2005

Virtual environment UAV swarm management using GPU calculated digital pheromones

Bryan Walter
Iowa State University

Virtual environment UAV swarm management using GPU calculated digital pheromones

by

Bryan Walter

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Co-majors: Human Computer Interaction; Mechanical Engineering

Program of Study Committee:
Adrian Sannier, Co-major Professor
James Oliver, Co-major Professor
James Bernard
Greg Luecke
Dirk Reiners

Iowa State University
Ames, Iowa
2005

Graduate College
Iowa State University

This is to certify that the doctoral dissertation of

Bryan Walter

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Co-major Professor

Signature was redacted for privacy.

Co-major Professor

Signature was redacted for privacy.

For the Co-major Program

Signature was redacted for privacy.

For the Co-major Program

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Our future military force will be complex: a highly integrated mix of manned and unmanned units. These unmanned units could function individually or within a swarm. The readiness of future warfighters to work alongside and utilize these new forces depends on the creation of usable interfaces and training simulators. The difficulty is that current unmanned aerial vehicle (UAV) control interfaces require too much operator attention and common swarm control methods require expensive computational power. This dissertation discusses how to improve upon current user interfaces and how to improve the performance of a common swarm control method, the digital pheromone field. This method uses digital pheromones to bias the movements of individual units within a swarm toward areas that are attractive and away from areas that are dangerous or unattractive. A more efficient method for performing pheromone field calculations is introduced, one that harnesses the power of the GPU (graphics processing unit) in today's graphics cards by reshaping the ADAPTIV swarm control algorithm into a form acceptable to the GPU's pipeline [1]. The GPU ADAPTIV implementation is tested in scenarios that involve up to 50,000 virtual UAVs. When compared to its counterpart CPU implementation, the GPU version performed over 30 times faster than the CPU version. This gain translates directly into lower costs for training the future warfighter today and fielding the swarms of tomorrow. Finally, this dissertation presents a vision for combining these new interface ideas and performance enhancements into an effective swarm control interface and training simulator.

INTRODUCTION

The military has a clear picture of the force it would like to field in the coming decades and this image has a single prevailing theme: the integration of manned and unmanned units. The addition of unmanned units will decrease the danger soldiers face in direct combat, and the Department of Defense (DoD) roadmap calls for an immediate and sustained increase in the use of unmanned units, starting with unmanned aerial vehicles (UAVs) [2]. By 2012, the DoD roadmap projects that F-16-sized UAVs will perform a complete range of combat and combat support missions, including suppression of enemy air defenses, electronic attack , and even deep strike interdiction. UAVs are to specialize in missions commonly categorized as "the dull, the dirty, and the dangerous."

Furthermore, the DoD roadmap requires that a single operator be able to control a UAV swarm. However, for this goal to be realistic, one of two research paths must be chosen and followed. Either the UAVs in the swarm will be totally autonomous and therefore require no human supervision, or the human interface to the swarm will need to be radically different from current UAV interfaces, which require multiple pilots for a single UAV. This dissertation makes the case for the second path and then elaborates on how to improve the current state of the art. The dissertation introduces two major research challenges, one a more efficient way to control the UAVs and the other a superior way to monitor and manage a swarm's progress, minimizing the required number of operators. It then presents my methods for attacking each challenge, discusses the noticeable performance gain achieved by the research, describes an interface that allows one user to operate a swarm and a discusses the future direction of this research.

MOTIVATION

UAVs have certainly come a long way since Elmer Sperry's 1918 "Aerial Torpedo" but further advancement is required to unlock the true force-multiplying potential of unmanned aircraft [3]. The Predator, the most common UAV in active duty, requires at least two operators, one to fly the aircraft and the other to manage the camera mounted on it [3]. More commonly, four people man the craft because controlling it is such a taxing task. The four people break off into teams of two and alternate controlling and resting. When controlling, they use an interface like the one shown in Figure 1.

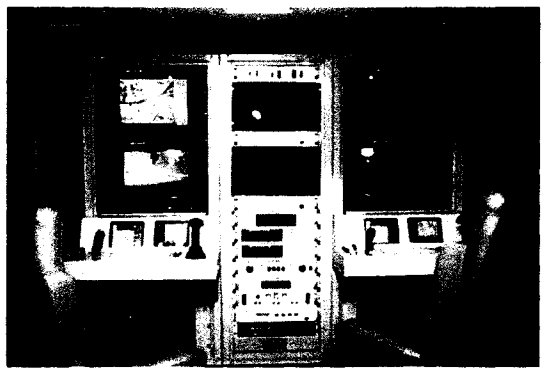

Figure 1. Predator ground control station

One of the main reasons the Predator is taxing to operate is that the field of view afforded to the pilot is poor, resulting in a loss of situational awareness. Situational awareness, knowledge of a location and what is occurring near that location, is critical when operating a vehicle in a hostile environment. According to a recent report on the loss of Predator aircraft during missions:

> A good number of them were lost due to operator error, since it is hard to land the UAV. The operator has the camera pointing out the front of the plane, but he really has lost a lot of situational awareness that a normal pilot would have of where the ground is and where the attitude of his aircraft is [4].

Predator operators liken piloting the plane to flying while looking through a soda straw [5]. If it takes at least two people to control one Predator it would take at least two thousand people to control one swarm of a thousand Predators. Clearly, this situation must be improved for swarm control to increase the capability of the military while maintaining its current number of personnel. This fact is reinforced by the DoD roadmap since it states that the ground control station must evolve as UAVs grow in autonomy. Specifically, UAV swarms "must be controllable by non-specialist operators whose primary job is something other than controlling the UAV." This demands "a highly simple and intuitive control interface ... and the capability for autonomous vehicle operation of one or more vehicles being controlled by a single operator" [2].

These requirements can not be met by current technologies. A reason is that current UAVs require too much operator attention, and this problem will only partially be solved by UAVs with more autonomous flight capabilities. These future UAVs will be capable of flying to a given set of coordinates while automatically avoiding collision. However, even such advanced UAVs will still require human supervision to know where to go and to confirm sensitive actions. Further improvement of UAV ground control stations will be necessary.

Due to the difficulty of maintaining situational awareness, it is tempting to suggest that people should not be involved in the control of UAVs. This argument calls for completely autonomous UAVs capable of taking off, flying their mission, taking pictures of

or striking a target, and then landing -- all without human input. While not entirely technologically feasible at present, it may be a viable path for future UAV research and development. In fact, some UAVs are already capable of taking off, flying a specified path, and landing on their own [6]. The trouble comes in the execution of the mission, be it either strike or reconnaissance. If this task is left entirely to the computer on the UAV, commanders will be unable to focus their swarm on rapidly changing areas of interest in the case of reconnaissance, and they will be unable to decide which targets are attacked in the case of strike. The latter case is of particular concern, because a glitch in the software a UAV uses to decide whether a building is an enemy HQ or a hospital could result in civilian casualties. Additionally, autonomous UAVs could confuse friendlies with enemies if their software fails to make the correct decision.

These concerns were expressed in the recent New York Times article "Who do you trust: G.I. Joe or A.I. Joe" [7]. This article outlines the dangers of letting artificial intelligence (A.I.) decide everything by taking the human out of the loop. Some might argue that if the A.I. were flawless then there would be no issue. However, it is unrealistic to expect flawless A.I. anytime soon and perhaps ever. After all, humans make false positive errors, and we are still much better at pattern recognition and high level decisions than computer A.I. Therefore, it seems that a human must be in the loop of control, be it for a single UAV or an entire swarm of them.

A human in the loop of swarm control requires three things. First, to enable either direct or indirect control, the human must have access to the control algorithm used by the swarm. Second, the human must be able to monitor the swarm's progress on a global level without getting caught up in the state of individual units of the swarm since there would be

hundreds or thousands of UAVs, far too many to monitor each one. Furthermore, direct individual communication with that many units would be expensive. Third, the operators must be trained adequately to manage swarms. This calls for a simulator to prepare commanders for swarm control.

In short, what is needed is a system that has simple controls with relatively uncomplicated swarming entities that perform their tasks without knowledge of the entire system, but whose combined actions exhibit complex aggregate behavior. Further, a simulation of this system must be made available to potential operators well in advance of swarm deployment to ensure the warfighter's readiness. Swarming UAVs should be kept as simple as possible to minimize costs and to enhance their robustness in the field. Despite this pressure toward simplicity and expendability, these groups of UAVs will still be required to perform complex tasks as a whole. While this may sound daunting, nature has provided a template for accomplishing these tasks in social insects such as ants. A swarm control algorithm can be created by making analogies between UAVs and social insects such as ants or bees.

A key concept in insect-inspired swarm control is pheromones. Pheromones can be thought of as markers to tell units whether an area is attractive or unattractive for future exploration. In this way, UAVs can use local pheromone levels to determine which direction they should go. With such an algorithm, the operator could change simple parameters relating to pheromones to influence the movement of the swarming UAVs. For this control to be effective, the operator needs a clear understanding of the entire battlefield and what the swarming UAVs should focus on. A fundamental challenge with insect-inspired algorithms is that they are computationally expensive for large fields.

A strategy explored in my research for improving the performance of insect-inspired algorithms stems from recent developments in graphics programming. Modern graphics cards in commodity PCs have become complex enough to require their own processing unit. This unit, commonly called the graphics processing unit (GPU), is a parallel vector calculator. Until recently, the GPU had fixed functionality – it took vertices in application world coordinates, converted them to screen coordinates and applied the appropriate color to the pixel. However, now there are two steps in the pipeline that can be programmed, ushering in a new set of algorithms that use the GPU for non-graphics calculations. To run an algorithm on the GPU it must be turned into a fragment program, often requiring that it be written differently than if it were designed for the CPU because some of the GPU pipeline is still fixed. The GPU deals in pixels, vertices and textures so an algorithm must be modified to fit this paradigm to benefit from the performance enhancement provided by the GPU's vector processor.

The GPU performs vector calculations much more efficiently than the CPU. This advantage stems from intense innovative pressure from the video game industry as well as the power of its specific design. Since the GPU does not have to handle general computation, its components can be optimized to calculate the specific vector equations involved in 3D graphics. Furthermore, today's GPUs have sixteen to twenty-four parallel pipelines per card to perform these calculations [8]. Due to their design optimization and parallel pipelines, GPUs can provide a considerable performance advantage. For example, a 3.06 GHz Intel Pentium 4 with Hyperthreading can perform six gigaflops with a memory bandwidth of 5.96 GB/s. In contrast, an ATI Mobility Radeon 9700 can perform 25.6

gigaflops with a memory bandwidth of 12.8 GB/s [9]. Further, each currently costs around $170.

This paper presents a vision for a system of systems that harnesses the power of the GPU to enable a simulation of human-in-the-loop UAV swarm control. Eventually, this vision could be expanded to manage UAV swarms in the field. This vision involves a swarm of semi-autonomous UAVs under the indirect control of an operator. This swarm will be controlled with an insect-inspired algorithm. Each UAV in the swarm will be small and expendable while still providing invaluable reconnaissance by being able to locate threats and targets with on-board sensors and computers. This reconnaissance will be displayed in a virtual world representation of the battlefield [10]. Further, the operator will directly manage a small number of large F-16 sized striker UAVs using the same virtual world. These strikers wait for orders to eliminate threats or targets. The operator could be deployed near the field in a vehicle to minimize communication delay with the swarm. The swarm would gather information about the battlefield with only minor input from the operator. The striker UAVs would then use this information to determine the best path to the desired target or threat set by the commander. Finally, the operator would monitor this striker and could take over control at any time.

Sound like science fiction? Although all of this is not currently technologically feasible, it is not far off the technological horizon. Prototypes and simulations of these future control systems can yield valuable insights on future operator interfaces. In fact, it is important that these future control systems be explored now, not only to spur UAV swarm development, but also to start training people on how to interact with swarms.

BACKGROUND

There are two main streams of research that are important to implementing the vision of an advanced swarm control interface: swarm control and battlefield visualization. This section first discusses battlefield visualization by covering prior work we have completed in this area such as the Virtual Battlespace and the VR-Aided Vehicle Teleoperation system. Then it covers a tested and validated insect-inspired swarm control method, Adaptive control of Distributed Agents through Pheromone Techniques and Interactive Visualization (ADAPTIV) [1].

Virtual Battlespace

In 2000, a research team at Iowa State University's Virtual Reality Applications Center (VRAC) began work with the Air Force Research Lab's Human Effectiveness Directorate and the Iowa National Guard's 133rd Air Control Squadron to develop an immersive VR system for distributed mission training called the Virtual Battlespace. The Virtual Battlespace integrates information about tracks, targets, sensors and threats into an interactive virtual reality environment that consolidates the available information about the Battlespace into a single coherent picture that can be viewed from multiple perspectives and scales [11]. Visualizing engagements in this way can be useful in a wide variety of contexts including historical mission review, mission planning, pre-briefing, post-briefing and live observation of distributed mission training scenarios.

The Virtual Battlespace immerses users in a virtual battlefield environment that provides them with greater context and awareness of the status of the units under their

control as well as the overall mission. By integrating UAV video feeds into this virtual environment, the virtual world can provide up-to-date access of the latest real time information from the vehicle. This virtual world, constructed from a mix of *a priori* information and real time sensor feeds, provides context to the operator. The result is a mixed reality system in which the real world video streams augment a dynamically constructed virtual world. Using real world data to augment the virtual world is an inversion of the more typical paradigm of augmented and mixed reality where virtual information is used to enhance real world data and imagery.

Our work with the Virtual Battlespace inspired us to begin creating a cohesive virtual world representing the status of real time and *a priori* information about an engagement [12]. Figure 2 shows the Virtual Battlespace environment displayed on a three-walled stereo projection system, the Baby-Cave, at VRAC. The Baby-Cave is powered by a cluster of six commodity Dell PCs running Linux. Since the Virtual Battlespace is developed on top of VRJuggler, it can be executed by a computer cluster to create a compelling virtual world. VRJuggler is an open-source software library that allows anyone who can write a graphics application to create a VR application complete with tracking, cluster capability and peripheral devices such as wands and pinch gloves [13].

The Virtual Battlespace is designed to aid its operator in understanding the complex interactions of units on the battlefield and in making effective decisions. There are many different streams of information that provide support for battlefield decision making. Some of these include radar and other sensor feeds, satellite imagery, communication links and weapons information. The Virtual Battlespace is designed to fuse multiple information streams and make them centrally available to command and control personnel. The goal of

this comprehensive presentation is to improve a user's ability to make effective and intelligent decisions [14, 15].

Figure 2. Battlespace environment

A stream of unit data to be displayed by the Virtual Battlespace can be generated by several diverse sources such as a simulated force generator like Joint Semi-Autonomous Forces (JSAF), or a live sensor such as a radar feed. The streams need not have a common format. JSAF created and monitored the bulk of the non-swarm battle participants in this research via a pre-recorded scenario. JSAF uses the Distributed Interactive Simulation (DIS) military communication protocol, and as a result, does not represent the most sophisticated military simulation software [16]. However, DIS is a sufficiently good format to create the proof-of-concept prototype of the Virtual Battlespace.

In the Virtual Battlespace, data streams are presented graphically to reduce the amount of textual information the commander is required to process. In other research, this

reduction was found to allow operators to focus more time on critical decisions [17] [18]. The graphical elements used to display the units (alternatively called entities) generated by the data streams are a major component of the system. They not only portray the physical attributes of entities in the Virtual Battlespace, such as relative position, orientation, status and speed, they also portray derived attributes such as prior paths or sensor and threat ranges. To maintain the system's flexibility with respect to the format of the input streams, the display of the data streams is separated from their management and from the base application, allowing integration of completely new data streams without having to rewrite large portions of the code.

Entity proxies, the graphical representation of units on the battlefield, allow the application to treat those entities the same way no matter what data stream they came from. Neither the graphics application nor the user need be concerned with the source of data being represented. This means that a proxy generated from a flight simulator stream can be displayed with the same graphical components as an entity generated from a DIS stream without the user having direct knowledge of the number of different information streams that are driving the system.

Another important aspect of proxies is that they can represent aggregates, or groups of entities. These aggregate objects combine the individual entity representations, such as a four-ship of F-16s, into one aggregate representation, such as a single plane symbol located at the four-ship's center, reducing information overload. In this way, aggregation simplifies a commander's view of a battle. The recursive nature of the proxy model allows aggregation at arbitrary levels by supporting aggregates of aggregates. This recursive aggregation is critical when viewing multiple battlefields in a larger theatre. Each battle could be

represented by a single aggregate until the commander visits that particular battlefield. Then the single battle aggregate could, for example, break up into aggregates of four-ships.

The operator interacts with the Virtual Battlespace and its unit proxies by using a wireless game controller. The controller used in this research was a Logitech Wingman equipped with a D-Pad, two analog sticks, four shoulder buttons (L1, L2, R1, R2), six buttons (A, B, C, X, Y, Z), a throttle slider and a start button. This input device was chosen for the Virtual Battlespace because game controllers are familiar to many people, especially young people. Because these people are the likely users of a near-future installation of the Virtual Battlespace, it seemed like the ideal interaction device. The operator uses the controller to navigate and turn on or off graphical information features. Some of these graphical features are shown in Figure 3.

Figure 3. Virtual Battlespace graphical features

The green wedge in Figure 3 represents the extent of the lead unit's radar. The operator might use this information to see when the unit can detect another unit. The red and white circles represent the threat range of a SAM site. The operator would use this visual information to know when units are vulnerable to attack. The blue wedges are aggregates of a four-ship of fighter jets. These aggregates leave a color-coded trail behind them to inform the operator of the path they took to arrive at their current location. All of these graphical features are designed to enhance an operator's situational awareness of the battle.

VR Aided Vehicle Teleoperation System

In 2002, the same VRAC research team began work on a new teleoperation control system. That system combined vehicle dynamics simulation, position and orientation tracking and a virtual representation of the operational environment to create a vehicle control station. That station provides superior situational awareness and vehicle control in the presence of signal lag [19] [20]. The primary components of this new VR aided teleoperation system are shown in Figure 4.

Using an interface, the operator controls a vehicle in the virtual environment displayed by the image generator. The operator's commands are sent to a dynamics simulation that uses these inputs to calculate the dynamic state of the virtual vehicle. The dynamic state includes information such as position, velocity, acceleration and heading. The state created by the dynamics engine is a simulated state, used to both position the virtual vehicle and provide a desired path for the teleoperated vehicle.

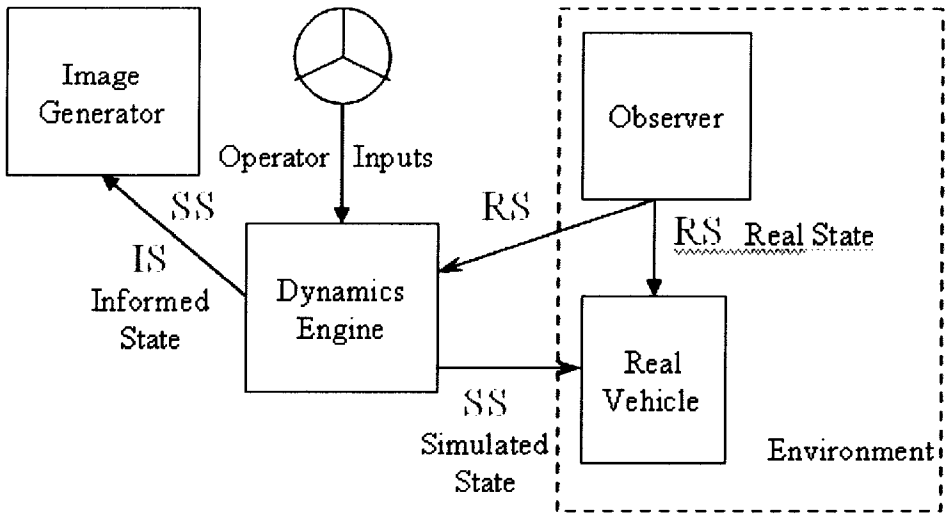

Figure 4. VR-Teleoperation general system model

As the teleoperated vehicle receives these simulated states they are synchronized to account for the lag and jitter generated by the communications delay. The vehicle uses these synchronized simulated states as a series of goal states. A simulation run locally on the vehicle determines the inputs required to get the vehicle to approach the simulated state from its current state. To calculate these inputs, the current state of the vehicle must be known. A tracking system, called the observer, provides this state information. The observer is responsible for reporting the vehicle's state information to the operator and the vehicle. The operator uses the reported vehicle position, corrected for lag and subsequent vehicle control, to visualize the likely future position of the vehicle. As shown in Figure 5, this predicted position is depicted graphically as a wire-frame box surrounding the virtual vehicle that grows and shrinks in response to the difference between the simulated state and the vehicle's projected state. This wire-frame envelope allows operators to adjust their control to obtain higher fidelity with the remote vehicle, closing the loop between the human and the computer

controlling the remote vehicle. This VR-Aided Teleoperation system could be used to operate the striker UAVs in concert with swarming reconnaissance UAVs.

