3D TERRAIN VISUALIZATION AND CPU PARALLELIZATION OF PARTICLE SWARM OPTIMIZATION

A Thesis

Submitted to the Faculty

of

Purdue University

by

Calvin L. Wieczorek

In Partial Fulfillment of the

Requirements for the Degree

of

Masters of Science in Electrical and Computer Engineering

May 2018

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Lauren Christopher, Chair

Department of Electrical and Computer Engineering

Dr. Brian King

Department of Electrical and Computer Engineering

Dr. John Lee

Department of Electrical and Computer Engineering

Approved by:

Dr. Brian King

Head of the Graduate Program

I would like to dedicate this to my loving family, especially my mother, Denise, for always being there for me and supporting me throughout my life.

ACKNOWLEDGMENTS

I would first like to acknowledge my professor Dr. Christopher for allowing me the opportunity to work with her on this project, she has been a great teacher and role model to me; her guidance and experience has been an invaluable part of this projects' success. I'd also like to thank my other committee members Dr. King and Dr. Lee, for their support and knowledge they've taught me. Thanks to Sherrie Tucker and Jane Simpson for their support and guidance. I want to acknowledge Dr. Eberhart and Dr. Kennedy for inventing PSO. I would like to acknowledge my previous members of the project and their contributions: Joshua Reynolds, Jonah Crespo, and Paul Witcher. I want to acknowledge my Md Saiful Islam for being a dedicated partner and his support to the project. I also want to acknowledge William Boler for being a close dedicated partner as well as for his invaluable help, support, and experience throughout the project. I would like to thank our contacts at Crane on this project: Dave Acton for your guidance and relationship to our research, as well as Scot Hawkins for this support, guidance for EW propagation, and sponsorship. Lastly, I would like to thank NSWC Crane for providing support and funding to this project.

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

Figure Page

ABBREVIATIONS

2D	2 Dimensional
3D	3 Dimensional
API	Application Program Interface
CPU	Central Processing Unit
CSV	Comma-Separated Values
EW	Electronic Warfare
FPGA	Field Programmable Gate Array
GPSO	Gaming Particle Swarm Optimization
GPU	Graphics Processing Unit
GIS	Geographic Information System
GUI	Graphic User Interface
JPEG	Joint Photographic Experts Group
NSWC	Navel Surface Warfare Center
NRCS	Natural Resource Conservation Service
NED	National Election Dataset
OOP	Object Oriented Programming
OpenCL	Open Computing Language
OpenMP	Open Multi-Processing
PSO	Particle Swarm Optimization
PNG	Portable Network Graphics
RAM	Random Access Memory
SRTM	Shuttle Radar Topo-graphical Mission
TIFF	Tagged Image File Format

ABSTRACT

Wieczorek, Calvin L. M.S.E.C.E., Purdue University, May 2018. 3D Terrain Visualization and CPU Parallelization of Particle Swarm Optimization. Major Professor: Lauren Christopher.

Particle Swarm Optimization is a bio-inspired optimization technique used to approximately solve the non-deterministic polynomial (NP) problem of asset allocation in 3D space, frequency, antenna azimuth [1], and elevation orientation [1]. This research uses QT Data Visualization to display the PSO solutions, assets, transmitters in 3D space from the work done in [2]. Elevation and Imagery data was extracted from ARCGIS (a geographic information system (GIS) database) to add overlapping elevation and imagery data to that the 3D visualization displays proper topological data. The 3D environment range was improved and is now dynamic; giving the user appropriate coordinates based from the ARCGIS latitude and longitude ranges. The second part of the research improves the performance of the PSOs runtime, using OpenMP with CPU threading to parallelize the evaluation of the PSO by particle. Lastly, this implementation uses CPU multithreading with 4 threads to improve the performance of the PSO by 42% - 51% in comparison to running the PSO without CPU multithreading. The contributions provided allow for the PSO project to be more realistically simulate its use in the Electronic Warfare (EW) space, adding additional CPU multithreading implementation for further performance improvements.

1. INTRODUCTION

1.1 PSO Background

Particle Swarm Optimization is a bio-inspired computational tool originally developed by Russell Eberhart and James Kennedy in 1995 [3]. PSO has two main roots of methodology: artificial life (A-life) and evolutionary computation. The artificial life component is focused on swarm theory; swarm theory attempts to determine the actions of collective behavior [4]. PSO's inspiration from swarm theory was mainly focused on determining movement of flocking birds or schooling fish. Evolutionary computation is sharing information among individuals to achieve some solution. Originally PSO was inspired by swarm simulation, although today it is used in a variety of optimization problems. PSO has been widely applied since its origin in 1995, solving such problems as: rule classification in data mining [5], bi-level linear programming problems [6], Railroad domain optimization [7], Asset allocation [8], and many more.

1.2 Project Background and Previous Work

This research is a continuation on a project worked on for Expeditionary Electronic Warfare Division, Spectrum Warfare System Department, at Naval Surface Warfare Center (NSWC) Crane; it is also a continuation from the work done by Joshua Reynolds [8], Jonah Crespo [2], and Paul Witcher [9]. The main idea for the research was to use PSO to optimize placement of assets in terms to 3D space and frequency. The optimization problem needs to be solved in near-real-time, with constantly updating complex naval battlefield conditions. Work done in [8] developed the integration of the PSO algorithm with the problem of asset allocation in the Electronic Warfare Environment; this worked developed the fitness function to

use, the majority of the 2D GUI development, and analysis of asset allocation using PSO. The work done in [2] introduced solving the PSO in 3D solution space, early integration of topological constraints, and human-in-the-swarm integration with 2D user input of moving keep-away boundaries; though it did not have visualization for the 3D solution space. The work done in [9] developed the PSO with the addition of real-time movement simulation of assets and transmitters in 3D space, the work also tracked simulated assets to the PSO solutions using the Hungarian algorithm to simulate a more realistic Electronic Warfare environment.

1.3 Project Concurrent Work

In this project there are two other students that work on the project from myself. These students work on the same project which is consistently merged using GitHub as source control. These students are Mr. Md Saiful Islam and Mr. William Boler. Md Saiful work is related to doing testing analysis of the project from the constant updates to the project as well as comparing the performance to the most current version of the projects PSO asset allocation against the Genetic Algorithm to solve the asset allocation problem. William Boler's work can be found in his nearly published thesis [1]. The work done in [1] includes refactoring and adding code so that the entire large code base becomes more Object Oriented structure rather than the procedural and function base that it was before, as well as adding the Human-in-the-Swarm construct of Pheromones, which act as movable attraction or repelling zones or beacon that the user can add to the 3D solution space. The work done in [1] also includes implementing solving for assets antenna direction in the 3D space based on radiation patterns of antennas and implementation of a Meta-PSO in that he uses PSO to solve for good weight to give to the fitness function.

1.4 PSO Algorithm for Asset Allocation Background

Work done on using the PSO for asset allocation can be seen in [8]. This section gives a brief summary of using the PSO for asset allocation with the current state of the project, as well as the work done that impacted the PSO algorithm.

1.4.1 Particles and Neighbors

In PSO, many particles search the solution space. After each generation of the Particle Swarm Optimization, the fitness is evaluated for each particle, and a particle and its neighborhood best fitness is chosen each generation. This fitness function and the neighborhood best drives the particles toward a solution in multi-dimensional space. In the Asset Allocation problem, this solution space consists of: 3D space, frequency, antenna azimuth, and elevation orientation. Therefore, each particle searches this solution space until converged onto an optimal solution.

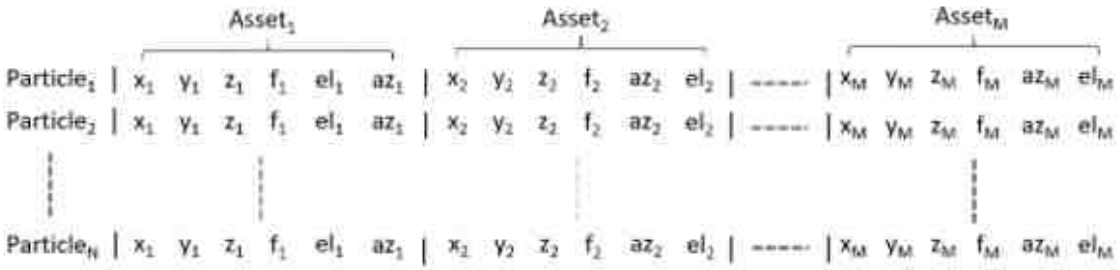

Fig. 1.1. Representation of how the particle is representing the data corresponding to the Assets.

As seen in Figure 1.1, each particle (row) is a possible solution. Each particle has 3D location, frequency, with antenna azimuth and elevation orientation additions regarding directional antennas for each asset. At the end of a generation, each particle has a personal best x_{pb}, which is the best solution that some particle has found, and

a neighborhood best x_{nb}, which is the best solutions that its neighbor have found. The neighbors of some particle i, are determined by the particle array in memory in a range defined by the number of neighbors for the particle. These particles are first initialized randomly throughout the solution space, then they are updated based on the following two equations from [3]:

$$v_{i+1} = v_i I + C_1 Rand()_1(x_{pb} - x_i) + C_2 Rand()_2(x_{nb} + x_i) \tag{1.1}$$

$$x_{i+1} = x_i + v_{i+1} \tag{1.2}$$

In equation 1.2, x_{i+1} is the future position (or frequency) for each element of each particle in the solution space. It is updated from the previous position (or frequency) and v_{i+1}, the future velocity. In equation1.1, v_{i+1} is computed by first Iv_i, where I is in Inertia and v_i is the current velocity. C_1 and C_2 are constants, $Rand()$ is a random generator. We use a random generator to give the algorithm randomness so that it is less prone to getting stuck in local solutions and so that it more robustly searches the solution space. $(x_{pb} - x_i)$ is the difference between the current position and the best position found so far, so that the particle will be given a larger velocity when it is further away from the current personal best. $(x_{nb} - x_i)$ is the difference between the current position and the best position found between all the particle i's neighbors. This will give a larger velocity when the current position is farther away from the best neighbors position. These 3 components are summed together so that the particles eventually converge onto a solution.

1.4.2 Fitness

In order to determine how good a particular solution is, it's important to have a way to measure it. Our problem is that of asset allocation, the environment is in Electronic Warfare (EW). The goal is so that the assets are placed in optimal location for communication or jamming. Each asset is assigned to certain transceivers

that are placed throughout the EW environment based on each asset corresponding bandwidth. The power from these assignments are checked to make sure that the feed-horn is not overloaded. Each assets' assignments are based on the assets frequency bandwidth. This assignment is important to consider for the components that make-up the fitness. In order so that placement is optimal the ongoing research done by [8] and [1] has constructed the following fitness function over the course of the research project:

$$Fitness = \alpha power + \beta priority + \gamma spread + \lambda distance + \delta pheromones \tag{1.3}$$

Component	Description
Power	The power of signals assigned to each asset
Priority	The Importance of the signals assigned to each asset
Distance	The Centroid distance between the signals and the asset
Spread	The physical distance between each of the assets
Pheromone	Zones/Beacons in the EW space

Fig. 1.2. This Table describes each of the components of the fitness function.

In equation 1.3, the Greek coefficients are the weights associated to determine a 'good' solution. Figure 1.2 shows a description of each component using in the fitness function.

1.4.3 Process and Generations

The PSO algorithm is run iteratively, until the solution's fitness is considered to be optimal. Each iteration of the PSO is call a 'generation'. The Overall process used for PSO asset allocation is as follows:

- Fly - update particles with velocity and position equations

- Evaluate - Determine fitness of each particle
- Update - update Particle ad neighbor particles
- Terminate - end PSO based on fitness slope/window size/max generation

This process is run over and over until the termination conditions are met. These are: fitness slope, window size, and max generation. Fitness slope is the measurement of the best slope from each generation. Window size is the number of times the PSO needs to run before ending. Max Generation is a limit used in case the termination process takes too long. Figure 1.3 is an example of running the PSO and showing the fitness measurements among runs.

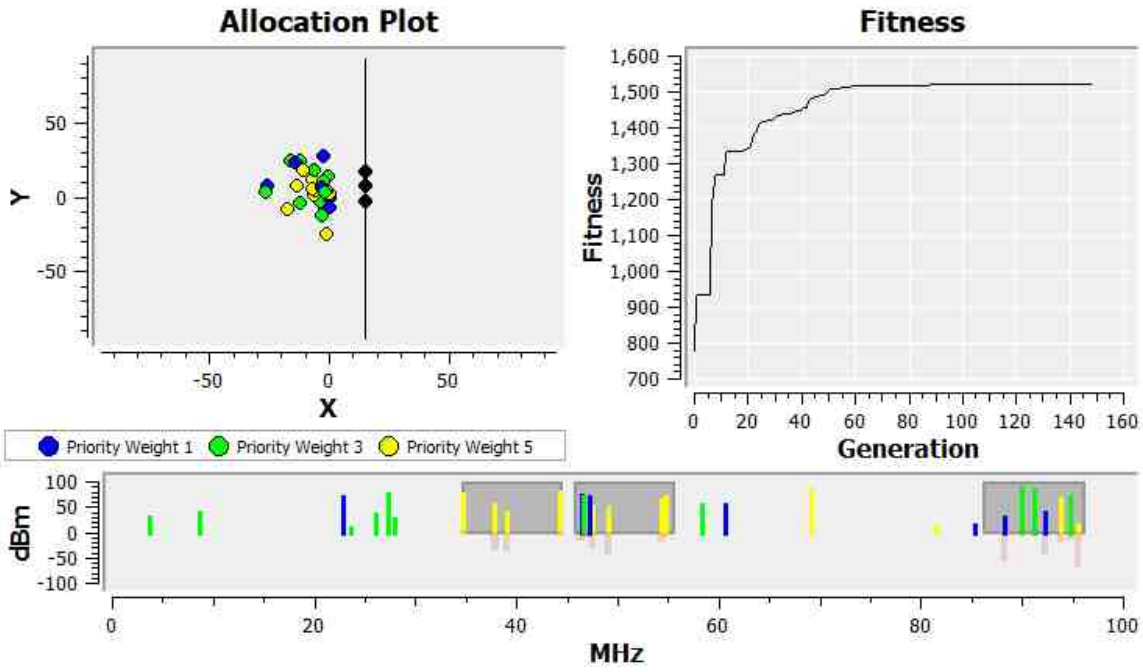

Fig. 1.3. A 2D representation of an optimized solution using the PSO algorithm. Allocation Plot show the 2D placement of the assets and transmitters. Fitness graph shows the best fitness measures between generations. The lower graph shows frequency space and the signal assignments between the assets and transmitters.

1.5 Literature Review

1.5.1 PSO Visualization Review

Visualizing the PSO algorithm is done to better understand the PSO. Two sources were found that closely represents similar visualizations of the PSO, which are the paper [10] and the video [11]. In [10] the authors use a 3D histogram visualization, and in [11] the authors visualized the particles convergence. In this research we visualize the final output, the solutions of the PSO.

In the work done by [10] visualizes the particles' probability distribution by using histograms to display the distribution of a -5000 to 5000 unit 2D space with 10,000

particles. The work used histogram visualization which displayed the particles' probability distribution in 3D; even though the solution space for the particles is in 2D. The visualization done in [10] was not to display solution as in this research, but to analyze the weaknesses of the PSO and fine tune the algorithm for better performance. Visualization done in this research displays just the solution of the PSO rather than the particles and can show realistic terrain domains rather than a set flat plane. It is also much more clear where the solutions of the PSO in the environment in this research compared to the work done in [10].

Other visualizations can be seen in videos such as [11]. The video in [11] shows the particles location and movements in 3D using MATLAB, they move based on either a maximum or minimum of a function that is displayed in 3D as well. This video shows the nature of the particles in a 3D space. The work done in this research displays a more realistic environment with elevation and imagery, as well as more than 1 solution point and more components of the EW environment (pheromones, assets, etc.). This research also better integrates into the previous project software since it uses supporting software of QT Data Visualization since the previous project is done using QT, rather than outside software such as MATLAB.

