Description of the Problem

Eikonal equations can be used in a number of different scenarios. They can be applied to such areas as crystal growth, computer vision, optimal control, and geometric optics [3]. The equation is the manifestation of how information travels from a boundary to an exit set defined inside of the domain [3]. In terms of the partial differential equation that the eikonal equation models, it specifically solves information that travels along the characteristics [3]. The equation is:

$$\begin{aligned} |\nabla u(\mathbf{x})| F(\mathbf{x}) &= 1 \text{ on } \Omega \subset \mathbb{R}^2 \\ u(\mathbf{x}) &= q(\mathbf{x}) \text{ on } \partial\Omega \end{aligned}$$

If F is equal to one and the boundary condition is zero, then the solution can be found by the distance between the point being considered, x, and the boundary [3]. If F is dependent on x, $u(x)$ can be thought of as the phase of a wave traveling in the direction of the boundary at a particular propagation or speed [3]. The F in the equation represents that speed and is given as input data [1]. The goal is to find U which is accomplished through discretizing each point by looking at its surrounding neighbors. In [1], an upwind scheme was used therefore that method was used in this paper as well.

Description of the Algorithm

Eikonal equations are solved graphically. The paper explores two core algorithms for achieving this: Fast Marching Method (FMM) and Fast Sweeping Method (FSM). Both FMM and FSM are graph solving methods where there is a predefined exit set and the equation is computed to find the quickest ways of getting from the boundary to the exit set. The Fast Marching Method is a label-setting method [1]. Label-setting methods are useful because they have the best worst case running time for asympototic complexity [1]. The FMM is best used on domains that have a complicated geometry and where the characteristic directions constantly change [1]. FMM is used in this type of problem because it is most efficient and outperforms FSM. FSM performs the best when the characteristic directions are constant. FSM is a label-correcting method which does not have as good of a worst case running time as label-setting methods but under certain conditions, will highly outperform the label-setting methods, such as FMM [1].

Because both of these methods have pros and cons, [1] proposes three hybrid solutions where the best of both FSM and FMM will be integrated into an algorithm. Their first approach involves placing a coarse graph on top of a fine graph where FMM will be run on the coarse graph to begin and then FSM will follow on the fine mesh. This combination will be known as the Fast Marching-Sweeping Method (FMSM). The next hybrid method considers errors that are introduced by FMSM. The potential for errors lies in the fact that on the fine grid there may be interdependencies that are overlooked by FMM only going over the graph once. The method that will account for the interdependencies will be known as Heap-Cell Method (HCM). Lastly, HCM will be refined to Fast Heap-Cell Method (FHCM) where the number of times a cell is looked at is dependent on the boundary conditions. These three methods all produce a new type of efficiency in solving the eikonal equation. When proper grid sizes are chosen, these hybrids methods are significantly faster and more efficient than the Fast Sweeping and Fast Marching Methods.

All of these algorithms are based around an upwind discretization to approximate each point on the grid, U. Using the method described in [1], we let $x_{ij} = (x_i, y_j)$ and let $U_{ij} = U(x_{ij})$ and $U(x_{ij}) \approx u(x_{ij})$. F is the speed at which we want to advance to the exit set so $F_{ij} = F(x_{ij})$. Upwind discretization would be as follows as described by [1]:

$$(\max(D_{ij}^{-x}U, -D_{ij}^{x}U, 0))^2 \quad + \quad (\max(D_{ij}^{-y}U, -D_{ij}^{y}U, 0))^2 = \frac{1}{F_{ij}^2}$$

where

$$\begin{aligned} u_x(x_i, y_j) &\approx & D_{ij}^{\pm x}U = \frac{U_{i\pm1,j} - U_{i,j}}{\pm h} \\ u_y(x_i, y_j) &\approx & D_{ij}^{\pm y}U = \frac{U_{i,j\pm1} - U_{i,j}}{\pm h} \end{aligned}$$

Through analyzing this equation, we are able to come up with the following two equations based on whether an x or a y direction is calculated or one of each is calculated.
Case 1: Only an x or y direction is calculalted

$$U = h/F + U_{\pm x} \text{ or } _{\pm y} \text{ where } U_{\pm x} \text{ or } _{\pm y} = U_N, U_S, U_E, U_W$$

Case 2: Both an x and y direction are calculated
As an explaination, we will consider the case where the x direction is U_W and the y direction is U_S

$$\begin{aligned} (\frac{U - U_W}{h})^2 \quad + \quad (\frac{U - U_S}{h})^2 &= \frac{1}{F^2} \\ U^2 - (U_W + U_S)U + (U_S^2 + U_W^2 - \frac{h^2}{2F^2}) &= 0 \end{aligned}$$

The solution to this will be the root U_{SW}. This root must satisfy $U \geq \max(U_N, U_E)$. In the event that this condition cannot be satified, then $U_{SW} = \frac{h}{F} + min(U_S, U_W)$. Once this is calculated, the updates from the other three quadrants, U_NE, U_NW and U_SW, need to be calculated and then U is defined as $U = \min(U_{NE}, U_{NW}, U_{SE}, U_{SW})$ [1].