Figure 5. Wire-frame envelope

ADAPTIV

Social insects provide a template for swarm control as they can have a complex aggregate behavior despite being simple creatures. An individual termite does not know engineering or physics or even what its fellow termites are doing, but a swarm of tropical termites can build complex multi-level mounds that can be five meters tall and weigh ten tons [21]. These mounds have impressive overall rigidity, are made from a material that is fire resistant, have a regulated interior temperature and contain enough rooms and passages to house the brood and all of its food reserves.

Ants provide another example. Individual ants gather together to perform complex social tasks such as aphid farming. These farming ants protect a herd of aphids (smaller insects) from predators so they can milk them of their honeydew [22]. Furthermore, most ant species form invisible roadways called "ant highways," where columns of ants follow one

another without the aid of street signs or painted lanes. These complex behaviors are accomplished via pheromones.

Pheromones are chemicals produced in different flavors. Insects release them into the air to use as markers, with each flavor conveying its own message. One chemical compound might signal food while another might signal danger. Insects use pheromone flavors to communicate with their fellows. If an ant finds food, it drops a pheromone that tells other ants, "food this way." As this pheromone reaches the receptors of other ants, they can use it to find the food source.

Pheromone-based swarm control algorithms for man-made units have been explored by other researchers. Most pheromone-based algorithms use the concept of digital pheromones, data markers that are passed between units in a swarm with a network instead of being carried by the wind. A digital pheromone contains a data field that can be used to differentiate its flavor, analogous to the way the chemical composition of biological pheromones can be used to determine their flavor. The unit's pheromone receptors are packet readers. In nature, the environment facilitates pheromone movement and interaction; in computer swarm control algorithms, some other entity must facilitate these pheromone activities. In this case, a global data structure serves that function. This data structure, a grid of all present pheromone flavors and intensities, called the pheromone field, is considered to be external to the units in the swarm.

ADAPTIV is a pheromone-based swarm control algorithm that uses a hex grid for its pheromone field. Each cell consists of a set of numbers that represent the magnitudes of the different pheromone flavors. The flavors represent the presence, as detected by the unit, of different items of interest, such as threats or targets. Alternatively, the flavor can be used to

enable swarm dispersion. This type of pheromone flavor is called UAV repulsion pheromone and is dropped at the current location of the unit in the swarm. The threat and repulsion pheromones are unattractive while the target pheromone is attractive.

The ADAPTIV algorithm computes this pheromone field through the repeated application of a three step procedure:

- update the pheromone field
- propagate pheromones
- move the units in the swarm

The first step updates the strength of the pheromones at each field cell with Equation 1. It is a vector equation with each of its components corresponding to a unique pheromone flavor.

$$\boldsymbol{S}(t+1,p) = E^{*}\boldsymbol{S}(t,p) + \boldsymbol{R}(t,p) + \boldsymbol{Q}(t,p) \quad (1)$$

Equation 1 states that the pheromone strength (S) at cell location p at time $t+1$ is equal to the S currently there (at time t) times the evaporation coefficient (E) plus any pheromones added to this cell from units in the swarm (R) at time t plus pheromones that have propagated from cell p's neighbors (Q). E is a constant between zero and one and it represents the percentage of pheromone that remains within a cell after evaporation. This evaporation is important as it makes pheromones linger for only a short period after the target of interest is gone, automatically keeping the known state of the world up to date.

Two variables, R and Q, must be known in order to calculate S in Equation 1. When a unit finds something in cell p it places a pheromone of the appropriate flavor in the p location of R. R is cleared each time since it represents new pheromones input into the system. Pheromones dropped in R are added to the current value making pheromones of the same flavor at the same location additive. With R in hand, only Q is left unknown. The calculation of Q is done in step two of the algorithm by using Equation 2.

$$\boldsymbol{Q}(t+1,p) = \sum [\ F/N(p')*(\boldsymbol{R}(t,p')+\boldsymbol{Q}(t,p'))] \quad (2)$$

Equation 2 describes pheromone propagation as the weighted sum of pheromones from each cell's neighbors. The p' in Equation 2 denotes the location of a nearest neighbor of cell p. $N(p')$ is the number of nearest neighbors that cell p' itself has. In the case of a non-ending hex grid $N(p')$ would always be six. F is the propagation coefficient with possible values ranging from zero to one. This coefficient is the percentage of pheromones at a cell that will propagate to its nearest neighbors. In Equation 2, Q represents new pheromones added to cell p by its neighbors. The value of Q would then be the sum of the individual contributions of each of p's nearest neighbors, hence the sum in Equation 2. The part inside the sum is the contribution of each nearest neighbor. This contribution, dampened by the propagation coefficient F, is split equally among its nearest neighbors. As a result, $F/N(p')$ of the newly input pheromones (given by R and Q) in cell p' propagate to cell p.

The third step of ADAPTIV is to move the units in the swarm based on the information encoded in the pheromone field. This step is accomplished with roulette selection. Roulette selection is a decision method used commonly in the field of

evolutionary computing and genetic algorithms [23]. The core idea of roulette selection is to choose randomly among the set of possible choices, but weighting that choice to give "better" choices a higher chance of being selected. As explained shortly, optimal performance comes from this type of probabilistic preference toward better choices and not from always selecting the best choice. Figure 6 shows two cases of a roulette selection.

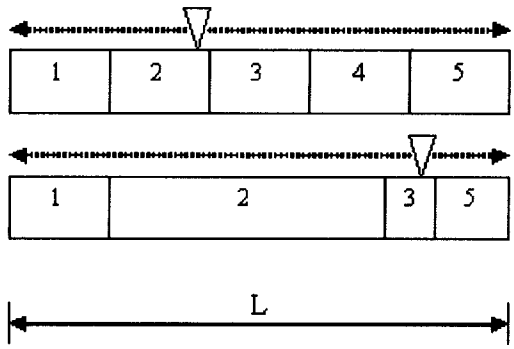

Figure 6. Roulette selection examples

In computer algorithms a random choice is made utilizing a random number. Often this number is a decimal value between zero and one. If instead, the random number varies between zero and L, then the range of the random number (L) would be the total bin length in roulette selection. Each choice is given a bin with a length proportional to how good a choice it is based on rankings determined before the start of roulette selection. If all N choices are equally good, then each bin would be of the length L/N. This scenario is illustrated with the top bar in Figure 6. All five bins in the top bar are the same length, indicating that each one of the five choices is as good as any other. However, if the N choices are not equal, then the bin lengths will vary with better choices having longer bins. The second bar in Figure 6 illustrates this scenario. Notice that the bin for choice 2 is much larger than the bin for

choice 3. This means that choice 2 is better with a length difference proportional to how much better it is. If X_i is the measure of how good a choice is and X is the sum of all the X_i's, then the length of a bin for a particular choice i is X_i*L/X. If X_i is zero, then the bin length for that choice is zero. This happens when a choice is deemed to have no value for selection. Choice 4 in the second bar at first appears to be missing, but actually is a choice with a zero bin length.

Once all the bin lengths are calculated, a particular bin must be selected. One might be tempted to pick the longest bin because it represents the best choice. However, doing this could lead to situations where the algorithm gets stuck in a local optimum and would have no chance of escaping it to find the global optimum Importantly, roulette selection is not guaranteed to pick the best choice and it can even choose the worst choice. By not always selecting the best choice, roulette selection algorithms can escape local optimum to find the global one [23]. However, better choices have longer bin lengths and will be chosen more often than those with smaller ones.

In ADAPTIV, the bin lengths (the "goodness" of a choice) are determined by the local pheromone levels. There is one bin for each of the cells' nearest neighbors and that bin is made longer by the magnitude of attractive pheromone flavors and shortened by the magnitude of repulsive pheromone flavors. In this way, the swarming units explore a space somewhat randomly with a preference for attractive cells. ADAPTIV was tested in 2002 in several simulated scenarios and was found to perform well in controlling a swarm of virtual UAVs [1].

ADAPTIV is not the only swarm control algorithm. Icosystem and the Air Force Research Lab created an agent-based swarm control algorithm that does not use the idea of

pheromones [24]. Another algorithm developed by Lookahead Decisions Inc. uses a decision tree (a common construction in artificial intelligence and machine learning) to control a virtual UGV (unmanned ground vehicle) swarm [25]. Yet another method developed at Wright-Patterson AFB uses a particle system with decision rules to model swarm behavior [26]. Another entire set of swarm control algorithms use the idea of a potential field. These algorithms model the units in a swarm as charged particles and endeavor to create a field to influence their behavior. Drawing on the analog with electrostatics, this field is called a potential field since it is used to move "charged particles" through its space in a predictable way. There are dozens of potential field control examples as this has been an accepted control method in the field of robotics for years [27].

Although pheromone-based systems in general and ADAPTIV specifically are not the only choices for UAV swarm control, I chose ADAPTIV for my research because it is both simple and robust. It has performed well in its recent tests (2002) and is amenable to calculation on parallel computers. Its main flaw is that it is computationally expensive. In 2002, Altarum was able to control over 100 UAVs in real time with ADAPTIV, but the ability to control more would require more powerful or clustered computers. I saw a way to improve the performance of this effective algorithm.

RESEARCH ASSUMPTIONS

This dissertation presents a software simulation of a UAV swarm that makes certain presumptions about the advancements in UAV technology that will be needed to make large UAV swarms possible. My research presupposes that these advancements, or swarm qualities, are technologically feasible, and therefore includes them as assumptions. Below is a description of these various assumptions followed by evidence that these problems are being actively researched. The section concludes with a brief discussion on why my research is important to conduct now even if the actual swarm hardware isn't currently available.

Near-Future Swarm Assumptions

A goal of my research is to create an interface that will allow one operator to manage the progress of a swarm of UAVs. These unmanned vehicles are more advanced than current UAVs and are not currently technologically feasible. These advanced UAVs will be called the near-future UAV swarm. This near-future swarm should be deployable in the next ten or fifteen years. For these swarms to become viable, technological advances must be made that will validate three large assumptions:

- They will fly themselves to a set of given coordinates while avoiding collisions with other members of the swarm.
- They will "detect" threats and targets through the use of AI, sensory information and image processing. This detection need not be 100% accurate.
- They will be connected to each other and the swarm commander via an ad-hoc wireless network.

Auto-Flight

True autonomous flight requires no input from human operators as all decisions are made by the machine as it travels along its path. The area of autonomous flight, and its larger research area autonomous movement, has a long and rich history. From early robotics research to the DARPA Grand Challenge, people have been trying to make vehicles and robots that can navigate by themselves. Despite its history, this area of research is by no means closed as there are still considerable challenges to address.

Luckily, my research does not require complete autonomous movement. Rather it assumes that UAVs in the swarm will be able to fly from one set of coordinates to another given set without colliding with another unit in the swarm. This movement task is more restrictive than complete autonomous movement in that it requires a goal state (a new set of coordinates) to be given to the vehicle. The vehicle must still determine what control inputs it must use to get to the goal state, but is freed of the task of figuring out where to go. This task elimination is critical to near-term deployability because as the DARPA Grand Challenge showed, there are still vital advances required in the areas of AI and sensors for truly autonomous travel to be a reality [28]. However, even though the UAVs do not have to decide their ultimate destination, they still must determine the best path to get from their present location to their designated destination. Further, they must avoid colliding with one another along the way.

Several researchers have been working on these problems. Singh et al. at the Carnegie Mellon Robotics Institute are developing an algorithm to allow a planetary rover to explore alien worlds nearly autonomously [29]. Their work stresses the idea of using a two-layered approach to control. First, the rover must react quickly to information from onboard

sensors about local features such as obstacles. At the same time, the rover must make its way to its goal. Clearly, it is more important to avoid obstacles than figuring out the best path to the goal. However, if the rover is too conservative and is not keeping track of and trying to reach its overall goal it may never arrive. Singh's algorithm attempts to solve this limitation by utilizing traversablilty measures on nearby obstacles (local navigation) and a human-planned path developed from a-priori information that is modified during operation by the rover (global navigation). This two granularity approach to vehicle control allows the processor to focus most of its capability toward avoiding obstacles (local navigation) and use its spare capability to continually update the best path to the given goal (global navigation).

Recently, Reza Olfati-Saber at Cal Tech has introduced new algorithms and theories for multi-agent flocking [30]. Flocking behavior is similar to swarming behavior except that all units in a flock share one global goal while units in a swarm typically have individual goals. In short, a flock is more structured than a swarm. Olfati-Saber's research provides a means to perform collision avoidance with neighboring units. Once again, this research promotes a two tiered approach to multi-agent control. This flocking method utilizes three types of agents for control. The first two types of agents are used in local navigation and include both α-agents (the units in the flock) and β-agents (obstacles). The basis of the local navigation is to take the path that leads each α-agent away from other α-agents and β-agents, thereby avoiding collisions. However if this local navigation were the only influence (the (α, α) protocol), the α-agents movements would be chaotic. It is the influence of the global navigation (via a λ-agent) that brings order to the system, because the λ-agent (the goal) attracts units, which results in flocking behavior as Reza Olfati-Saber explains:

> First, we discussed that the use of the (α, α) protocol alone generically leads to regular fragmentation. Then, Algorithm 2 was constructed from the (α, α) flocking protocol by adding a navigational feedback. This navigational feedback takes the group objective into account using a static/dynamic λ-agent. It was demonstrated that Algorithm 2 generically leads to flocking [30].

The λ-agent's position is set a-priori and is the same for all units in the flock. For swarm control to be effective using this paradigm, a dynamically generated unique λ-agent for each unit must be programmatically determined. This dissertation provides an example of how to determine similarly generated goal agents.

Automatic Threat and Target Detection

Automatic target recognition (ATR) is a research field that has matured over the last five years. While still not foolproof, algorithms exist to search an image for a predefined target. Companies such as C&P Technologies sell software products for performing ATR. An excerpt from their website claims that their "STAP toolbox can be used currently to detect targets and estimate the parameters such as angle, Doppler, and range" [31]. Their ATR toolkit uses the latest pattern-matching techniques to determine how well an object in question matches the pre-defined target specification. So for this software to be effective, the target must show up in the sensory image similar, in some way, to how it was defined. Even though multiple definitions for the same target (as viewed from different angles in different lighting perhaps) can be specified and checked, an exhaustive set of patterns to match with targets is too costly and time intensive to create and maintain. As a result, research into ATR is still needed to make the systems more generally effective. Many corporations hope to use advanced AI and data mining to relax the required precision of the pattern image to the actual

target image while still being able to discern objects that are similar to, but not, the target. One such company, Rockwell-Collins, has allocated substantial funds to improve the ATR capabilities of its current software offerings. In August 2004 it won the US Army MCAP III contract to create, among other things, improved ATR software [32].

Academia is also actively researching ATR. A research team at the University of Southern Florida working on an Army Research Lab grant called "Robust Recognition of Interesting Objects in Images" created an algorithm that uses fuzzy logic and computational geometry to improve target detection rates [33]. Their approach used one of two relatively simple image processing techniques to find regions of interest (ROI) where targets are likely to be located within an infrared image. One technique used edge finding on the infrared image while the second used simple intensity thresholding. These two methods could be used to find ROIs because targets were either likely to show up as really hot (tires, engines) or really cold (metal surfaces reflecting most of the sun's rays). These two qualities of likely targets means that their thermal signature is different from their surroundings – making both intensity thresholding and edge finding promising means to find targets. Once the ROIs are identified, they use a fuzzy logic algorithm called 2rfcm to form logical clusters and decide if the ROI contains a target. These logical clusters attempt to identify a group for each pixel within an ROI. For example, the decision tree could decide that certain pixel parameters likely mean that it is grass or that it is a tree.

Arguably the most important logical cluster, the target, is found using fuzzy logic rules to pare down the relatively large number of ROIs. The 2rfcm algorithm applied rules such as "targets are often near tree/grass edges in an attempt to be hidden from view" and "targets are generally too small to be clustered individually, and often target pixels are

assigned to both tree and grass clusters." Their procedure achieved partial success. Their paper emphasized the fact that there is still much work to be done in this area to get a perfect or near-perfect automatic target detection system. They conclude that in trying to "recognize every target and only recognize targets as objects of interest ... [the] goal is only approximately met" [33].

Connectivity

The third quality of a near-future swarm is the ability to create a communications network between all the units in the swarm. One of the first ideas is to create a special router-like unit that manages the connection with the human operator and then sends network communication packets to the unit it is meant for. This structure parallels the familiar land-based internet connectivity in our homes and places of work. While this method of communication is well understood, it has one serious flaw in swarm communication. In a swarm no unit is supposed to be more important than any other. This quality is clearly broken by a special router unit. The problem with having such a unit in hostile territory is clear. If the special unit were disabled, all of the units would lose connection with the controller and also likely be lost. Clearly, the swarm must be able to lose any member of its rank and still be able to function. This leads to the more complicated peer-to-peer networking often called ad-hoc networking.

Ad-hoc networking is an area of active research. Dozens of academics and corporations are working on the many challenging problems posed by ad-hoc networks. One such research group, led by Timothy Brown, at the University of Colorado has created a prototype ad hoc network among small UAVs using IEEE 802.11b (WiFi) radios [34]. They

are currently able to maintain a network with up to 8 low-cost UAV nodes. One big problem is that each unit of the swarm must use the same communication channel to listen to requests from any number of units all without the help of an overall scheduler (the router). This means that a communication protocol must be developed to allow only one sender to put information on a unit's communication channel at a time all while maintaining acceptable communication speeds. The Colorado group's test network only suffered a 10% performance loss due to the efficiency of their monitoring module [34]. However, going beyond 8 nodes reduces this efficiency and increases the likelihood of a message not reaching its target in a reasonable time or possibly at all. Clearly, advances must be made for large scale swarm ad-hoc networks to be feasible.

Mario Gerla et al at UCLA have worked with Xiaoyan Hong at the University of Alabama to explore one potential way to alleviate the issue of packet management in ad-hoc networks. In this work, they expand upon their LANMAR algorithm to create motion groups [35]. Commonly, a global ad-hoc network is broken into sub-groups of units that communicate locally with every other unit in their group and talk to units in other groups through a designated group leader. To avoid the fragility of having a special unit (a fixed leader) as described previously, each unit is capable of being a group leader and the leaders are chosen dynamically and can change over time. Typically, groups are formed by geographical proximity, i.e., each unit forms a group with other units close to it. The definition of "close" is typically two or three communication hops. Wireless communication has a maximum effective range, as users of laptops can attest to. A communications hop is the maximum distance a unit can be away from another given unit and still be within direct communication. With this limitation in mind, creating groups that are close to each other

makes sense, because the group leader will need to act like a router, managing the communication channels for the units in its group. This requires that the UAV's in a group remain in close proximity (only a few jumps apart) to achieve acceptable performance.

Group formation, however, takes time and puts stress on the network. As a result, limiting the amount of group formation and membership change becomes critical in maintaining acceptable performance. This is the main problem with proximity group formation. In many cases units travel at different speeds and in different directions. A group may be formed from three units all located at coordinates (x,y). However this group could break up quickly if one unit is stationary, one is flying east and the other is driving west. The units making up a LANMAR network have to maintain a similar position, velocity and heading in order to continue to act as a group. Swarming UAVs will function well under these parameters because they will travel at similar speeds and will be attracted in small groups to similar locations of interest. LANMAR was found to perform better in dynamic network communication than simple proximity grouping and promises to be useful in UAV swarm inter-unit communication.

