REAL-TIME CROWD SIMULATION IN VIRTUAL URBAN ENVIRONMENTS USING ADAPTIVE GRIDS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BİLKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Ateş Akaydın

July, 2010

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

Assoc. Prof. Dr. Uğur Güdükbay (Advisor)

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

Prof. Dr. Volkan Atalay

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

Asst. Prof. Dr. Selim Aksoy

Approved for the Institute of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Institute

ABSTRACT

REAL-TIME CROWD SIMULATION IN VIRTUAL URBAN ENVIRONMENTS USING ADAPTIVE GRIDS

Ateş Akaydın
M.S.in Computer Engineering
Supervisor: Assoc. Prof. Dr. Uğur Güdükbay
July, 2010

Crowd simulation is a relatively new research area, attracting increasing attention from both academia and industry. This thesis proposes *Adaptive Grids*, a novel hybrid approach for controlling the behavior of agents in a virtual crowd. In this approach, the motion of each agent within the crowd is planned considering both global and local path planning strategies. For global path planning, a cellular adaptive grid is constructed from a regular navigation map that represents the 2-D topology of the simulation terrain. A navigation graph with efficient size is then pre-computed from the adaptive grid for each possible agent goal. Finally, the navigation graph is used to generate a potential field on the adaptive grid by using the connectivity information of the irregular cells. Global path planning per agent has constant time complexity. For local path planning, Helbing Traffic-Flow model is used to avoid obstacles and agents. Potential forces are then applied on each agent considering the local and global decisions of the agent, while providing each agent the freedom to act independently.

Keywords: Crowd Simulation, Real-Time Animation, Motion Planning, Urban Visualization.

ÖZET

UYARLANABİLİR IZGARALARDAN YARARLANARAK SANAL KENTSEL ORTAMLARDA GERÇEK-ZAMANLI KALABALIK SİMÜLASYONU

Ateş Akaydın
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Uğur Güdükbay
Temmuz, 2010

Kalabalık simülasyonları, akademik ve endüstriyel çevrelerden gördüğü ilgi giderek artmakta olan nispeten yeni bir araştırma alanıdır. Bu tez, sanal insan kalabalıkları içinde bulunan bireylerin davranışlarının modellenmesi ve kontrolü için özgün ve hibrid bir yaklaşım olan *Uyarlanabilir Izgaralar* yaklaşımını önermektedir. Bu yaklaşımda kalabalığa dahil olan her bireyin hareketi yerel ve genel yol planlaması stratejileri kullanılarak planlanır. Genel yol planlaması için simülasyonun gerçekleştiği arazinin 2-boyutlu topoloji bilgisinden yararlanılarak düzenli bir navigasyon haritası çıkartılır. Bu düzenli navigasyon haritası düzensiz, uyarlanabilir hücrelerden oluşacak şekilde parsellenir ve bu hücreleri barındıran uyarlanabilir bir ızgara elde edilir. Uyarlanabilir hücrelerin topoloji bilgisi kullanılarak bir navigasyon grafiği hazırlanır. Bu grafik üzerinden kalabalık içerisindeki bireylerin olası her hedef noktası için bir potansiyel alan oluşturulur. Oluşturulan potansiyel alanlar sabit zamanlı erişim için kaydedilir ve bireylerin genel hareket planlamasında kullanılır. Yerel hareket planlaması için ise Helbing Trafik-Akış modelinden yararlanılmıştır. Yerel hareket planlaması sayesinde bireyler komşuluklarındaki engellerden ve diğer bireylerden kaçınabilirler. Yerel ve genel hareket modeli sonucu oluşan potansiyel kuvvetler toplanarak bireylerin hareketi sağlanır. Bu yaklaşım bireylerin herhangi bir sınırlamadan bağımsız olarak davranış sergilemesine imkan verir.

Anahtar sözcükler: Kalabalık Simülasyonu, Gerçek Zamanlı Animasyon, Hareket Planlaması, Kentsel Alanların Görüntülenmesi.

Acknowledgement

I would like to thank Assoc. Prof. Dr. Uğur Güdükbay for his invaluable guidance throughout my research.

I am grateful to Prof. Dr. Volkan Atalay and Asst. Prof. Dr. Selim Aksoy for evaluating this thesis.

I would also like to thank my parents for their unconditional love and continuous support and Miray Kaş for her valuable assistance throughout my research.

Finally, I would like to thank The Scientific and Technological Research Council of Turkey (TÜBİTAK) for providing me financial support for my M.S. studies. This work is supported by TÜBİTAK with grant number EEE-AG 104E029.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Scope

Real-time path planning for virtual crowds in dynamic environments is a challenging problem that has many applications. Virtual crowd simulation has applications in emergency evacuation, architecture design, urban planning, personnel training, education and entertainment. In robotics, the same problem occurs in multiple-robot coordination and planning in dynamically changing physical environments.

In both video gaming and film industry there is an increasing demand for massive crowd simulations. For interactive systems such as video games computational complexity of such crowd simulation systems still form the main bottleneck. For non-real time systems it is still important to reduce production times of off-line animations including massive crowds.

Traditionally, in existing crowd simulation systems, the path planning behavior of the agents are realized with two complementary approaches: *(i)* local path planning and *(ii)* global path planning. Local path planning refers to the immediate decisions made by virtual agents to avoid contact with the neighboring entities such as other agents and obstacles. The second approach is the global path planning, which represents the global, long-term decisions agents make to satisfy their goals. This long term goal usually includes reaching to a specific position in the simulation space.

In the current literature, both global and local path planning are usually costly. Often, global path planning either entails the calculation of some form of potential field to guide the agents towards their goals (local minima) or the implementation of graph-based search methods (e.g., A^* Search) over a graph which captures the connectivity of the underlying simulation space.

This thesis proposes *Adaptive Grids*; a novel approach to simulate large crowds. The approach focuses on crowd simulation in virtual urban environments; especially aiming to address the challenges centered primarily around the trade-offs between physical correctness, realism and computational complexity. The main idea behind this work is to reduce complexity of global path planning to constant time.

1.2 Contributions

The main contributions of this thesis can be summarized as follows:

- The concept of *per-region adaptive grids* is introduced and a *grid adaptation* algorithm is proposed to perform traditionally-costly global path planning in $O(1)$ time, while achieving real-time performance.
- Unlike many potential field approaches the proposed approach is not unrealistically perfect. An agent is not expected to know everything and does not have a priori information about static and/or dynamic obstacles it will encounter along its path.
- Unlike many common systems, the proposed approach does not restrict the behaviors of the agents. Agents are not bound by crowd group decisions and they are allowed to make individual path planning.

