***The Art Gallery Problem***

**Research Report**

**CS 253 – Data & File Structures**

**Section: 70**

**Central Connecticut State University**

Instructor: Neli Zlatareva

Date: xxx

Prepared By: xxx

**Table of Contents**

[Introduction to The Art Gallery Problem 2](#_Toc438017174)

[Naïve Algorithm and 3-Coloring 4](#_Toc438017175)

[Monotone Decomposition 10](#_Toc438017176)

[Delaunay Triangulation 13](#_Toc438017177)

[Conclusion 15](#_Toc438017178)

[Sources 17](#_Toc438017179)

Introduction

The Art Gallery Problem dates back to 1973, when it was first proposed by American mathematician Victor Klee (Rourke 1). The proposition was: “how many guards are needed to observe an art gallery, such that the entire gallery is monitored between those guards?” Why choose an art gallery, and not a post office, or a shopping mall? Well, the problem can technically be applied to those situations as well, however art galleries tend to have unique two-dimensional floor layouts that can often present challenges to the often simple placement of security guards or cameras. Two years later, Chvatal responded with the first “Art Gallery Theorem”, proving that “ guards are occasionally necessary and always sufficient to cover a polygon of *n* vertices” (Rourke 9). That, is indeed the essence of The Art Gallery Problem – any solution should provide *at least* an approximation of an upper bound on the maximum number of guards necessary to “see” the entire art gallery floor. More formally, for any polygon *P*, define *G(P)* to be the minimum number of vertices *k*, of *P* that cover all of *P*, where coverage is defined by “line of sight” that is only blocked by intersection with boundaries of *P*. The number of vertices *k*, must be a minimum such that there is a set of *k* points in *P*, {*x1, x2,…, xk*}, so that, for any *y* *P*, some *xi*, 1 ≤ *i* ≤ *k*, covers *y*. Lastly, *g(n)*, our solution, is defined to be the maximum value of *G(P)* over all polygons of *n* vertices (Rourke 2).

The Art Gallery Problem now has several variations (if not many), but this paper focuses on the problem as it was initially proposed by Klee: guards can only be placed at vertices and guards can survey 360o from their fixed location. For those interested in combinatorics, solving the problem as it is defined above is the same as solving the dominating set problem on the visibility graph of the polygon, although discussion of that ends there. One more very important restriction to consider is the fact that the simple polygons that are being triangulated are assumed to have no “holes” in the interior. That is, there are no existing walls other than the boundary of the polygon.

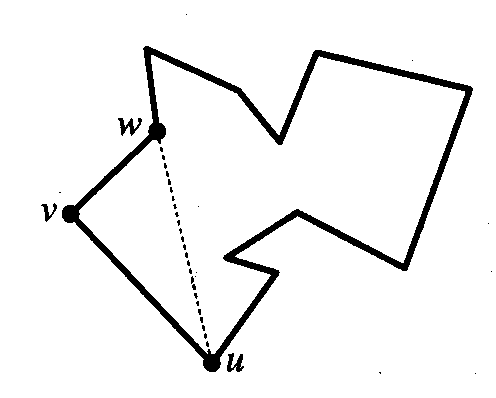
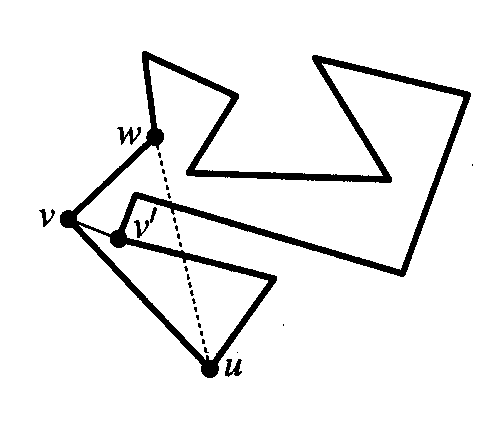
In this paper, several solutions to The Art Gallery Problem are introduced, in order of simplicity. First, it is shown that any simple polygon that can be mapped to the boundaries of the floor of an art gallery, can be represented using an undirected graph. Next, a discussion of polygon partitioning, namely triangulation, is initiated to show that no efficient solution can be found for a non-convex polygon without first dividing up that polygon into smaller pieces. Indeed, “obtaining an optimal algorithm for triangulation is perhaps the outstanding open problem in computational geometry” (Rourke 14). Thus, triangulation can formally be described as decomposition of a polygon into triangles by a maximal set of non-intersecting diagonals (Vlasic 4). A very simple, albeit inefficient, approach to triangulation is shown with, appropriately named, “The Naïve Method”, along with its complement, the “3-Coloring” algorithm. The next chapter addresses the exploitation of convex partitioning, using a technique known as Monotone Decomposition, where an increase in algorithm efficiency can be found by dividing the initial polygon into y-monotone components, and then triangulating those instead. To wrap up, this paper touches briefly on a far more advanced method of triangulation, Delaunay Triangulation. There are far more methods available than those addressed here, but they are beyond the scope of this introductory article on The Art Gallery Problem.

