REAL TIME MISSION PLANNING FOR VIRTUAL HUMAN AGENTS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÇAĞATAY ÜNDEĞER

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF COMPUTER ENGINEERING

JANUARY 2001

Approval of the Graduate School of Natural And Applied Sciences.

Prof. Dr. Tayfur ÖZTÜRK
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master Science.

Prof. Dr. Ayşe KİPER
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master Science.

Assoc. Prof. Dr. Faruk POLAT
Supervisor

Examining Committee Members :

Assoc. Prof. Dr. Faruk POLAT _____

Assoc. Prof. Dr. Veysi İŞLER _____

Assoc. Prof. Dr. Göktürk ÜÇOLUK _____

Assist. Prof. Dr. Halit OĞUZTÜZÜN _____

Y. Müh. Yb. Ziya İPEKKAN _____

ABSTRACT

REAL TIME MISSION PLANNING FOR VIRTUAL HUMAN AGENTS

Ündeğer, Çağatay
M.S., Department of Computer Engineering
Supervisor : Assoc. Prof. Dr. Faruk Polat

January 2001, 94 pages

Path searching and mission planning are challenging problems in many domains such as wargames, robotics, military mission planning, computer-generated forces, etc. The objective of this study is to develop a multi-agent system for virtual human agents on three-dimensional large landscapes (terrain) to accomplish a specified mission by group synchronization. The terrain contains natural and artificial entities such as rivers, lakes, forests, rocks, roads, houses, bridges, etc. The agent groups enter a specific area to perform a specified mission, which may be to attack, escape or just pass through a selected tactical area. The terrain contains static and dynamic platforms that carry different kinds of sensors such as DayTV, infra-red, radar, night-vision. The goal of the agents is to complete their mission under control of a group commander without being detected or caught by any platform. Monitors of the platform sensors are

observed by the user at tactical command center in order to make the detection process more realistic. The agents plan path in real-time and follow their path in order to complete the mission. For that purpose, an off-line path planning, a real-time path update, and a real time goal directed path search algorithm are proposed to find suitable routes passing through mission control points considering the terrain, weather and the threat information known or gathered on the fly. When an agent is detected or identified, it tries to alter its plan to accomplish the mission.

Keywords : Computer generated forces, mission planning, off-line path planning, real-time path update, multi agent real-time search.

ÖZ

SANAL İNSAN ETMENLER İÇİN GERÇEK ZAMANLI GÖREV PLANLAMA

Ündeğer, Çağatay
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Faruk Polat

Ocak 2001, 94 sayfa

Güzergah ve görev planlama, oyunlar, robotik, askeri görev planlama, yarı otonom kuvvetler gibi bir çok çalışma alanın ilgi konusu olmuştur. Bu çalışmanın amacı, sanal insan etmenlerle, geniş arazi üzerinde bir görevi grup senkronizasyonunu da göz önünde bulundurarak tamamlamak için bir çoklu etmen modeli geliştirmektir. Arazi üzerinde, nehirler, göller, ormanlar, kayalar, yollar, evler, köprüler gibi doğal ve yapay detaylar bulunmaktadır. Etmenlerimiz önceden tanımlanmış bir görevi başarmak (saldırı, kaçış, intikal vb.) için stratejik bir alana girmektedir. Bu alanda, gündüz kamera, infra-red kamera, radar, gece görüş cihazı gibi farklı tip sensörleri üzerinde bulundurabilen statik ve dinamik platformlar bulunmaktadır. Etmenlerin amacı, herhangi bir platform tarafından

tespit edilmeden veya yakalanmadan bir grup lideri öncülüğünde görevini tamamlayabilmektir. Tespitin gerçekciliğini arttırmak için platform sensörlerin görüntüleri kullanıcılar tarafından taktik komuta merkezinden izlenmektedir. Etmenler, gerçek zamanlı olarak güzergah planlamakta ve görevi tamamlamak için planlanan güzergahı izlemektedir. Kontrol noktalarından geçen uygun güzergahların gerçek zamanlı güncellenen bilgilerle, arazi, hava koşulları ve düşmanı da göz önünde bulundurarak tespit edilebilmesi için bir çevrim dışı güzergah planlama, bir gerçek zamanlı güzergah güncelleme ve bir de çoklu etmenli gerçek zamanlı arama algoritması geliştirilmiştir. Her hangi bir etmen düşman tarafından tespit edilir veya tanınırsa, görevini tamamlamak için planını değiştirmeye çalışacaktır.

Anahtar Kelimeler : Yarı otonom kuvvetler, görev planlama, çevrim dışı güzergah planlama, gerçek zamanlı güzergah güncelleme, çok etmenli gerçek zamanlı arama.

Dedicated to my parents,

ACKNOWLEDGEMENTS

I would like to thank my supervisor Assoc. Prof. Faruk Polat for his encouragement and guidance during the development of my thesis. Also, I would like to thank Assoc. Prof. Veysi İşler, Asist. Prof. Halit Oğuztüzün, Ziya İpekkan, Şükrü Bilir and all TUAF-METU ModSim Staff for their great effort to get work done.

TABLE OF CONTENTS

ABSTRACT iii

ÖZ v

ACKNOWLEDGEMENTS viii

TABLE OF CONTENTS ix

LIST OF FIGURES xiii

LIST OF TABLES xvii

CHAPTERS

1 INTRODUCTION 1

1.1 The Subject 1

1.2 Scope and Objective 2

1.3 Outline 3

2 RELATED WORK 4

3 SIMULATION ARCHITECTURE 8

3.1 Introduction 8

3.2 Static Environment 11

3.2.1 Landscape 11

3.2.1.1 Terrain: Polygons vs. Voxels 11

3.2.1.2 Loading DEM Files 14

3.2.1.3 Generating Slope Matrix 15
3.2.1.4 Mapping Texture 17
3.2.1.5 Surface Materials 18
3.2.2 Features 21
3.2.3 Weather Conditions 21
3.3 Dynamic Entities 23
3.4 Command & Control Systems 25
3.4.1 Test Environment 25
3.4.2 Command Center 25
3.4.3 Driver Consoles 26
4 DISPLAYING ENVIRONMENT INFORMATION 28
4.1 Visualization 28
4.2 Orthographic Viewing 28
4.2.1 Displaying Elevation Data 28
4.2.2 Displaying Slope Magnitudes and Directions 29
4.2.3 Displaying Texture 31
4.2.4 Displaying Mixed Information 32
4.2.5 Displaying Surface Materials 34
4.2.6 Displaying Weather Conditions 34
4.3 3D Rendering 35
5 AGENTS 38
5.1 Introduction 38
5.2 Goal Description (Mission) 38

5.3 Perceptions....40
5.3.1 How Perceptions Work....40
5.3.2 Seeing....40
5.3.3 Being in Line of Sight....41
5.3.4 Being in Viewing Frustum....43
5.3.5 Stochastic Detection....45
5.3.5.1 Introduction....45
5.3.5.2 Detection Range....46
5.3.5.3 Weather Condition....46
5.3.5.3.1 Ideal Conditions....46
5.3.5.3.2 Deviate From Ideal Conditions....47
5.3.5.4 Plant Cover....49
5.3.6 Hearing....50
5.4 Classifying and Storing Perceptions....51
5.5 Decision and Reaction....55
5.6 Group Formation....55
5.7 Group Synchronization by Radio Communication....57
5.8 Location Availability....58
5.9 Physical Modeling....61
6 PATH PLANNING....62
6.1 The Objective....62
6.2 Off-line Path Planning....63
6.3 Real-Time Path Refinement and Update....70

6.3.1 Energy Minimization 70
6.3.2 Internal Energies 70
6.3.3 External Energies 72
6.4 Real-Time Path Search: Real-time Horizontal A* 75
7 PERFORMANCE ANALYSIS 80
7.1 Test Platform 80
7.2 Test Results of Real-Time Horizontal A* 84
8 CONCLUSION AND FUTURE WORK 89
REFERENCES 91

LIST OF FIGURES

FIGURES

3.1 Architecture context diagram of the simulation 8

3.2 The static environment and the dynamic entities form the whole environment 9

3.3 Views captured from a DayTV camera and an infra-red camera.....10

3.4 A polygonal based terrain and a voxel-based terrain generated from a matrix of elevations 11

3.5 An image rendered using both digital elevation data and polygons. 13

3.6 Four elevations let two different triangulation styles 15

3.7 Calculating slope directions and magnitudes.....16

3.8 Visualization of slope directions and magnitudes of a terrain 17

3.9 The texture is mapped on the terrain using two diagonal corners, which are given as geographic coordinates 18

3.10 Rivers are extracted from the texture using color similarity.....20

3.11 Static features 21

3.12 Regions having various weather conditions.....22

3.13 The group structure of the dynamic entities.....24

3.14 A snapshot from command center 26
3.15 A snapshot from a driver console 27
4.1 Visualization of elevation matrix 29
4.2 Visualization of slope magnitude matrix 30
4.3 Visualization of slope direction matrix.....31
4.4 Visualization of texture matrix.....32
4.5 Visualization of elevation and slope direction matrix using alpha blending ...33
4.6 Visualization of surface materials blended on slope direction matrix.....34
4.7 Visualization of weather condition distribution blended on slope direction matrix.....35
4.8 3D Visualization of the plant-cover density.....36
4.9 Polygonal and voxel rendering.....37
5.1 Scenario control points for two agent groups.....39
5.2 Two points are in line of sight of each other.....42
5.3 Two points are not in line of sight of each other.....42
5.4 Line of sight test 43
5.5 Viewing frustum of an agent.....44
5.6 Testing viewing frustum 45
5.7 A probability graph, which is constructed from a set of sample points 47
5.8 The graph on the left is probability distribution for ideal condition and the one on the right is a scaled graph multiplied by 0.7 48
5.9 The image on the left is the coverage of a DayTV while there is no rain and the one on the left is while there is heavy rain.....49

5.10 The effect of plant cover density and viewing angle 50
5.11 Perception is classified into three categories: detection, recognition and identification 51
5.12 A sample scenario for similarity detection.....52
5.13 Group formation of a group with a commander and the three subordinates..56
5.14 A group of agents walking in a formation 56
5.15 Location availability matrix after culling voxels above 1500 m.....59
5.16 Location availability matrix after culling voxels having slope magnitude more than 30 degree 60
5.17 Location availability matrix after culling voxels above 1500 m, having slope magnitude more than 30 degree and having material water, rock or forest.....60
5.18 The 3D human model used for agents 61
6.1 The image shows an off-line path search result between two given points.....63
6.2 A set of tactical control points and generated paths between these points 65
6.3 The initial state of the algorithm 66
6.4 The recursive search process continues until reaching the line that the target point is on.....67
6.5 An off-line generated path and an optimized version of the same path 68
6.6 Illustration of a high and low curvature.....71
6.7 Illustration of a balanced and not balanced path 71
6.8 Threat energy causes the path points go far away from the threats.....73
6.9 A path update sample.....74

6.10 The path on the left aims to flow right, but an obstacle prevents this movement....74
6.11 State transition of Real-Time A*....75
6.12 The agent, directed by Real-Time A* may stuck in semi-closed regions having large open areas inside for a long time searching the same voxels hopelessly....76
6.13 State transition of Real-Time Horizontal A* 77
6.14 The agent, directed by Real-Time Horizontal A* can go out of semi-closed regions quickly....77
7.1 The test program for Real-Time A* and Real-Time Horizontal A* 80
7.2 Test region 1....81
7.3 Test region 2....82
7.4 Test region 3....82
7.5 Test region 4....83
7.6 Test region 5....83
7.7 Test region 6....84
7.8 Moves that are not allowed for both RTA* and RTHA* 85
7.9 Performance of RTA* and RTHA* on a widely open area 87
7.10 Having large horizontal regions has a bad effect on RTHA* 88

LIST OF TABLES

TABLES

3.1 Type definition of RecordA and RecordB 14

3.2 Loading a DEM File 15

5.1 The algorithm, which controls the agents 54

5.2 The list shows a set of radio communications occurred on a mission.....58

6.1 The basic algorithm for the off-line path planning.....69

6.2 The basic algorithm for Real-Time Horizontal A* 78

7.1 RTA* vs. RTHA*.....86

CHAPTER I

INTRODUCTION

1.1 The Subject

Multi agent systems can be used to model computer-generated environments where intelligent agents react suitably to various events. Many of the applications in this context need realistic environment generation, efficient search algorithms and heuristics suitable for real-time simulations. Multi agent systems are integrated into these simulations for supporting automatic and semi-automatic human and group behaviors to complete a given mission. Planning a mission usually means to plan suitable paths and actions that lead to the goal-state.

The problem of path planning can be described as finding a sequence of state transitions through a graph from some initial state (starting point) to a goal state (target point), or determining that no such sequence exists. Path-planning algorithms can be off-line or on-line. Off-line path planning algorithms like A* [SP95] find the whole solution before starting execution. They plan paths in advance and usually find optimal solutions. Their efficiency is not considered to be crucial and the agent just follows the generated path. Although this is a good solution for a static environment, it is completely infeasible for dynamic environments, because if the environment or the cost function changes, the remaining path may need to be re-planned, which is not efficient for real-time applications. Real-time path planning algorithms such as Real-Time A* [MY96],

D* [A94], Focused D* [A95] are on-line and offer more efficient solutions. Some of them produce optimal solutions for dynamic changes such as D* and Focused D*, and some only bring efficiency but not optimality such as Real-Time A*.

1.2 Scope and Objective

In this thesis, we study dynamic simulation environments and mission planning algorithms in military applications. We have constructed a dynamic simulation environment, which consists of defense forces, assault forces and other entities, and furthermore developed suitable scenario generation, group synchronization and path planning algorithms meeting requirements of real-time applications.

Assault forces, which are intelligent bodies, learn about the environment, terrain and moving entities, and react according to changing situations under some assumptions. To support decision-making, path planning, and suitably reacting based on some goals, we have proposed techniques to gather, store and evaluate environment information.

We have proposed a new off-line path planning, a real-time path update, and an improved version of Real-Time A* [MY96] algorithm. The off-line path-planning algorithm does not guarantee optimal solutions, the generated paths do not contain loops and they are usually near to optimal path. It is efficient and may be applied to many problems. We used the basic idea behind our off-line path planning algorithm to modify Real-Time A* and managed to develop a new real-time search algorithm (Real-Time Horizontal A*), which performs better in large and mountainous landscapes.

As we need modifications on off-line generated paths on the fly during the simulation, we have also presented a path refinement and update algorithm based on energy minimization technique, which is frequently used in image processing called Snakes [EA98]. Thus, we combined the advantages of off-line path planning and real-time path search to support real-time simulation systems.

1.3 Outline

In Chapter II, a survey of related work on path planning and its background needs for simulation systems are given.

In Chapter III, the simulation architecture, which serves the environment and control systems for our study, is introduced in detail. The static and dynamic environment and the analysis performed on this datum are described on samples. The command and control interface that allows the users, the defense forces, to observe the environment is introduced in brief.

In Chapter IV, the analysis on the landscape and their displaying techniques are described in detail and also the 3D rendering concept for landscape rendering is briefly given.

In Chapter V, we introduce our intelligent agent concept and the agent capabilities: perception gathering, probability of detection, perception evaluation by similarity analysis, knowledge, group formation, group synchronization by radio communication, location availability and physical appearance.

In Chapter VI, the proposed path planning algorithms, their advantages and drawbacks are given in detail.

In Chapter VII, our test-bed is introduced. Test results and their evaluation are given on example test runs.

Finally, the conclusion and the future work are given in Chapter VIII.

CHAPTER II

RELATED WORK

Multi-agent systems are used in many domains such as robotics, computer generated forces, games, training, RoboCup soccer and their simulators. In robotics [SJ99] and Robocup soccer [RM98], intelligent planning aims to find out ways for interacting with the physical world, which makes the problem hard to solve. In contrast, intelligent planning for computer generated forces [RA98, CVZ00, JG99, E98] and games [M00] aim to generate behaviors similar to the real world in virtual environments. Simulating real world actions in a virtual environment is basically used to test some conditions that are not possible in the real world. Intelligent agents such as airplane, chopper, tank, soldier that behave much like real world entities are frequently used for pilot trainings in flight simulators [RJMP94, RMJP93]. In such simulations, realistic modeling of agent behaviors is important for the realism of training. In order to be able to plan actions realistically, deciding on appropriate parameters, modeling environment, and using suitable algorithms for gathering information are very important.

Parallel to the development of intelligent systems, the demand for integrating them to simulation systems also increased especially by the help of military domains. Thus, in 1999, the Defense Modeling and Simulation Office (DMSO) of USA established a working group of government and industry representatives and tied to decide the standards of intelligent agents within the High Level Architecture, HLA [PS99].

In simulation, environment modeling and visualization is very important. Although there are many powerful techniques that have capability of rendering very realistically such as ray-tracing [A97], and modeling landscapes and their plant covers characteristics natural such as the technique used in [OPBRMP98], they are not applicable to real-time simulations because of their efficiency problems. The 3D modeling and rendering techniques generally used in these kinds of simulations are rendering polygons or voxels by Z-buffer [A97] or voxel based rendering [DEUV96]. These algorithms offer efficiency, but they are usually lack of realism.

3D environments are generally represented using polygons or voxel representations (digital elevation data). In the proposed approach of Champhell [CM99], the terrain database and features are stored and optimized using triangles. In contrast, the terrain and features are modeled and rendered using voxel rendering techniques in [DEUV96].

In order to gather necessary information from the virtual environment, physical or stochastic methods can be used. In the proposed technique of Knuffner [JJ99], a physical based method is used. To collect information from the 3D environment and to check which objects are visible to a particular character, the scene is rendered off-screen from the character's point of view, using flat shading with a unique color (object ID) for each object. Knowledge base of agents is organized as link-lists to store the information about the objects that are seen. The proposed approach makes use of stochastic perceptions [CVZ00] for gathering environment information. Visual perception is simulated using some criteria such as being in line of sight and viewing angle, range, volume, moving state, and plant cover density. Similarly audio perception is also simulated stochastically using information such as range and being in line of sight.

In multi agent simulations, evaluating the environment information, learning and reacting in time is essential. Erol Gelenbe proposed modeling computer-generated forces with learning stochastic finite-state machines whose state transitions are controlled by state and signal dependent random neural networks [E98]. In Knuffner's approach [JJ99, JJ98], rendering off-screen from the

character's point of view and real-time path planning is used. His path-planning module aims to find a collision free path between a starting and ending point over the 3D terrain using the information gathered from vision based perceptions.

As we have gathered the environment information, we also need to plan valid path to reach the goal state. In the study of Knuffner [JJ99, JJ98], the terrain is divided into embedded graph cells, which have vertical, horizontal and diagonal costs of walking through. Then, the suitable path is found using Dijkstra's algorithm, which is actually an optimal off-line path-planning algorithm integrated into a real-time application.

The help of some guidance such as admissible heuristics can increase efficiency of these path-planning algorithms. A* [SP95] is one of best-known efficient path planning algorithms, which is guided by a heuristic function. A* always finds the optimal solution and uses linear distances between points for the heuristic function.

Although heuristic can lead to efficient algorithms, it is not enough and optimal path planning algorithms cannot be used for large and dynamic landscapes because of its complexity. To avoid this drawback, some partial path update algorithms are also proposed such as D* [A94] and focused D* [A95]. These algorithms plan an off-line path, let the agent follow the path, and if any new environment information is gathered, they partially re-plan the existing solution.

A number of algorithms exist for supporting real-time simulations such as Real-time A* (RTA*) [MY96], which is actually one of the best known ones. RTA* is usually used for maze environments and it uses a greedy search strategy and a heuristic together to guide the search. It guarantees to find a solution if one exists, but the solution may not be optimal. In this thesis, we also propose an improvement for Real-Time A* for the domain of large mountainous landscapes.

There are some path-planning algorithms that use random search techniques such as genetic algorithms, random tree generators. In [KJ97], an adaptive path-planning algorithm based on genetic approach is proposed. In the study, they assumed that a valid path that is not optimal is initially found and they refine this given path by genetic algorithm. Considering this concept, the proposed not

optimal off-line path-planning algorithm seems to be applicable to the study successfully. In the study of LaValla and Knuffner [SJ99], a randomized planning technique based on a version of random tree generation called rapidly exploring random tree is presented. They generated two random trees starting from the goal and the target points, and try to catch an intersection among the points of distinct trees to find a path.