Control Center Qualities

My research makes a few assumptions about the likely future UAV control station. The following assumptions about the characteristics of the station were generated mostly through discussions with engineers at Rockwell-Collins. First, it is likely that the UAV control station will need to be geographically close to the UAV swarm. Proximity to the swarm will reduce the time it takes for communication signals to get between the operator and the swarm. Second, the control station will be placed within a mobile platform such as a

HUMVEE or helicopter to allow the operator to move with the swarm or even between swarms. Additionally, the military prefers mobile stations when possible since war is a fluid affair. One immediate consequence of this assumption is that the interface will have to fit within a restricted physical space.

While the future swarm described is not currently technologically feasible, it is not so far off that it would not make sense to begin developing prototypes and simulations of these future control systems. Further, even if the assumptions about the nature of the future control stations are false, my research will still provide a useful base for further development. In fact, it is crucial that future UAV swarm control systems be explored now, not only to spur UAV swarm development, but also to start training people on how to interact with swarms. This need for a prototype system led directly to my research and the development of an approach for swarm management.

METHODS

A two pronged approach was developed in an attempt to create an interface that allows one operator to manage the progress of a swarm of UAVs while maintaining battlefield situational awareness. The first part of the approach is to use a pheromone-based algorithm (ADAPTIV) to provide the swarming units with goal positions. In this way, the commander need not be concerned with directly controlling any of the swarming units. The ADAPTIV algorithm is implemented on the GPU to gain performance improvements. The second part of the approach is to integrate swarm management controls with the Virtual Battlespace. This fusion includes creating interfaces to manipulate parameters in the ADAPTIV algorithm as well as representations for the swarm within the virtual battlefield visualization to put the swarm in the context of the larger battlefield. This section first reviews the entire system and how all the pieces interact. Then it covers the actual GPU implementation of the ADAPTIV algorithm, the measurement of GPU ADAPTIV performance, the dynamics simulation, the modification of Virtual Battlespace to interact with the swarm and finishes with discussion of the Swarm Manager and Menu Manager.

The Overall System

Figure 7 shows all of the major components that make up my research vision for a more effective swarm management system. Two computers drive the prototype system used in my research – the Swarm Computer and the Virtual Battlespace Computer. The Swarm Computer is responsible for running ADAPTIV, simulating the dynamics of the aircraft, and simulating their detection of threats and targets. The Virtual Battlespace Computer is

responsible for running the virtual world representation of the battlefield and providing all of the interfaces needed for the operator to manage the swarm.

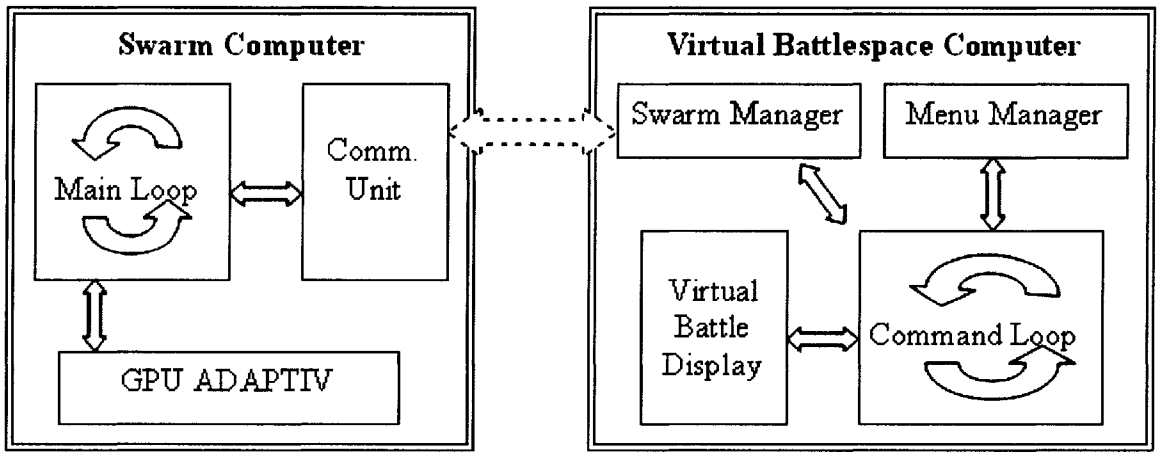

Figure 7. Overall system diagram

The Swarm Computer in the prototype system was the author's laptop. It is a Dell Inspiron 9100 that runs Windows XP with a 3GHz Pentium 4 processor, 1GB of RAM, and a 128MB Mobility Radeon 9700. The computer runs the GPUSwarm application to control the swarm. GPUSwarm is a C++ OpenGL application that runs the GPU ADAPTIV implementation on the GPU, simulates the movements of the UAVs as they travel to the goal positions assigned by the GPU ADAPTIV algorithm, simulates the detection of threats and targets, and maintains both a TCP and UDP connection with the Virtual Battlespace Computer. The specifics of the first two items are described later in this section.

GPUSwarm reads in two files to configure both the swarm and the battlefield respectively. The first configuration file begins with the number of units in the UAV swarm and then defines properties of each unit of the swarm including:

- initial position, velocity, acceleration and heading.
- sensor radius, sensor field of view, low sensor detection percentage and high sensor detection percentage.
- maximum turning rate, maximum speed, minimum speed and cruising speed.
- repulsion pheromone strength.

If several units in the swarm start at the same position with the same initial state, they can be listed in the file as a group, greatly reducing the size of the file and making it easy to create large swarms. In my research, for example, the test cases involving a swarm with 400 units had four groups of 100 that were released at four different places at the start of battle. The second configuration file defines the properties and locations of threats and targets that the swarm can find within the battlefield. This file begins with the number of threats and targets and then describes properties such as:

- initial positions and velocities of the threats and targets.
- the low and high hit probability, time between shots and lethal range of threats.
- the strength of pheromones released by the threats and targets.

With these two configuration files, the GPUSwarm application is able to create the battle scenario. The main loop of the Swarm Computer first moves the UAVs and then performs target and threat detection. On this step, each swarming UAV is visited and if any target or threat is found to be within its sensor range and field of view, it has a chance to detect the threat or target. To make this searching step more efficient, the targets and threats are stored in `std::map` data structures (red-black trees) that have the cell number currently

containing the threat or target as their key. Cell numbers start at one at the upper left and are increased by one along each cell of the same row. At the end of the row, the next cell assigned is the first element of the next column. With this setup, the UAV's position can be used to query for any threats in cells that are within its sensor range. These cell numbers are passed into the maps which efficiently return any threats or targets in the cell because the map is a red-black tree.

Once a threat or target is found to be in the sensor range, the UAV has a chance to detect it. The detection probability is a linear interpolation between the low and high detection rates. The distance and detection rate are inversely related; the closer the distance is to the maximum sensor range, the closer the detection rate is to the minimum. Finally, if a random number between zero and one is less than or equal to the detection chance (also a number between zero and one), then the target or threat is detected and pheromones of the appropriate flavor will be dropped in ADAPTIV steps one and two. Additionally, if a threat is found to be in the sensor range and is able to fire (the time between shots has elapsed), the threat has a chance to destroy the UAV. Once again, the shoot-down probability is inversely related to the distance between threat and UAV with the farthest distance correlating with the threat's low hit chance. If a UAV is shot down, it still gets to drop its pheromones one last time.

After the main loop completes target and threat detection, the GPU ADAPTIV algorithm is run, thereby generating goal states for the swarming units. These goal states are then used to change the UAV dynamics controls such as heading and speed as described later in this section. Finally, the main loop sends any needed updates to the Virtual Battlespace Computer and then cleans up any shot-down UAVs. A separate thread within the

communication unit listens for information from the Virtual Battlespace Computer and updates the main loop in a thread-safe way when needed. The communication unit of the Swarm Computer is connected to the Swarm Manager of the Virtual Battlespace Computer.

The Swarm Manager uses the Virtual Battlespace command loop architecture to communicate the status of the swarm with the rest of the application. Further, when the user wishes to change swarm behavior, it is done through menus in the Menu Manager, which sends requests via the command loop to the Swarm Manager. The Swarm Manager can then pass these requests to the Swarm Computer where they can be executed. These menus and the swarm itself are drawn within the virtual environment created by the Virtual Battlespace application (see the Background section for further information). The Virtual Battlespace Computer is either a single Dell PC (for desktop implementation) or a cluster of six identical Dell PCs running three screens of a large VR display in stereo. In both cases the computers are Dell Precision 670s running Red Hat Enterprise Linux with 2GB of RAM, dual 3.2 GHz Intel Xeon Processors, and a 512 MB NVidia FX4400 graphics card.

The rest of this section is describes the various components of the prototype system outlined above. It should be mentioned, however, that since this system is a prototype, it contains components that would not be needed in a real system and is missing some that would be. In a real system with physical UAVs attached to the swarm controller, it would be unnecessary to simulate the UAV's dynamics to generate a position since that information would be sent by the UAV via GPS. However, a dynamics simulation would probably still be needed for dead-reckoning between updates. More importantly, the computationally expensive step of "detecting" threats and targets would be done in a distributed fashion by on-board UAV computers. Each UAV would use its on-board sensors and computer to

decide if there is a threat or target within its sensor range (as is discussed in the Assumptions section). One important omission of this prototype system is security. Any real control system would need to send encrypted messages, but this is an unnecessary complication for a proof-of-concept prototype.

Converting ADAPTIV to the GPU

To run an algorithm on the GPU it must be written differently than if it were designed for the CPU, because some of the pipeline is still fixed. The GPU still deals in pixels, vertices and textures so an algorithm must be modified to fit this paradigm. GPU Gems covers the differences between GPU and CPU programs in great detail, but I shall summarize some of the main points [9]. First, to gain benefit an algorithm must be parallelizable – that is, it must contain several repeated calculations that are independent of one another. ADAPTIV's two pheromone field calculations (Eq. 1 and 2) are examples of parallelizable equations. Second, the data must be stored in textures, so rectangular grid data structures are preferable. ADAPTIV uses a hex grid which does not match the rectangular grid of textures. A simple change from six nearest neighbors to eight allows ADAPTIV to be modified for use on the GPU. Third, vectors represent colors. A texture is simply an array of colors. A color is generally represented with three or four numbers called channels: one for red, one for blue, one for green and possibly one for transparency (alpha). This color representation is referred to as RGBA color. Since the GPU can put only textures in its memory, it can store arrays of four numbers per cell. This setup works well with ADAPTIV because each color channel can represent a particular pheromone flavor.

To gain the GPU performance advantage, the ADAPTIV algorithm has to be converted into a form acceptable to the GPU. The first step is to represent pheromone fields with textures. The different color channels in this texture represent different pheromone flavors; red for threats, green for targets and blue for the UAV repulsion pheromone. Figure 8 shows a pheromone field as a texture with the flavors color-coded as previously described.

Figure 8. Example Pheromone Field

By representing the pheromone field with a texture, Equations 1 and 2 can be turned into fragment programs. Equation 1 written as a fragment program in the GLSlang language is shown below.

```
uniform sampler2D PherField;
uniform sampler2D PherPropField;
uniform vec4 PherEvap;

varying vec2 TextCoord;

void main(void)
{
      gl_FragColor = texture2D(PherField, TextCoord).stpq * PherEvap
                   + texture2D(PherPropField, TextCoord).stpq;
}
```

The command `texture2D` simply reads the value of the texture given by the first argument at the location given by the second. It returns a four component RGBA vector. Notice that R is missing from the code block. This is because the pheromones input by units at this time step are not saved in a texture like S and Q. Instead the R pheromones are taken into account by additively drawing points of the appropriate color (based on the pheromone flavor discovered by the UAV at that location) after the fragment program has been run. GLSlang fragment programs like the one above are run from within an OpenGL graphics program. The execution of the fragment program is triggered by the CPU within the OpenGL rendering pass. In this case, the application first draws a screen-filling polygon with the fragment program enabled. This execution fills the OpenGL frame buffer with a screen filling texture containing colors corresponding to the pheromone levels E*S+Q. We are still missing the R pheromone contributions. So we next set OpenGL to blending mode. This means that anything now drawn gets added to whatever was already there instead of replacing it. Then appropriate colored points are drawn for each UAV in the swarm, thereby adding the R pheromones. To complete the calculation, the frame buffer is rendered to the texture containing S.

The second ADAPTIV algorithm step, pheromone propagation using Equation 2, is completed in a manner similar to that of the first step. First, the previous values of Q are put in the frame buffer by drawing a screen-filling polygon with the Q texture. Points are drawn where the UAVs drop pheromones. Then the frame buffer is copied to texture containing Q. This texture, which contains R(t,p') + Q(t,p'), can now be used to calculate Q(t+1,p) as described by Equation 2 using the fragment program below on a screen-filling polygon.

```
uniform sampler2D Q;
uniform vec4 F;
uniform float delR;
uniform float delC;

varying vec2 p;

void main(void)
{
  gl_FragColor = (texture2D(Q, p+vec2(-delC, -delR)) +
                  texture2D(Q, p+vec2(0, -delR))     +
                  texture2D(Q, p+vec2(delC, -delR))  +
                  texture2D(Q, p+vec2(-delC, 0))     +
                  texture2D(Q, p+vec2(delC, 0))      +
                  texture2D(Q, p+vec2(-delC, delR))  +
                  texture2D(Q, p+vec2(0, delR))      +
                  texture2D(Q, p+vec2(delC, delR))    ) * F * 0.125;
}
```

As mentioned above, the texture Q really contains Q+R and all neighbors are considered to have eight nearest neighbors so N(p') is 8 (hence the multiplication of 0.125). Since N(p') and F are constant, they can be taken out of the sum as in the program. The purpose of `delC` and `delR` is to gain access to the values of a cell's nearest neighbors. These values are constants related to the texture size (they are in fact the reciprocal of the texture size). N(p') is only constant for internal cells, those on the border of the texture can have other values of N(p'), namely five or three. However, if the borders are declared off-limits to UAVs, threats and targets, then the complications at the border can be ignored.

The third and final step of ADAPTIV is to use the pheromone field to move the units in the swarm. To emulate nature correctly, this decision metric cannot be complicated. After all nature suggests "that stupid things swarm and smart things team" [36]. To mirror Nature's simplicity, each unit in the swarm uses roulette selection to determine where to go next. Despite the trend toward simplicity, this step involves a more complex fragment

program. This program is too complex to embed in the text, but its functionality is described below.

To accomplish roulette selection, a point is drawn for each UAV with the roulette selection fragment program active. These points are drawn in a horizontal line starting at the upper edge of the screen and then proceeding to the next line if necessary. This setup allows for efficient collection of the results since only a small subset of the frame buffer needs to be retrieved after running the fragment program. A unique random number, UAV sensitivities, and nearest neighbor cell positions are sent to the fragment program as vertex attributes. The main task of the roulette selection program is to calculate the length of the bin for each of the eight choices (each of the cell's nearest neighbors) and then pick one. It accomplishes this task by using the color of S at each nearest neighbor of the UAV to create a corresponding bin length. Each UAV maintains its sensitivity to each pheromone flavor in a vector called PherSen which is passed to the roulette selection fragment program. Since red and blue channels represent repulsive pheromones, the scalars in those two components of PherSen are negative. A direct result of this is that cells that contain more green than red or blue will be considered "better" choices. However, both the amount of each color present and the magnitude of the components in PherSen determine how much "better" a nearest neighbor cell is over the others. Specifically, the length of each of the eight bins is determined by the dot product of the color vector at a cell with the PherSen vector. As a result, the larger the magnitude of the components of PherSen, the longer (or shorter if the number is negative) the bins get. Additionally, there is a constant value called the "shift" added to each bin length. The purpose of the shift is to make cells empty of pheromones (black cells) somewhat

attractive. Without a shift, black cells would have a bin length of zero and have no chance of selection.

With all the bin lengths calculated, roulette selection can be performed. The random number between zero and one is multiplied by the total bin length. Then, the fragment program selects the first bin it finds whose cumulative length (based on the sum of all the bin lengths that came before it) exceeds the random number. This selection is accomplished with the GLSlang `step` function. The location of the cell whose bin was chosen is then returned in the red and green channels of the pixel. After all selections are done, a rectangle containing the new UAV goal positions is read from the frame buffer. These positions are assigned to the corresponding UAV. The UAVs then try to reach the positions, and are thus controlled by the pheromone field.

Measurement of GPU ADAPTIV Performance

In order to determine how effective the GPU conversion of ADAPTIV is, a plan to measure its performance is required. The algorithm performance was split into two categories: speed and swarm behavior. Speed performance simply involves timing how long each step of ADAPTIV takes to complete. Recording the time to complete the algorithm steps allows for a direct comparison of the performance of the CPU and GPU ADAPTIV implementations. This type of analysis was performed and is detailed in the Results section under the "GPU ADAPTIV vs. CPU ADAPTIV" subheading.

The second type of performance, swarm behavior, is harder to measure. The goal of a behavior measurement is to gain an understanding of how changing the inputs of ADAPTIV (pheromone sensitivities, evaporation and propagation) affect the behavior of the swarm.

This connection between the ADAPTIV inputs and swarm behavior must be explored to allow user modification of swarm behavior. To understand this connection, output data was recorded at each time step. This output data consisted of the number of remaining UAVs, the percentage of threats and targets found, and the coverage of the UAVs. Of the output data, the UAV coverage is the only one that requires further discussion. The purpose of measuring UAV coverage is to determine how well the swarm is exploring the space. My research would like to make three swarm behaviors available to the user and they are aggression, care and exploration. It was the early hypothesis that cases with high percentage of threats and targets found would also result in fewer remaining UAVs. As a result, it seemed likely that cases with the highest detection percentages would represent aggressive swarm behavior while cases with the highest remaining number of UAVs would represent the careful swarm behavior. This left no mechanism to determine which cases led to the most exploration. Thus, the idea of UAV coverage was added to the output.

Coverage is not as simple to measure as detection percentage or number of remaining UAVs. The idea of coverage is less concrete. For instance should coverage at any time in the simulation be the percentage of cells that are currently occupied by UAVs or should coverage be the cells that are within the sensor range of UAVs at that instant? Or is it shortsighted to only include cells covered in the current time step and ignore those that were covered in the last one? A space is considered covered in my research if a UAV currently occupies it. However, if a cell was recently visited, it is considered partially covered. Instead of coverage being a Boolean value, it takes on a continuum of values that represent how recent the coverage was.

A fragment program is used to measure coverage primarily because it is another cell-wise operation within the pheromone field and those types of calculations were found to be faster when done by the GPU. The code block below shows this fragment program.

```
uniform sampler2D CoverField;
varying vec2 TextCoord;

void main(void)
{
      // Get the color at this cell
      vec4 color = vec4( texture2D(CoverField, TextCoord).stp, 1.0);

      // See if this place has been visited - blue value > 0
      float visited = step(1.0/300.0, color.b);

      // Subtract 1/256 from the blue component
      color.b -= 1.0/256.0;

      // Clamp the blue component to 1/256 to reserve zero for
      // never visited sites
      color.b = clamp(color.b, 1.0/256.0, 1.0);
      color.b *= visited;

      gl_FragColor = color;
}
```

This program is executed in manner similar to the pheromone field fragment program described in the methods section. First, the coverage field from the previous frame is drawn on a screen filling polygon. Then, a blue point is drawn where each UAV is located and the resulting frame buffer is saved to the coverage texture. Finally this new texture is applied to a screen filling polygon with the fragment program active. The program itself gets the color at the current cell and then checks the blue component to see if this cell has ever been visited. It accomplishes this task by checking to see if the blue value is greater than or equal to 1/300. Since the GPU works with 8 bit numbers, the smallest resolution of a number between zero and one, such as color, is 1/256. Therefore, the only attainable number less than 1/300 is

zero. However, a zero blue value can represent a never visited cell only if coverage never drops below 1/256. The final three lines of the fragment program ensure that this is the case. The main function of this program is to reduce the blue value by one increment (1/256) every frame. In this way, coverage stays at the maximum value (255/256) only when a UAV is occupying the cell.