1.5.2 Elevation and Imagery Extraction Review

Extraction of elevation data can be done using data sources such as those described in [12]. NRCS, or Natural Resources Conservation Service, requires high quality elevation data. They use the NGCE, or the National Geospatial Center of Excellence, which provides digital elevation data to NRCS staff and partners. The NRCS uses sources such as NED, or National Elevation Dataset, that comes in resolutions 3, 10, and 30 meters. The work done in this research uses ARCGIS to extract not just elevation data, but imagery as well. ARCGIS uses elevation sources from a wide variety of dataset to provide the highest resolution possible, so it will be equal or better than the elevation data provided by NED.

The author in [2] ended up using data from NASAs Shuttle Radar Topo-graphical Mission (SRTM). Although, we found that the resolution was too poor, and the project no longer collects or updates data. The ARCGIS extraction method has higher quality resolution data than the SRTM and is regularly updated. The extraction process used in this research is also more dynamic and provides more control over the data and resolution being extracted; whereas the data from SRTM was a set resolution and only certain intervals of ranges regarding latitude and longitude. In this research the method provided allows a user to extract from any given latitude and longitude entered; as well as provides resolution settings and extracts the corresponding imagery data with the elevation data.

1.5.3 Threading Review

It's not uncommon that applications and algorithms attempt to increase their performance by utilizing the growing field of parallelization. With multi-core CPU processors and hundreds core GPU's, it is not surprising to see growing utilization of parallelization. In [13], the authors created a parallel Particle Swarm Optimization (PSO) algorithm for bus voltage optimization. The parallel PSO strategy was to run separate PSO's in parallel to get a better solution, rather than decrease the runtime. The authors used 3 threads in parallel, meaning 3 PSO runs at a time. They took 1/3 of the best solutions from each PSO, and filled the other 2/3's randomly from the remaining population. They found that the parallel PSO implemented outperformed the normal PSO with higher fitness solutions and slightly faster convergence time at the cost of increased runtime. The work done in this research decreases the runtime and parallelizes a single PSO via threading between particles and keeping fitness performance the same.

Similar to how threading is used in this research, some other work has been done using GPU to increase speed of the PSO seen in [14]. This paper uses a variant of the PSO so that it can run fully in parallel, this is called Gaming Particle Swarm

Optimization (GPSO). GPSO uses multiple small swarms of particles that run independently, rather than a single PSO to solve asset allocation. The solution to be solved is similar in [14], which uses GPSO to solve the radio frequency resource allocation problem. An analysis was implemented using an Nvidia GTX 465 against a previously state-of-the-art AMD Phenom 3.4GHz quad-core CPU. The paper found that using the Nvidia GTX 465 was measured to be 5 times faster than that of the AMD Phenom when using the GPSO. Although, this was dependent on the number of particles used; the CPU was measured to be faster around 500 particles or fewer, where the GPU became faster when there were 500 particles or more. The 5 times speed increase was determined with 2,300 particles. In our implementation we use CPU threading with 4 threads, giving an increase of 42% - 51%. This research intention was to increase the performance of the PSO rather than implementing a variety of PSO like the GPSO to increase parallelization. This research also uses a relatively small number of particles, general 200, which would run faster on a CPU according to [14].

1.6 Contribution Summary

A current representation of the project is shown in Figure 1.4. This work shown is an accumulation of the work from this research, [1] and [9]. Where the previous work is shown in Figure 1.3 with work done by [8] and [2]. The addition from the previous work to the current work are as follows: 3D Visualization, Elevation and Imagery Integration from ARCGIS, CPU threading (thread GUI not displayed), asset tracking [9], transmitter movement [9], pheromone integration [1], pheromone display and interface [1], directional antennas [1], Meta PSO [1], Statistic Integration [1], Line-of-Sight integration [1]. Other improvements include general code improvements from this research and [1], and Object Oriented integration from this research and from [1].

The work done presented in this research is the following: 3D Data Visualization, Elevation and Imagery Extraction, and CPU Threading. Chapter 2 describes visualizing the 3D work done in [2] and improves upon it to add more realistic 3D PSO solution space environment; this visualization was also used to support the visualization used in the asset tracking and transmitter movement done in [9]. The work done in [2] had early extraction of elevation data into the PSO space, Chapter 3 describes the improvements of the extraction and integration of the elevation extraction from [2] with high resolution elevation as well as imagery extraction methods using ARCGIS which are more dynamic and flexible. Chapter 4 introduces the use of CPU threading in order to improve the runtime of the PSO using OpenMP as well as an analysis of the performance improvement that CPU threading provides. Lastly, Chapter 5 gives a summary and Chapter 6 gives recommendations for the future work.

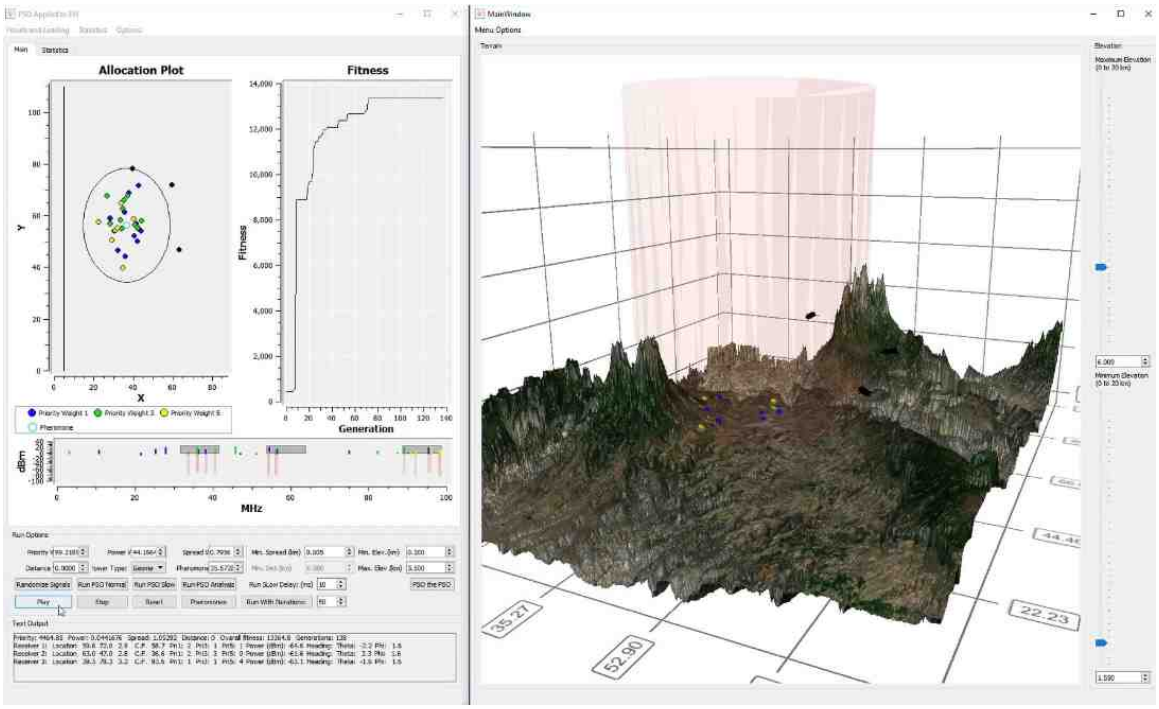

Fig. 1.4. The current applied project with contributions from this research and [1]. The right side shows a 3D visualization of the PSO, solutions, ARCGIS extracted elevation/imagery, pheromones, LOS implementation, and directional antennas. This current project implementation also integrated and improved the work done from [9] with real-time simulation of movement.

2. 3D VISUALIZATION

This section will cover how the project shows the output of the PSO algorithm in 3D.

2.1 Software

The software used to display information is done using Qt Data Visualization [15]; which is a Qt module used to visualize information in bar, scatter, or surface graphs. From Qt Data Visualization we use its following classes: QCustom3DItem, QCustom3DVolume, and Q3DSurface. These classes from Qt Data Visualization are used to display the graphics in the project.

2.2 Terrain

This section describes how the elevation information is loaded, accessed, and manipulated. As well as describes the functions of the class TerrainData, which deals with anything regarding the terrain information.

2.2.1 TerrainData

The elevation data needs wide access and manipulated to be integrated into the project, so there needs to be something that handles this kind of terrain information. This project as well as contribution from [1] implemented a class called "TerrainData" to handle everything related to the elevation data, access, and storage. Such functions include: parsing elevation data from a CSV file, setting the dimensions of the terrain, looking up elevation data, and much more. The majority of the functions implemented can be seen in Figure 2.1.

Function	Description
loadTerrainFile	Parse elevation or load default (splash)
updateDimensionsFromTerrain	Set X,Y,Z domains from Lat/Long
Flip Axis	Flips Index Axis
getLengths	Lengths of terrain vector from plot
LookUpElevation(x Km, y Km)	Returns plot z height from (x,y) in Km
getHeightFromIndex(x,y)	Returns height in Km from plot (x,y)
getCoordIndex(x Km, y Km)	Returns plots scaled (x,y)
getMin/Max/Center	Gets X,Y,Z information
updateWithLatLongs(l)	Update plot dimensions with new Lat/Long info
getResolution	Gets distances resolution for (x,y)

Fig. 2.1. This table shows the descriptions of the majority of the functions implemented in the TerrainData Class.

2.2.2 Loading Terrain and 3D Visualization Access GUI

In order to integrate loading terrain files and 3D Visualization, the menu is in Figure 2.2 was added, Where 'Loading Terrain File' gives the user access to load in files and '3D Surface' opens a new window that displays the PSO solution in 3D.

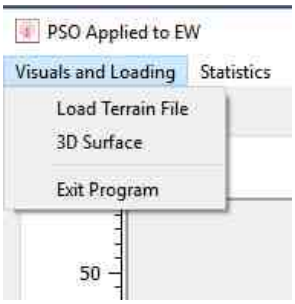

Fig. 2.2. Dropdown menu showing access to loading terrain interface and 3D Data Visualization interface.

2.2.3 Loading Terrain Files GUI

The work done in [2] used hard-coded file paths into the source code to store elevation data. In order so that the application can be more dynamic and uses more than 1 terrain file at a time, a terrain loading GUI was implemented so that the information about the terrain can be edited and updated without closing the program. This GUI integration also is much more intuitive than the hard-coded method in [2]. This implementation can be seen in Figure 2.3

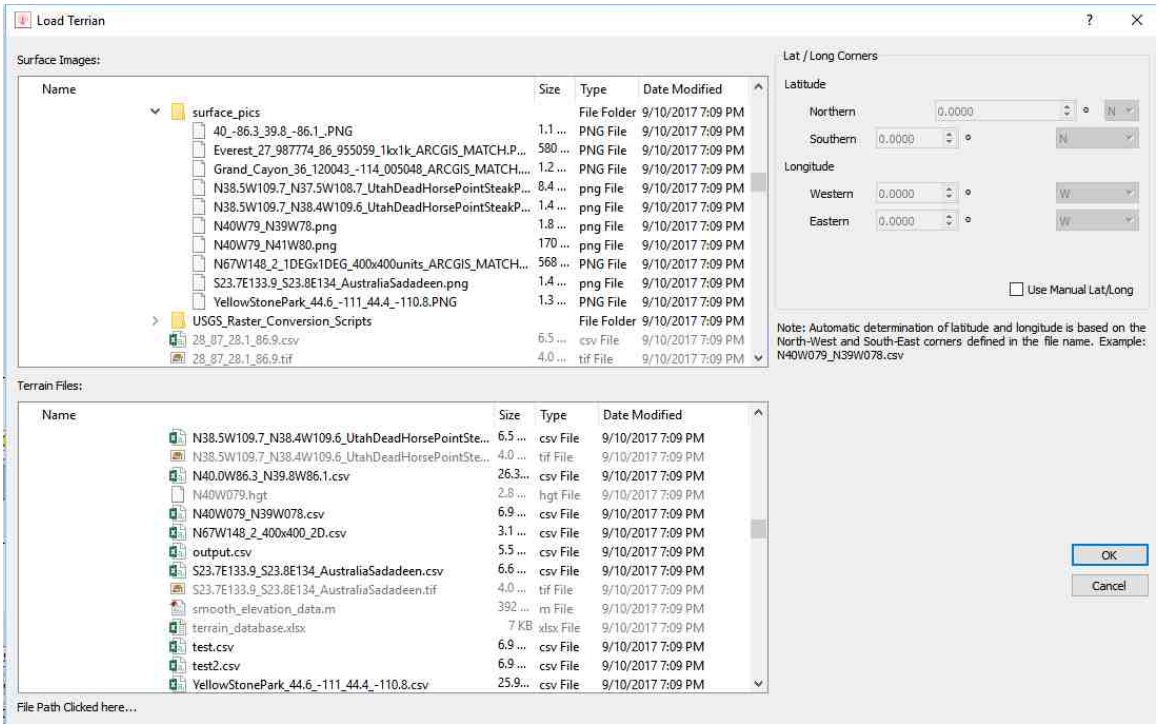

Fig. 2.3. The Interface so that a user can select image and terrain files. As well as Latitude and Longitude information.

From this GUI the user can select an image file in the 'Surface Images' top file menu, and the elevation CSV file in the 'Terrain Files' bottom file menu. Only the supported file formats are allowed for each file menu, such as PNG or JPEG for the top menu and CSV for the bottom, unsupported file types are not able to be selected.

The selected files for terrain can also be read if the file is named in a particular format, the file format is 'lat1_long1_lat2_long2_name'; this takes in the latitude and longitude information from the named file, for example N40_W49_N39_W48_Place. The User can also manually user latitude and longitude information with the given menu in the top right.

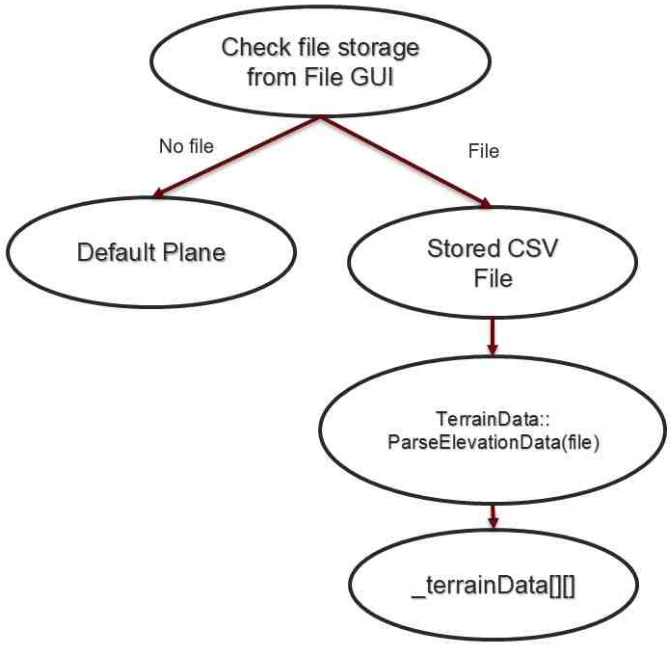

Fig. 2.4. This flow chart shows the functionality of the loadTerrainFile function in TerrainData.

2.2.4 Loading and Storage of Terrain Files

In order for elevation data to be displayed, it needs to be stored inside the program to be referenced. The work done in [2] stored elevation data in a 2D array 1201 by 1201. The work done in this research improves upon the work done in [2] by making the storage a 2D vector for any sized input. To populate the 2D vector without hard coding elevation data in the program, reading from an integrated file explorer GUI interface was used. The function in TerrainData called "loadTerrainFile" was imple-

mented as an improved version to [2] that reads files in a CSV (Comma Separated Values) file format or loads a default flat plane into the 2D vector that contains the elevation data; this process is shown in 2.4, once the file reading is completed the 2D vector will contain the elevation data that can be used in the application. The elevation data extracted from the user defined files are in kilometers.

