may match at most  $(n-1)$  segments of the  $N$ -sided polygon contour. Therefore,  $N/(n-1)$  primitives can be expected, and the minimum number of primitives  $k$  necessary to tile a polygon with  $N$  sides is at least

$$k = \max\left(2, \frac{N}{(n-1)}\right)$$

if the polygon is not a primitive itself. The assumption that each primitive matches  $(n-1)$  segments of the polygon contour is equivalent to the best case and thus ensures that  $k$  is not overestimated.

The proposed heuristic estimation function is monotonic. Refer again to the  $N$ -sided polygon. The estimated number of primitives is given by the above equation. If this polygon is split into two new polygons, the total number of contour segments is increased by at least 2 (if the split

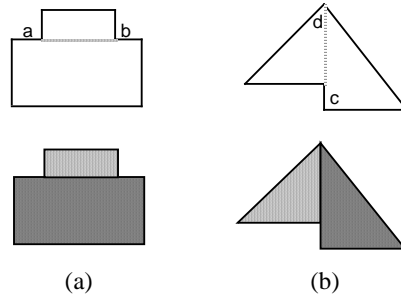


Figure 5. (a) 4 collinear points require a reduction of 2 in the number of sides. (b) 3 collinear points require a reduction of 1.

line coincides with a diagonal of the polygon). The sum of the estimations of both split pieces  $k_1+k_2$  must therefore be at least  $k$ . Since the values of  $k_1$  and  $k_2$  are rounded upwards, cases where  $k_1$  or  $k_2$  are not factors of  $N$  can only increase the sum  $k_1+k_2$ .

Special cases where one or more lines are **collinear** in the given polygon or in the resulting split pieces must be considered and treated separately. In such cases, a single primitive may simultaneously match more than one line of the contour. This may cause the heuristic function to overestimate the actual number of primitives. To remedy this, collinear lines or collinear points are counted differently. In cases of  $m$  ( $m>2$ ) collinear vertices, the number of lines that may be created using these vertices is  $(m-1)$ . These  $(m-1)$  lines can be matched simultaneously (in the worst case) by one primitive. To ensure that the heuristic function does not overestimate, these  $(m-1)$  lines are counted as one, i.e.,  $(m-2)$  lines must be subtracted from the original  $N$  lines of the polygon before  $k$  is calculated. In Figure 5 (a),  $N'=8$  but there are 4 collinear points. Thus,  $(m-2) = (4-2)$  is subtracted from  $N$ , so that  $N=6$ . An estimation of tiling with rectangles where  $n=4$  produces the estimation of  $k = N'/(n-1) = 6/3 = 2$ , which is a correct estimation. Similarly, an estimation of triangle tiling of the shape illustrated in Figure 5 (b) is also corrected by this amendment.

### Primitive Identification

All primitive objects must supply a function for identifying a polygon as the primitive. Precise implementation depends on the primitive type, but care must be taken to overcome pathological cases of zero-length lines, subdivided lines and near-accurate angles. Typically, some tolerance is required in the identification.

### Algorithm steps

Given an initial polygon to decompose, the decomposition proceeds as follows:

1. Compute the cost  $f(n)$  of the given polygon and store it in the Open-State list.
2. Select the lowest cost state from the Open-State list. Check whether all the pieces are primitives. If so, the optimal solution has been obtained - Stop.
3. Obtain splitting suggestions for the lowest cost state from all primitive functions. Merge and eliminate duplicate suggestions. Move the state to the Closed-State list.
4. Duplicate the current state, split according to each suggestion in turn, compute cost, and store the new states in the Open-State list.
5. Go to step 2.

This procedure always converges into the optimal decomposition, namely, achieving the minimal number of non-overlapping primitives required to tile the original shape. It is important to note that optimality is guaranteed subject to the options spanned by the splitting suggestions. If a certain subdivision is never suggested, it will never be reached.

## **"Plug-in" primitives**

The estimation term in the cost computation is obtained by taking the most optimistic forecast of the estimator functions of all the primitives. A primitive is identified sequentially by calling the identification routine of each primitive, and split suggestions are obtained by merging the suggestions of all primitive split-suggesting functions. That is, addition of a new primitive to the decomposition system involves only the addition of a new primitive object supplying the three member functions.