In addition to path planning, many agent systems have a module called "Intelligent Reactive Planning" for the purpose of deciding and reacting efficiently to various events considering the goals, knowledge base and the previous experiences [JP94, RM98]. For example, reactive planning modules may be used with rule sets such as a goal-directed decision tree.

Group coordination is also an important concept in multi agent simulations. Without coordination, the agents can only be considered as individual groups with no relation. Baxter and Horn [JG99], organized a command and control hierarchy used by the agents, which is based upon the military command and control structure. In the hierarchy, the groups are under control of a squadron commander. In addition, groups have their own troop commanders. Organizing the groups along the same lines as the military formations allows emulating the change of getting plausible behaviors. It also provides a framework to guide the communication among the agents and allows the planning of complex group orders to be divided into several smaller problems. Communication of orders passes straight down the hierarchy and intelligent information is shared between peers and communicated to superiors. The commanders are responsible for gathering information about their own situation, passing it up to their superiors and peers, and giving orders to their subordinates to achieve the commander's high level objective. That structure shows similarity to the proposed group and command structure. We have organized the agents in groups with a commander to achieve a specified group mission. Commanders are responsible for executing mission, passing the state up to other group commanders by radio messages and guiding the group. The subordinates are responsible for following their group commanders and executing the commander's orders.

CHAPTER III

SIMULATION ARCHITECTURE

3.1. Introduction

A simulation environment is designed and implemented. The developed simulation architecture consists of two main parts: an environment and a command and control system. The architecture context diagram of the simulation is shown in Figure 3.1.

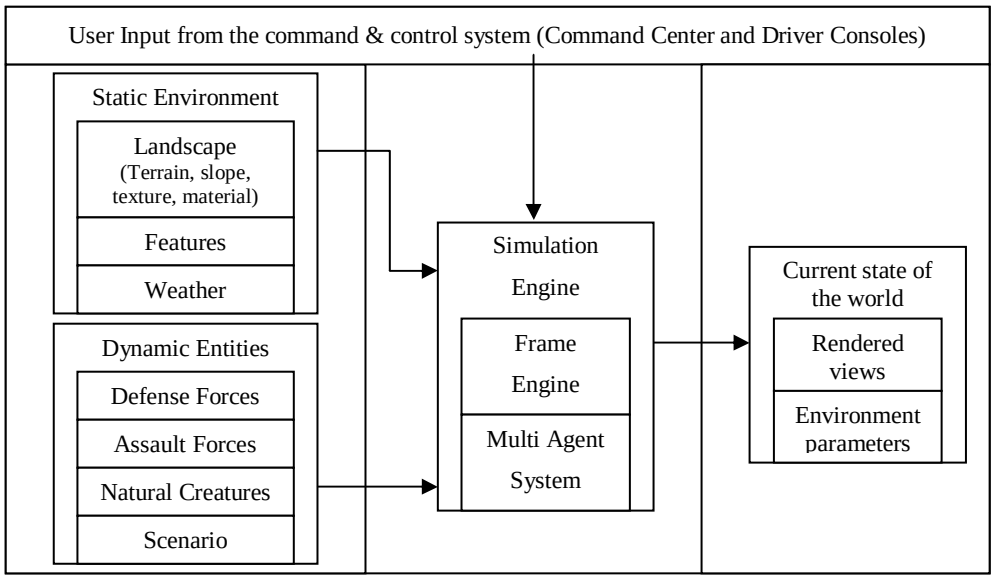

Figure 3.1: Architecture context diagram of the simulation

The static environment (landscape, features and weather condition) and the dynamic entities (the defense forces, the assault forces and the natural creatures) form the environment altogether. The structure of the environment is illustrated in Figure 3.2.

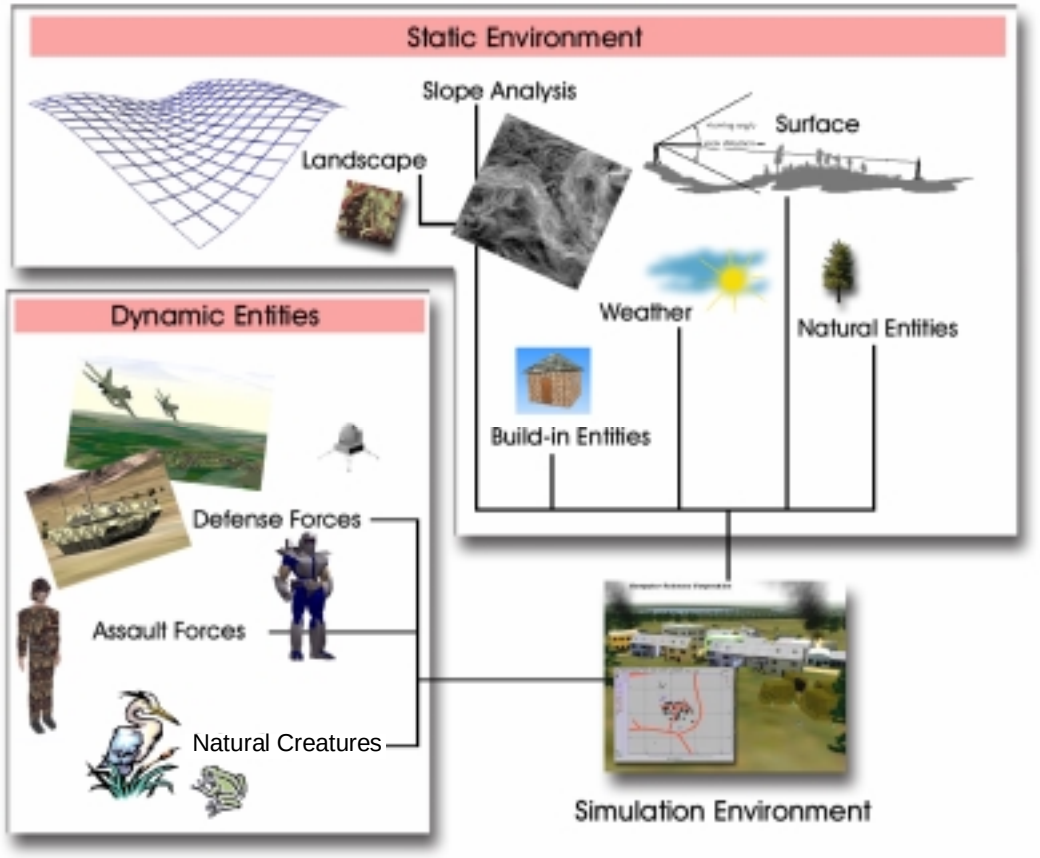

Figure 3.2: The static environment and the dynamic entities form the whole environment.

The command and control systems enable the defense forces (platform systems) to partially observe the environment by the help of their sensor systems. The

detection of a sensor may belong to a member of the friend forces, or a member of the assault forces, or just an animal. Figure 3.3 shows the display monitors of two different sensor types; DayTV (normal camera) and infra-red camera (heat sensitive camera).

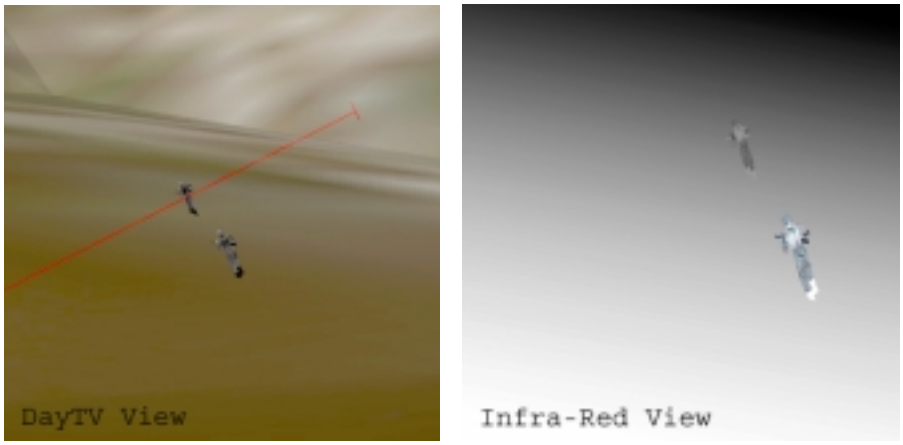

Figure 3.3: Views captured from a DayTV camera (left) and an infra-red camera (right). A DayTV views the visible light reflected from any surface, instead an infra-red camera shows the temperature differences of the surfaces.

The users control the defense forces, and the command and control systems help them to locate, view and control their platform/sensor systems on the terrain. Their objective may be to defend a tactical area from any assault forces.

The assault forces are our intelligent human agents. Their objective is to complete their high-level mission without being detected by any sensor systems of the opposing forces. The mission may be to attack, escape or just pass through the selected tactical area.

Animals have simple automated behaviors and they are used for misleading the platforms and agents' perceptions.

3.2. Static Environment

3.2.1. Landscape

3.2.1.1. Terrain: Polygons vs. Voxels

Terrain, the landscape surface, is the fundamental part of a simulation environment and how to model a terrain is one of the most important decisions to be made, as it affects efficiency, accuracy and computational cost of the simulation. There are two common methods to be used: the polygonal base modeling, and voxel base modeling. A polygonal-based and a voxel-based landscape are illustrated in Figure 3.4.

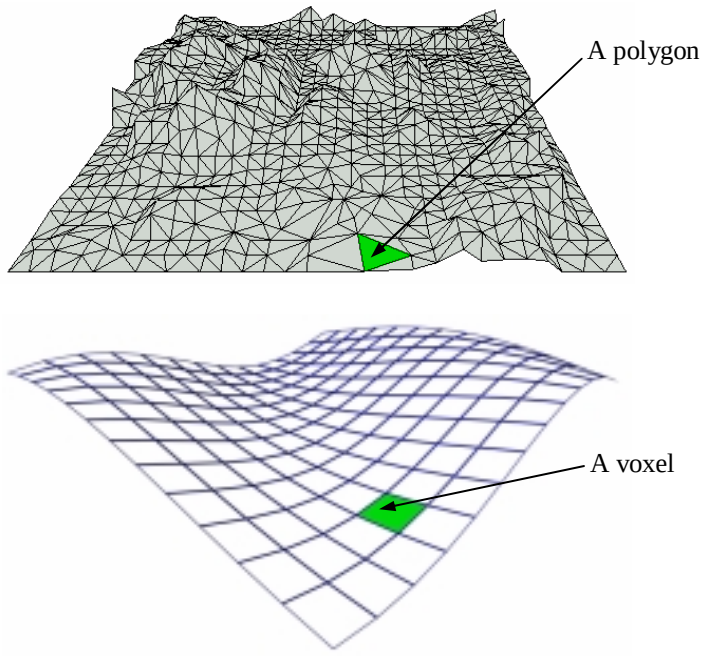

Figure 3.4: A polygonal based terrain (top) and a voxel-based terrain (bottom) generated from a matrix of elevations

Polygons are frequently used in 3D modeling of virtual environments such as in computer simulations, computer games and 3D modeling tools. There are powerful hardware and software support for displaying polygonal objects, and they are simple to model and easy to use in any 3D visualization system. That is why they are commonly used in 3D world. But using polygons has also some disadvantages. For some special purposes they may be incapable or sometimes they may be too complicated. For example, In the filed of health, voxel based rendering techniques are most commonly preferred because of its accuracy and capability of storing and representing the volume data. In the field, they do need the data of a volume such as the inside of a body. Polygonal representations can only handle a surface modeling; in contrast voxels can handle a volume of an object. The critical point is that the advantage brings also a drawback. Using volume data reduces the efficiency sharply and the hardware support for the technique rarely found such as ONYX-II.

When we consider using voxels for landscape, the condition changes in some ways. The data representation becomes a matrix of sampled heights (elevations), on earth, which is called "digital terrain elevation data" (DTED). The samples are taken equally distanced in x and y directions and a matrix is generated. Thus, this time the data is not a 3D volume, because only the information of surface elevations exists. So the computational cost decreases significantly and the technique becomes an alternative to polygonal representations in some cases such as in the game called “Delta Force II”. Triangle calculations are hard to perform on large terrains, such as mountains, generated from millions of polygons. Using elevation matrix, computations can be more accurate and efficient because of the homogenous distribution of the voxels. It can be used in the background computations, but the similar problem of volume rendering also appears for this kind of representation. For the polygons, we have hardware accelerators to calculate triangle intersection tests efficiently, but the voxel rendering techniques suffer from the lack of hardware support.

To be more efficient, there are several optimization techniques for polygonal terrains. But, on a mountainous area, optimization cannot be performed well and

accuracy can be lost in many cases. A third alternative is to use polygons and voxels all together. The terrain data is loaded as a matrix of elevations. The background computations are all performed on the data, but the rendering is done using polygons. Thus, the polygons are extracted from the matrix of elevations, but left hardware to be drawn by introducing some additional culling phases, which increases efficiency. Figure 3.5 shows an image rendered on a high-resolution terrain using a version of the mentioned technique.

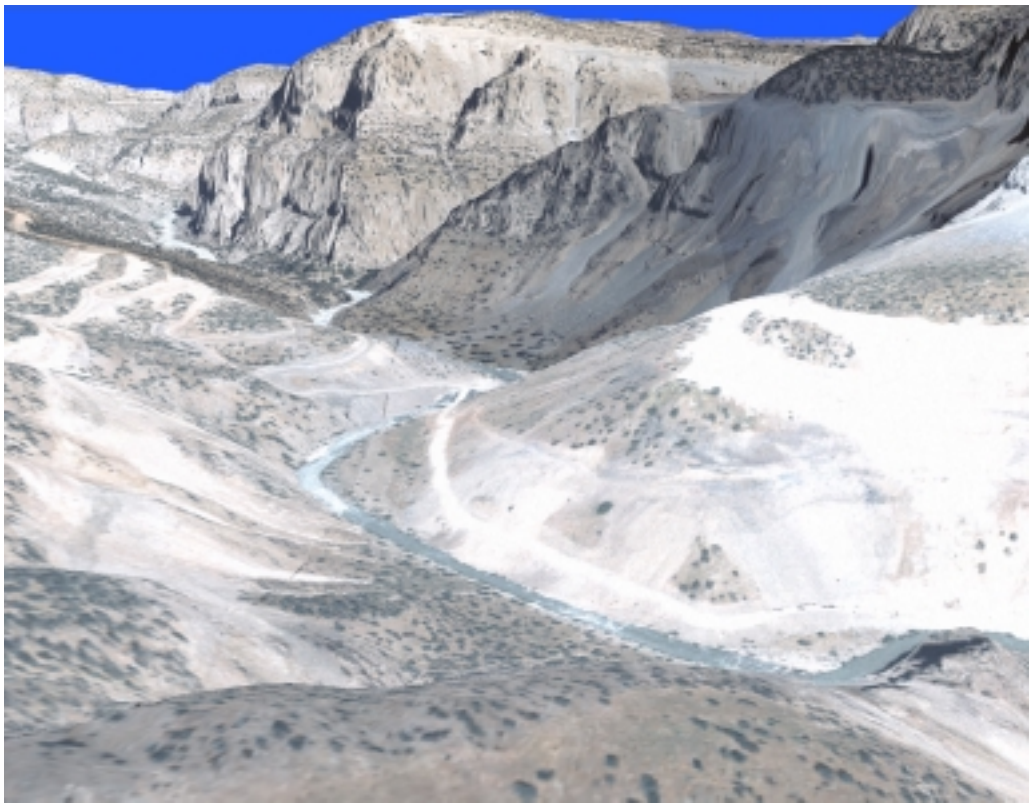

Figure 3.5: An image rendered using both digital elevation data and polygons. The rendering operation is performed using polygonal rendering of elevation matrix.

3.2.1.2. Loading DEM Files

A DEM (Digital Elevation Model) [USGS98] file is a matrix of elevations sampled on a limited region of the earth. DEM files are used in the model because of its data representation capacity. An alternative file structure could have been the DTED (Digital Terrain Elevation Data) [NIMA99] file format, but it cannot model a terrain as accurate as a DEM file, because a DTED file can store at most 1-second frequency of elevations, which is 30 meters on the average. The general file structure of DEM is as follows:

DEM is saved as an ASCII file and does not contain a specific file signature. The matrix of elevations is stored as profiles, which are one or more columns of elevation samples starting from south west. The Table 3.1 and 3.2 is the C-like pseudo code for loading a DEM file.

Table 3.1: Type definition of RecordA and RecordB

```
struct RecordA {
  int    DEM_Level;           // 3
  int    Pattern_Code;        // 1
  int    Plan_Ref_Sys_Code;   // 1
  int    Zone_Code;           // 0
  float  Map_Projections[15]; // should be ignored for geographic systems
  int    Units_Code;          // the units stored; UTM(meter)/GEOGRAPHIC(degree or second)
  int    Units_Code2;         // 1; elevation unit 1 indicating meter
  int    Num_of_Sides;        // 4; number of corners; assumed to have 4 corners
  float  Four_Corners[4][2];  // four corner coordinates of the map
  float  Min_Elev;            // minimum elevation value on the map
  float  Max_Elev;            // maximum elevation value on the map
  float  CCW_Angle;           // 0; counterclockwise angle of dem
  int    Accuracy_Code;       // 0; indicates no accuracy
  float  Spatial_x;           // resolution of each sample on x axis
  float  Spatial_y;           // resolution of each sample on y axis
  float  Spatial_z;           // resolution of each sample on z axis; elevation
  int    Profile_Dimension;   // 1; number of columns in a single profile
  int    Num_of_Profiles;     // number of profiles; profiles are located from west to east
};
```

```
struct RecordB {
  int    Row;                 // the profile rows number
  int    Col;                 // the profile column number
  int    Num_of_Rows;         // number of rows in a profile
  int    Num_of_Cols;         // number of columns in a profile
  float  First_ElevC1;        // x coordinate of first elevation in the profile
  float  First_ElevC2;        // y coordinate of first elevation in the profile
  float  Elev_of_Local_Datum; // 0; elevation local datum for the profile
  float  Min_Elev;            // minimum elevation value in the profile
  float  Max_Elev;            // maximum elevation value in the profile
};
```

Table 3.2: Loading a DEM File

```
Open the input DEM file for reading.

Read 145 characters of string file description.

Read RecordA, which usually contains 39 header items (If we assume that the area is
rectangular and there are four corners). The Record A is shown in Table 3.1.

if (Units_Code == 2)              // Unit Type is in Metric Coordinates as meters
    unit_type_of_dem = utMetric;
else
if (Units_Code == 3)              // Unit Type is in Geographic Coordinates as seconds
    unit_type_of_dem = utGeographic_sec;
else
if (Units_Code == 4)              // Unit Type is in Geographic Coordinates as degree
    unit_type_of_dem = utGeographic_deg;

Calculate area properties.

Read a set of RecordBs, which contains profiles. The Record B is shown in Table 3.1.

for ( int pf = 0; pf< Num_of_Profiles ; pf++ )
    {
      Reading Record B, which is single profile information.

      for ( cx = 0; cx< Num_of_Cols; cx++ ) // Number of columns in the profile
          {
            for ( cy = 0; cy< Num_of_Rows; cy++ ) // Number of rows in the profile
                  read one elevation item
          };
    };

Close the input DEM file.
```

3.2.1.3. Generating Slope Matrix

Loading the digital elevation data file, we analyze the elevations to find out the slope properties of each voxel.

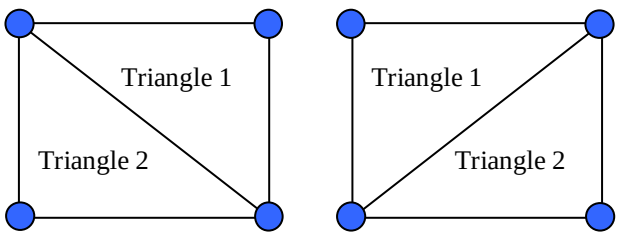

Figure 3.6: Four elevations let two different triangulation styles.

For each voxel, there is a single elevation value, which is measured form the sea level in meters. Every group of 4 corners forms a surface. The surface properties are not known exactly, because we only know four elevation points, but the inner part is uncertain. We can triangulate the points by connecting four corners in two ways, which results in two triangles. The connection alternatives of four points are shown in Figure 3.6. The problem is that we cannot known which way is the right one because the connection method is unknown. So, any of them can be selected.