Knowing the core of the algorithms, we will now delve into the details of FMM and FS-M. FMM was designed with the idea of front propagation. In this algorithm, an exit set is defined, and the values around the exit set are set initially. From here, the points are discretized in the order of the smallest node first. To do this efficiently, all the nodes to be considered are put into a binary heap which allows for least cost expensive removal from the queue making this the best structure to choose. Every time the minimum value is removed, the list is then reordered. This allows for a total complexity to be $O(MlogM)$ where M is the number of grid points [1].

FSM has a different construct. It depends on Gauss-Siedel iterations while looking in alternate directions, which are the sweeps. These alternating sweeps eventually converge to a discretized U value at each point. Due to this method being defined through iterations, its complexity is $O(M)$ where, once again, M is the number of grid points. These two different approaches combined together form the FMSM, HCM, and FHCM hybrid methods [1].

What was Implemented

For the purpose of this experiment, a full version of the Fast Marching Method was implemented. By dissecting, thoroughly analyzing, and implementing this method, a deeper understanding of FMM was achieved. The reason only FMM was considered was so that it could be thoroughly and completely understood.

The algorithm presented in [1] for the FMM was used as the framework for this particular algorithm. The layout of the algorithm is found on page 554 of [1]. The input for the program includes a grid size, so that the domain is always square, and an exit set. The speed for every point starts out at being set to infinity. The initialization of the program includes that if a point x_{ij} is in the exit set, then it is labeled as *Accepted* and its speed is updated based on the upwind discretization. If x_{ij} is not part of the exit set, it is labeled as *Far* and its speed remains at infinity. Then for every *Far* neighbor of each *Accepted* point, it is labeled as *Considered* and then its temporary speed is calculated using the upwind scheme. If the temporary speed is less than the initial speed, then the speed is updated to the temporary speed. This is the end of the initizliation process.

The next step in the algorithm is while the *Considered* list is non empty, the smallest x_{ij} should be removed. From here, its temporary speed is calculated using the upwinding discretization. If this is less than the initial speed, the speed is updated to the temporary speed. Then for each neighbor of that point, it is labeled as *Considered* and the process is repeated. For this specific implementation of this algorith, a binary heap was not used but rather just an array. The purpose a binary heap was not implemented was because the algorithm was being designed for workability and understanding not efficiency. As stated, this algorithm comes from the framework proposed by [1].

Experiments Performed and the Results

To analyze the algorithm, several different types of experiments were performed. It is important to know that the exit set is shown by the darkest blue on the graph. The first was run on an open domain with the exit set defined in the middle. What this shows is how the eikonal equation finds the fastest way from the edge to the exit set. It follows that starting at the north, south, east, and west poles would give the quickest paths. This can be seen by noting the darkest colors in the corners where it would take the eikonal equation the longest to get to the center (Figure 1).

