Computer-Aided Design 46 (2014) 233–238

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Technical note

Puzzhull: Cavity and protrusion hierarchy to fit conformal polygons

Shanaz Mistry, U.N. Niranjan *, M. Gopi

University of California, Irvine, USA

HIGHLIGHTS

- We define cavities and protrusions of a 2D polygon which are key geometric features of a polygon.
- We propose an algorithm to compute cavities and protrusions.
- We propose a method for 2D polygonal piece fitting using a hierarchy of geometric features defined by cavities and protrusions.

ARTICLE INFO

Keywords:
Cavity polygon
Protrusion polygon
Convex hull
Hierarchy
Geometric transformation

ABSTRACT

In this paper, we present a simple definition for, and a method to find, cavities and protrusions of a 2D polygon. Using these, we fit conformal polygons with each other, which is similar to a jigsaw puzzle and in a general case is NP-hard to solve. We first build a hierarchy of cavities and protrusions for each polygon and use this hierarchy to check for matches between these geometric features of two polygons. This data structure allows for early rejection of mismatches and thus speeds up the fitting process. We show using many examples, that most of the common configurations in exact polygon fitting can be handled by this algorithm in polynomial time. In case of exact, yet non-unique matches, this algorithm will solve the problem in exponential time.

1. Introduction

Given a finite number of 2D simple polygonal pieces (i.e., genus zero, orientable, no self-intersections), our goal is to find the pieces which are completely conformal with each other and fit them together. Closely related problems include the 2D irregular nesting problem, the cutting stock problem, and scheduling problems [1]. Most optimization problems of practical interest are computationally hard to solve [2]. In recreational mathematics, the problem of solving jigsaw puzzles is NP-complete [3]. In this paper, we focus on scenarios that are arguably common and solve such an instance using a novel and efficient data structure.

The classes of the fitting problem which are handled by our algorithm are:

(i) *Exact unique matches:* In this case, for every site (cavity or protrusion of a polygon), there exists at most one matching polygonal site, which has a perfect geometric matching in terms of area, perimeter, and edge length.
(ii) *Exact non-unique matches:* In this case, for every site (cavity or protrusion of a polygon), there may be more than one matching polygonal site.

1.1. Main contributions

In this paper, we consider the problem of fitting conformal 2D simple polygonal pieces, wherein the pieces completely fit with each other and we are allowed access to only the geometric information of the polygonal pieces, and not other features such as colors and textures provided in the jigsaw puzzle problem. Following are the main contributions of the paper:

(i) We present a new definition of the geometric features of a polygon such as *cavities* and *protrusions*. We then present an algorithm using convex hulls to compute these features, each of which is a polygon by itself.
(ii) We propose to recursively compute the cavities and protrusions present in a polygonal piece, and build a hierarchy of these geometric features. Essentially, it decomposes any input piece into multiple geometrically related sub-parts. Our hierarchy is rotation, uniform scale, and translation invariant.
(iii) We propose a method to use the hierarchy to find the fitting between two input polygonal pieces. The nodes representing cavities of one polygon is compared with the nodes representing the protrusions of another. This is how we reduce the search space to find the conformal regions. We use a variant of tree isomorphism on this hierarchy to do early rejection of non-matching polygons. Since the number of nodes in the hierarchy is also dependent on the number of features rather than the number of edges, the total complexity of the algorithm is output-sensitive.

* Corresponding author.
E-mail addresses: symistry@uci.edu (S. Mistry), un.niranjan@uci.edu (U.N. Niranjan), gopi@uci.edu (M. Gopi).

0010-4485/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cad.2013.08.038

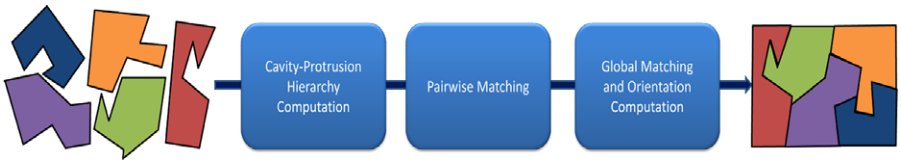

Fig. 1. Schematic summary of our method for fitting polygons that are conformal using the geometric features extracted into a hierarchy.

