Optimal Double Coverage In The Art Gallery

Scott Dalane
sdalane1@swarthmore.edu

Andrew Frampton
aframpt1@swarthmore.eu

Abstract

We plan to examine a well known visibility problem termed the art gallery problem. The general idea behind the problem is that a museum wants to minimizethe number of cameras present in an art gallery while still recording everythingthat happens in the room. We take the problem a step further and investigate the optimal arrangement of cameras so that every spot in the room is covered by not just one, but two cameras. This will allow constant surveilance even if any one camera were to fail. We do this by using a bottom-up approach where we use a three coloring algorithm to find the optimal placement for single coverage and a basic dobuble coverage of a polygon representing a floorplan. We then use a optimization algorithm with a line of sight algorithm to check each camera of the double coverage and remove any unnecessary cameras, leaving the optimal double coverage of the polygon.

1 Introduction

The problem is named after art galleries because the art needs to be under surveilance at all times and they tend to be shaped in irregular ways which can make this security difficult. The problem was first proposed by Viktor Klee in 1973 (Klee, 1979). It has since been proven that a maximum of $\lfloor n/3 \rfloor$ cameras will be needed, n being number of vertices in the polygon that is the room, to cover the entire room. This is known as the Art Gallery Theorem and was stated by Vaclav Chvatal (Chvatal, 1973). For us to tackle this problem we designed a program that takes a set of points that make up the polygon that will be guarded, and then returns an image of the floorplan with the cameras placed upon it. By using a three coloring algorithm on each of the points in the triangulation determine the ideal single camera placement and the basic double coverage placement of cameras, which can then be optimized.

2 Triangulation

In order to perform the 3-coloring on the points of the polygon, it is first necessary to break the polygon down into its constituent triangles, so that we can accurately determine whether the 3-coloring is accurate. We do this by finding any splitpoints within the polygon so that it can be seperated into monotone polygons. These monotone polygons are then individually triangulated, then put back together to create an ideal triangulation of $n-2$ triangles for n points.

2.1 Finding Split Points

The first step in our triangulation algorithm was to locate any split points in the polygon that we can use to break the polygon down into smaller, monotone polygons to triangulate. The split point is a point that is determined to be interior to its neighboring points on the y-axis, in other words, when the point causes the polygon to be convex on the top or bottom. This is done by going through each point and shooting a rays up and down the point's x-axis (Seidel, 1991). We then use helper functions to determine whether or not the rays intersect any other segments of the polygon. If both rays hit a segment of the polygon, they are then tested on the basis of how many times they intersect the polygon to determine whether they are interior to the polygon. If the point and its rays

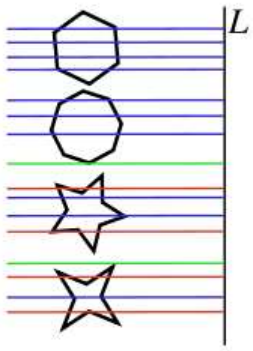

Figure 1: The top two polygons are monotone because every line only intersect the polygon twice.[1]

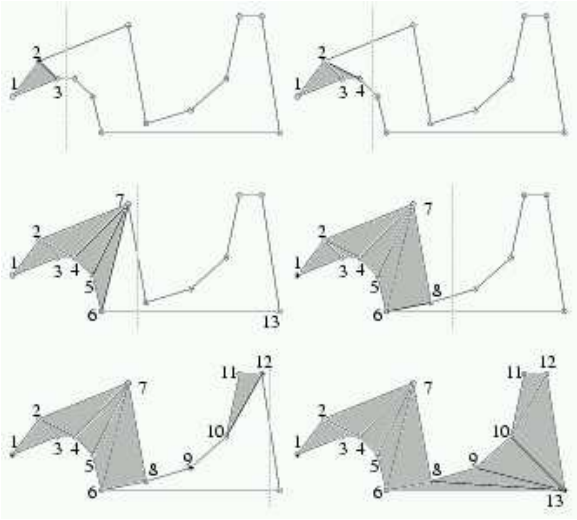

Figure 2: An example of triangulation around start points.[2]

meet these criteria they are then flagged as a split point and the polygon is then split into three polygons, one to the upper right of the point, one to the upper left, and one beneath the point, with the bases of the upper two polygons and the top of the lower polygon formed by the rays along the x-axis of the point. This continues until the original polygon has been completely broken down into monotone polygons (de Berg et al., 2000).

2.2 Monotone Triangulation

Since monotone polygons have no interior points on the y-axis, they get rid of several possible degeneracies when being triangulate as opposed to trying to triangulate the entire poylgon at once. Once the monotone polygons have obtained, each point within them is checked as the start point for the triangulation using a method similar to the one used for finding split points, but this time we check to see if they are interior to their neighboring points in regard to the x-axis. If such a point is found, then it is chosen as a start point, otherwise the split point is chosen. The triangulation algorithm then walks around the polygon's points using a list of connections stored in each point to determine the points it is connected to, and if there is no connection to the start point, then one is added and a line is formed. The shared point between the two previous points is added and then the three are saved as a triangle, and this continues until the entire polygon has been traversed. After each monotone triangle has been triangulated, they are all put back together in the polygon, and the horizontal rays from the split points are removed, creating an optimally triangulated polygon (de Berg et al., 2000).