1.3 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 briefly reviews previously proposed approaches for crowd simulation along with their strengths and weaknesses. Chapter 3 explains the proposed approach to path planning for crowd simulation. Chapter 4 describes the evaluation metrics and parameters that are used to generate different types of adaptive grids. These metrics and parameters are then discussed in further detail in order to find a suitable, optimized configuration for the adaptive grid generation algorithm. Chapter 5 provides statistical and empirical results obtained by simulations with different configurations and interprets the results. Chapter 6 concludes by discussing key results and future work.

Chapter 2

Background and Related Work

In this chapter, we first qualitatively evaluate the Continuum Crowds approach proposed by Treuille et al. [26] along with its key points, advantages and disadvantages. Continuum Crowds is discussed in detail as it is the work that inspired us to propose our *Adaptive Grid* technique. The rest of the section provides a brief survey of other crowd simulation related proposals available in the current literature.

2.1 Evaluation of Continuum Crowds

Continuum crowds approach performs crowd simulation using *potential fields* calculated over regular grids covering the whole terrain of the virtual city environment. In this approach, the local path planning (*i.e.,* the interactions of virtual agents with the surrounding environment and other neighboring agents) and global path planning (*i.e.,* making progress towards long-term goal points) are entangled together. The calculation of potential fields encompasses the crowd density in unit area, the length of a chosen path as well as the time it would take a virtual agent to follow that path. An agent moves on an optimal path following the negative gradient vector located at its current point. In this way, the agent not only does get closer to the ultimate destination (goal) point of its group but also dodges its neighboring agents.

Continuum crowds approach has a couple of note-worthy advantages; it is easy to implement and suitable for large crowd simulations while providing optimal paths. However, these advantages come at the cost of the following disadvantages, which we try to address in this thesis:

1. *The algorithmic complexity per group is quite high.* In continuum crowds approach, the complexity of generating a path for agents on a per-group basis is $O(n^2 logn)$, where N is the total number of cells in the grid. While continuum crowds approach can provide real-time performance for simulations using low-resolution grids and a small number of groups, using a finer-grained resolution for the grid or an increase in the number of simulated crowd groups lead to an exponential increase in the processing time, which might result in the loss of real-time performance guarantees. It can be concluded that continuum crowds approach becomes extremely costly for a realistic simulation of large human crowds.

2. *The agents are not allowed to have individual goal points.* Since the potential field calculation is computationally intensive, to ensure real-time performance, the potential fields can only be calculated on a per-group basis, forcing the individual agents in a group to share a common goal point. In this case, it is not possible for individual agents to have independent goal points and decisions. All agents in a group end up following similar paths, resulting in piling of agents at narrow/crowded exits, which in turn causes loss of naturalness.

3. *Congestions may disturb the behavior of the crowd.* While the groups are on their way to their goal points congestion can occur at local maxima in the potential fields. Congestions are typically observed around narrow exits when the number of agents close to that exit is high. Since global and local calculations are entangled in continuum crowds, this congestion situation affects the behaviors of all agents in the system. This is an undesired situation as it would have been more realistic when only the agents close to the congested point were affected. At a congestion, instant agents whose

goal points are located within the congested area, got permanently trapped and start trembling. An intuitive solution to this problem would be to increase the number of cells in the grid. However, since the complexity of potential field calculation is proportional to the square of the number of cells in the grid, the complexity grows drastically, leading this solution to be infeasible.

4. *Although the obtained path is optimal, it is not natural.* Another major drawback of continuum crowds approach is the calculation of optimal path for each agent on the potential field. Although this might appear as an advantage at a first glance, it is a problem in terms of the reality of the simulation. The virtual agents, which are expected not to know all possible paths and their current states in reality, happen to know all optimal paths. In other words, while calculating the optimal path it should follow, an agents can account for the actions of all other agents in the system whether or not those other agents are within its field of vision. The solution to this problem is twofold: *(i)* the local path planning should be handled separately from global path planning, and *(ii)* in local path planning, only the interaction between the agent and the close-by agents/objects should be considered.

2.2 Other Related Work

A great majority of crowd simulation approaches derive from Helbing's Social Force Model [8] which applies tangential, repulsion and attraction forces to simulate interactions between individuals and other obstacles. Social Force Model is quite similar to a particle system behaving under the influence of physical forces. Beginning with Helbing's Model, particle systems and dynamics approaches are widely adopted to simulate motion of virtual crowds. Hodgins and Brogan have used a dynamics approach to simulate behavior of synthetic animal flocks with significant physics [10]. Braun and Musse have generalized Helbing's model to account for individual behaviors of the agents during an evacuation scenario [3].

There have been numerous extensions of the Helbing's model like Helbing-Molnar-Farkas-Vicsek social force model [11] and self-organized pedestrian crowd dynamics model [7]. A more recent and popular approach based on continuum dynamics, *Continuum Crowds*, is proposed by Treuile and Cooper [26]. Their approach basically evaluates a potential field over the simulation grid at each frame to direct the agents. This approach has significant impact on the motivation and development of this thesis. Thalmann et al. further extended *Continuum Crowds* by integrating a navigation graph that is composed of circular search nodes for global path planning [16]. In this approach, potential fields of *Continuum Crowds* model are used for computing within node paths only.

One other alternative approach is cellular automata. These models discretize the underlying simulation space as a flow grid. Meta-data which can be used for decision making or routing for the agents is associated with each cell on the flow grid. Graph search algorithms may be used to search for paths towards goal points on the flow grid. Cellular Automata approach does not involve agent interactions. A flow on a particular grid cell is only available if the target cell it leads to is not occupied by other agents. The work of Chenney [4] and Loscos et al. [13] are good examples of approaches using cellular automata.

Another, conceptually different but similar approach to crowd simulation is to use behavioral (rule-based) models which are derived first by Reynolds [22, 23]. Behavior models are more individual centric and they define a set of rules to steer the individuals with the crowd they belong to. In this approach individuals are considered as simple state machines which transitions in between different rules depending on different environmental conditions. It is common practice in behavior models that collisions are either ignored; or detected and avoided completely. Most rule-based approaches do not model physical interactions due to collisions but rather they apply simple wait rules to avoid collisions all together. Rule-based models can be very realistic for low density crowds but they are not physically accurate. It is also much easier to use a rule-based approach in combination with cognitive models. Terzopoulos and Shao proposed such an extended rule-based model, which includes motor, perceptual, cognitive and behavioral aspects of the individuals altogether [25]

Finally there are some hybrid studies which incorporate the best parts of the above three approaches. Pelechano et al. proposed High Density Autonomous Crowds (HiDAC) [20] which uses a social force model to plan local behavior of the pedestrians via tangential, repulsion and attraction forces. Social Force approaches in general suffer from shaking especially for high density crowds due to large numbers of interaction forces. To address this problem HiDAC also uses a behavioral model to set stop/wait behavior for pedestrians in crowded narrow passes.