(iv) For non-unique exact matching, we propose a backtracking-based algorithm to find the solution.

However, we also note that our algorithm handles neither approximate fits nor those in which there exists no edge-to-edge correspondence between cavities and protrusions. Our algorithm will also work only if every cavity or protrusion polygon has at least one edge that is not part of the original polygon. In other words, an all-convex polygonal set of pieces has no cavity in any of its polygon and hence will not yield any result from our algorithm. In such a case, a simple edge–edge comparison between the polygonal pieces and an exhaustive search for match [4] can be used.

The pipeline of our algorithm is shown in Fig. 1.

2. Related work

In computer vision, image recovery from pieces of the original image is a fundamental problem. There are learning-based approaches to solving puzzles, which typically use features of the dataset, rather than using just the geometric information to recover a broken image. In [5], matching is performed using global features called *isthmii* which can be either positive or negative. These features are extracted using the exoskeleton of the pieces. For example, when the location is given but the orientation of the pieces has to be determined, a model based on Markov Random Field has been used in [6]. In the patch-based approach of [7], the solution is obtained by determining the most likely patch configuration.

Another method of solving 2D cutting stock problems is using the Minkowski sum hull, as described in [8], that gives the outer and inner envelopes which are then used to perform matching. In the No Fit Polygon (NFP) approach [9,10], one polygon is kept fixed while a second polygon is moved along the boundary of the first polygon, keeping a specific point of the second polygon in contact with the first polygon at all times without overlap between polygons. The external polygon created by such a procedure gives the NFP. In [11], NFP is used by fitting the shapes using rectangular enclosures to minimize wastage.

In [12], the similarity of curves is computed using a matching cost function which takes into account the length and the orientation of curve segments. They also extend their approach to 3D where they perform space curve matching by computing a metric based on speed, curvature and torsion of the space curve. This method, being an instance of local shape matching, is similar to our method in checking for local fits.

In [13], convex and concave features of parts are extracted to provide matches between different shapes. The extracted features have a three edge limit. This process is similar to our algorithm in that it extracts convex and concave edges, but our algorithm considers the area as a whole, which allows for efficient local matching to be performed.

Many solutions have been studied and proposed to solve the 2D nesting problem and its variants. One such method, which is similar to our approach, is that of geometric matching using polylines [4]. They perform partial shape matching by computing the similarity of a set of polylines to a polygon, both provided by the user. In the current work, the main contribution is to automatically generate the polylines for computing the similarity through a decomposition of the polygon using a novel convex hull hierarchy called the cavity–protrusion hierarchy. Further, our matching is performed simply by comparing the area, the perimeter, the edge lengths, and the structure of the hierarchy.

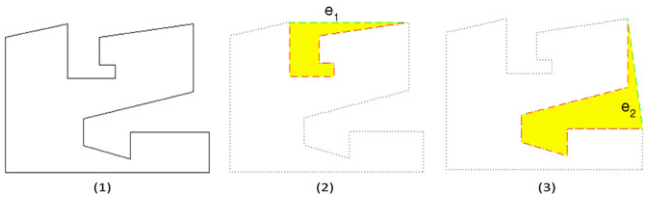

Fig. 2. Results of cavity polygon computation showing the input polygon, the cavity polygon capped by the virtual edge e_1, and the cavity polygon capped by the virtual edge e_2.

3. Cavities, protrusions, and the hierarchy

Let $Q \subset \mathbb{R}^2$ be a simple polygon. Let $\mathcal{H}$ be the convex hull of Q. Let C_j be the set of all open connected regions of $\mathcal{H} \setminus Q$. We define each polygon given by the closure $\bar{C}_j$, as a cavity. The edges of Q in each cavity are called *cavity edges*, and the edges of $\mathcal{H}$ in the cavities are called *virtual edges*. Every cavity will have exactly one virtual edge "capping" it (Fig. 2). The protrusions are formed by the regions of the input polygon between two consecutive cavities. We define the *protrusion polygon* in Section 3.2 constructively.