In short, the brightness of the blue color determines how recently the cell has been visited. Additionally, perfectly black cells represent cells that have not yet been visited. To record this coverage, the average blue value of all the cells was calculated and recorded in a file. The case with the highest blue component is doing the best at exploring new areas. It should not come as a surprise that coverage is dependent on the number of remaining UAVs. Clearly more UAVs will result in more blue points being drawn and will likely result in a higher coverage. This dependence is expected because the more UAVs there are; the more likely they are to explore unique cells. Finally, as an additional measure of coverage, the number of unexplored cells was also recorded. The Results section contains more details about the swarm behavior analysis performed under the "Effect of Pheromone Parameters on Swarm Behavior" subheading.

Achieving UAV Movement

In this research, a simple control mechanism is used to get UAVs to their desired goal state. However, before the method can be described, critical dynamic properties of the swarming UAVs must be assumed. These assumed dynamic properties are shown in Table 1.

Property	Value	Units
Maximum Speed	350 (239)	ft/s (mph)
Cruising Speed	300 (205)	ft/s (mph)
Minimum Flying Speed	100 (68)	ft/s (mph)
Maximum Yaw Rate	60 (π/3)	deg/s (rad/s)

Table 1. Dynamic properties of units in swarm

The above properties are estimates of the capabilities of units in the near-future swarm already described. The MQ-9A Predator commonly used today has a listed maximum speed of 230 knots with a cruise speed of 200 knots [37]. These speeds are equivalent to 264.5 mph and 230 mph. As a result a maximum speed of 239 mph is reasonable. The assumed cruising speed of 205 mph matches favorably with the already achievable 230 mph cruising speed. The minimum speed of 68 mph also matches closely to the Predator's minimum cruising speed of 77 mph [37]. At this point, the reader might be wondering if these estimates are unrealistically conservative. However, there are two big differences between the Predator and the future UAVs likely to see swarm action: cost and size. Each Predator costs $2 million and has a wingspan ranging between 36 and 64 feet [38]. As previously mentioned, the UAVs in the swarm will need to be small (wingspan in the neighborhood of 5-10 ft) and relatively inexpensive (cost between $10,000 and $100,000). Miniaturization and improvement of current technology should allow a craft of that price and size to perform at similar speeds to today's Predator. The quality that comes most in question is the maximum yaw rate of 60 degrees per second. Predators cannot match such a rate of change in their heading, but in 1991 NASA was able to outfit F-18s to achieve a controlled yaw rate of 50 degrees per second [39]. Given that F-18's are manned craft with large wingspans, a smaller unmanned plane should be able to achieve 60 degrees per second. While it is impossible to prove that a UAV in a near-future UAV swarm will achieve all of

these dynamic characteristics, they are within a reasonable envelope of likely possibilities based on capabilities of modern aircraft.

Given these properties, it is possible to show that the UAV can reach the goal state it is assigned by ADAPTIV. The UAV must get to this assigned goal state within a preset time limit. This time limit was set to five seconds in this research. Once a UAV receives a goal state, its first task is to line up its heading so that it matches with the vector between its current state and its goal state. If the required turn is less than 60 degrees, it will continue at current speed and apply the needed turn rate for one second and then fly straight. If the needed angle is between 60-120 degrees then the UAV slows to 275 ft/s (188 mph) and makes the turn at a rate of half the required angle for two seconds. If the angle is greater than 120 degrees, then the UAV slows to its minimum speed of 68 mph to make a tight turn at a rate of a third the required angle for three seconds. This behavior was chosen to ensure that the turn could be made within a 200 foot space. Figure 9 shows the three cases: easy ($\theta \leq 60^{\circ}$), moderate ($60^{\circ} < \theta \leq 120^{\circ}$), and tight ($\theta > 120^{\circ}$).

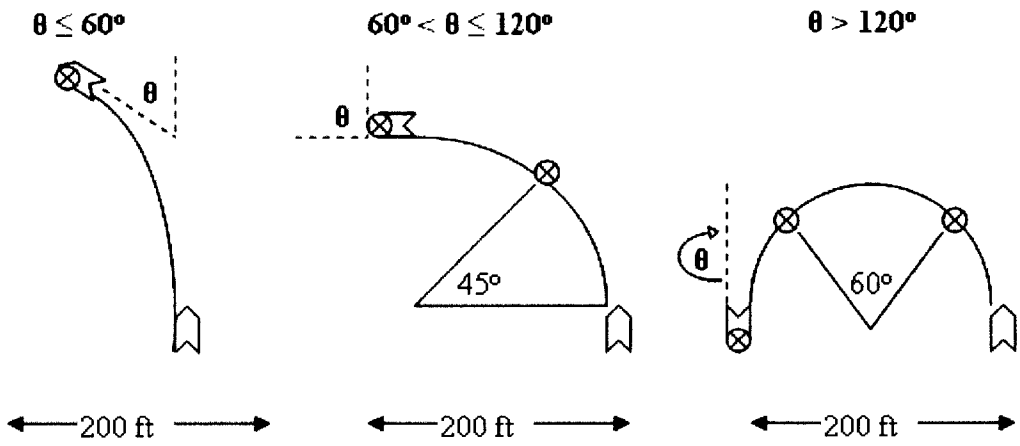

Figure 9. UAV turning cases

The circled X symbol in Figure 9 represents the UAV's location after a second of travel. Notice in the first case that the UAV reaches the desired heading within the first second, the second case requires two seconds and third requires three. Further, the radius of curvature is different in each case. With an easy turn, a speed of 363 ft/s would result in just making the 60 degree heading change in 200 feet. Clearly, 363 ft/s is faster than the maximum UAV speed of 350 ft/s. This value was obtained with the knowledge that the space taken by a turn of 60 degrees is half the radius of curvature (in the x-direction) and 0.866 of radius of curvature in the y-direction. Since the turn must be made in a 200'x200' square, the y-direction has the more stringent speed requirement. The maximum speed value of 363 ft/s for a 60 degree turn is found using Equation 3 to determine the maximum radius of curvature based on the 200 foot restriction and then using Equation 4 to determine the distance traveled. Since this distance is traveled in one second, it also equals the maximum allowable speed in ft/s.

$$r_{allow} = 200\,ft \,/\, max(cos\theta,\ sin\theta) \text{ for } 0<\theta<\pi/2 \quad (3)$$

$$\theta * r_{allow} = dist \quad (4)$$

The moderate turn case introduces additional complexity when dealing with its obtuse angles. In the worst case, at a change of 120 degrees, the radius of curvature is determined by the x-axis change. Specifically, Equation 6 can be used with Equation 5 to determine the maximum allowable speed for a 120 degree turn. Note, however, that the travel time is two seconds so the maximum speed is half the calculated distance.

$$r_{allow} = 200/(1+sin(\theta-\pi/2)) \text{ for } \pi/2<\theta<2\pi/3 \quad (6)$$

Using Equation 6, the maximum allowable speed for the maneuver in the second case of Figure 9 is 279 ft/s. This is why the UAV changes its speed to 275 ft/s to make moderate turns. Finally, the tight turn case requires only direct application of Equation 5 since the worst case scenario is a 180 degree turn. In this case, the half circle's diameter must be less than or equal to 200 ft, making its radius 100. Thus the distance calculated is 314 ft which must be traversed in three seconds. This results in a maximum speed of 104.7 ft/s. Thus, the UAV slows to its minimum speed of 100 ft/s when making a tight turn.

Once the UAV has achieved the proper heading, it needs only to fly straight to get to its destination. At this point it has spent anywhere between one and three seconds getting the heading correct and must make it to the goal before the five second time limit elapses. To accomplish this, the UAV will first check to see if flying at cruising speed (300 ft/s) will be too fast for the remaining distance. This would only occur if the remaining distance was less than the cruising speed multiplied by the time step of the simulation. Since the application has a time step of 0.1 seconds, this cutoff distance is only 30 feet. More likely, the UAV will be far enough away to fly at cruising speed. However, the UAV must check to make sure it can make it to the goal flying at cruising speed for the remainder of the five seconds. If it can not, it will fly at its maximum speed of 350 ft/s until it can switch back to cruising speed.

If the UAV's maximum speed is not enough, then the UAV will fail to make its goal state in time. In the worst case of a 180 degree turn, it still has the entire distance to travel in only two seconds. With a maximum speed of 350 ft/s, the farthest the UAV could get in two seconds (with instantaneous acceleration) would be 700 ft. Therefore, the maximum distance that can safely be traveled by a UAV in the given five seconds is only 700 ft. Fortunately all goal states given by ADAPTIV are close to the UAV's starting position. Additionally, as long as a cell in the pheromone field is less than $700/\sqrt{2}$ ft (495 ft) on a side (the diagonal of such a square would be 700 ft), the UAV can make it from a cell to any of its nearest neighbors. Any goal state selected by ADAPTIV will be in a cell's nearest neighbor. In my

research, the battlefield was 50,000 ft long and 50,000 ft wide. When the field is split into a 128x128 pheromone field, each cell is 391 ft on a side. Because this is less than the maximum square side length of 495 ft, a UAV can make it from any point in a cell to any of the cell's nearest neighbors. Additionally, with a side length of 391 ft, the maximum turning area of 200x200 ft^2 is approximately ¼ of the cell's area. This means that in many cases, a 180 degree turn could be completed within a single cell.

This control setup is not the most sophisticated algorithm and would not be acceptable in a real-world scenario. First of all, it does not have any collision avoidance capabilities. Second, the UAVs are frequently required to change speed abruptly, which would compromise fuel efficiency. Additionally, this algorithm does not accomplish altitude deconfliction. However, it is sufficient to demonstrate the concept in this prototype system and to allow for the development of operator interfaces. A real-world system might employ an algorithm similar to the one described by Reza Olfati-Saber using the goal states given by ADAPTIV as λ-agents.

Preparing Battlespace for Swarm Management

After developing a GPU version of ADAPTIV and a means for moving the UAVs to their goal positions, the operator interface remains to be designed. Since the Virtual Battlespace is the platform for the swarm interface, it required updating. Specifically, the architecture of the application was reworked in part so that it could be implemented on a clustered architecture and in part to accept the swarm control interface. This section discusses the new overall Battlespace architecture developed and then discusses the two main components involved in swarm management: the swarm manager and the menu manager.

Battlespace Architecture

Battlespace uses the command architecture outlined in *Design Patterns* and is illustrated in Figure 10 [40]. The essence of the command pattern is to create a method of communication between objects that do not know about each other. The idea is similar to broadcasting on a UDP channel. In that case, the sender streams data on a given port without knowing who, or even if, another object will get the information. This is often useful behavior when the recipient changes frequently since the sender need not know any IP address or port number for any of its potential listeners. In an object-oriented program, the objects can be thought of as individual computers and a function call between two objects can be thought of as network communication. In this case one object (the sender) calls a method on another object (the receiver) and passes along data (the packet) through the call. This is the typical object-to-object interaction, but like TCP it requires that the sender have information (a pointer) about the receiver. Often times this requirement is not difficult, however, if the receiver object changes, the sender object would also likely need to be changed. Additionally, the receiver needs to maintain pointers to all the objects it wishes to call, leading to interconnected code and the potentially unfortunate need to have the same two objects maintain pointers to each other. These issues can be solved with a pattern that mimics the UDP model while providing additional benefits.

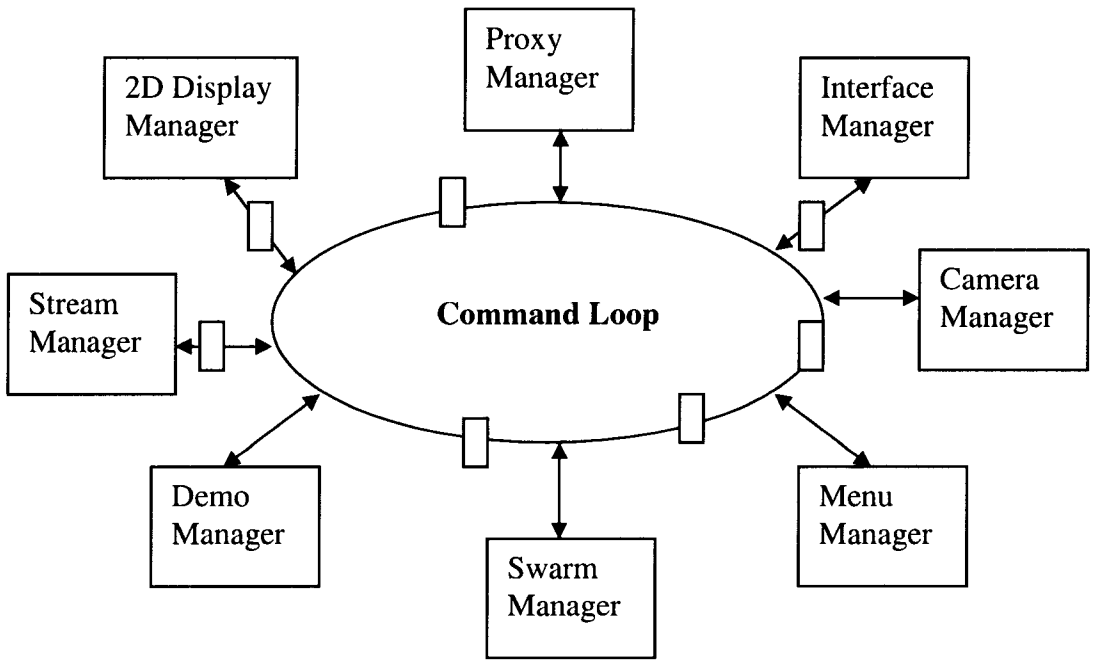

Figure 10. Battlespace architecture

The command architecture allows an object to send a request (call a function) without having to know about the receiving object. This is accomplished through an intermediary object called a command. A command consists of a directive, and optionally a qualifier and/or data. The directive identifies the type of command and is often implemented as a string so that it is easy to read the code. A qualifier, again often a string, is used to specialize the command as a different version of the given command type. The data contains any numerical values needed. These data would be the arguments of a typical function call. As an example, a command might have the directive "Move User." This directive informs all the objects that read this command that it contains information about changing the viewpoint. If the command's qualifier were "Absolute," it would tell the reading object that the viewpoint should be changed to the given coordinates (in the command's data) while a

qualifier of "Incremental" would tell the reading object that viewpoint needs to be moved from its current location by the amounts given in the command's data. It quickly becomes clear that different commands could have different amounts of data items, and even the same type of command could have varying amounts of data. For example, if three numbers were supplied with the "Move User: Absolute" command, then the reading object would move the viewpoint to the given (x,y,z) position but leave the heading, pitch and roll alone. However, if there were six data values then the reading object would not only change the (x,y,z) position but then would also change the heading, pitch and roll of the viewpoint.

The data of a command requires an implementation that allows for a variable number of items and an easy way to check to the number of items present. The `std::vector` container class is a perfect candidate for this task so long as a single data type is sufficient. In Battlespace, the data are stored as floats. If multiple data types are required then, a `std::vector` for each type could be included in the command. This would not increase the command's size much since vectors of an unused type would be empty and therefore add only a few bytes to represent the base vector container data.

A command is essentially an agreed upon communication protocol, and to effectively communicate, the objects must have a means to pass commands to one another. In the Virtual Battlespace this is accomplished via the command loop as shown in Figure 10. The command loop is a conduit that carries commands (the boxes in Figure 10) between objects. The loop must be maintained by a master object that manages the flow of commands between objects. In Virtual Battlespace, this object is the Component Manager. The Component Manager's only tasks are to register Battlespace Components thereby hooking them into the command loop, and to manage the list of commands as it travels around the

loop. Battlespace components are objects capable of sending and receiving commands and the Managers in Figure 10 are derived from the BattlespaceComponent class. BattlespaceComponents have two primary virtual methods involving commands. These are the *updateStateAndCreateCommands* and *processCommands* methods. The first method generates commands while the second receives and executes commands. The Component Manager, which has a `std::vector` of all registered BattlespaceComponents, calls *updateStateAndCreateCommands* on each and passes in a modifiable `std::vector` of commands. Each component could then add commands to the list of commands during this call. Once all components are called, the Component Manager takes the command list and calls *processCommands* on each of its BattlespaceComponents passing in a non-modifiable version of the command list. During this call, components can search the command list for commands they care about and then react to them.

As an example, when the operator of Battlespace uses the controller to navigate by pulling on the controller's analog thumb stick, the command loop facilitates this navigation. When the stick is pulled, the Interface Manager detects this input and generates a "Move User: Incremental" command with three data values based on the specific controller input. These three values represent the desired amount of movement of the viewpoint. When the Interface Manager's *updateStateAndCreateCommands* function is called by the Component Manager, it pushes this new command onto the command list. Then, when the Component Manager calls the Camera Manager's *processCommands* with the same command list, the Camera Manager gets the message that the user wants to change perspectives by that desired amount. By controlling the users viewpoint, the Camera Manager gives the user the sense of having moved through the scene. In this way, one component can effectively call the

function of another component it doesn't know about via a command. In fact, since more than one component can respond to a command, the sending component may actually call multiple functions at once.

The true versatility of the architecture is shown when a different interface device is used. For instance, Battlespace can be controlled using only an interface on a tablet computer, so the operator must be able to use the tablet to navigate. Because of this requirement, the Interface Manager contains the code to communicate with the tablet and receive the operator's request to shift the viewpoint. The command generated by this input would be the same as that created by changing view with the controller and the process that followed would be exactly the same. So even though the source of the data in the command is different, the impact on the application is minimized to the code that is written to interact directly with the new interface device. This versatility can be applied in the other direction as well. For example if an avatar were drawn where the user is located in the virtual world, a theoretical BattlespaceComponent, perhaps called the Avatar Manager, could also listen for "Move User" commands. It could then use these commands to position the avatar within the virtual world. This frees the sender (the Interface Manager) from having to check for the presence of the Camera Manager and Avatar Manager and then having to send the desired navigation information to whichever managers are present. Instead it can simply issue the move user command and not be concerned with the status of the Avatar or Camera Managers.

Swarm Manager

The Swarm Manager is the first of two Battlespace components created for this research. The main purpose of the Swarm Manager is to interact with the Swarm Computer running the GPU ADAPTIV algorithm. It must provide the link between the swarm and the operator. Its first task is to receive the pheromone field from the Swarm Computer and store it for display within the virtual space. It must also be able to send operator inputs of swarm control parameters to the Swarm Computer. In addition, it has to retrieve position data and video footage from a UAV of interest in the swarm as selected by the operator. Finally, it must be able to alert the rest of the Virtual Battlespace to new unconfirmed threats or targets found by the swarm. The remainder of this section describes how the Swarm Manager accomplishes all these tasks.

The Swarm Manager accomplishes its pheromone field display task by communicating via a UDP connection with the Swarm Computer. The Swarm Computer sends the pheromone field once every second. This slower rate of update was chosen to reduce the amount of network traffic. The field is stored as an array of red, green and blue values on the Swarm Computer. When the field is sent, it is first compressed into a JPEG image and then sent across the network in small chunks. The swarm manager listens for these updates and when the whole field is sent across, it decompresses the field from the JPEG image. It then stores this field as the current available field. The Swarm Manager does two things with the current field; either it displays it in context on the terrain or it displays the field for the Menu Manager. Figure 11 shows the pheromone field placed on the terrain.

Figure 11. In context pheromone field display

Figure 11 shows the advantage of placing the field over the terrain. The operator is given immediate context for the swarm's operational location. Further graphical information is provided by a white dot that is placed on the terrain where each unit in the swarm is located. These dots give the operator a rough idea where the swarming UAVs are, and more importantly, where there are no swarming UAVs. Further, the texture representing the pheromone field is alpha blended to be mostly transparent where there is a lack of pheromone (black areas). This allows the operator to see the underlying terrain through uninteresting areas of the pheromone field. The Swarm Manager accomplishes this in-context placement of the field by passing the pheromone field texture to the Terrain Manager with the coordinates of the field's upper left corner and its width and length. The Terrain Manager then applies the texture over the appropriate area of the terrain.