2.2.5 Elevation Initialization

If no file is loaded and the '3D Surface' window is selected, the surface is loaded with a 1000x1000 'unit' surface then transformed to match the original domains of 200kmx200km; it also isn't loaded with a corresponding image file like the elevation files. Figure 2.5 shows a comparison between the 2D Allocation plot and the 3D Surface Window's 3D plot; notice that the ranges are matched between the two:

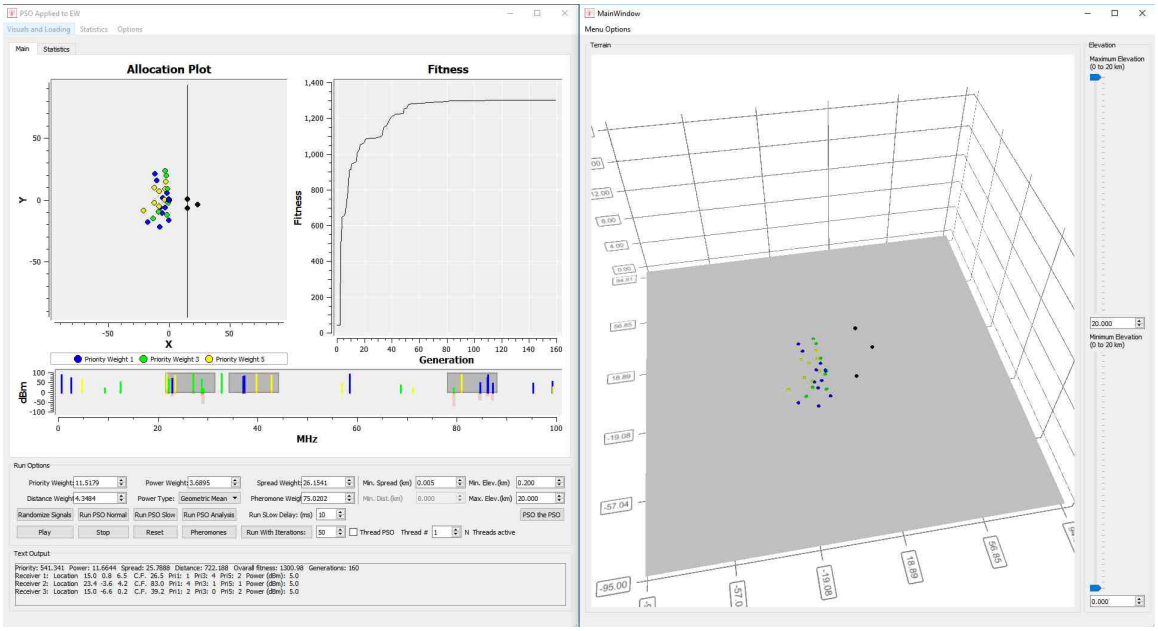

Fig. 2.5. Comparing the previous 2D display with the implemented 3D display when the surface is initialized with no elevation or image data.

2.2.6 Latitude and Longitude Ranges

When loading in a new terrain file, it's necessary to create new ranges in the x and y dimensions. This is implementation in TerrainData as "updateDimensionsFromTerrain". The terrain data is represented as a flat square surface on the earth where the latitude and longitude are the corresponding boundaries. When calculating distances between the latitude and longitude points, the spherical nature of the Earth is taken into account and the arc-distance is calculated, as seen in 2.1; this is then projected onto a flat plane. To make the data ranges into a square, the ranges are calculated based on the average of the two measured arc-distances from the latitude and longitude points $((Latitude_1, Longitude_1), (Latitude_2, Longitude_2))$.

$$Arcdistance = R\sqrt[2]{(\phi)^2 + (cos(\phi_m)\Lambda)^2} \tag{2.1}$$

$$R = 6,371,000meters(Radius - of - Earth) \tag{2.2}$$

$$\phi = Latitude_2 - Latitude_1 \tag{2.3}$$

$$\phi_m = (Latitude_2 + Latitude_1)/2 \tag{2.4}$$

$$\Lambda = Longitude_2 - Longitude_1 \tag{2.5}$$

Figure 2.6 shows an example of the ranges being calculated and displayed given a terrain file, the coordinates are ((N3.85,W109.7), (N37.5,W108.7)).

2.2.7 Vector to Terrain Dimension Scaling

As previously mentioned, elevation data is extracted from a CSV file then placed into a 2D vector. It's important to consider the source elevation files resolution. Chapter 3 describes the extraction process in detail, but when we extract information, the user enters latitude and longitude coordinates and an image size; such as 1000 by 1000 pixels. Although there is no corresponding relationship between the coordinates and the pixel resolution of the extracted image. The image size determines the size of the 2D vector, so if an image is 1000 by 1000, the 2D vector becomes 1000 by 1000.

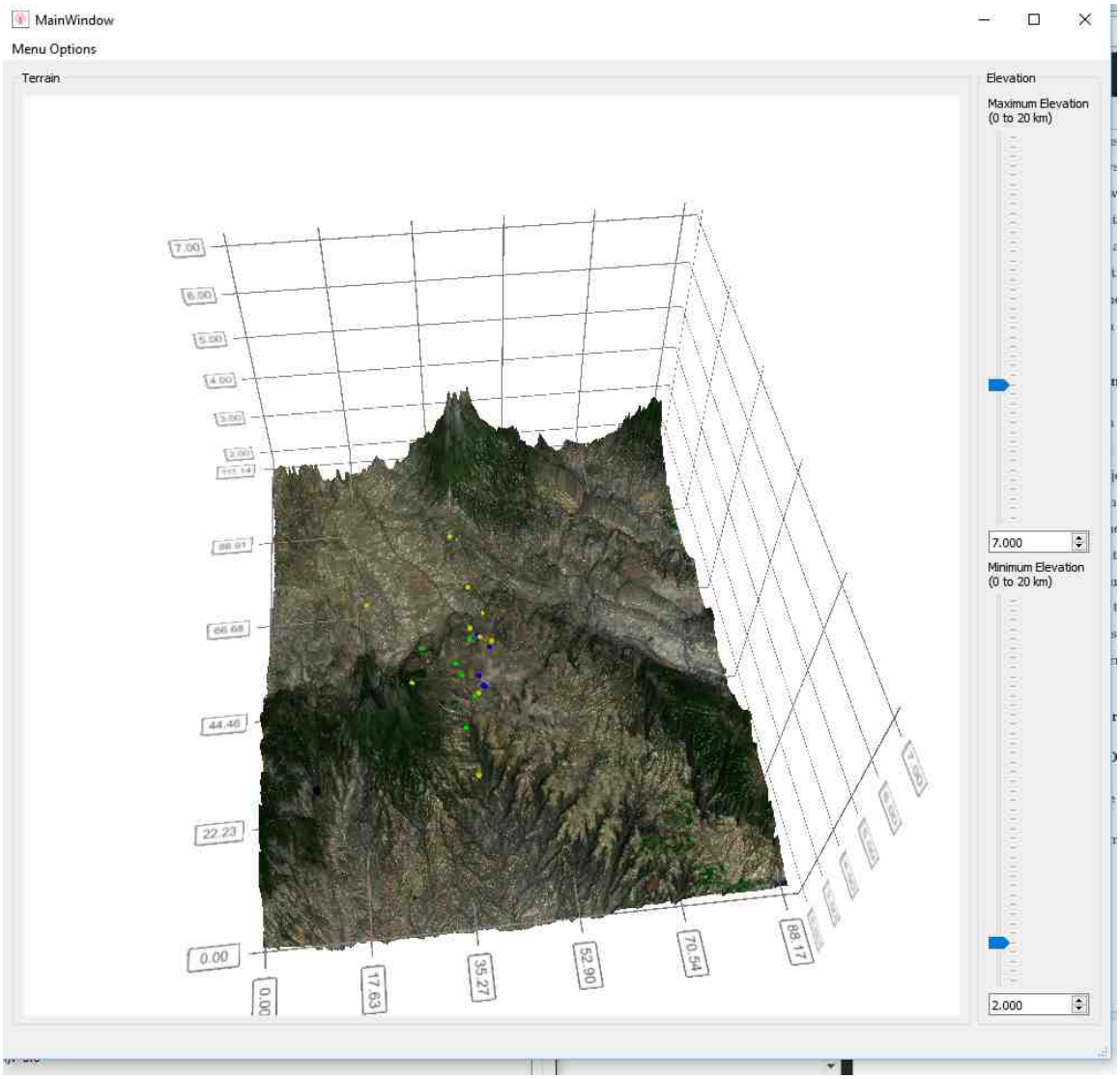

Fig. 2.6. Demonstration of how the domain ranges change based on the given latitude and longitude information.

We call the dimensional space of the 2D vector the vector space, and the dimensional space calculated using the arc-distance formula the terrain space. In order to handle any coordinates and vector size, we determine the step size of the terrain space in relationship to the vector space.

Equation 2.6 shows the equation used to scale from the Terrain dimension, stored in kilometers to the vector dimension, which is unitless. Where $Vector(x)$ is the location of the x value for the stored 2D vector, $x_{terrain}$ is the x location in the terrain space, $xmin_{terrain}$ is the minimum of the x dimension in the terrain space, $xmax_{terrain}$ is the maximum of the x dimension in the terrain space, and $xLength_{vector}$ is the x length of the 2D vector.

$$Vector(x) = (x_{terrain} - xmin_{terrain})(xLength_{vector})/(xmax_{terrain} - xmin_{terrain}) \quad (2.6)$$

We call this scaling between the vector space and the Terrain space. This was implemented for practicality and flexibility purposes, since the 3D Visualization is shown in a 3DSurfaceGraph; it can only display a finite resolution due to the limitations of the visualization. For example, a very large area, such as 1,000km by 1,000km, with a fine resolution 10m by 10m, would take up 10,000,000,000 points that would need to be displayed on the surface graph. The displayed resolution in the 3DSurfaceGraph cannot display or zoom in further enough to represent such data, and it's highly likely that the computer used would not be able to support such a large 3D surface graph due to limited RAM memory size. Thus, during the extraction, a resolution size is chosen so that the data is still displayed in fine detail without taking too much resource cost. The resolution used in most of the extractions is 1,000 by 1,000, this resolution was found to not take too much memory and still display the data in fine detail; although any resolution size can technically be chosen.

2.2.8 Looking up Elevation Data

Since the elevation is stored in the vector space, and the calculations for the PSO are done in the terrain space; it's necessary to correspond any (x,y) in the terrain space to same elevation element stored in the vector space. We implement this as "LookupElevationData" in TerrainData. This function first scales the (x,y) coordinates from the terrain space to the vector space, implemented as "getCoordIndex" in TerrainData, then we return the stored value from the calculated coordinates, implemented as "getHeightFromIndex" in TerrainData. This process is shown in Figure 2.7.

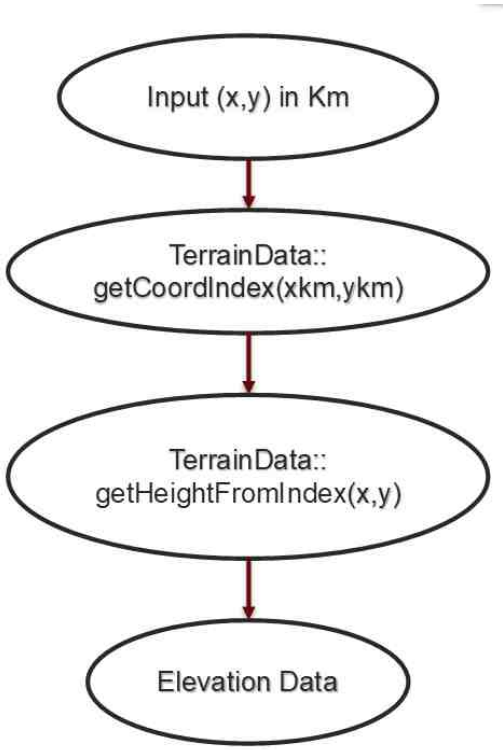

Fig. 2.7. A flow chart showing the process of LookupElevationData in TerrainData.

2.3 3D Visualization

This section describes how the PSO and the EW space is rendered and displayed using 3D Visualization from QT, as well as the interface for the 3D Visualization Window.

2.3.1 3DSurfaceWindow

Similar to TerrainData, a class called "3DSurfaceWindow" was implemented from the contributions of this project and [1]. 3DSurfaceWindow deals with anything regarding the 3DSurfaceGraph as well as the updates to the window that contains the 3D Visualization information. Such functions include: building the 3DSurfaceGraph from the 2D vector, setting the Theme and Lighting of the Visualization, updating elevation axis limits, and more. The majority of the functions implemented can be seen in Figure 2.1.

Function	Description
getsurfaceFormat	Sets up formatting for Surface Graph
rebuildSurfaceGraph	Rebuilds SurfaceGraph from terraindata[][]
setSurfaceTheme	Color, theme, and lighting setup for SurfaceGraph
rebuildSurfaceGraphObjects	Rebuilds all other objects besides surface plot
updateAxisLimts	Updates internal axis limits used in PSO
updateElevationLimts	Updates plot displayed Z axis limits

Fig. 2.8. This table shows the descriptions of the majority of the functions implemented in the 3DSurfaceWindow Class.

2.3.2 3DSurfaceWindow

In order to simulate visualization of the stored elevation data, it needs to be represented as a 3DSurfaceGraph from Qt Data Visualization. In Figure 2.9 shows an example from the Qt Data Visualization documentation [16] of 3DSurfaceGraph implementation.

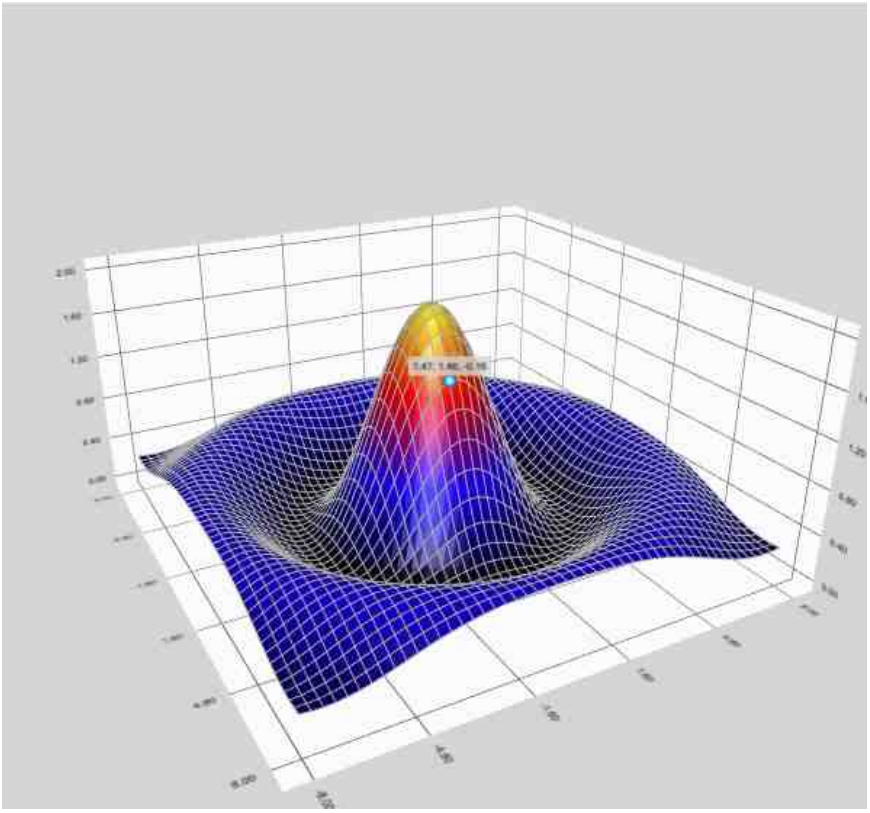

Fig. 2.9. An example from [16] displaying a 3D Surface Graph using a sinusoidal function.