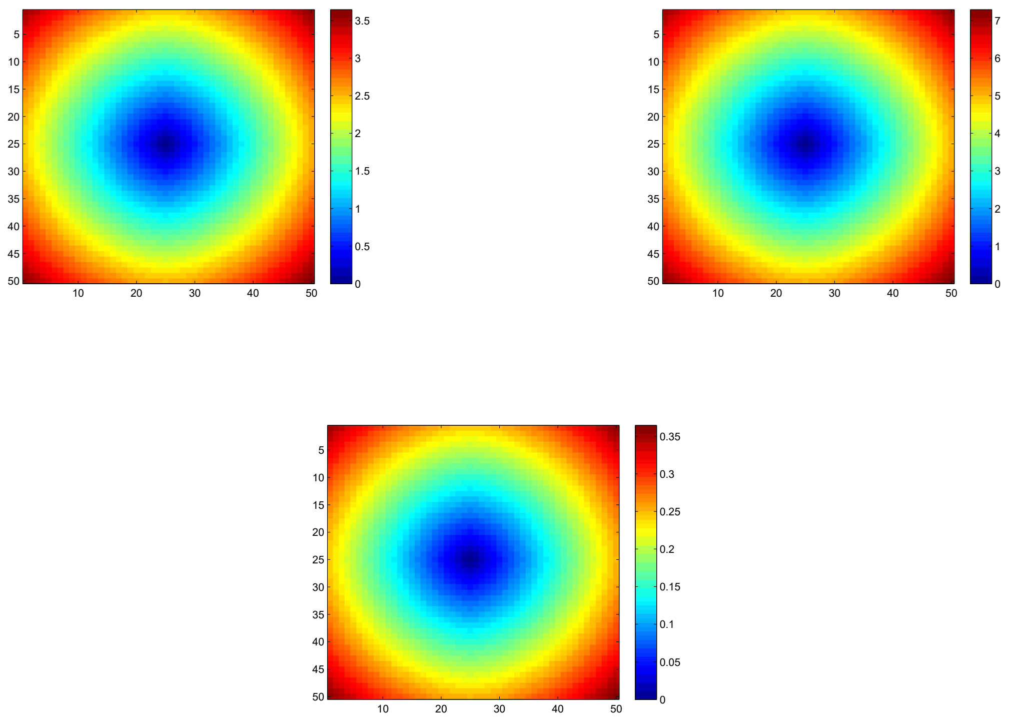

Figure 1: The upper right figure has $F = 1$, the upper left figure has $F = \frac{1}{2}$ and the bottom figure has $F = 10$. It is important to notice here that all the pictures are similar but their scales are different.

Next we introduce boundaries around the outside. This experiment was done to correctly implement the ideas of boundaries and how this should affect the eikonal equation. In addition to looking at the boundary conditions, one can see how the exit set has been moved from the center and what effect this has on the eikonal equation searching for the exit set. It can be seen how the colors to the exit set change with where the exit set is located (Figure 2).

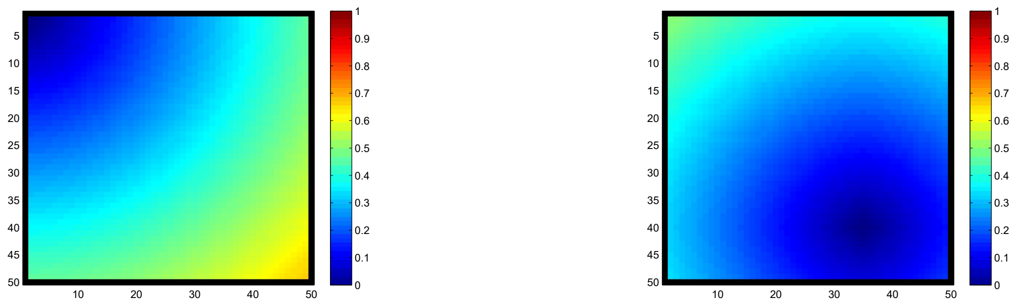

Figure 2: An example of outer boundary conditions and a moved exit set

The next step was to look at how the graph handled boundaries that were introduced inside of the graph. This way it could be analyzed how the pattern of the eikonal equation changed as it worked around boundaries. To see this, experiments were run on a maze boundary problem as well as a spiral boundary problem. Using the FMM algorithm with the eikonal equation on the maze, it solved the maze, and when it was implemented on the spiral, it solved from the middle of the spiral out (Figure 3).

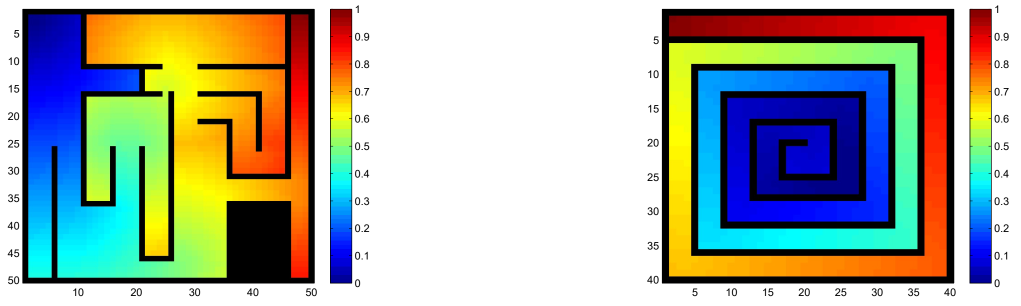

Figure 3: The graph on the left has FMM solve a maze and the graph on the right has the eikonal equation travel out of a spiral to it's exit set. The F value of both was 1.