For global path planning, in almost all cases, some sort of high level representation of the simulation environment is required to support interactive simulations. Most common techniques are portal graphs [12, 21], roadmaps [2] and potential fields [4].

Chapter 3

The Proposed Approach: Adaptive Grids

In this chapter, we explain the details of the proposed approach, *Adaptive Grids*. The motivation of the proposed approach is to reduce global path planning cost per agent to $O(1)$ time complexity. Constant time per agent local planning approaches are already available in the current literature [4]. Therefore, the primary purpose of this thesis is to achieve linear time complexity for path planning of the entire virtual crowd.

In most crowd simulation approaches, a regular grid is defined on a sampled 2-D terrain space that includes information about agent neighborhood, terrain topology and meta-data (e.g. slopes, heights, and speeds) on a per-cell basis. This regular cell grid is commonly used to compute both global and local paths of agents. Different appraoches, such as cellular automata and social force models (e.g., [26]) use this regular grid directly. Some recent approaches, such as [16], attempt to create a hierarchical navigation graph from the regular grid to further reduce the global path planning costs for large regions.

Theoretically, constant time complexity for global path planning is easily achievable if the optimal paths from each cell to every other cell are pre-computed before beginning the simulation. This basically means that for each cell-pair, the optimal paths (and perhaps alternative paths) are known and stored. Practically, such an approach would be infeasible as the space complexity would be excessive. For large grids that are composed of thousands of cells at each dimension, the memory requirements would easily exceed even the amount that the current high-end hardware provides. Moreover, since the size of the domain computed by the preprocessing algorithm is very large, the preprocessing time becomes impractically large with the growing number of cells.

A potential method of improving such an approach would be to reduce search domain by grouping the regular cells with respect to their common properties. A navigation graph can then be imposed on the cell groups and the edge costs can be approximated. If these groups are convex and the regular cells included in a group are topologically connected then it is guaranteed that the linear paths from any point to some other point in such groups are free of obstacles. Therefore, global path vectors within convex groups are nothing more than the normalized Euclidean distance vectors for any taken subject point pairs. Such vectors can easily be generated in constant time, which also implies that path planning within a group can always be achieved in constant time. Hence, the problem boils down to performing constant time path planning between points defined in separate group pairs. A smart, optimized grouping algorithm maximizes convex group size, minimizes group count, best approximates path costs and covers the entire domain, would enable us to pre-compute paths for all pairs of groups.

The proposed *Adaptive Grids* approach is such a way of grouping the underlying regular grid in an attempt to make constant time path planning possible. The approach consists of five steps. In the first step we generate a boolean navigable grid from the city model. In step two, an adaptive grid is formed and its respective navigation graph is created. In the third step Dijkstra or Floyd Warshall algorithm is applied on the navigation grid to find shortest paths and to store them. Forth step includes the generation of the vector field to direct agents towards their global goals at constant time. In the last step, agents query the vector field at their position to determine their global paths. Steps 1-3 are explained in detail in Section 3.1.3.2. Step 4 is discussed in Section 3.1.3.3. Final step also incorporates local path planning and hence is described in Section 3.1.1.

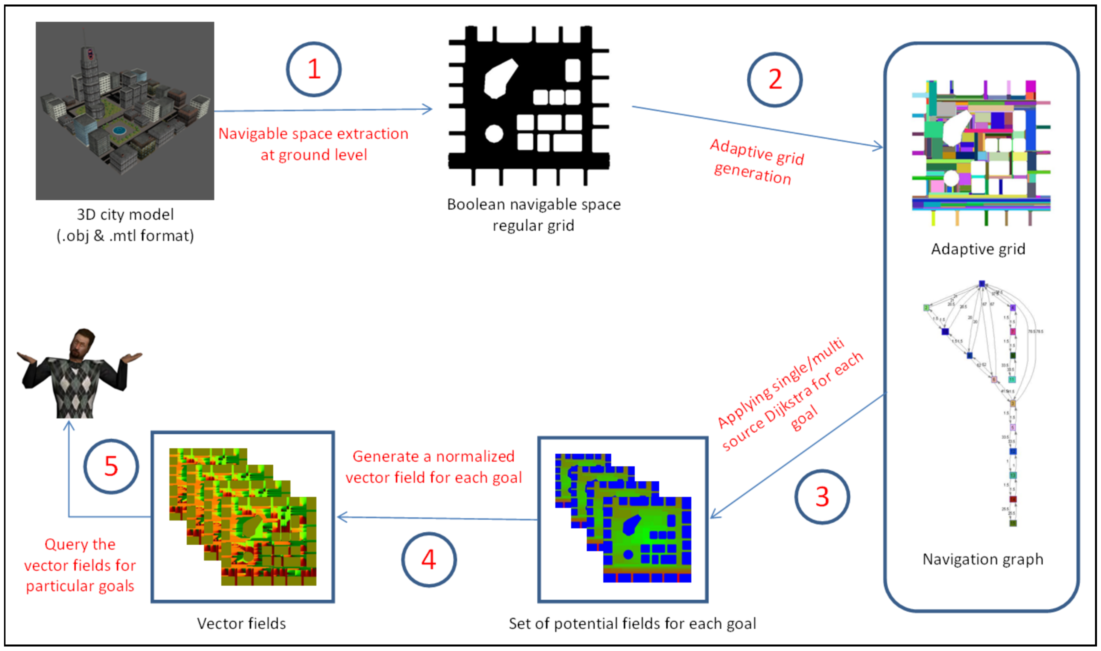

1
Navigable space extraction at ground level
2
Adaptive grid generation
3
Applying single/multi source Dijkstra for each goal
4
Generate a normalized vector field for each goal
5
Query the vector fields for particular goals
3D city model (.obj & .mtl format)
Boolean navigable space regular grid
Adaptive grid
Navigation graph
Set of potential fields for each goal
Vector fields

Figure 3.1: The flow diagram of the proposed approach.

Figure 3.1 depicts the flow of these steps in the proposed approach.

