Finding the Maximum Area Centrosymmetric Polygon in a Convex Polygon

December 23, 2011

Contents

1 Introduction **3**

2 Notation and Technique **4**

3 Finding the MACP in a Convex Polygon **7**

4 Correctness and Complexity **9**
4.1 Correctness of the Algorithm 9
4.2 Complexity Analysis 13

5 Experiment **14**
5.1 Experimental Algorithm 15
5.2 Experimental Design 17
5.3 Experimental Operations 19
5.4 Experimental Data and Result 21
5.5 Experimental Experience 22

6 Conclusion and Discussion **23**

7 Acknowledgement **24**

1 Introduction

Finding the largest figure of some property contained in a given figure has been a common and important problem in computational geometry in the past 30 years. Many efforts have been made on this field to achieve good results. Researchers have looked at several instances of this problem such as Dobkin et al generated an algorithm to find the largest triangle inscribed in a convex polygon in [1], Boyce et al developed a method to compute the largest k-gon in a convex polygon in [2]. Also Finding the largest convex polygon and the largest axis-parallel rectangle in an arbitrary polygon was involved in [4] and [18], while [5] telling how to find the largest square in a convex polygon. Recently this problem has also attracted many researchers' attention such as in [19] Kin Jin improved the way to find the maximum area parallelogram in a convex polygon with complexity $O(n^2)$ which is a more surprising result. In this report, we also take one instance of the problem into our consideration: finding the maximum area centrosymmetric polygon in a convex polygon.

Finding the maximum area centrosymmetric polygon (MACP) problem can be defined: given a convex polygon in the plane, find the largest polygon in the convex polygon that is centrally symmetric. Here centrally symmetric means there exists a "center" such that for every point on the polygon, when reflecting it about the center we can produce another point which must be on the polygon). This special problem was first introduced in [19] that an $\frac{2}{\pi}$-approximation algorithm given by finding the maximum area parallelogram. In this paper, we give the way to find the largest centrally symmetric polygon in the convex polygon more quickly and precisely.

MACP problem attract our attention because the resemblance is very important in research. We know that matching plays an important role in many areas such computer vision and motion planning. Generally speaking, when we are given two figures, try to determine how much the two figures resemble each other, that is: we want to find a rigid motion of one figure that maximize the resemblance with the other figure. Many ways have been used such as Hausdorff distance in [6, 9, 10, 14, 15] or Frechet distance in [11] is designed to detect the distance between the two figures' boundaries and maximizing the area of the overlap of them is also another appropriate way because we can minimize the area of the symmetric difference by finding the largest overlap. Thus we choose this topic to find how symmetric the given convex polygon is by finding the largest centrosymmetric polygon in it which is meaningful.

The rest of this report is structured as follows: We start by introducing some important notations and techniques in section 2. Then we propose the way how to find the maximum area centrosymmetric polygon theoretically in section 3. The correctness of the method is analyzed in Section 4 and Section 5 gives the experimental result with the details in designing such practical search to find the largest centrally symmetric polygon. Finally we conclude this report with some future improvements may be made.

2 Notation and Technique

The general way to find such centrosymmetric polygons can be finding the "center" firstly. However, it's hard to fix the appropriate "center" since there can be many such polygons when we don't care about the area of them. Thus we need some new techniques to tackle with. Considering the resemblance between two figures, we can find their largest overlap to achieve and this may be a good idea to use. Thus we consider whether we can use the same method when we care about the resemblance. The answer is YES. So we keep our method as follows: For the given convex polygon P, find the centrosymmetric convex polygon Q, move Q on the plane to intersect with P. If we can find the maximum area of the overlap of P and Q, we also get the largest centrally symmetric polygon in P (This will be proved in section 4). In this section, we introduce some important notations to stand our method of finding the maximum area centrosymmetric polygon.

From the way above, suppose the original convex polygon P is fixed while the centrally symmetric Q is free to translate and move. Thus define the reference point of the convex polygon Q:

Definition 2.1. *Let r_Q be the reference point of Q when it's the lexicographically smallest point of all the vertices in Q.*

After defining the reference point of the convex polygon Q, we can translate Q to a given state by moving the reference point to some given responding point and compute the area of the overlap. Thus we can have the following definition of placement:

Definition 2.2. *For a point r in the plane, let $Q(r)$ denotes Q when r_Q is placed at the point r, we call $Q(r)$ a placement of Q.*

Similarly, we can let $e(r)$ and $v(r)$ denote the edge e and vertex v of Q when its reference point is placed at r. Now we will see how to present the intersection set of P and $Q(r)$.

Definition 2.3. *Let $I(r)$ denote the intersection set of P and a placement $Q(r)$ consists of all pairs (f, g) where f is the interior, an edge or a vertex of P, g is the interior, an edge or a vertex of $Q(r)$ and f intersects g.*

Since we have the description of the intersection set about two convex polygons, call two placements $Q(r_1)$ and $Q(r_2)$ are combinatorially distinct if the intersection sets are different i.e $I(r_1) \neq I(r_2)$. Now look at the following figures: Figure 2 shows that the two placements are the same because they have the same intersection set as we defined above. Thus we see the two points are in the same region because their placements are combinatorially equivalent.

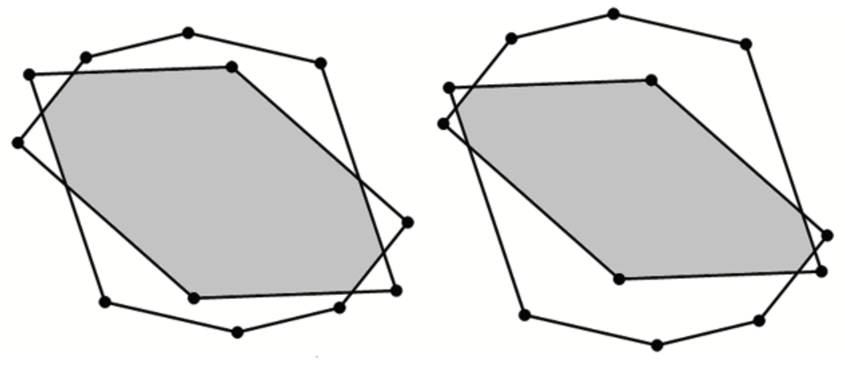


Figure 1: Example of Equivalent Placement

Figure 2 gives an example of combinatorially distinct placements.

