Making Many Polygons by Simple Fold and One Straight Cut

Hu Guoxin, Uehara Ryuhei
Uehara Lab
School of Information Science
Japan Advanced Institute of Science and Technology

Outline

1. Fold-and-cut problem and the solutions.

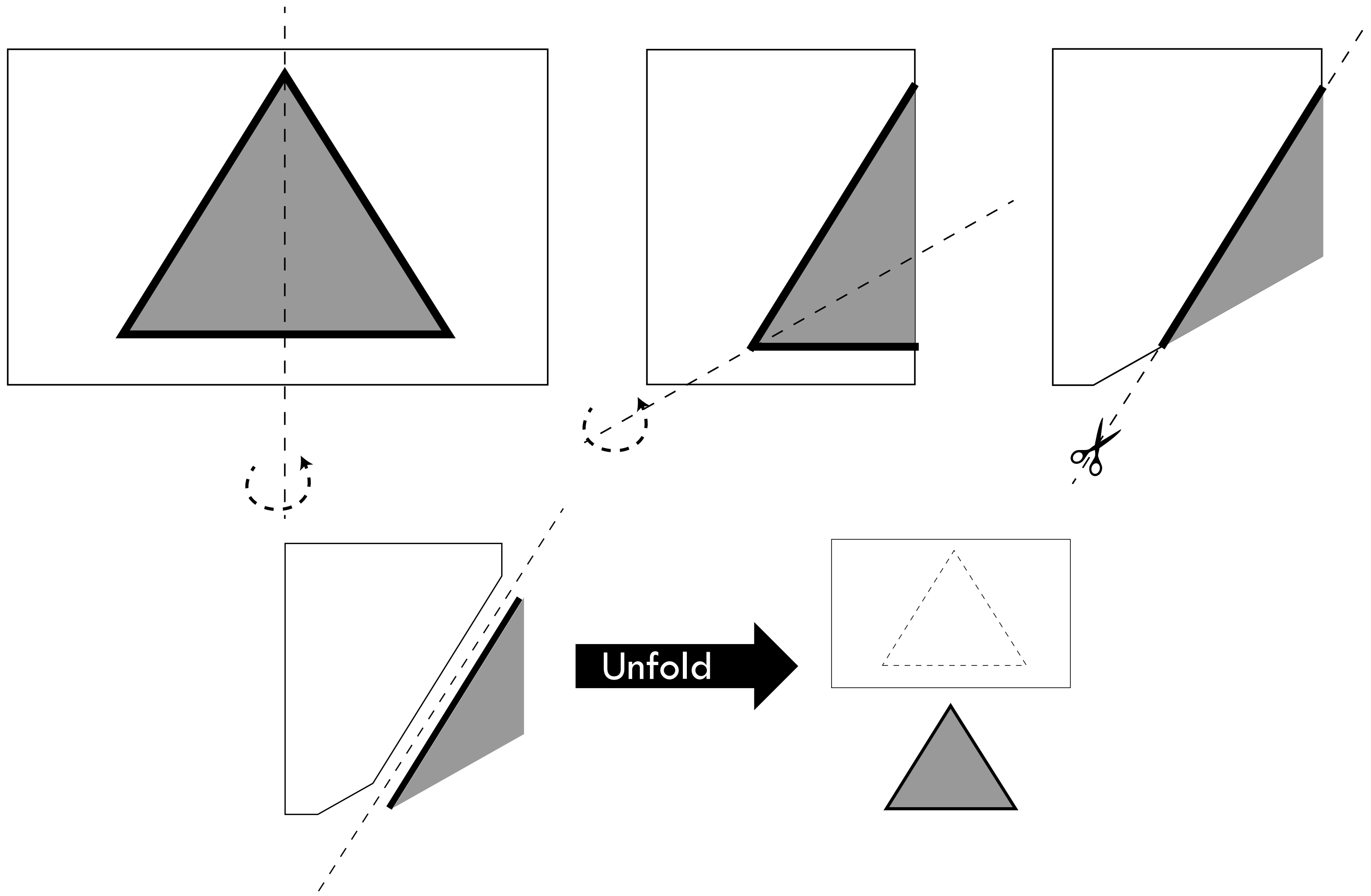
2. Simple fold and cut problem and the solution for one simple polygon

3. Our algorithm to solve two simple polygons

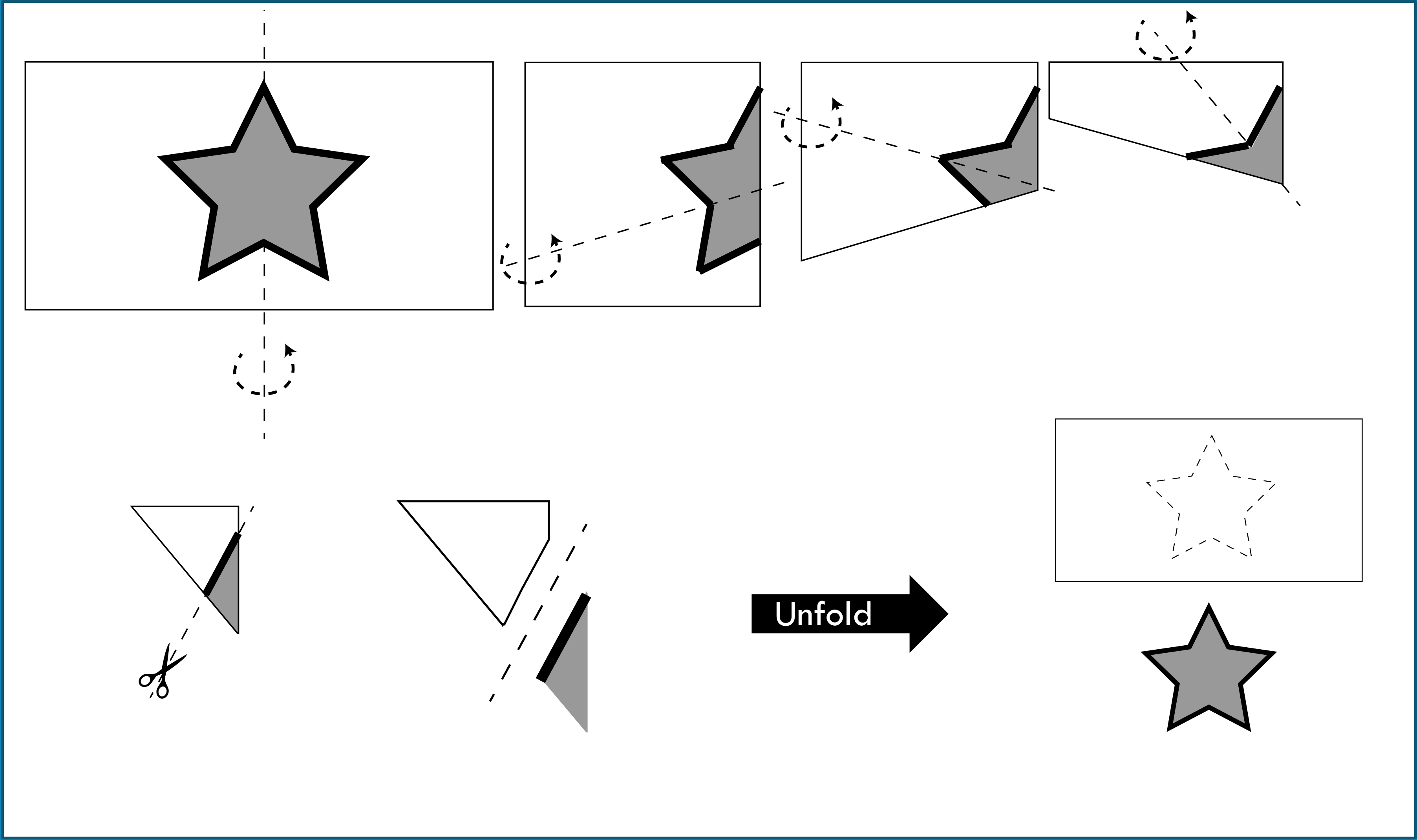
Fold-and-cut problem

Take a sheet of paper, fold it flat however you like and make one complete straight cut, what shapes can you get from the unfolded pieces.