Convergence of the search procedure can be guaranteed only if a goal state is feasible. In the polygon decomposition problem domain, feasibility corresponds to the ability to tile an arbitrary shape with *some* primitive. To ensure this, a *triangular* primitive must always be available to the system, and the triangular split-suggestion function must be able to decompose any shape.

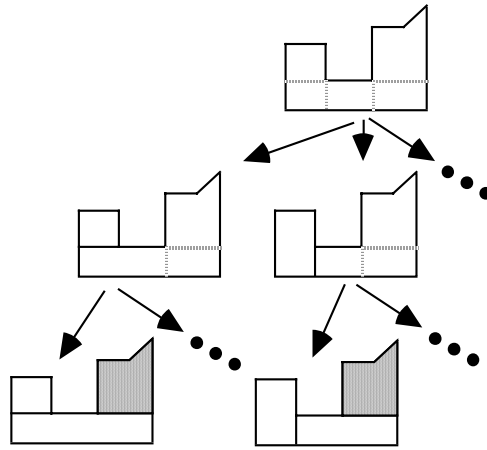


Figure 6. Partially duplicate decomposition states.

## Elimination of duplicate states

One of the reasons for the large number of decomposition states is the duplication of state configurations. A state duplication occurs under two circumstances. The first case is a consequence of applying the same set of split operators in a different sequence. Figure 3 shows four final decomposition states, each duplicated twice (e.g., state 1 and state 5). The second case is a *partial* state duplication, where one or more of the pieces in a state appears in another state as well. Here, the decomposition algorithm will "work" on decomposing that specific piece more than once. Such a situation can occur as a result of different sequences, as is illustrated in Figure 6.

The first situation stems from the fact that the heuristic search algorithm does not search only states but also paths to states; there are several optimal decomposition states and several paths leading to each. The search algorithm may simultaneously expand several paths that ultimately lead to the same decomposition. In the decomposition problem, however, optimality is a function of the final state and not of the path leading to it. State repetition can be avoided by posing a constraint so that any decomposition state can be reached from only one decomposition path (i.e., one splitting sequence). The constraint is enforced by assigning some arbitrary order to the split operators. The order, for example, can be the chronological order in which the split

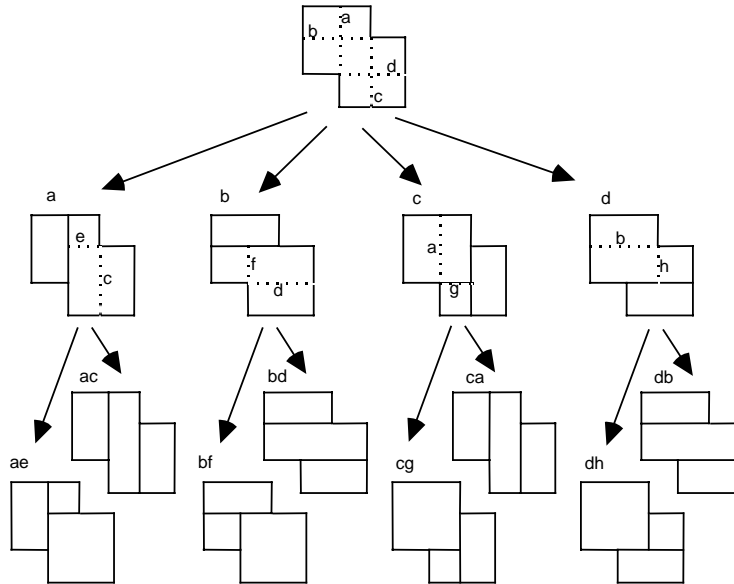


Figure 7. Duplicate decompositions eliminated by disallowing descending split sequences.

operators are first suggested. Once an order is set, the search process should disallow the application of split operators in an order different from their assigned order. This eliminates the possibility of the same state being found by following different paths without missing any decomposition state. Only one sequence of split operators can correspond to any given decomposition state. The number of paths eliminated by this technique is proportional to the number of paths leading to each decomposition state, which is roughly a factorial of the number of split operations.