2.3 Handling Degeneracies

It is worth noting that the algorithm that we have come up with in this is capable of handling just about any degeneracy that we encountered. If the polygon is concave, our algorithm can deal with the problems of possible outside segments as well as any problems that may arise from having a vertex in the middle of the polygon. the outside segment problem is done by our suprisingly simple intersection test which takes the modulus of the number of interstections that the ray encounters with both edges and vertices. If the number of intersections is even then the resulting modulus is zero, and the ray is considered outside and discarded, as was previously explained in section 2.1.

3 Optimizing Camera Placement

3.1 3-coloring and Camera Placement

With the triangulation in place it is now possible to perform an easy 3-coloring of the polygon, so that cameras can be placed. Using the triangulation within the polygon, we can now proceed to use

[1] image from http://en.wikipedia.org/wiki/Monotone_polygon

[2] from http://www.cs.ucsb.edu/šuri/cs235/Triangulation.pdf

three coloring to determine where to place the cameras. Using the list of triangles that was created during the monotone triangulation, our algorithm takes the first triangle from the list and colors each point a different color. The algorithm then looks for triangles that share sides with the previously colored triangle and color the remaining point is then colored based on what coloring of the other two points are. This continures until all of the triangles in the list have been colored, with the number of the points with each coloring stored as an integer. The integers are then compared with one another, cameras being placed at the color with the lowest integer value, creating optimal single coverage for the polygon (Urrutia, 1991) . But in order to achieve double coverage another set of cameras have to be placed at the next smallest coloring to provide a basic double coverage which can be optimized.

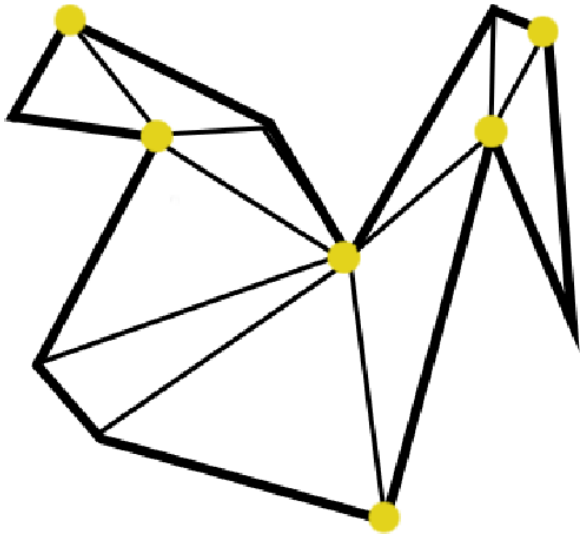

Figure 3: Simple double coverage on a test polygon using only 3-coloring.

3.2 Visibility Graph

Now that the cameras have been placed it is time to determine what points they see and and what how many cameras see each point, assuming that if a camera could see a point then it generally sees the area surrounding the point, barring degenerate cases which are taken care of when the cameras are optimalized later on. Using a line of sight function,we created a visibility graph by traversing every point from each camera, drawing a line between the two points, making sure that the line did not intersect a segment of the polygon and remained inside of it. By doing this we then built up a list of each point seen by each camera, with thenumber of cameras seeing each point, allowing our heuristic to optimalize the camera placement (O'Rourke, 1987).

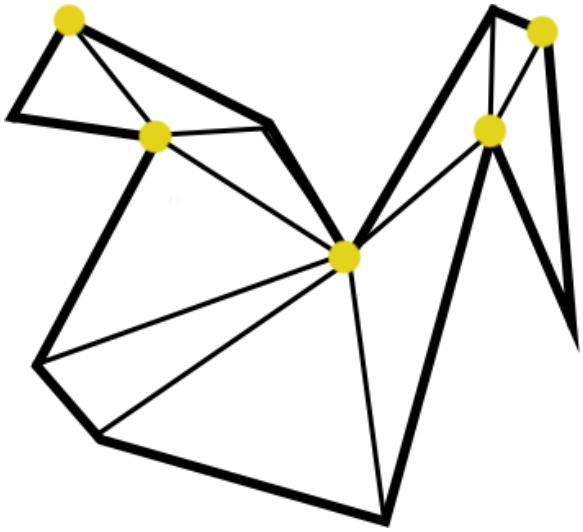

Figure 4: Optimized double coverage using a visibility graph and optimization.

3.3 Camera Optimalization

Now we have our optimalize the camera placement by going through each camera and seeing if it can be removed from the polygon, updating the each point it sees and decrements the number of cameras that can see them accordingly. If one any point's count of cameras that can view it drops below 2, then the camera has to be replaced, otherwise, it remains taken off and the next camera is then checked. After the optimization algorithm is complete, the cameras left are the optimal.

4 Results and Conclusion

After testing our algorithm on progressively more complex polygons, our algorithm easily does better than $\lfloor 2n/3 \rfloor$ camera placement for double coverage of cameras outside of the the worst case, while the entire program runs in n^2 time, due to the fact that the points are looked at multiple times espescially during the optimalization step. It also proves fairly adept at making a strong placement for coverage greater than double, although it needs to be more thoroughly tested to make any conclusions in that regard.

References

V. Chvatal. 1973. A combinatoral theorem in plane geometry. Number 18, pages 39–41.

M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. 2000. Computational geometry: Algorithms and applications (2nd ed.). pages 45–61.

V. Klee. 1979. Some unsolved problems in plane geometry. volume 52, pages 131–145.

Joseph O'Rourke. 1987. Art gallery theorems and algorithms. pages 11–23.

Raimund Seidel. 1991. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. In *Computational Geometry The- ory and Application, vol. 1, no. 1, pp. 51-64.*

Jorge Urrutia. 1991. Art gallery and illumination problems.