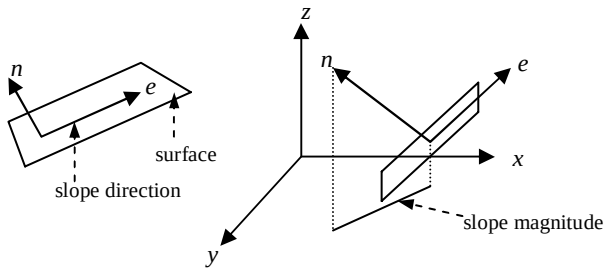

Figure 3.7: Calculating slope directions and magnitudes

For simplicity, it is better to have a matrix of slopes corresponding to each voxel data of elevation matrix. The slope matrix will have the same dimension as the elevation matrix. So we have to generate a single slope value for each group of four-points. Each of the alternative triangles has its own surface normal; we get the average of two surface normals. But, a single row and column from the upper and right of the matrix remain unfilled because of the lack of elevation data. Computations are illustrated in Figure 3.7 and a sample result is shown in Figure 3.8.

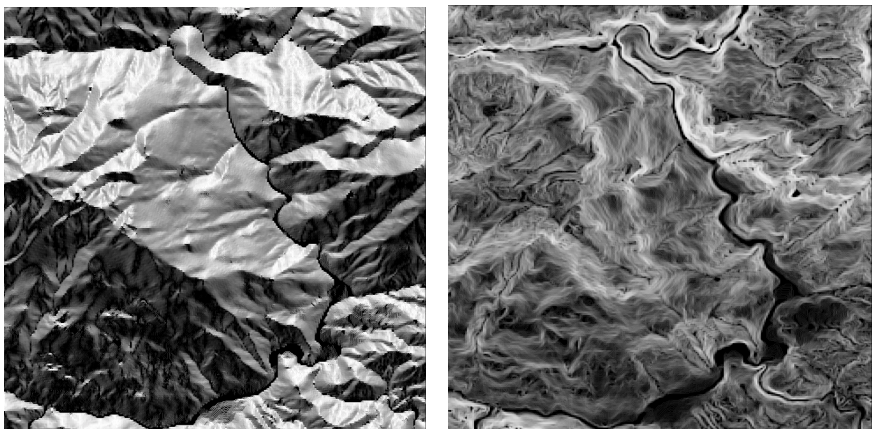

Figure 3.8: Visualization of slope directions and magnitudes of a terrain

3.2.1.4. Mapping Texture

Raster or satellite images are mapped on the terrain in order to let us recognize the area or realistically display the environment. A raster image is an artificial map and a satellite or other air vehicles images are orthographically taken photographs of the landscape. A texture matrix, which is the same size as the terrain, is allocated for texture mapping. Different regions can be loaded from different texture files and merged on that matrix to shape the whole area. For that purpose, windows' bitmap file format is supported by the system. When a texture is loaded, it is mapped by using two diagonal corner coordinates, which is sampled in Figure 3.9. The coordinates are given as geographic coordinates in degrees or seconds. Then they are converted into row and column numbers of the texture matrix and the image is scaled onto the matrix using the row and column coordinates of the corners.

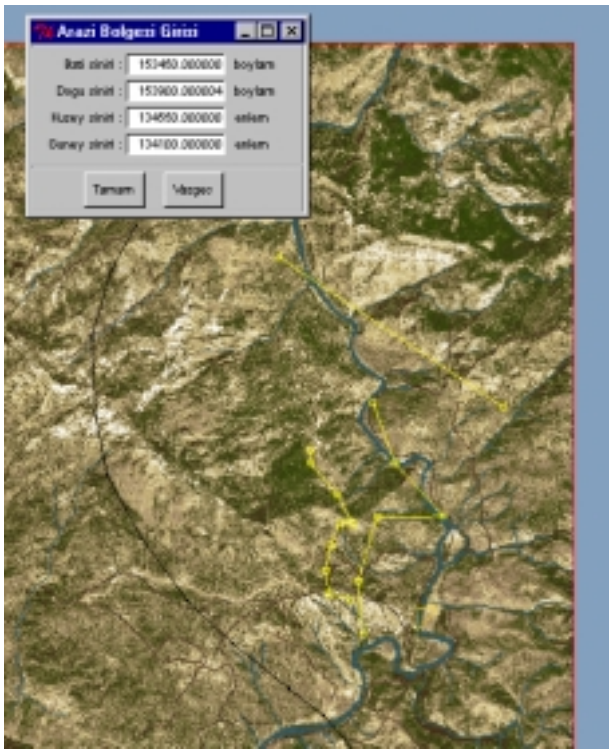

Figure 3.9: The texture is mapped on the terrain using two diagonal corners, which are given as geographic coordinates.

3.2.1.5. Surface Materials

Surface materials are the features of landscape surface. In the system, surface materials are classified into 12 main categories. These are as follows:

- Soil,
- Rock,
- Forest,
- Pasture,
- Shrubbery,
- Agricultural area,

- Road,
- Urban area,
- Dry snow,
- Wet snow,
- Salty water,
- Fresh water

The above list is the main categories, but they are not directly used. Instead they are selected to make a list of materials to be used on the surface. Changing the parameters (such as average height of the trees, salt level) generates variations from the basic material types.

Semi-automated tools are used in order to guide the user while defining region properties. First, the user can select a region and assign a material. This is the manual operation. A second option is to select regions by color similarities on the texture. For example the user can select a blue color from a river, and give a color tolerance to define how similar colors are accepted to be the same material. Thus, an automated image analysis is performed on the whole texture and similar colored regions are marked as the same material.

The tolerance is an integer value between 0 and 255. To assign the selected material to a voxel the following condition must be satisfied:

```
If ( abs(R-SelectedColorR) <= Tolerance) and
( abs(G-SelectedColorG) <= Tolerance) and
( abs(B-SelectedColorB) <= Tolerance) and
( abs((R-SelectedColorR)-(G-SelectedColorG)) <= Tolerance)) and
( abs((R-SelectedColorR)-(B-SelectedColorB)) <= Tolerance)) and
( abs((G-SelectedColorG)-(B-SelectedColorB)) <= Tolerance))
then assign the material.
```

It is our experience that, the best similarity detection is obtained with the criteria described above. The first three lines helps a lot to find a color near to the selected one which is given as red, green and blue components, but it is not enough alone because the change can be in positive or negative direction. If red

changes in +tolerance and green changes in –tolerance, the distance between red and green will be 2 times of the tolerance. So, the next three lines are also needed. The result of the applied operation is shown in Figure 3.10.

Figure 3.10: Rivers are extracted (right) from the texture (left) using color similarity. The white region on the right image is the extracted river.

Applying similarity criteria to the whole area is one approach. Depending on the state and region distributions, different methods are needed. So, the tools can be extended in many ways such as selecting similar neighbor voxels or selecting zero-sloped areas such as lakes, rivers, etc.

3.2.2. Features

Features are the natural or build-in static objects located on the terrain such as lakes, trees, rocks, roads, houses, bridges, etc.

There are three different feature types according to their location: point located, path located and area located features. A point located feature is positioned on the terrain by its x, y coordinate and face direction (e.g., houses, trees, bridges). A path located feature is defined as a list of x and y coordinates forming a path such as roads, rivers, etc. By using area located features, you can define areas such as lakes and seas. 3D models of some sample features can be seen in Figure 3.11.

Figure 3.11: Static features: a 3D hut (left) and tree (middle), specification of road segments and trees (right).

3.2.3. Weather Conditions

Weather condition is a very important environmental effect. It may even cause a sensor or a platform completely unusable. For example, an UAV (Unmanned Air Vehicle) cannot fly on a windy day or above a limited altitude. A normal camera (DayTV) can not see at night because of low light, an infra-red camera can not

work fine on hot days because of its sensitivity to temperature, or a Doppler radar can not be used on windy days because of too much movement which increases false alarm rate. Also fog, rain, snow, humidity, etc. reduce the maximum seeing distance of many optical sensors.

Different weather conditions on a large landscape can be defined. A matrix of indexes similar to the landscape elevation matrix is stored to assign different weather conditions to different regions. Each member of the matrix holds an index pointing to a member of weather conditions list. The matching of each voxel to a weather condition is obtained by a simple linear distance check. A voxel gets the condition of the nearest weather condition object in the list. The result of a sample assignment is illustrated in Figure 3.12.

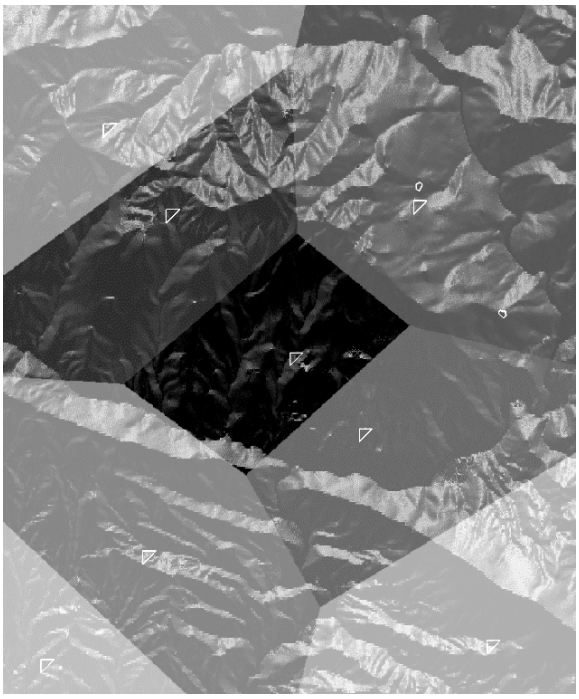

Figure 3.12: Regions having various weather conditions are colored in different colors. Triangles shapes are weather condition objects.

Each weather object stores a set of weather parameters defining the current state. These parameters are as follows:

- Day time temperature,
- Night time temperature,
- The type of rain,
    - Rain,
    - Snow,
    - Hail,
- Level of rain,
- Density of fog,
- Density of humidity,
- Density of cloud,
- Direction of wind,
- Strength of wind.

The value of parameters except the temperature and direction of wind are discritized as *none*, *very low*, *low*, *mean*, *high*, and *very high*. The reason of not using real values and units is the difficulty of finding them.

3.3. Dynamic Entities

Dynamic entities on the terrain are grouped into three categories: the blue group, the red group and the white group. The group structure is shown in Figure 3.13.

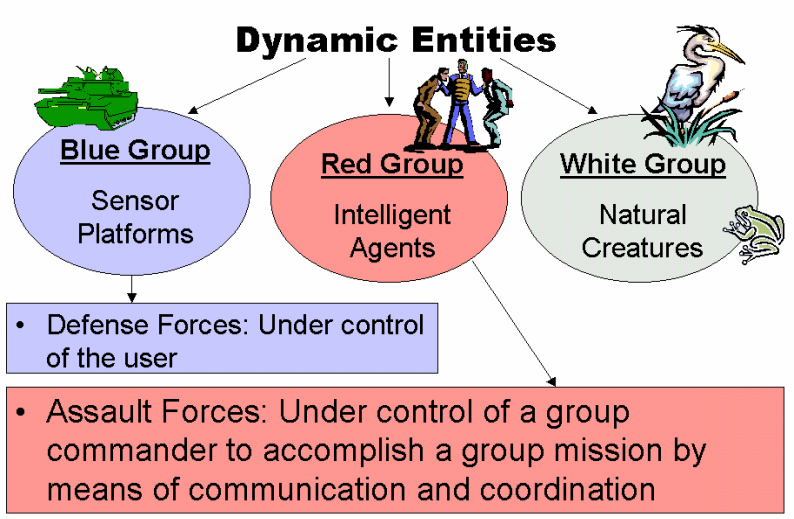

Figure 3.13: The group structure of the dynamic entities

The blue group is the defense forces that have platforms carrying different kinds of sensor systems such as DayTV, infra-red camera, etc. They are controlled by the users. The control can be achieved in two different ways: *the remote control* and *the driver control*. The remote control means watching and controlling the sensors from a far central system (command center), and the driver control means watching and controlling the sensors as the driver of platform carrying the sensor itself. The objective of the blue group is to defense a tactical area from any assault forces.

The red groups are the assault forces that are intelligent human agents. They have a semi-automated behavior system. They are organized in one or more groups each having a commander. Their objective is to complete a given mission without being detected or caught by any defense forces. The mission may be to attack, escape or just pass through the selected tactical area.

The white group is the animals that are natural creatures such as birds, bears, horses, wolves, etc. They have a simple automated behavior system making them

move randomly on the terrain. The idea behind introducing them to the environment is to make the platforms and agents' perceptions go wrong.

3.4. Command & Control Systems

3.4.1. Test Environment

In order to simulate defense and assault forces, a real-time simulation system is generated using the previously described environment. The simulation system enables the users to defend the selected area by monitoring and controlling the sensors and platforms from the field or tactical command center. It also supplies the environment with our intelligent agents.

The tactical command center has the capability of monitoring the views of one or more sensors at the same time, and controlling them remotely. It can also assign a driver console to interactively control a platform in the field. There are two driver consoles; so two platforms can be controlled at the same time.

3.4.2. Command Center

The command center has four sensor monitors. The user can use any of the monitors to view a selected sensor of a platform. Each sensor assigned to a monitor can be controlled remotely. For example, the looking direction or the zoom level of a sensor can be changed at any time. The command center console is shown in Figure 3.14.

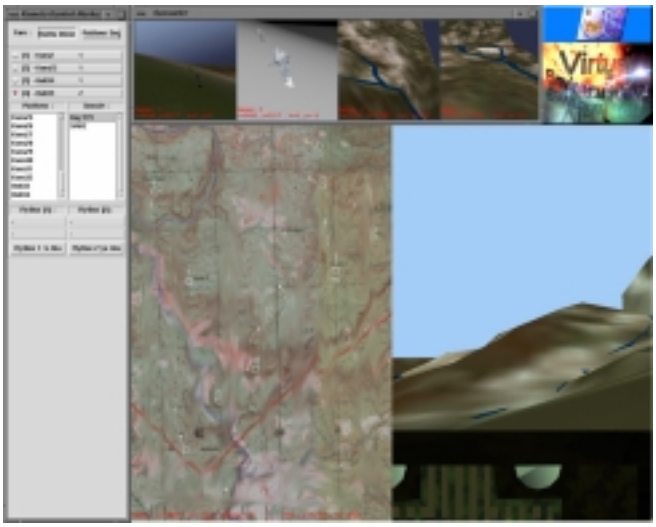

Figure 3.14: A snapshot from command center

The current coordinates of the platforms can be seen from the command center. An orthographic raster map showing all the platforms on the area gives this capability. The map can be moved or zoomed, or the 3D model of any region can be viewed by just using mouse.

An automatic alarm generation support is also added to the system. When a new object is detected, the laser distance finder can find the estimated coordinate of the object. It is also possible to send any platforms to the area to recognize and identify any detected object. The driver consoles are used for that purpose.

3.4.3. Driver Consoles

The driver consoles are used to control movement of platforms in the field. There are consoles that can be used at the same time. The controlled platform can either be a land or an air platform. The driver console has three display channels. These are the view of the driver eyes, the view of one of the sensors at the platform, and

the orthographic map centering the current coordinate of the platform under control. The Figure 3.15 is a snapshot form the driver console.

Figure 3.15: A snapshot from a driver console

CHAPTER IV

DISPLAYING ENVIRONMENT INFORMATION

4.1. Visualization

The user needs to see and edit the information gathered. So, a visualization method is essential in order to view elevation, slope, surface, texture, and weather condition data.

In the system, there are two main displaying types according to their projection, which are the orthographic (top view with no perspective) and the perspective projection. The orthographic displaying of the landscape is indeed a top view of the area. You can move, zoom-in or zoom-out to any region loaded using mouse. The 3D displaying (rendering) is used to view the area in a camera manner. It allows you to see or investigate a specific region in detail, or just to travel in the area such as being in a helicopter.

4.2. Orthographic Viewing

4.2.1. Displaying Elevation Data

The elevation data is a matrix of heights sampled equally distanced on the terrain. The values are elevations from the sea level in meters. We can find the minimum and maximum elevation on a loaded terrain to limit the values, so that we can colorize the map. The color of a voxel is calculated using a linear interpolation

between the dark green (minimum elevation) and light green (maximum elevation). The result is shown in Figure 4.1.

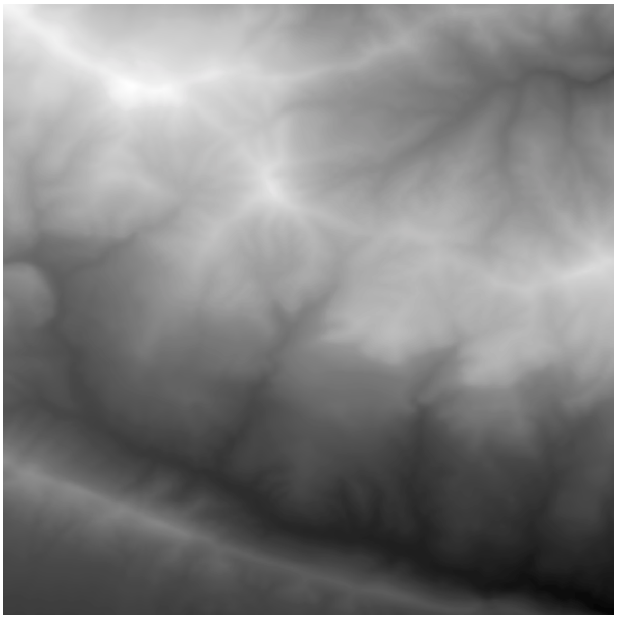

Figure 4.1: Visualization of elevation matrix

4.2.2. Displaying Slope Magnitudes and Directions

When the elevation matrix is loaded, slope magnitudes and directions are calculated automatically, which is described in Section 3.2.1.3. The slope magnitudes are values between 0 and 100, so the color of a voxel is calculated using a linear interpolation between the dark green (0) and light green (100). A sample slope magnitude map is shown in Figure 4.2.

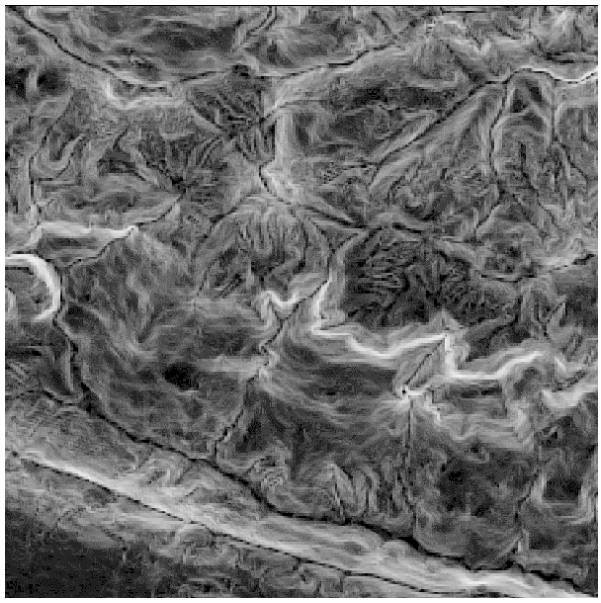

Figure 4.2: Visualization of slope magnitude matrix

The slope directions are values between 0 and 360, but we cannot use a simple linear interpolation this time because the values near to 0 and 360 are similar directions and must be colorized with similar colors. So the value-color matching is white for 0 degree and black for 180 degree. The color turns to white again as the degree increases to 360. The result of colorizing the directions is illustrated in Figure 4.3.

Figure 4.3: Visualization of slope direction matrix

4.2.3. Displaying Texture

Texture is the easiest information to display because of the data format used. The matrix of texture stores the red, green and blue values for each voxel. So no conversion is needed at all. A sample satellite texture is shown in Figure 4.4.