The concept of cavities and protrusions forms the building block of the hierarchy. Note that the cavities and protrusions are polygons themselves. Hence, cavity–protrusion construction can be recursively performed on each of these polygons to create the hierarchy. This hierarchy is an n-ary tree. Each node of the tree represents a polygon, and each child represents a cavity or a protrusion polygon. Henceforth, we use the term polygon to refer to either the input polygon Q, or the cavities and protrusions of Q (which are denoted as P), the meaning being clear from the context. We define *negative space* as the area being outside the input polygon Q and *positive space* as the area being inside Q.

3.1. Cavity polygon computation

From the definition of a cavity and a virtual edge, we observe that every convex hull edge that is not a polygon edge is a virtual edge. Further, the number of cavities is same as the number of virtual edges, and along with the sequence of edges between the end points of the virtual edge, constructed only from the polygon edges and not convex hull edges, we can compute the corresponding cavity. This leads to a very simple cavity polygon computation method (Algorithm 1).

In this paper, we assume the cyclic ordering of edges. We also assume that both the convex hull polygon and the given polygon edges are ordered in the same direction. Given this assumption, for each convex hull edge that is not the given polygon edge, i.e., for each virtual edge, we find the sequence of polygon edges in the same cyclic order from the start and end points of this virtual edge. These polygon edges are guaranteed not to be part of the convex hull. These polygon edges along with the corresponding virtual edge will form a cavity polygon. Algorithm 1 can be applied

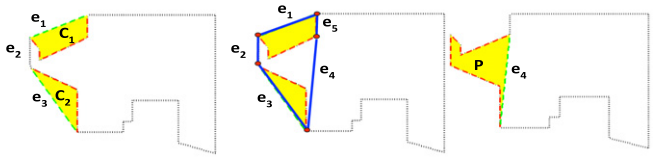

Fig. 3. Results of protrusion polygon computation showing the input polygon with cavities C_1 and C_2, the convex hull of the cavity edges and the polygon edges between them, and the protrusion polygon capped by e_4.

recursively on each generated cavity polygon. Note that the cavity of a negative space polygon is a positive space polygon and the cavity of a positive space polygon is a negative space polygon.

Algorithm 1 Cavity Polygon Computation

Require: Hierarchy tree node for a polygon P
Ensure: Cavity polygons C_e created as children of P
1: Compute convex hull $\mathcal{H}$ of P
2: **while** edge $e \in \mathcal{H}$ **do**
3: **if** $e \notin P$ **then**
4: Find edges $e_1, \ldots, e_k \in P$ between the end-points of e
5: $P' =$ polygon using edges $e, e_1, \ldots, e_k$
6: Tree node $C_e = (P', \text{area}(P'), \text{perimeter}(P'))$
7: Add C_e as a child of node P
8: **end if**
9: **end while**

3.2. Protrusion polygon computation

Let C_i and C_{i+1} be two consecutive cavity polygons of a polygon P. Consider the cavity edges of C_i and C_{i+1} along with the edges of P between them (in sequential order). Compute the convex hull of this combined set of edges. From this convex hull, eliminate the virtual edges of C_i and C_{i+1}. For each virtual edge that remains, the processing is similar to the cavity polygon computation: We consider all the polygon edges that lie between the end-points of the virtual edge in order. These edges along with its corresponding virtual edge constitute the edges of the *protrusion polygon*. This procedure is summarized in Algorithm 2.

Algorithm 2 Protrusion Polygon Computation

Require: Polygon P and its list of cavities C
Ensure: Protrusion polygons B created as children of P
1: **for** $i = 1 : n \ (mod \ (n + 1))$ **do**
2: $E = C_i, C_{i+1}$, and edges of P that lie between C_i and C_{i+1}
3: $\mathcal{H} =$ convex hull of E
4: **end for**
5: **while** virtual edge $e \in \mathcal{H}$ **do**
6: **if** e is not a virtual edge associated with C_i or C_{i+1} **then**
7: Find edges $e_1, \ldots, e_k \in P$ between the end-points of e
8: $P' =$ polygon using edges $e, e_1, \ldots, e_k$
9: Tree node $B = (P', \text{area}(P'), \text{perimeter}(P'))$
10: Add B as a child of node P
11: **end if**
12: **end while**

In Fig. 3, consider the cavities C_1 and C_2. The edge of the polygon between C_1 and C_2 is e_2. Taking the convex hull we obtain the edges e_1, e_2, e_3, e_4, e_5. Consider the virtual edges in this polygon e_1, e_3 and e_4. Since e_1 and e_3 are the virtual edges of the cavities, we remove them. The polygon capped by the rest of the virtual edges are the protrusion polygons. In this case, we obtain the one capped by the edge e_4.