Example



Example



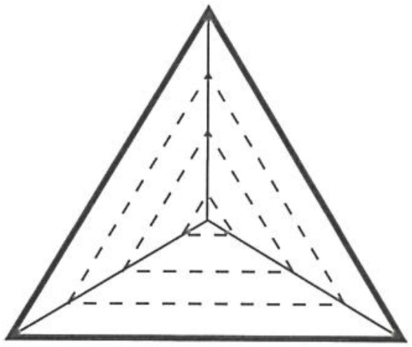
Generic Result

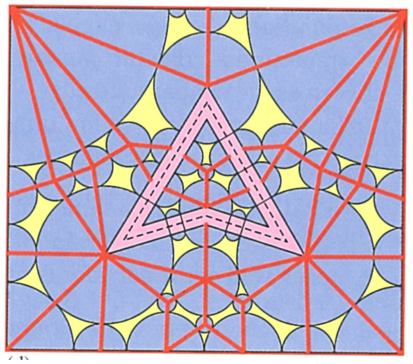
Take a sheet of paper, fold it flat however you like and make one complete straight cut, what shapes can you get from the unfolded pieces.

The outcome to the fold-and-cut problem is a universality result:
Every plane graph of desired cuts can be made by folding and one complete cut

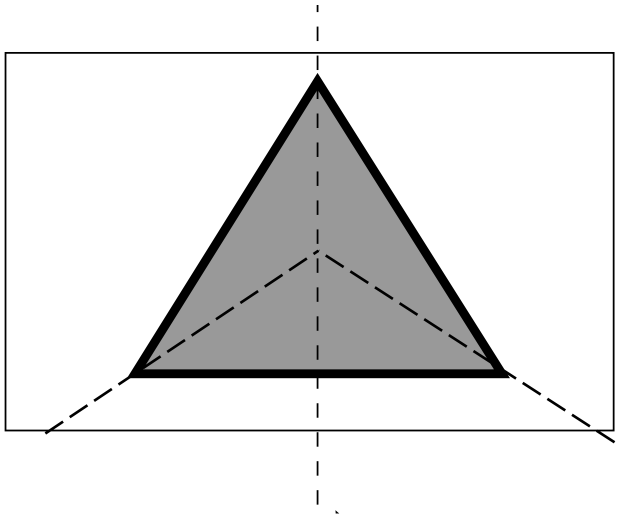
There are two algorithms to handle the generic cases, straight skeleton and disk packing

Problem


Straight Skeleton

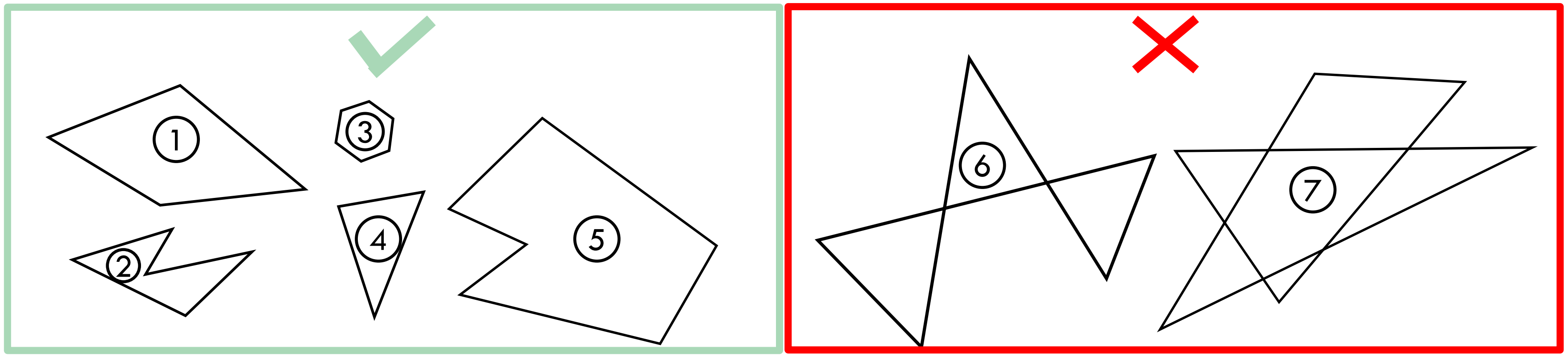

Disk Packing

By limiting graphs to simple polygons and folds to all-layers simple fold



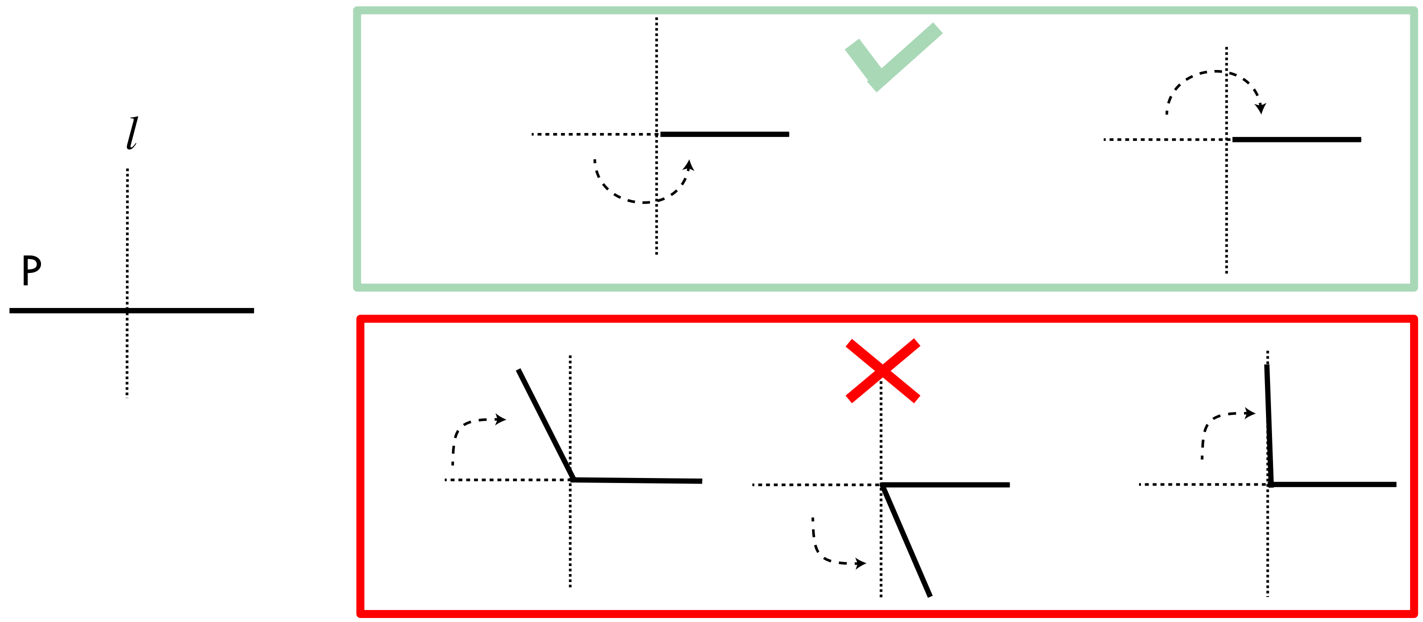
Simple Polygon

In geometry, a **simple polygon** is a flat shape consisting of **straight**, **non-intersecting** line segments or "sides" that are joined pair-wise to form a closed path.