The creation of the 3DSurfaceGraph is done in rebuildSurfaceGraph in the 3DSurfaceWindow class. This function creates a 3D series from the 2D vector data, although when plotting the information using buildSurfaceDataArray from 3DSurfaceWindow, not shown in 2.8, it scales from the vector space to the Terrain space so that the 3DSurfaceGraph's information is seen in the kilometer terrain space. This is how the

plotting was done in the range example shown in 2.6. A flow chart of rebuildSurfaceGraph is shown in Figure 2.10.

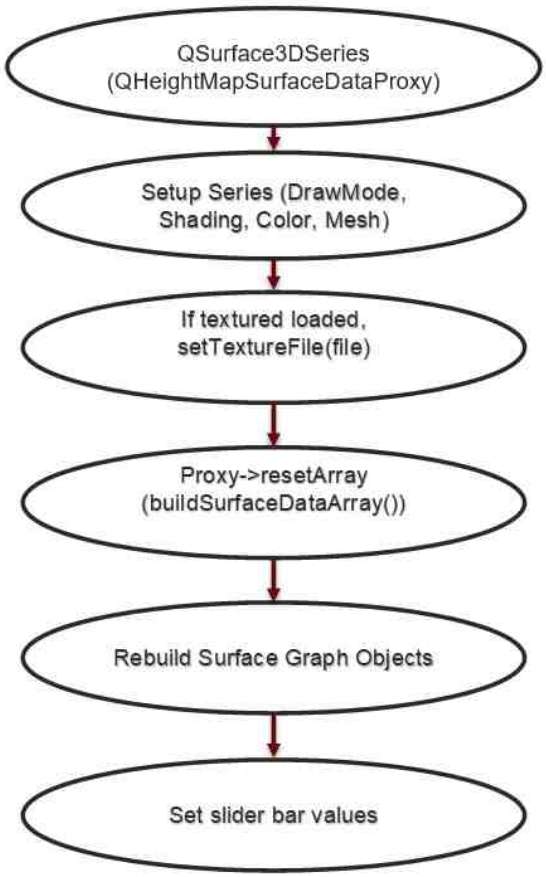

Fig. 2.10. A flow chart showing the process of the rebuildSurfaceGraph function in 3DSurfaceWindow.

An example of the results from rebuildSurfaceGraph can be seen in Figure 2.11.

2.3.3 Custom 3D Objects

In order to add more information to the surface graph, a class called "3DCustomController" was implemented from the contributions from this work and [1]. This class handles all the other 3D visualizations that are not the Surface Graph. This includes: targets, assets, solutions and pheromones. In order to be separated from the 3DSur-

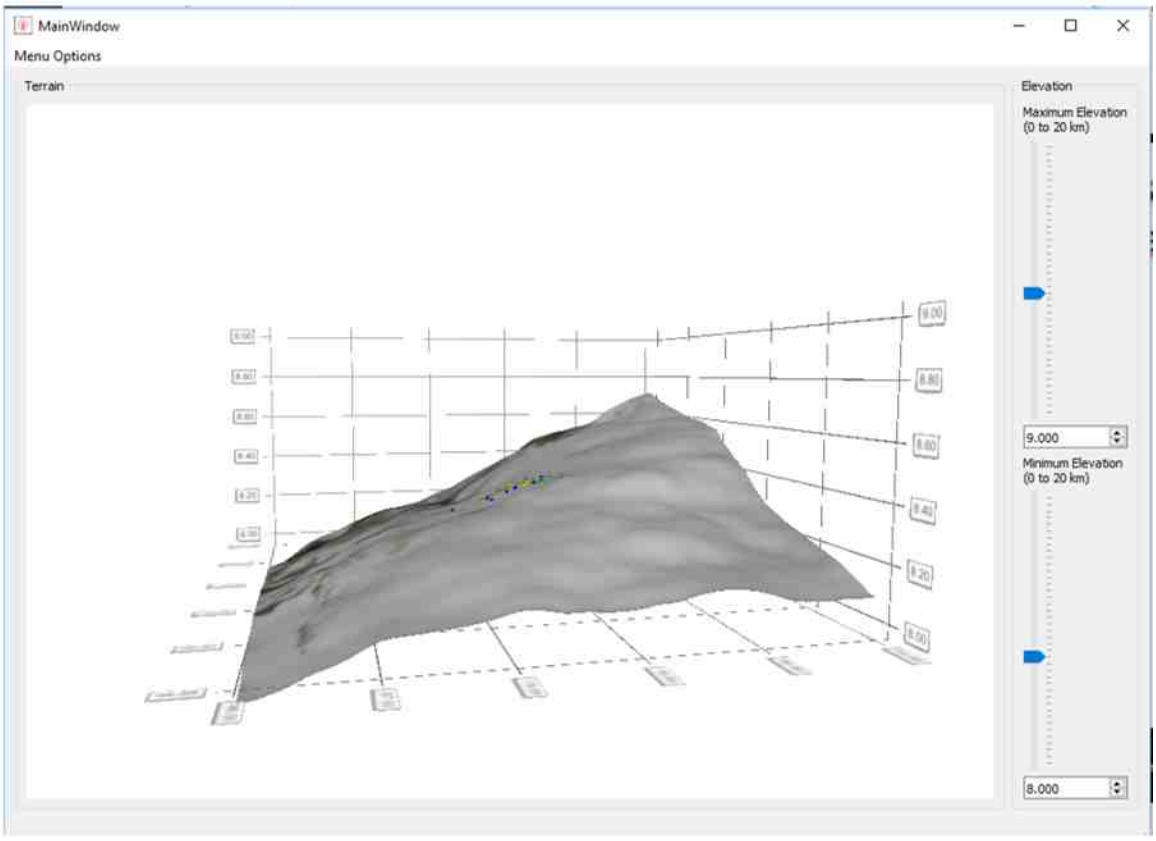

Fig. 2.11. A 3DSurfaceGraph output example from rebuildSurfaceGraph in 3DSurfaceWindow.

faceGraph, these objects are added to the 3DSurfaceGraph as QCustom3DItems. These objects are used to display, render, mesh, rotate, and texturing the signals, solutions, assets, and pheromones. This allows the visualization to handle both surface and other data. Figure 2.12 shows the majority of the functions implemented by 3DCustomController.

In order to display the objects, 3DCustomController uses a function called "rebuildSurfaceGraphObjects". This function renders, sets the position, and adds each object to the 3D Visualization. A flow chart showing the process is shown in Figure 2.13.

Function	Description
initialize3DRenders	Loads Textures and Meshs
getNew_Object	Makes new _ QCustom3DItem Object, sets mesh, texture, scaling, position, and rotation.
rebuildSurfaceGraphObjects	Builds all QCustom3DItems
updateSurfaceGraphObjects	Updates all QCustom3DItems

Fig. 2.12. This table shows the descriptions of the majority of the functions implemented in the 3DCustomController Class.

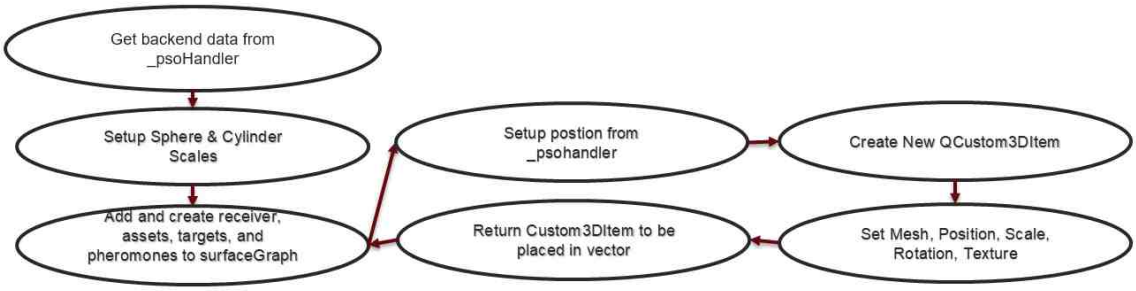

Fig. 2.13. A flow chart showing the process of rebuildSurfaceGraphObjects in 3DCustomController.

2.4 3D Surface Window Interface

The 3D data display window gives the users different kinds of interface options so that they can interact with the display to allow for better visualization. Basic features are integrated by using the 3D Surface Graph, such as zooming, and click and dragging the camera to view the data at different angles. Timers and activators are integrated into the 3D window so that the display can be updated with new information when the conditions are activated; such as updating every second when the users hit the 'Play' button. The movements and tracking from the 'Play' interface are

from contributions from [9]; though the visualization and refreshing constructs were developed by this research and [1]. When the terrain is displayed, it will look different based on what is set for the z-axis limit display boundary. In order to give the users a better representation of the z-domain in the display, sliders and input boxes were implemented to give the user control over the z-axis ranges that are displayed. The activators sense whenever there is a change in the slider or input box values and updates the graph accordingly. Figure 2.14 show a loaded terrain file without modifying the min-max elevation sliders; Figure 2.15 shows the same terrain information with the sliders modified. The user is also able to move pheromones during PSO computation from contributions from [1], as well as other interface options. This interface controls all interactions from the user and the 3D display.

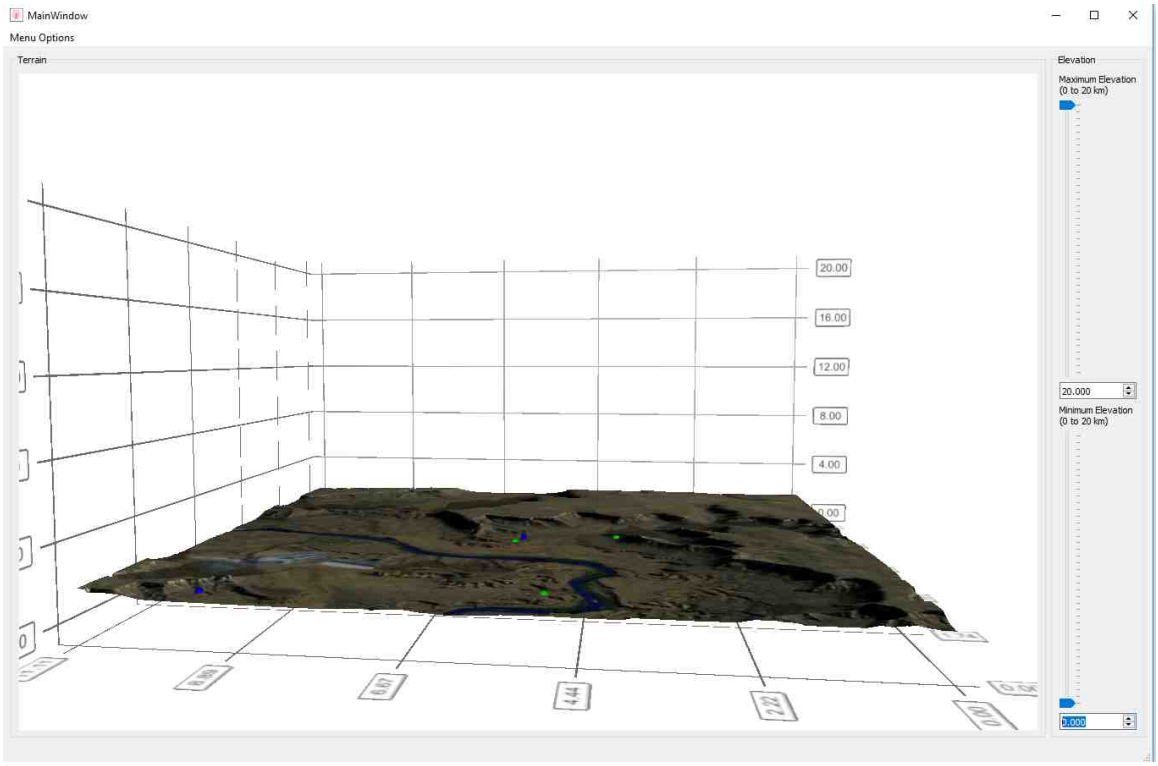

Fig. 2.14. Terrain information loaded without changes the min-max sliders. The vertical range displayed is from 0km to 20km.

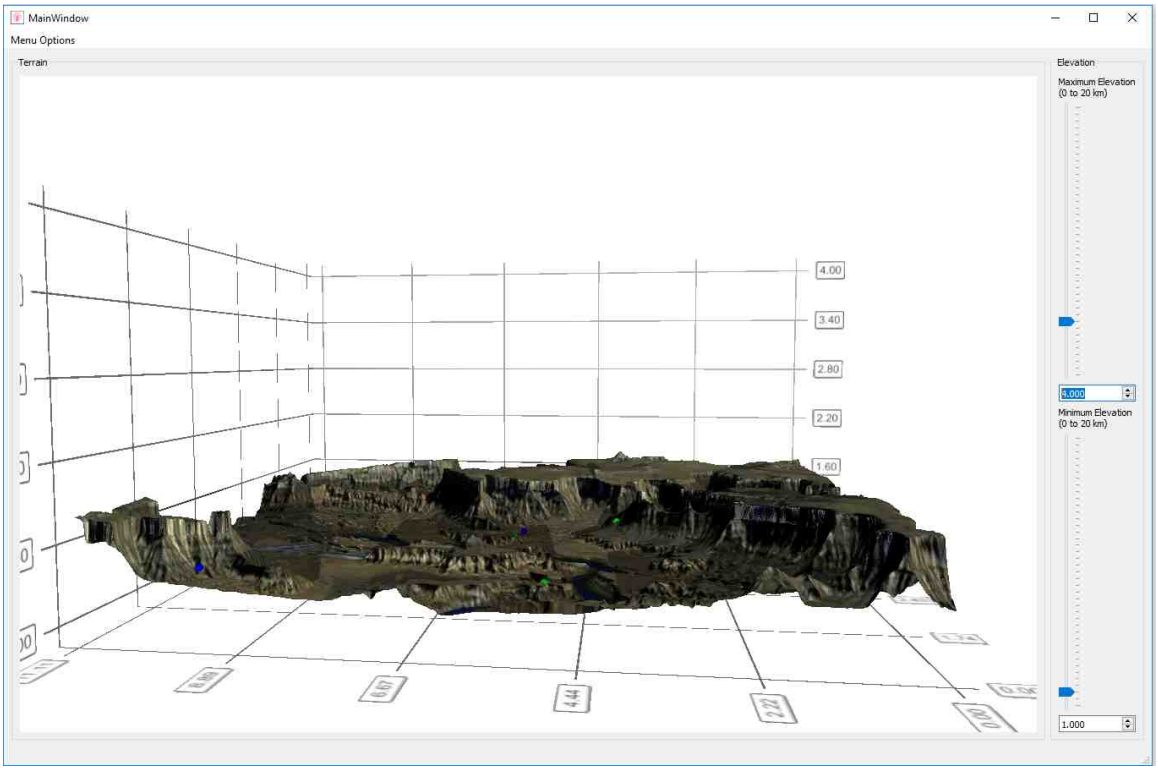

Fig. 2.15. Terrain information loaded with the min-max sliders change to more appropriate values. The vertical range displayed is from 1km to 4km.

3. ELEVATION AND IMAGING

This section will cover where and how the elevation and image data was extracted, as well as how it is converted into a format usable by the current project.

3.1 ARCGIS

We extract our elevation and imagery data from ARCGIS. ARCGIS is a geographic information system used for management, analysis, and displaying geographic data. The research team collaborated with ARCGIS in order to get familiar with what ARCGIS can offer as well as ARCGIS access and software. Working with ARCGIS, various methodology for extraction were implemented.