Note that the protrusion polygons will be inside the convex hull of the original polygon, but parts of it may be outside the original polygon; this helps in better matching between cavities and protrusions (Fig. 4).

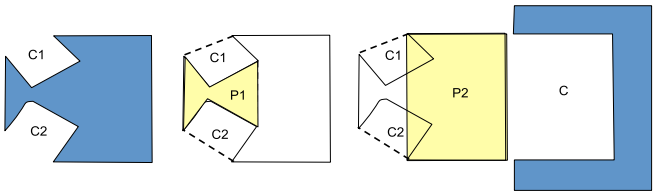

Fig. 4. The given polygon and two cavities can be processed to get two protrusions. Note that parts of $P2$ are outside the given polygon. But this helps in finding an appropriate cavity (C) that this protrusion can fit into. Geometric features in the regions of the protrusion that is farther from the cavity does not affect the physical fitting into the cavity. But these features will affect the area/perimeter of the protrusion—the quantities that are used to find a matching cavity. In our case, the effects of these farther features are nullified by design, by capping that region to be convex using a virtual edge, uniformly for all protrusions. This will lead to portions of the protrusion to be outside the given polygon.

The protrusion polygon computation is performed for every pair of consecutive cavity polygons obtained from the immediately preceding cavity polygon computation step. Algorithm 2 can be applied recursively on each generated polygon. Note that the protrusion of a negative space polygon is a negative space polygon and the protrusion of a positive space polygon is a positive space polygon.

3.3. Construction of the hierarchy

Algorithm 3 Hierarchy Computation

Require: Polygon P
Ensure: Hierarchy
1: **if** P is present in the hierarchy OR P is convex OR P is a sectional polygon **then**
2: **return**
3: **end if**
4: $C \leftarrow$ Compute Cavity Polygons(P) {C are children of P}
5: $D \leftarrow$ Compute Protrusion Polygons(P, C) {D are children of P}
6: For each polygon F in $C \cup D$, Hierarchy Computation(F)

In the hierarchy, the root node is the input polygon and the children are its cavities and protrusions. This procedure is done recursively on each of the computed polygon (Algorithm 3). We also compute and store the area and perimeter of each polygon in the nodes of the hierarchy. We also store the sign of the polygon indicating whether it is a positive space polygon or a negative space polygon. This is later used to compute the canonical names while checking for tree isomorphism during the matching process.

An interesting consequence of the hierarchy construction procedure is that cavity analysis of a polygon results in a reversal of signs, and protrusion analysis results in retaining the sign of the parent. If a polygon has n virtual edges, the node that represents the polygon in the hierarchy has $2n$ children, except when multiple protrusion polygons are formed. We note that the hierarchy itself (nodes and their parent–child relationship) is invariant to rotation, uniform scale, and translation. The area and perimeter can be recomputed for each node in case of uniform scale.

An example of the hierarchy construction is illustrated in Fig. 5.

4. Matching of polygonal pieces

Once the hierarchy has been extracted for each input polygonal piece, we match one piece to another. In exact matching, we specifically look for edge–edge matches. Ideally every edge sequence of every length of a polygon has to be compared with all edge sequences of the same length in the other polygon. Our hierarchy chooses edge sequences that define certain geometric features (such as cavities and protrusions) and are more likely to form