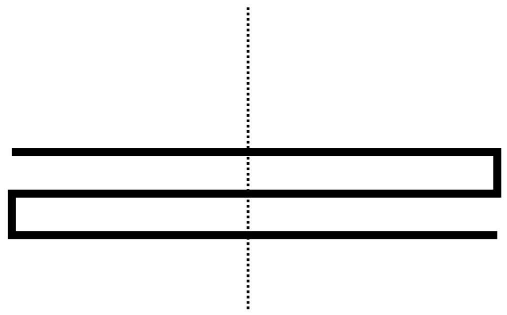
Simple Fold

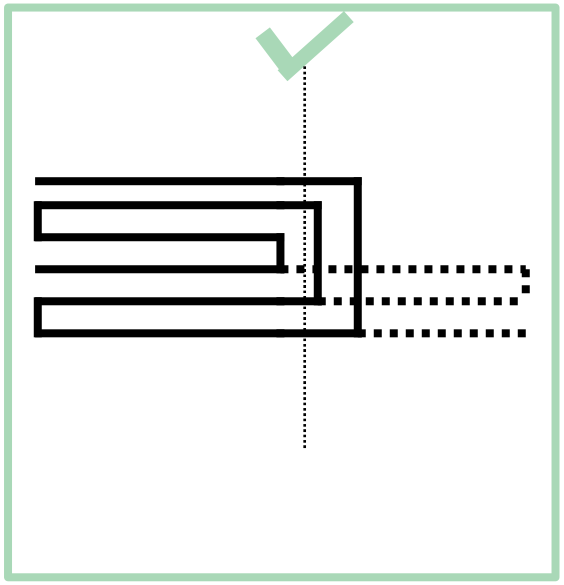
You can only fold the paper by ±180°

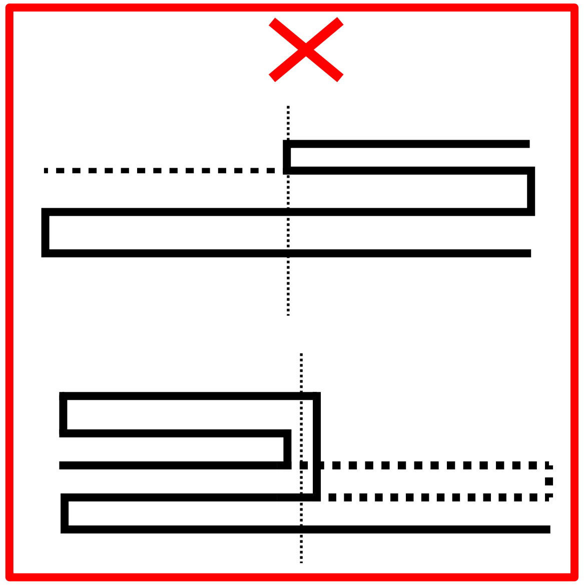


All-layer Simple Fold

All-layer simple fold will fold through all layers of the paper by $\pm 180°$







Algorithm for solving one simple polygon

Input:

Configuration (G, P). Simple polygon G drawn on the paper P. Initially, the paper P contains the convex hull of G

Output:

Whether (G, P) is simple-fold-and-cuttable after a sequence of feasible all-layers simple folds

Feasible fold:

Every point of G must be folded to either another point of G or a point outside P

Algorithm(Demaine et al. 2011)

Theorem (Demaine et al. 2011)

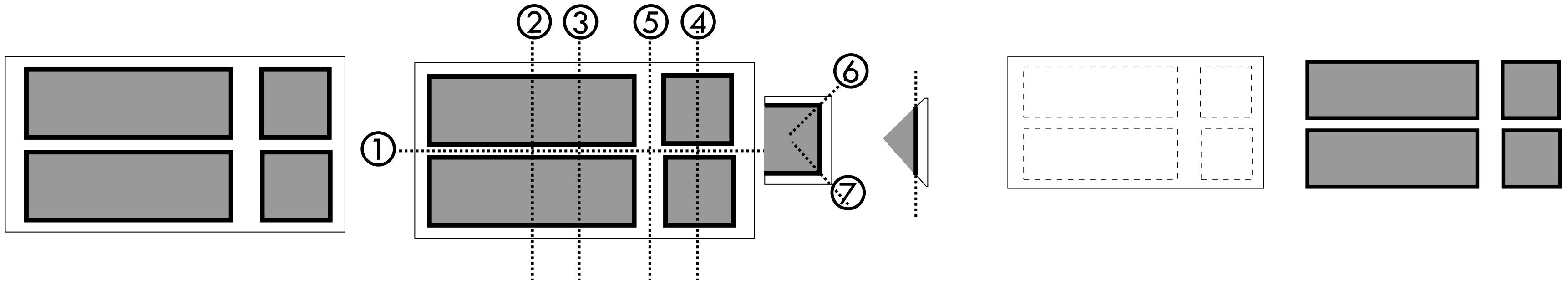
There is a strongly polynomial-time algorithm for determining whether a given (not necessarily convex) simple polygon is simple-fold-and-cuttable, starting from a piece of paper strictly containing the convex hull of the polygon

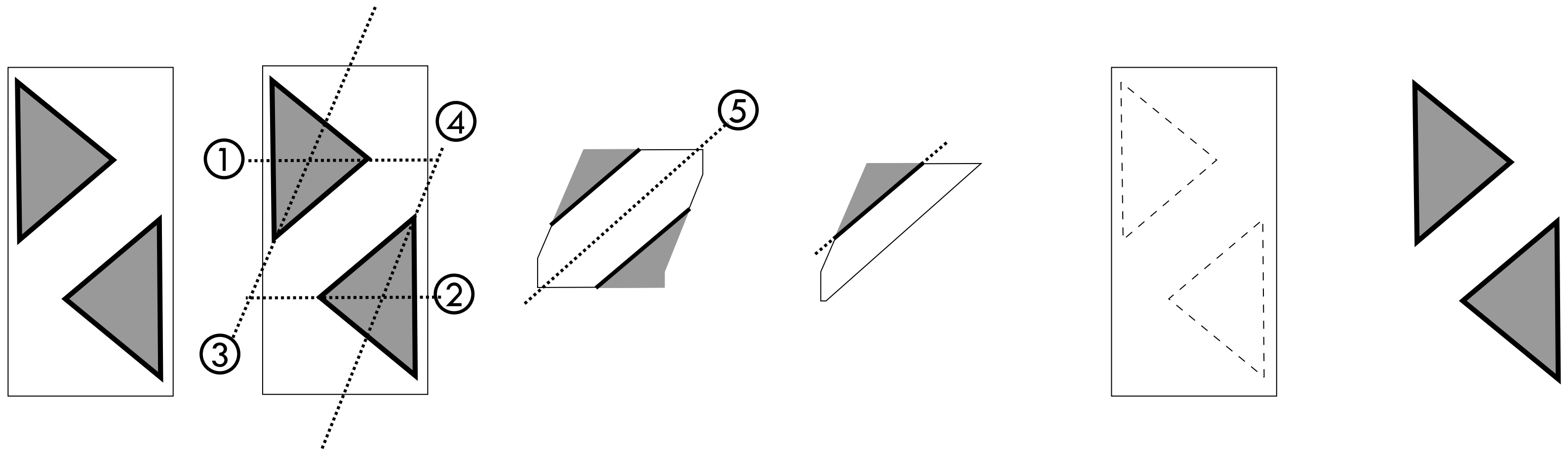
The time complexity for this algorithm is $O(n^2)$

Reference

Demaine, E. D., Demaine, M. L., Hawksley, A., Ito, H., Loh, P. R., Manber, S., & Stephens, O. (2011). Making polygons by simple folds and one straight cut. In *Computational Geometry, Graphs and Applications* (pp. 27-43). Springer, Berlin, Heidelberg.