3.2 Earth Projections

Since the Earth is a sphere, or more accurately an oblate spheroid, it is more difficult to project proper elevation data onto a flat surface; such as what is represented with the 3D Surface Graph. To display the data it's necessary to project the data of the Earth's surface onto a plane. Although when extracting data, it will be different based on how the Earth's surface was projected. There are many types of data projections when it comes to recording data from the earth surface; each one with its set of advantages and disadvantages.

Figure 3.1 is an example of a conformal projection from [17], this type of projection is used to keep accurate shapes over small areas. The projection type shown is called the Mercator projection, commonly used for navigation. It used to keep angular relationships correctly displayed. [17]

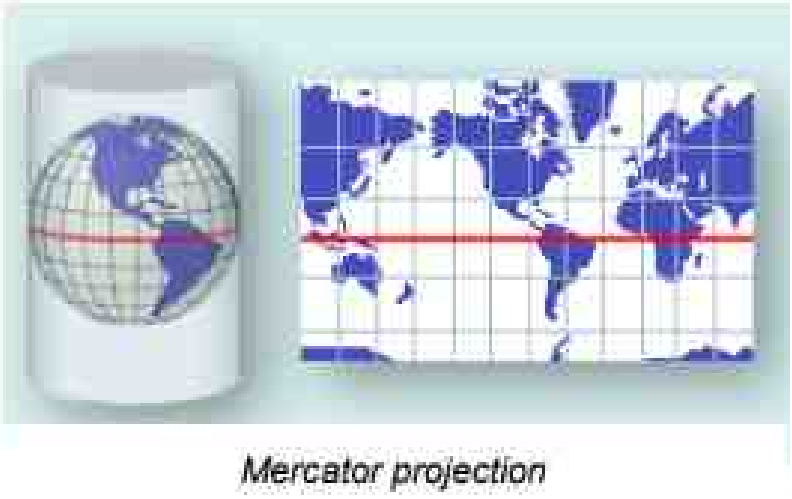

Fig. 3.1. Map projection using a Mercator projection. It projects by projecting the earth as a cylinder wrapped around the earth then flatted out.

Another type of projections are conical projection from [17], shown in Figure 3.2, which uses a cone shape to project onto the Earth. This projection are have the least amount of distortion where the cone interests, useful for upper or lower hemispheres.

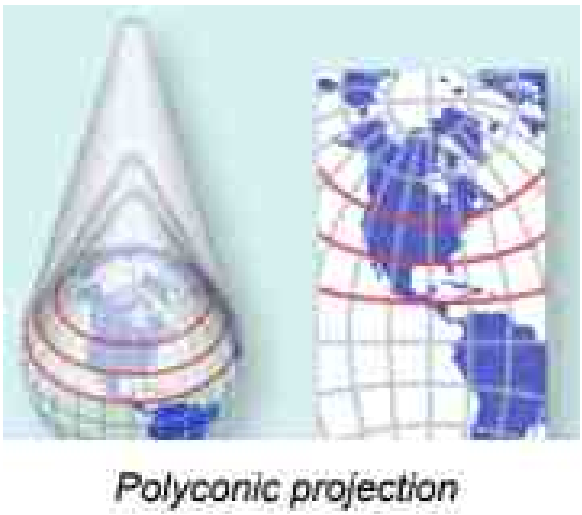

Fig. 3.2. Map projection using a Polyconic projection. It projects by projecting the earth as a cone wrapped around the earth then flatted out.

There are many more kinds of map projections that are used to display the Earth data, each one should be conditioned based on which areas on the earth want to be displayed with minimal distortion. ARCGIS can extract from any kind of projection, so it can be used for any kind of projection considered. The projection that is used in this research is the Mercator projection; due to its property of keeping shapes accurate over small areas. We assume that the 3D Surface Graph will display an area that is somewhat limited within a few degrees, and not too massively large that the Earth sphere shape will effect it too strongly. Although the distortion is somewhat minimized when the ranges are measured using arc-distances as explained in 2.4.6, where latitude and longitude are shown in Figure 2.6. Since the average of the arc-distances is calculated, the distortion will become large based on the difference in the latitude and longitude calculated arc-distances. When this is the case, then a different the projection would need to be picked based on the geographical location so that distortion is minimized.

3.3 Extraction Methods

ARCGIS has a variety of software and interfaces that can be used to extract geographic for a very large of problems that cover more than the elevation and imagery data extracted that is done in this project.
When extracted elevation and imagery data for the project, two main extraction methods are used:

- Fishnet Extraction
- ARCGIS API Image Extraction

Each one has advantages and disadvantages and will be further explained in the coming sections.

3.3.1 Fishnet Extraction

For the Fishnet extraction, software called ARCGIS for Desktop was provided to our research team by ARCGIS. After some training on the software, we developed a method to extract evaluation and imagery data using this software. This method uses ARCGIS software to extract elevation and imagery data.

3.3.2 Set-up

ARCGIS for Desktop uses the constructs of overlapping data based on what they call "layers". These layers can provide any kind of geographic information so that the software can be used to solve problem or analyze the geographic data. The data layer that is necessary to use is the Imagery data layer as well as the Terrain data layer. These layers needed to be created and then integrated in the software. Figure 3.3 shows an example of loading an elevation layer by connecting to ARCGIS and extracting the Terrain Layer from ARCGIS's servers. The layer is an image-based data layer that is represented in the Mercator Projection.

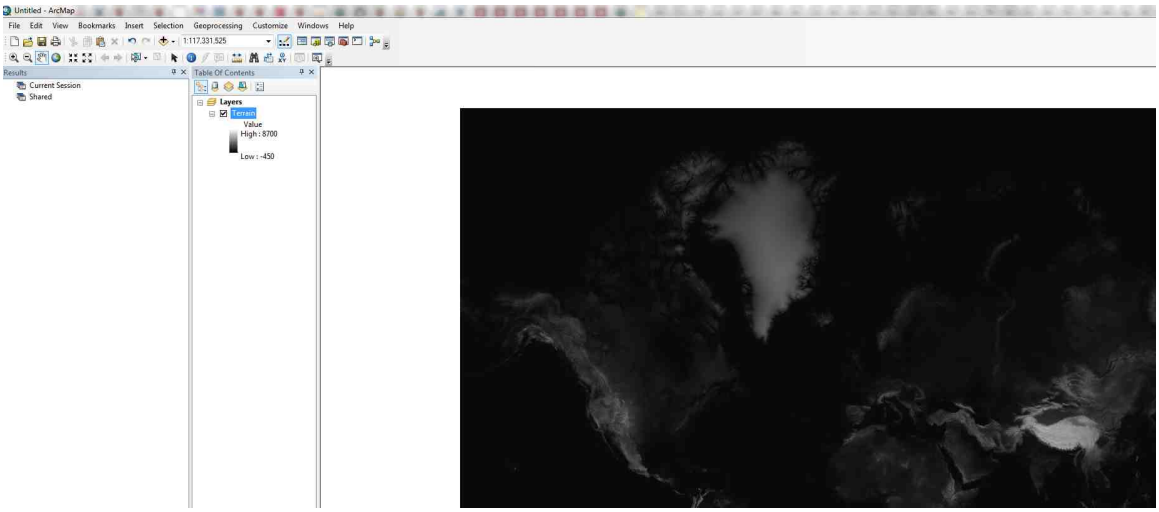

Fig. 3.3. Using ARCGIS for Desktop and loading in the Terrain data layer from ARCGIS servers.

3.3.3 Point-by-Point Extraction

Once the Terrain and image layers are imported into ARCGIS for Desktop, it necessary to read in the elevation values from the Terrain layer in order to extract elevation data from the layer to be able to use with the project. A simple methodology to do this is to place points into the system and compare the points location with the Terrain Layers position and extract the Terrain Layers elevation data at that point; ARCGIS calls this process 'Identify'. Although this would only give one point at a time and be time consuming to create all the point individually. The next section discuses a more usable version of this point by point extraction method.

3.3.4 Fishnet Implementation

Rather than getting elevation information point-by-point, we use what ARCGIS calls a 'fishnet' to isolate a section of the Terrain data layer to be extracted. This feature allows the user to create a section anywhere given a range of area, then it creates 2D array feature of points based on the given input of the fishnet creation inputs. A creation example for fishnet is shown in Figure 3.4.

Once the appropriate inputs are given to the Fishnet creation interface is shown in 3.4, a series of points are created as a layer into ARCGIS for desktop and are displayed in the corresponding area. Although these points have no meaning in relation to the Terrain data layer; in order to read in the Terrain layers information at these points, the layers need to be compared. To do this we use a tool in ARCGIS for Desktop call 'Points to Values' that given a series of points returns the values of some layer at those points. In our case, we compare the points created by the fishnet and the Terrain data layer as our values. The fishnet along with its extracted values using 'Points to Values' with the Terrain layer is shown in Figure 3.5.

Fig. 3.4. The Interface for creating a fishnet in ARCGIS for Desktop.

3.3.5 Imagery Data

To extract the imagery data using ARCGIS for Desktop we add something called a 'basemap' in the software. This creates an overlaying image data layer based on some other layer. So we access ARCGIS's Imagery layer and integrate into the geographic representation of the Terrain layer. This looks at the coordinate and projection system that the Terrain layer uses to overlay the Imagery layer correctly corresponding to the elevation data from the Terrain Layer. This allows us to simply cut-out the image from the Imagery data based on where the Fishnet is created.

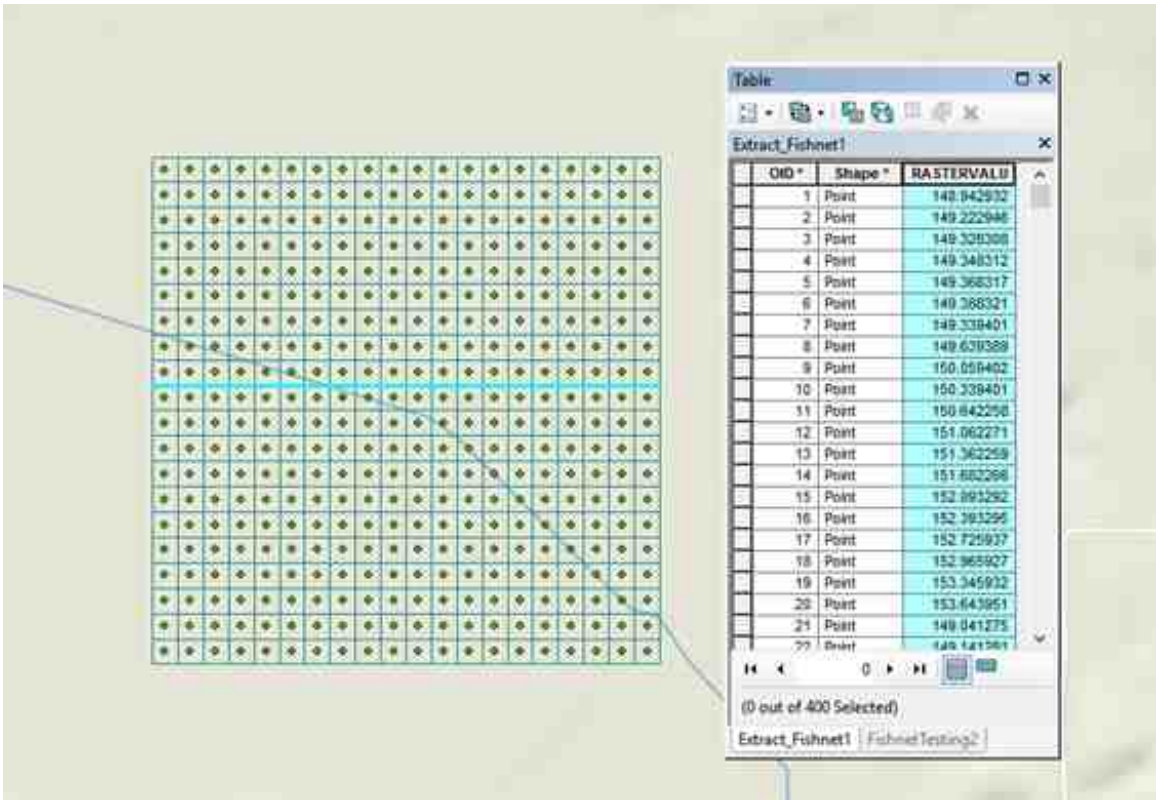

Fig. 3.5. A display of a Fishnet along with a table showing its values after comparing them to the Terrain layer using 'Points to Values'.

3.3.6 Elevation Reformatting

To be usable in the project, the data needs to be extracted from ARCGIS for Desktop and reformatted so that the project can read it in properly. The Table information shown in 3.5 can be extracted using either Excel or as just text. Once extracted into either text or Excel we reformat the data so that is it proper CSV format that is readable for the project. Reformatting the data was done both manually and handled by creating code using MATLAB that automatically reformats the data into the correct CSV format based on its output from ARCGIS.

Advantages and Disadvantages

There are some advantages to using the Fishnet method. One advantage is that using the ARCGIS for desktop software allows any kind of data to be extracted rather than just elevation data. For example, if we needed to extract terrain type based on a raster data set we could get such information using this method. This allows for any kind of information based on the layer information. Layers can be created by a user or given by ARCGIS, the layer database is quite vast and can cover a wide variety of data.

However the Fishnet method has a number of drawbacks as well. One disadvantage is that referencing the Terrain Layer using the 'Points to Values' uses ARCGIS credits. These credits are used by accessing certain kinds of data from ARCGIS's servers, a limited number is given to our team so the number of references are limited without further credits to be given to us by ARCGIS. Another disadvantage was the processing time of 'Points to Values' using ARCGIS. Extracting this data from the Terrain Layer required the software to query each point comparison to ARCGIS's server for the Terrain Layer. This led to the processing time of a 100-by-100 fishnet to be around 5-15 minutes. Although it is possible that this processing speed might be biased based on the hardware we used to run the ARCGIS for Desktop application. We found it was too slow for very large fishnets needed for large granular data. Another disadvantage is that the process of extracting data is more tedious and complex than the ARCGIS API Image Extraction methods that will be covered next.

Although the processing time and credit usage issues can be remedied if the layer is not connected to ARCGIS's server; the ARCGIS API Image Extraction method allows unlimited sections of the Terrain layer to be downloaded and used with the ARCGIS for desktop software. The ARCGIS API Image Extraction method can be used to supply the images for ARCGIS for Desktop; so it's still possible to use those extracted images with the Fishnet Extraction.

3.3.7 ARCGIS API

The next method used is ARCGIS API Image Extraction, this method uses a REST service from ARCGIS to extract images that contain elevation and imagery data. Instead of ARCGIS software, this method uses MATLAB to reformat the exported images from the REST service into a format usable by the project.

3.3.8 Set Up

To use this method only requires using MATLAB script files and using an ARCGIS account to access the REST Service. These methods rely on extracting images from two different REST services provided by ARCGIS: World Terrain and World Imagery.

3.3.9 REST Service

The ARCGIS API REST Service allows a user to extract an image based on input given to the service. Both inputs are fairly similar, shown in Figure 3.6 is the Terrain API REST Service's input interface. For the Terrain API it will return a grayscale image that represents elevation. Each pixel value is displayed by the brightness of the pixel; more intensity whiteness represents higher values and darker intensity represents lower values. The image format that that is used with the MATLAB files are TIFF images, which can be chosen in the image format option. The input options for elevation data extraction is shown in Figure 3.6. In order to get granular data the image format we use either 32bit or 64bit format. This allows the elevation data to be precise and cover the entire domain of possible values regarding elevation.
The ARCGIS API REST Service for imagery simply returns an image, such as PNG or JPEG, with similar input as shown in Figure 3.7. An example of the Imagery API is shown in Figure 3.8.