In the rest of this chapter, we explain local and global path planning in detail along with the integration of global and local path planning and the underlying social force model. The algorithms for the adaptation of the regular grid and the generation of the corresponding navigation graph are also provided.

3.1 Adaptive Grids

The proposed adaptive grids approach considers both global and local path planning methods. Social force model [9] is used as the base dynamics model that integrates both global and local path planning concepts.

To clarify, global path planning is used to plan paths to an individual's long term goals. For the sake of generality, these goals are always meant to be the

Table 3.1: Social force model parameters.

$v_i^0(t)$	Scalar value of desired velocity
$e_i^0(t)$	Desired direction vector, calculated via global path planning.
$v_i(t)$	Agent's current speed vector
τ_i	Unit time
m_i	Agent's mass
$\sum_{j(\neq i)} f_{ij}$	Sum of avoidance forces other agents apply on agent i
$\sum_w f_{iw}$	Sum of non-agent based avoidance forces applied on agent i

positions in the 3-D space at terrain level. On the other hand, local path planning is considered to be the short-term goals and maneuvers of the individuals due to immediate factors such as possible collisions. Immediate decisions can be made to avoid neighboring agent and obstacle collisions.

3.1.1 Social Force Model

Social force model describes the movements of pedestrians based on the concept of *social forces*. Social forces are not the forces that are directly enforced on the individuals (agents [1]). They rather stand for the motivational forces that motivate individuals to take certain actions. Social forces idea is built on an analogy between the particles in gas/fluid materials and the agents in crowds. The social forces model accounts for social and physical forces among neighboring agents as well as the direction the agent would like to go given its ultimate destination (*i.e.,* goal vector). It is based on Equation (3.1) (cf. Table 3.1):

$$m_i \frac{dv_i}{dt} = m_i \frac{v_i^0(t) e_i^0(t) - v_i(t)}{\tau_i} + \sum_{j(\neq i)} f_{ij} + \sum_w f_{iw} \tag{3.1}$$

The physical and social forces applied on an individual agent due to its interaction with *(i)* neighboring agents and *(ii)* the surrounding environment is handled in the local path planning part of our proposed mechanism while the desired direction vector of the agent is calculated via global path planning part of

[1]From this point onwards the terms individual and agent are used interchangeably.

Table 3.2: Parameters for local path planning.

$A_i exp[(r_{ij} - d_{ij})/B_i]n_{ij}$	Psychological repulsion force applied by agent j on agent i
r_{ij}	Sum of radius of agents i and j
d_{ij}	Distance between the centers of agents i and j
n_{ij}	Unit direction vector from j to i
A_i and B_i	Constants for psychological forces
$kg(r_{ij} - d_{ij})n_{ij}$	Reaction force on collision instant
$\kappa g(r_{ij} - d_{ij})\Delta v^t_{ji} t_{ij}$	Friction force on collision instant
$g(x)$	Constants for psychological forces
t_{ij}	Unit tangent vector between agents i and j
v^t_{ji}	The difference between the velocities of agents i and j in tangent vector direction
k	Friction constant
κ	Collision constant

our mechanism. Since the velocities of the agents are differentiable, sharp direction and speed changes are not allowed, resulting in smoother and more realistic movements.

Performing real-time simulation of large crowds requires the complexities of local and global path planning to be kept at minimum. Using Equation (3.1) as the basis, global path planning (*i.e.,* calculation of $e^0_i(t)$) can be performed independently from local path planning.

3.1.2 Local Path Planning

Local path planning encompasses the instant short term decisions made by the agent such as the manoeuvring to avoid collisions with other agents or surrounding objects. In social forces model [9], the local force applied on the agent due to its interaction with the other agents in the system is defined as in Equation (3.2) (cf. Table 3.2):

$$f_{ij} = \{A_i exp[(r_{ij} - d_{ij})/B_i] + kg(r_{ij} - d_{ij})\}n_{ij} + \kappa g(r_{ij} - d_{ij})\Delta v^t_{ji} t_{ij} \qquad (3.2)$$

Similar to the agent's interaction with other agents, its interaction with the objects in the surrounding environment is also defined as a part of local path planning (Equation (3.3)). Note that Equation (3.3) is the same as Equation 3.2); except that the force is now being applied by an object in the environment, rather than another agent in the system.

$$f_{iw} = \{A_i exp[(r_{iw} - d_{iw})/B_i] + kg(r_{iw} - d_{iw})\}n_{iw} + \kappa g(r_{iw} - d_{iw})\Delta v^t_{wi} t_{iw} \quad (3.3)$$

3.1.3 Global Path Planning via Adaptive Grids

We next discuss the ideas and notions that motivate us to devise *Adaptive Grids* followed by the details of navigable space extraction and adaptive grid formation.

3.1.3.1 Motivation for Adaptive Grids

Being able to perform global planning in $O(1)$ time is the key to achieve real-time crowd simulations. If a static potential field for each individual grid cell is calculated in advance and the gradient vectors associated with these grid cells are saved in the memory through pre-processing, then global path planning could easily be achieved in $O(1)$ time. However, the memory-cost of this naive technique is unacceptable as the number of cells is extremely high for a regular grid with enough resolution. Without loss of generality, assume that the resolution of a regular grid G is $n \times n$. The complexity of calculating all possible potential fields on G and saving them in the memory is as high as $O(n^4)$. Handling the problem within this perspective, at a first glance, it seems like performing $O(1)$ time global path planning with acceptable performance could only be achieved by reducing the number of cells in a grid.

In this thesis, *adaptive grids* is proposed as an alternative solution to this problem. In the preprocessing stage, rather than sampling the global path planning space using regular cells, we advocate the use of irregular and adjustable (*i.e.* adaptive) cells which reduces the number of cells significantly. In this case, performing global path planning on these preprocessed, static and adaptive grids significantly reduces the required $O(n^4)$ complexity.

3.1.3.2 Adaptive Grid Formation

To be able to perform global path planning, first of all, a grid composed of adaptive cells should be constructed. Such a grid can be constructed considering the topology of the virtual urban environment. The formation of adaptive grids involves two main steps:

1. The navigable space for the virtual environment should be extracted.
2. An adaptive grid should be built based on the extracted navigable space and the navigation graph should be constructed.

Step-1: Navigable Space Extraction

Z-buffer image (Figure 3.2) is taken from an axis-aligned top-view of the city. This image represents the height map of the out-door city environment. Objects that may obstruct this height map (e.g., trees, traffic lights, and banks) are excluded, leaving only the buildings and terrain.