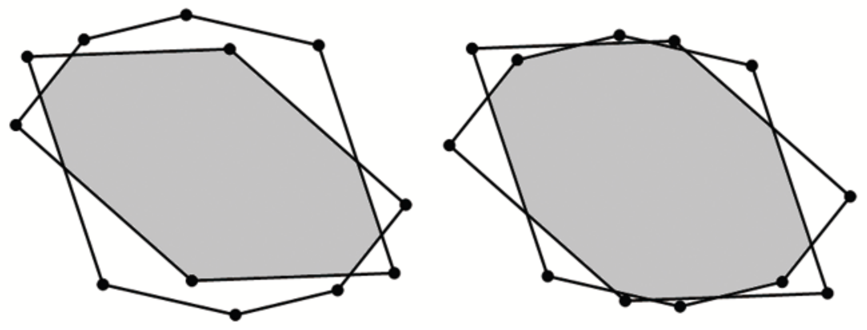


Figure 2: Example of Distinct Placement

If we sum up all the possible placements of Q which are combinatorially distinct from each other, they will be a 2-dimensional space which will be defined as:

Definition 2.4. *We call all the possible combinatorially distinct placements of Q form the configuration space.*

From the definition of *configuration space*, we can divide the plane into some regions where two points are in the same region when their placements are combinatorially equivalent. Thus, we can use configuration to bound the number of distinct placements which will be meaningful.

Configuration space has been researched by many scientists. It's inspired by motion planning how a robot can move with no collide with some given polygon where the region of possible configuration space is called free space. [7] gives the information about how to connect configuration space and motion planning, while research into placements that the intersection set is not empty is given in [8]. The connection about the placement and the overlap has been also researched by many interested researchers.

What's more, [16] shows that: The maximum number of combinatorially distinct placements of two convex polygons with n and m vertices respectively can be bounded by $\Theta(n^2 + m^2 + \min(nm^2, mn^2))$. Considering our problem, if we translate the centrally symmetric polygon Q on the given P, we can also have the bound of maximum combinatorially distinct placements number as $\Theta(n^3)$ and we can also construct a polygon that suits the lower bound which means there is indeed some polygon P can have $\Omega(n^3)$ distinct placements with its centrally symmetric convex polygon. The special case is given in Figure 2 and it can be verified easily.

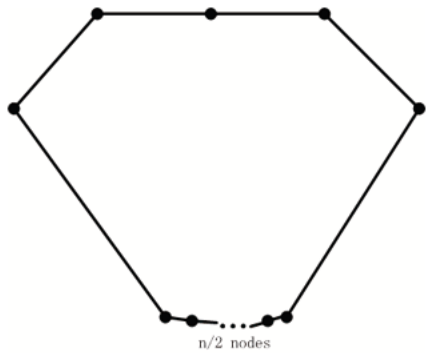


Figure 3: Example of a special convex polygon which has $\Omega(n^3)$ distinct placements with its centrally symmetric polygon

From now on, we will give some notations and methods about the way to compute the maximum overlap.

Definition 2.5. *Given a non-negative function f, we call f unimodal if there exists an interval $S = [a_0, a_1]$ and two points $b_0, b_1 \in S, b_0 \leq b_1$ such that:*

- *For any point x not in S, which means $x < a_0$ or $x > a_1$, $f(x) = 0$;*
- *For any two points $a_0 \leq x_1 < x_2 < b_0$, we have $f(x_1) < f(x_2)$;*
- *For any two points $b_0 \leq x_1 < x_2 \leq b_1$, we have $f(x_1) = f(x_2)$;*
- *For any two points $b_1 < x_1 < x_2$, we have $f(x_1) > f(x_2)$.*

The definition of *unimodal* is just opposite to *multimodal* because there can be only one place where local area maximizing means global area maximizing. The function in Figure 2 is unimodal.

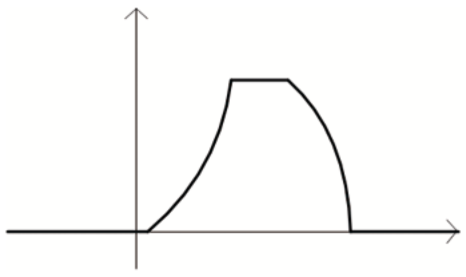


Figure 4: Example of an unimodal function

The importance of *unimodal* definition relies on efficient search algorithms can be generated for a unimodal function such as binary search or Fibonacci search. Thus we need to find the connection with the area overlapped. Now define the overlap area function $A(r)$ as:

Definition 2.6. *The overlap area function* $A(r) : R^2 \rightarrow R$ *is:* $A(r) = S(P \bigcap Q(r))$ *where* $S(A)$ *is the area of polygon* A *computed.*

Now we convert our problem into finding the best placement of $Q(r)$ which maximize $A(r)$. Since we only give the definition of *unimodal* in 2-dimensional case, it's not good to tackle with the area function because it concerns with both $r.x, r.y$ which will be 3-dimension. So we try to restrict the problem by define the polygon can only translate along some given direction. Technically, we restrict the y_i value and assume Q can only move along the line $y = y_i$. Now for the given y_i value, we can also define the restricted overlap area function $A_{y_i}(r)$ as:

Definition 2.7. *The restricted overlap area function* $A_{y_i}(r) : R \rightarrow R$ *is:* $A_{y_i}(r) = S(P \bigcap Q(r))$ *where* $S(A)$ *is the area of polygon* A *computed and* $r.y = y_i$.

In the next section, we will give the reason why restricted overlap area function is introduced to find the maximum area centrosymmetric polygon.

3 Finding the MACP in a Convex Polygon

In this section, we propose an algorithm finding the maximum area centrosymmetric polygon in a given convex polygon P theoretically.

```
Algorithm 1 Finding the MACP in a Convex Polygon
1: Get the data structure of the give convex polygon $P$;
2: Random choose one node in $P$ and assume it to be the *center*;
3: Compute the convex polygon $Q$ which is centrally symmetric about the *center*;
4: Find the reference point $r_Q$ of polygon $Q$;
5: Compute the List Reference-Y ($RefY$) with $P$ and $Q$;
6: Binary search the List $RefY$ maximizing $A_{y \in RefY}(r)$ to find the strip $(-\infty, +\infty) \times [y : y']$ contains the best placement;
7: Construct a 1/4-cutting and find the triangle contains the best placement recursively
```

Now we describe the algorithm in details. After we get the convex polygon P, it's easy to get the centrally symmetric polygon Q by randomly choosing a *center.* After the construction, we compute the Reference-Y List which consists of y_i-values that y_i is the r_Q's y-coordinate such that some vertex $v_p \in P$ and $v_q \in Q((x, y_i))$ have the same y-coordinate. Clearly there can be at most n^2 different elements in the list $RefY$ and suppose they are sorted as $y_1, y_2, \cdots, c_{n^2}$ where $y_i < y_{i+1}$. Take binary research on the list $RefY$ as follows: Initially suppose $k_{min} = 1$ and $k_{max} = n^2$, let $k = \lfloor (k_{min} + k_{max})/2 \rfloor$. Then compute $\max_{r=(x,y_k)} A_{y_k}(r)$ using the method in [17] (we will give a sketch in the next section) and $\max_{r=(x,y_{k+1})} A_{y_{k+1}}(r)$, do the operations above recursively if:

- $\max_{r=(x,y_k)} A_{y_k}(r) < \max_{r=(x,y_{k+1})} A_{y_{k+1}}(r)$, set $k_{min} = k$;
- $\max_{r=(x,y_k)} A_{y_k}(r) > \max_{r=(x,y_{k+1})} A_{y_{k+1}}(r)$, set $k_{max} = k + 1$;
- $\max_{r=(x,y_k)} A_{y_k}(r) = \max_{r=(x,y_{k+1})} A_{y_{k+1}}(r)$, set $k_{min} = k$ and $k_{max} = k + 1$, the strip $(-\infty, +\infty) \times [y : y']$ is found where $y = y_k$ and $y' = y_{k+1}$.