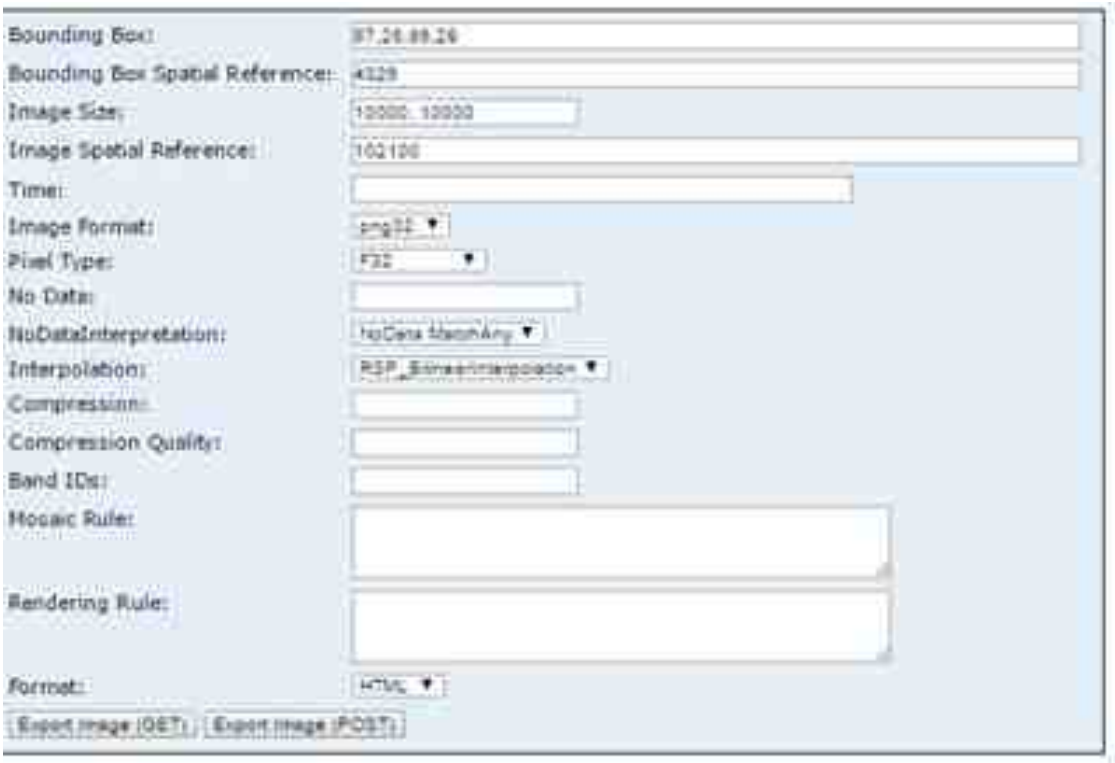

Fig. 3.6. Input GUI from the ARCGIS World Terrain API. The inputs give a user control of where to get the elevation data, set its resolution, file type, and more; after submission it returns an image that represents elevation.

3.3.10 MATLAB Implementation

MATLAB script was developed in order to take in the grayscale image from the Terrain API and transform it into a CSV format so that it can be read into the project application properly. The script has the user choose the downloaded TIFF file for the ARCGIS API. In order to reduce user error during the file format transfer, the script expects the TIFF file to be named in the proper format as discussed in 2.4.5 Loading Terrain Files; e.g.N40E40_N41E41_AreaName.tif.

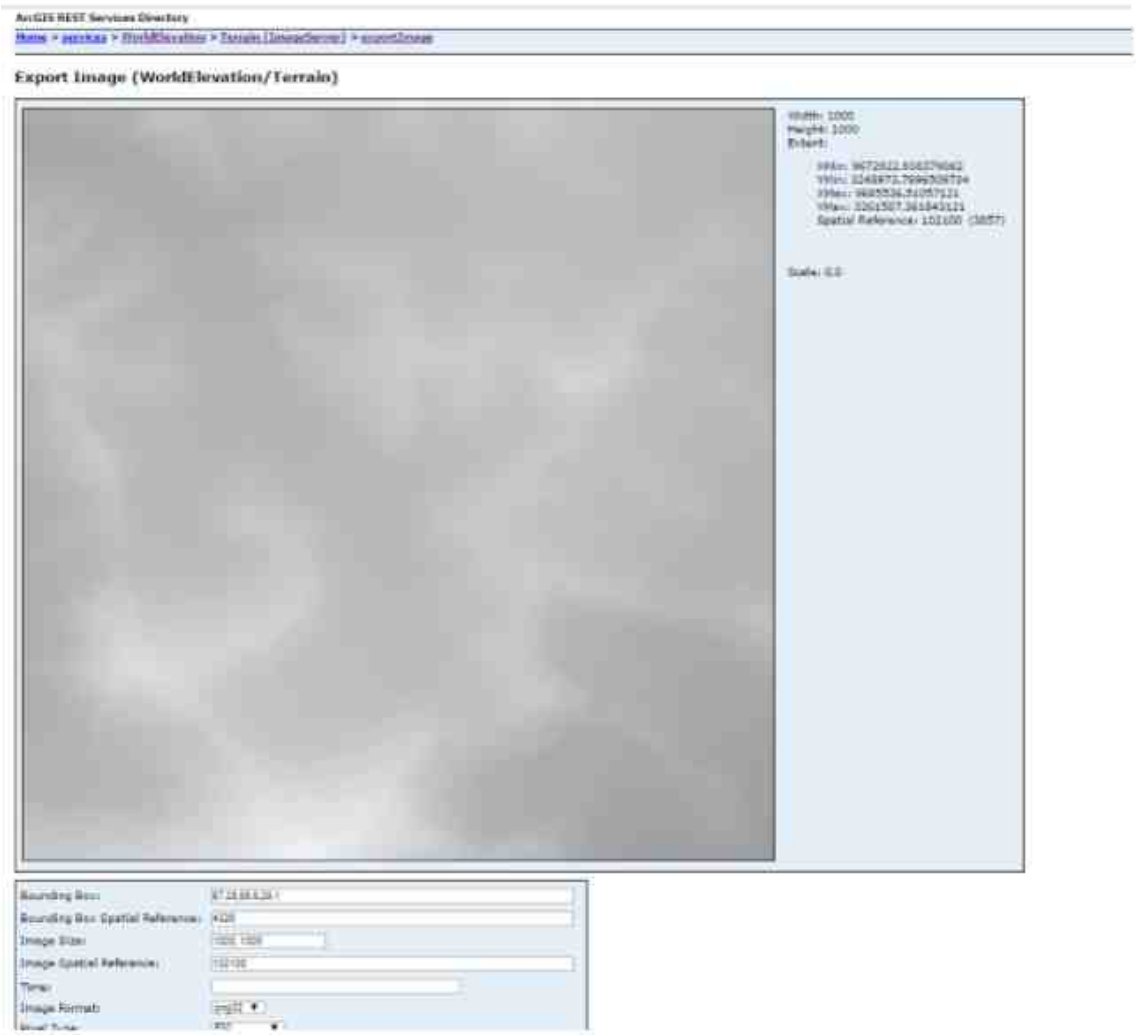

Fig. 3.7. Output from the Terrain API is returned as a light intensity image.

When the MATLAB script is run it takes the TIFF and converts into a proper CSV format. This format is so that the elevation in one row is separated by commas and each row is separated by new lines. A CSV file is outputted that is named the same as the input file to remove user input error; e.g. N40E40_N41E41_AreaName.csv.

Fig. 3.8. Output from the Imagery API is returned as a normal image in formats such as PNG or JPEG.

Advantages and Disadvantages

There are some useful reasons to why it's advantageous to use the ARCGIS API Image Extraction method. First, it does not require the use of ARCGIS credits and can be used as long as the API is in service by ARCGIS. Second, the processing speed

is much faster than the Fishnet Method. Lastly, it doesn't need external knowledge of ARCGIS software and it much easier and simpler to extract terrain data than the Fishnet method.

This method has its disadvantages as well. One disadvantage is it less robust than the Fishnet method, which can handle a wider variety of data and data layers. Although, there are more geographic information in the API than just imagery and terrain; but the data layers in the Fishnet method can be user made and cover a wider variety than what the API provides.

3.3.11 Method Choice

Either extraction can be used, depending on the need to extract terrain and imagery. While both methods of extraction were done for displaying terrain visualization, the primary choice that was used is the second method; ARCGIS API Image Extraction.

4. CPU THREADING PSO

This section covers how and why CPU threading was done in order to in increase the performance of the PSO. It will also evaluate the increased performance, consideration, and current issues of the CPU threading implementation.

4.1 Threads

A thread, short for "thread of execution", is a sequence of instructions to be processed. With growing multitasking and time-consuming applications in machines, threads can be used for running many processes simultaneously or to parallelize a single execution. With growing processing hardware advancements such as multi-core processors, each of which can handle different instruction sequences executions in parallel. Single-core processors handle multi-threaded tasks by using what is called simultaneous multi-threading; where multi-core processors handle threads using multiple cores [18].

Single-core processors use simultaneous multi-threading which is where the front end of the processor alternates between threads by fetching batches of instructions from one thread then the other [18]. The processor handles which stream of instructions go with each thread with embedded hardware [18].

Multi-core processors handle threads by using more than one core in parallel. Although, when doing parallel processing, it is important to have mutually exclusive data between the streams of instructions so that data collisions do not occur.

4.1.1 CPU and GPU Threading

Computers process information with two different kinds of processors, CPUs, or Central Process Units, and GPUs, or Graphics Processing Units. CPU processing is where most of the information on a computer gets processed, it handles information quickly and is close to the main memory so that processing is fast. GPU processing handles graphics processing and can be used to run programs in parallel, it can run large parallel processes very quickly but is further away from the main memory making data transference slower.

Parallel execution using threads are done in two general manners: CPU threading and GPU threading. CPU threading is done by using the multiple cores of the CPU; these are 4-16 high preforming cores, although high-end CPUs can have an even higher number of cores. GPU threading is done by processing the threads on the GPU cores; these cores are generally hundreds of low preforming cores.

In the project, we use CPU threading. The reason why CPU threading was chosen over GPU is because the threaded process has a fast execution time and the data transfer between the CPU and GPU is too slow in comparison. In this research the parallelization is done on the Evaluation function in the PSO for each particle, this process is very quick and runs between less than 1ms to 5ms; depending on the flow of the pso and the complexity of particle representation. The PSO converges using information gathers between generations; therefore it would not be possible to run all the generations in parallel.

GPU threading computation performance was tested using OpenCL [19], which is software that runs a C-based kernel in parallel on any kind of computation platform; such as CPU, GPU, FPGA, etc. OpenCL runs a defined kernel in parallel. However with the OpenCL version that we used it could only handle C programming language inside the kernel. Our PSO algorithm would need to be re-written without the object based support from C++. However, after testing matrix addition, we found that the program would run significantly slower on the GPU for applications that would only

take a few milliseconds due to the transfer time from the CPU to the GPU. The Fly function in the PSO was re-written to test the GPU performance, since it was simple. The data was reformatted so that it was able to use OpenCL buffers; we found that the OpenCL using GPU computation greatly reduced the overall performance of the project, and so decided to not use GPU threading from that observation. CPU threading was chosen because it doesn't require transfer of data, and has low overhead, using it were able to improves the runtime of the PSO.

4.2 OpenMP

This project uses OpenMP to do CPU threading. The OpenMP API supports multi-platform shared-memory parallel programming in C/C++ [20]. OpenMP was chosen due to its C++ support and increased thread performance due to optimization and thread polling as described in [21]. [21] describes 4 main overhead costs of OpenMP: thread library startup, thread start up, per-thread (loop scheduling), and lock management. OpenMP uses thread pooling, which re-uses threads rather than recreating them, this makes the cost of thread start-up a one time cost rather than each time the threads are used. We found that OpenMP outperformed standard threading, which is why it been chosen.

4.3 PSO Analysis

To improve the speed of the PSO with threading, it's important to analysis the PSO to determine where threading would be most effective first. This section covers a time complexity, run time analysis, as well as covering the theoretical speedup of threading.

4.3.1 PSO Time Complexity Analysis

Time complexity is the computational complexity that estimates the runtime based on variables of the function or algorithm. In this analysis we use big O, which measures the limiting behavior of some function when it towards a certain value or infinity. A flow chart showing an analysis of time complexity of the PSO is shown in Figure 4.1.

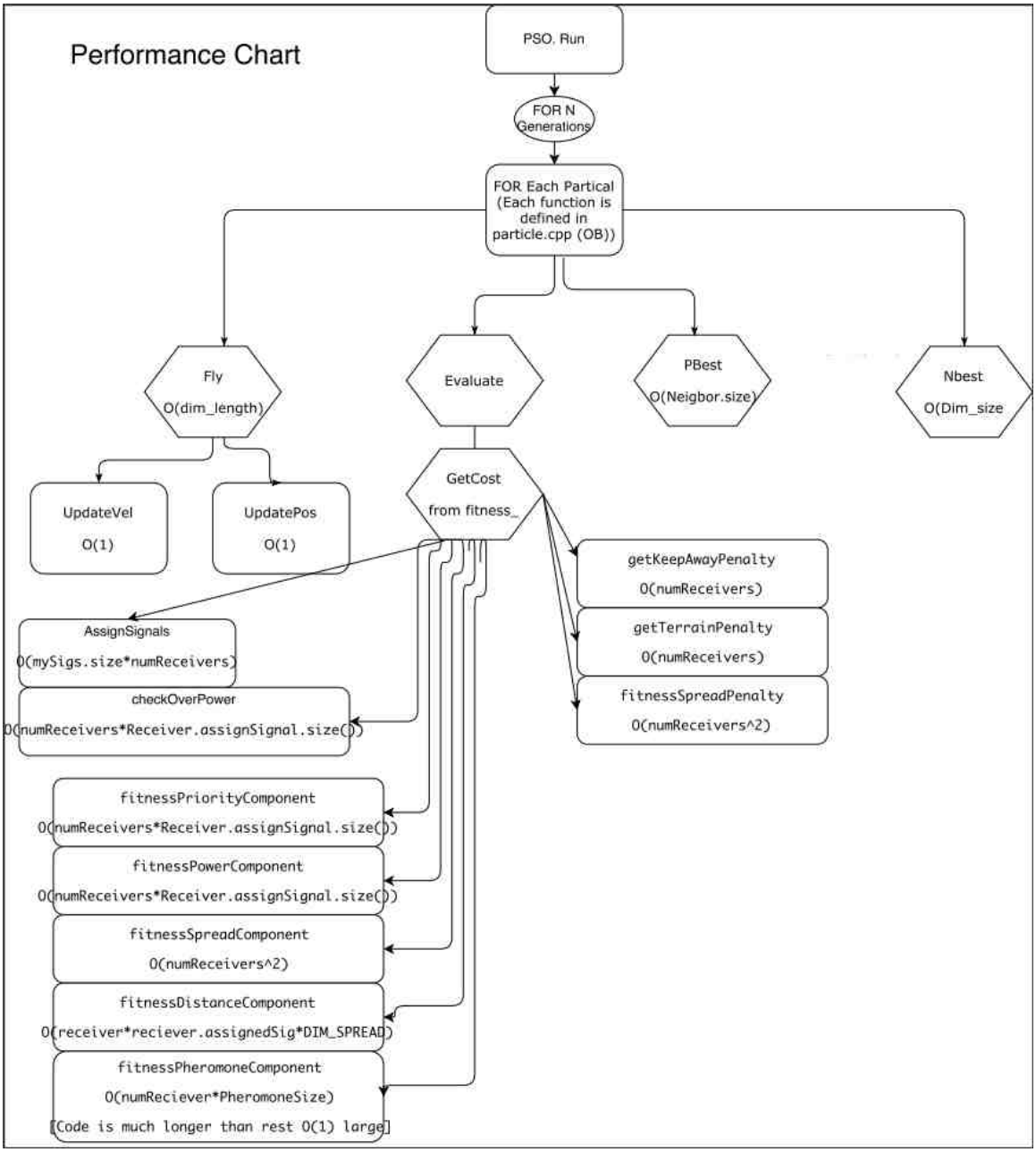

Fig. 4.1. A Flow chart showing the time complexity of the PSO. The PSO.Run function is broken up to run for every particle and then separated into 4 main function calls: Fly, Evaluate, PBest, and NBest. Each function's time complexity shown as big O(n).