Figure 4.4: Visualization of texture matrix

4.2.4. Displaying Mixed Information

In some cases, we may need to see a mixture of the environment information such as elevation and slope direction values in a mixed form. In this case, two colors must be mixed in a weighted manner called alpha blending. For example, when we want to merge the elevation matrix to slope direction matrix using 70% transparency, the color of each image pixel is calculated using the following formula:

$$MergedColor_{Red} = (0.7)\ ElevationColor_{red} + (0.3)\ DirectionColor_{red}$$
$$MergedColor_{Green} = (0.7)\ ElevationColor_{Green} + (0.3)\ DirectionColor_{Green}$$
$$MergedColor_{Blue} = (0.7)\ ElevationColor_{Blue} + (0.3)\ DirectionColor_{Blue}$$

So the general formula of alpha blending an image A over an image B for an transparency value, α (between 0 and 1) is:

$$BlendedColor_{Red} = (\alpha)\, A_{red} + (1-\alpha)\, B_{red}$$
$$BlendedColor_{Green} = (\alpha)\, A_{Green} + (1-\alpha)\, B_{Green}$$
$$BlendedColor_{Blue} = (\alpha)\, A_{Blue} + (1-\alpha)\, B_{Blue}$$

This technique is used for displaying elevation-slope matrix, surface materials and weather conditions. The blending of elevation and slope direction matrix is shown in Figure 4.5.

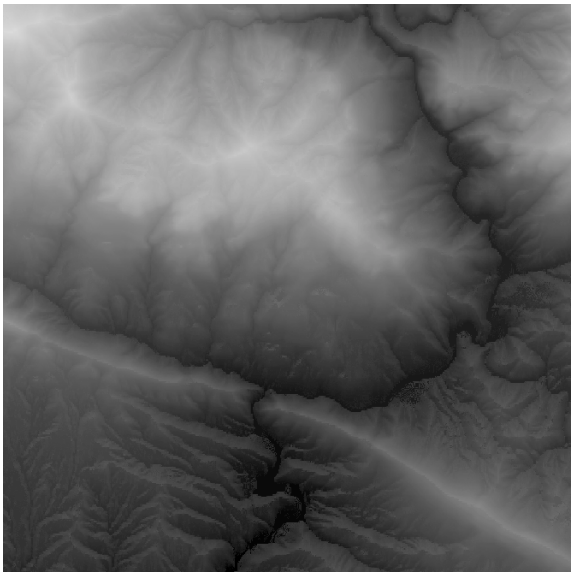

Figure 4.5: Visualization of elevation and slope direction matrix using alpha blending

4.2.5. Displaying Surface Materials

We have 12 different surface material types that are described in Section 3.2.1.5. For each type, a specific color is assigned and the information is displayed by merging the result to the slope direction matrix using alpha blending technique. A sample snapshot is shown in Figure 4.6.

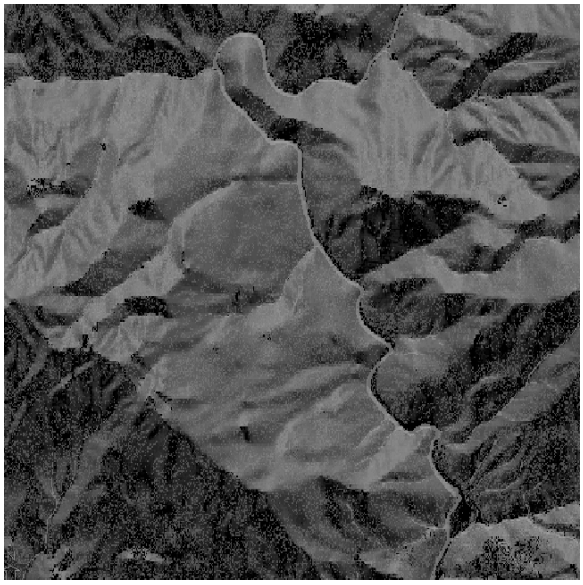

Figure 4.6: Visualization of surface materials blended on slope direction matrix

4.2.6. Displaying Weather Conditions

The weather condition of an area is determined by the weather objects added to the environment. Each voxel takes the properties of the nearest weather object on the area. So, the visualization of the weather condition depends on the number of weather objects added to the scene, which changes from 1 to 255. If we use similar color values for different weather objects (such as 1, 2, and so on), it may

be impossible to see the color difference. In practice there are at most several weather objects in an environment. So it is better to assign color values dynamically using the number of objects in the environment, covering the whole color spectrum. For example, when we have 3 weather objects in the environment, the color of first object will be 0, the second object will be 127, and the third one will be 255, which is the upper bound of the color. The result of coloring the weather distribution of 6 weather objects is shown in Figure 4.7.

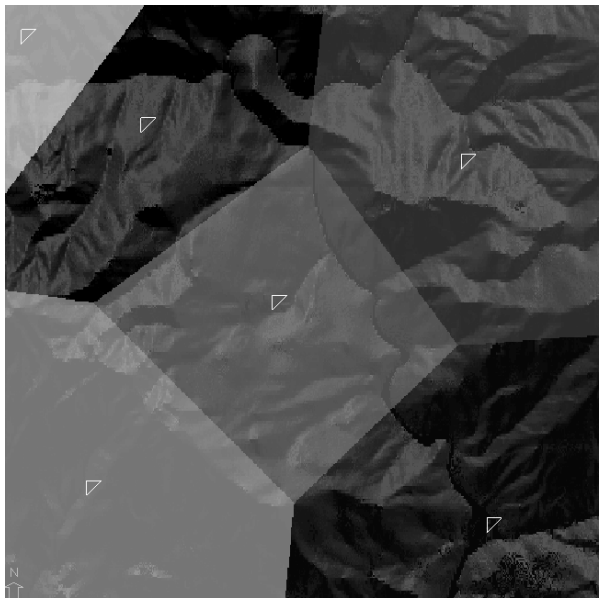

Figure 4.7: Visualization of weather condition distribution blended on slope direction matrix

4.3. 3D Rendering

In order to visualize the environment in details, 3D rendering by perspective projection is also supported in the system developed. The common techniques for

rendering a landscape are polygonal and voxel based rendering, which are discussed in Section 3.2.1.1. In this study, both techniques are implemented, but the polygonal based rendering is mainly used because of the efficiency increase by the help of hardware support. But, voxel based rendering enables some features that cannot be managed by using just polygons. For example, the plant cover density of an area can be rendered using the technique. The information is the average plant height and density of the area, and the output is the probability of seeing for each voxel and the air above. The result of rendering is shown in 4.8.

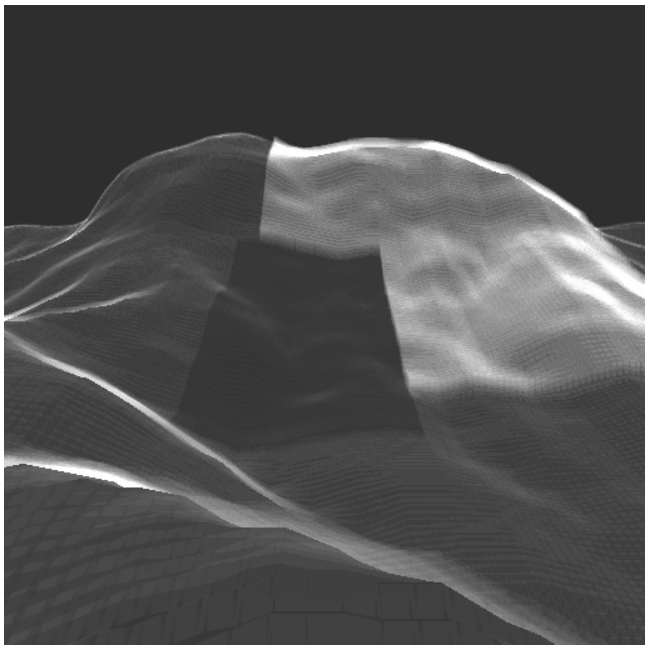

Figure 4.8: 3D Visualization of the plant-cover density. On the image, the plant cover density of the dark colored regions is low, and density of the light colored regions is high.

If we don’t need the additional capability of voxel based rendering techniques, the polygonal rendering is commonly preferred because of hardware support. An area rendered by both polygons and voxels using different textures is shown in Figure 4.9.

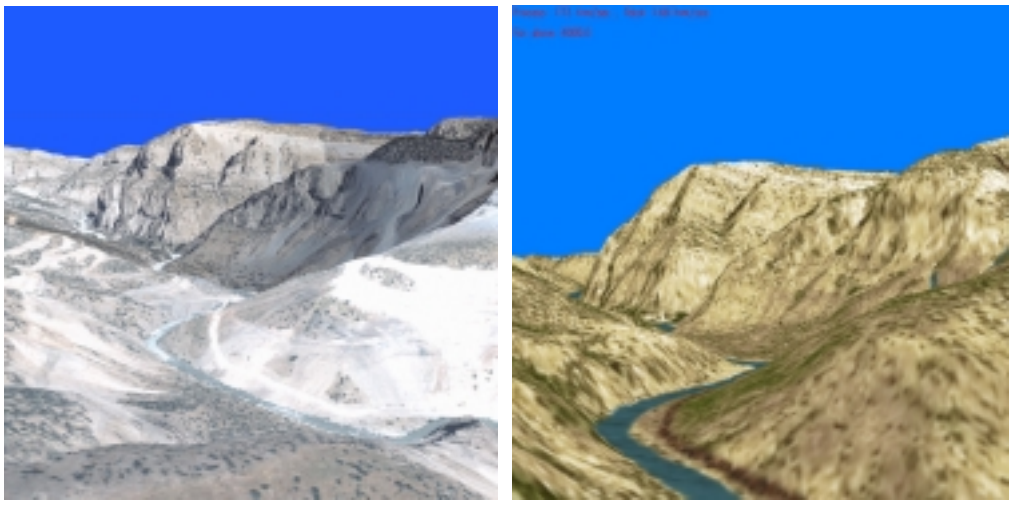

Figure 4.9: Polygonal and voxel rendering: The image on the left is rendered using polygons and the one on the right is rendered using voxels. Note that the textures, the camera positions, and the resolutions are not the same.

CHAPTER V

AGENTS

5.1. Introduction

This thesis involves modeling and representing actions for virtual human agents that should accomplish a given tactical mission in a virtual battlefield, which is a part of a sensor simulation system. The objective of developed software is to test a sensor optimization algorithm using scenarios that are executed by the intelligent human agents. The main goal of the agents is to accomplish the given mission without being detected or caught by a sensor platform.

5.2. Goal Description (Mission)

In the proposed approach, the goal belongs to the group, not the individuals. Every group has its own goal plan (group mission) and moves with its commander's orders. The group commander gives decisions to accomplish the mission and subordinates follow their commander under normal conditions. All the group missions form the higher-level mission of the assault forces altogether.

Goal description is given as a set of tactical control points and a list of actions to be achieved at each point. Path control points are categorized into five group, which are *starting point*, *target point*, *home point*, *pass through points* and *tactical points*. To reach these control points, commander generates a path

considering the terrain and currently detected threat information. Then, the commander follows that path. A screen snapshot of a sample scenario is shown in Figure 5.1.

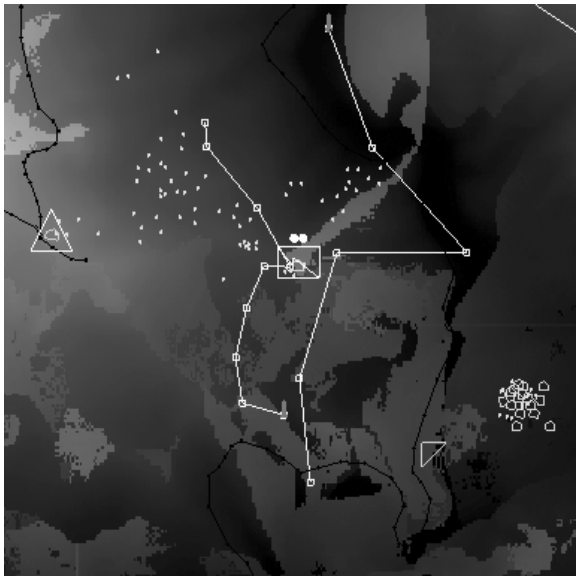

Figure 5.1: Scenario control points for two agent groups. Before planning a real path, the connections between control points are just straight lines.

The objective of a group is to pass through each given tactical point, but all the points do not have the same priority. For example, the group must pass through *the target point*. However it is not obligatory for a group to go through the points whose types are *pass through*.

The group can accomplish some tasks/actions at control points. Actions define the activities to be done such as wait 5 minutes, put bomb, etc. Synchronization among groups is also handled using a special action type called *the radio messages*.

5.3. Perceptions

5.3.1. How Perceptions Work

Agents gather information from the environment by seeing and hearing based on probabilistic computations. Actually, the sensor perceptions do not work stochastically, but the notification of a sensor perception by an agent is based on probability. That means; the sensors percept every thing around, but the agent may not notice it depending on the probability of detection. The sensor perceptions are sent to agent even the probability is very low. Then the agent checks the probability. If the perception is owned by an unknown entity, which means the attention is low; a probability test is done before accepting it. If the agent decides to notice the entity, then the entity is always seen or heard without calculating the probability until it is away from the agent's point of view. That can be described by the following example. A person always looks around. Eyes capture everything that is possible to be seen, but human may not notice them because of his low attention. Once an entity is noticed, seeing and following it becomes continuous by the help of high attention. Hearing can also be considered similarly. The difference is that hearing doesn't depend on being in viewing angle.

5.3.2. Seeing

The detection probability of image gathering perceptions, seeing, is related to the optical capabilities. Therefore, the probability of detection by seeing depends on the following parameters:

- Being in line of sight
- Being in viewing angle
- Volume of the target
- Range to the target
- Movement of the target

- Plant cover
- Weather condition

The being in viewing frustum and in line of sight means; the agent looks in the direction of the target and there is no other object between them to prevent seeing each other. The volume of target also affects the probability, because if an object is big, it is easy to see, but if small, then it is hard to see. The increase in the range between the agent and the target reduces the probability, and the movement of target increases the probability. Plant cover is also an important property affecting the probability. For example, if the target or the agent is in a forest, the probability of detection decreases sharply. Bad weather condition such as fog, rain, snow also reduces the probability.

5.3.3. Being in Line of Sight

If a location p1 is visible from another location p2, we say that p1 is in line of sight (LOS) of p2, and the line of sight test of two coordinates; p1 and p2 results in true. To do the test, we check the existence of any other objects intersecting the line connecting p1 and p2. If so, that means p1 is not visible from p2, and the line of sight test returns false.

In the system, LOS is only tested using landscape, but not the objects on the terrain. LOS test for landscape is illustrated in Figure 5.2 and 5.3.

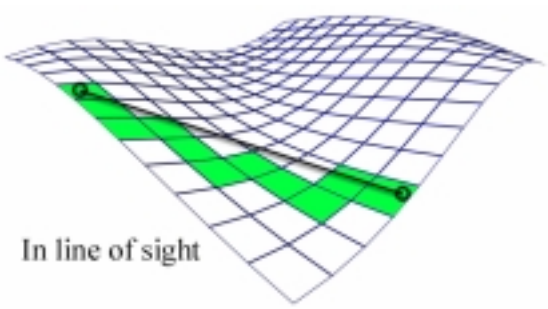

Figure 5.2: Two points are in line of sight of each other.

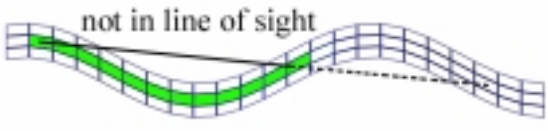

Figure 5.3: Two points are not in line of sight of each other.

The voxel base line of sight tests are much more efficient than polygonal intersection tests. In polygonal models, we don't have a systematic polygon distribution. For that reason many polygonal grouping techniques such as *oc-trees*, are used to increase efficiency, but voxels are already in that systematic format. Thus, intersection test are very efficient. Our implemented line of sight algorithm uses an incremental technique that works on voxels, which is commonly used in ray tracing for increasing efficiency of polygonal intersection tests. The algorithm finds the boundary intersection points of the voxels and the ray sent, which is illustrated in Figure 5.4. For each intersection point, the height of the ray and the landscape elevation at the point is compared. If ray is under the terrain in any point, it means; a hit is found and the two points are not in line of sight of each other.

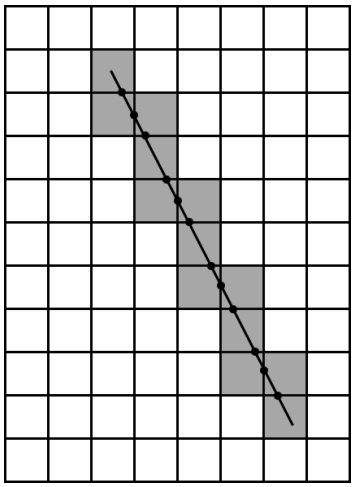

Figure 5.4: Line of sight test: The boundary intersection points of the voxels and the line are founded.

The line of sight algorithm or similar algorithms are applicable to many problems. For example, the algorithm is used for the computation of sensor coverage, agents' visibility tests, radio communication availability tests, direct path availability tests between two coordinates, voxel rendering, ray tracing, etc.

5.3.4. Being in Viewing Frustum

To see a target, being in line of sight is not enough alone, but also the agent must be looking nearly at the direction of the target. The eyes of an agent have a vertical and horizontal angle limit that bounds the seeing capabilities at a given time. This limited volume depending on the viewing direction, and vertical and horizontal angle limits is called viewing frustum which is illustrated in Figure 5.5.

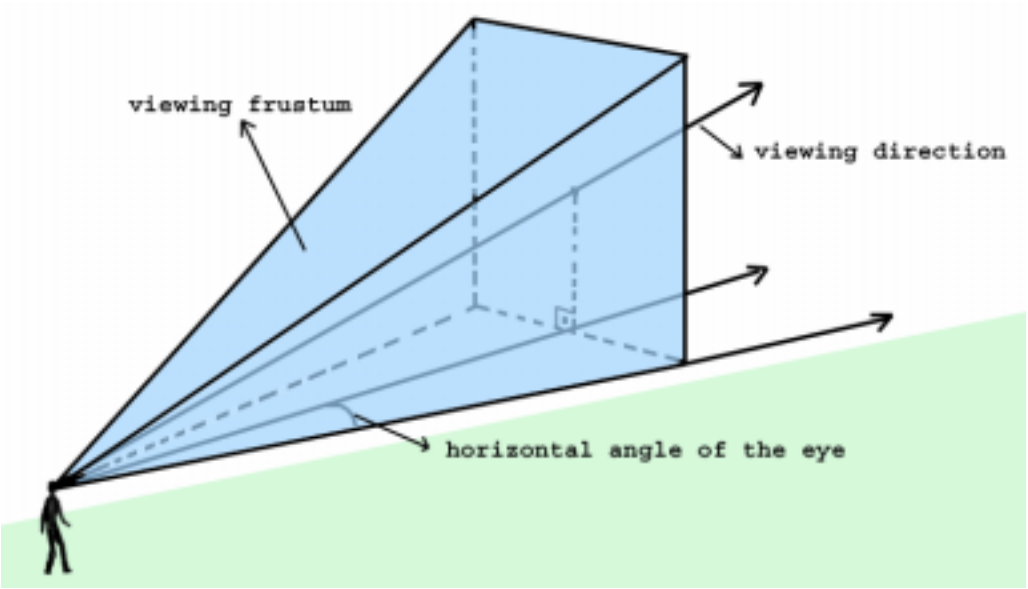

Figure 5.5: Viewing frustum of an agent

If the angle difference between the viewing direction and the direction of the line passing through the agent and the target coordinates is less than the angle limits of the eyes, the target is in viewing frustum and it can be seen by the agent. That is illustrated from top view in Figure 5.6.

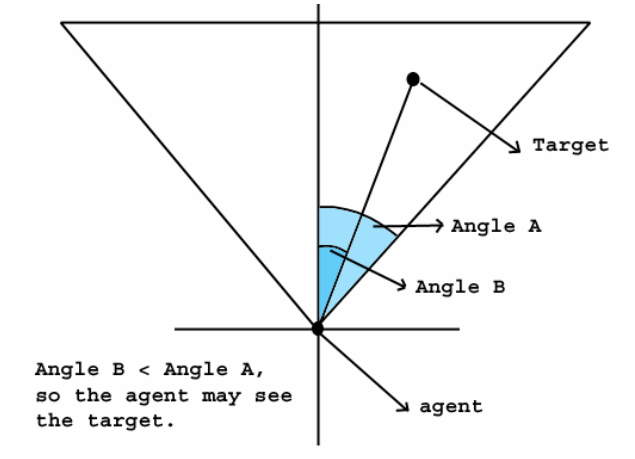

Figure 5.6: Testing viewing frustum: Angle A is the horizontal angle of the eye. Angle B is the angle between the viewing direction and the direction of the line passing through the agent and the target coordinates.