Up to this point, the F value had been held constant across the domain. This led to the next experiment investigating what the graph would look like if there were multiple F values present on the graph. To explore this, the graph was set on a domain where the exit set was centered. Next, the graph was split to contain two different F values (Figure 4).

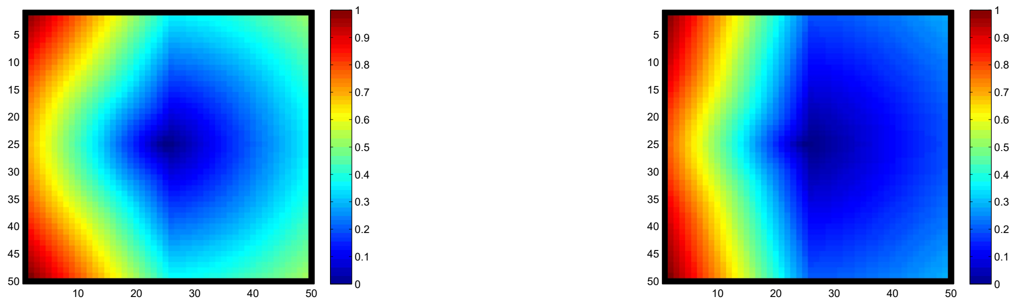

Figure 4: An experiment to look at multiple F values. The graph on the left has $F = 1$ on the left and $F = 2$ on the right. The graph on the right has $F = 1$ on the left side and $F = 4$ on the right side.

Once all of these different scenarios were explored, it was important to try and replicate some of the experiments performed in the paper. One of the major scenarios they looked at was a checkerboard problem where all white (odd) squares had an F value of 1 and all black (even) squares had an F value of 2. The following figure closely replicates the one in the paper. It shows how the eikonal equation flucates and changes in waves on the graph.

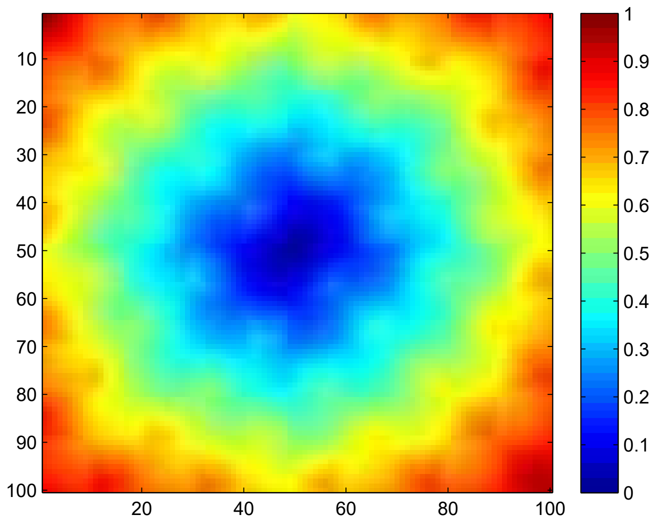

Figure 5: Checkerboard problem from the paper

The following two graphs show more extreme values of the previous figure.

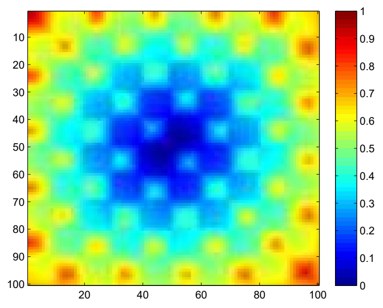
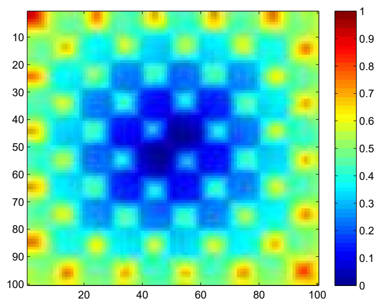

Figure 6: The graph on the left has $F = 1$ on the odd squares and $F = 6$ on the even squares. The graph on the right has $F = 1$ on the odd squares and $F = 12$ on the even squares.

How the Algorithm Performed

Following the analysis of the performance of the algorithm, it was important to look at the running times for several different grid sizes (Figure 7).