So basically, having the restriction that guards can only be placed on vertices, but they can see 360o from their fixed location (but not through walls!), what is the *fewest* number of guards we can employ, and exactly *where* should we put them?

Naïve Algorithm and 3-Coloring

Released only three years after Chvatal’s Theorem, Fisk’s Proof was established, simplifying Chvatal’s theorem by formally explicating the steps and dividing them into two discrete categories: 1) triangulation and 2) 3-coloring of the resulting triangulation graph (Rourke 4). As it may not be obvious at first, there are several properties about triangulations that are important to note in order to get a better understanding of how these algorithms work. First of all, triangulations are not unique, however every polygon has *at least* one triangulation. Also, any triangulation of a polygon has exactly *numberOfNodes*-2 triangles (Vlasic 5). Using Chvatal’s proof by induction to perform a *naïve* triangulation of the polygon, we have an algorithm that is quadratic at best: *O(n)* searches for a special diagonal *d* ends up costing *O(n2)* time (Rourke 10).

The proof by induction is as follows: for a simple triangle (base case: *n=3*), the proof is trivial – it is already triangulated, and there is *n*-2 = 1 triangles. The inductive step (for *n*>3) is a bit easier to visualize with a picture, as there are two cases that one might come across during naïve triangulation:

***Case 1:***

***Case 2:***

Case 1 above shows the simplest case where, starting at *v*, adjacent vertex *u* is connected with adjacent vertex *w*, and the rest of the polygon is triangulated recursively. If the diagonal *uw* exits the polygon at any point (as it does in case 2), the *farthest* point from that imaginary diagonal contained within triangle uvw is connected with starting vertex *v*. The next node is examined recursively in this fashion, until all nodes have been visited. Thus, in either of the two possible cases, a triangulation can be completed.

So, assuming the algorithm starts at node *v* in the above example, finding a diagonal from *any* node takes *O(n)* time, because each node in the graph may have to be examined in the event of case 2, so that the node *farthest* from diagonal *uw* can be located. Traversing the graph to find this farthest node *v’*, must be done recursively at each node, making this naïve method of triangulation run in worst-case *O(n2)* time. Below is the pseudo code for the Naïve Triangulation Algorithm:

Algorithm: NaiveTriangulate(*D*, *D1*)

Input: A simple polygon *P* stored in doubly-connected edge list *D*. Doubly-connected edge list *D1* to hold the triangulated polygon.

Output: Polygon *P1,* represented by doubly-connected edge list *D1*, that is a decomposition of *P* into triangles created by adding new diagonals between certain vertices.

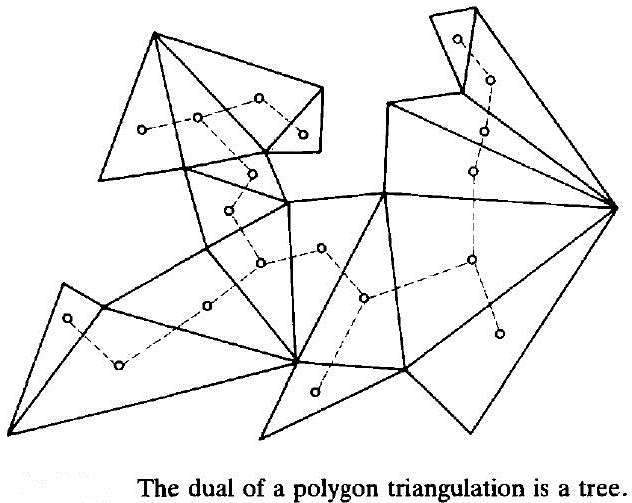
1. Start at the leftmost vertex in *D* (vertex with lowest x-value), and label it *v*.
2. Get list of both adjacent nodes to *v.* Call them *u* and *w*.
3. If ( *D*.isEmpty() ) // All nodes have been moved from *D* to *D1*.
4. Return;
5. If (3 edges representing triangle *vuw* already exists) // Base Case.
6. Remove edges *vu* and *vw* from *D* and insert them in *D1.*
7. If (open line segment *uw* does not leave the polygon) // Case 1 from example.
8. Add edge *uw* to edge list *D*, making it part of *P*.
9. Else // Case 2 from example.
10. Find vertex of *P* with greatest distance from open line segment *uw* that is inside triangle *uvw* (call it *v’*), and connect to vertex *v, adding edge vv’ to D.*
11. NaiveTriangulate(*D, D1*);