At the top of Figure 4.1 *PSO.Run* represents the process of the PSO running, the PSO is run for every generation shown as *FOR N Generations* below *PSO.Run*. On top of that it is also run for each particle. Then the PSO is broken up into 4 main functions: *Fly*, *Evaluate*, *PBest* and *Nbest*. The parameters of the PSO will change which functions take up more overall time in the PSO. For example having more targets, called signals in the code, will increase the runtime cost of *AssignSignals*; where more assets, called receivers in the code, will increase the runtime cost of the fitness component functions. It's also noted that most of the function's runtime in *getCost* is mostly controlled by the number of assets and signal assignments; and *AssignSignals* is controlled by the total number of targets.

4.3.2 PSO Runtime Analysis

The runtime analysis looks at where the PSO is spending its time. To do this we have to implement high resolution timers in micro-seconds to get a proper analysis. This analysis compares the runtime of the total PSO runtime against its components as well as the subroutines called within *Evaluate* and *getCost*.

Test Environment

The hardware and software using for testing is shown in Figure 4.2. In this case the most important components would be the Intel processor, the NVIDIA GPU is only used for displaying the 3D Visualization.

The following measurements were done by running the application with the default loaded environmental setup shown in 2.5 with the options' setup shown in 4.3.

- **Hardware:**
  - **Processor:** Intel(R) Core(TM) i5-6400 CPU
    - 2.70GHz , 4 Cores, 4 Logical Processors
  - **Installed memory (RAM):** 12GB DDR3
  - **Graphics:** NVIDIA GeForce GTX 970
    - DirectX 12.0
- **Software:**
  - **OS:** Windows 10 with 64-bit Operating System, x64- based processor
  - **Libs:**
    - Qt Creator 4.5.0
    - Desktop Qt 5.8 – 32Bit
    - Microsoft Visual Studios 2015
    - QWT 6.1.3
    - Boost 1.61.0
    - DLIB 19.3

Fig. 4.2. The hardware and software used to run the runtime analysis.

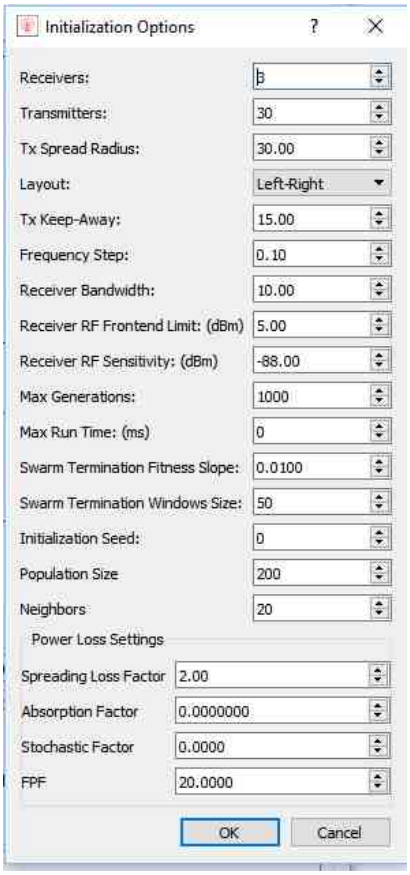

Fig. 4.3. Settings used for thread and PSO analysis.

Test Measurements

Figure 4.4 shows a table of the runtime comparisons between the functions of the PSO. The data gathered is from running the environment described in Test Environment for 5 runs and taking the average amongst the runs. The average runtime takes 154.2 generations to converge. The setup environment is shown in Figure 4.3 and Figure 2.5. Important data from Figure 4.4 is that about 80% of the total runtime lies inside *Evaluate*. *Evaluate* returns the fitness value based on the current generation and particle environment between generations.

Function	Runtime	Percent
Total PSO Runtime	346.5382ms	100%
Fly	17.5258ms	5.06%
Evaluate	276.6432ms	79.83%
PBest	0.2868ms	0.08%
NBest	0.749ms	0.22%
Other	51.3334ms	14.81%

Fig. 4.4. Runtime Analysis of the overall PSO.

Figure 4.5 shows the runtime analysis of *Evaluate*, where *getCost* takes up 89.7% runtime; the other 10.3% creates a variable to store particle information from the current particle and passes it as an argument to *getCost*.

Figure 4.6 shows the runtime analysis of *getCost*, where the critical sections, described in the next section, Threading Implementation, take up 64.3% of the function. This critical section is shown in 4.10 and 4.9. OpenMP describes a critical section as "...sections to prevent multiple threads from accessing the critical section's code at the same time, thus only one active thread can update the data referenced by

Function	Runtime	Percent
Total Evaluate Runtime	276.6432ms	100%
getCost	248.2064ms	89.7%
Other	28.4368ms	10.3%

Fig. 4.5. Runtime Analysis of Evaluate.

the code. " [20]. The other 35.7% of the code which contains the fitness function components and penalty functions, and are freely threaded without critical sections.

Function	Runtime	Percent
Total getCost Runtime	248.2064ms	100%
Critical section	159.5784ms	64.3%
Other	88.628ms	35.7%

Fig. 4.6. Runtime Analysis of Evaluate.

It's important to emphasize that the recorded measurements come from running the functions for every particle as well as every generation. In these measurements we ran the PSO using 200 particles and it took an average of 154.2 generations to converge. Thus, to get the runtime of the PSO per generation, the results would need to be divided by 154.2, and to get the runtime per particle they would need to be divided again by 200. Using the runtime from 4.4 that would make the runtime of the PSO per generation 2.2473ms; and each particle in that generation take 0.011237ms. These are important because we can only parallelize each generation, this is because of the convergence property of PSO.

From the observation, it was decided that it would be best to only use threading on *Evaluate* since took the most runtime compared to the rest of the functions in the PSO.

Theoretical Threading Speedup

The maximum possible speed up from parallelizing code is calculated by using Amdahl's law:

$$SpeedUp = 1/((1-p) + p/s) \tag{4.1}$$

Where p is the proportion of execution time that the part benefiting from improved resources originally occupied, and s is the speedup of the part of the task that benefits from improved system resources. In our case p is the section of code that is run using threading, and s is the number of CPU cores using for threading.

It is important to notice that this describes perfectly breaking the code into fully serial and fully parallel sections. For our research, we have a serial section and a parallel section with a critical section in it. Critical sections can only have 1 thread running them at a single time, but that doesn't mean that the code is completely running in serial during all the critical sections. Hypothetically, consider 2 threads running code in which 60% of the start of the code is critical and the remaining 40% can run in parallel. Thread 1 can run and finish its critical section, blocking thread 2 since only 1 thread can run a critical section at a time. Once thread 1 finishes its critical section, thread 1 can freely run the remaining 40% in parallel with the critical section of thread 2; meaning that a certain part of the process can be run by both threads at the same time even if one thread is in a critical section. Although the portion of runtime in true parallel is decided by the software and hardware that handles thread scheduling and blocking; this makes it difficult to measure the theoretical speedup with the given configuration. An important thing to remember with equation 4.1 is that it assumes that there is no overhead or delay when

running code in parallel. Depending on the configuration, it can be very difficult to achieve the speedup results from the calculations from 4.1.

Using information from 4.4, 4.5, and 4.6; we can determine the total portion of the code that is in serial, the portion of the code that can run in parallel, and the portion of the code which is in critical sections. Since we can't measure an absolute speedup with the critical sections, we will consider a number of scenarios to analyze our maximum speedup and compare it to our results from Threading Implementation. In these scenarios we will use 4.1 to determine the maximum speed between 2-4 threads, we will also determine different ways to break the code into serial and parallel sections based on the understanding of the code being parallelized.

The scenarios that will be considered are the following: scenario 1 assume critical sections act as parallel sections, and scenario 2 assume critical sections act as serial. This implies that for scenario 1, all of evaluate runs in parallel; and for scenario 2 only the noncritical sections run in parallel. The percentage split of serial and parallel, as well as the speedup calculated using 4.1, is shown in 4.7.

Scenario	Threads	Serial %	Parallel %	Speed Up
1	2	20.17%	79.83%	**1.66 (+66%)**
1	3	20.17%	79.83%	**2.14 (+114%)**
1	4	20.17%	79.83%	**2.49 (+149%)**
2	2	66.22%	33.78%	**1.20 (+20%)**
2	3	66.22%	33.78%	**1.29 (+29%)**
2	4	66.22%	33.78%	**1.34 (+34%)**

Fig. 4.7. A Table showing the serial and parallel splits when considering between scenarios 1 and 2. Where scenario 1 assume critical sections act as parallel sections and scenario 2 assume critical sections act as serial.

From 4.7 we can assume that the range of maximum speed up using 4 threads is between 34% and 146%.

4.4 Threading Implementation

In order to thread *Evaluate*, we need to use OpenMP pragma's on the outermost function call. This was done in pso.cpp and is shown in Figure 4.8.

```
225             #pragma omp barrier
226             #pragma omp parallel for default(none) num_threads(threadNum) private(k)
227             for(k = 0; k < (signed) _impl->parameters->pop_size_; k++)
228             {
229                 //#pragma omp critical
230                 //{
231                 //std::cout << "thread id" << omp_get_thread_num() << std::endl;
232                 _impl->particles[k]->Evaluate();
233                 //}
234             }
235             #pragma omp barrier
```

Fig. 4.8. Implementation of threading Evaluate in pso.cpp.

The variable *Threadnum* is passed in the pragma to determine the number of threads to be used in the for loop. Threading existing code that runs in serial often is difficult because the original development often doesn't consider how the code will run in parallel. Just doing the following to thread *Evaluate*, causes the program to crash due to a data collision. Because of this, it was necessary to look at the data relationship and possible areas of data collision in the code.

After analyzing *Evaluate* as well as everything that was connected to it, we determined that there was a function reference to a file called *fitness_SpatialReceiver.cpp*, which is the majority of the work is done in *Evaluate*, that every particle object would share access to. This shared access to the same function file causes a data collision between the classes representing the *Receivers*, or *Assets*, which caused the program to crash when attempting to thread the function.

To remedy this we needed to create memory exclusion between the threads, so that each thread would access its own memory so that it didn't cause a data collision.

Unfortunately, we never were able to completely remove all dependencies within the code so that each thread was completely mutuality exclusive. Since the code that is being parallelized is very complex and shared between many objects and dependencies in the project, it's difficult to determine data dependencies and if they can be remedied. Although, we determined that *Receivers* is able to become exclusive if we separated the data into a vector and access it exclusively between threads. This was done using a map to access the vector of Receivers so that each thread had its own copy of the receivers object, shown in Figure4.8. We also use openMP's critical pragma to deal with section that needed to run in serial. The critical pragma only allows 1 thread to run the section at a time; without this critical section the receiver placement and assignment would not be acted as expected in the project.

The data separation of the receiver object was done in two ways to try to make the threading more efficient: thread ID separation and particle ID separation. With thread ID separation we create a vector of receiver objects based on the number of threads, so the mapping would be based on the thread ID. This method uses the least amount of memory possible while keeping the data excluded. This implementation is shown in Figure 4.9.

```
//critical Receiver creation, assignment, and power check
#pragma omp critical
{
std::vector<XcvrObject> & receivers = m_receiver_map[std::this_thread::get_id()];
// Assign signals to receivers based on which signals fall into the bandwidth of each receiver.
assignSignals(receivers);
// Check each receiver to make sure the RF front end is not overloaded.
checkOverPower(receivers);
}
```

Fig. 4.9. Implementation of remedying data collision by creating a map of the receiver objects separated by thread ID and making critical the creation, assignment, and power check of the receiver objects.

The other method to further stabilize the threading possess is to create more data exclusion. This was desired because of OpenMP's thread pooling. Thread polling is creating the threads only once and then re-uses them. In this method we separated

by particle ID, where every thread instance references a different receiver object. For example, if there are 200 particles, there would be a vector of 200 receivers and each particle would reference a different copy of the receiver object. This implementation is shown in Figure 4.10

```
//critical Receiver creation, assignment, and power check
#pragma omp critical
{
std::vector<XcvrObject> & receivers = m_receiver_map[particleId];
// Assign signals to receivers based on which signals fall into the bandwidth of each receiver.
assignSignals(receivers);
// Check each receiver to make sure the RF front end is not overloaded.
checkOverPower(receivers);
}
```

Fig. 4.10. Implementation of remedying data collision by creating a map of the receiver objects separated by particle ID and making critical the creation, assignment, and power check of the receiver objects.

A GUI was also implemented so that threading was enabled and changed, this implementation is shown in the bottom right in Figure 4.11.

4.5 Results

Performance testing for threading was done on an Intel Core i5-6400 CPU @ 2.70GHz with 4 cores, using 4 threads. Default start-up settings were used, shown in Figure 4.11 with the options shown in Figure 4.3. The environment was a flat plane that is 200km by 200km.

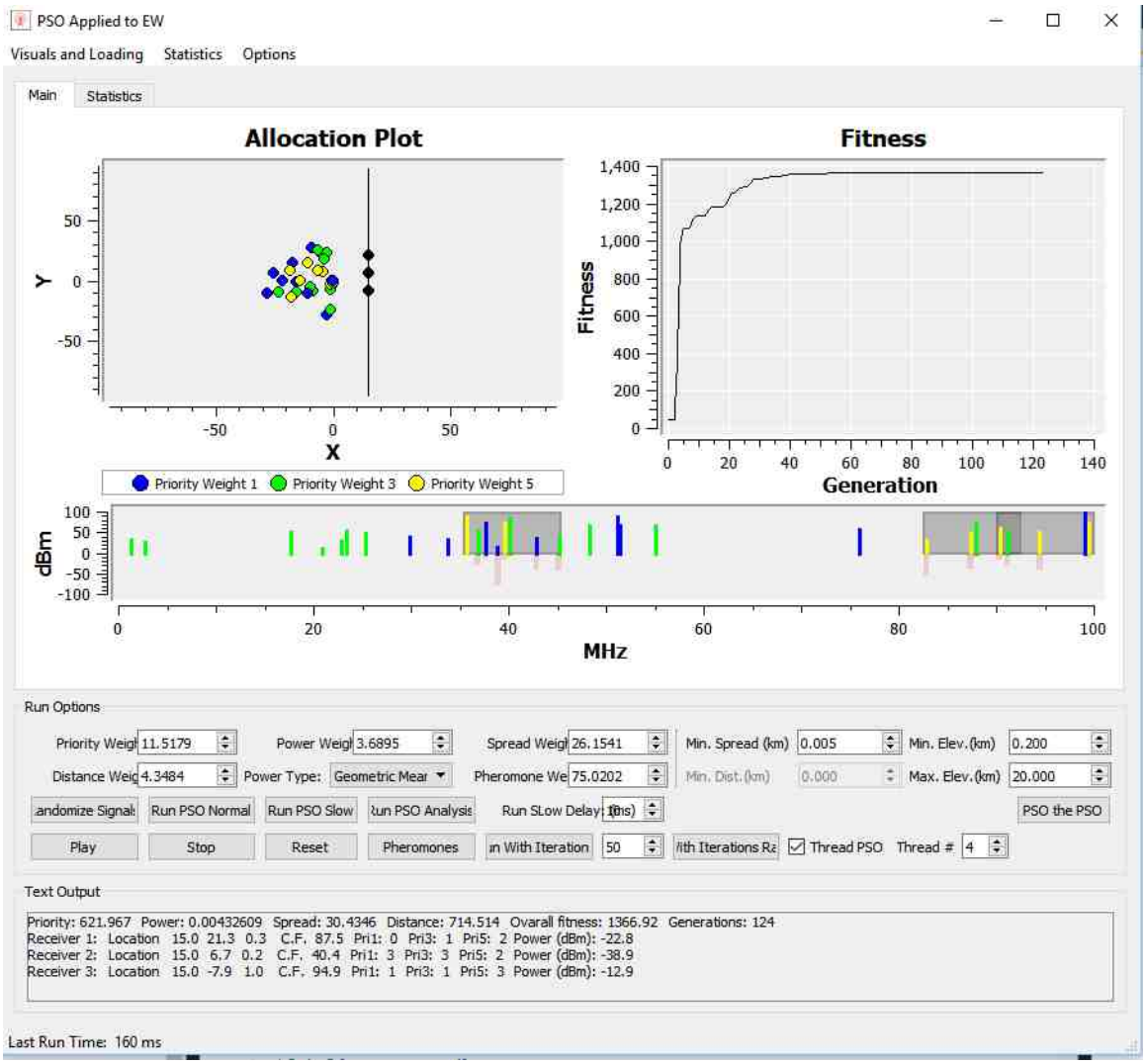

Fig. 4.11. Threading GUI Implementation so that the user can enable, disable, and set the number of threads to be run.