After finding the stripe containing the best placement, we generate a 1/4-cutting and find the candidate triangle recursively which is used in [16].

This algorithm is almost based on [16] such as the binary search and cutting, although its complexity can be analyzed $O(n \log n)$ theoretically, we don't take it into the experimental stage because the subroutine in computing the maximal area along a line $y = y_i$ in [17] is hard to implement which will use high-dimension expanding that is time consuming and another important reason is such binary search above is not always efficient compared with some other methods. In the experimental stage, we use a normal way to compute the maximum area that Q can intersect P which will be declared later. Since the algorithm above involves the idea of using *unimodal* property to design efficient algorithm which leads our implemented algorithm. We will continue the analysis about the correctness and complexity in the next section.

4 Correctness and Complexity

The algorithm is given in Section 4 to find the largest polygon that is centrally symmetric in a convex polygon. In this section we will prove the correctness of the algorithm and give the complexity theoretically which should be a necessary part among algorithm design process.

4.1 Correctness of the Algorithm

In order to keep the algorithm correct, there are two things to be satisfied:

- The polygon found by the algorithm is centrally symmetric at some point;
- The polygon found is the largest one that is centrosymmetric, which means there exists no other centrosymmetric polygon with larger area.

Before the proof of the two aspects listed above, we should give a claim that the largest centrally symmetric polygon is a convex polygon because the algorithm computes the overlap of two convex polygons and it's clearly that the overlap of two convex polygons should be also a convex polygon.

Claim 4.1. *The centrosymmetric polygon with the maximum area in a given convex polygon is a convex polygon.*

Now we give the reason. By contradiction, suppose the given convex polygon is P and suppose the largest centrally symmetric polygon is Q, thus: there exists two points $x, y \in Q$ and $\lambda \in (0, 1)$ such that $\lambda x + (1 - \lambda)$ is not in Q. Look at the figure below:

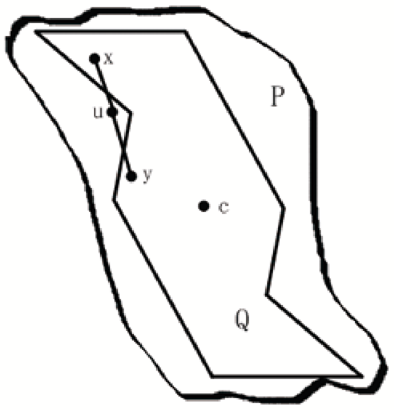


Figure 5: Example of Q is not convex polygon

In Figure 4.1, suppose the polygon found is Q which is centrally symmetric with the maximum area and the center c is not convex polygon. There are two points x and y in Q, but there is another point $u = \lambda x + (1-\lambda)y$ where $\lambda \in (0,1)$ suits $u \in P$ but u is not in Q. Because the center of Q is c, we can compute the corresponded point x', y' and u' easily. It's also easy to see that $u' = \lambda x' + (1-\lambda)y'$. From the definition of convex polygon, u' must belongs to P for the sake of P contains the two points x' and y'. Thus we know u and u' are both in P and they are symmetric at center c. In the same way, we can know the segment from x to y are the segment from x' to y' are in P and they are symmetric at center c. Thus we can find a larger polygon compared with Q, which leads to a contradiction. So we know the polygon we want to find is a convex polygon.

Now we return to the proof about the two points ensuring the correctness of the algorithm.

Lemma 4.1. *The polygon found in Algorithm 1 is centrosymmetric.*

From the algorithm we will compute the centrally symmetric polygon of P and we keep it as Q. The largest overlap of P and Q is kept as R. Now we need to prove R is centrosymmetric.

Because R is a convex polygon from Claim 4.1, if all the vertices or all the edges of R can be proved to be centrally symmetric, it's easy to know R is centrosymmetric. For any placement $Q(r)$, clearly there exist a center point $c(r)$ that P and $Q(r)$ are centrally symmetric at $c(r)$. Now consider the overlap of the two polygons. For any vertex v of R, there are three cases: v is a vertex of P, v is a vertex of $Q(r)$ and v is the intersection point of two edges $e_1 \in P$ and $e_2 \in Q(r)$. If v satisfies the first two cases, it's easy to see that corresponding vertex v' that is centrally symmetric at $c(r)$ is also in R since it belongs to the overlap of P and $Q(r)$. If v is the intersection point of two edges as above, we can first find the centrally symmetric edges $e_1' \in Q$ and $e_2' \in P$, since $e_1 \bigcap e_2 = v \in R$, the intersection point v' of e_1' and e_2' is also in R. Thus we know for every vertex of R, we can find another vertex on R such that they are centrally symmetric at point $c(r)$. So we know the convex polygon we found is centrally symmetric.

Lemma 4.2. *The polygon found in Algorithm is the largest maximum area centrosymmetric polygon.*

In order to prove the polygon we found is the largest one, there are two points meaningful:

- Algorithm 1 can find the largest overlap of two convex polygons when one is free to translate in the plane.

- For any centrosymmetric polygon R in P, we can construct another polygon Q which is centrally symmetric with P and $R \subset P \bigcap Q$.

Clearly if the two side above are both satisfied, the largest centrosymmetric polygon in a convex polygon can be found by Algorithm 1. Now we proof the two side. Firstly, give a constructive proof about the second one.

Lemma 4.3. *For any centrosymmetric polygon R in P, there exist a centrally symmetric polygon Q with P such that $R \subset P \bigcap Q$.*

For any given centrosymmetric polygon R in P, suppose the center is c, construct the symmetric polygon Q by computing the centrally symmetric vertices of P, then connect them to be Q. Look at the figure below:

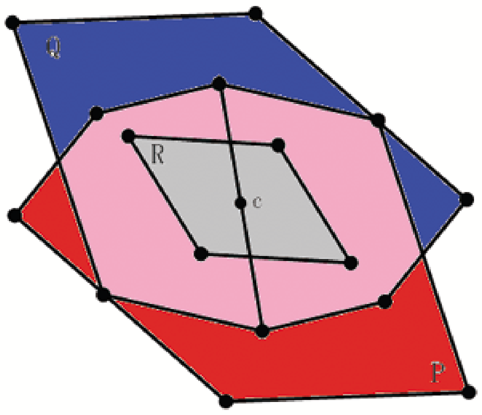


Figure 6: Construct Q on the given P and R

The way to construct the polygon Q is given above and we need to prove $R \subset P \bigcap Q$. For any point $v \in R$, it's easy to see $v \in R \subset P$, keep the symmetric point about c is v', we can have: $v' \in Q$. Since R is centrosymmetric at center c, $v' \in R \subset P$, which means $v' \in P \bigcap Q$. Then we can conclude $v \in P \bigcap Q$. So $R \subset P \bigcap Q$.