5.3.5. Stochastic Detection

5.3.5.1. Introduction

The detection process of an agent depends on the probability of detection. A probability test is done for each new perception received from the sensors, which is described in Section 5.3.1. The important question is how to compute the probability in a given situation. Being in line of sight and viewing frustum is a must, but they don't affect the probability. The main property that affects the probability of detecting a target is the distance between the target and the agent. Theoretically, there is a maximum detection range, which has a probability of detection above zero percent. Up to this range, an agent may notice the target depending on the probability. A probability curve can be given in this maximum range. But, in this study it is usually accepted as a linear graph decreasing from

100% probability to 0% probability for zero and maximum detection range in meters.

5.3.5.2. Detection Range

Detection range is the maximum distance between a specific type of target and an agent such that agent can detect the target by its visual perception. The detection range is not a static value and mostly depends on the weather condition. So we get the range values for ideal conditions and dynamically compute the new detection ranges for current state using a set of parameters defining the amount of possible changes. For example, if there is fog or rain, the detection range of a DayTV decreases by some rate. In the study, we usually get these parameters from the domain experts.

5.3.5.3. Weather Condition

5.3.5.3.1. Ideal Conditions

The weather condition that maximizes the maximum detection range of a sensor is called the ideal weather condition. Although the conditions are ideal, there is a limit on detection range because of many effects such as the volume of the target (type of the target), and the capabilities of the sensor. For example, the detection range of a tank is much more than a human because of the volume. A DayTV cannot detect far at night; in contrast an infra-red works fine in dark.

The ideal detection ranges can be found by real-life experiments. In our system, these values are taken from the experts for specific kinds of sensors and target types. Than a curve is constructed using linear interpolation. A sample probability curve is shown in Figure 5.7.

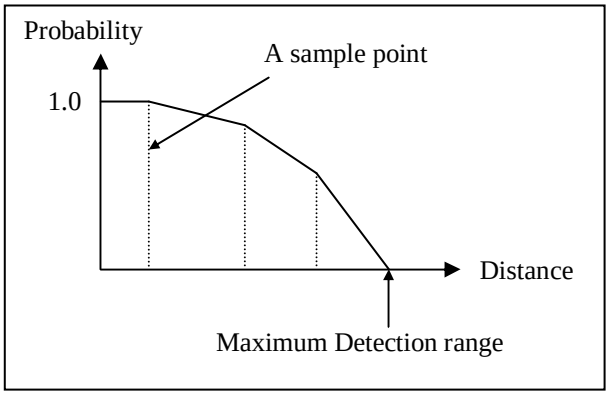

Figure 5.7: A probability graph, which is constructed from a set of sample points

5.3.5.3.2. Deviate From Ideal Conditions

After getting the maximum detection range of the ideal condition, derivations from ideals are calculated from the current weather condition and a set of parameters and graphs defining multipliers. These parameters and graphs are as follows:

- Temperature:
  multipliers as a graph constructed from a set of sample temperatures
- Light:
  multipliers as a graph constructed from a set of sample light levels
- Wind:
  multipliers for very low, low, mean, high and very high wind levels
- Rain:
  multipliers for very low, low, mean, high and very high rain levels
- Wind:
  multipliers for very low, low, mean, high and very high wind levels

- Snow:
  multipliers for very low, low, mean, high and very high snow levels
- Hull:
  multipliers for very low, low, mean, high and very high hull levels
- Fog:
  multipliers for very low, low, mean, high and very high fog levels
- Humidity:
  multipliers for very low, low, mean, high and very high humidity levels

For example, to generate the temperature graph, a set of sample temperatures and their multiplier values are given such as 0.9 for 23 degree, 0.6 for 10 degree. Than a graph is constructed from the given values using interpolation. This process is also applied to light level graph. We don't construct a graph for wind, snow, hull, fog and humidity multipliers, because these conditions are discritized as very low, low, mean, high and ver high. From these parameters and graphs, a merged multiplier is computed. Than the maximum detection range is multiplied by this multiplier to find the deviated range. This operation is sampled in Figure 5.8.

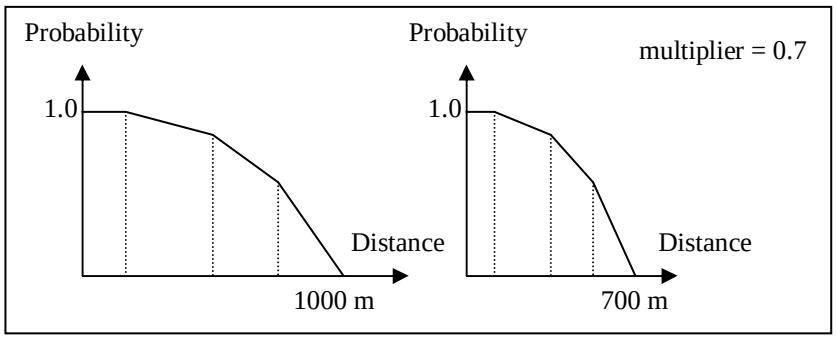

Figure 5.8: The graph on the left is probability distribution for ideal condition and the one on the right is a scaled graph multiplied by 0.7.

It is possible to visualize the probability values using sensor coverage (visible areas from a sensor). The results are shown in Figure 5.9.

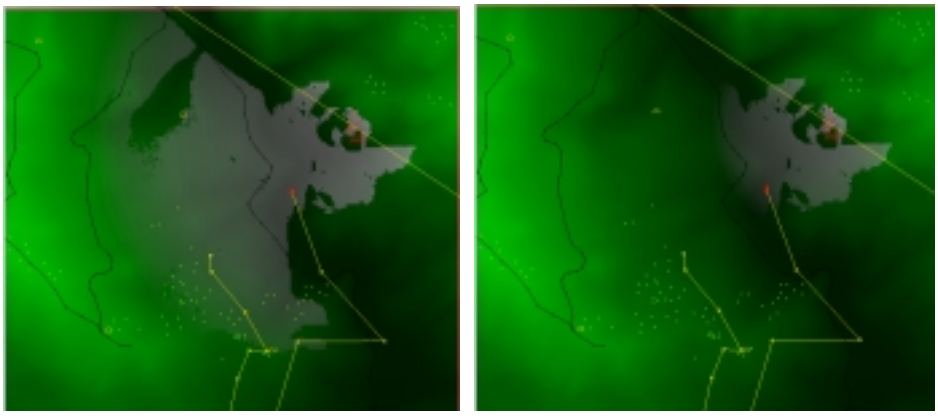

Figure 5.9: The image on the left is the coverage of a DayTV while there is no rain and the one on the left is while there is heavy rain. The change from light gray to dark green shows the probability reduces.

5.3.5.4. Plant Cover

The plant cover density also reduces the probability of detection by a multiplier between 0 and 1. To find this multiplier, all the voxels along the ray connecting the agent and the target must be traversed. This traversal is performed by the LOS algorithm, which is described in Section 5.3.3. Initially the multiplier is 1. While going along the ray, if the ray passes near to the ground (below the average height of the plant cover), the plant cover densities are used to update the multiplier. This process is illustrated in Figure 5.10.

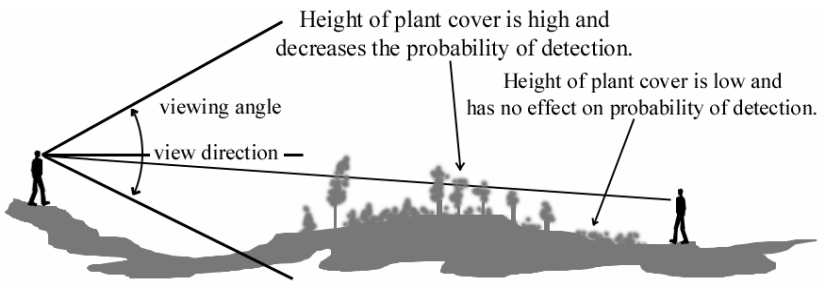

Figure 5.10: The effect of plant cover density and viewing angle

The density is a value that shows how the probability will reduce for passing a distance of 1 meter along the ray. For example, if a target is in 25 meters away in a forest, this 25 m travel may reduce the probability of detection by a multiplier 0.1, which actually depends on the plant cover density of the forest. The hard thing to do is to find out the densities in real world.

5.3.6. Hearing

There is no chance of detecting by seeing the objects that are neither in the line of sight nor viewing angle. However the objects can be sensed using audio cues. Hearing is modeled as a probability function depending on the range and the speed of the movement of the object that causes noise. The noise from a truck is because of its engine while the noise of a human is caused because of his steps. If the object is not in the line of sight (e.g., the target is behind a wall), the probability of hearing is also decreased using reasonable multiplier.

Detection by hearing is very important for an agent. For example, for a commander, being aware of his subordinates is important, but it is not possible to walk all the time by looking at every one in the group. However, the commander can hear and detect the average position of a person by its food steps. This enables the commander to aware of his subordinates without looking at them.

5.4. Classifying and Storing Perceptions

The gathered information from seeing and hearing is classified into three categories by using range and type of the target. These are detection, recognition and identification. Classification criteria are shown in Figure 5.11.

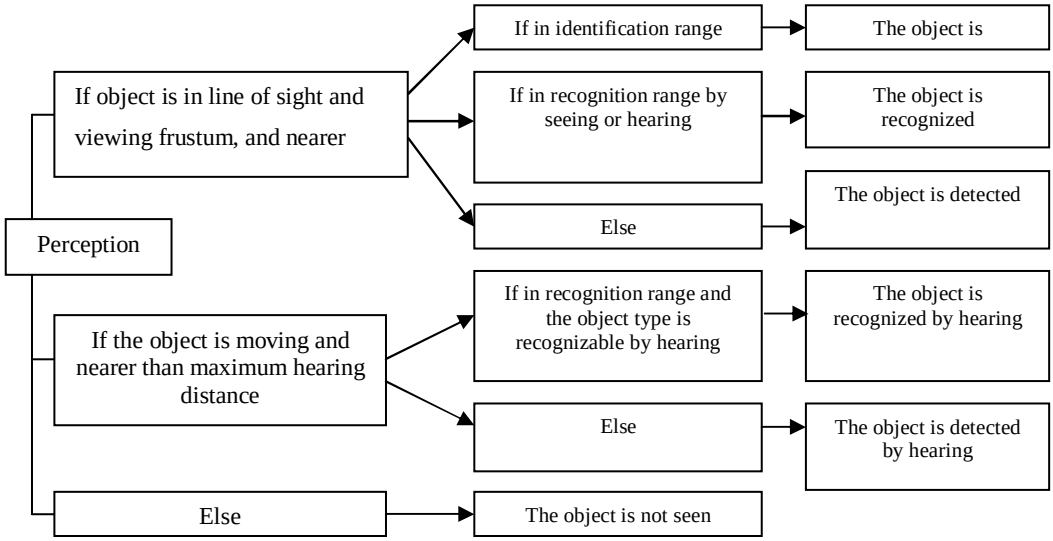

Figure 5.11: Perception is classified into three categories: detection, recognition and identification.

If a new perception occurs, it is added to the knowledge base of the agent. The knowledge base is stored in a dynamic link list. The problem is to decide whether the perception is a new object or an update of a previous detected object. Although we are in a computer-generated world and have the information of all the objects in the environment, the ID of detected object is not sent to the agent unless it is identified. The agents have to find the similarities themselves and update the knowledge base. The similarity is found out using estimated positions

of previous detected objects and the positions of new detected objects. If a similar perception is searched for the previous detected object information in the knowledge base, an estimate of position is calculated using previous movement direction. If the range between the estimated position and the new detected object position is smaller than a threshold that is calculated considering the previous speed, we may accept that they are similar. If a similar perception is found, the previous knowledge base is updated using new detected perception; otherwise the perception is added to the knowledge base as a new item. The algorithm, which controls the agents, is shown in Table 5.1.

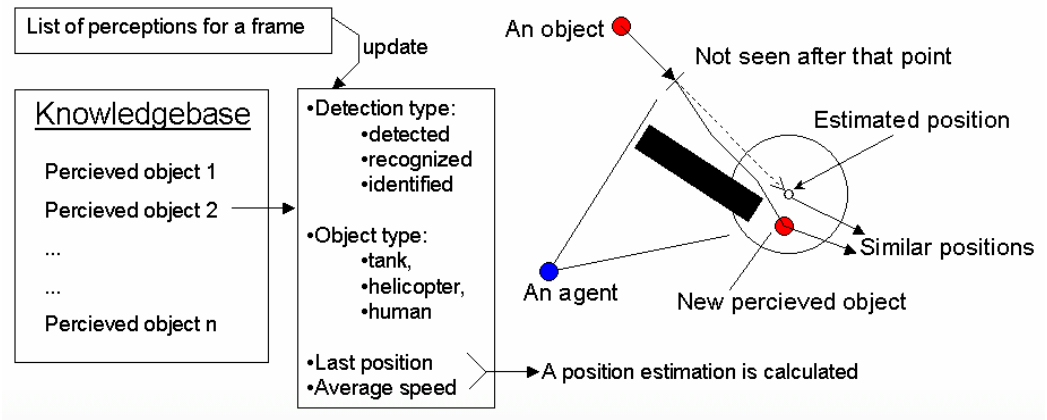

Figure 5.12: A sample scenario for similarity detection

The similarity detection is sampled in Figure 5.12. In the figure, an agent detects an object walking in the south east direction and adds it to its knowledge base. It also stores its detection type (detected, recognized, identified), object type (if known), last seen position, and average speed. After some time, the object is not seen any more because of a wall, but the agent has the previous object information and it starts to make estimation of the object position. Time passes

and the object is again seen in some other position. This time it is a new perception because it is not recognized yet. It starts to search the new perception in the knowledge base. The previous object positions were update in the previous simulation frame, so if the estimated position and the new perception position are similar, that is near than a similarity threshold which is also dynamically calculated using previous average speed (error tolerance of speed), than the new perception is said to be the same object detected before. In that situation, the previous knowledge base is just updated, else if no similar perception is found; the new perception is added to the knowledge base.

Table 5.1: The algorithm, which controls the agents

Main loop

```
For each group
    For each agent
        If the agent is a red team member (intelligent agent)
            Construct the sensor perception list for the agent; (A)
            Analyse the perception list and update the knowledge base; (B)
            Execute behavior module; (C)
            Update the physical appearance in virtual environment;
```

A) Construct the sensor perception list for the agent

```
Backup the previous perception list and create a new list;
For each group
    For each agent except himself
        Calculate the seeing and hearing statistics between the agent and the target; (D)
```

B) Analyze the detected list and update the knowledge base

```
For each member of perception list
    If the perception exists in the previous list and unsensed because of the probability test and no probability change
     occurred after that time
        Mark the member of new detected list as “not-sensed” again;
For each sensed member (not checked as “not-sensed”) of new perception list
    Do a comparison to knowledge base, find the similarities;
        If similarity found
            Update the knowledge base and check the member of perception list as “similarity-found”;
For each member of new detected list which is not checked as “similarity-found”
    Do a probability test and if it passes the test
        Add the list member to the knowledge base as a new perception;
      Else
        Check the member of perception list as “not-sensed”;
```

C) Execute behavior module

```
Find who the commander is;
Find the status of the mission plan by using previous actions and radio messages;
If the agent is a commander
    Wait or Update the path and follow the path depending on the mission status;
  Else
    If the commander position is known
        Follow the commander;
      Else
        Stop and search for the commander;
```

D) Calculate the seeing and hearing statistics between the agent and the target

```
Compute the statistics between the agent and the target (line of sight, viewing angle, range, etc.);
If there is any possibility of detection, add the perception to the perception list;
```

5.5. Decision and Reaction

Agents give decisions and react to various events using only their knowledge base. In addition to object positions, knowledge base also contains the information of who the commander is and the radio and face-to-face messages. By using this information, agents find out their commander and the status of the mission plan. If an agent is a commander, it executes the mission otherwise it follows the commander. If an agent doesn't know the position of his commander, it stops and tries to find out the commander by looking around.

Route between control points is generated using "Off-line Path Planning Module". Unless an abnormal situation occurs, the commander follows that path. Otherwise, the path is updated considering the objective, not to be seen by any sensor platform. The off-line path planning and the path update algorithm will be described in Section 6.

5.6. Group Formation

The group formation is concerned with the distribution of members within a group. A group formation is necessary for a systematic achievement of tasks. The position of a member in the group is defined by (dx,dy) coordinates relative to the commander, which is shown in Figure 5.13.

The member coordinates are calculated using the current position and direction of the commander and this calculated position is given to the member as a target position to be reached. If a subordinate is too far from its expected position, it runs for a while to take the right position else it only walks until arriving at the target position. A snapshot from a group formation is shown in Figure 5.14.

In our study, the user gives the group formation to the group. In fact, changing formations using military rules is also possible while executing the mission. This can be a future work.

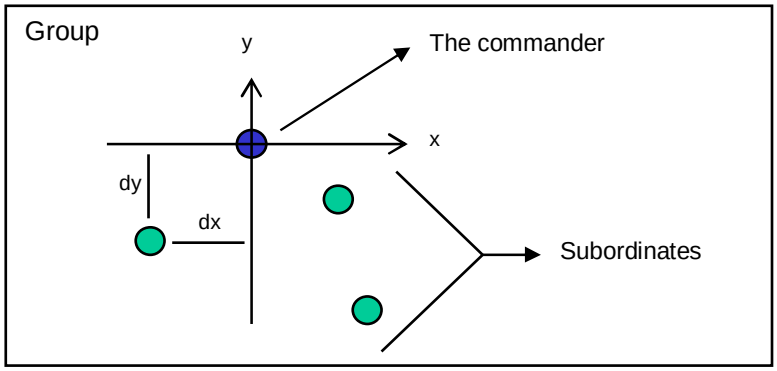

Figure 5.13: Group formation of a group with a commander and the three subordinates.

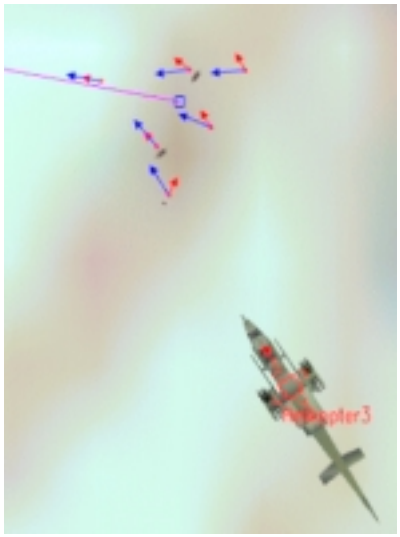

Figure 5.14: A group of agents walking in a formation: The commander is in front of five subordinates and a helicopter is about to capture them.

5.7. Group Synchronization by Radio Communication

Every group has a mission, which is a part of a high level mission. The mission is a set of control points and a list of goals to be achieved at these points. Some of these goal pieces contain radio messages to be sent at these points. Radio messages are used for coordinating the groups allowing a coordinated mission. These radio messages are:

- arriving at a control point,
- becoming ready to leave a control point,
- continuing the mission (leaving a control point),
- canceling the mission,
- sending a keyword,
- an enemy identified,
- being in danger (I am in danger),

Sending a radio message is managed using a message string carrying the information to be sent. A message string includes the following fields:

- Radio message type,
- The sender group,
- The receiver group,
- The sender group member,
- The receiver group member,
- Where the message is sent from,
- A keyword string.

Although the message may contain who the message is for, it is heard by all the agents having a radio message receiver, so all the agents know the status of the groups and the mission. A set of radio communications occurred while performing a mission is shown in Table 5.2.

