Dsimulator Redesign:

Geometric Inclusion Algorithm Analysis

Written by Matthew Brown May 25, 2018

# Abstract

The Dsimulator portion of the Slitmask Redesign Project requires the functionality of the current IRAF Dsimulator to be copied to another tool that does not rely on IRAF. The current prototype developed by Dr Shui Kwok uses JavaScript to manipulate a Digital Sky Survey (DSS) image and a detector canvas overlay in order to select a section of the sky to focus on. The tool takes an input file with a catalog of stars in order to determine what stars in the section of the sky are of interest. Part of the functionality of the tool is to show which stars are within the geometry of the detectors. The geometry of the detectors is stored within a JavaScript array and can be used in the determination of inclusion within the CCDs. The trick to this problem is that we are dealing with four shapes instead of one. In this document, we look over simple solutions while aiming to keep the performance of the tool to a reasonable time frame.

# Introduction

The DEIMOS dsimulator tool is an essential part of the DEIMOS pre-planning phase if the observer wants to take spectra of multiple objects. However, the core of the tool is built on the IRAF (Image Reduction and Analysis Facility) language which has not seen a stable release since March 2012[1]. The version of IRAF that dsimulator runs on is 2.14 which was released in December of 2007[4]. This dependency makes it very hard for users to install dsimulator on their local machine and use it to create their masks ahead of time as required. Thus, a version has been installed on aging machinery in the operations network and exposed to the world. This creates an unnecessary security risk and unreasonable maintenance requirements. The goal of this project is to update the dsimulator tool in a more modern, easy to maintain language. The purpose of this paper focuses on one specific task this tool must be able to accomplish: whether a translated point lies on one of the four detectors that DEIMOS uses. It explores an application of set theory known as the Geometric Point Inclusion or more commonly as the Point in Polygon problem.

## Geometric Point Inclusion Problem

The basic concept of geometric point inclusion is a simple question: Does a given point exist within a given shape? The trivial solution requires the point to be checked against the boundary of the shape in order to determine if the point is within the shape and runs in O(n) time. While the base case of a single point is trivial to evaluate, extrapolation of the trivial solution to hundreds or thousands of points can become cumbersome. Since the tool should be designed for efficient and fast use, it is important to look for ways to do this point classification as quickly as possible. Luckily, the Geometric Point Inclusion problem is a well-studied problem with a lot of solutions and literature to back those solutions up. For example, two common solutions include the Crossing Number and the Winding Number methods.

## Convexity vs Concavity

Whether a shape is convex or concave can change how efficient the algorithm is. For example, a convex shape can simplify the checks needed to classify a point and drop the complexity to O(log n)[3]. A shape is convex shapes if you can choose any two of its points and connect them without leaving the shape[5]. For concave shapes, there is at least one pair of points where the line drawn exits the shape before reentering it[5]. However, since the detectors are blocked by a cylindrical vignette, it introduces a concave element to the overlay. While the convexity simplifications cannot be applied to this problem, it is worth noting for future ventures.

# Literature Review

## Crossing Number or Ray Casting Method

In the Crossing Number method, a ray is drawn from a given point to a place outside the boundary of the shape. The number of times the ray crosses the boundary of the shape determines whether the point is inside (odd) or outside (even) of the shape[2]. There are a few things to consider when applying this method. Mainly, if a ray crosses a boundary, is it crossing at a vertex or edge? This can throw off the times it has crossed the boundary and thus misclassify a point.

## Winding Number Method

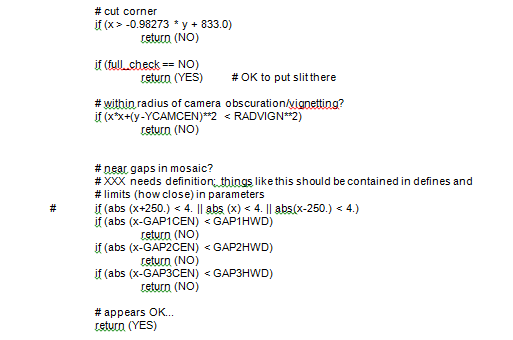
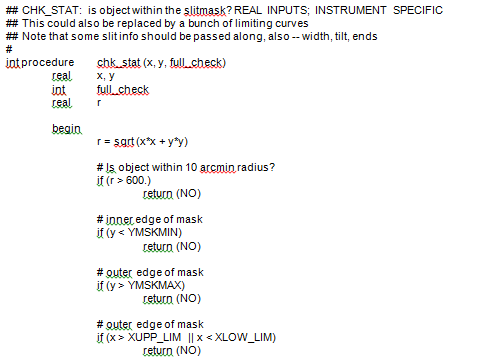
In the Winding Number Method, the inclusion of a point in a closed shape is based on how many times the shape wraps around the point[2]. Specifically, if the winding number of the point is greater than 0, then it is considered inside the shape. This method allows for points to be correctly classified even when dealing with complex shapes that overlap layers[2]. Dan Sunday modifies the Crossing Number method with the Winding Number checks to handle many of the shortcomings of the Crossing Number method while maintaining the efficiency of the Crossing Number method.

# Use Cases

The algorithm will be used after a user stops moving a piece of the canvas and releases the mouse. It will have to reclassify all the points based on the new position of the detector overlay on the given star list. The star list could be a single entry up to hundreds of thousands of entries.

# Old Solution

In the old dsimulator code, each point was run through a series of checks to see if they were within the detector geometry. The old code is shown below:



# Proposed Solution

Since the trivial solution is at worst O(n)[3], it should handle a thousand, ten thousand, or even a hundred thousand points with relative ease. Some optimizations can be added to make it more efficient. For example, the inclusion algorithm does not have to be applied to every single point. A lot of the work can be saved by checking if either the x or y-value of a point exceeds the respective minimum or maximum x or y-value of the box that runs along the outer edges of the detectors. Values that fall within the box can then be checked with with the old solution checks. In order for this to work easily, a function modelling the camera vignette and the box corners would need to be formulated. Once these things have been assembled, the application should be able to handle a list of points provided by the user without much effort. If there are performance issues, then this situation will have to be revisited.

# Conclusion

If it ain’t broke, don’t fix it! That old adage has some application as much of the old algorithm is still a good solution to the problem. The thing that needs to be fixed is the implementation of it. IRAF is no longer a viable language to be maintained by the organization. The jump to a web application will provide access to this tool for observers while they are not at the headquarters without requiring them to tunnel into the operations network. Now it just needs to be implemented.

# References

[1] IRAF V2.16 Release Now Available. (March 2012). Retrieved May 25, 2018 from http://iraf.noao.edu/

[2] Dan Sunday. 2012. Inclusion of a Point in a Polygon. (2012). Retrieved May 23, 2018 from http://geomalgorithms.com/a03-\_inclusion.html

[3] David Eppstein. 1996. ICS 161: Design and Analysis of Algorithms Lecture notes for March 7, 1996. (March 1996). Retrieved May 25, 2018 from https://www.ics.uci.edu/~eppstein/161/960307.html

[4] Drew Phillips. "Important Notes." DSIMULATOR. (April 2018). Retrieved May 25, 2018 from https://www.keck.hawaii.edu/realpublic/inst/deimos/dsim.html.

[5] Weisstein, Eric W. "Concave." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/Concave.html