The second method of displaying the pheromone field involves the Pheromone Input menu of the Menu Manager (described in the next sub-section). When the Pheromone Input menu is chosen, the pheromone field must be displayed in an area of the menu. This is not a difficult task since it amounts to displaying a texture on a 2D polygon. However, the data making up the texture, the pheromone field, is dynamic and maintained by the Swarm Manager. Instead of having the swarm manager send the texture data to the Menu Manager to display, the Menu Manager tells the Swarm Manager the coordinates where the field needs to be drawn. In this way, it appears that the Menu Manager is invoking the pheromone field when the Pheromone Input menu is selected, but in fact, it is the Swarm Manager that displays it when that menu is selected.

The Swarm Manager uses a TCP connection with the Swarm Computer to send changes to swarm control parameters. These parameters come from the Swarm Management menu described in the next sub-section. Once the Swarm Manager receives a new control setting from the Menu Manager, it simply sends a packet with that information to the Swarm Computer. The new control settings then immediately take effect.

The operator can use the Pheromone Input menu to designate a UAV of interest in the swarm. This designation allows the operator to go the viewpoint of that unit as well as receive streaming video from that unit. This capability is critical when trying to decide if a threat indicated by the swarm (which is not 100% accurate) is indeed a threat. With this capability, the operator can choose the unit closest to the threat of interest and play the video captured of that area. From that footage, the operator can hopefully decide if the threat is harmless or truly something to be concerned with. In order to view the unit's position and video, a new connection to the Swarm Computer must be made. This two-way TCP

connection allows the Swarm Manager to request that the UAV nearest the point of interest (as specified by the operator) become the selected unit. Once a unit is selected, the Swarm Computer sends position updates to the Swarm Manager, which then uses this information to create a unit within the Virtual Battlespace. Figure 12 shows a video feed from the viewpoint of a selected swarming UAV within the Virtual Battlespace.

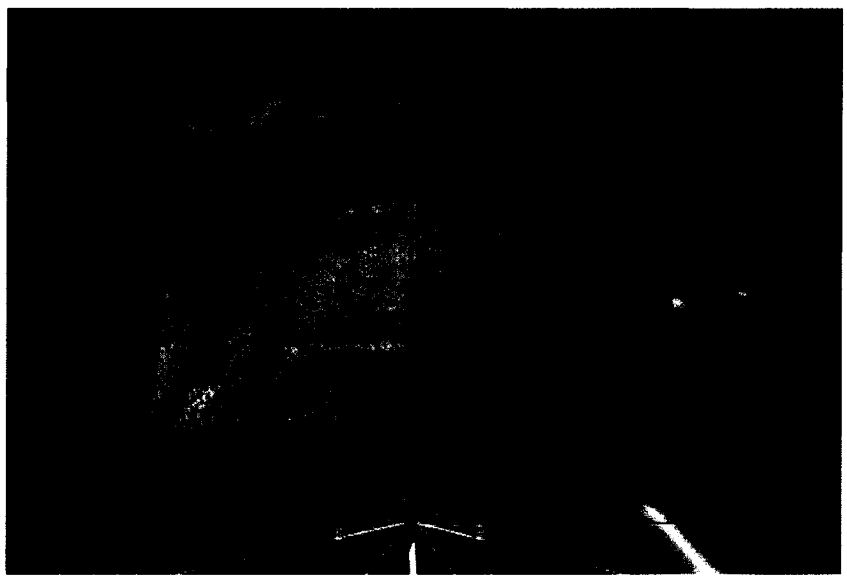

Figure 12. In-context display of selected swarming UAV's video feed

Once a UAV is selected, the IP address and port number of the streaming video from the unit is sent to the Swarm Manager. This information is used to create a connection between the computer playing the video and the Swarm Manager. The same JPEG compression UDP communication setup used to send the pheromone field is used to send the video images to the Swarm Manager. Then, when the operator wants to display the video from the selected swarming UAV, the current video texture is displayed on a 2D polygon positioned in front of the aircraft as shown in Figure 12.

The final task of the Swarm Manager is to alert the rest of Virtual Battlespace when the swarm thinks it has found something of interest. Because the units in the swarm are unable to flawlessly classify objects found in their videos as non-combatants, friendlies, targets or threats, these decisions must be left to the operator. As mentioned in the GPU ADAPTIV section, the units in the swarm drop pheromones of a particular flavor designating an object of interest as a threat, target or neutral. This swarm designation, which may be incorrect, is then sent to the Swarm Manager for confirmation from the operator. Once the Swarm Manager receives this information, it immediately creates an unconfirmed threat or target. It then sends a message to the rest of Battlespace that there is a new unconfirmed finding of the swarm that the operator might want to investigate. This message is called an alert and is used by other parts of Battlespace to draw the operator's attention to this area.

The UAV swarm generates alerts, but they are a general feature of the Virtual Battlespace. Alerts can be generated automatically by the system or by another operator. In practice, alerts could come from the battle scenario, a manned unit or another commander viewing the same virtual battlefield in a different physical location. For example, a manned unit could generate both kinds of alerts. If the pilot sends a request for the commander's attention, a human-generated alert would pop up in the virtual battlefield. If, however, the plane is designed to send an alert when an enemy locks its radar on it, the manned unit could send a computer-generated alert to the virtual battlefield. Clearly, the swarm can only issue computer-generated alerts. Further, only two types of alerts, unconfirmed targets and unconfirmed threats, are generated in this research. These alerts are labeled as unconfirmed because a human has yet to confirm the UAV's classification of a threat or target. Since

computers cannot be trusted with making the decision alone for fear of costly false positive matches, the operator must classify the alert to solidify its status as a threat or target.

When an alert is received by the Virtual Battlespace, flashing "New Alert" text appears on the screen (or front wall of a large display). The operator can then hit the "C" button on the controller to immediately travel to a viewpoint that shows the alert. The alert will appear as a unit on the terrain as shown in Figure 13.

Figure 13. Alert icons on the virtual battlefield

A new alert has a textbox hovering over it that asks the operator to investigate it. This message stays over the alert until the operator visits it for the first time. This is the cue for the operator to gather whatever evidence is available to make the alert classification. Video footage of the alert is the primary evidence source in this prototype system and the Menu Manager section describes how the operator would view such footage. Once the operator has viewed all the evidence and is prepared to classify the alert, the operator presses

"C" repeatedly until they reach the alert of interest. Then, if this alert has already been visited, the classification menu shown in Figure 14 is displayed over the alert icon.

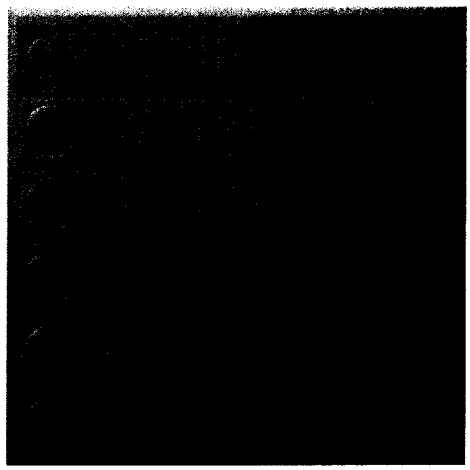

Figure 14. Alert classification menu

Once the operator sees this menu, the alert can be classified as a target, threat, ally, non-combatant or false alarm with a press of the appropriate button. Once classified as a threat, it will appear to all other users as an enemy unit within the space. This allows the operator's units to engage and eliminate the newly discovered threat. In this way, the operator can truly close the loop between human and machine in this system and provide the swarm with human guidance.

Menu Manager

The Menu Manager has only one purpose – collecting operator inputs. It handles inputs of several different kinds ranging from selection of any unit the Virtual Battlespace to changing swarm control parameters. The Menu Manager is designed to display various 2D

menus on a polygon positioned within the Virtual Battlespace. This polygon is attached to the scene graph in such a way to make sure its position relative to the user is constant during navigation.

The key feature of the menus displayed by the Menu Manager is that even though they are displayed in a 3D environment, the interaction with the menu is 2D. Commonly, menus in VR applications use 3D interaction methods for user interaction. For instance, a user might have to move the wand in their hand and try to spear a button floating in 3D with a long virtual "poker" extending from the end of the wand. This kind of button selection is cumbersome and prone to error, especially if the menu rotates as the user navigates or is far from the user's position [41]. Additionally, tracking of the wand position in practice is not completely stable or reliable. Small inaccuracies in tracking are common, and a small tracking error could cause the user to select the wrong button on the menu. With menus displayed by the menu manager, a virtual cursor arrow appears within the menu's visible area. This cursor moves in the familiar and effective 2D manner seen on virtually every computer.

The Menu Manager contains many different menus that the operator can cycle between. Only one menu, the active menu, is displayed at a time. There are menus for unit selection, firing missiles and enabling graphical aids such as height sticks on units. However, these menus provide interaction with the non-swarm parts of the Virtual Battlespace. There are two menus that are used to interact with the swarm: the Swarm Management Menu and the Pheromone Input Menu.

Swarm Management Menu

The Swarm Management menu is used to change the overall behavior of the swarm. Three behavioral qualities can be adjusted via the menu. These qualities are aggression, exploration and care. The final swarm behavior is a combination of these qualities. Because there are three independent qualities to blend, Barycentric Coordinates are used to determine their combination. Figure 15 shows an example of Barycentric Coordinates.

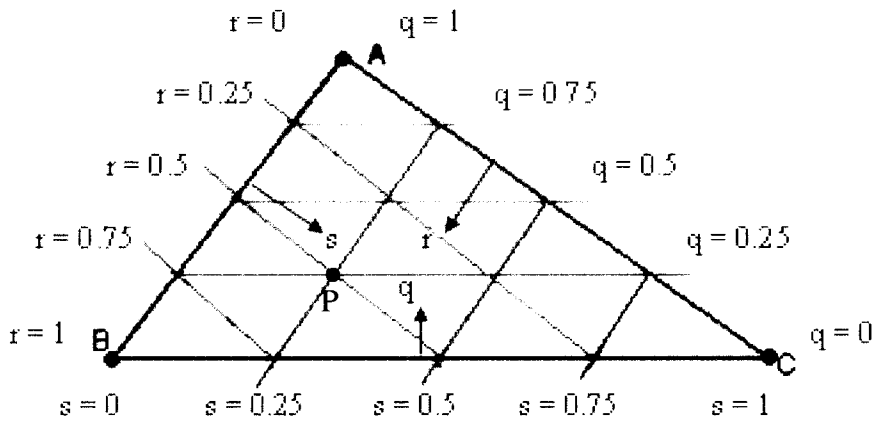

Figure 15. Barycentric coordinates

Barycentric coordinates have characteristics that are ideal for the selection of the swarm behavior. Chief among these is that they can be used to easily mix three variables. A Barycentric coordinate is a point within a triangle that is specified by three numbers (q, r, and s) that range from zero to one. These numbers can be considered as the masses of objects that must be placed at the three vertices to make the point of interest the triangle's center of gravity. The first coordinate, q, is the mass placed at vertex A, while the second coordinate, r, specifies the mass at vertex B and the third coordinate, s, specifies the mass at vertex C.

The masses are normalized such that they sum to one. For example, if the point of interest is the geometric center, then to balance the triangle, a weight of equal mass would need to be placed at each vertex. Since the weights are normalized, this means a mass of 1/3 would be placed at each vertex, so the geometric center of the triangle is given by the Barycentric coordinate (1/3, 1/3, 1/3). Figure 15 shows how to determine the Barycentric Coordinate of an arbitrary point inside the triangle. Point P is given by coordinates of (0.25, 0.5, 0.25) since it lies at the intersection of the q=0.25, r=0.5 and s=0.25 lines. Note that since the point is closest to vertex B, its mass (r) is the largest.

If, instead of literal masses, the vertices represent the "weights" of different colors, then the Barycentric coordinates of a triangle can be used to create a color mixing effect. Figure 16 shows this mixing with greater distance from the left vertex resulting in a smaller red component, greater distance from the top vertex resulting in a lower blue component and greater distance from the right vertex resulting in a lower green component [42]. The effect is that the corners of the triangle are the colors specified above and the center of the triangle is a blending of these colors. For example, half-way down the side AC, the color is purple – a mix of equal parts red and blue.

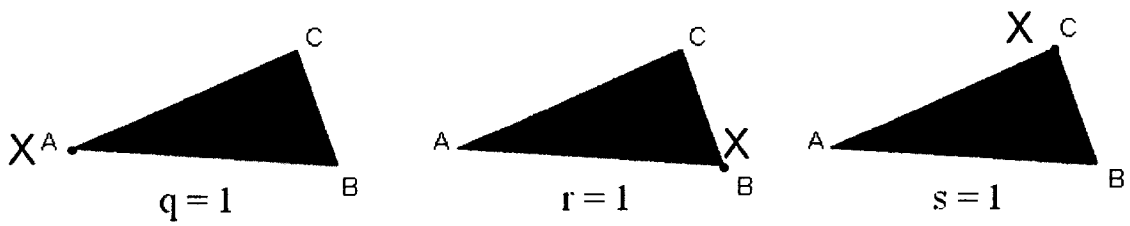

Figure 16. Barycentric color mixing

If swarm behavior characteristics replace colors, then a triangle with Barycentric coordinates represents a way to mix these characteristics. If a user interface were made of a triangle in which the user could select an interior point, a mix of behaviors would be easily calculated. For example, if the user wants a mostly aggressive swarm, but one that also has slightly exploratory behavior, then the user would select a point near the "aggressive" vertex along the line between the "aggressive" and "explore" vertices. Figure 17 shows the Swarm Management menu which contains just this type of triangle interface.

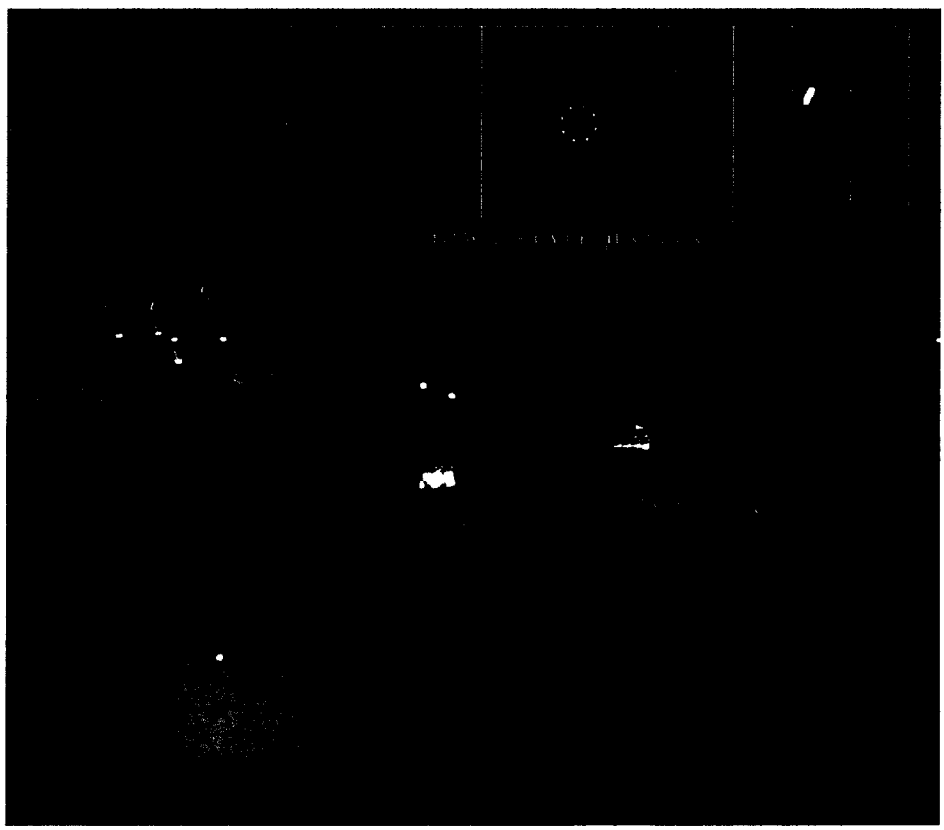

Figure 17. Swarm management menu

The triangle interface of the Swarm Management menu contains snap points for quick selection. There are ten snap points along each side and more points for all the intersections of these side snap points. Since Barycentric coordinates go from zero to one, this means there is a snap point for every combination of the three numbers in which each is a multiple of one tenth. These snap points represent the pre-set swarm behaviors. When a user clicks in the triangle, the click selects the nearest snap point to set the swarm's behavior. This is accomplished by defining the control settings for the ten different levels of each Barycentric coordinate. The swarm control settings that can be changed in this manner are:

- UAV pheromone sensitivities to threats, self-repulsion and targets
- pheromone propagation rates for threats and others
- pheromone evaporation rates for threats and others.

The actual values assigned to these control settings by the selection are stored in file that is read in at the beginning of the application's execution. The Barycentric point selected by the user is used to select from and mix the settings stored in the file. For example, if the chosen coordinate is (0.1, 0.3, 0.6) then the second of the eleven "aggression" stored cases, the fourth of the "explore" cases, and the seventh of the "safe" cases are selected. Then a weighted sum of the settings desired by each of the three cases is used to determine the control setting values. The weights in the sum are the Barycentric coordinates themselves. In the example above, if the aggressive case calls for a threat sensitivity of one, the explore case calls for five, and the safe case calls for ten, the actual selected threat sensitivity would be 7.6 (1*0.1 + 5*0.3 + 10*0.6).

The control setting values stored in the file were determined by performing test runs with different control setting values and measuring the results. These results were examined to determine which control setting changes led to the desired behavior. The results section discusses this process and its results in detail. The control settings from these tests are not the only values stored in the file. The outputs of the test runs included the number of remaining UAVs, percentage of threats found and percentage of targets found. These three outputs are displayed to the user as green numbers in the upper bar of the menu shown in Figure 16. These numbers are meant to give the operator an idea of the likely performance of the swarm under the chosen behavior settings. To make the cost of a setting clear, the number of UAVs output is displayed as the total cost of the UAVs likely to be lost. This puts a dollar amount on the chosen behavior.

The reader may be wondering how data stored in a file about the number of UAVs that are lost and the percentage of threats and targets found can possibly be applicable to a battle scenario as it is unfolding. The data stored in the file cannot predict the outcome of the battle; instead it is used to warn the operator of the likely performance of the swarm based on results using the chosen settings in previous experience. These numbers should be considered as a guide and not as absolute predictions. As an additional aid the operator can view the difference in performance between the current behavior setting and any other behavior setting, by moving the cursor over different Barycentric Coordinates in the swarm behavior-selection triangle. The current expected performance is shown in green. The change to performance that would occur if the operator selected the new coordinate is shown in red (Figure 17). Displaying these differences allows the operator a simple, efficient way to see how the different control settings change the swarm performance.

The final functionality provided by the Swarm Management menu is the ability to turn off the snap points within the behavior triangle. If the snap points are turned off, then the data from the nearest snap point in the file is used, but the weights in the sum are from the point selected, not the values of the snap point. The Swarm Management menu allows the user to modify the overall behavior of the swarm and get performance predictions from one interface, all without directing a single swarming unit.

Pheromone Input Menu

The main purpose of the Pheromone Input Menu is to provide a means to view the swarm and understand its situation. The menu is used to show the pheromone field (so it can be watched even if it is not placed on the terrain), show alerts generated by the swarm and allow the user to designate areas off-limits to the swarm and areas attractive to the swarm. Figure 18 shows the Pheromone Input Menu.

The first task, understanding the swarm and its movements is in part accomplished by the display of the pheromone field. As mentioned in the Swarm Manager section, the Pheromone Input Menu does not actually display the field, but rather asks the Swarm Manager to display it. However, to the user, it appears as if the field is part of the menu. Indeed, the Pheromone Input Menu augments the field display by adding other useful graphics on top of it. First, it adds a green pie-slice with its radius equal to half the length (or width) of the field display. Its vertex is fixed to the center of the display and it shows the direction the camera is pointing in the virtual world. This device allows the user to understand the orientation of the field (if it were placed on the terrain) relative to the current view point.