Observation





How about extending to many simple polygons ?

Our Algorithm for two simple polygons

Input:

A configuration (G_1, G_2 , P). G_1 and G_2 are simple polygons drawn on paper P. They do not intersect with each other and, initially, the paper P contains the convex hull of G_1 and G_2

Output:

Whether configuration (G_1, G_2 , P) is all-layers simple fold and straight cuttable or not after a sequence of feasible all-layers simple folds

Feasible fold:

Every point of G_1, G_2 must be folded to either another point of G_1, G_2 or a point outside paper P

Algorithm outline

1. Apply algorithm(Demaine et al. 2011) for both G_1 and G_2

2. Preprocessing and define some variables.

3. If two simple polygons are congruent and with sufficient distance, we will try to overlap them

4. Otherwise, we will apply the greedy algorithm

Algorithm Step 1

1. Apply algorithm(Demaine et al. 2011)

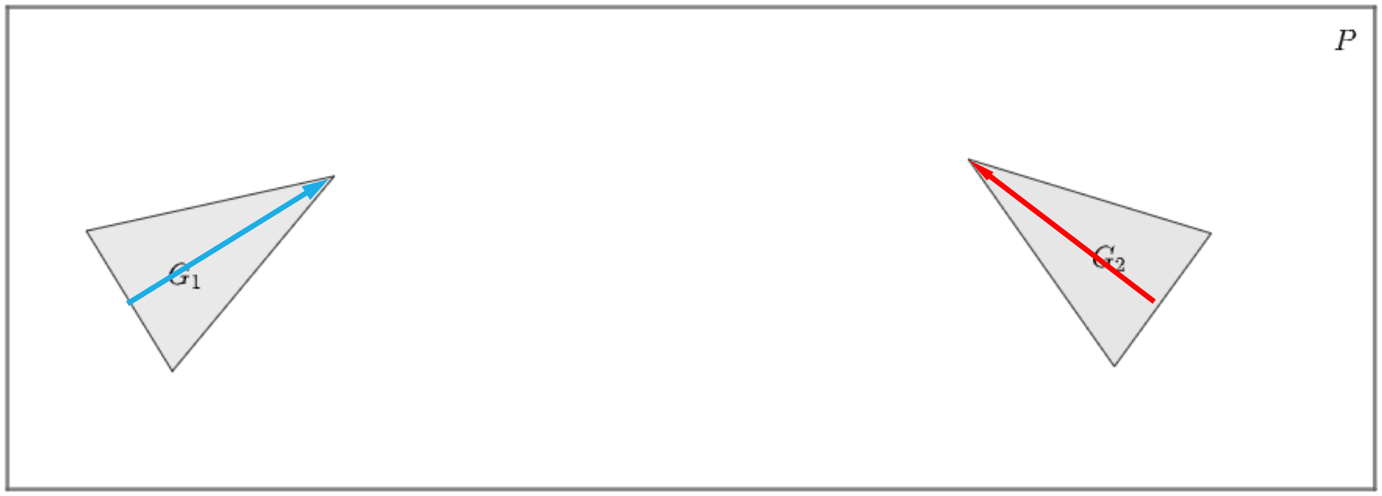
Theorem (Demaine et al. 2011)
There is a strongly polynomial-time algorithm for determining whether a given (not necessarily convex) simple polygon is simple-fold-and-cuttable, starting from a piece of paper strictly containing the convex hull of the polygon

If one of the simple polygon is not simple-fold-and-cuttable, then the configuration (G_1, G_2, P) will not be simple-fold-and-cuttable.

Preprocessing

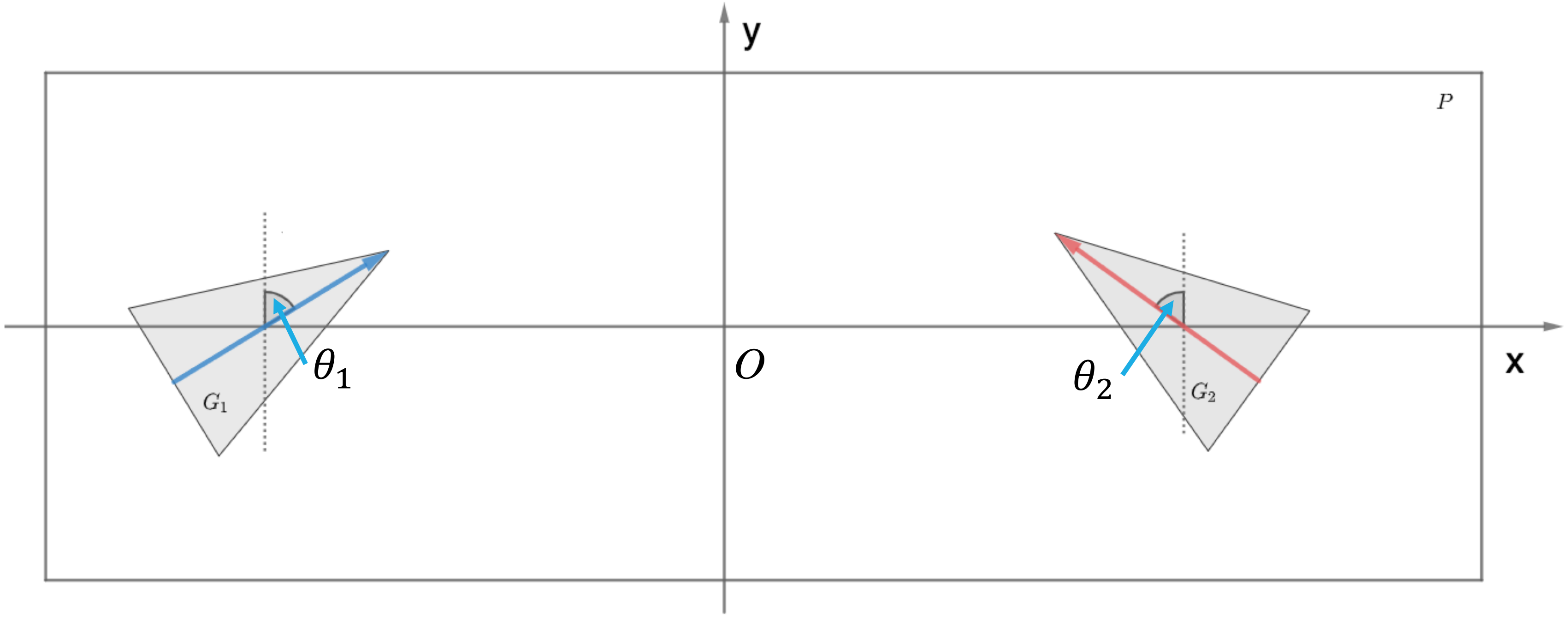
1. View reflectional symmetry lines as vectors.