In order to prevent noise, the z-buffer image is filtered by median-filter. Then, the values are normalized to [0,1] range, and passed through Sobel filters[6] in each direction (e.g., North, South, East, and West) as seen in Figure 3.3. Sobel-filtered images in each direction are converted to boolean images considering whether they are less than a certain threshold or not. These four Boolean images are then merged (using ‘or’ operator) into a single image, so that the non-navigable spaces in the city are bounded by rectangles (see Figure 3.4).

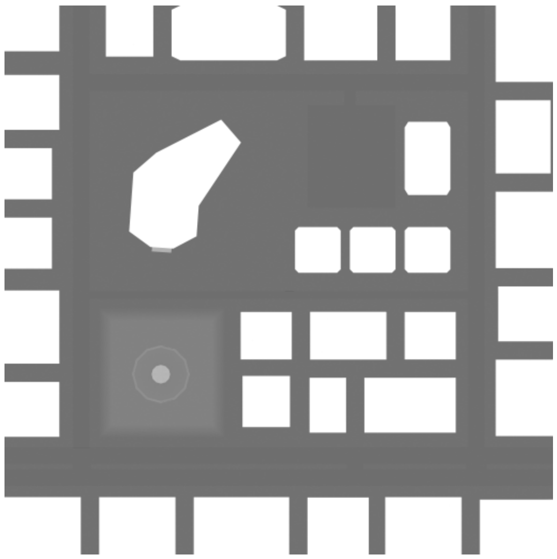
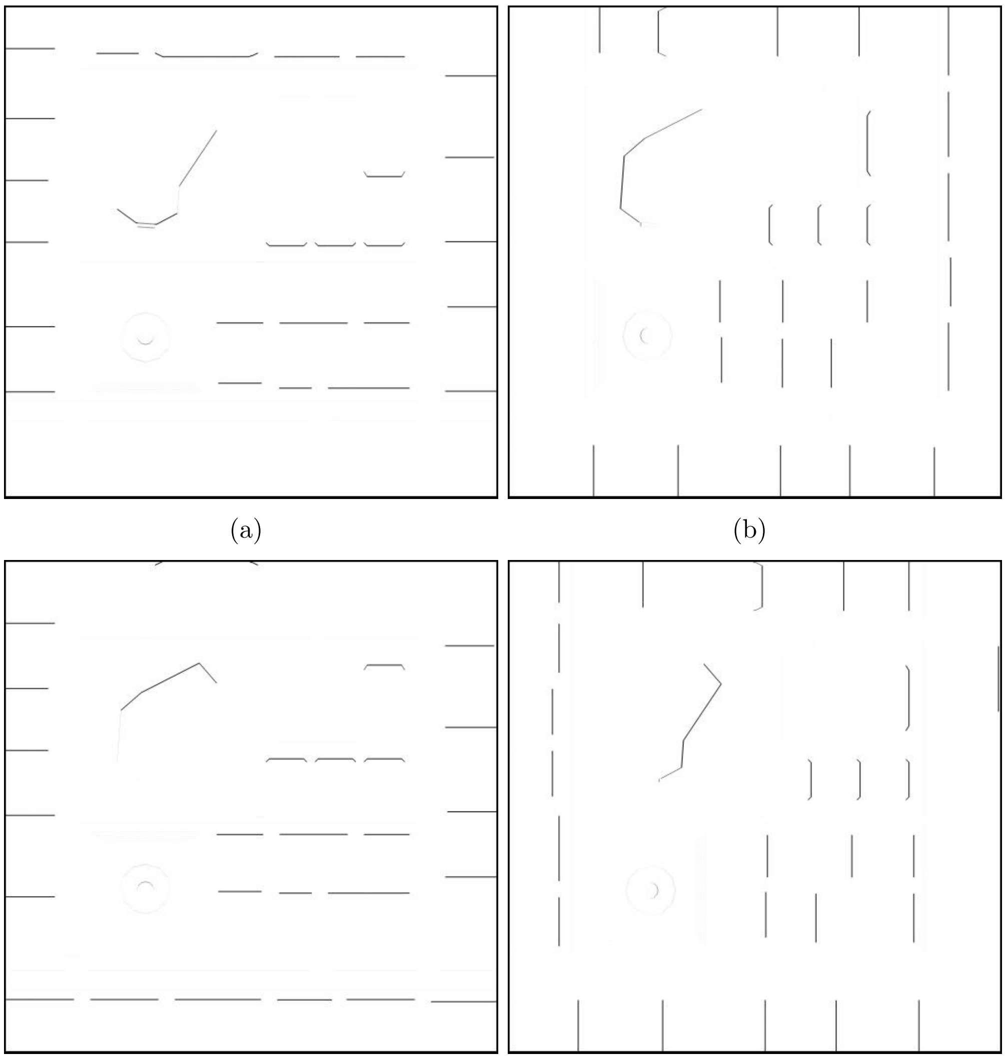


Figure 3.2: Z-Buffer image of the city model.



(a) (b) (c) (d)

Figure 3.3: Sobel filtering of the Z-buffer image; (a) North derivative; (b) East derivative; (b) South derivative; (d) West derivative.

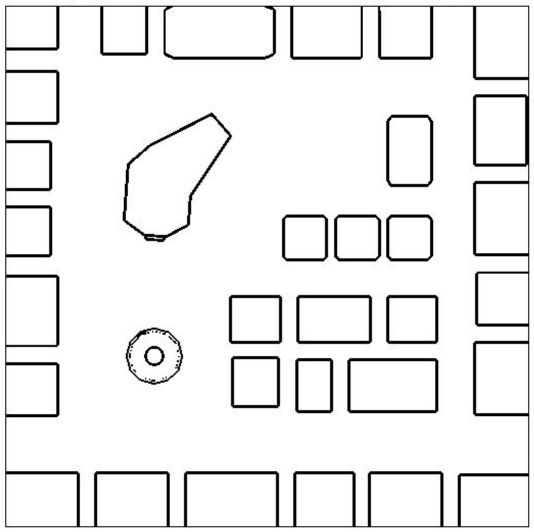


Figure 3.4: Merging of Sobel filtered images with an ‘or’ operator.

Performing connected component analysis[6] on this boolean image leads to segmentation of each separate rectangle. The segmented image is then converted to a Boolean image where the unnavigable rectangles are set to true and others are set to false. Finally the desired navigable space is extracted along with the topology information. In order to prevent the agents from getting too close to the buildings, morphological dilation process is applied on the navigable-space image using a small kernel. The final navigable-space image is given in Figure 3.5.