The following chapter on Monotone Decomposition reveals a method of triangulation that performs better than *O(n2)* by exploiting unique properties of convex polygons, but first, 3-Coloring will be discussed to wrap up this simple approach to solving The Art Gallery Problem.

Once a polygon has been completely triangulated, a method of guard-placement must be established. There are a few approaches to take here:

1. One guard can be placed at every triangle:
   * In this case there would be *n*-2 guards. This is more than is necessary.
2. Guards can be placed on diagonals:
   * This would give us about *n*/2 guards, but this would violate the property mentioned earlier that guards would only be placed on vertices.
3. Guards can be placed on a subset of vertices:
   * This gives us Fisk’s proven guards – NOTE: this is not necessarily optimal. Instead, it is a close approximation of the minimum number of guards necessary. The complexity of the problem prevents this solution from providing an exact solution.

In this case, a subset of vertices is chosen to place guards at. A *k-coloring* of chromatic number 3 is performed on each triangle, using the underlying dual graph (tree) that represents the triangulated polygon. It is from this characteristic that 3-coloring gets its name.



Under normal circumstances, k-coloring the nodes of a graph is an exponential-time algorithm. That is, for any *k*, k-coloring generally runs in *O(n2n)* time. Unfortunately, that kind of efficiency would make this algorithm entirely prohibitive. Instead, the fact that a tree is produced where each node of the tree represents a triangle of the triangulation, is exploited.

*Any* node of the dual tree can be used as a start node, and then just that corresponding triangle’s vertices are colored in *O(1)* time: one color is arbitrarily chosen for each vertex of the first triangle, represented by the first node of the dual tree. The second step simply involves a Depth-First-Search of the aforementioned dual tree. Each edge of the dual tree corresponds to a diagonal that was created by triangulation. This has a very important side effect – it means that once a new node is visited, two of its triangle’s vertices have already been colored, and only the last vertex need be colored, and the color to choose is trivial, nay, mandatory. Below is the pseudocode for 3-Coloring a triangulated polygon:

The dual tree of the triangulation is always a considerably sparse graph, so the number of edges are low, and a doubly-linked-list data structure will suffice for storing the nodes and edges. The DFS algorithm is chosen to traverse the dual tree for its *O(E+V)* running time, which is especially useful for sparse graphs. This holds true because DFS is called exactly once on each vertex, and every edge is examined exactly twice, once from each of its end vertices (Goodrich 614). Thus, for this type of 3-coloring using a dual tree of a triangulated polygon, the time complexity is *O(v+e)* for traversing the dual tree with *O(1)* work being done at each node(triangle) where only one node has to be colored at each iteration, except the very first. There is still one more step, however. Assuming that counters were implemented for each color as it was assigned, then the color with least assignments will already be known at the end of the traversal. The last step involves actually selecting the color with the least occurrences throughout the polygon’s boundary and, using counters, can be done in *O(1)* time. So, the overall run-time efficiency of solving The Art Gallery Problem using Chvatal’s Naïve Method of triangulation and Fisk’s Proof for 3-coloring is *O(n2)*.

Algorithm: ThreeColorDriver(*D*)

Input: A simple polygon *P* that has been triangulated already, stored in doubly-connected edge list *D*.

Output: Polygon *P* with all of its vertices 3-colored, one color of which will have the least occurrences, indicating the resolved guard placement.