If simple polygon G passes algorithm(Demaine et al. 2011), it means that G must contain at least one reflectional symmetry line



Preprocessing

2. Build a coordination system with origin at the middle of the center point of two reflectional symmetry lines and define two angles θ_1 and θ_2

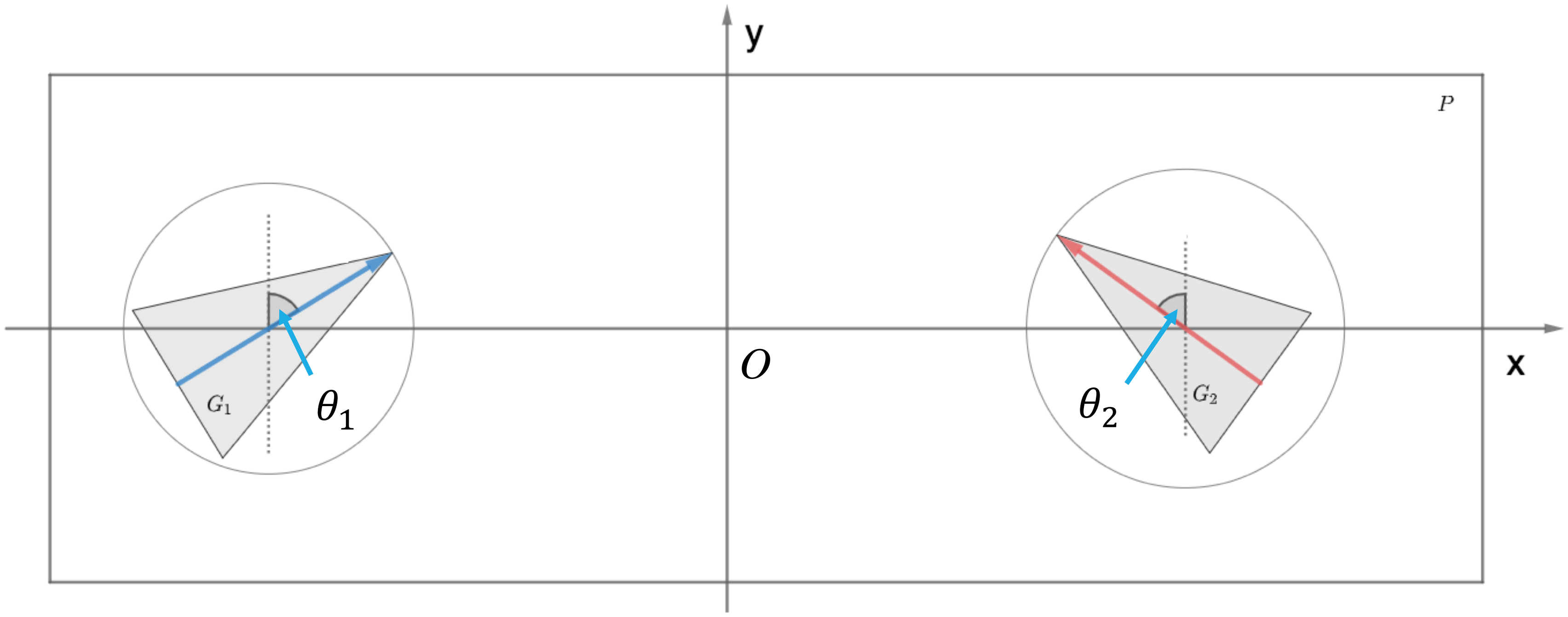


Clockwise positive

Counterclockwise negative

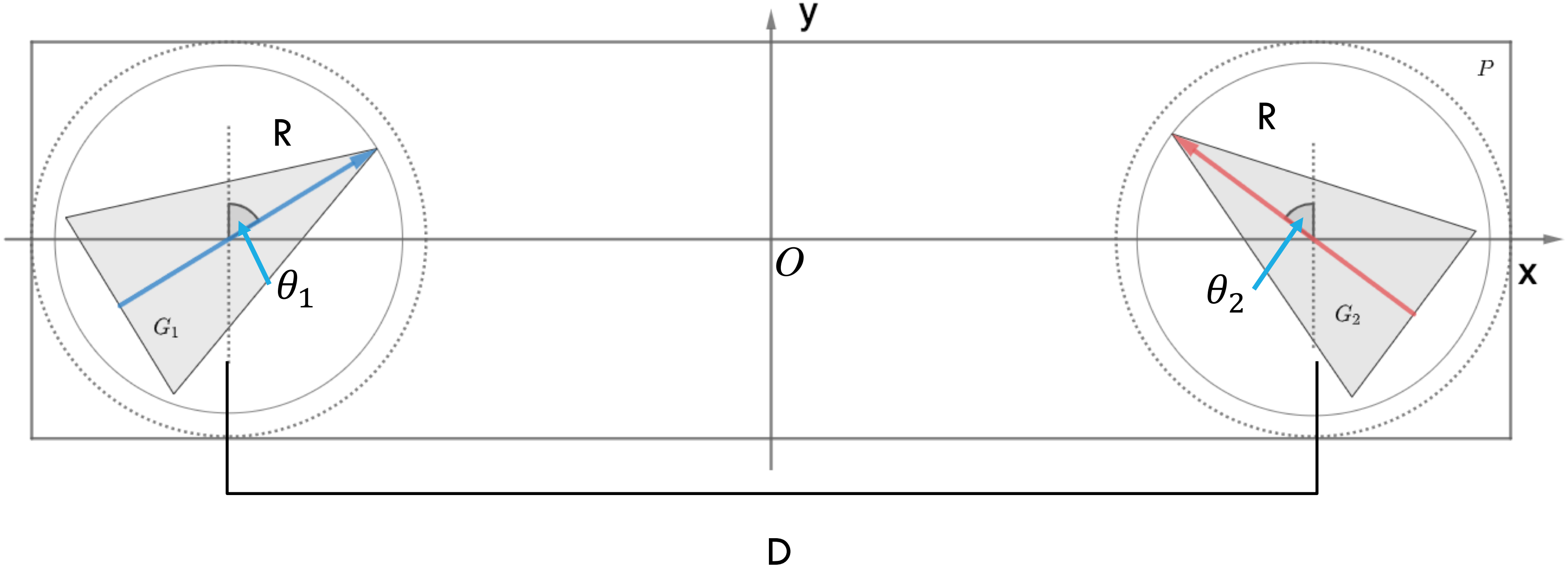
Preprocessing

3. Encapsulate two simple polygons with disks and the radius R will be the distance between the center point of reflectional symmetry line and the farthest point of the polygon



Preprocessing

4. Shrink the paper P ε away from the disks for any given small ε, and define D as the distance between the center point of two reflectional symmetry lines.