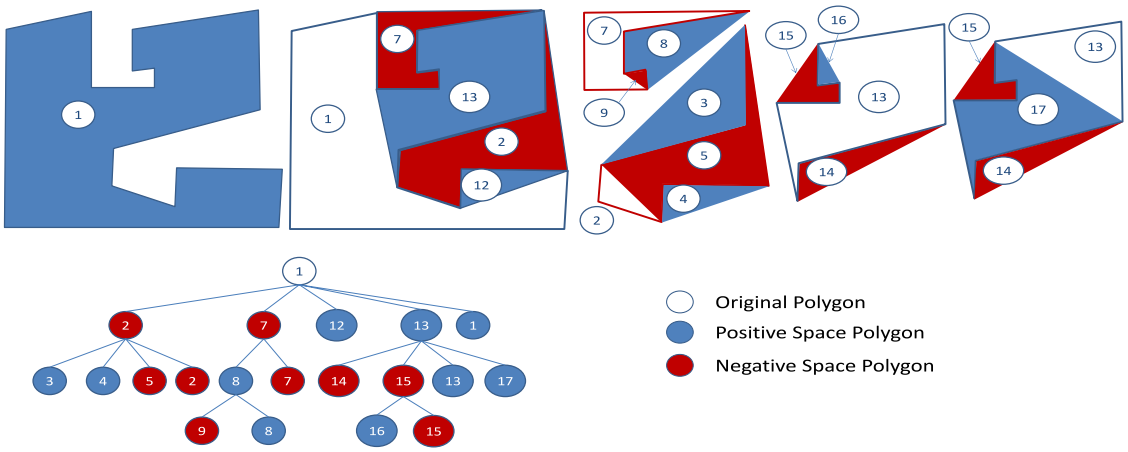

Fig. 5. Hierarchy construction: polygon 1 is the given polygon. Protrusions between the cavities 2 → 7 are 12 and 13, but the protrusion between the 7 → 2 is the original polygon itself which is one of the terminating conditions of the hierarchy construction. A similar condition holds for polygons 2 and 13. Further, if there is only one cavity, the protrusion formed by the lone cavity is the original polygon (polygons 7, 8, 15). Finally, when a positive space polygon has more than one edge that is not part of the original polygon, such a polygon is less meaningful to our application and hence we terminate that subtree of the hierarchy (e.g. polygon 17). We term such polygons *sectional polygons*.

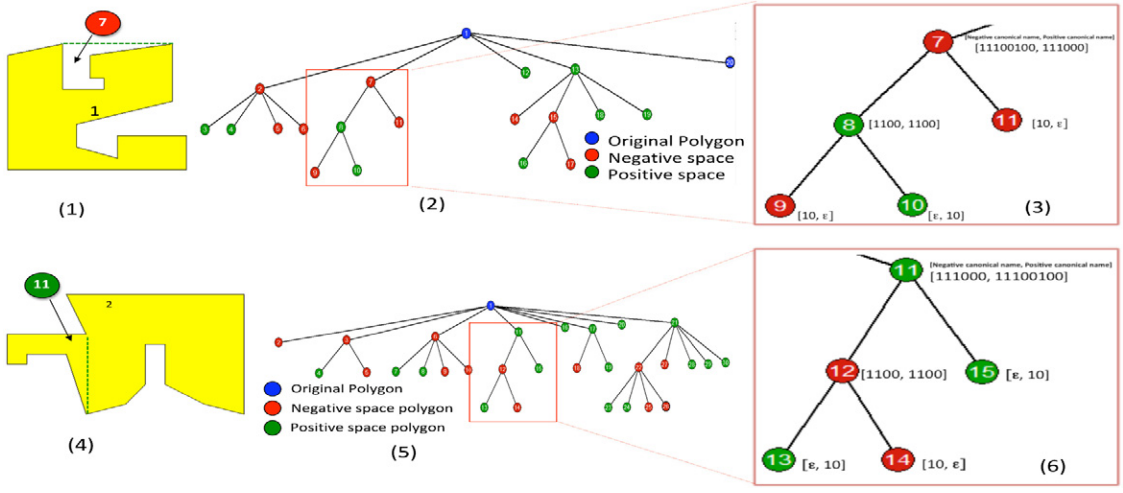

Fig. 6. Subtree isomorphism: (1) Original piece S_1. (2) Hierarchy for the polygon S_1 with highlighted subtree for polygon 7. (3) Subtree for polygon 7 showing the parameters for performing tree isomorphism. (4) Original piece S_2. (5) Hierarchy for the polygon S_2 with highlighted subtree for polygon 11. (6) Subtree for polygon 11 showing the parameters for performing tree isomorphism.

matching parts of two polygons. Of course, any edge sequence based on any other relevance criteria can be chosen, and its cavities and protrusions can be computed and added. This property makes the hierarchy more general and suitable for other applications also.