1. Create doubly-connected edge list *D1* to store dual graph of *P*, using diagonals of *P* as edges of *D1*.
2. Visit first node of *D1*, call it *v*, and color all three vertices of corresponding triangle arbitrarily such that no two adjacent vertices share the same color.
3. ThreeColor(*D, D1, v*)

Algorithm: ThreeColor(*D*, *D1*, *v*) // Modified DFS

1. Label *v* as visited.
2. Label remaining uncolored vertex of underlying triangle with last remaining color.
3. For all edge *e* in *D1*.incidentEdges(*v*) do {
4. If (edge *e* is unvisited) then {
5. *w* = *D1*.opposite(*v, e*)
6. If (vertex w is unexplored) then
7. Recursively call ThreeColor(*D*, *D1*, *w*);
8. }
9. }

Conclusively, the Naïve Triangulation Algorithm, alongside a 3-coloring approach for guard placement solves the problem in quadratic time, and gives us an approximation that matches Chvatal’s theoretical guards. It is important to note however, that this is not always the “optimal”, or minimum number of guards necessary to monitor a floor, but it is indeed a very good approximation of that minimum in most situations. The next chapter shows a slightly more advanced algorithm for solving The Art Gallery Problem that has a better time complexity than this basic approach.

Monotone Decomposition

An alternative triangulation method that is better than *O(n2)* involves first decomposing a polygon into monotone pieces and triangulating those pieces. Through this approach we can achieve *O(nlogn)* triangulation.

A polygon *P* is l-monotone if, for every line orthogonal to l, *P* is intersected at most twice. A polygon *P* is convex if and only if, for every line l, *P* is l-monotone. Ideally, we would want to work with convex polygons, as for any convex polygon, the Naive Algorithm runs in *O(n)* time. Unfortunately, splitting a polygon into convex pieces is just as difficult and time-consuming as triangulating. Instead, we consider monotone decomposition to partition our polygon into monotone pieces, which lend themselves nicely to triangulation (Vlasic). For the purposes of this paper we will be looking at y-monotone decomposition, where y is the y-axis on the coordinate plane.

This method involves a more complex algorithm than the Naïve approach. The vertices of this method take on one of five different types: start, end, merge, split, and regular. A start vertex *s,* is a vertex whose two adjacent vertices are both below it (having a smaller y value) and the interior angle of *s* and its two edges is less than 180o. An end vertex has its adjacent vertices above it and also has an interior angle of less than 180o. A regular vertex is one where one adjacent vertex is above it and the other below. A merge vertex has both adjacent vertices above it with an interior angle greater than 180o, and a split vertex has both neighbors below it with an interior angle greater than 180o.

To partition *P* into y-monotone pieces we can imagine a horizontal line sweeping top to bottom over *P*. This line stops at each vertex and evaluates whether or not the polygon must be partitioned at that point. The polygon needs special care at vertices which are either split or merge vertices. Our goal is to eliminate split and merge vertices, in so doing we will partition the polygon into monotone pieces.

In order to eliminate the split and merge vertices we need to introduce diagonals that connect the split and merge vertices to other vertices in the polygon. To achieve this we give each edge *ej,* new information *helper(ej)*, which provides a pointer to the lowest vertex above the sweep line that, when connected to the current vertex, creates an edge that lies inside *P*.

For a split vertex, *ej* is the edge that is left of the vertex. In this case we can add a diagonal from the current vertex to *helper(ej)*. For merge vertices the vertex becomes *helper(ej)* and when we reach a vertex that would become the new *helper(ej)* we connect the two vertices with a diagonal.

Pseudocode derived from the work of de Berg, Cheong, Kreveld, and Overmars:

|  |
| --- |
| Algorithm: Monotone(*P*)  Input: A simple polygon *P* stored in doubly-connected edge list *D*  Output: A partitioning of *P* into monotone subpolygons   1. Construct a Priority Queue *Q* of the vertices of *P* using their y-coordinates as their priority. If two points have the same y-coordinate, the one with the smaller x-coordinate has higher priority. 2. Initialize an empty binary search tree *T*. 3. While *Q* not empty 4. Remove vertex *vi* with the highest priority from *Q*. 5. Use appropriate method to handle vertex based on its type. 6. End While   Functions:  StartVertex(*vi*)   1. Insert *ei* in *T* and set helper(*ei*) to *vi*   EndVertex(*vi*)   1. If helper(*ei-1*) is a merge vertex 2. Insert diagonal connecting *vi* to helper(*ei-1*) in *D* 3. End If 4. Delete *ei-1* from *T*.   SplitVertex(vi)   1. Search *T* for *ej* directly left of *vi* 2. Insert diagonal connecting *vi* to helper(*ej*) in *D* 3. Helper(*ej*)←*vi* 4. Insert *ei* in *T* and set helper(*ei*) to *vi*   MergeVertex(*vi*)   1. If helper(*ei-1*) is a merge vertex 2. Insert diagonal connecting vi to helper(*ei-1*) in *D* 3. End If 4. Delete *ei-1* from *T* 5. Search *T* for edge *ej* directly left of *vi* 6. If helper(*ej*) is a merge vertex 7. Insert diagonal connecting vi to helper(*ej*) in *D* 8. End If 9. Helper(ej) ←*vi*   RegularVertex(*vi*)   1. If interior of *P* lies to the right of *vi* 2. If helper(*ei-1*) is a merge vertex 3. Insert diagonal connecting ci to helper(*ei-1*) in *D* 4. End If 5. Delete *ei-1* from *T* 6. Insert *ei* in *T* and set helper(*ei*) to *vi* 7. Else search *T* for edge *ej* directly left of *vi* 8. If helper(*ej*) is a merge vertex 9. Insert the diagonal connecting *vi* to helper(*ej*) in *D* 10. End If 11. Helper(*ej*)←*vi* 12. End If |

After partitioning the polygon into y-monotone pieces a simple triangulation is made available where we sweep down over each individual polygon, connecting the current vertex to all vertices above the sweep line that, in doing so, does not create a diagonal that intersects another edge. If this triangulation were to eliminate triangles as it went, then the edge would try to connect the current vertex to all vertices above the sweep line that still are part of the polygon.

Delaunay Triangulation

There are cases when we wish to have not only a triangulation, but we would like those triangles to have some special properties. We end our discussion of triangulations with an *O(nlogn)* method that, while its implementation is not discussed in depth, deserves to be mentioned. Delaunay triangulation is an efficient method of producing triangles that hold certain unique properties, but we concern ourselves only with the property of an aesthetic triangulation.

Through the previously mentioned triangulation methods, we could produce “ugly” triangulations - those composed of many long and skinny triangles. Delaunay triangulation works to maximize the minimum angle of individual triangles. This produces triangles that are close to equilateral and whose vertices are relatively close together, giving a pleasant aesthetic look to the triangulation. Note, however, that this does not mean that no “ugly” triangles will exist, but that if an “ugly” triangle *does* exist, it exists for every triangulation.

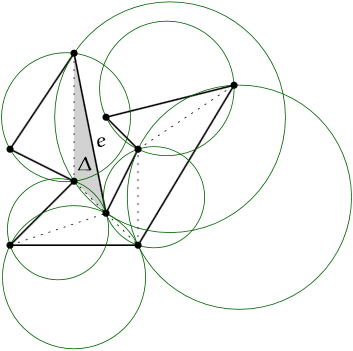
Strictly speaking a Delaunay triangulation operates on a point set, but we are interested in a polygon which is a point set with additional information. A polygon has edges that must be a part of some of the triangles. This means that a perfect Delaunay triangulation might not be possible. In these cases we call this a *Constricted Delaunay Triangulation*. Appropriately named because it is constricted in its choice of triangles to include the given edges.

Gärtner and Hoffman give us the following theorem about the existence of a Delaunay triangulation for a given polygon:

|  |
| --- |
| *For every finite point set P and every plane graph G = (P, E), there exists a constrained Delaunay triangulation of P with respect to G.* |

A triangle in a Delaunay triangulation exists if, for three vertices *v1, v2, v3* in the point set, their circumcircle exists such that no other vertex in the set lies within the circle. In the case where more than three vertices are concyclic, multiple triangulations can occur.

For a constrained triangulation let *P* be a finite point set, let *E* be a list of edges defining the polygon, and let *G = (P, E)* be a geometric graph with vertex set *P,* and edge set *E*. A triangulation *T* of *P* respects *G* if it contains all segments *e* ∈ *E* and is said to be a constrained Delaunay triangulation of *P* with respect to *G* if for every triangle *t* in *T,* the circumcircle of *t* contains only points of *P* in its interior that are not visible from the interior of *t*. A point *q* ∈ *P* is visible from the interior of *t* if there exists a point *p* in the interior of *t* such that the line segment does not intersect any segment *e* ∈ *E* (Gärtner & Hoffman 69).