Figure 18. Pheromone input menu

Second, when the swarm generates an alert, an icon representing the type of alert is shown over the area on the field it is generated from. This allows the operator to get an idea of the alert's position in the swarm's operational area. If the operator clicks the arrow cursor on the pheromone field display, the view in the virtual world is changed to show the UAV in the swarm that is closest to the selected point, in a "chase-plane" view, a view that is slightly behind it and aligned with its heading. From this viewpoint, the operator can quickly gain appreciation for the situation surrounding the unit. Further, the operator can play the video from the unit's on-board cameras. When the operator selects the point on the field nearest an alert, the viewpoint shifts to the UAV in the swarm that is closest to that alert. Once the view

has shifted to a position behind the designated UAV, the operator could then play the unit's video to look at footage of the object that instigated the alert. From this footage, the operator can classify the alert as a threat, target, ally, non-combatant or false alarm. This footage is the primary means to verify an alert generated by the swarm. This is how the loop is closed between human and machine, adding a needed safeguard to the swarm's management. Once the operator has made the classification, the swarm and all other allied units on the battlefield may respond quickly.

The final task of the Pheromone Input Menu is to allow the operator to designate regions of the swarm's operational area as attractive or off-limits. The purpose of such an interface is to provide a more direct way to control the swarm's behavior in those particular regions. If for example, a group of manned units is operating within the field area, it might be desirable to ensure that no unmanned units are in the same airspace. This requires that the operator be able to select the area and then tell the swarm controller to bar units from entering that space. Additionally, if there are units already within the newly off-limits area, they must be told to leave the area as quickly as possible.

When an operator wishes to designate an area as off limits, they click on the "Repulse" button to set the area creation mode to generate restricted areas. Then the operator would press the "New Area" button to start the creation of an area. Next, the operator would click on the pheromone field display at the location of one of the corners of the area. Then, a yellow box is drawn continuously from that point to the current arrow cursor point, allowing the operator to change the size of the box easily. A second click sets the area and the finished box is displayed in red. This newly created area's dimensions are sent to the Swarm Manager, which communicates this information to the Swarm Computer over a TCP

connection. The Swarm Computer portions out a region of the operational area that will be declared off-limits to its swarm.

Any swarm units caught inside the now off-limits area are given a goal position that is outside of the area. Since the area is an axis-aligned box, it is easy to calculate which way a unit in the swarm should head to get out of the area most efficiently. The distance along the x-axis between the unit's current position and the left and right box walls are calculated. Likewise, the distance along the y-axis between the unit's position and the top and bottom box walls are calculated. The shortest of all of these four distances is the direction that is chosen. The offending swarm unit is then told to head to the nearest edge of the area box by traveling parallel to an axis. As long as the off-limits area exists, any unit given a goal inside the area will instead go to the nearest edge of the area.

Should the operator desire to make a region more attractive, the operator would first click the "Attract" button to set the area creation mode to generate areas of interest for the swarm. Such areas of interest might be useful to entice the swarm to investigate places where information from other sources has hinted at potentially valuable intelligence. Once the "Attract" button has been pressed, the operator then creates a box in the same manner as creating an off-limits area. The newly created area is represented with a green box and the information is sent to the Swarm Computer in the same way as the off-limits area information. Once the Swarm Computer receives the command to create an attractive area, the coordinates of the area are stored. Afterwards, whenever swarm units "drop" pheromones into the field (the R pheromones in ADAPTIV steps one and two), they will be non-propagating attractive pheromones that blanket the region. This pheromone is stored in the alpha channel of the texture and is treated the same way as the other three pheromone

flavors by the GPU. The strength of the pheromone dropped by the area is set by the operator via the Pheromone Input Menu.

The pheromone strength can be changed to eleven preset levels equally spaced between zero and one. The strength is defaulted at 0.5, and the operator can increase or decrease it by 0.1 with a press of the "Strengthen" or "Weaken" buttons respectively. If a previously created area is selected when this change is made, that area's new pheromone strength will be sent to the Swarm Computer and changed accordingly. Any new areas created will also use the new strength value. To select an already created area, the operator would click the "Prev" or "Next" buttons on the left side of the Pheromone Input Menu. Once an area is selected, it is drawn in white. The "Next" button selects the first box created and then selects from older areas toward newer ones on successive clicks. The "Prev" button starts with the newest (last created) area and then proceeds to the oldest box on successive clicks. Once selection mode is entered, an area is selected until either the "New Area" button or the "None" button on the left side of the menu is clicked. Figure 19 shows the Pheromone Input Menu with areas of both types created and one selected.

The pheromone areas are axis-aligned boxes mostly for convenience and would not have to be limited to this geometry in a final version. A change from axis-aligned boxes would only make the vertex input and the calculation of shortest path out of an off-limits area slightly more complex. However, both of these problems have been solved before in CAD systems and collision detection respectively. The axis-aligned boxes, however, sufficiently demonstrate the concept and portray that aspect of the interface.

Figure 19. Pheromone areas created in the pheromone input menu

The final option provided by the Pheromone Input Menu is the ability to control the display of the pheromone field on the terrain by pressing the "On Terrain" button. This button toggles between showing the terrain and showing a small swarm icon. The swarm icon is used to alert the operator to the presence of a swarm when its pheromone field is not displayed on the terrain.

RESULTS

This section covers the primary numerical results of this research. It begins by exploring the performance advantage gained by converting ADAPTIV to the GPU. It then discusses the data gathered from the test runs designed to determine the swarm control parameters needed to make the swarm exhibit the desired behavior. This data was used in the Swarm Management Menu described in the previous section. This section concludes by exploring the effectiveness of roulette selection in ADAPTIV against two alternatives: random and gradient selection.

GPU ADAPTIV vs. CPU ADAPTIV

To quantify the performance gain of the GPU version of ADAPTIV, it was compared with a CPU version. The time it took to complete one execution of all three ADAPTIV algorithm steps was recorded for certain combinations of battlefield size and UAVs in the swarm. The battlefield size, measured in rows and columns of the pheromone field was tested at sizes 64x64, 128x128, 256x256, 512x512, and 1024x1024. The number of UAVs in the swarm was tested at 5, 50, 500, 5000, and 50,000. In each case, 120 data points were gathered. The test computer had a 3GHz Intel Pentium 4 with Hyper-threading and an ATI Mobility Radeon 9700.

In the cases where swarm size was investigated, the battlefield size was left constant at 128x128. Figure 20 shows the results for these cases.

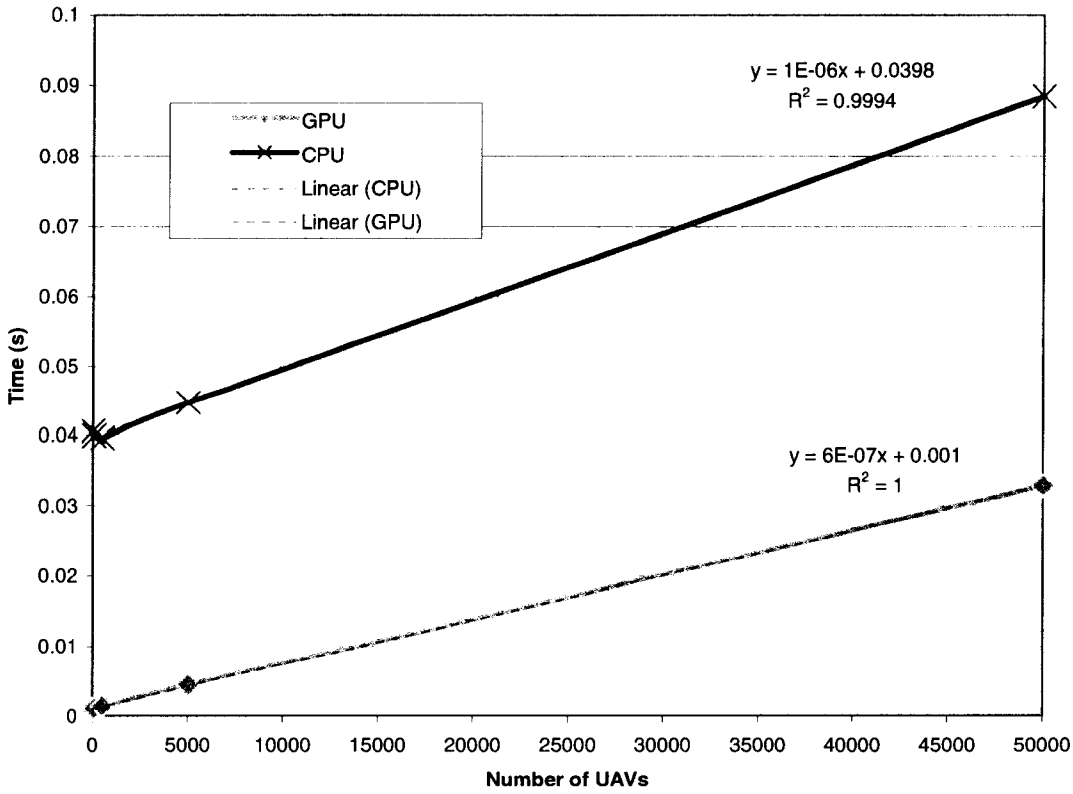

Figure 20. Performance vs. swarm size

After analyzing the swarm size cases, the GPU's performance advantage was found to grow slowly with increasing swarm size. At 5 UAVs the GPU was about 30 ms (1.8x) faster and by 50,000 UAVs the GPU was about 55 ms (2.7x) faster. In the cases where battlefield size was investigated, swarm size was kept constant at 4000 UAVs. The results of those cases are in Figure 21. The GPU's performance advantage was found to grow quickly with field size.

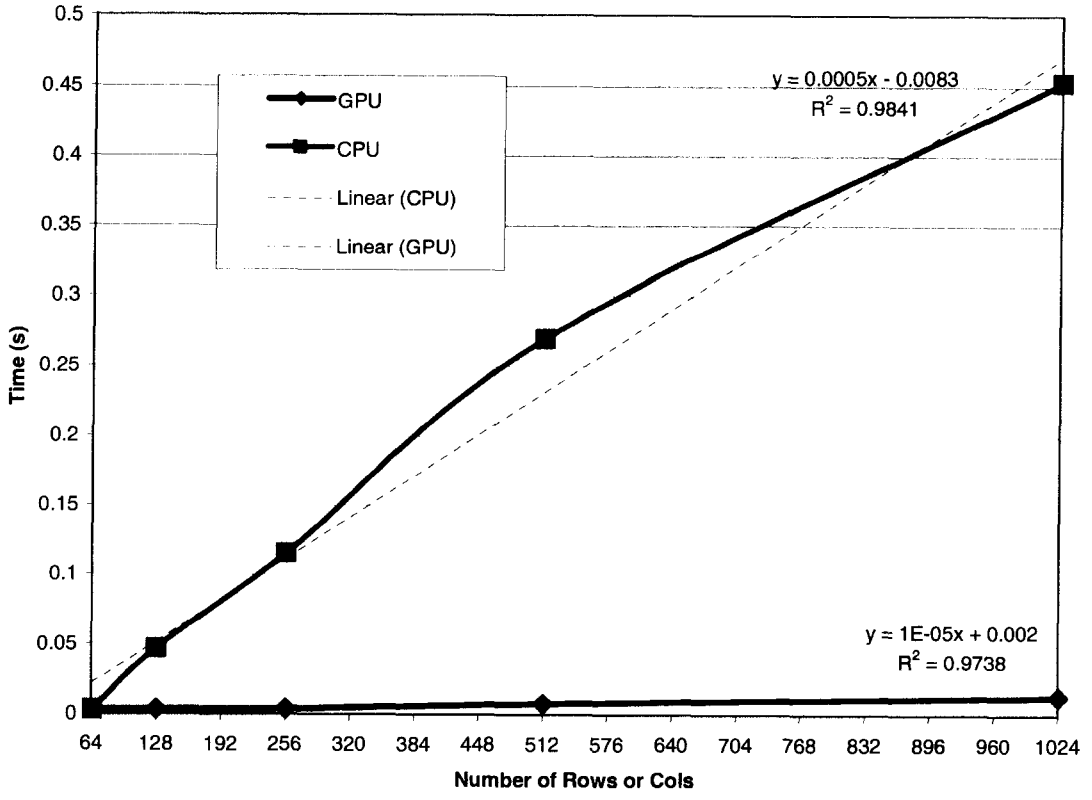

Figure 21. Performance vs. battlefield size

For example, at a size of 64x64, the CPU is actually faster by 5 ms (1.2x) but by a size of 128x128 the GPU is faster by about 43 ms (1.4x) and at a size of 1024x1024 the GPU is faster by about 440 ms (34.8x). The results of these cases show that the GPU algorithm outperformed the CPU version in almost all combinations of swarm size and battlefield size.

These initial results spurred ideas on how to further improve the performance of the GPU algorithm. One idea was to sort the UAVs according to their position on the CPU before performing the roulette selection. Since the UAVs are not visited in order of their location in the battlefield, subsequent UAVs visited could be in entirely different areas of the battle. This can lead to severe texture cache thrashing due to random texture accesses for

large battlefield sizes. One approach to alleviate this effect would be to pass in the UAVs in an approximately spatially sorted order.

To evaluate this idea, an experiment was run with a modified algorithm that sorted the UAVs on the CPU according to their position before performing the roulette selection on the GPU. To determine if any advantage could be gained by sorting, conditions most likely to result in a performance improvement due to sorting – a large (1024x1024) battlefield with a spread out swarm – was used. The sorting causes each UAV in the swarm to be placed in a tile representing the section of the battlefield it is in. In the case of four tiles, the battlefield is quartered. The tiles are then visited in order during the roulette selection step. Both the time to perform the roulette selection by itself and the time to perform roulette selection in combination with the sorting were recorded. Figure 22 shows the former while Figure 23 shows the latter.

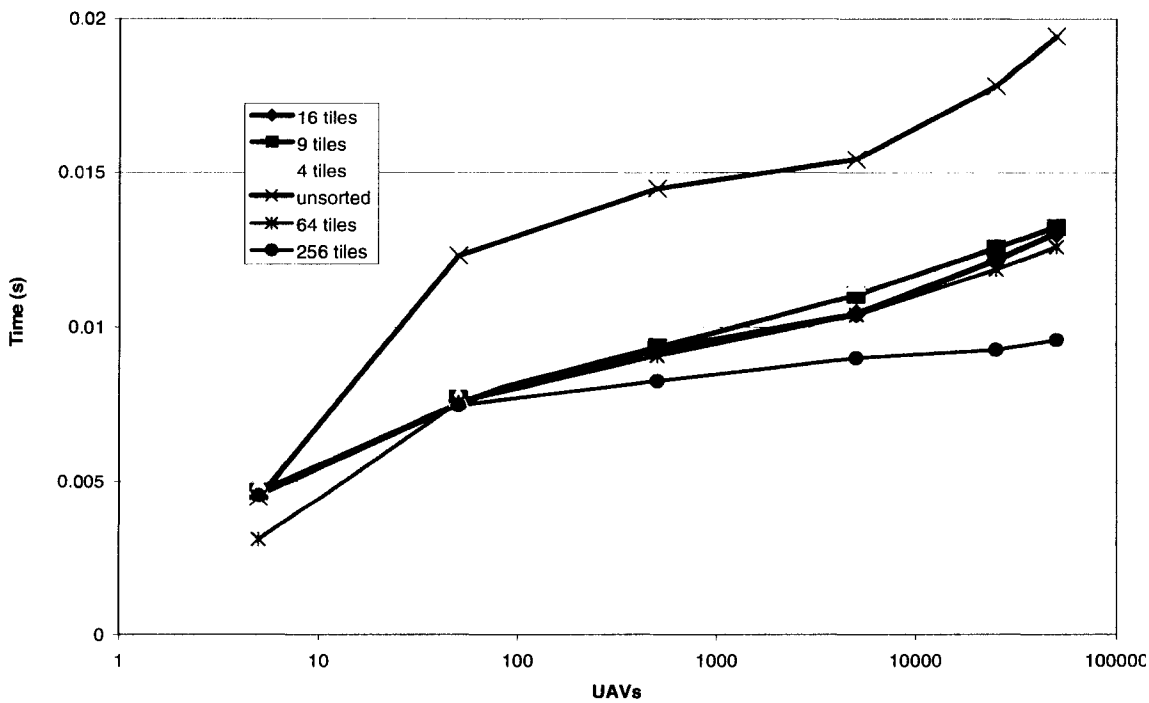

Figure 22. Roulette selection time

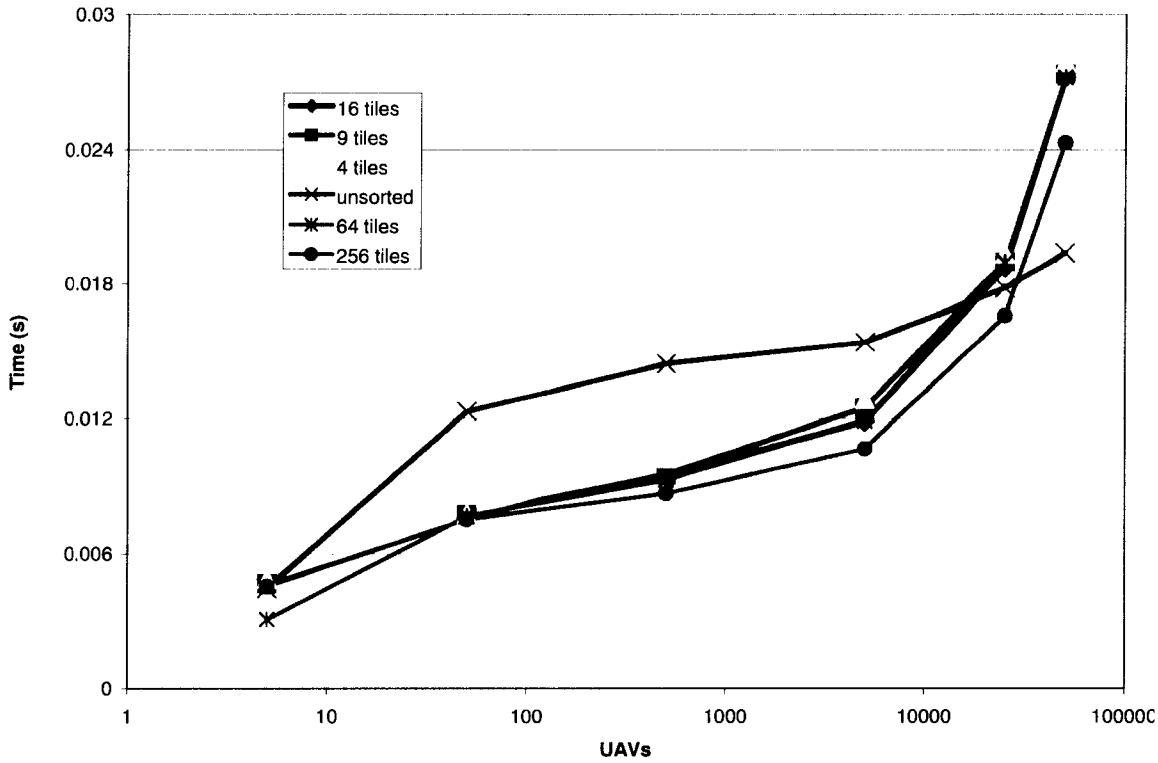

Figure 23. Total ADAPTIV step 3 time

There is a performance advantage to sorting when the battlefield is 1024x1024 and the swarm is spread out as long as the size of the swarm is not too large. Figure 22 shows that at 500 UAVs, the non-sorted version took 14.5 ms to perform the third ADAPTIV step, while the time for the 256, 64, 16, 9 and 4 tiled versions were 8.2 ms, 9.1ms, 9.4 ms, 9.5 ms, and 10.2 ms respectively. Sorting provides an advantage for swarm sizes of 50, 500, and 5000. In these cases sorting resulted in between a 1.4x and 1.8x improvement in speed. With 5 UAVs the sorted and non-sorted versions performed about the same with the exception of an odd outlier when using 64 tiles. In this case, the entire step took only 3.1 ms when the others averaged 4.5 ms. This case will require further examination. When the swarm size reached 25,000 the overall performance between sorted and non-sorted was about

the same. Even though Figure 22 shows that all sorted versions were faster at the roulette step for 25,000 UAVs, this performance gain was entirely offset by sorting time. The situation is worse at 50,000 UAVs. In these cases, the performance gain in the roulette selection is much less than the performance penalty for sorting. These results seem to indicate that sorting the UAVs can result in a performance advantage for large battlefields with spread out swarms that are less than 10,000 units.