Consider the example shown in Figure 7. The given shape can be initially decomposed using four suggested split lines. These split lines are assigned the letters *a*, *b*, *c* and *d*, according to some arbitrary order. The shape in the initial state is then decomposed into four new states, each created by splitting the shape in the initial state according to one of the four splitting suggestions. The letter above the new state indicates which split operator was used to obtain that state. Next, two split suggestions are provided for each new state. These split suggestions are also labelled with letters; note however, that split lines previously considered are recognized and assigned their original letter. For simplicity, the heuristic function recommendation is temporarily disregarded and all the new states are expanded. Eight new states are obtained, with the split sequence written above each. Of these eight states, only six are different, and two are duplications. More importantly, the two duplicates are those with a non-ascending lexical order of split operators. If the rule suggested above had been applied, these two states would not have been expanded.

The second case of partial-state duplication, as illustrated in Figure 6, does not result in the same set of operators and will not be eliminated by the above technique. It still occurs an exponential number of times and contributes to the overall size of the search domain. The technique used to overcome this difficulty involves the actual *identification* of the duplicate piece and management of a list of "known pieces". Whenever a contour is encountered for decomposition, it is first matched against the list of known pieces (using a hashing function). If this is not the first instance of the shape, then data on its decomposition will be available. Conversely, if it is the first instance, decomposition continues while back-propagating data to the list and other states using this shape. This technique accelerates the decomposition process but requires space resources that are exponentially proportional to the depth of the search and therefore may be impractical on some machines.

## Results

Figure 8 shows decompositions obtained by implementing the proposed decomposition algorithm. The search was accomplished using a triangle, a rectangle and trapezoid primitives (a trapezoid was defined as a quadrilateral with at least two parallel sides). The search algorithm used was *iterative deepening A\** (*IDA\**) [12], which is a space-conservative variant of the A\* algorithm.

## Extensions

### Fast Time-Limited Decomposition

Some applications require real-time solutions to decomposition problems, for example in an interactive system where the user is kept waiting while a shape is decomposed. In such cases, speed may be traded for optimality.

In the proposed algorithm, the most prominent time-consuming factor is the number of splitting suggestions. Optimality is sacrificed by selecting only a limited number of the most promising splits (according to the heuristic estimation), thus reducing the search domain. A parameter specifying the maximum number of suggestions taken at each