Table 5.2: The list shows a set of radio communications occurred on a mission. At specific coordinates (1.4, 2.3, 3.3), all the groups are rendezvoused and synchronized, than continued the mission at the same time.

group (2 and 3) is waiting for keyword “First step go” from group (1).
- ♦ Radio Msg. From group (1) to all : we are leaving point 1.1 to continue mission
- ♦ Radio Msg. From group (1) to all : send keyword >> “First step go”

Radio Msg. From group (2) to all : we are ready to continue from point 2.1
- ♦ Radio Msg. From group (3) to all : we are ready to continue from point 2.1

Radio Msg. From group (2) to all : we are leaving point 2.1 to continue mission
- ♦ Radio Msg. From group (3) to all : we are leaving point 2.1 to continue mission

Radio Msg. From group (2) to all : we arrived at point 2.2
Radio Msg. From group (2) to all : we are ready to continue from point 2.2
Radio Msg. From group (2) to all : we are leaving point 2.2 to continue mission
- ♦ Radio Msg. From group (1) to all : we arrived at point 1.2
- ♦ Radio Msg. From group (1) to all : we are ready to continue from point 1.2
- ♦ Radio Msg. From group (1) to all : we are leaving point 1.2 to continue mission
- ⇒ Radio Msg. From group (3) to all : we arrived at point 3.2
- ⇒ Radio Msg. From group (3) to all : we are ready to continue from point 3.2
- ⇒ Radio Msg. From group (3) to all : we are leaving point 3.2 to continue mission
- ♦ Radio Msg. From group (2) to all : we arrived at point 2.3

group (2) is waiting at point 2.3 for message “arrival at point 1.4” from group (1).
- ⇒ Radio Msg. From group (3) to all : we arrived at point 3.3

group (3) is waiting at point 3.3 for message “leaving point 2.3” from group (2).
- ♦ Radio Msg. From group (1) to all : we arrived at point 1.3
- ♦ Radio Msg. From group (1) to all : we are ready to continue from point 1.3
- ♦ Radio Msg. From group (1) to all : we are leaving point 1.3 to continue mission
- ♦ Radio Msg. From group (1) to all : we arrived at point 1.4

group (1) is waiting at point 1.4 for keyword “Ready for Mission” from group (2).

Radio Msg. From group (2) to all : we are ready to continue from point 2.3
Radio Msg. From group (2) to all : send keyword >> “Ready for Mission”

group (2) is waiting at point 2.3 for keyword “Target is Kartal Gozu” from group (1).
- ♦ Radio Msg. From group (1) to all : we are ready to continue from point 1.4
- ♦ Radio Msg. From group (1) to all : send keyword >> “Target is Kartal Gozu”

Radio Msg. From group (2) to all : we are leaving point 2.3 to continue mission
- ⇒ Radio Msg. From group (3) to all : we are ready to continue from point 3.3
- ♦ **Radio Msg. From group (1) to all : we are leaving point 1.4 to continue mission**
- ⇒ **Radio Msg. From group (3) to all : we are leaving point 3.3 to continue mission**

5.8. Location Availability

If an agent can pass through a specified voxel, that voxel is said to be an available location for the agent. For the agents, a matrix storing the available and not-available locations is allocated. The size of location availability matrix is equal to the terrain elevation matrix. The matrix is filled by the user in design phase. For the process, a set of tools is supported. These are setting elevation limit, slope limit, and surface material limit.

The elevation limit culls the voxels being above, below or equal to an elevation value. The result of culling the voxels above 1500 m is shown in Figure 5.15.

The slope limit similarly culls the voxels being above, below or equal to a slope magnitude value. The result of culling the voxels having slope magnitude more than 30 degree is shown in Figure 5.16.

The surface material limit culls the voxels being equal to a selected material type. For example, culling fresh water means setting rivers not-available places.

All these culling operations can be applied any number of times and in any order resulting a location availability map mixed of different operators shown in Figure 5.17.

Figure 5.15: Location availability matrix after culling voxels above 1500 m (light colored regions are available locations)

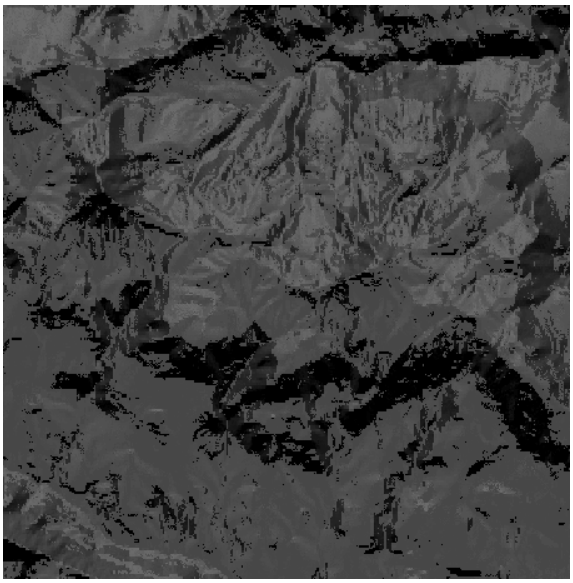

Figure 5.16: Location availability matrix after culling voxels having slope magnitude more than 30 degree (light colored regions are available locations)

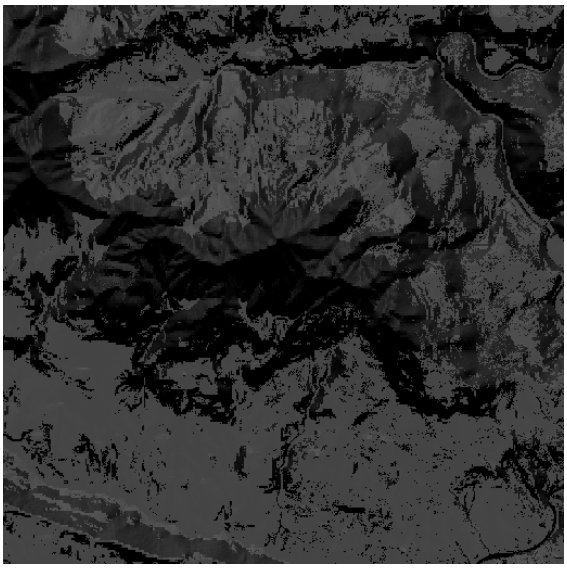

Figure 5.17: Location availability matrix after culling voxels above 1500 m, having slope magnitude more than 30 degree and having material water, rock or forest (light colored regions are available locations)

5.9. Physical Modeling

A 3D model is generated for the agents in order to be sensed by the platform sensors. A sample model is shown in Figure 5.18. For the physical appearance of the agents in the environment, only the coordinates are used and body motions are ignored. The motion kinematics or motion capture may be used to generate their body movements, but this may be a future work.

Figure 5.18: The 3D human model used for agents

CHAPTER VI

PATH PLANNING

6.1. The Objective

The group mission is given by a set of tactical control points. So we need to plan a path connecting these points in order to complete the mission. If a path is generated, the commander may follow the path to complete the given task.

Off-line path planning is not suitable for real time mission execution. It serves a static path that is only available for short period of time, but agents are in a dynamic environment and unknown threats may appear in any time, so a new path has to be generated under abnormal conditions. Generating an off-line path for every frame is an inefficient way of mission planning. Indeed, only a partial update is enough in many situations.

In this study, two different approaches are proposed for real-time mission planning. First one is an off-line path planning followed by a real-time path update algorithm, which partially update the path during the simulation cycle. And the alternative is a real-time goal directed path search algorithm that does not need any off-line path-planning phase.

Both approaches have some advantages and drawbacks. Planning of an off-line path is a natural way of deciding a mission plan before going out to field. You generate a plan on the map, and want to keep it same as much as possible. However, during the mission execution, many abnormal events may happen and the previous plan can be completely unfeasible. Partially updating the path may

not be enough and off-line path planning may be inevitable. The second approach, an efficient real-time path search algorithm, can lead to a solution, but this is the way that ignores a fact. In our problem, we assume that the map is completely known, so ignoring that information is not a good way of planning.

6.2. Off-line Path Planning

In this study, an off-line path-planning algorithm that finds a path connecting two points on a large landscape is proposed. The algorithm doesn't guarantee the shortest path, but finds an acceptable short path that does not contain any loops. A sample result is shown in Figure 6.1.

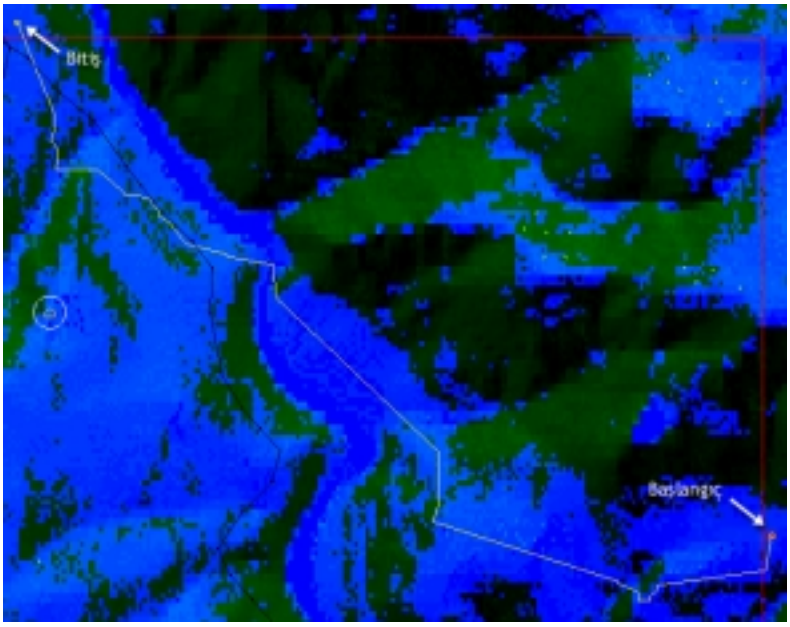

Figure 6.1: The image shows an off-line path search result between two given points. The dark regions are not available for passing through.

The main advantage of the algorithm is its efficiency as it runs in linear time and prevents loops. It is similar to the breath-first search algorithm, which uses heuristics aiming to the goal point. It searches block regions, checks where it has gone before and never comes back again. Thus, the complexity of the algorithm is $O(nm)$, where n and m are the number of cells in horizontal and vertical axes. Thus, n times m gives the total number of voxels.

The second advantage is its possibility to be integrated to real-time applications by doing simple modifications. The off-line search steps can be distributed over simulation frames, which allow the algorithm to search a path in real-time. This approach is used in our proposed real-time path search algorithm. Another advantage is that it allows finding random paths by changing the parameters of heuristic function randomly during the search.

Although it has the advantage of being efficient, it has also disadvantages. The main drawback of the algorithm is its generated paths, which are not the shortest paths in general. The result is usually near to the shortest path, but not the shortest one. Also the generated path usually contains sharp edges, which is illustrated in Figure 6.1 and Figure 6.2. These sharp edges can be fixed using the proposed path update algorithm, which usually aims to minimize the curvature.

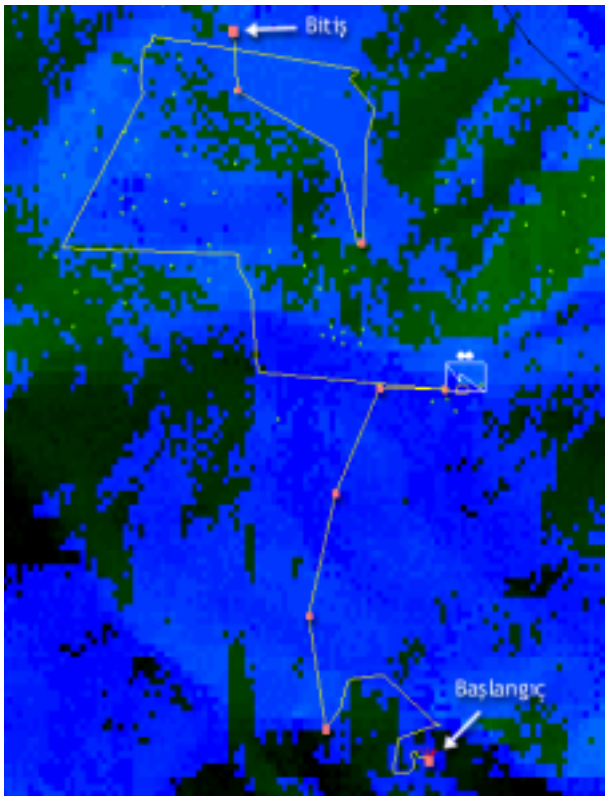

Figure 6.2: A set of tactical control points and generated paths between these points.

We assume that the landscape is defined as a 2 dimensional matrix of elevations, which forms a grid. Each grid member (voxel) of the landscape is marked as available or not available for passing through by considering slope magnitudes, surface materials, etc. The planning starts from the initial position (sx,sy) called the starting point and ends at the position (tx,ty) called the target point.

First, we horizontally move left from the point (sx,sy) until a not available voxel is reached. Then, the same search is done to the right direction. Thus, the left and right boundaries (a1 and a2) of the point (sx,sy) are found. The points, a1 and a2 are x coordinates of two voxels that are on the same row of starting point (sx,sy). Indeed, the found boundaries form a 1-dimensional horizontal line (the y

coordinate of the line is ly) and it splits the local area into north and south part, which is illustrated in Figure 6.3.

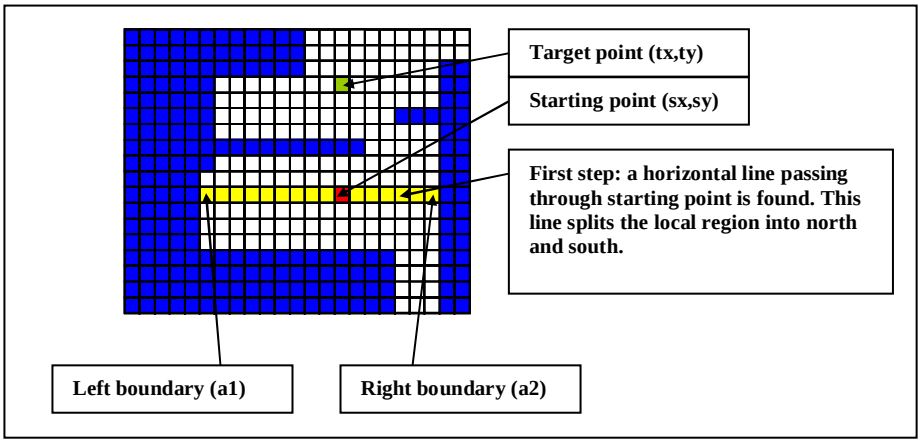

Figure 6.3: The initial state of the algorithm

Next step is to call a recursive function, “boundary”, by sending the current parameters ly, gx, a1, a2, where ly is the current y coordinate of the line and gx is the x coordinate of entry point to the line that are initially the sx and sy coordinates, a1 is the left bound of (gx,ly) and a2 is the right bound of (gx,ly). Each recursive function checks the north and south neighbor voxels of the bounded line (ly,a1,a2) for available exits. If one or more new neighbor exits are found, one of them is chosen using a heuristic function and “boundary” function is called again for the new exit line. After entering each recursive function the bounded line (ly,a1,a2) is filled with a sign for the sake of preventing loops, which means “gone before”. The process steps are illustrated on a sample in Figure 6.4.

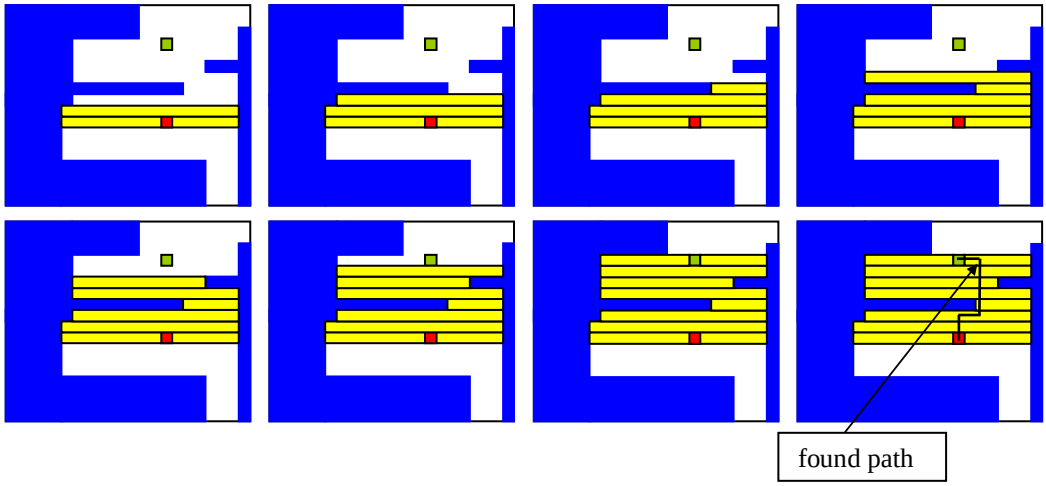

Figure 6.4: The recursive search process continues until reaching the line that the target point is on.

When a path is found as a result of the search, the result usually contains too many control points. So, it needs to be optimized. The result is optimized while going out of the recursive functions. Thus, the process occurs from target point to starting point, which is actually the reverse order. While each point is added to the list, a linear connection test is done with the previous added control points. If any linear connection is available among the points am and an, all the interval control points in the range of am+1 and an-1 are deleted. The C-like pseudo code of the basic path-planning algorithm is shown in Table 6.1 and an optimized path is illustrated in Figure 6.5.

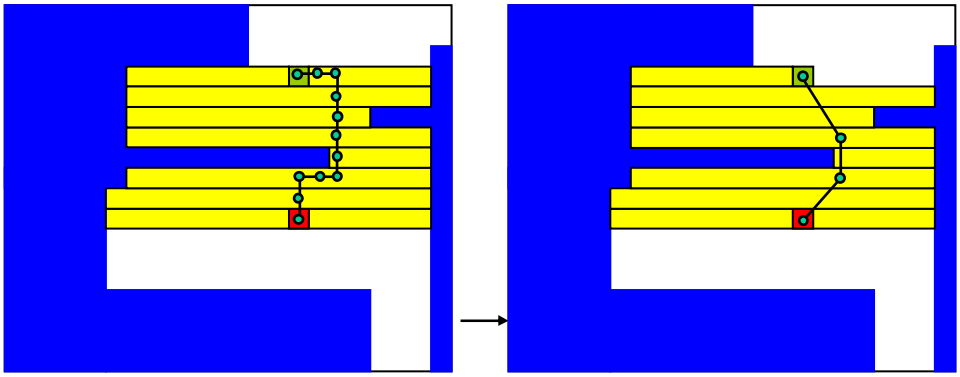

Figure 6.5: An off-line generated path (left) and an optimized version of the same path (right)

The optimization decreases the number of path points significantly, but the drawback is the time spent while checking the availability of linear path between two points. The LOS algorithm described in Section 5.3.3 is used for this purpose.

Table 6.1: The basic algorithm for the off-line path planning

```
short tx;
short ty;
short found;
```

```
void boundary( short ly, short gx, short a1, short a2 )
{

          short tar;
          short goinfrom;
          char xyon, yyon;

          if ((ly == ty) && (tx>=a1) && (tx<=a2))
                    {
                      found = true;
                      return; // path is found, generate the path now
                    }

check the voxels on line (ly,a1,a2) as “gone before”

// if target is at north, search north first else search south first.
          if ( ly<ty ) yyon = 1; else yyon = -1;

// if target is near to a1, start search from west else start search from east.
          if ( abs(a1-tx)<abs(a2-tx) ) xyon = 1; else xyon = -1;

search the north and south sides considering the priority determined
by xyon and yyon, find all the available neighbor regions (ly±yyon,n1,n2) and
their suitable entry points (goinfrom)
call boundary( y±yyon,n1,n2,goinfrom);
                              if (found)
{
// path is already found, continue generating the path
                              return;
}
          }
```

```
char Search_Path( short sx, short sy, short targetx, short targety )
{
          short a1,a2;

          tx = targetx;
          yy = targety;

          a1 = left boundary of sx,sy
          a2 = right boundary of sx,sy

          found = false;

          boundary( sy, sx, a1, a2 );

          return found;
          }
```

6.3. Real-Time Path Refinement and Update

6.3.1. Energy Minimization

After having generated an off-line path, the commander follows the path under normal conditions. When a threat is detected, this path needs to be updated considering the new situation. For that purpose, a path update algorithm is developed based an energy minimization of path points, which is a commonly used algorithm in image processing called the snakes.

The algorithm assumes that every tactical control point has a potential energy, which is harmful such as radioactive energy. The objective is to get rid of or minimize this harmful energy. The energy sources are in two forms: internal or external. Internal energies are the curvature of path on the control point and deviation of initial distances between the neighbor control points. The external energy sources are the elevation, slope, and the threats.