Step-2: Formation of the Adaptive Grid and Navigation Graph Construction

As seen in Figure 3.5, the navigable-space image includes detailed information regarding the topology of the virtual urban environment. Using this topology information, it is possible to construct the adaptive grid structure. Using one of the most well known 2D-space partitioning methods, *quadtree*, the number of cells for the navigable space presented in Figure 3.5 is 5953. Figure 3.6 depicts the image obtained using quadtree partitioning.

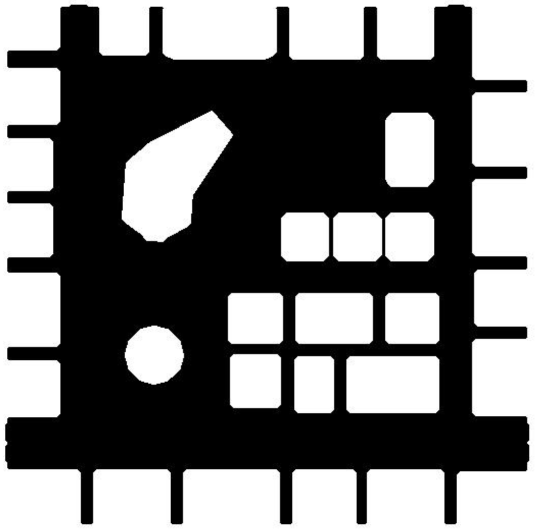


Figure 3.5: Connected component analysis and the formation of final navigable space image.

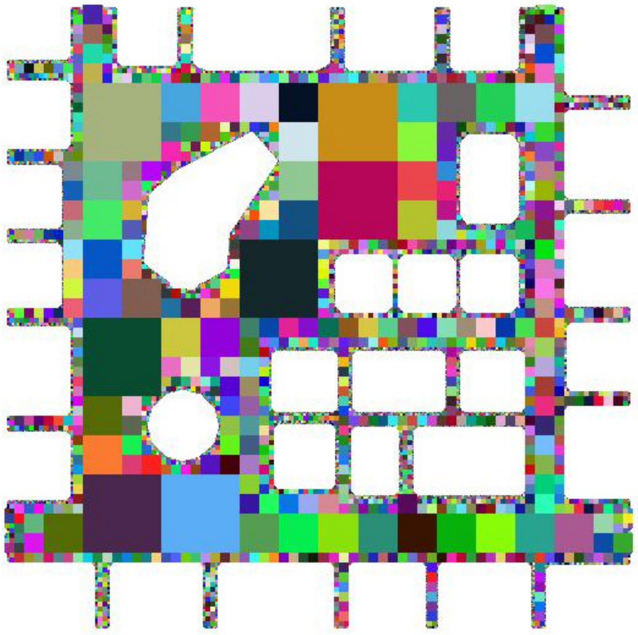


Figure 3.6: Quadtree partitioning of the navigable space.

For a navigable space of 512x512 resolution, the quadtree partitioning algorithm significantly reduces the number of seeds. However, to achieve O(1) time access, all of the cells should be indexed and the relevant direction vectors should be stored on each cell. For the *quadtree* case, there are around 35 million pairs and the space cost is close to 300 MB memory. Therefore, the *quadtree* approach is not feasible and the total number of seeds should be decreased more in order achieve feasible space complexity. Hence, an adaptive grid formation algorithm is proposed, which will further reduce the amount of seeds generated by the grid partitioning process.

Algorithm 1 describes the adaptive grid formation and navigation graph construction. In `line-2` of the algorithm, *genInitialSeeds* function is used to select a subset of adaptive cells from the navigable grid (*NavGrid*) and add them to *InitSeeds* data structure. The initial seeds act as starting points for the algorithm and they form the first set of seeds to be expanded. Initial seed selection can determine the end result of the algorithm in terms of seed size and seed count. Therefore, different initial seed selection methods are proposed and evaluated. These methods are described below:

1. *Single seed initialization:* A single initial seed is generated at the first available (navigable) cell on the regular grid. Tracing this regular cell is row-wise and starts at the top left cell of the regular grid.
2. *Random seed initialization:* A number of initial seeds are automatically generated at the navigable cells on the regular grid. The number of such seeds can be provided as a parameter.
3. *Sampled seed initialization:* The regular grid is sampled with the desired period. For each sample if the cell is navigable then a seed is initialized at this cell. The sampling period can be provided as a parameter.

In `line-9` of the algorithm, *expand* function is used to expand a single given seed. A seed can only expand to navigable and unoccupied cells. Expand function returns false if the given seed cannot be expanded possibly because it is

Algorithm 1: Adaptive Grid Formation and Navigation Graph Construction

```
Input: A regular boolean grid NavGrid that stores connectivity of the space.
Data: Queue object that stores seeds that are being evaluated. It can be
      configured to be a stack, queue, min heap or max heap (with respect to
      the seed size).
Data: InitSeeds is a set of initial seeds
Output: A directed seed graph SGraph=(V,E) where V is the set of vertices or
        seeds on adaptive grid and E is the set of edges connecting the
        vertices in V.
Output: A regular grid SGrid to store adaptive Seed-ID on the regular grid.
Result: Adapts the given regular connectivity grid NavGrid and stores
        adaptive seed information on regular grid SGrid and the relevant
        navigation graph in SGraph.
begin
    InitSeeds ← genInitialSeeds(NavGrid);          /* Choose initial seeds */
    foreach Seed s ∈ InitSeeds do
        Queue.push(s);                            /* Push seeds to the queue */
        SGrid.set(s.row, s.col, s.id);
        SGraph.addVertex(s);                      /* Add seeds to the graph */
    while Queue ≠ ∅ do
        s ← Queue.pop();
        if expand(s, SGrid, NavGrid) then                  /* Expand the seed */
            Queue.push(s);
        else               /* If cannot be expanded, generate new seeds */
            genSeedsFromSeed(s, SGrid, NavGrid, SGraph, Queue);
    foreach s ∈ SGraph.V do               /* Construct navigation graph */
        connectSeedToNeighbors(s, SGrid, SGraph);
```

surrounded by non-navigable cells or all cells towards the expansion direction are already occupied by other seeds. If a given seed cannot be expanded it is popped out of the *Queue* as it does not require any further processing. Otherwise, the seed is pushed back into the queue as it can be expanded further. The behavior of the expansion method can significantly affect the end-result of the algorithm again in terms of seed size, shape and count. Several different characteristics of the seed expansion methods are identified and provided below.