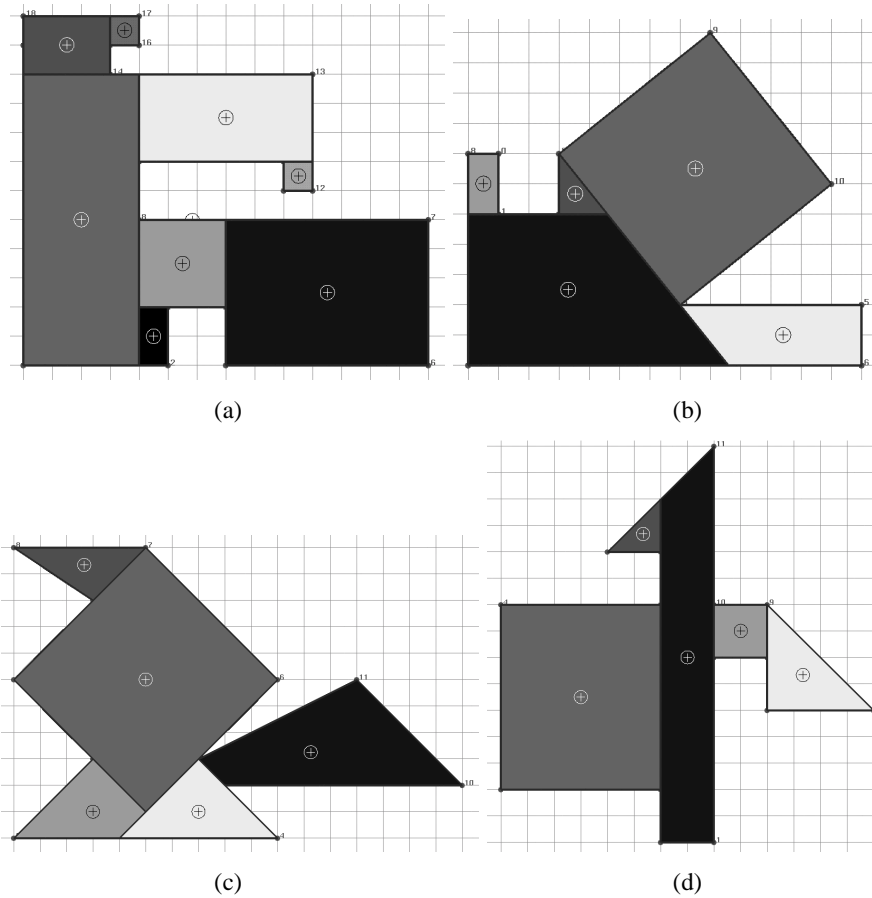


Figure 8. Decompositions obtained using the search process.  
 (SG Iris 4D: (a) 2.3 (b) 0.6 (c) 0.8 (d) 1.1 CPU secs.). (Shape (d) adopted from Parry-Barwick et al. [15]).

level can be used to govern the solution speed versus deviation from optimality. Figure 9 illustrates a near-optimal decomposition obtained in this way.

An important consideration is that because of the mechanism for identifying and storing partial decompositions, the search algorithm can be reactivated with a more elaborate search without actually recomputing parts of the tree already visited. Thus, the decomposition may be improved when more time is available (or "in the background") while minimizing resource waste.

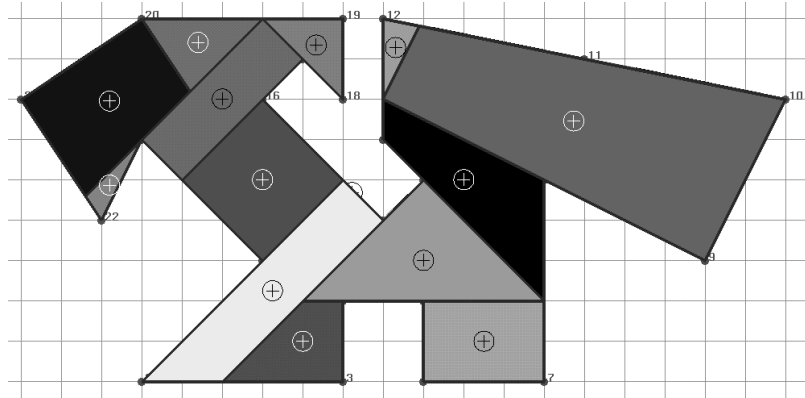


Figure 9. Near optimal decomposition obtained under a time limit (SG Iris 4D - 3.7 CPU Seconds).

## Tie-breaking

One of the important properties of  $A^*$  is that of all search algorithms using the heuristic estimation function  $h(n)$ , it expands the fewest number of nodes, up to tie breaking between equal cost nodes [16]. Ties in the cost function  $f(n)$  of candidate nodes can have a detrimental effect; at the specific point where a tie occurs, no information is available to distinguish candidate nodes. In the absence of this information, a local breadth-first search is performed. The effect of ties is especially noticeable when the cost function is discrete or of low resolution, and hence the probability of a tie in the cost function is high. In the polygon decomposition problem, the cost function is expressed as an integer estimating the number of primitives necessary for tiling a shape. This function is indeed both discrete and of low resolution (in the range of approximately 1 to  $n$ ).

To improve performance, the search algorithm must be supplied with more information to enable a wiser distinction between equal-cost decomposition states. Provided that this information is only used in tie situations, i.e., the information will never overrun the cost-function recommendation, its usage will not impede the optimality ensured by  $A^*$ .

The tie-breaking information chosen in implementing the polygon decomposition algorithm is *split-length*. This information describes the total length of lines along which the initial polygon was split, up to the current decomposition state. This function is heuristic; it cannot guarantee selecting the better node in a tie, but it tends to do so. The

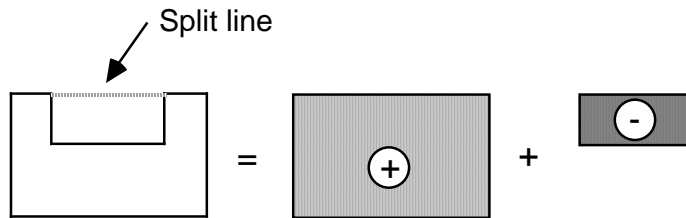


Figure 10. External split lines form subtraction operations.

split-length value is added as a fraction beyond the integer specifying the number of primitives. The value was scaled so that it always remains a fraction and never overruns the true cost function but only influence ties.