Now we focus on the point that Algorithm can find the largest overlap of convex polygon P and Q. First we will introduce some important properties.

Property 4.1. *Let P and Q be the two polygons in Algorithm 1, for any fixed $y_i \in R$, the restricted overlap area function $A_{y_i}(r)$ is unimodal.*

This property is very important because we can use it to compute the maximum overlap of P and Q when Q is confined to translate along some fixed line $y = y_i$, thus binary search will be useful to find the strip $(-\infty, +\infty) \times [y_k, y_{k+1}]$ that contains the best placement $Q(r)$ which resemble P mostly.

This property is a corollary from [17] in which the sectional area of convex polytopes is proved to be unimodal. Considering the 2-dimension case, we can also generate the property that $A_{y_i}(r)$ is unimodal. More visually, image Q move from the place that it's entirely left to P to the place that it's entirely right to P. During the process, the maximum area for each vertical line behaves like: First being zero for some time, then it increases strictly until there is such a y_k that it reaches the maximum area, it may also stay constant at the maximum are for some time until it starts to decrease strictly, eventually it becomes zero when it entirely passes the fixed polygon.

Property 4.2. *Let P and Q be the two polygons in Algorithm 1, suppose r_Q is defined to translate along a line l, the restricted overlap area function $A_l(r)$ is unimodal.*

This property is a more general one compare with Property 4.1 because we can let the line l be $y = y_i$ which makes sense. This property contains more meaning: If we move the polygon Q along any line l in the plane, the overlap area function is unimodal. This can be also verified from [17]. Now we will give another important fact that using the binary research above, we can find such a strip $(-\infty, +\infty) \times [y_k, y_{k+1}]$ contains the best placement.

Lemma 4.4. *For any two lines $l_1 : y = y_k$ and $l_2 : y = y_{k+1}$, let v_1 and v_2 be the points on the two lines such that $S_{y_k}(v_1) = \max_{r \in l_1} S(P \bigcap Q(r))$ and $S_{y_{k+1}}(v_2) = \max_{r \in l_2} S(P \bigcap Q(r))$. Without loss of generality, suppose $S_{y_k}(v_1) \geq S_{y_{k+1}}(v_2)$, then the open half-plane defined by $l_2 : y = y_{k+1}$ containing point v_1 contains the best placement.*

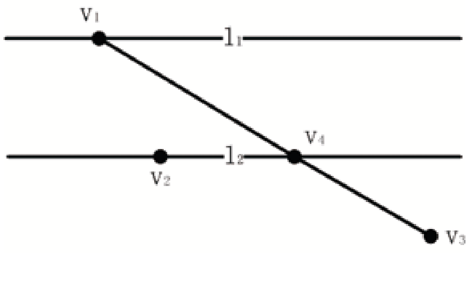


Figure 7: Half space bound of the best placement

Look at Figure 4.1, we want to prove the open half-plane defined by l_2 containing v_1 contains the best placement. By contradiction, for any point v_3 such that when Q is placed to v_3 the area is larger, i.e. $S(P \bigcap Q(v_3)) > S(P \bigcap Q(v_1))$. Connect point v_3 with v_1 and it will intersect l_2 at some point v_4. From the Property 4.2 the overlap area function along line l_{v_1,v_3} should be unimodal, because

$S(P \bigcap Q(v_3)) > S(P \bigcap Q(v_1))$, we know $S(P \bigcap Q(v_4)) > S(P \bigcap Q(v_1))$ from the definition of *unimodal*. Thus $S(P \bigcap Q(v_4)) > S(P \bigcap Q(v_1)) \geq S(P \bigcap Q(v_1))$. Since v_2 and v_4 are on the line l_2 and v_2 is the point that maximize the overlap area when Q is translated along the line, which makes an contradiction. So we can restrict the best placement to the half-plane containing v_1.

Lemma 4.5. *The binary search in Algorithm 1 can find the strip* $(-\infty, +\infty) \times [y_k, y_{k+1}]$ *such that the best placement is in the strip.*

Recall the binary research on the *RefY* list, each time compute the two value y_k and y_{k+1}. Then we need to compute the maximum overlap area along the line, if $\max_{r=(x,y_k)} A_{y_k}(r) < \max_{r=(x,y_{k+1})} A_{y_{k+1}}(r)$, we know the half plane $(-\infty, +\infty) \times [y_k, y_{max}]$ which means we can let $k_{min} = k$ and continue the binary search. The same analysis for the other two cases. Finally we can find such strip contains the best placement.

Return to the proof of Lemma 4.2: From Lemma 4.5 Algorithm 1 can find the stripe containing the best placement for any two convex polygon P and Q, now perform a 1/4-cutting and find the triangles containing the best placement recursively. Here a 1/4-cutting means divide the plane into some disjoint triangles that cover the entire plane, each triangle will intersect the segments no more than $k/4$ and the number of all segments is k. The method's correctness can be found in [16]. Generally speaking, the way to find the placement in the fixed strip can be operated as another search process will may be efficient as well. Thus the algorithm finds the largest overlap of P and Q.

Theorem 4.1. *Algorithm 1 finds the maximum are centrosymmetric polygon.*

From Lemma 4.3 and Lemma 4.2 we know that the method of finding the maximum area centrosymmetric polygon is correct and Algorithm 1 can output the largest polygon in P that is centrally symmetric. $\square$

4.2 Complexity Analysis

In this section, we will give the complexity of Algorithm 1 in a theoretical way. Before the analysis, we have the following lemmas.

Lemma 4.6. *For each line* $l : y = y_i$*, we can compute the maximum restricted overlap area* $\max_{r \in l} A_{y_i}(r)$ *in* $O(n)$ *time where the convex polygon has* n *vertices.*

In [17] we learn that for a convex polyhedron K with n vertices, the maximum area cross-section orthogonal to a given direction can be compute in $O(n)$ time using the algorithm in [12]. Thus we can add another coordinate to the xy-plane which means extending the convex polygon P to a right cylinder in direction $(0, 0, 1)$. At the same time, extend the polygon Q to a slanted cylinder in the direction $(1, 0, 1)$ for the line $y = y_i$. Thus we can compute the intersection of the two cylinders which can be seemed as the polyhedron K. Then compute the section area that is using the xy-plane to intersect the polyhedron K to find the maximum overlap. This can be computed in $O(n)$ time from [12]. Though we can achieve the complexity, the algorithm is too complex and not practical because in involves dimension expanding. Thus we may use some other ways in our experiment stage.

Lemma 4.7. *The binary search that finds the strip* $(-\infty, +\infty) \times [y_k, y_{k+1}]$ *containing the best placement takes time* $O(n \log n)$.