1. *Operating dimensions:* At a single call, a seed expansion method may expand along one or multiple dimensions. For example, a one dimensional

algorithm expand at each iteration in a single direction (east, west, north, or south), whereas a multi dimensional algorithm may expand towards all directions at once or towards intermediate directions.

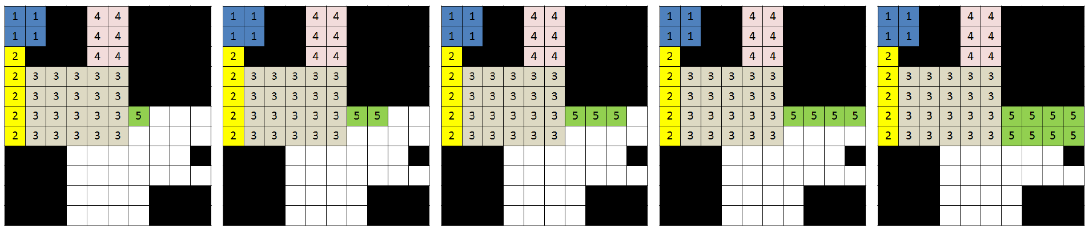
2. *Expansion ordering:* Expansion ordering specifies direction priorities. For instance, a clock-wise ordering may cause the algorithm to expand the seed along east, south, west, and north directions, respectively. The ordering may change the end-result and different orderings may lead to better results for different navigable grids.

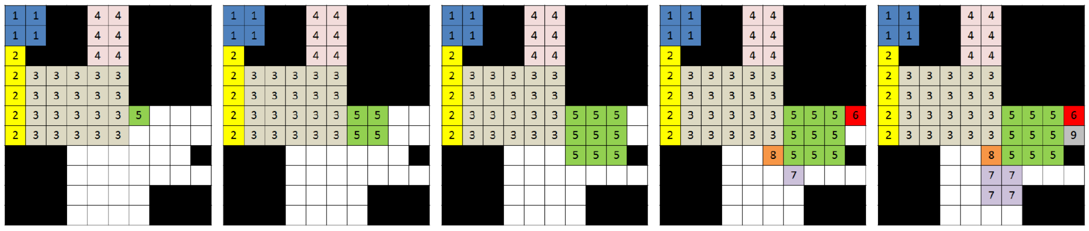
3. *Memory property:* A seed expansion method may remember the last direction it expanded to and continue from the successive directions in a breadth first manner. A memoryless method on the other hand will attempt to expand towards the same direction first at each try. Once that direction is unnavigable it will try the other directions in a depth first fashion with successive calls.

4. *Restrictiveness:* Seed size can be restricted, and hence, a seed expansion method can be forced to return false whenever the subject seed achieves a certain size. This property ensures that the seeds are restricted by a maximum size. Although larger seeds lead to less seed count it also increases the error made for the path costs on the navigation graph. Restricting seed size may keep that error in an acceptable threshold.

Figure 3.7 shows the behavior of three different expansion methods. The results are collected for running the algorithm for 5 iterations starting from the seed with *Seed-ID=5*.

For each seed that cannot be expanded further, new seed generation is necessary to continue the adaptive grid formation process. New seeds are generated on neighboring unoccupied and navigable cells on the adaptive grid (*SGrid*). The default algorithm (*genSeedsFromSeed*) takes a fully expanded seed and operates in clock-wise order, evaluating all cells within one cell proximity of the expanded seed excluding corners. Each unoccupied navigable cell, following a series of occupied or unnavigable cells, is marked. New seeds are generated on the marked


(a) Clock-wise memoryless 1D expansion


(b) Clock-wise memoryless 2D expansion

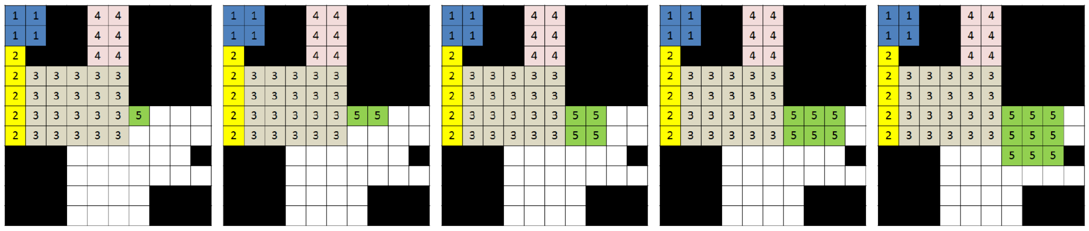

(c) Clock-wise breadth-first (with memory) 2D expansion

Figure 3.7: Seed expansions for five consecutive calls using different methods. Expansion starts from the seed with *Seed-ID=5*.

cells and added as a vertex to the seed graph and pushed to the seed queue as well to continue the adaptive grid formation process. The seed generation process is illustrated in Figures 3.8 and 3.7 (b).

When all of the seeds within the *Queue* are evaluated and all expansions are done, graph construction step begins (*connectSeedToNeighbors* in `line-14` of Algorithm 1). For each seed, a sweep is performed within one-cell proximity on the adaptive seed grid (*SGrid*) to determine neighboring seeds. For each neighboring pair, an edge is introduced with an attached cost.

The final part of the proposed algorithm (1) constructs the navigation graph. Edges are created between connected seeds and assigned a cost. Edge costs can be calculated using different methods as described below:

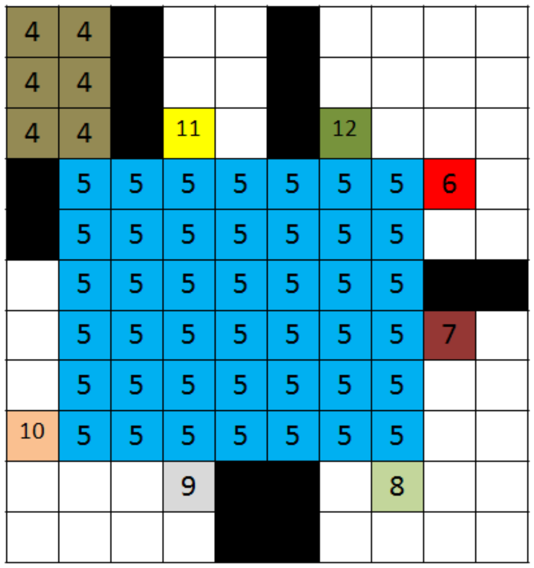


Figure 3.8: Seed generation starting from a fully expanded seed with seed id=5. Only the initially encountered free cells are marked during the clock-wise sweep.