One of the advantages of the split-length value as a tie-breaker is that it is simple to calculate and forecast. A conservative forecast is made by multiplying the number of expected primitives by the length of the shortest vertex-line distance in the polygon. Provided that the rules for selecting split suggestions are obeyed, this forecast never overestimates the actual split length. Introduction of this heuristic tie-breaking mechanism has significantly improved search time performance.

## Non-additive decomposition

Until now, the search algorithm was confined to those split operators that expanded the decomposition state by splitting a shape into two *disjoint* pieces. Relaxing this requirement provides more decomposition options. Although such relaxation may enlarge the search domain and extend the search process, it also improves the chances of encountering a more efficient decomposition. The scope of splitting operators can be expanded in various ways. In this implementation, the splitting operator scope was enhanced by allowing split lines to run *outside* the shape. The polygon is split into two parts: one is larger than the original shape and the other is marked by a "negative sign" indicating that it should be subtracted from the larger shape. Figure 10 illustrates such a decomposition.

Introduction of this type of external split operator in the proposed decomposition algorithm allows the basic CSG operation of subtraction. While this introduction does increase the search domain size, it does not necessarily increase solution time, as might have been expected. In some cases, execution time was even reduced, perhaps because better opportunities were introduced, causing complete branches of the search tree to be pruned by the A\* algorithm.

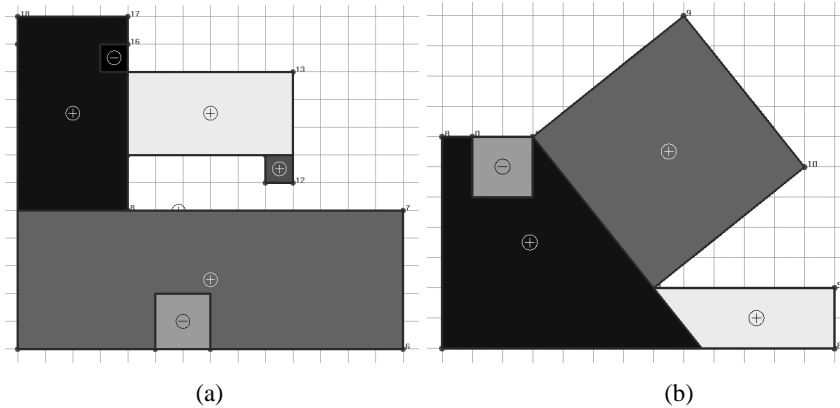


Figure 11. Decomposition obtained using external split line, of shapes in Figure 8. (SG Iris 4D: (a) 3.6 (b) 0.5 CPU secs.)

Furthermore, the number of primitives was reduced, thus reducing the maximum depth of the tree. Figure 11 shows results obtained using external split operators. Note that the introduction of external splits has allowed instances of primitive subtraction, thus making it possible for shape 11(a) to be decomposed into six primitives instead of eight (Figure 8(a)) and shape 11(b) to four pieces instead of five (Figure 8(b)). Also note that the decomposition time in these examples has not necessarily increased.

## Expansion to curved edge and three-dimensional solid decomposition

The proposed algorithm has been described and implemented for a restricted geometrical problem domain consisting of two-dimensional straight-edged polygons. In developing the algorithm, however, no assumptions have been made about this restricted domain. Relaxing the restriction to include curved objects or three-dimensional solids would require a new complexity analysis and the following modifications:

- A basic primitive or set of primitives capable of tiling any shape in the domain must be available. If curved shapes are introduced, triangles will not suffice, and curved edge primitives must be provided. For three dimensions, a tetraheder primitive could be used to ensure convergence for planar-faced solids.
- More elaborate splitting-suggestion and splitting execution procedures must be provided, according to the type of primitives supported. The basic rules regarding preferred splitting operators may still be observed (continuation, collinearity, etc.), however, it is expected that execution of the splitting itself, which is a

trivial matter in two dimensions, may significantly slow down the performance time for three dimensional solids.