Our polygon matching algorithm is given in Algorithm 4. Note that the root node is the original polygon, and is a container node. So the polygon pair checking is done from their children. The goal of this method is to do as many early rejections as possible. We reject polygon pairs if both of them are positive or both are negative space polygons. We also reject if the negative space polygon is smaller than the positive space polygon, since the positive piece cannot be fit into the negative space. We go down the hierarchy if the negative space polygon is larger than the positive space. If the areas are equal, then there are two possibilities: either they fit perfectly or do not fit. If they fit, then the geometric structure of these negative and positive pieces should be exactly the same, and hence the subtrees rooted at these trees have to be isomorphic. But the trees may look similar even with minor changes in the geometry. So the leaf nodes during isomorphism checking are also ordered based on their area, and then compared with the other tree. If they match, then the rotation and translation of one piece to fit to the other is computed. If such a transformation does not cause the polygon–polygon intersection, then this pair is selected as one of the matching pairs.

Subtree isomorphism: The positive and negative space polygons form subtrees in the hierarchy. If the pieces fit, the subtree of the hierarchy corresponding to these two polygons will exhibit isomorphism which corresponds to a consistent mapping of vertices and edges of the puzzle pieces, in the fitting region, up to scale. We use a variant of the classical Aho–Hopcroft–Ullman tree isomorphism algorithm for our implementation (for other related algorithms, see [14] and the references therein). In our isomorphism algorithm, instead of computing just one canonical name for each node, we compute two. The negative canonical name computed by combining the canonical names of the root node's negative children and the positive canonical name computed by combining the canonical names of the root node's positive children. We store the area, perimeter and number of edges of each leaf node in the subtree while computing the canonical names. If the negative canonical name of C_i matches with the positive canonical name of P_j and vice-versa, we say that the two subtrees are isomorphic and hence could be a potential match. We compare the stored geometric parameters to find exact matches. Fig. 6 illustrates our isomorphism algorithm.

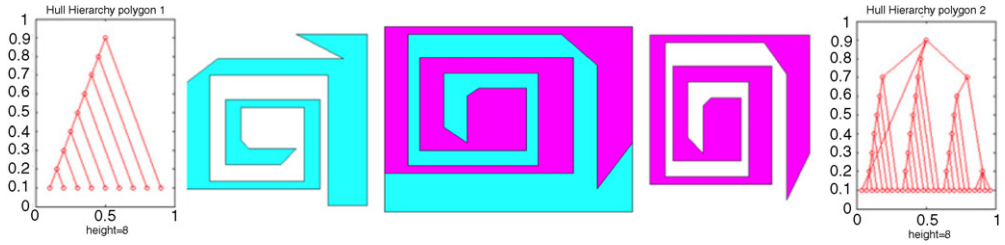

Fig. 7. Results of our algorithm, along with the hierarchies of the input polygons, on an example of the exact unique matching case.

Algorithm 4 Matching

Require: Hierarchies of two polygons P_1 and P_2 which are compared for a possible match
Ensure: List of matches
Enqueue children of root of P_1 and P_2 as $Queue_1$ and $Queue_2$
while $Queue_1 \neq \phi$ **do**
  Dequeue node N_1 from $Queue_1$
  while $Queue_2 \neq \phi$ **do**
    Dequeue node N_2 from $Queue_2$
    Reject, if sign(N_1)=sign(N_2) {since both are positive space or negative space polygons}
    Reject, if area(N_1)>area(N_2) AND N_1 is positive space
    Reject, if area(N_1)<area(N_2) AND N_1 is negative space
    if area(N_1)>area(N_2) AND N_1 is negative space **then**
      Matching(child of N_1, N_2), ∀ children of N_1
    else if area(N_1)<area(N_2) AND N_1 is positive space **then**
      Matching(child of N_2, N_1), ∀ children of N_2
    else if area(N_1) = area(N_2) **then**
      Check isomorphism of subtrees rooted at N_1 and N_2
      Use point correspondences to compute affine transformation
      Reject, if polygons intersect, else add (N_1, N_2) to List of matches
    end if
  end while
end while
return List of matches