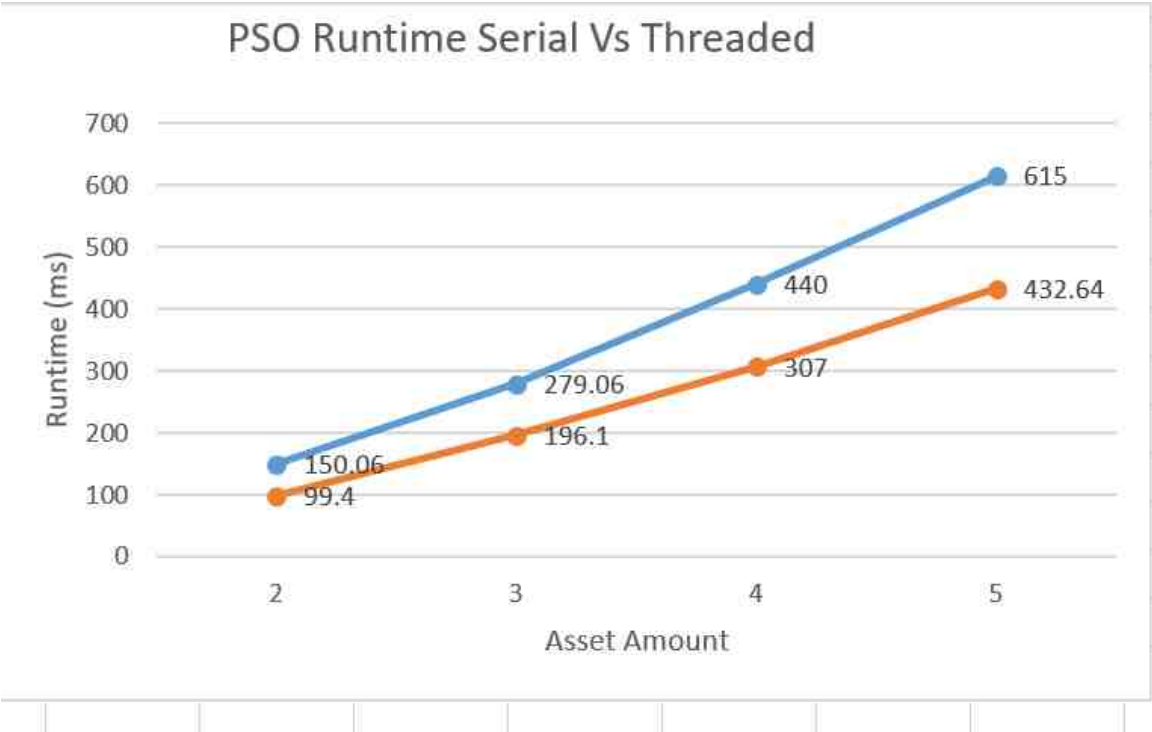

Fig. 4.12. A graph showing the runtime vs asset number and comparing the serial runtime against the threaded runtime.

A graph showing the performance improvements with changing asset amounts is shown in Figure 4.12. Each recorded point was done by randomizing the transmitter placement before each run while keeping the radius the same, this was done so that a more generalize solution was measure rather than a specific configuration of the transmitters. Each test took the average runtime of 50 runs. The speed increase from threading was from 42% to 51% higher than serial runtime from the graph shown in Figure 4.12.

The ideal thread number is generally automatically determine by OpenMP's *get_num_procs()* command; which returns the number of usable logical processors on the machine that is running. On the machine that was used for testing, it has 4 logical processors; meaning 4 threads would be optimal. An analysis was done on the effect of the number of threads vs. performance, shown in Figure 4.13. This graph

shows that the impact of 2 threads still increases the performance similar to 3 or even 4 threads; this has to do with the overhead with more threads and blocking, as well as the short runtime of a few milliseconds. Since the processor can only handle up to 4 threads at a time, the high number of threads doesn't change performance.

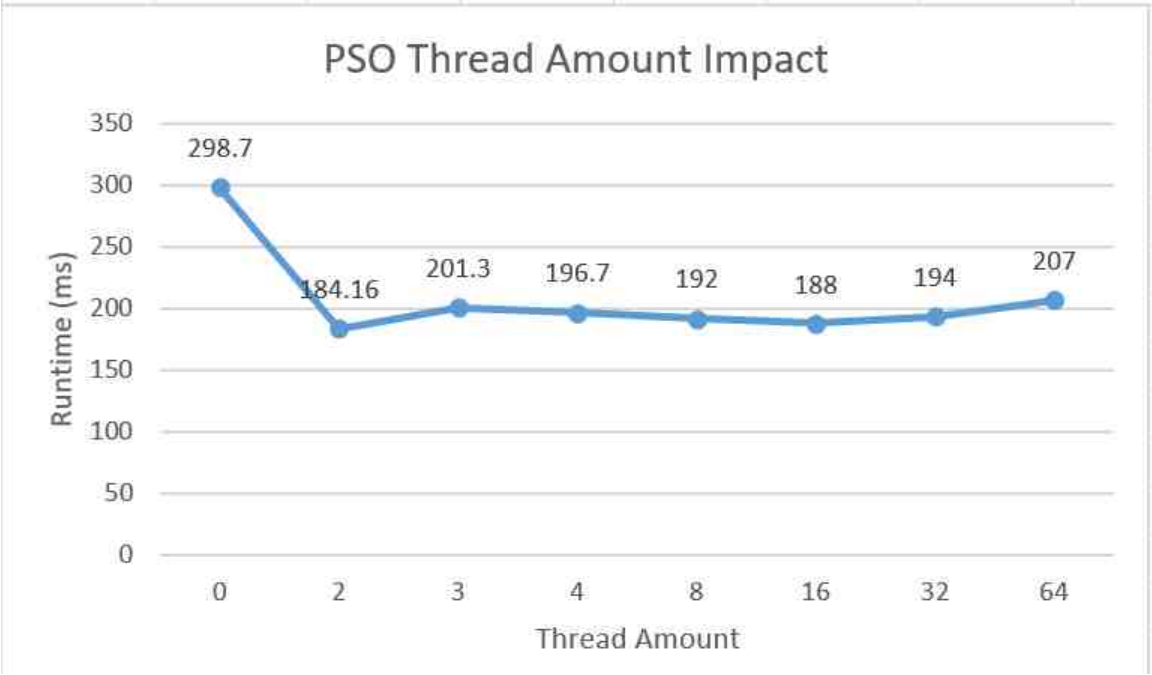

Fig. 4.13. A graph showing the runtime vs asset number and comparing the runtime from different number of threads.

It is important to remember that the section being threaded runs very quickly, around 2.2473ms. This is important because the overhead cost of OpenMP threads becomes more significant with smaller runtime [21]; thread overhead is mainly from the blocking cost from the critical section and the scheduling cost amongst threads. Additionally, if a processor has 4 cores, it doesn't necessarily mean a program will necessarily run 4 times faster. This has to do with the hardware and memory resources available on the processor, as well as the scheduling cost of threading.

4.6 Results Vs Theoretical

From our results, it was found that the maximum speed up was around 51% with 4 threads. We determined that our theoretical maximum speed up was between 34% and 146% from Figure4.7. If we consider scenario 1, in which critical sections run as if they are parallel, with only 2 thread we get a maximum speed up 66%. Also, if we analysis figure 4.13 we notice that runtime didn't increase past 2 threads; and the measured performance increase with two threads is 61.6%. From this we see that the critical sections have an effect on the number of threads being used and will somewhat limit the performance achieved to running just 2 threads. This makes sense since 57.7% of Evaluate is in a critical section, meaning 46% of the entire PSO runtime is in a critical section. Around half of the time spent in the parallel section is in a critical section, that implies that 1 thread would at least block all other threads 50% of the time if they are all in critical sections. With the performance comparison and portion of the parallel code that is in a critical section; it means the that maximum speed up is similar to the maximum speedup found in scenario 1 with 2 threads, which is assuming the critical section runs in parallel with 2 threads. This implies that the maximum speed up from threading with any number of threads in this application lies around 66% with the current setup.

4.7 Issues and Problems

Unfortunately there are some issues with the current implementation of threading the PSO. One issue is that we were unable to completely remove all critical section, this is important because critical section slows down the threading speedup since only one thread can enter a critical section at a time. Another issue is that, on a rare occurrence, the output of the PSO with threading doesn't appear to be what is expected. Although, this occurrence only happens in about 1 in every 40 runs or so. It's possible that it is a data collision, an unexpected outside effect of threading, or an outside bug somewhere in the project.

5. SUMMARY

This research uses QT Data Visualization to display the PSO solutions, assets, transmitters in 3D space from the work done in [2]. Elevation and Imagery data was extracted from ARCGIS (a geographic information system (GIS) database) to add overlapping elevation and imagery data to that the 3D visualization displays proper topological data. This research also improves the performance of the PSOs runtime, using OpenMP with CPU threading to parallelize the evaluation of the PSO by particle. The use of 4 threads improves the performance of the PSO by 42% - 51% in comparison to running the PSO without CPU threading. The contributions provided allow for the PSO project to be more realistically simulate its use in the Electronic Warfare (EW) space, adding additional threading implementation for further performance improvements.

6. RECOMMENDATIONS

Further analysis and work towards optimizing and improving upon the threading should be continued. If possible, it is recommended to completely remove all critical sections while preventing data collisions. Further work on the movement of transmitters to simulate more realistic movement found in the EW space should be done. To do this a transmitter movement scheduler that defines movement based on time and user defined input schedule is recommended. This scheduler would need to give the user enough freedom so that they can move the transmitters based on the time passed. Further classification of both transmitters and assets can better simulate real life objects, there these objects would be created based on people and vehicles that would commonly be found in the EW space; such as cars, people, planes, jeeps, tanks, etc. This implementation can be done by inheriting OOP created objects already found in the source code that was implemented from [1]. It is recommended to use ARCGIS to extend the elevation and imagery data by including terrain types (such as grass, gravel, cement), this can be used in signal refection for the RF propagation model. Lastly, it is recommended to extend the project to simulate a cityscape environment. Cityscape data would have to be extracted from a data source, such as ARCGIS. This data would need to have a high enough resolution to distinguish between ground and buildings. Although, to support cityscape environment; additions would need to be added to the existing project in order for the application to run as expected. These additions would include: crash detection for assets, path finding for ground and air units, cityscape propagation models, extraction of building and ground type data, proper transparency visualization, as well as other features necessary for a cityscape environment to be used for simulating asset allocation in the EW space using the PSO.

REFERENCES

REFERENCES

[1] W. Boler, "Electronic warfare asset allocation with human-swarm interaction (unpublished master's thesis)," Master's thesis, Purdue (IUPUI), Indianapolis, IN, 2018.

[2] J. Crespo, "Asset allocation in frequency and in 3 spatial dimensions for electronic warfare application," Master's thesis, Purdue (IUPUI), Indianapolis, IN, 2016.

[3] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," *MHS95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, vol. 1, pp. 39–43, 1995.

[4] X. Hu, "Pso tutorial," (last date accessed: 4/11/2018). [Online]. Available: http://www.swarmintelligence.org/tutorials.php

[5] N. Holden and A. A. Freitas, "A hybrid pso/aco algorithm for discovering classification rules in data mining," *Journal of Artificial Evolution and Applications*, p. 111, 2008 (last date accessed: 4/11/2018). [Online]. Available: https://www.hindawi.com/journals/jaea/2008/316145/abs/

[6] R. J. Kuo and C. C. Huang, "Application of particle swarm optimization algorithm for solving bi-level linear programming problem," April 2009 (last date accessed: 4/11/2018). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0898122109002260#!

[7] Q. Wu, C. Cole, and T. McSweeney, "Applications of particle swarm optimization in the railway domain," April 2016 (last date accessed: 4/11/2018). [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/23248378.2016.1179599?journalCode=tjrt20

[8] J. Reynolds, "Particle swarm optimization applied to real-time asset allocation," Master's thesis, Purdue (IUPUI), Indianapolis, IN, 2015.

[9] P. R. Witcher, "Particle swarm optimization in the dynamic electronic warfare battlefield," Master's thesis, Purdue (IUPUI), Indianapolis, IN, 2017.

[10] B. Secrest and G. Lamont, "Visualizing particle swarm optimization - gaussian particle swarm optimization," *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS03 (Cat. No.03EX706)*, June 2003.

[11] CalcProgrammer1, December 2011 (last date accessed: 4/11/2018). [Online]. Available: https://www.youtube.com/watch?v=bbbUwyMr1W8

[12] "Natural resources conservation service," (last date accessed: 4/11/2018). [Online]. Available: https://www.nrcs.usda.gov/wps/portal/nrcs/main/national/ngce/elevation/

[13] O. Ivanov, B. C. Neagu, and M. Gavrilas, "A parallel pso approach for optimal capacitor placement in electricity distribution networks," *2017 International Conference on Modern Power Systems (MPS)*, July 2017.

[14] M. Rabinovich, P. Kainga, D. Johnson, B. Shafer, J. J. Lee, and R. Eberhart, "Particle swarm optimization on a gpu," *2012 IEEE International Conference on Electro/Information Technology*, June 2012.

[15] "Qt data visualization," (last date accessed: 4/11/2018). [Online]. Available: http://doc.qt.io/qt-5/qtdatavisualization-index.html

[16] "Surface example," (last date accessed: 4/11/2018). [Online]. Available: https://doc.qt.io/qt-5.10/qtdatavisualization-surface-example.html

[17] "Map projections - types and distortion patterns," (last date accessed: 4/11/2018). [Online]. Available: http://geokov.com/education/map-projection.aspx

[18] J. Stokes, "Ask ars: what is a cpu thread?" (last date accessed: 4/11/2018). [Online]. Available: https://arstechnica.com/information-technology/2011/04/ask-ars-what-is-a-cpu-thread/

[19] "Opencl - the open standard for parallel programming of heterogeneous systems," (last date accessed: 4/11/2018). [Online]. Available: https://www.khronos.org/opencl/

[20] "Home," (last date accessed: 4/11/2018). [Online]. Available: http://www.openmp.org/

[21] P. Lindberg and Intel, "Performance obstacles for threading: How do they affect openmp code?" June 2017 (last date accessed: 4/11/2018). [Online]. Available: https://software.intel.com/en-us/articles/performance-obstacles-for-threading-how-do-they-affect-openmp-code