## Conclusions

In this paper, a polygon decomposition algorithm based on Artificial Intelligence (AI) heuristic search techniques has been proposed and its implementation has been described. The proposed approach can decompose a polygon into any given set of primitives. The use of AI provides flexibility in defining both the primitives and the decomposition target function. Various primitives can be added, including semi-analytic and special shapes. (For example, an L-shaped primitive can be used if its associated member functions are provided.) The target function can be modified to yield any goal condition, such as minimal number of primitives, minimal cut length, or pieces of similar area. This flexibility can be naturally expanded to support features fulfilling the three-function requirement.

The implementation supports two modes of operation: optimal and fast time-limited decomposition. There are two partitioning options: disjoint addition and subtraction. In addition, it should be noted that since every step in the solution is a function of the current state only, implementation of this algorithm can make effective use of parallel processing environments.

## Acknowledgments

This work was carried out in partial fulfillment of the requirements for the degree of D.Sc. in Mechanical Engineering at the Technion [Hod Lipson]. This research has been supported in part by the Fund for the Promotion of Research at the Technion, Research No. 033-028.

## References

1. A.A.G. Requicha, "Representations for rigid solids: Theory, methods and Systems", *ACM Computing Surveys*, **12,4** (437-464), 1980.

2. A.A.G. Requicha and H.B. Voelcker, "Boolean operations in solid modeling: boundary evaluation and merging algorithms", *Proc IEEE*, **73**,1 (30-44), 1985.
3. D. Dobkin, L. Guibas, J. Hershberge and J. Snoeyink, "An efficient algorithm for finding the CSG representation of a simple polygon", *ACM Computer Graphics*, **22**,4 (31-40), 1988.
4. V. Shapiro and D.L. Vossler, "Construction and optimization of CSG representations," *Computer Aided Design*, **22**,1 (4-20), 1991.
5. V. Shapiro and D.L. Vossler, "Separation for boundary to CSG conversion", *ACM Transactions on Graphics*, **12**,1 (35-55), 1993.
6. S.B. Tor and A.E. Middleditch, "Convex decomposition of simple polygons", *ACM Transactions on Graphics*, **13**,4 (244-265), 1984.
7. T.C. Woo, "Feature extraction by volume decomposition", *Proceedings of the Conference on CAD/CAM Technology in Mechanical Engineering* (76-94), 1982.
8. J. O'Rourke, *Art Gallery Theorems and Algorithms*, Oxford University Press, 1987.
9. B. Chazelle, "Approximation and decomposition of shapes", in J.T. Schwartz and C.K. Yap, eds., *Advances in Robotics*, Vol 1, *Algorithmic and Geometric Aspects of Robotics*, Lawrence Elbaum (145-185), 1987.
10. P. Schmitt, "Problems in discrete and combinatorial geometry", in P.M. Gruber and J.M. Wills, Eds., *Handbook of Convex Geometry*, Vol. A, Elsevier (461-483), 1993.
11. J.M. Keil, "Decomposing a polygon into smaller components", *SIAM J. of Computing*, **14**,4 (799-817), 1985.
12. J. Pearl and R.E. Korf, "Search techniques", in J. Pearl, *Heuristics*, Addison Wesley (451-467), 1984.
13. D.H. Winston, *Artificial Intelligence*, 2nd Edition, Addison Wesley, 1984.
14. P.E. Hart, N.J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths", *IEEE Trans. Syst. Sci. Cybernetics*, **4**,2 (100-107), 1968.
15. S.J. Parry-Barwick and A. Bower, "Minkowski sums of set-theoretic models", *CSG 94 Set-theoretic Solid Modelling: Techniques and Applications*, Proceedings of the conference held in Winchester (101-116), April 1994.

16. R. Dechter and J. Pearl, "Generalized best-first search strategies and the optimality of A\*", *J. Assoc. Comput. Mach.*, **32**,3 (505-536), 1985.