Before the binary search, the ordinary idea is computing the Reference-Y list directly where $O(n^2)$ time is necessary. However, all the vertices of P can be sorted by y coordinate as $\{p_1, p_2, \cdots, p_n\}$ in $O(n \log n)$ time and the same for Q as $\{q_1, q_2, \cdots, q_n\}$. The *RefY* list is the matrix as $M = (c_{ij})$ where $c_{ij} = a_i - b_j$. Frederickson et al found an algorithm in 1984 which can compute the k-th value of the n^2 elements in $O(n)$ time when the matrix M is a sorted one (here M is sorted means the elements in each column and each line are sorted), thus we can use $O(n)$ time to find the k-th value y_k and use another $O(n)$ time to compute $\max_{r \in l: y=y_k} A_{y_k}(r)$ from Lemma 4.6, also $O(n)$ time is needed to compute $\max_{r \in l: y=y_{k+1}} A_{y_{k+1}}(r)$. Then continue the binary search process until we find the strip contains the best placement. There are $O(\log(n^2)) = O(\log n)$ search loops and each search can be finished in $O(n)$ time, thus $O(n \log n)$ time is enough to find such strip with the preprocess of sorting time which can be also bounded by $O(n \log n)$.

Theorem 4.2. *Algorithm 1 can find the maximum area centrosymmetric polygon in* $O(n \log n)$ *time.*

Now we give the complexity step by step. When n vertices convex polygon P is given, we can choose any vertex as the "center" and compute the centrally symmetric polygon Q. This step can be done in $O(n)$ time clearly. Then we can compute the strip $(-\infty, +\infty) \times [y_k, y_{k+1}]$ in $O(n \log n)$ time from Lemma 4.7. The last step is 1/4-cutting and triangle search recursively which can be also finished in $O(n \log n)$ time from [17] (this part is not so clear in the paper and we choose some other ways in our implementation). Sum up all the time complexity, we conclude the theorem. $\square$

5 Experiment

In this section, we will introduce the details of our project that finds the maximum area centrosymmentric polygon in a convex polygon. This section contains the following parts: First we will introduce the algorithm we implement which is not exactly the same as Algorithm 1 in section 3, then give an overview about the design of our experiment, the following part gives the details of the operations permitted and we will give some simulation results. Finally, we conclude the experience our team gain from the project with some problems we come across during the process and how we solve these problems.

5.1 Experimental Algorithm

Although Algorithm 1 in section 3 can efficient find the largest centrosymmetric polygon in a convex polygon theoretically, there are many difficulties in implementation. The hardest two stage is:

- For the given line $l : y = y_k$, suppose Q is free to translate along the line, we can find the maximum overlap area $\max_{v \in l} S(P \bigcap Q(v))$ in $O(n)$ time.
- When we find the strip $(-\infty, +\infty) \times [y_k, y_{k+1}]$ containing the best placement, a 1/4-cutting is involved to search the triangle containing the best placement recursively.

The first one is very hard to implement because the algorithm employed is expanding the problem into high-dimension case that computes the intersection of two convex polytopes and find the largest sectional area of the intersection polytope corresponding to the largest overlap when one polygon is free to translate along some fixed line. The way to find the intersection of two high dimension convex polytopes is very hard, let alone computing the largest sectional area of the polytope. So we give up the method to compute it. The second stage is also very hard in practical implementation because the construction of 1/4-cutting is difficult and the recursively search of all the triangles increase the hardness. Under the circumstances, we return to the idea of finding the maximum area centrosymmetric polygon which involves the *unimodal* definition, some easy implementing way can be found which can also achieve the aim of the above two parts. Now we give the experimental algorithm below:

Algorithm 2 Experiment Algorithm: Finding the MACP in a Convex Polygon

```
1: Construct the convex polygon $P$ based on the points given;
2: Compute the centrally symmetric polygon $Q$ to $P$ at some fixed center point;
3: Find the reference point $r_Q$ of $Q$;
4: Compute all the $y$-coordinate values of $r_Q$ when $P$ and $Q$ intersect at some
   vertex ($RefY$ List);
5: Compute all the $x$-coordinate values of $r_Q$ when $P$ and $Q$ intersect at some
   vertex ($RefX$ List);
6: Binary search the List $RefY$ maximizing $A_{y \in RefY}(r)$ (using Algorithm 3
   below) to find the strip $(-\infty, +\infty) \times [y, y']$ contains the best placement;
7: Using optimum seeking method on $[y, y']$ to find the best placement;
```

In the above algorithm, we may find some difference compared with Algorithm 1. The first one the input of the algorithm is many points on the plane instead of the given convex polygon P. This difference is legal because when we design the project, we know that users may give an arbitrary polygon in the plane and it may disturb the users how to input a convex polygon, so we change the input such that the users can point many points on the plane, with no worry whether they consist a convex polygon, then our program computes the convex using the algorithm we learned in class. The second difference we also compute the Reference-X List($RefX$) because in our experiment we will compute the largest overlap when Q is restricted to translate along some line. The third one is we bring in Algorithm 3 below to compute the maximum area of $P \bigcap Q(r)$ when r is restricted to translate along the line $y = y_i$, the algorithm will be given later. The last difference is we don't need 1/4-cutting as Algorithm 1 after we find the strip containing the best placement, in the other way, we use the optimum seeking method instead of recursively searching the triangles cover the plane. Algorithm 3 is like:

Algorithm 3 Finding Maximum area of $P \bigcap Q(r)$ when r is along line $y = y'$

```
 1: Compute the rough lower $x_{lower}$ such that $S(P \bigcap Q(x_{lower}, y')) > 0$;
 2: Compute the rough upper $x_{upper}$ such that $S(P \bigcap Q(x_{upper}, y')) > 0$;
 3: Double goldenRatio = $\frac{\sqrt{5}-1}{2}$;
 4: $Dis = x_{upper} - x_{lower}$;
 5: $x_1 = x_{lower} + Dis * goldenRatio$, $x_2 = x_{lower} + *(1 - goldenRatio)$;
 6: while Not finding the maximum area do
 7:     Compute the area $S_1 = S(P \bigcap Q(x_1, y'))$ and $S_2 = S(P \bigcap Q(x_2, y'))$;
 8:     if $S_1 < S_2$ then
 9:         $x_{upper} = x_1$; $x_1 = x_2$; $x_2 = x_{lower} + (x_{upper} - x_{lower}) * (1 - goldenRatio)$;
10:     end if
11:     if $S_1 > S_2$ then
12:         $x_{lower} = x_2$; $x_2 = x_1$; $x_1 = x_{lower} + (x_{upper} - x_{lower}) * goldenRatio$;
13:     end if
14:     if $S_1 = S_2$ then
15:         $x_{lower} = x_2$, $x_{upper} = x_1$, $Dis = x_{upper} - x_{lower}$;
16:         $x_1 = x_{lower} + Dis * goldenRatio$, $x_2 = x_{lower} + Dis * (1 - goldenRatio)$;
17:     end if
18: end while
```