Size	**Time** (s)
50 x 50	1.1856
100 x 100	5.7720
250 x 250	37.5062
500 x 500	436.0072

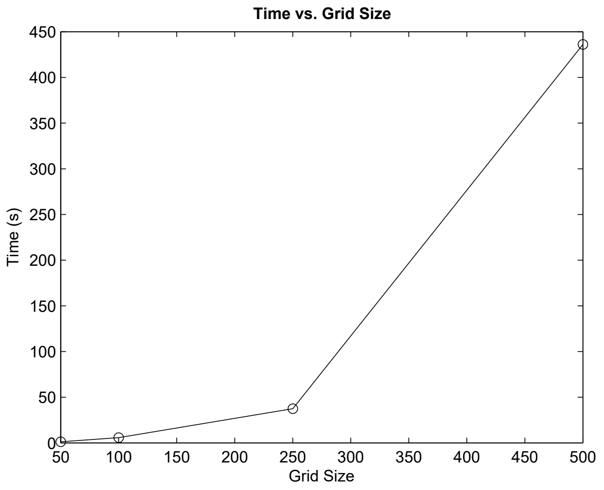

Figure 7: This shows the exponential growth of the time given grid size.

The experiment performed in the paper involved an 11 x 11 checkerboard split into a 1408 by 1408 grid. Unfortunately the experiment was not able to be exactly replicated. The algorithm is able to process a 1000 by 1000 case, it was just not possible to exactly replicate the experiment due to a computer that was not powerful enough. However, the checkerboard was able to be run on a 10 by 10 checkerboard that was split into 100 by 100 mesh. When the white squares had an F value of 1 and the black squares had an F value of 2, the running time was 4.5396 seconds. When the black squares had an F value of 6 and the white squares had the same F values, the running time was 4.3992 seconds. Lastly, the F value was switched to 12 on the black squares, and the running time was 4.6332 seconds. As one can see, the running times are very comparable to each other when the mesh remains the same and the F values are changed.

Advantages, Disadvantages, and Limitations of the Algorithm and General Usefulness

When implementing the Fast Marching Method, the biggest area of concern was how to deal with the boundary conditions. All boundaries created some type of problem that had to be accounted for when looking at which directions to consider. For example, if the point is the upper left hand corner, the program should only look east and south. This creates the need for multiple classes to handle certain cases of boundary conditions.

The limitation of this FMM algorithm lie in the fact that even in the simplest cases of constant characteristic directions, it is not any better than Fast Sweeping Method. FMM excels in the area of domains with complicated geometries and has characteristic directions that are constantly changing. FSM performs best on problems where the characteristics are constant. However, domains with constant F can be run on FMM, and the goal of this paper was to implement [1]'s experiment of the checkerboard using FMM where constant F was used.

When one extends the analysis of limitions and disadvantages and usefulness to the hybrid methods, there is more to discuss. The hybrid methods presented in [1] have more set up involved but the payoff to that is that the resulting algorithms, FMSM, HCM, and FHCM, are all more efficient overall. By combining the best aspects of FMM and FSM and having them run over two different grids to highlight those aspects, the running times become much more efficient and the solutions converge faster. Despite the efficiency that is gained, some errors are introduced into the program. However, the authors of [1] performed many experiments where they found that these newly introduced errors did not have a large effect that would cause concern especially when compared to the erros already introduced by discretization.

Open Problems

With the experiment for this paper, there are many more directions that could be taken. To start with, running large grid sizes on different domains could be simulated. Additionally, the F input values could be something other than constant scalars. Also, the program could be set up to accept multiple exit sets. From there, FSM could be implemented and the rest of the experiement in [1] could be completed.

[1] also had further research that could be completed. The authors proposed that rather than using uniform grid sizes that varying grid sizes could be explored which would make their program more autonomous by sllowing cells to be non uniform. Eventually, they want to be able to solve higher order problems. Their main future goal is that the data and algorithms presented in their paper can serve as a basic way of decompositiong domains and help with static nonlinear PDEs.

Conclusions

The paper fully analyzed one method of solving the eikonal equation on a graph. The Fast Marching Method (FMM) does a front wave propagation from the boundaries of a domain to a predefined exit set. By looking at different experiments of constant and split domains as well as different problems to solve such as mazes, spirals, and checkerboards, it was shown that the method solved the eikonal equation effectively and efficiently. With this kind of understanding, the rest of the method from [1] could be implemented with solid understanding of one of the core algorithms, a Fast Marching Method.

References

1. A. Chacon and A. Vladimirsky, Fast Two-scale Methods for Eikonal Equations. Sci. Comput. 34. A547-A578.
2. J. A. Sethian, Level Set Methods and Fast Marching Methods. 2000 Cambridge University Press. 86 - 90.
3. M. Dehghan and R. Salehi, A Seminumerica Approach for Solution fo the Eikonal Partial Differential Equation and Its Applications. Wiley InterScience 2009 July. 702 - 703.