For the worst-case asymptotic running time of our pairwise matching algorithm, note that, since we use convex hull computation as subroutines, the tighter bounds on the time complexity are output-sensitive, the worst case being $O(p \log p)$, where p is the number of points whose convex hull is computed to obtain a single node of the hierarchy. The time complexity of a single execution of the tree isomorphism subroutine is $O(s)$ where s is the number of nodes of the subtree, i.e., it depends on the number of features, and specifically, not on the number of edges of the polygon. In the worst case, matching two input polygons using their hierarchies takes $O(NM)$ time, where N and M are the number of nodes in the hierarchies of the input polygons.

4.1. Global fitting of polygons

The final step is performing the fitting itself. When subtrees of the hierarchies corresponding to the input polygons exhibit isomorphism, we compute point correspondences based on any leaf node combination. We then transform the second polygon based on the affine transformation matrix generated using the point correspondences, and check if the polygons intersect. If the polygons do not intersect, they are added to the list of matches.

Given the list of matches, if the polygons have exact unique matches, each cavity will have no more than one protrusion matched with it. But if the matches are non-unique, then we need to pick the best match for a cavity out of many protrusions that would match it. In such cases, we use a backtracking based method (Algorithm 5) to find the best combination of matches.

Algorithm 5 Graph-based Search

Require: List of all matches {Sorted in descending order of area of the fitting region, for each pair of polygons}
Ensure: The solution to the fitting problem
Transform the polygon pair having the highest area of the fitting region
if Polygons do not intersect **then**
  Add transformed polygons to the solution
else
  Backtrack by removing the intersecting pair and checking the next pair in List of all matches
end if

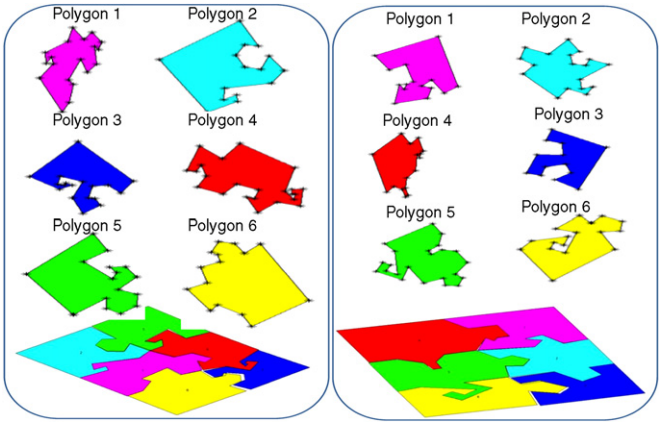

Fig. 8. Results of our algorithm on other examples of the exact unique matching case.

Table 1
Summary of the results of the experiments in Figs. 7–9 (compute times are in seconds).

Number of features	Hierarchy depths	Hierarchy build time	Matching time	Fitting time
16, 22	8, 8	1.5	0.25	0.07
28, 14, 19, 33, 25, 17	4, 5, 4, 4, 5, 3	3.22	9.26	0.04
19, 26, 13, 28, 31, 22	4, 3, 4, 5, 4, 4	2.85	8.54	0.07
24, 34, 33, 50	4, 6, 4, 6	3.31	5.84	0.02

5. Test cases and results

Exact unique match: In an exact unique match, every input polygon has unique negative and positive space polygons. As a result, each negative space in a piece matches with exactly one positive space in another piece and vice-versa. Figs. 7 and 8 illustrate this. Table 1 shows the time taken to compute the hierarchy, matching and fitting for the examples shown in this paper.

Exact non-unique match: In an exact non-unique match, there can be polygons that have non-unique cavities or protrusions. As a result each cavity in a piece can match with more than one protrusion in other pieces and vice-versa. This produces ambiguous matches (Fig. 9).