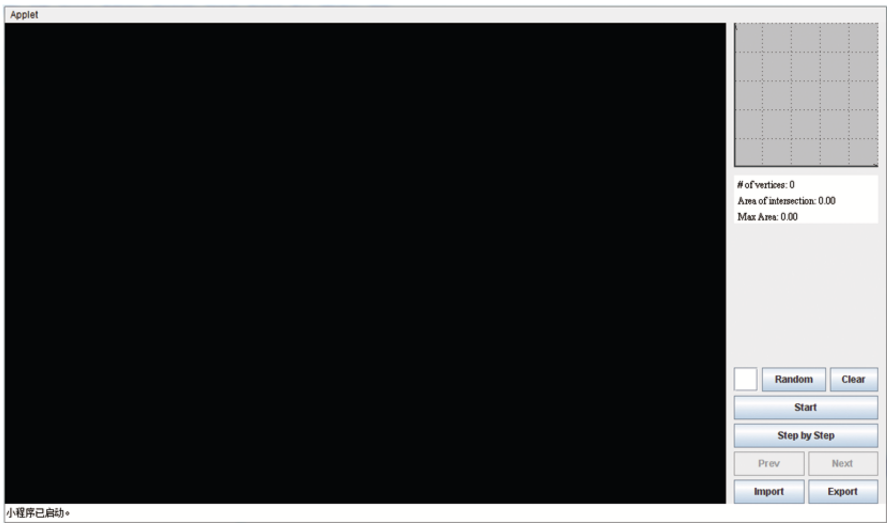
Algorithm 3 uses the optimum seeking method to find the largest area when the rough lower value and upper value is found. The two values can be found by some easy search way since we only want to get the x_i value such that the area corresponded is larger than 0. Algorithm 3 can correctly finds the maximum area because the function $A_{y'}(r)$ is unimodal and we can find the largest area efficiently by optimum seeking method. Since the optimum method is a very classic searching method for unimodal function, we leave out some details in Algorithm 3.

This is an important part to bring in the new algorithm which is easy to implement and more practical to find the largest area of two convex polygons when Q is restricted to translate along some line. This algorithm may be worse than the algorithm in [17] from the theoretical complexity way, however, this one is easy to implement and we take fully use of the *unimodal* property in the research and the optimum seeking method also performs well for its high efficiency. So we can continue to find the strip $(-\infty, +\infty) \times [y, y']$ containing the best placement. Then we use the same way for the section $[y, y']$ to seek the best placement based on the unimodal property, which is also very efficient in practical project.

From the analysis above, the correctness can be hold using the experimental algorithm and the efficiency may be not so good as $O(n \log n)$ time, however the optimum seeking method can also be efficient in practical. So we claim this project can be continued based on the method.

5.2 Experimental Design

In order to put the project on the web for more users, we choose Java to write the project as an applet and the appearance is designed as below:



# of vertices: 0
Area of intersection: 0.00
Max Area: 0.00

小程序已启动。

Figure 8: The appearance of project design

On the left side of the design, it's a 900×600 panel where users can input the nodes, the algorithm can compute the convex polygon from the input, finding the maximum area centrosymmetric polygon and some other operations that will be introduced in the following part. The right side of the design a 200×600 toolbar which is made up of two parts. The upper part is the output and display region where we can see the area of two overlap polygons visually with both result output and the diagram. Moreover, the diagram will be useful in verifying the *unimodal* property when one polygon is move along some fixed line with the restricted overlap area curve. The lower part is the function region which consists of many buttons and a text field for input information.

The design above is simple and clear with on more useless region. The size of the appearance is appropriate for putting on a web site and the user needn't page down to check for the button or to find the output result. Now we will tell more about the details in our experimental design. Since the project contains both appearance and algorithm implementation, the interfaces between them should be very important. Listed below are the interfaces:

- *public static LinkedList <Node> getConvexHull(LinkedList <Node> nodes)*: This interface gives the result from the input nodes which is kept in a linked list, the return value is the linked list representation of the convex polygon found;

- *public static LinkedList <Node> getCentroSymmetry(LinkedList <Node> convexHull)*: This interface aims at computing the centrally symmetric polygon of the given convex polygon, the input is a convex polygon represented by a linked list of nodes and the output is also a linked list of nodes;

- *public static void getRefenceVal(LinkedList <Node> P, LinkedList <Node> Q, ArrayList <Integer> refX, ArrayList <Integer> refY)*: This interface compute the reference-X and reference-Y list that are useful in the implemented algorithm. The input are the two convex polygons and the two lists for keeping the reference values. It has no return value with changing the $refX$ and $refY$ lists inside the implementation;

- *public static double computeIntersect(LinkedList <Node> P, LinkedList <Node> Q, Integer xbias, Integer ybias)*: This interface is important because it will return the overlap area for the two convex polygons when Q is allowed to move $(xbias, ybias)$ increment, where means the reference point r_Q is moved to $(r_Q.x + xbias, r_Q.y + ybias)$. Easy to see the input of the interface is a given fixed convex polygon P and a free to translate polygon Q with the increment $(xbias, ybias)$, the return value is the overlap area;

- *public static RtnVal computeMaxIntersect (LinkedList <Node> P, LinkedList <Node> Q)*: This interface returns the information of the maximum overlap area. The class $RtnVal$ includes $(xbias, ybias, area)$ where

$(xbias, ybias)$ is the increment of move and $area$ is the computed maximum overlap area. The input of the interface is two convex polygons and the latter one is assumed to move freely in the plane, the return class contains the information the increment to move and the largest are;

- *public static RtnVal computeMaxIntersectForY(LinkedList <Node> P, LinkedList <Node> Q, int ybias)*: This interface is almost like the above one with the difference that it computes the largest area when Q is restricted to move along a fixed line. The input of the interface is two convex polygons and a increment of y-coordinate which means r_Q is allowed to move along the line $y = r_Q.y + ybias$. The return value is the information where we can find the largest overlap for the restricted movement and the largest value of the overlap.

Based on the six interfaces above, we can design the project in a more structural way. In the next section, we will give all the operations supported in our program.