Algorithm outline

1. Apply algorithm(Demaine et al. 2011) for both G_1 and G_2

2. Preprocessing and define some variables.

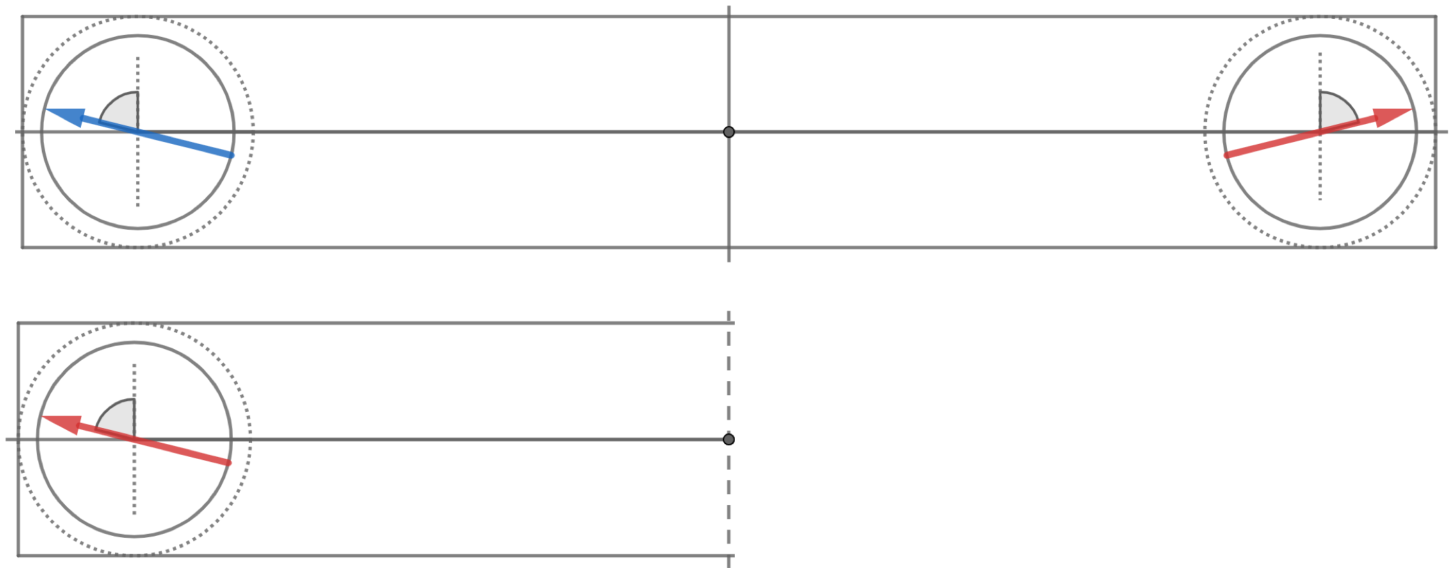
3. If two simple polygons are congruent and with sufficient distance, we will try to overlap them

4. Otherwise, we will apply the greedy algorithm

Algorithm Step 3

2. If two simple polygons are congruent, we will try to overlap them

If $\theta_1 + \theta_2 = 0$, we only need to fold once to overlap two simple polygons and the necessary distance will be D $> (2R + \varepsilon)$

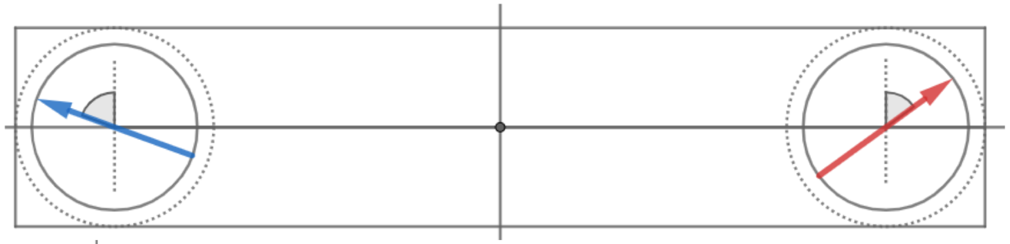


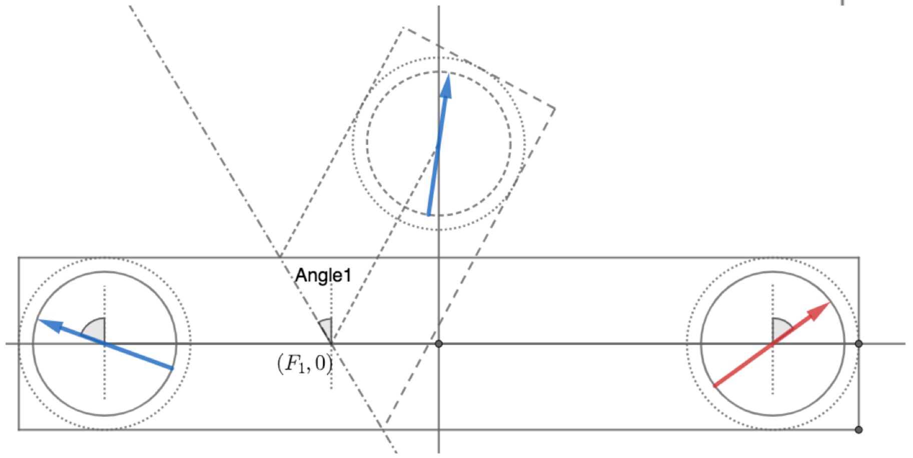
Algorithm Step 3

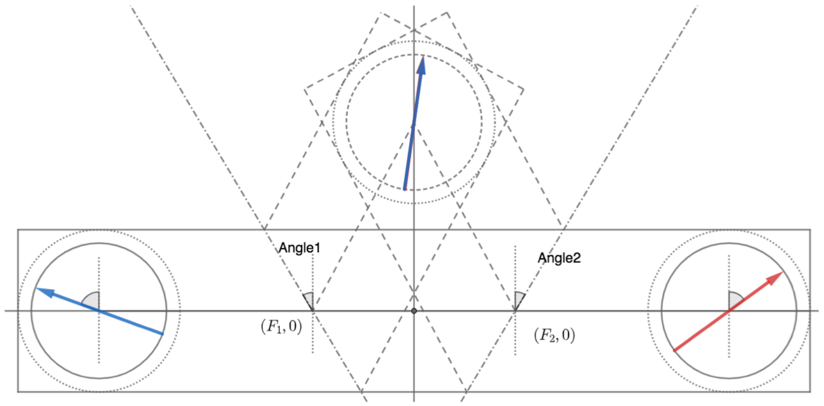
2. If two simple polygons are congruent, we will try to overlap them