6.3.2. Internal Energies

The two neighbors of a control point form a curvature, which is illustrated in Figure 6.6. Having high curvature on a path point is an unwanted effect, because this may cause the path become longer. So the curvature increase also increases the negative energy on a control point.

In the Figure 6.6, the curvature of point 2 is computed using the following formula:

$$Curvature = sqrt (cx2 + cy2) / (d1 + d2) ,$$

$$where \;\; cx = x1 – 2x2 + x3 \; , \; cy = y1 – 2y2 + y3$$

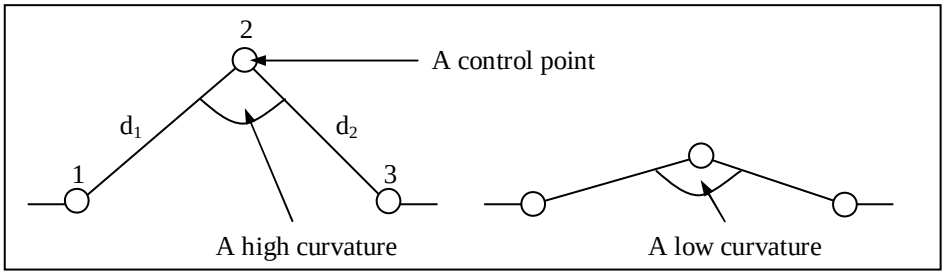

Figure 6.6: Illustration of a high and low curvature. Moving a control point where causes a low curvature shortens a path.

Another internal energy is the continuity. In ideal conditions, the distances between the neighbor points (edge distances) must be equal to each other and also the average of edge distances. The deviation from the average distance causes to increase the energy, which is shown in Figure 6.7.

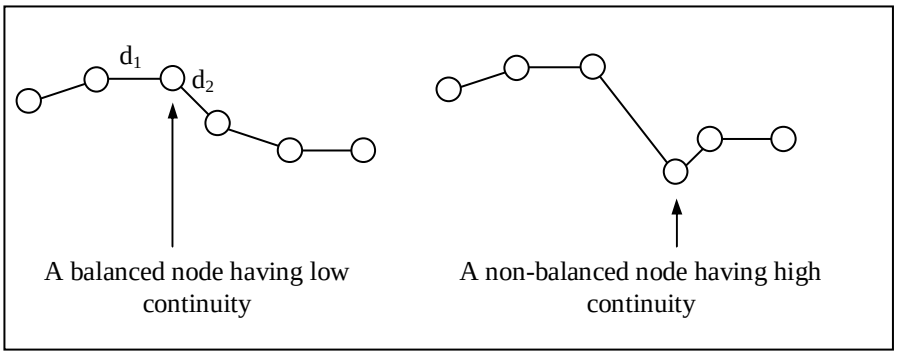

Figure 6.7: Illustration of a balanced (left) and not balanced path (right)

In Figure 6.7, the continuity of the left node is computed using the following formula:

$$Continuity = (r_1^2 + r_2^2) / 2 ,$$

$$where \ r_1 = abs (d_1 - average\ distance) / average\ distance ,$$

$$r_2 = abs (d_2 - average\ distance) / average\ distance$$

6.3.3. External Energies

Slope magnitude is one of the external energy functions to be minimized. If slope magnitude increases, the negative energy also increases. The slope magnitude is a value between 0 and 90, so the Slope function is as follows:

Slope Energy = slope magnitude / 90

Another introduced energy source is the elevation. The objective is to minimize the control point elevations. So the function is as follows:

Elevation Energy = elevation / maximum elevation on the terrain

The last external energy function is for the threats. We assume that the threats are energy sources, and being near to a threat increases the collected negative energy on a control point. So the aim of the control points is to escape from threats, which is illustrated in Figure 6.8. Every threat has a limited visual capability and the agents has some believes about these capabilities. Using these believes, they find the maximum detection range (maximum range) for a specific type of threat. If a threat is detected and it is near to a control point, the energy on the point starts to increase depending on the maximum range. The function is as follows:

If (*distance to threat>maximum range*) then *Threat Energy = 0 else*
Threat Energy = (maximum range – distance to threat) / maximum range

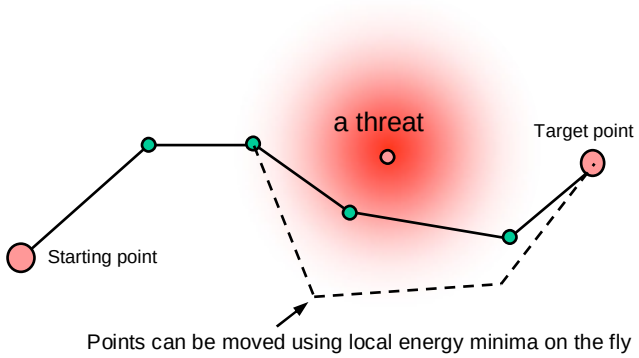

Figure 6.8: Threat energy causes the path points go far away from the threats.

A total energy is computed from the energy functions using suitable weights, and all the control points are updated each frame. To update a control point, the energy of each voxel, which is near to the control point and has an available linear connection to the next point, is calculated. Then the control point jumps to the voxel, which has the minimum energy. The result of this process is shown in Figure 6.9.

Using the minimization algorithm, the path refinement can be done to a limit, but the result is not satisfactory all the time, because this process cannot modify a path radically. For example, the path cannot pass over an obstacle, which is illustrated in Figure 6.10.

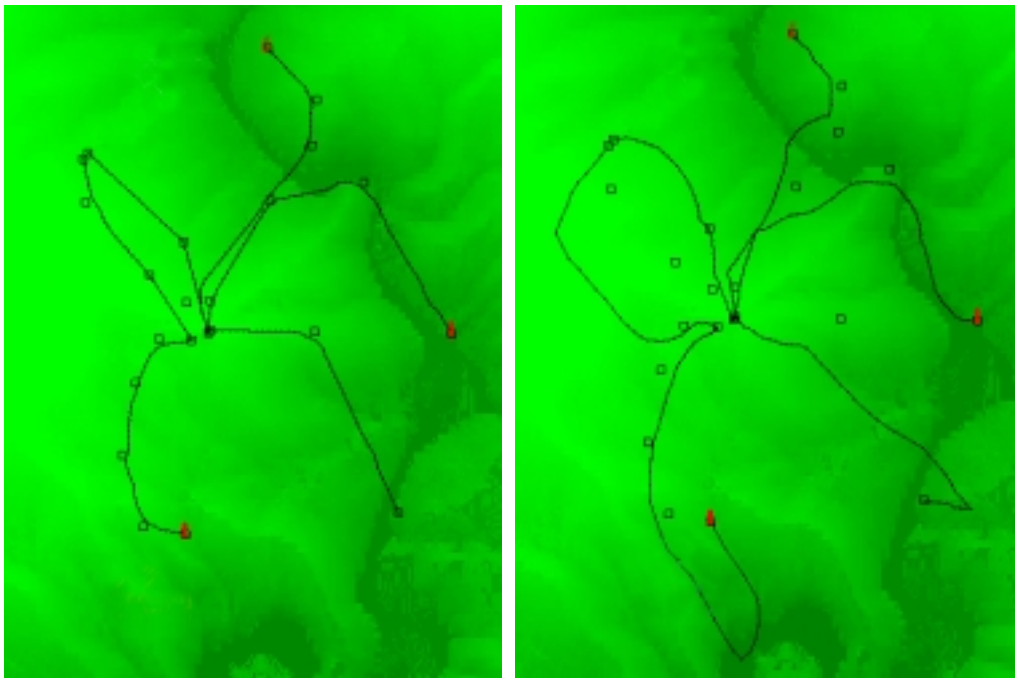

Figure 6.9: A path update sample: Left figure shows the initial paths of three agent groups. Their target, which is a non-deformable control point, is at middle of the area. Right image shows the updated paths after some time.

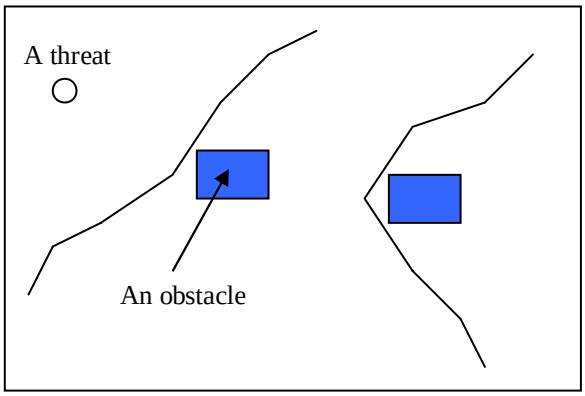

Figure 6.10: The path on the left aims to flow right, but an obstacle prevents this movement.

6.4. Real-Time Path Search: Real-time Horizontal A*

If the agent doesn't have a complete map or an off-line generated path, it has to decide on the fly while executing the mission. Thus, you don't plan an off-line path, but you search a path that will make you reach the goal state. Our proposed real-time goal directed search algorithm is based on off-line path planning algorithm introduced in Section 6.2. This is a modified version of the algorithm integrated with real-time A* [MY96]. In the algorithm, linear regions (horizontal lines) and real-time A* approach are used to direct the search, so we call it Real-Time Horizontal A* (RTHA*).

In the off-line path planning, the search is started from the bounded horizontal line passing through the starting point (sx,sy). Then the search goes on by jumping available neighbors of the current line. Similar to off-line path planning, RTHA* search the area horizontally, but the process is done on the fly distributed to the whole simulation period.

Real-Time A* is a greedy search algorithm that uses heuristics to direct the search. It evaluates the costs of the neighbor voxels at the current position and jumps to the voxel having minimum cost. While jumping to the next voxel, the algorithm writes 1 plus the cost of the second best neighbor to the previous voxel. This is illustrated in 6.11.

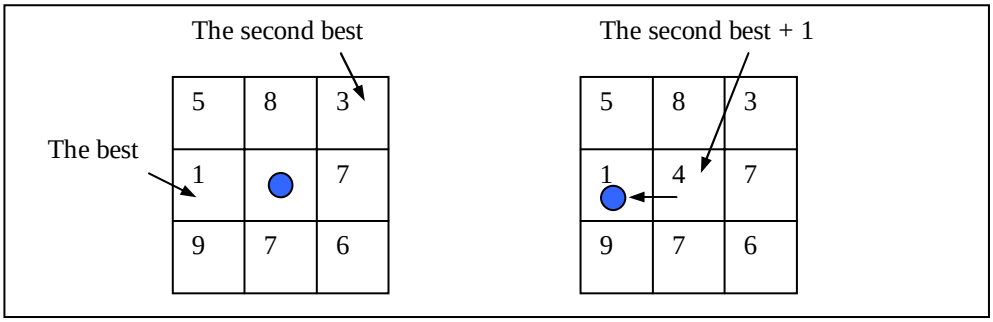

Figure 6.11: State transition of Real-Time A*

The algorithm is effective for maze environments, but if the terrain is large and there are many semi-closed regions having large open areas inside, the agent may be stuck in the regions for a long time, because the search strategy is too local, only the neighbor voxels are evaluated. This state is simulated in Figure 6.12. Real-Time Horizontal A* is proposed to prevent that problem. RTHA* extends the search space from single voxels to large linear regions, so making the algorithm more useful in large environments having many open areas and semi-closed areas.

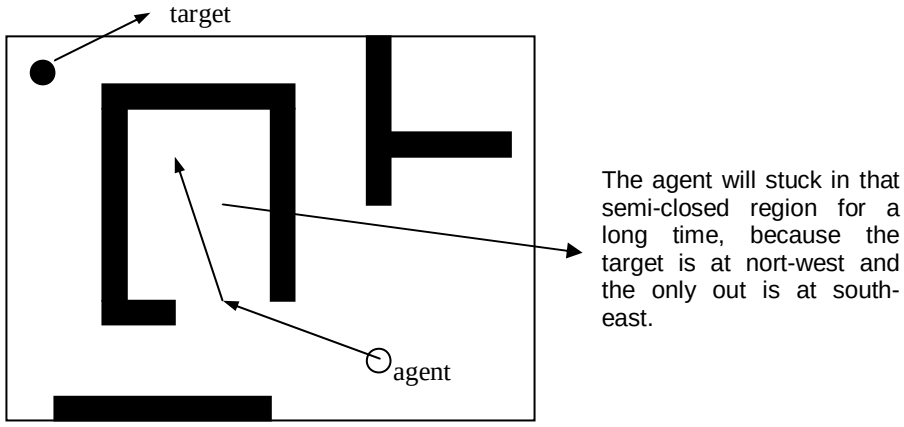

Figure 6.12: The agent, directed by Real-Time A* may stuck in semi-closed regions having large open areas inside for a long time searching the same voxels hopelessly.

RTHA* uses the same cost update technique as real-time A*, but it evaluates the costs of the neighbor linear regions instead of a single voxel and decides to reach the region having the minimum cost. While going to the next region, the algorithm writes 1 plus the cost of the second best neighbor to the previous region. This is illustrated in Figure 6.13.

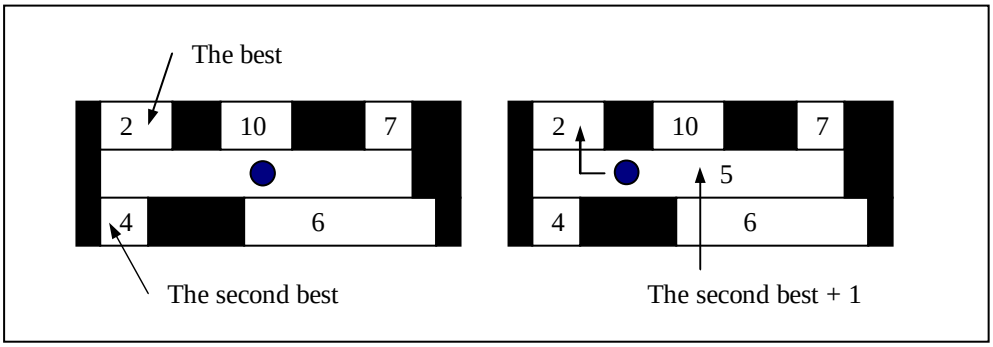

Figure 6.13: State transition of Real-Time Horizontal A*

The algorithm almost gives the same results for maze environments, but if the terrain is large and there are many semi-closed regions having large open areas inside, the agent can go out of the region much more earlier than real-time A* does, because the search strategy is not as local as real-time A*. This state is simulated in Figure 6.14.

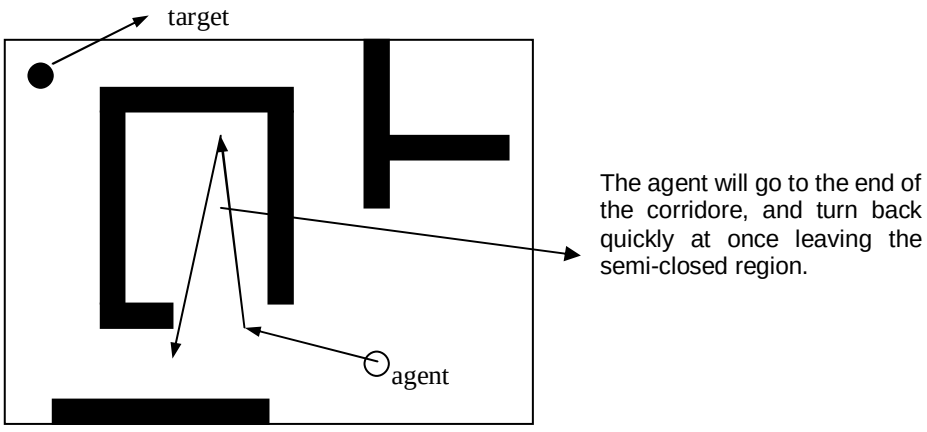

Figure 6.14: The agent, directed by Real-Time Horizontal A* can go out of semi-closed regions quickly.

Table 6.2: The basic algorithm for Real-Time Horizontal A*

```
x          = current x coordinate of the agent;
y          = current y coordinate of the agent;
current_a1 = left bound of x,y , initially –1;
current_a2 = right bound of x,y , initially –1;
```

```
float GetCost( short a1, short a2, short ay )
{
return minimum distance of the target to the bounded horizontal line (a1,a2,ay);
}
```

```
aa1        = left bound of best region found;
aa2        = right bound of best region found;
yy1        = y coordinate of the best region found;

if (current_a1 == -1) // initially, current_a1 and current_a2 must be found
{
       current_a1 = left boundary of x,y;
       current_a2 = right boundary of x,y;
       };

if (abs(targetx-x)<1 && abs(targety-y)<1) the target is reached, stop search;

if (current_a1<=targetx && targetx<=current_a2 && y=targety) // at the row of the target
{
if (targetx<x) x-- else x++; // go horizontally to the direction of the target
}
     else
{
Search all the neighbor linear regions and
find the best and the second best regions by calling GetCost evaluation function
(the variables aa1, aa2 and yy1 is found)

// go to the best linear region if any found

if (aa1<=x && x<=aa2) // go one voxel up or down in the direction of the target
{
               current_a1   := aa1;
               current_a2   := aa2;
               y            := yy1;          // jump to the row of best region
if (x<targetx && x<aa2) x++   // go right voxel mean while
     else
if (x>targetx && x>aa1) x--;  // go left voxel mean while
update the region cost of the previous region as the second best + 1;
}
      else
if (aa1<x) x--   // go left voxel
      else x++;  // go right voxel
};
```

The algorithm for RTHA* is given in Table 6.2. The path search function is called at each frame in the simulation. When the function is called for the first time, the left and right bounds of the initial agent coordinate (x,y) will be found. Then, the current state is evaluated. If the agent is at the target point, it stops. If the agent is at the same row as the target, then it just goes left or right, else it searches for the best linear region to be targeted.

The main drawback of both RTA* and RTHA* is being not capable of handling dynamic environments successfully. For example, if new threat information is gathered, than it takes time to converge to a new direction, because the cost function output for each region changes and the previously stored cost values become partially invalid. While time passes, the costs are updated to new values, but during this time period, the probability of been captured by the threat increases significantly.

CHAPTER VII

PERFORMANCE ANALYSIS

7.1. Test Platform

In this study, off-line path planning and path update algorithms are implemented by C++ programming language under both Windows and SGI IRIX platforms. The test environment for Real-Time A* (RTA*) and Real-Time Horizontal A* (RTHA*) are implemented by Borland Delphi programming language under Windows platform, which is shown in Figure 7.1.

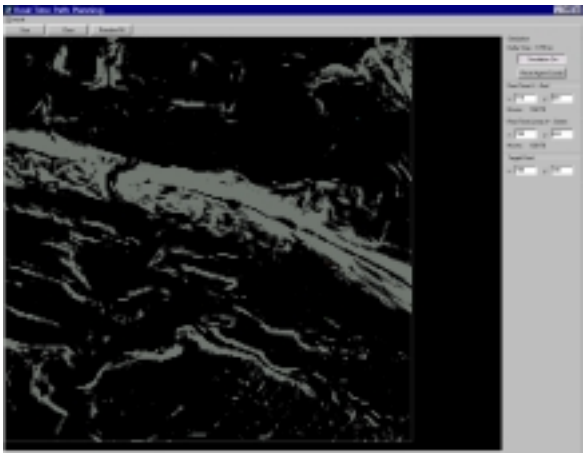

Figure 7.1: The test program for Real-Time A* and Real-Time Horizontal A*

We used a high resolution mountainous terrain having 2251x2251 voxels and split it into 6 regions to perform tests. These 6 regions are shown in Figure 7.2, 7.3, 7.4, 7.5, 7.6, and 7.7. White colored regions denote accessible regions and dark colored ones denote not accessible.