5.3 Experimental Operations

This part contains the operations and functions related in our design. Let's take a closer look at them:

Input: There are three ways for input: randomly generating some points, clicking on the black panel and import some points by input the position. Following graph is an example of generating 30 nodes randomly:

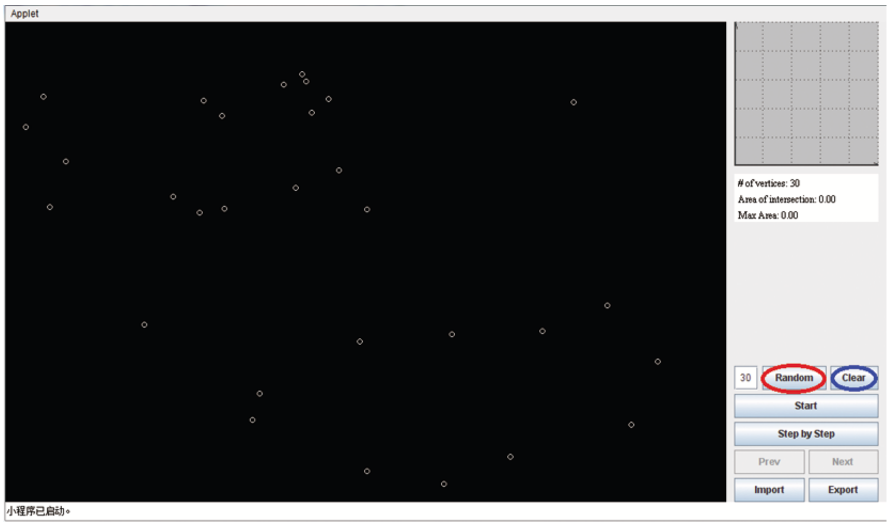
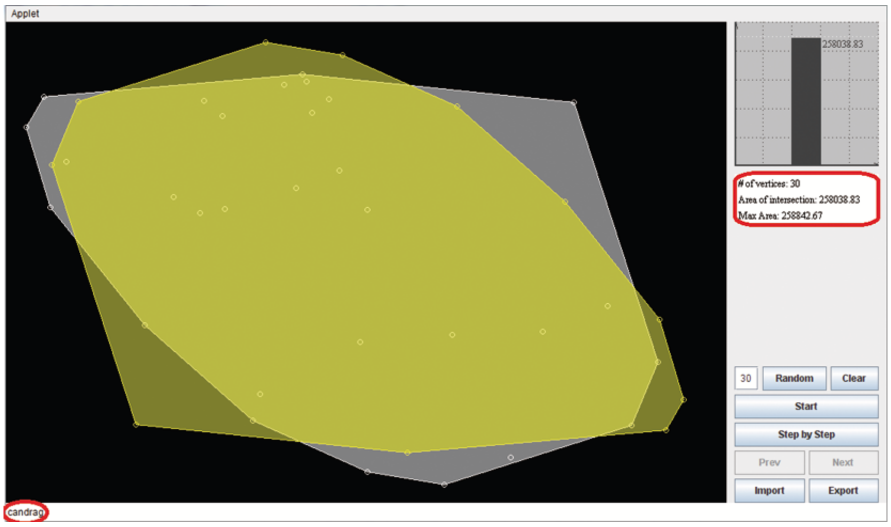

# of vertices: 30
Area of intersection: 0.00
Max Area: 0.00

Figure 9: Generate 30 nodes randomly

After randomly generating, we can also click on the black panel to add more points. The output region will give the information about the number of points in the panel, duplicated points will be counted only once with prompt message. When you find the input has no meaning or you want to start a new test, the button in blue cycle "clear" will help you clean all the nodes in the panel.

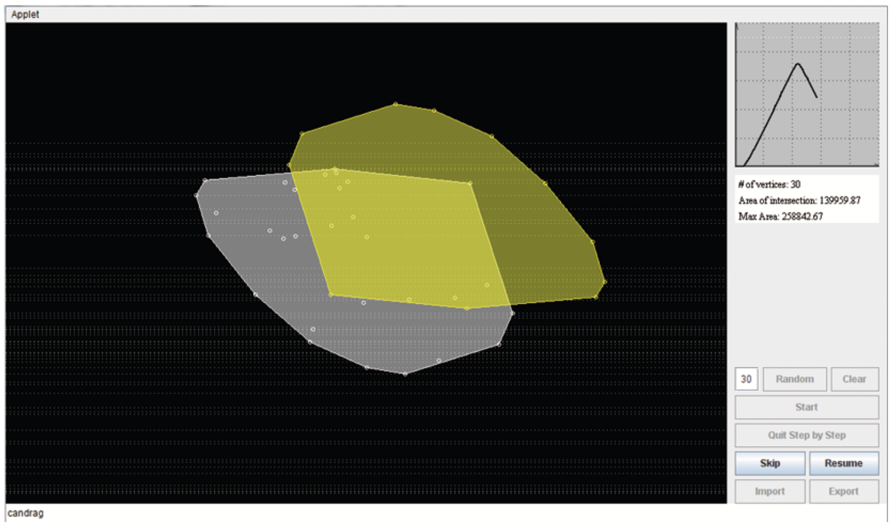
Algorithm Start: Click the "start" button will compute the largest overlap area quickly and the output region will give the maximum value with a histogram. This one is useful because we also supported another operation: **Drag:** Users can drag the polygon Q consisted of yellow lines to intersect the fixed convex polygon to see the overlap area, the output region will give the current area compared with the maximum area at the same time. Following is the example:



```
# of vertices: 30
Area of intersection: 258038.83
Max Area: 258842.67
```

Figure 10: Example of finding the maximum area with Drag operation

Step by Step: This operation gives how we begin our algorithm, the difference is we don't give all the steps in our implemented algorithm, instead we do the following things: Compute the Convex polygon P first; Find out the centrally symmetric Q; Draw all the $refY$ lines in the panel; Scan all the lines when Q is entirely above P until it's entirely below P, when scanning each line, we draw the curve of overlap area function when Q is move along the line from left to right in the diagram region. We can also pause at every time to see the correctness of *unimodal* property. Following is an example:



```
# of vertices: 30
Area of intersection: 139959.87
Max Area: 258842.67
```

Figure 11: Step by step operation

5.4 Experimental Data and Result

In order to make a project robust and meaningful, test on different types of data is very important. In this part, we will talk about the way we deal with all the cases.

- Normal data: Our program runs efficiently on the normal data based on the input from users;
- Small number of nodes: When the number of nodes is small $(0, 1, 2)$, we will give the message to input more nodes;
- Duplicated click: When a point is click many times, we won't add the number of points, while a message telling the user it's a duplicated click.
- Points on a line. Generally speaking, there is no polygon when all points are on a line, in order to make the algorithm robust, we also run the case with output 0 which is meaningless. The user can click more points to restart the algorithm and find the result.
- Large number of input nodes. We have also test these cases when many nodes are just around a circle such as 100 points, the algorithm can also give the correct answer quickly.

5.5 Experimental Experience

During the process of the project, we achieve many experiences and overcome some problems we came across in the implementation. Through the way in solving these problems, we gained so much not only the way of solving problems, but also the importance of teamwork. Firstly, we will list some significant problems in our implementation:

The first vital one is how to compute the maximum overlap area when the polygon Q is restricted to move along a given line $y = y_i$. As we describe in section 3, [17] proposed an algorithm which involved dimension expanding and it's not practical to implement. Under the special circumstance, we have to think of another algorithm for implementation. In the beginning, we came up with binary search which is used on $refY$ list and this one seems correct in our first edition. However, we test some special data: only three points in the panel and their x-coordinate are almost the same, the result we generated is 0! This urgency makes us crazy but we found the reason very soon. The binary search about the $refY$ list is not the ordinary one, if the x region suits $S(P \bigcap Q(x, y)) > 0$ is very narrow, the search will terminate and output the area 0. Thus we came across a new algorithm which involves optimum seeking method after finding the rough bound the area is larger than 0 (the details are omitted here). This problem makes sense why special data test should be considered when designing a project. Another important problem is how to read file for input? This problem seems easy because we can use many *io* operations. However, this project is designed as an applet which will be put on the web and Java is very cautious about operating local files for security reasons. This problem can be also solved by some special techniques by enquiring the security certificate. During the process to solve problems, we gained many experiences not only in coding, but also how to think of the project and how to make the project strong and robust.