The most surprising results came from changing the number of tiles used. As mentioned before, for many swarm sizes, having tiles was better than having no tiles (unsorted). However, varying the number of tiles between 4 and 64 did not result in much of a difference. Using more tiles in this range did improve performance slightly, but not as much as expected. This fact can be seen in Figures 22 and 23 in that the lines for 4, 9, 16, and 64 tiles are nearly the same. Interestingly, using 256 tiles resulted in a substantial difference in performance when compared to using fewer tiles. For all cases with more than 50 UAVs, the 256 tile case was faster and its advantage is most noticeable at large swarm sizes, especially at 50,000 UAVs. Selection only took 9.6 ms for the 256 case, while the average time for the other cases was near 13 ms. This jump in performance could be due to the fact that it is not until there are 256 tiles that each tile is a 64x64 texture. If the Radeon used in this test had a 64x64 texture cache, this could explain the boost in performance and it would mean that these results are only correct for the GPU/CPU combination used. This last fact introduces an important caveat to these results in that they were generated by one computer with one battlefield size. As a result, they can not be taken as absolutes for every swarm control case or specific hardware. What these results do show is that experimentation is needed to optimize performance and that such improvements are possible.

Effect of Pheromone Parameters on Swarm Behavior

The Swarm Management Menu allows the operator to use its triangle interface to manipulate the overall swarm behavior as described in the methods section. However, for this interface to be effective in changing the swarm's behavior, it must select values for the swarm parameters that result in the desired behavior. Fourteen variations of the default pheromone control settings were tested to determine these values. The pheromone control settings tested were the three UAV pheromone sensitivities (to targets, threats and other UAVs), and the propagation and evaporation of the threat, target and repulsive pheromones. The default values for these unitless parameters are shown in Table 2. These default settings were chosen to match those found to be optimal by ADAPTIV's creators. Table 3 shows how the pheromone control settings were changed for the fourteen different test cases.

Table 2. The pheromone control setting defaults

Target Sensitivity	Threat Sensitivity	Repulsion Sensitivity	Threat Propagation	Target/Rep. Propagation	Threat Evaporation	Target/Rep. Evaporation
9	1	3/16	0.9	0.9	0.4	0.4

Table 3. Pheromone control settings for the test cases

Case	Setting Changed	Value		Case	Setting Changed	Value
2	Target Sensitivity	90		9	Threat Propagation	0.1
3	Target Sensitivity	900		10	Target/Rep. Propagation	0.5
4	Threat Sensitivity	10		11	Target/Rep Propagation	0.1
5	Threat Sensitivity	100		12	Threat Evaporation	0.1
6	Repulse Sensitivity	3/32		13	Threat Evaporation	0.9
7	Repulse Sensitivity	3/8		14	Target/Rep. Evaporation	0.1
8	Threat Propagation	0.5		15	Target/Rep. Evaporation	0.9

In each case, only one of the parameters was changed from the default values defined in Case 1. Each case used the same battlefield scenario with the same number of UAVs and generated 600 data points. This data, recorded at each time step, consisted of the number of remaining UAVs, the percentage of threats and targets found, and the coverage of the UAVs. Data was collected with a swarm of 400 UAVs on a 256x256 battlefield with moving threats and targets. Each case was run three times for 100 seconds recording values each half second resulting in 600 data points per case. The average value and standard deviation for each of the five output variables was calculated from this data. Once the data for all of the cases was collected, a standard statistical t-test was performed to see if the averages of each case were significantly different from the default case. For the following discussion, significance will be defined with a 90% confidence interval. With 600 data points, a t-value of at least ±1.65 will then indicate significance. Figure 24 shows the differences of the average values with the default case.

Figure 24 was used to match the control settings on the Swarm Management Menu with desired types of swarm behaviors. The cases with the highest number of remaining UAVs are good candidates for the safe behavior. The results indicate that case 5 is the best at preserving UAVs. Case 5 is characterized by high threat sensitivity and so it was expected to result in the safest behavior. Note that the safety does come with a cost as Case 5's threat and target detection are well below those of the default case. In fact these differences are all significant at 90% confidence with an increase in remaining UAV percentage (12.8%, $t = 17.3$, $n = 600$), decrease in threat detection percentage (-9.3%, $t = -3.5$, $n = 600$) and a decrease in target detection percentage (-10.7%, $t = -3.7$, $n = 600$). These results indicate that with the settings in case 5, the UAVs are shot down less often, probably because they stay

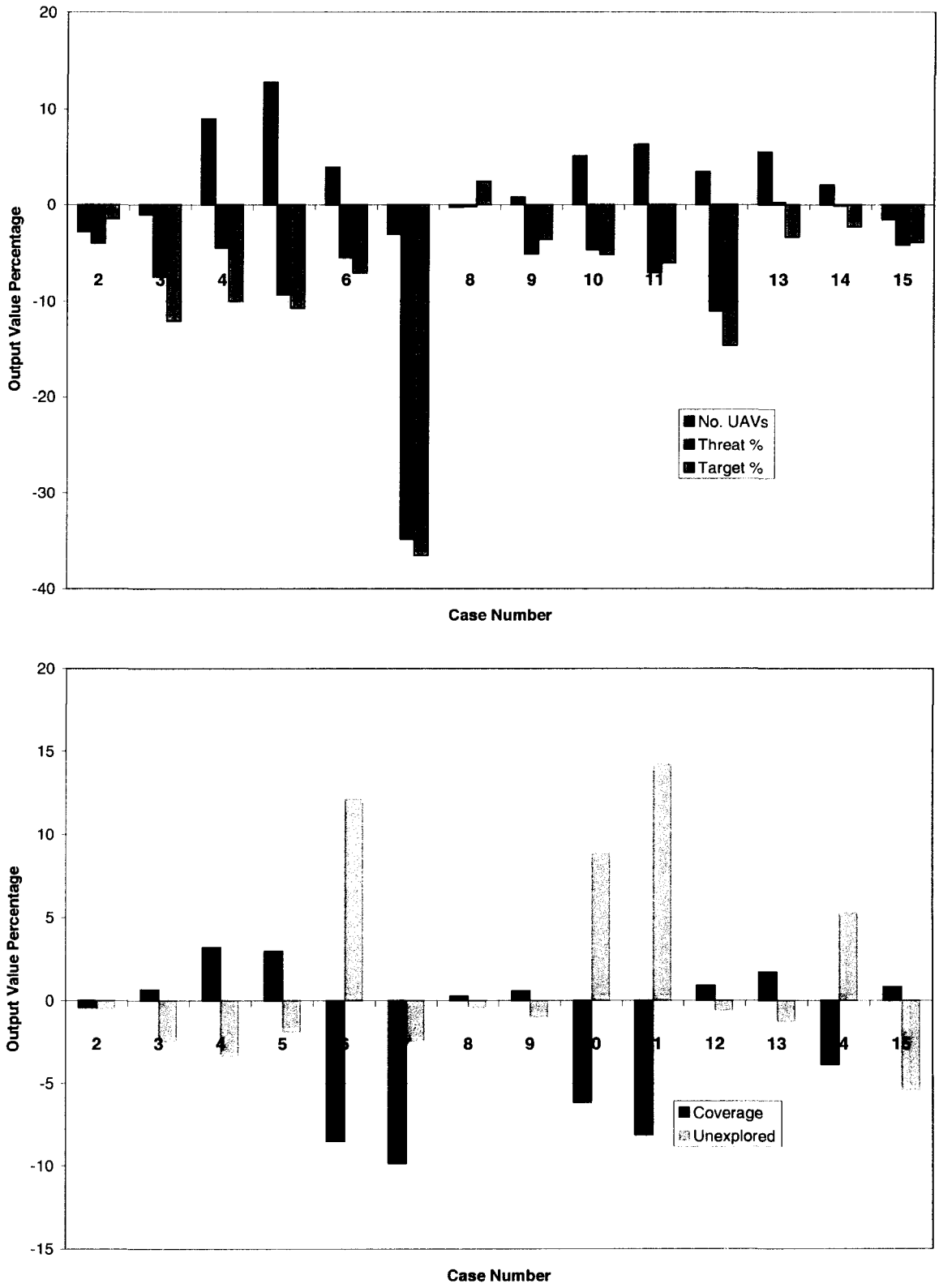

Figure 24. Average values compared to the default case

farther away from threats, but, as a result, they cannot find all of the threats or targets.

The opposite behavior, aggression, is likely to be marked with high detection and high losses. The results indicate that case 8 is the most aggressive setup. However, the differences between the test cases and the default case are less pronounced. In fact, no case is significantly more aggressive than the default case. Case 8 has a higher target detection percentage (2.4%, t = 0.9, n =600), a higher threat detection percentage (0.2%, t = 0.03, n =600), and more UAV losses (0.3%, t = 0.07, n =600). All of these qualities show aggression as expected, but not one of them is statistically significant. The conclusion from these results is that the default case is fairly aggressive. This conclusion is supported by the fact that many cases have higher number of UAVs remaining and none has a significant advantage in detection percentages.

The final behavior, exploration, is marked by high coverage percentage and low percentage of unexplored areas. Additionally, the number of remaining UAVs in an exploration case is also expected to be high, since the more UAVs there are; the more likely unique cells will be explored. Case 4 has all of these characteristics, with a high coverage (3.2%, t = 1.7, n =600), low unexplored area (-3.4%, t = -1.1, n =600), and high number of remaining UAVs (9.0%, t = 11.4, n =600). The difference in coverage and the number of surviving UAVs is significant at 90% confidence. The unusual characteristic of this case is that its control settings are somewhat surprising. It is a similar case to the safe behavior Case 5 in that the only difference in the control settings from the default case is threat sensitivity. Case 4 differs from Case 5 by the magnitude of the change. Case 5 sets the threat sensitivity to 100 while Case 4 sets it to 10. It appears that the factor of 10 increase in the threat sensitivity changes the swarm behavior from explorer to careful. This behavioral change is

sensible because if the swarm becomes too "fearful" of threats, it will be unable to explore as effectively. The fact that some restraint in nearing threats results in good exploration characteristics makes sense because as mentioned earlier, it is easier to explore when more UAVs remain until the end of the scenario.

Figure 25 shows the effects of using these different behavior settings. It is a snapshot of the same scenario unfolding at the same time with the swarms executing the different behaviors. On the left side of the image, an aggressive behavior was used while a careful behavior was chosen for the image on the right side. Notice the large buffer zone between the threats and UAVs with the careful swarm and the close proximity of the UAVs to the same threats in the aggressive swarm. The aggressive swarm has lost more members at this point – it has 362 UAVs left while the careful one has 392 left.

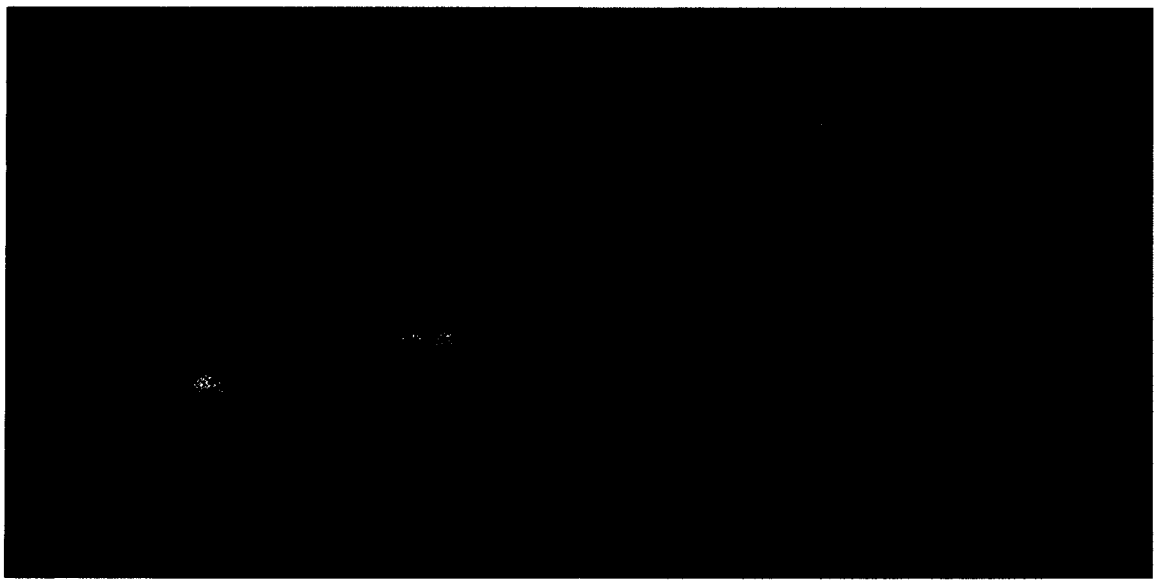

Figure 25. Swarm behavior examples

The most surprising result might be that only a few control settings actually have an impact on the chosen behaviors. Most of the other control settings provide few advantages (usually only in saving UAVs) while providing mostly disadvantages. For example, changing the evaporation and propagation constants resulted in only small negative changes. In case 7, the high sensitivity to UAV repulsion pheromone seemed to make it a good candidate for exploration. Instead it turned out to be bad at everything. This result was unexpected, but it might have to do with the fact that at least a trace of UAV repulsion pheromone is likely to exist in all cells around the UAV. With such a high sensitivity to this commonly encountered pheromone type, the negative effect on the length of a cell due to this pheromone type, even in cells with only a little, would overcome the effect of the other pheromone types (which often only have trace amounts in a cell). It would also overcome the shift designed to make empty cells have a chance at selection. As a result, the chosen cell will often be chosen randomly (this occurs when all cells have the same length – in this case zero). As is shown later, random selection of cell direction is by far sub-optimal.

Analyzing the effectiveness of roulette selection

ADAPTIV uses roulette selection to choose among a cell's nearest neighbors to determine the next goal state. This choice is sensible since roulette selection is a common tool in genetic and evolutionary programming since it prefers good options while not always taking the best. Taking good choices more often is clearly the smart option, but one might wonder if the best choice should always be chosen. Such an approach is known by different names in varying fields. In computer science an algorithm that chooses the best choice at the time is a greedy algorithm and in robotics such a control strategy is referred to as a gradient

follower. While these types of algorithms are very successful in certain applications, they often fail due to their biggest weakness – local optima. Once these algorithms find the best choice in an area, they will always choose it, even though a better one may exist somewhere else the algorithm cannot sense at that moment. In the exploration task of finding targets and threats, such a quality means that a UAV that found a target would not be able to leave it unless it could sense another one in its immediate vicinity.

To support this intuition, a gradient following version of ADAPTIV and a random selector were created. The only modifications were to the fragment program that chooses the next goal state (ADAPTIV step three). In the gradient version, the fragment program chose the longest bin. Like the roulette selection program, the gradient fragment program is too complex to include here. In fact, the gradient program was the most difficult to write and fit within the GPU's limited memory, highlighting some of the challenges of GPU programming. The same fifteen control setting cases that were used with roulette selection were run with gradient selection. Because random selection does not use any of these control settings, it was run as one case (case 31). Figure 26 shows the results of these cases along with the fifteen roulette selection cases.

Figure 26 shows the same five ADAPTIV output variables shown in Figure 24: the number of UAVs, percentage of threats detected, percentage of targets detected, coverage, and unexplored locations. Like Figure 24, it shows these values as a percentage increase or decrease of some baseline set of values. In Figure 24, those baseline values were the output of Case 1. This allowed for the comparison of the various roulette selection cases with the

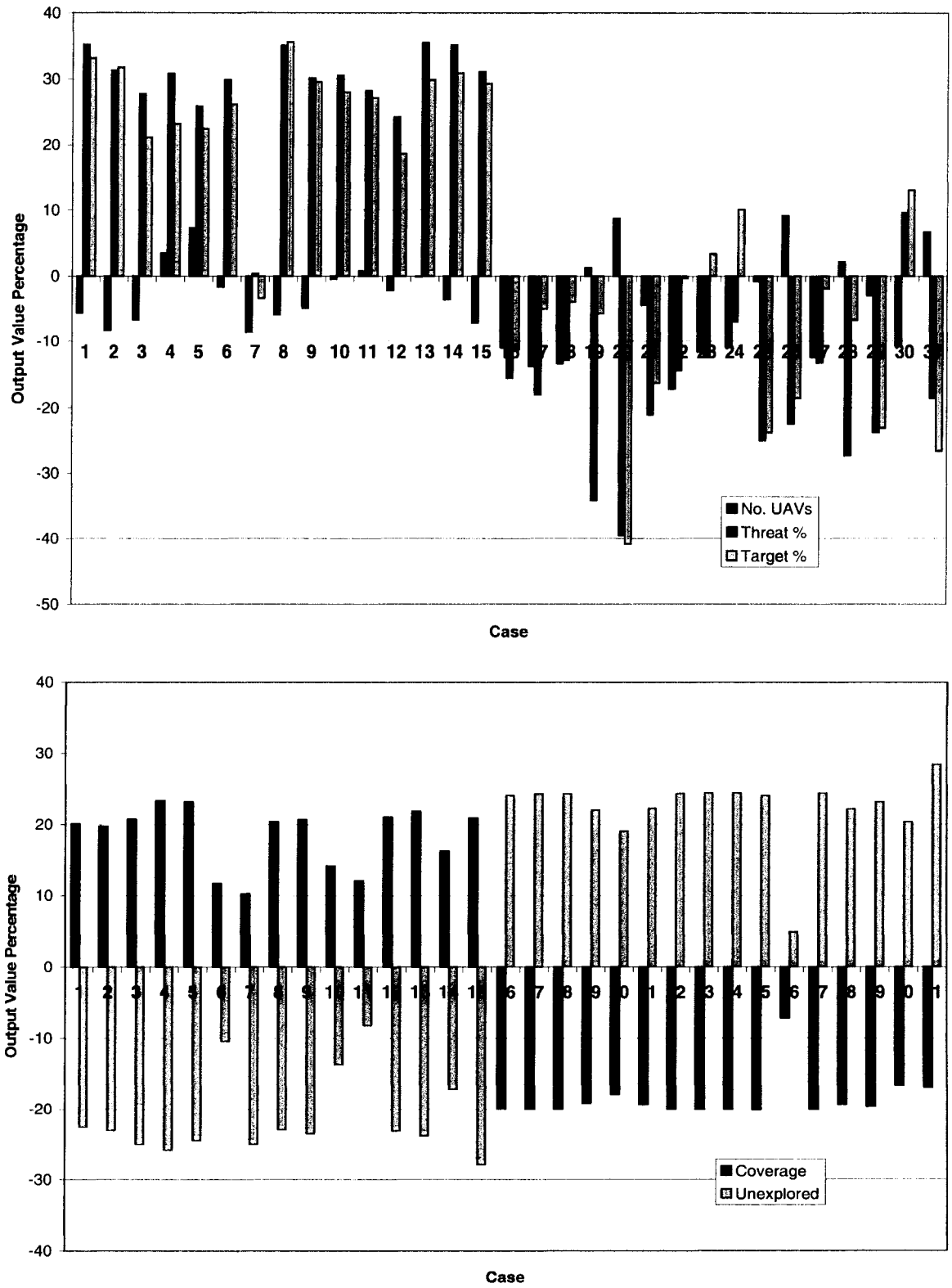

Figure 26. Output values compared to the overall averages

default roulette selection case. Figure 26 is intended to allow comparisons between the selection cases. Therefore, the set of baseline values chosen for this graph are the average values of each output variable over all the cases. For example, with respect to the output variable, the number of remaining UAV's, on average for all cases 90% of the UAV's survived. This percentage then becomes the baseline of zero and then each case would either have a greater, smaller or equal percentage of surviving UAV's as that baseline. Therefore, Figure 26 shows the output values of each case as a percentage increase or decrease of its overall average value with respect to each output and accounts for all 31 cases. Note that cases 1-15 use roulette selection, 16-30 use gradient selection and case 31 uses random selection.