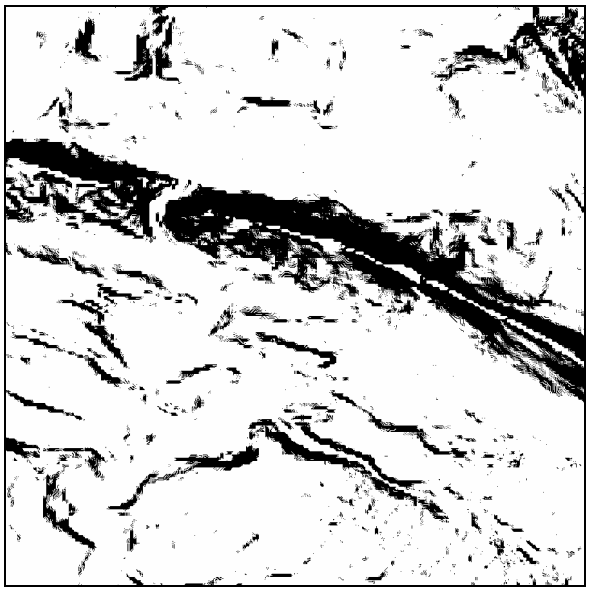

Figure 7.2: Test region 1: A map having widely open area is north and south part, but there is only one mountain pass between these regions.

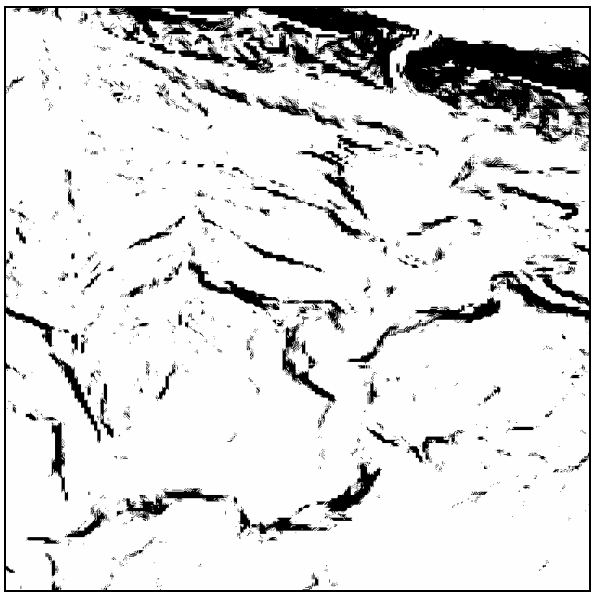

Figure 7.3: Test region 2: A map having widely open areas, which is the south path of the one shown in Figure 7.2.

Figure 7.4: Test region 3: A mountainous area having many semi-closed regions.

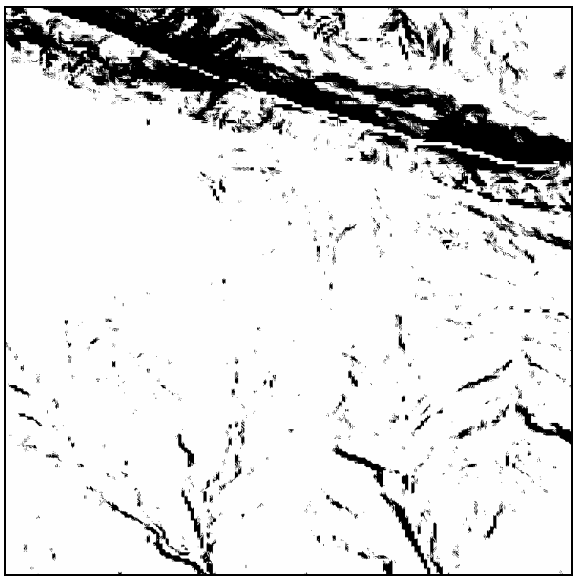

Figure 7.5: Test region 4: A map having widely open areas, which is the east path of the map shown in Figure 7.3.

Figure 7.6: Test region 5: A mountainous area having many semi-closed regions. The map is the west part of the one shown in Figure 7.4.

Figure 7.7: Test region 6: A mountainous area having many semi-closed regions. The west and east parts are separated by narrow mountain passes. The map is the west part of the one in Figure 7.6.

*7.2. Test Results of Real-Time Horizontal A**

In the tests, two agents are used, which are implemented by RTA* and RTHA* algorithms. Various starting and target points are chosen on these maps illustrating different initial states and region conditions. For each test, a target point and a starting point are given, and the two agents start from the same starting point and end at the same target point.

Agents using RTA* and RTHA* can move in 9 different ways: north, south, east, west, north-east, north-west, south-east or south-west. There are only four states that an agent cannot move diagonally although the voxel is available, which are shown in Figure 7.8.

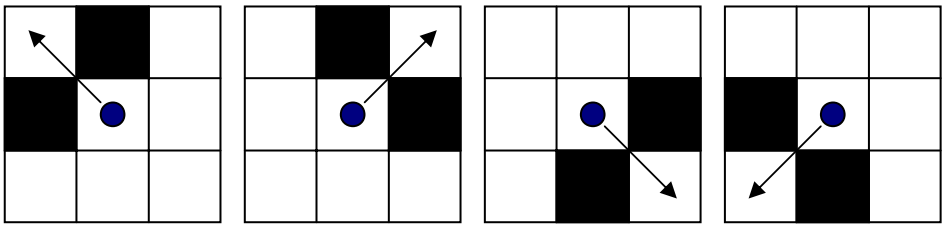

Figure 7.8: Moves that are not allowed for both RTA* and RTHA*

Concerning the performance, we used one criterion that is the number of moves performed by an agent before reaching the goal. In each test run, two outputs are obtained (one from RTA* and one from RTHA*). Having lower values for number of moves means spending less time and energy to reach the goal state. For a chosen coordinate pair, tests are performed in both forward and backward directions. The first test is done choosing the first coordinate as the starting and the second as the target point. Second test swaps these points making the moving direction reverse. The results are shown in Table 7.1.

In addition, we have done experiments to compare efficiency of both algorithms on an Intel Celeron-466, 128 MB Ram without displaying the process on the screen. RTA* performed 1000 moves in 0.135 seconds (7407 frames/second) and RTHA* performed 1000 moves in 1.666 seconds (1666 frames/seconds). The efficiency of RTA* is 4.444 times better than RTHA*, but both of them are acceptable for real-time applications.

Table 7.1: RTA* vs. RTHA*: the gray cells are the ones RTA* performs better. The performance increase is usually slight in RTA*, but in RTHA* it is generally very high.

Area	RTA* moves (forward)	RTHA* moves (forward)	RTLA/RTA ratio (forward)	RTA* moves (reverse)	RTHA* moves (reverse)	RTLA/RTA ratio (backward)	Average ratio
R1.1	133,351	7,560	17.639	4,280,827	2,575,641	>1.662	>9.650
R2.1	149,344	361,859	0.412	1,234	5,380	0.229	0.320
R2.2	727	1,240	0.586	3,535	2,177	1.623	1.19
R3.1	598	1.156	0.517	54,360	4,785	11.360	5.938
R3.2	148,382	2,012	73.748	519,759	50,814	10.228	41.988
R3.3	13,094	1,932	6.777	1,408	10,427	0.135	3.456
R4.1	832	4,882	0.170	982	1,348	0.728	0.449
R5.1	78,509	21,505	3.650	139,096	21,429	6.491	5.070
R5.2	49,976	10,395	4.807	98,060	4,080	24.034	14.420
R6.1	580	551	1.052	542	320	1.693	1.372
R6.2	>2,858,922	705,010	>4.055	>662,805	45,778	>14.478	>9.266
R6.3	1,721	1,799	0.956	1,074	1,346	0.797	0.876
R6.4	1,337	586	2.281	447	654	0.683	1.482
R6.5	13,404	5,173	2.591	893	1,009	0.885	1.738
Total	3,450,777	1,125,660	-	5,765,022	2,725,188	-	-
Avg.	246,484	80,404	8.510	411,787	194,656	5.359	6.943

The tests on RTA* and RTHA*, shows that in many situations RTHA* is more powerful than RTA*, but RTA* usually performs better in widely open areas such as the ones in Figure 7.3, 7.5. RTHA* usually aims to move in east west direction. So if the area is widely open, first, the agent goes in the direction east west and centers the target in x coordinate, than moves north or south direction, which is illustrated with one of the sample areas (Result R4.1) in Figure 7.9. In the figure, the target is at north west and the agent is at south east. So the agent moves west for a while to center the target in x-axis. Than, it moves north. This causes the RTA* algorithm to reach the target slightly earlier than RTHA*, but the difference is not so much in general.

Having large horizontal regions has another bad effect on RTHA*. RTHA* search algorithm is based on horizontal lines and their neighbor linear regions. If

the horizontal lines are wide, this causes the agent move a lot in the direction east west making it spend so much effort and time while traveling around. This effect is illustrated with one of the sample areas (Result R2.1) in Figure 7.10.

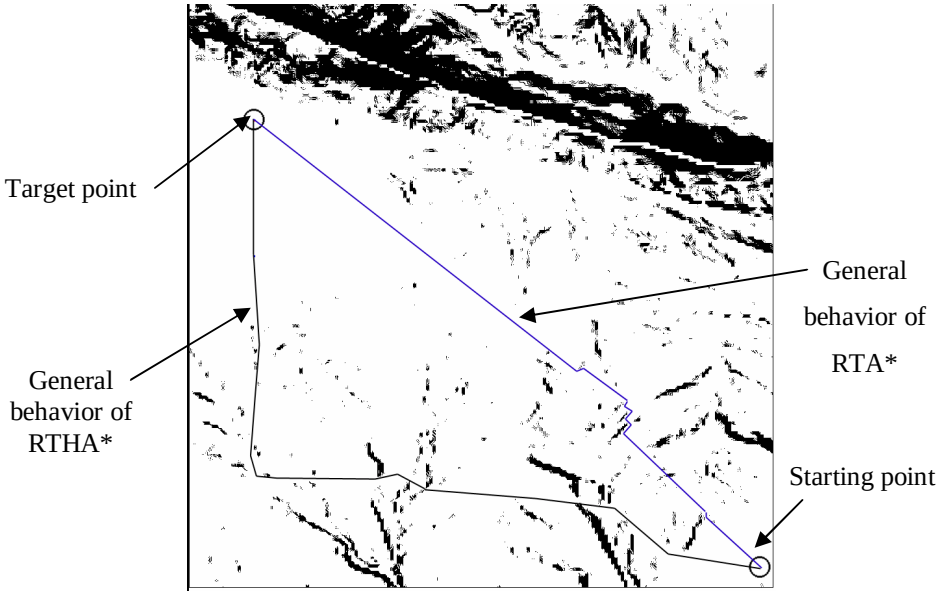

Figure 7.9: Performance of RTA* and RTHA* on a widely open area

If the area is complex, having semi-closed regions a lot as illustrated in Figure 7.4, 7.6 and 7.7, than RTA* becomes very ineffective and time consuming, in contrast the RTHA* becomes a very good alternative. In that condition, the performance of RTA* reduces significantly, whereas the performance of RTHA* increases, because the semi-closed regions badly effect the RTA* algorithm and make it get stuck inside these areas for a long time period.

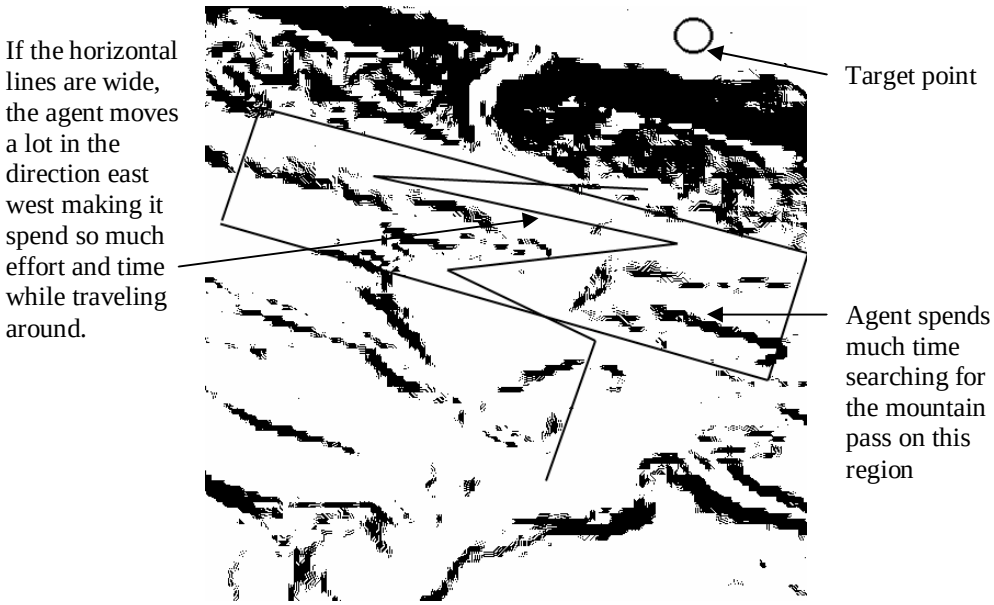

Figure 7.10: Having large horizontal regions has a bad effect on RTHA*.

CHAPTER VIII

CONCLUSION AND FUTURE WORK

In this thesis, we have studied the concept of computer-generated forces in order to construct a real-time simulation system. We have formed defense and assault forces against each other on a large mountainous landscape and tried to solve real-time path planning problem for mission planning purposes.

We have developed a complete simulation architecture which consists of the environment generation and analysis, geographic information systems, environment displaying techniques, command and control systems, group and command hierarchy, radio messaging, group formation, group synchronization, stochastic perception gathering, physical modeling and real-time path planning.

A test environment is generated using high-resolution real data of a mountainous area. Automatic and semi-automatic techniques are developed for analyzing slope, defining surface materials. A user interface is provided to modify the environment properties and to add features on the terrain.

The defense and assault forces are generated in a group manner and a set of scenarios is generated to evaluate performance of developed algorithms: off-line path planning, real-time path update and real-time path search. It has been observed that planning an off-line path and updating real-time are a good choice if the partial updates are not significant, but if the mission plan changes sharply during the execution, real-time search algorithms serves more efficient solutions. Tests on real-time path search algorithms show that the proposed algorithm, Real-

Time Horizontal A*, makes remarkable improvements on time spent for reaching the goal state.

As a result of mentioned observations, we state that the real-time path planning techniques can be improved by increasing the visual depth, which helps a lot to escape earlier from the local semi-closed regions. But we have also noticed that there is much to do for better intelligent search strategies. More visual perception and evaluation techniques are needed to go one step forward.

The techniques presented in this thesis mainly attempt to solve the problem "where to go for reaching a goal state", but there is also an important question "how to behave while on the way", which is a reactive behavior problem. For example, you may decide to follow a generated path, but how an agent must behave on the way (for example, running, walking or waiting) may not be known. Future research on behavior concept will help much to increase the success of a mission.

Another important future research area is group coordination and corporation among the agents in order to complete a mission. Although this is a crucial need in computer-generated forces and military applications, we have seen that there is not much study in the area mainly because of its difficulty. So we have to focus on the domain and come up with new approaches.

In this thesis, we have also studied briefly the group formation, which is a coordination technique among the team members of a group. This study was only aiming at moving a group according to a given group formation, but there can be future research on how to decide and change a group formation on the fly.

REFERENCES

[A94] Anthony Stentz. *Optimal and Efficient Path Planning for Partially-Known Environments*. Proceedings of the IEEE International Conference on Robotics and Automation. May, 1994.

[A95] Anthony Stentz. *The Focussed D* Algorithm for Real-Time Replanning. Proceedings of the International Joint Conference on Artificial Intelligence*. August, 1995.

[A97] Alan Watt. *3D Computer Graphics*. Addison Wesley Publishing Company Inc. 1997.

[CM99] Charles E. Campbell and Michael A. Craft. *Advancements in Synthetic Natual Environment Representation*. Proceedings of 8th conference on Computer Gererated Forces and Behavioral Representation. Orlando, Florida, pp. 81-86, May 1999.

[CVZ00] Cagatay Undeger, Veysi Isler and Ziya Ipekkan. *An Intelligent Action Algorithm for Virtual Human Agents*. Proceedings of the 9th Conference on Computer Generated Forces and Behavioral Representation, Orlando, Florida. May 2000.

[DEUV96] Daniel Cohen-Or, Eran Rich, Uri Lerner and Victor Shenkar. *A Real-Time Photo-Realistic Visual Flythrough*. IEEE Transactions on Visualization and Computer Graphics, Vol. 2, No. 3. September 1996.

[E98] Erol Gelenbe. *Modelling CGF with Learning Stochastic Finite-State Machines*. Proceedings of 8th conference on Computer Gererated Forces and Behavioral Representation. Orlando, Florida, pp. 113-115. May 1999.

[EA98] Emanuele Trucco and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall, Inc. 1998.

[JG99] Jeremy W. Baxter and Graham S. Horn. *A Model for Co-ordination and Co-operation Between CGF Agents*. Proceedings of 8th conference on Computer Gererated Forces and Behavioral Representation. Orlando, Florida, pp. 101-111. May 1999.

[JJ98] James J. Kuffner, Jr. and Jean-Claude Latombe. Goal-Directed Navigation for Animated Characters Using Real-Time Path Planning and Control. Proceedings of CAPTECH '98: Workshop on Modelling and Motion Capture Techniques for Virtual Environments, Geneva, Switzerland, pp. 26-28, Nov 1998.

[JJ99] James J. Kuffner, Jr. and Jean-Claude Latombe. Fast Synthetic Vision, Memory, and Learning Models for Virtual Humans. Proceedings of Computer Animation, IEEE, pp. 118-127, May 1999.

[JP94] Jin Joe Lee and Paul A. Fishwick. Real-Time Simulation-Based Planning for Computer Generated Force Simulation. Simulation, pp. 299-315, 1994.

[KJ97] Kazuo KJ97 and John K. Smith. Genetic Algorithms for Adaptive Planning of Path and Trajectory of a Mobile Robot in 2D Terrains. Technical

Report, number ICS-TR-97-04, University of Hawaii, Department of Information and Computer Sciences, May 1997.

[M00] Mark A. DeLoura. *Game Programming Gems*. Charles River Media, Inc. 2000

[MY96] Makoto Yokoo (NTT Communication Science Laboratories) and Yasuhiko Kitamura. *Multiagent Real-Time-A* with Selection: Introducing Competition in Cooperative Search*. Proceedings of International Conference on Multi-Agent Systems (ICMAS-96), 409-416. 1996.

[NIMA99] National Imagery and Mapping Agency (NIMA). Digital Terrain Elevation Data (DTED). Standards and Specifications Publications: MIL-PRF-89020A Amendment-1, 27 April 1999.

[OPBRMP98] Oliver Deussen, Pat Hanrahan, Bernd Lintermann, Radomír Mech, Matt Pharr and Przemyslaw Prusinkiewicz. *Realistic Modeling and Rendering of Plant Ecosystems*. Proceedings of SIGGRAPH '98, Orlando, FL, USA. 1998.

[PS99] Paul T. Barham and Shirley M. Pratt. *The Development of High Level Architecture (HLA) Human Starter Simulation Object Model (SOM)*. Proceedings of 8th conference on Computer Gererated Forces and Behavioral Representation. Orlando, Florida, pp. 145-151. May 1999.

[RA98] Richard W. Pew and Anne S. Mavor. *Modeling Human and Organizational Behavior: Application to Military Simulations*. National Academy Press. 1998.

[RJMP94] Randolph M. Jones, John E. Laird, Milind Tambe and Paul S. Rosenbloom. *Generating Behavior in Response to Interacting Goals*. Proceedings

of 4th conference on Computer Gererated Forces and Behavioral Representation. Orlando, Florida. 1994.

[RM98] Rune M. Jensen and Manuela M. Veloso. *Interleaving Deliberative and Reactive Planning in Dynamic Multi-Agent Domains*. In Proceedings of the AAAI Fall Symposium on Integrated Planning for Autonomous Agent Architectures, AAAI Press. October 1998.

[RMJP93] Randolph M. Jones, Milind Tambe, John E. Laird and Paul S. Rosenbloom. *Intelligent Automated Agents for Flight Training Simulator*. Proceedings of 3th conference on Computer Gererated Forces and Behavioral Representation. Orlando, Florida, pp. 33-42. May 1993.

[SJ99] S.M. LaValle and J.J Kuffner. *Randomized Kinodynamic Planning*. In Proceedings of IEEE International Conference on Robotics and Automation (ICRA'99), Detroit, MI. May 1999.

[SP95] Stuart Russell and Peter Norving. *Artificial Intelligence: a modern approach*. Prentice Hall, Inc. 1995.

[USGS98] USGS National Mapping Information. Digital Elevation Models. Technical Instructions for Digital Elevation Model Standards, January 1998.