Besides the problems we came across and solved, there are some other important points we learned from the project. The first one is theory is not always reality. Many algorithms may perform well in theoretical ways but they are very difficult to implement and may be not so practical. However, the trade-off between theoretical and practical should be concerned cautiously. What's more, teamwork is also very significant in the experiment. Before the project, our team has discussed several times about the topic we chose and we made clear division. During the process, we gained the precious friendship as well as the good result of the project. In a word, we think highly of this project and cherish all the experiences from the experiment.

6 Conclusion and Discussion

In this report, we choose the topic: finding the maximum area centrosymmetric polygon (MACP) in a convex polygon as our project research. This problem is one instance of the important problem of computational geometric area: how to find the largest figure with some property contained in a given figure, which has been attracting the scientists' attention since its first appearance. In this report, we propose the method: construct a centrally symmetric convex polygon first, and compute the largest overlap where the latter polygon can move around the plane to solve the problem. We give the algorithms along the method in finding the largest centrally symmetric polygon from both theoretical and practical ways. The theoretical algorithm can find such polygon in $O(n \log n)$ time and the correctness is also proved. However, it's not practical in implementation. So we come up with another experimental algorithm which is easier and more understandable in the experiment. Luckily, the algorithm implemented can also find the largest centrally symmetric polygon and the efficiency is also acceptable. In the experimental stage, we give a good design: for any input by randomly generated or points from users, we can find the largest centrally symmetric polygon quickly and we can make users more clear about why the algorithm can do such a search by showing the unimodal function curve recursively. What's more, users can drag the centrally symmetric convex polygon of the given one to see the area of their overlap. This problem is solved in such a program successfully with good appearance design and user experience.

During the research process, we find several problems we can continue keep attention and further research. In our experimental stage, we use the optimum seeking method both on the two dimension finally, thus if it's possible to search on the xy plane of the two coordinates at the same time? From the unimodality of the special function, can we give another similar property on the three coordinates: x, y and the area $S(x, y)$? Intuitively speaking, the overlap area function $S(x, y)$ will be also unimodal (here we should redefine unimodal in another way such that it's suitable for the function with two variables) which may produce some other quicker search method such as Newton downhill method or Newton uphill method. Another related problem may draw our attention: if the given polygon is not restricted to convex, can we also find the largest centrally symmetric polygon in the similar way? This problem seems not so easy to give a determined answer. These problems discussed above are very important in both theoretical and practical ways and they play an significant role in understanding the essence of these problems in computational geometric.

7 Acknowledgement

This work has been supported with efforts from many other people. Thanks Kai Jin for mentioning the topic to choose, and he also gave us many constructive opinions about the problem. Thanks Guanyu Wang and Jinbiao Chen for some good ideas about how to design the project better and how to make the problem easier to implement. Thanks all the projects before for their excellent analysis and beautiful design. The most important person we want to acknowledge is Junhui Deng, the teacher of this course, not only for his comment about finding more applications for the problem, but also the wonderful lecture and the important knowledge we learned from him.

From this project we learned many lessons we won't come across such as how to choose an appropriate topic, how to select high related paper for further reading, how to cohere with other members in the team and how to make progress step by step. This course not only gives us some insightful sense to the problems of Computational Geometry area, but we can also learn some essential principles from the method or the algorithm. Whatever our major are, all the knowledge will be useful in our future research.

References

[1] D.P. Dobkin and L. Snyder. On a general method for maximizing and minimizing among certain geometric problems. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, pages 9-17, 1979.

[2] James E. Boyce, Dvaid P. Dobkin, Robert L. Drysdale, III and Leo J. Guibas. Finding extremal polygons. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, STOC'82, pages 282-289, 1982.

[3] G. N. Frederickson and D. B. Johnson. Generalize selection and ranking: sorted matrices. *SIAM J. Comput.*, 13:14-30, 1984.

[4] J. Chang and C. Yap. A polynomial solution for the potato-peeling problem. *Discrete and Computaton Geometry*, 1:155-182, 1986.

[5] Joseph O'Rourke N. Adlai DePano, Yan Ke. Finding largest inscribed equilateral triangles and squares. In *Proc. Allerton Conf.*, pages 869-878, 1987.

[6] H. Alt, B. Behrends, and J. Blomer. Approximate matching of polygonal shapes. In *proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 186-193, 1991.

[7] J. C. Latombe. Robot Motion Planning. Kluwer Academic Publisher, Boston, 1991.

[8] D. Parson and C. Torras. The combinatorics of overlapping convex polygons in contact. In Proc. 4th Canad. Conf. Comput. Geom., pages 83-92, 1992.

[9] Pankaj K. Agarwal, Micha Sharir, and Sivan Toledo. Applications of parametric searching in geometric optimization. In *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, pages 72-82, 1992.

[10] D. P. Huttenlocher, K. Kedem, and J. M. Kleinberg. On dynamic Voronoi diagrams and the minimum Hausdorff distance for point sets under Euclidean motion in the plane. In Proc. 8th Annu. ACM Sympos. Comput. Geom., pages 110-120, 1992.

[11] H. Alt and M. Godau. Measuring the resemblance of polygonal curves. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 102-109, 1992.

[12] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comp.*, 21(4):671-696, 1992.

[13] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9(2):145-158, 1993.

[14] L. P. Chew, M. T. Goodrich, D. P. Huttenlocher, K. Kedem, J. M. Kleinberg, and D. Kravets. Geometric pattern matching under Euclidean motion. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 151-156, Waterloo, Canada, 1993.

[15] D. P. huttenlocher, K. Kedem, and M. Sharir. The upper envelope of Voronoi surfaces and its applications. *Discrete Comput. Geom.*, 9:267-291, 1993.

[16] D. de Berg, O. Devillers, M. van Kreveld, O. Schwarzkopf, and M. Teillaud. Computing the maximum overlap of two convex polygons under translations. Manuscript, 1995.

[17] D. Avis, P. Bose, T. Shermer, J. Snoeyink, G. Toussaint, and B. Zhu. On the sectional area of convex polytopes. In *Communication at the 12th Annu ACN Sympos. Comput. Geom.*, 1996.

[18] Milenkovic Daniels and Roth. Finding the largest area axis-parallel rectangle in polygon. *CGTA: Computational Geometry: Theory and Applications*, 1997.

[19] Kai Jin. Finding the Maximum Area Paralleogram in a Convex Polygon. *The 23rd Canadian Conference on Computational Geometry (CCCG'11),* 2011.