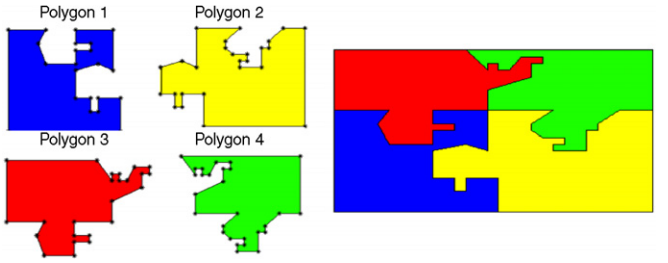

Fig. 9. Results of our algorithm on an example of the exact non-unique matching case.

Limitations of our method: In a general problem setting, any sequence of edges of two polygons can fit with each other. This requires comparing the exhaustive list of edge sequences of all lengths. Our method prioritizes specific edge sequences to be more descriptive of the geometric features and arguably more likely to fit with each other. Since in a general case, this problem is NP-hard, obviously there are many instances of this problem in which those edge sequences that are not chosen by our method are used in the final fitting. On the other hand, any sequence of edges that is found suitable for a particular application can be added to the hierarchy, along with its cavity–protrusion analysis. This hierarchy can then be used in the fitting process. In that sense, our fitting algorithm using subtree isomorphism is general enough to handle all exact fits.

If all are convex pieces, then there is no cavity to build the hierarchy. Since in such a situation, no more than one edge of each of the two polygons will be matched to each other, an exhaustive search for these edge pairs would be the best way to address this problem.

6. Summary

In this paper, we propose a novel algorithm for fitting polygonal shapes by constructing hierarchies of geometric features in polygons. We use it to perform pairwise shape matching, followed by a modified tree isomorphism algorithm to perform early rejection, thus reducing the search space. We selectively compare geometric parameters such as area, perimeter, and edge lengths of the decomposed polygons. We use a backtracking-based graph search algorithm to generate the solution of the fitting problem.

References

[1] Dyckhoff H. A typology of cutting and packing problems. European Journal of Operational Research 1990;44(2):145–59.
[2] Lodi Andrea, Martello Silvano, Monaci Michele. Two-dimensional packing problems: a survey. European Journal of Operational Research 2002;141(2):241–52.
[3] Demaine ErikD, Demaine MartinL. Jigsaw puzzles, edge matching, and polyomino packing: connections and complexity. Graphs and Combinatorics 2007;23(1):195–208.
[4] Tanase M, Veltkamp RC, Haverkort H. Multiple polyline to polygon matching. Lecture notes in computer science 2005;3827:60.
[5] Webster RW, Ross PW. A computer vision system that assembles canonical jigsaw puzzles using the euclidean skeleton and isthmus critical points. In: Proc. IAPR workshop machine vision applicat. 1990 p. 28–30.
[6] Gallagher AndrewC. Jigsaw puzzles with pieces of unknown orientation. In: Proceedings of the 2012 IEEE conference on computer vision and pattern recognition (CVPR). Washington (DC, USA): IEEE Computer Society; 2012. p. 382–9.
[7] Cho Taeg Sang, Avidan Shai, Freeman William T. A probabilistic image jigsaw puzzle solver. In: CVPR. 2010. p. 183–90.
[8] Lam TF, Sze WS, Tan ST. Nesting of complex sheet metal parts. Computer-Aided Design & Applications 2007;4:169–79.
[9] Yao LH, Jun HY. NFP-based nesting algorithm for irregular shapes. In: Proceedings of the 2006 ACM symposium on applied computing. ACM; 2006. p. 963–7.
[10] Burke EK, Hellier RSR, Kendall G, Whitwell G. Complete and robust no-fit polygon generation for the irregular stock cutting problem. European Journal of Operational Research 2007;179(1):27–49.
[11] Adamowicz M, Albano A. Nesting two-dimensional shapes in rectangular modules. Computer-Aided Design 1976;8(1):27–33.
[12] Kong W, Kimia BB. On solving 2D and 3D puzzles using curve matching. In: Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition, 2001, vol. 2. IEEE; 2001. p. II-583.
[13] Lamousin H, Waggenspack WN, et al. Nesting of two-dimensional irregular parts using a shape reasoning heuristic. Computer-Aided Design 1997;29(3):221–38.
[14] Buss SamuelR. Alogtime algorithms for tree isomorphism, comparison, and canonization. In: Kurt Gödel colloquium. 1997. p. 18–33.