The results help to vindicate the choice of roulette selection in ADAPTIV. With the exception of case 7, the roulette selection cases all have detection percentages and coverage well above average. Save for case 7, the worst roulette case has better qualities than even the best case of the other selection types. As previously mentioned, Case 7 behaves like a hybrid between roulette and random selection and Case 33 explains its poor performance. The number of UAVs remaining is similar across all of the selection methods and the unexplored cells are far fewer for roulette than the other cases. The gradient cases suffer low detection rates and coverage because of the local optima problem. Local optima also explain why more targets are found than threats in gradient selection. The random selection case does a good job of protecting UAVs, but a particularly poor job of detection and coverage. From these results, it is clear that roulette selection outperforms the other selection methods.

CONCLUSION

My research is a beginning – it lays out a vision for a next-generation UAV swarm control station – a vision that is tested with the prototype system outlined in this dissertation. It is important to remember that currently a battle commander must look at multiple screens to find important tactical information such as the radar readings from an AWACS, video feed from a UAV, and geographical information. Furthermore, the commander must mentally position the data from all these sources into a unified model. Additionally, it is important to remember that the use of ADAPTIV is hindered by computational costs. The research done for this dissertation has provided important insights into how to overcome these limitations. Specifically, the contributions of my research are:

- creating a user interface that allows the operator to manage a swarm of UAVs while keeping situational awareness of the entire engagement.
- offering a means to generate individual goal states for the UAVs in the swarm (e.g. λ-agents) without requiring operator attention.
- obtaining a performance advantage by converting ADAPTIV to the GPU.

In any swarm control system, the operator's attention is going to be the most limited resource. As a result, the ability to generate a comprehensive synthetic battlefield that provides interaction methods such as the swarm interaction menu and graphical information sources such as the ability to overlay the pheromone field on the terrain will be critical. UAV swarm control interfaces need to include methods to focus the operator's attention when

needed, maximize the operator's view of the battle and minimize the cognitive load on the operator. My system was designed and modified with these requirements in mind.

The Virtual Battlespace, with its comprehensive synthetic battlefield, provides the operator with an excellent view of the battle, especially if displayed on a large system such as a CAVE. The operator can change perspectives, views, and scales on the fly to help visualize the battlefield and alerts focus the operator's attention to where it is most needed. The Virtual Battlespace and ADAPTIV help to minimize the cognitive load on the operator, by fusing many different data streams onto one unified display, freeing the operator from having to maintain a mental picture of how the data streams interact. In addition ADAPTIV further helps minimize operator cognitive load by providing a means to control the behavior of the swarm without having to control each unit separately.

An important consequence of my research is that training simulators could be quickly developed and deployed to prepare our warfighters for swarm interaction. There is already an established tradition of using simulators for training in the military and the systems created for my research could be incorporated within the U.S. military's existing systems. Doing so would give the military a tool to start training people on how to manage swarms. Additionally, the adoption of my research into training simulators would allow pilots to begin learning how to share airspace and complete missions with unmanned units. In addition to flying their planes, F-16 pilots must already practice on simulators which are so realistic that the time spent on them counts as flight hours. It is not hard to imagine simulated swarming UAVs flying in the same virtual battlefield as that displayed on the simulators.

My research also offers ideas on how to deploy this technology in a cost effective manner both as a simulator now and on the battlefields of the future. The primary idea

revolves around the use of the GPU. As shown in the results section, the GPU provides a noticeable improvement in the performance of parallelizable vector calculations. Since these are exactly the type of calculations performed in ADAPTIV, and algorithms like ADAPTIV are likely to be used to control swarm behavior, the GPU offers incredible value. It is 30 times faster than the CPU in some cases for the same monetary cost. This means the costs incurred from buying, maintaining, and powering 30 CPU's can be replaced by one GPU. This fact immediately impacts the bottom line of military training organizations by reducing the cost of training tomorrow's warfighters today.

To take advantage of the power of current GPUs in this research, the pheromone flavors had to be stored as 8-bit numbers as this is the best resolution of current GPUs. Consequently, the highest precision of a number between zero and one is 1/256. It is clear that additional resolution would be useful. This increased resolution will come in the next generation of GPUs that will boast 32-bit floating point calculations. More importantly, GPUs are improving at a faster rate than CPUs so their performance advantage only promises to increase.

My research makes extensive use of the ADAPTIV swarm control algorithm and it has proven to be effective in creating and evaluating the overall system. It does, however, have its limitations. The UAVs fly hectic patterns, often having to turn completely around to make the next goal. Such true insect-like patterns might confuse the enemy, but it would also consume fuel more quickly and watching the units fly tends to disorient people. Additionally, even with the UAV self-repulsion pheromone, the swarm is a little sluggish in getting to new areas. This latter problem may be fixed with more tweaking of the control settings. However, despite ADAPTIV's limitations, it still provides a useful base to explore

human interaction with a UAV swarm. The military has agreed with us by awarding an Air Force Office of Scientific Research grant of $2.8 million to the VRAC research team to continue our swarm control research [43]. Stay tuned.

FUTURE WORK

The initial test results hint at this prototype's great potential, but there is a mountain of work still to be done on this topic. My research will be generating future work for years to come. In the short term, the effectiveness of the prototype's interface will be tested with a user study. A research group at VRAC will conduct a study in the fall of 2005 to measure how effectively the swarm management system allows the operator to manage the swarm. This test will give the operator control of a single swarm maneuvering within a large battlefield containing both manned and unmanned simulated units. Several alerts will be generated by the swarm, and the operator's task will be to use the swarm to gather intelligence about the alerts (via video) and decide if the alert should be attacked. The video will be either of a civilian convoy or an enemy military convoy. The operator's performance in correctly classifying these alerts as hostile or noncombatants will be the focus of the study. Additionally, the operator's ability to maintain situational awareness of the battle as a whole using the Virtual Battlespace will be tested. Questions will be asked after the test run about the battle and how it turns out. This initial user study should provide invaluable insights on how to improve the interface and how to focus my next user study.

In addition to continually upgrading the visual quality and increasing the amount of graphical information available in the Virtual Battlespace, the swarm system itself will see further development. It would be fascinating to see how combinations of pheromone control settings would affect swarm behavior. The dynamics simulation of the swarming UAVs could be replaced with a more sophisticated aircraft mathematical model. ADAPTIV's goal states could be augmented by another control algorithm, perhaps Reza Olfati-Saber's, to

smooth out the path of the UAVs. The operational area of the swarm, currently static and set at application start, could be made dynamic in that the operator could change its size or move the swarm to different parts of the battle. The prototype system could be run in a distributed manner so multiple operators could share the same virtual battle. This collaborative swarm control application could be run at several different locations across the country to address the difficulties created by signal delay. Different methods for representing the users within the virtual world could be tested for their effectiveness in enabling collaboration.

In the mid-term, ADAPTIV could be replaced by another swarm control method and the two could be compared. A version of the research could be created to be run on a GPU cluster. A cluster of GPUs would be useful if one GPU cannot provide enough power to run ADAPTIV, as might be the case with extremely large swarms or very large battlefields. This is already a proven strategy to boost the performance of CPU's beyond their current limits. Stony Brook University has recently introduced a GPU cluster and has used it successfully to calculate air borne contaminant spreading in Manhattan [44]. The main complexity in using a GPU cluster is syncing up the different nodes to create one uniform application state. Again, this same problem affects CPU clusters and syncing efforts can often be the limiting factor in the cluster's performance. The research group at Stony Brook is exploring ways to speed up this syncing. The application can aid in this endeavor by minimizing the data that must be sent to each node in the cluster to sync the application state. If the cluster is spread out geographically such that each node is responsible for a different area in the space, then the problem of syncing is usually one of matching the borders [43]. In a clustered ADAPTIV, that would only entail syncing up the borders of the pheromone field. Recall that the second ADAPTIV step, pheromone propagation, requires information from neighboring

cells. This is the information that would need to be sent between nodes of the GPU cluster. Each cell consists of four 8-bit numbers and with a potential 2048 cells on a side (limited by the likely maximum texture size to be supported by GPUs), that is approximately 8.2KB per side, or 32.8KB per GPU. Data compression techniques might also be used to reduce the size of the data that must be shared.

The reader wonder if there would also be a need to send information across the cluster about UAVs near the edge of a GPU's area of influence. However, the UAVs are stored on the CPU, and therefore do not directly interact with the GPUs. The only tricky part would come in dropping input (R) pheromones. In the non-clustered version, the CPU draws a point on the screen at a location that translates to the same point in the pheromone field. This convenient conversion exists because there is only one GPU, and therefore, only one pheromone field texture to send to the GPU. This means that the entire field is managed by one GPU, so the field is not split up. In the case of multiple GPUs, the field would be split into regions, with each GPU in charge of one. If texture resolution is sufficiently large, one field could still be maintained by the CPU and then split up as its pieces are sent to the GPUs. In this case, the point would be drawn into the global pheromone field the same way it would with a single GPU. If the resolution is not sufficient, then the pheromone field would be stored in chunks and the CPU would have to draw the R pheromone point in the correct location of the appropriate chunk. This is more complicated, of course, but not difficult.

Another mid-term improvement of the system would come from the development and deployment of a training program for user testing. Research UAVs could be instrumented with communications capability sufficient to create a very small "swarm" of three or four planes. Voice commands and eye tracking could be included to add new ways for the

operator to interact with the virtual world and the swarm. The operator's interface could be placed within vehicles likely to house such systems in the near-future such as a HUMVEE or rotorcraft. In the long term, larger swarms would be tested until the swarm size reached that desired by the military. Further user studies would allow for continued improvement of the interface. Security and failure analysis would be addressed as the system matured.

ACKNOWLEDGEMENTS

This work is the culmination of the four and a half years I spent as a research assistant at the Virtual Reality Applications Center. The Virtual Battlespace, the platform on which my research is attached, was started in 2000 and has evolved to its present state today (2005). The evolution of the Virtual Battlespace involved the hard work of many talented people. I thank all of the researchers that worked with me on the various generations of the Virtual Battlespace for their efforts. These researchers include Tom Batkiewicz, Liangshou Wu, Mathew Wach, Mark Knight, Chad Austin, and Jared Knutzon. The application, and as a result, my research was made much more professional in appearance thanks to talented artists including Ted Martens, Karl Hauber, and Josh Delaney. My apologies to the researchers and artists involved with the Virtual Battlespace project not mentioned.

I thank Greg Luecke for helping me find literature on flocking algorithms, and James Bernard for his sage advice and introducing me to the VRAC all those years ago. Dirk Reiners' excellent Advanced Graphics course provided the impetus for this whole research. It was in his class that I learned how to program GPUs. Adrian Sannier and James Oliver, my co-major professors, are veterans of the Virtual Battlespace project and I would not have gotten this far without their mentoring, insight, encouragement and, of course, fund-raising. I am truly blessed with an insightful, knowledgeable and energetic committee.

This research, and the Virtual Battlespace, was developed at the VRAC. The VRAC would grind to a halt without the amazing expertise of our system administrator, Glen Galvin. His IT coworker, Kevin Teske, has also been invaluable to my research with his boundless energy and willingness to make sure the lab machines are working. He also

offered his time to film the stock footage for the promotional videos of the Virtual Battlespace and my research. Thanks goes to the entire VRJuggler team for their development of the library that turns the Virtual Battlespace, and my research, into a true VR application. Finally, I would like to thank the wonderful administrative staff at VRAC who helped me get demos scheduled and made sure all forms got signed. This staff includes Karen Koppenhaver, Lynette Sherer and Beth Hageman.

I would like to extent special thanks to my family and friends for helping me get through my graduate career. Finally, and most importantly, I thank my wife, Jessica Walter, for her love, support, enthusiasm, and her professional writing and editing talents. Thanks to you all!

REFERENCES

[1] Parunak, V., Purcell, M., O,Connell, R., "Digital Pheromones for Autonomous Coordination of Swarming UAV's", American Institute of Aeronautics and Aerospace, 2002.

[2] US Department of Defense, Unmanned Aerial Vehicles Roadmap 2002-2027, http://www.acq.osd.mil/usd/uav_roadmap.pdf, December 2002.

[3] Newcome, Laurence R., **Unmanned Aviation: A Brief History of Unmanned Aerial Vehicles,** American Institute of Aeronautics and Astronautics, Inc., ISBN: 1563476444, 2004.

[4] Charles B., Zimet, E., "UCAVs Technological, Policy, and Operational Challenges," *Defense Horizons*, Number 3, Center for Technology and National Security Policy, National Defense University, 2001.

[5] Grant, Rebecca, "Reach-Forward," *Air Force, Journal of the Air Force Association*, vol. 85.10, 2002.

[6] Adams, Charlotte, "UAVs That Swarm," *Avionics Magazine*, retrieved October 2004 from http://www.defensedaily.com, 2003.

[7] Johnson, George. "Ideas & Trends: Who Do You Trust: G.I. Joe or A.I. Joe?" *New York Times,* retrieved Feb. 2005 from www.nytimes.com, 2005.

[8] Harris, M., Coombe, G., Scheuermann, T., Lastra, A., "Physically-Based Visual Simulation on Graphics Hardware, Department of Computer Science at North Carolina University", *Graphics Hardware,* 2002.

[9] Fernando., R., **GPU Gems**, Addison-Wesley, N-Vidia Corporation, ISBN: 0321228324, 2004.

[10] Walter, B., Knutzon J., Sannier A., Oliver J., VR Aided Control of UAVs. *3rd AIAA Unmanned Unlimited Technical Conference,* Paper AIAA 2004-6320, Chicago, 2004.

[11] Knutzon, J., Walter, B., Sannier, A., Oliver, J., "Command and Control in Distributed Mission Training: an Immersive Approach", *NATO Conference*, August 2003.

[12] Knutzon, J., Walter, B., Sannier, A., Oliver, J., "An Immersive Approach to Command and Control", *Journal of Battlefield Technology*, March 2004.

[13] Homepage of VRJuggler.org, visited August 2005 from http://www.vrjuggler.org.

[14] Alberts, D., *Information Age Transformation: Getting to a 21st Century Military*, Washington, DC: CCRP Publication Series, June 2002.

[15] Posdamer, J., Dantone, J., Gershon, N., Hamburger, T., Page, W., "Battlespace Visualization: A Grand Challenge," *Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS'01)*, San Diego, CA, October 2001.

[16] Jense, G., Kuijpers, N., Dumay, A., "DIS and HLA : Connecting People, Simulations and Simulators in the Military, Space and Civil Domains," *Simulations and Simulators in the Military, Space and Civil Domains*, 48th International Astronautical Congress, Turin, Italy, October 6-10, 1997.

[17] Durbin, J., Swan II, J., Colbert, B., Crowe, J., King, R., King, T., Scannell, C., Wartell, Z., Welsh, T.,. "Battlefield Visualization on the Responsive Workbench", *Proceedings IEEE Visualization '98*, IEEE Computer Society Press, pp. 463-466, 1998.

[18] Hix, D., Swan II, J., Gabbard, J., McGee, M., Durbin, J., King, T., "User-Centered Design and Evaluation of a Real-Time Battlefield Visualization Virtual Environment", *Proceedings IEEE Virtual Reality '99*, IEEE Computer Society Press, pp. 96–103, 1999.

[19] Walter, B., "Virtual Reality Aided Teleoperation," Thesis, Mechanical Engineering Department, Iowa State University at Ames, August 2003.

[20] Knutzon, J., "Tracking and Control Design for a Virtual Reality Teleoperation System," Thesis, Computer and Electrical Engineering Department, Iowa State University at Ames, August 2003.

[21] Parunak, Van Dyke, "Go to the Ant: Engineering Principles from Natural Multi-Agent Systems", Altarum Institute, *Annals of Operations Research,* Vol. 75, 1997.

[22] Holway, D., Lach, L., Suarez, A., Tsutsui, N., Case, T., "The Causes and Consequences of Ant Invasions", *Annual Review of Ecological Systems*, Vol. 33, 2002.

[23] Ferreria, Candida, **Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence**, ISBN: 9729589054, December 2002.

[24] Gaudiano, P., "Swarm Intelligence: a New C2 Paradigm with an Application to Control of Swarms of UAVs", Icosystem Corp. & Air Force Research Lab, *8th ICCRTS Command and Control Research and Technology Symposium*, 2002.

[25] Prieditis, A., Dalal, M., Arcilla, A., Groel, B., Van Der Bock, M., Kong, R., "Model-Based Swarm Control of Unmanned Ground Vehicles", Lookahead Decisions Inc., *Military Government and Aerospace Simulation Symposium*, 2003.

[26] Kadrovach, B., Lamont, G., "A Particle Swarm Model for Swarm-based Networked Sensor Systems", Air Force Institute of Technology, Wright-Patterson AFB, 2003.

[27] Murphy, R., **Unifying Artificial Intelligence Robotics: An Undergraduate Robotics Textbook: Introduction to AI Robotics**, MIT Press, 2000.

[28] Walton, Marsha, "Robots Fail to Complete Grand Challenge," CNN Website, http://edition.cnn.com/2004/TECH/ptech/03/14/darpa.race/, visited January 2005, May 6, 2004.

[29] Singh, S., Simmons, R., Smith, T., Stentz, A., Verma, V., Yahja, A., Schwehr, K., "Recent Progress in Local and Global Traversability for Planetary Rovers," *IEEE Conference in Robotics and Automation,* San Francisco, April 2000.

[30] Olfati-Saber, Reza, "Flocking for Multi-Agent Dynamic Systems: Algorithms and Theory," *IEEE Transactions on Automatic Control,* Technical Report CIT-CDS 2004-005, June 2004.

[31] "Products Page: Automatic Target Recognition Toolbox," C&P Technologies Website, http://www.cptnj.com/web/prod_atr.htm, visited July 2005.

[32] "Financial Releases Page: Rockwell Collins selected by Army for MCAP III program," Rockwell Collins Website, http://www.shareholder.com/col/ReleaseDetail.cfm?ReleaseID=141058, visited July 2005.

[33] Hall, L., Eschrich, S., "Draft Final Report: Robust Recognition of Interesting Objects in Images," prepared for the Army Research Lab, November 2000.

[34] Brown, T., Doshi, S., Jadhav, S., Himmelstein, J., "Test Bed fro a Wireless Network on Small UAVs," *AIAA 3rd Unmanned Unlimited Technical Conference,* Chicago, September 2004.

[35] Gerla, M., Xu, K., Hong X., "Exploiting Mobility in Large Scale Ad Hoc Wireless Networks," *IEEE Computer Communication Workshop* (CCW 2003), Dana Point, CA, 2003.

[36] Clough, Bruce, "UAV Swarming? So What Are Those Swarms, What Are The Implications, And How Do We Handle Them?", Air Force Research Lab, AFRL-VA-WP-2002-308, 2002.

[37] "Predator," NASA Website, http://uav.wff.nasa.gov/UAVDetail.cfm?RecordID=Predator, visited March 2005.

[38] "Intelligence: MQ-9A Predator," Global Security website, http://www.globalsecurity.org/intell/systems/predatorb.htm, visited March 2005.

[39] "F-18 HARV High Alpha Research Vehicle Movie Gallery," Dryden Flight Research Center NASA Website, http://www.dfrc.nasa.gov/Gallery/Movie/F-18HARV/HTML/EM-0013-02.html, visited March 2005.

[40] Gamma, E., Helm, R., Johnson, R., Vlissides, J., **Design Patterns,** Addison-Wesley Professional, ISBN: 0201633612, January 1995.

[41] Mulder, Jurriaan D., "Menu Selection in Desktop Virtual Reality," Center for Mathematics and Computer Science CWI, Amsterdam, the Netherlands, prepared for the *Central European Multimedia and Virtual Reality Conference,* 2005.

[42] Reiners, Dirk, "Computer Science 657: Advanced Topics in Computer Graphics class notes," Iowa State University class, taken Fall 2004.

[43] Krapfl, Mike, "Air Force awards virtual reality center $2.8 million," Iowa State University News Service, retrieved from http://www.iastate.edu/~nscentral/releases/2005/jun/af.shtml, visited July 2005, June 2005.

[44] Fan, Z., Qiu, F., Kaufman, A., Yoakum-Stover, S., "GPU Cluster for High Performance Computing", Stony Brook University, *ACM/IEEE Supercomputing Conferenc*e, 2004.