If $\theta_1 + \theta_2 \neq 0$, then we need to fold twice to overlap two simple polygons and the necessary distance is $\boxed{D > 2(2R + \varepsilon)\left(\cot\theta + \frac{1}{\sin\theta}\right), \text{ where } \theta = \left|\frac{\theta_1 - \theta_2}{2}\right|}$

$$|\theta_1| - |\theta_2| \to 0$$
$$\theta \to 0$$
$$D \to \infty$$



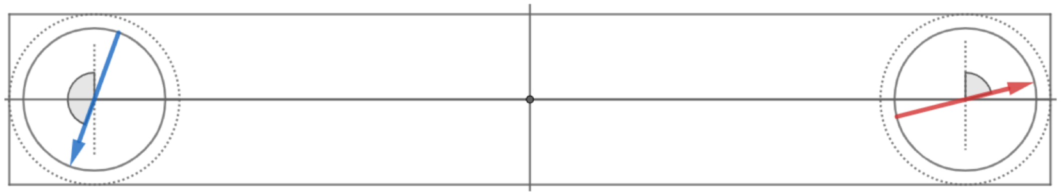


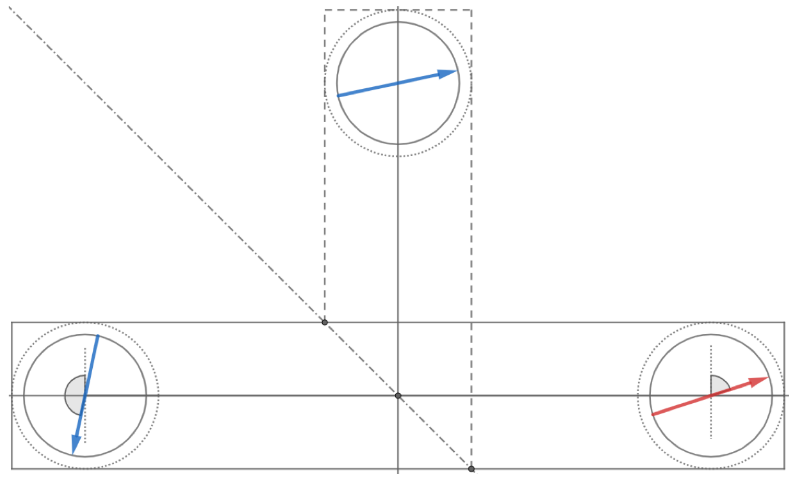


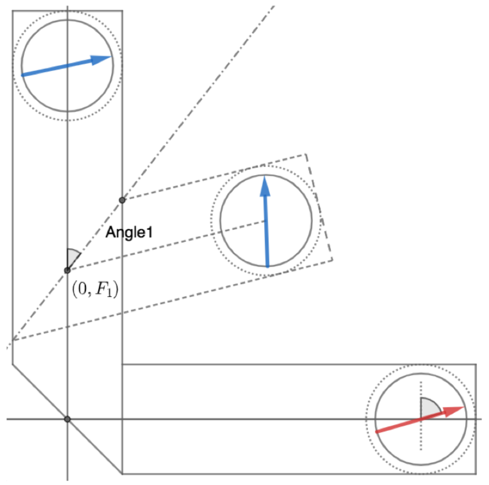
Algorithm Step 3

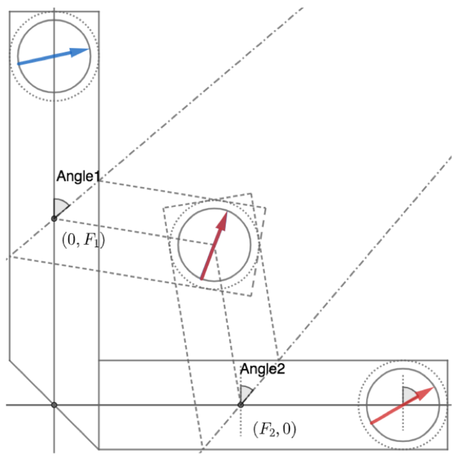
2. If two simple polygons are congruent, we will try to overlap them

For those special cases, we need to fold three times to overlap two simple polygons









Algorithm outline

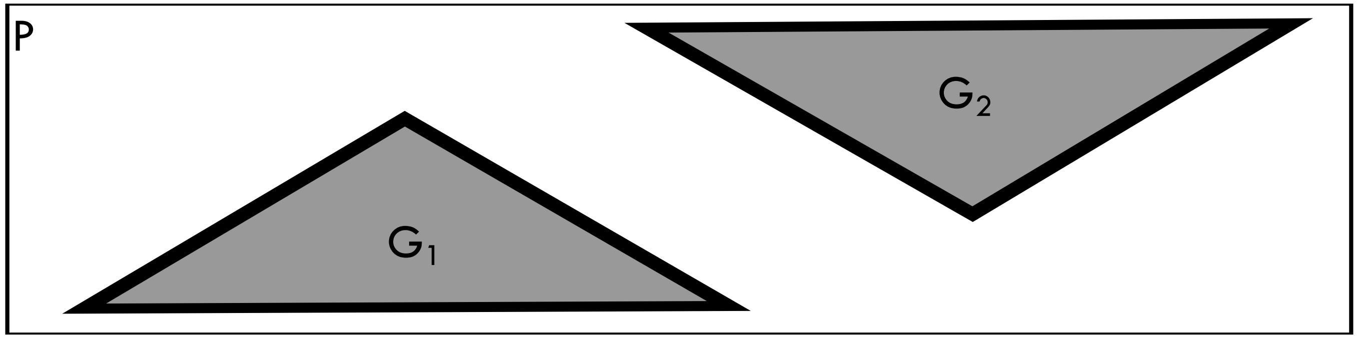
1. Apply algorithm(Demaine et al. 2011) for both G_1 and G_2

2. Preprocessing and define some variables.

3. If two simple polygons are congruent and with sufficient distance, we will try to overlap them

4. Otherwise, we will apply the greedy algorithm

Algorithm Step 4, Greedy algorithm

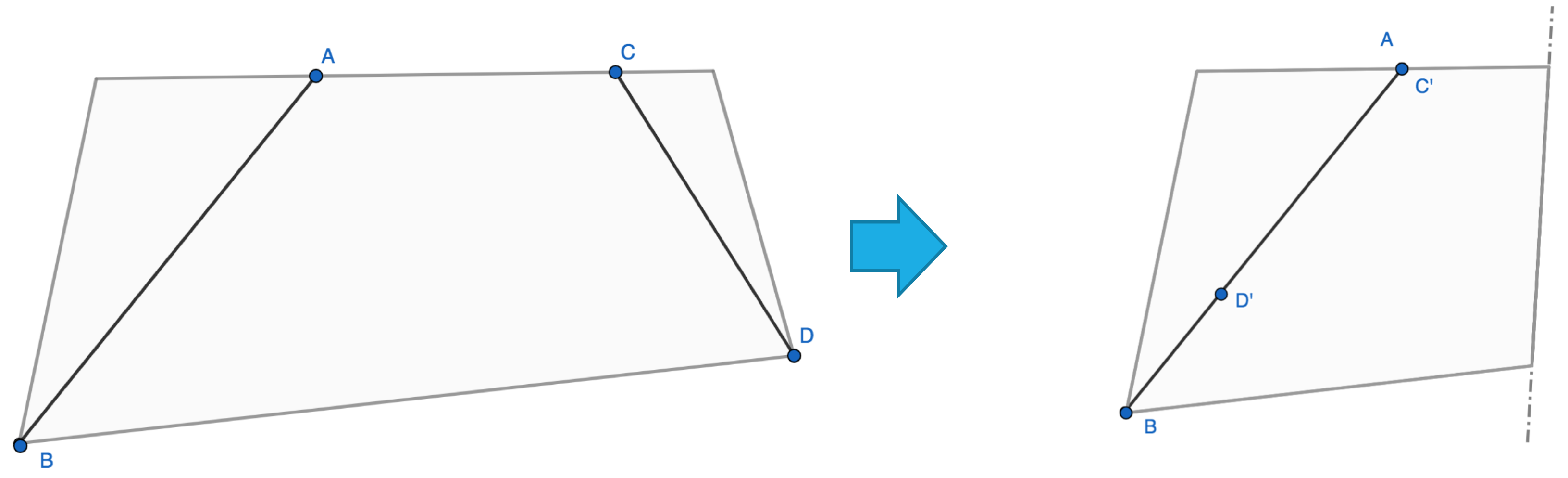


1. Search for feasible folds on G_1 and fold it.
2. Search for feasible folds on G_2 and fold it.
3. Fold the paper between G_1 and G_2
4. Shrink paper P into the convex region of G_1 and G_2

Until we fold two simple polygons into lines or we cannot find feasible folds any more.