****

Constrained Delaunay Triangulation

Conclusion

These are just a few algorithms for solving The Art Gallery Problem. The time complexity of solutions to this problem have a significant dependence upon the technique that is used for triangulation. Much research has been done that has made many of these more basic algorithms deprecated in preference of newer algorithms that are much faster. Here is what two experts had to say about it recently:

Notwithstanding that the best known theoretical bound for

convergence is Y(n

3

) iterations, our experiments show that an optimal solution is always found within a

small number of them, even for random polygons of many hundreds of vertices. Herein, we broaden the

family of polygon classes to which the algorithm is applied by including non-o rthogonal polygons.

Notwithstanding that the best known theoretical bound for

convergence is Y(n

3

) iterations, our experiments show that an optimal solution is always found within a

small number of them, even for random polygons of many hundreds of vertices. Herein, we broaden the

family of polygon classes to which the algorithm is applied by including non-o rthogonal polygons.

“Notwithstanding that the best known theoretical bound for convergence is *O(n3)* iterations, our experiments show that an optimal solution is always found within a small number of them, even for random polygons of many hundreds of vertices. Herein, we broaden the family of polygon classes to which the algorithm is applied by including non-orthogonal polygons.” (Couto, de Rezende & de Souza 2011)

That being said, the Naïve Algorithm with 3-Coloring is one available solution that runs in *O(n2)* time. Monotone Decomposition provides a solution that completes in *O(nlogn)* time. In a scholarly article by a computer science professor at Princeton, there is a theorem that states that “it is possible to compute the visibility map of a simple polygonal curve, and hence, a triangulation of a simple polygon, in linear time” (Chazelle 521). All of these recent advancements make it difficult to keep up with the most efficient algorithms being used, especially as their design complexity increases, but it is a very interesting subject nonetheless. All of The Art Gallery’s variations, amongst hundreds of others problems in computational complexity theory, fall under the class of non-deterministic polynomial time-hard (NP-Hard) decision problems (Rourke 239). There is a long-running debate over the “P=NP problem” in computational complexity theory, and The Art Gallery Problem, as a decision problem, is considered to be at least as hard as any NP problem, but possibly harder. The answer will depend on the answer to whether P=NP or not. This study continues on today. The Art Gallery Problem is not only NP-Hard and NP-Complete, but also “APX-Hard”, meaning it is hard even to efficiently get a good approximate answer, that is, one within less than a constant approximation factor.

These characteristics make The Art Gallery Problem a popular area of study, even today. Progress is being made in computational complexity theory, as well as the field of combinatorics, that are allowing solutions to certain problems be used to solve all sorts of other similar problems. Since its introduction in 1973, The Art Gallery Problem has seen some of the most incredible evolutions in algorithm design that computer science has ever seen. Going from worst case quadratic time to average case linear time in under 20 years, it is a true example of why data structures are studied the way they are.

References

Chazelle, B. (n.d.). Triangulating a Simple Polygon in Linear Time. *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*. Retrieved December 15, 2015, from https://www.cs.princeton.edu/~chazelle/pubs/polygon-triang.pdf

Couto, M., Rezende, P., & Souza, C. (2011). An exact algorithm for minimizing vertex guards on art galleries. *International Transactions in Operational Research*, 425-448. doi:10.1111/j.1475-3995.2011.00804.x

de Berg, M., Cheong, O., Kreveld, M., & Overmars, M. (2008). *Computational Geometry* (3rd ed.) Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg

Gärtner B., & Hoffmann M., (2013) *Computational Geometry Lecture Notes HS 2012*.Personal Colelction of B. Gärtner and M. Hoffmann, ETH Zürich, Zürich, Switzerland

Goodrich, M., & Tamassia, R. (2010). *Data structures and Algorithms in Java* (5th ed.). New York, New York: J. Wiley & Sons.

Rourke, J. (1987). *Art Gallery Theorems and Algorithms*. New York, New York: Oxford University Press.

Vlasic, D. (n.d.). Polygon Triangulation. Retrieved December 15, 2015, from http://groups.csail.mit.edu/graphics/classes/6.838/F01/lectures/PolygonTriangulation/