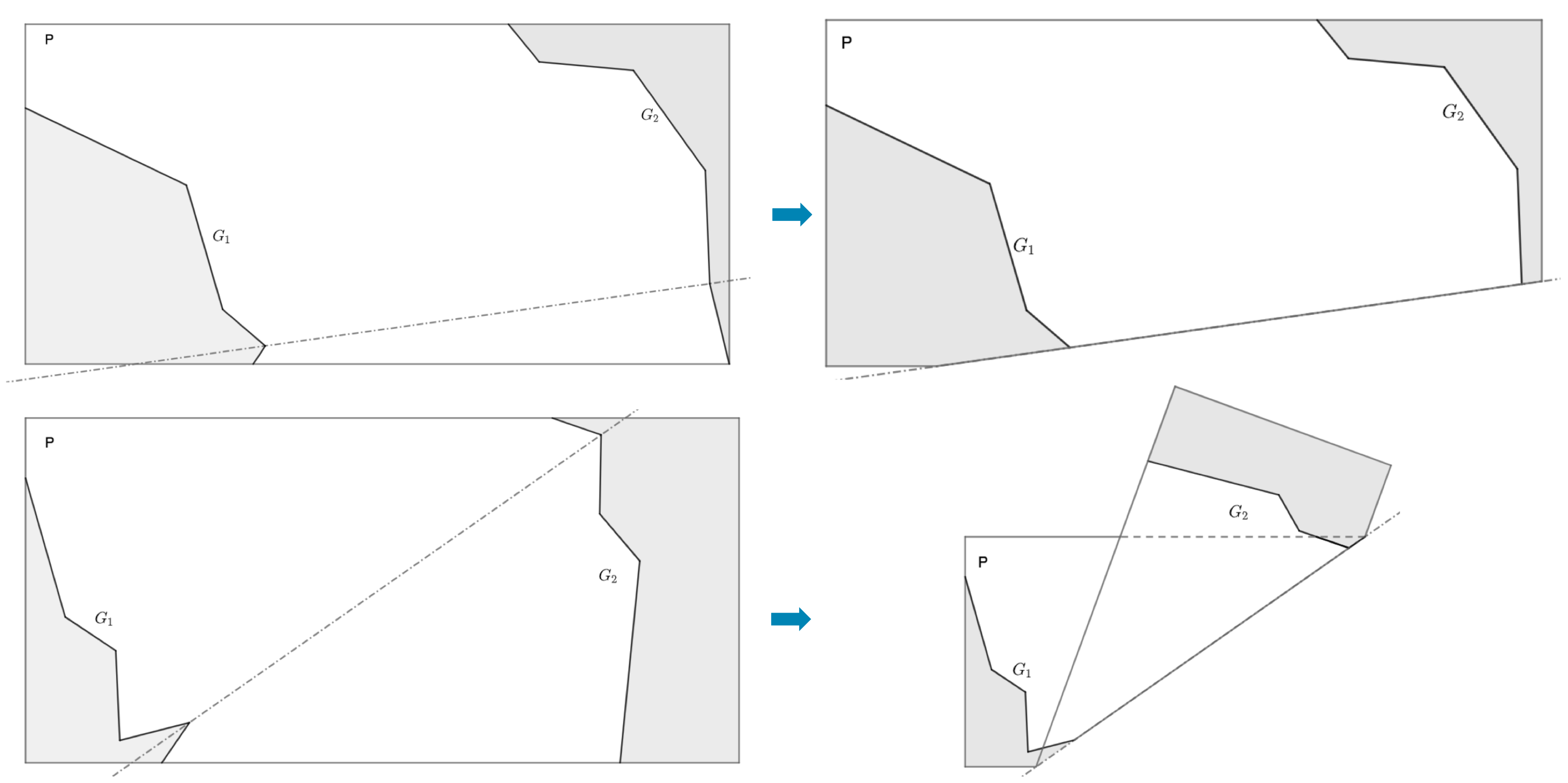
Algorithm Step 4

Any two line segments (without intersection) with their endpoints on the boundary of the paper P (not necessary convex), can always be overlapped into one line segment by a sequence of all-layers simple fold.



Two states

During the folding process, the paper will change between two states, whether paper contains the convex hull of two simple polygons or not.



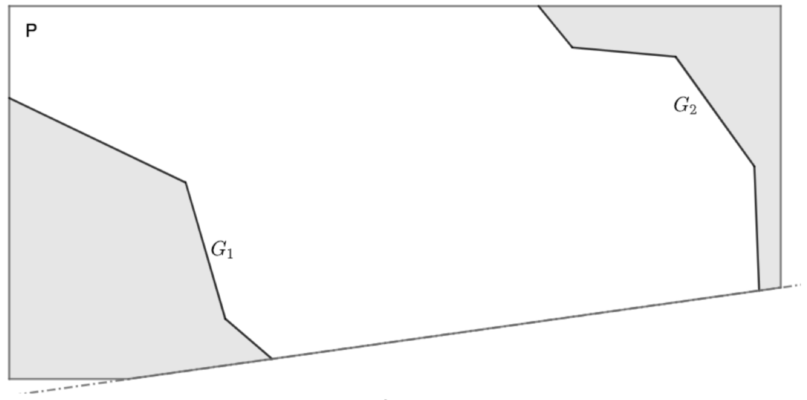
Two states

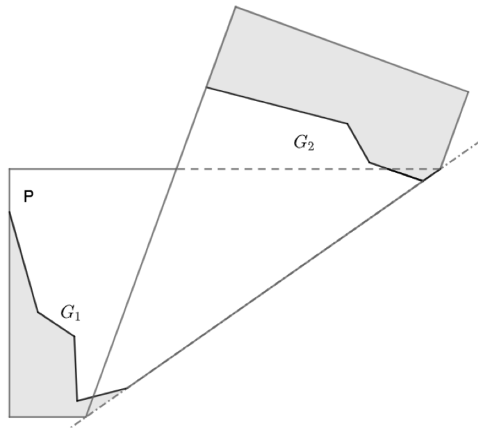
During the folding process, the paper will change between two states, whether paper contains the convex hull of two simple polygons or not.

We prove that it is always “safe” to fold when paper P contains the convex hull of G_1 and G_2

A safe fold is a fold which will not turn a feasible fold to an infeasible fold.

However, whether is always “safe” to fold when paper P does not contain the convex hull is still a question ?





Time complexity

1. Apply algorithm(Demaine et al. 2011) for both G_1 and G_2 — $O(n^2)$

2. Do some preprocess and define some variables. — $O(1)$

3. If two simple polygons are congruent and with sufficient distance, we will try to overlap them — $O(1)$

4. Otherwise, we will apply the greedy algorithm — $O(n^2)$

Therefore, the time complexity for our algorithm will be $O(n^2)$

Future work

1. Prove that it is always “safe” to fold even the paper does not contains the convex hull of the simple polygons

2. Prove the NP hardness for n simple polygons

Reference

1. Demaine, E. D., Demaine, M. L., Hawksley, A., Ito, H., Loh, P. R., Manber, S., & Stephens, O. (2011). Making polygons by simple folds and one straight cut. In *Computational Geometry, Graphs and Applications* (pp. 27-43). Springer, Berlin, Heidelberg.