1. *Manhattan distance:* The block distance, which is the sum of the absolute differences between x and y coordinates of the two seed centers, respectively.

2. *Euclidean distance:* The length of the line connecting two neighboring seed centers.

3. *Shortest Navigable Distance (SND):* Shortest navigable distance is the shortest distance connecting two seed centers which does not intersect with unnavigable cells. If the seed centers are in line of sight of each other then it is equal to euclidian distance. Otherwise, it is the sum of separate Euclidean distances between the seed centers and the shared unnavigable neighbor corners.

Figure 3.9 shows the Manhattan, Euclidean and Shortest Navigable Distance (SND) metrics for evaluating edge costs on the seed grid. Using one of these methods, edge costs are calculated for each seed pair by the adaptive grid formation algorithm and a navigation graph is formed. Examples of navigation graphs

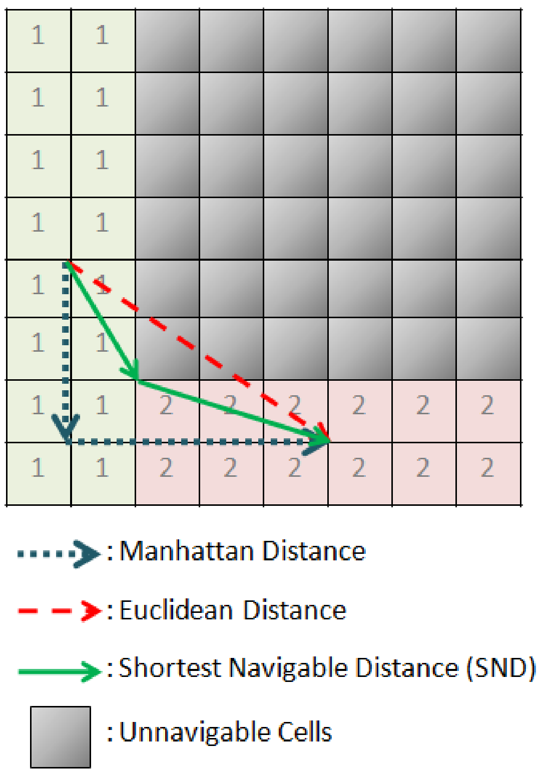


Figure 3.9: Different distance metrics for evaluating edge costs visualized on seed graph.

for a simple, restricted seed grid (Figure 3.10) are provided in Figure 3.11.

3.1.3.3 Extraction of the Vector Field

Having formed the navigation graph, it is necessary to find and store paths from each seed to every other seed on the seed grid. To this end, Floyd-Warshall Algorithm or Dijkstra Algorithm can be applied on the navigation graph for all seeds to compute all-pair shortest paths [5]. Seed-IDs are given in consecutive order starting from '1' on the grid; '0' denotes unnavigable cells. An array indexed by target Seed-IDs can store the immediate links lying on the shortest path from the current seed towards the target seed. At any time, a query for a particular target seed from a root seed can be answered with $O(1)$ time.

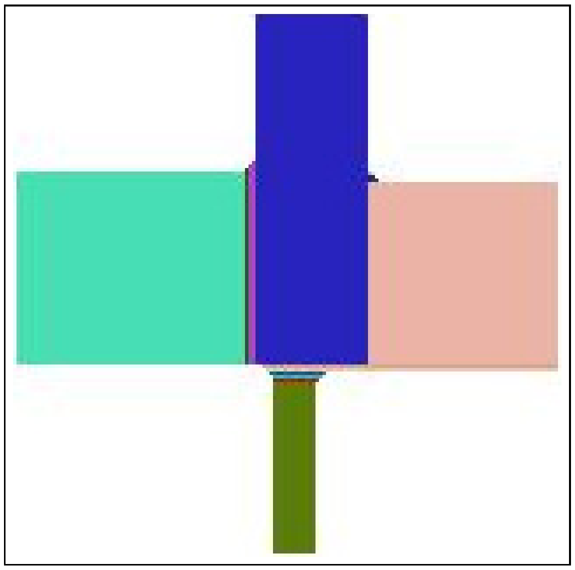


Figure 3.10: A simple seed grid instance.

With a careful selection of adaptive grid formation parameters (i.e., queue type, seed expansion method), it is possible to achieve acceptable error on path costs without wasting memory. In Sections 4 and 5, the error made on path costs is defined and comparable results are provided that are obtained with different parameters. The targets picked by the agents may not necessarily be seeds. They can be points contained by the seeds as well. Since all seeds are convex (rectangular) it is guaranteed that the internal paths within seeds are free of obstacles.

It should also be noted that the majority of people in real crowds choose from a very limited number global targets, such as important building entrances; thus, computing paths for all seed pairs is not necessary. If such target points within interest can be marked, the Dijkstra Algorithm can be run only for these interest points, generating all paths to these points. In this way, it is possible to further reduce memory requirements of the algorithm.

Having computed and stored all shortest paths, a vector field should be generated on all points within the seed grid in order to smoothly direct the agents towards their global goals. Such a kind of vector field may be generated with many different methods. The simplest one is to direct each agent towards the center of the common edge that is shared by the agent's current seed and the next seed, which leads to the agent's global target. For the sake of generality, such edges may be named as *gates* (or portals), which connect two neighboring seeds together.

A visualization of such a vector field is provided in Figure 3.12. This image is a two-channel image that contains only the red and green components. Hence, pure red values represents an upward vector and pure green represents a left vector. All vectors are normalized. The adaptive grid for this example is performed using a min-heap priority queue, border seed initialization and Shortest Navigable Distance (SND) metric for edge costs. A memoryless, one dimensional clock-wise seed expander is used. However, this approach would lead to lining of all agents who share a common goal as they progress through their path since they all aim for a single point at each seed.

A better approach would be to direct all agents that reside inside the gate's coverage towards gate's direction. Gate coverage defines the rectangular zone within the seed where moving towards the gate direction would be enough to exit the seed on the desired path. Agents that are out of the gate's coverage can pick the closest end-point of the gate as their primary target and move towards these points before exiting the gate. An example that shows the behavior of the gate coverage approach is shown in Figure 3.13. The corresponding vector field visualization is shown in Figure 3.14, which is generated with min-heap priority queue, single seed initialization and Shortest Navigable Distance (SND). A memoryless, one-dimensional clock-wise seed expander is used. Even though the vector flows are not evenly distributed with this technique, it is much better compared to the simple approach presented in Figure 3.12.

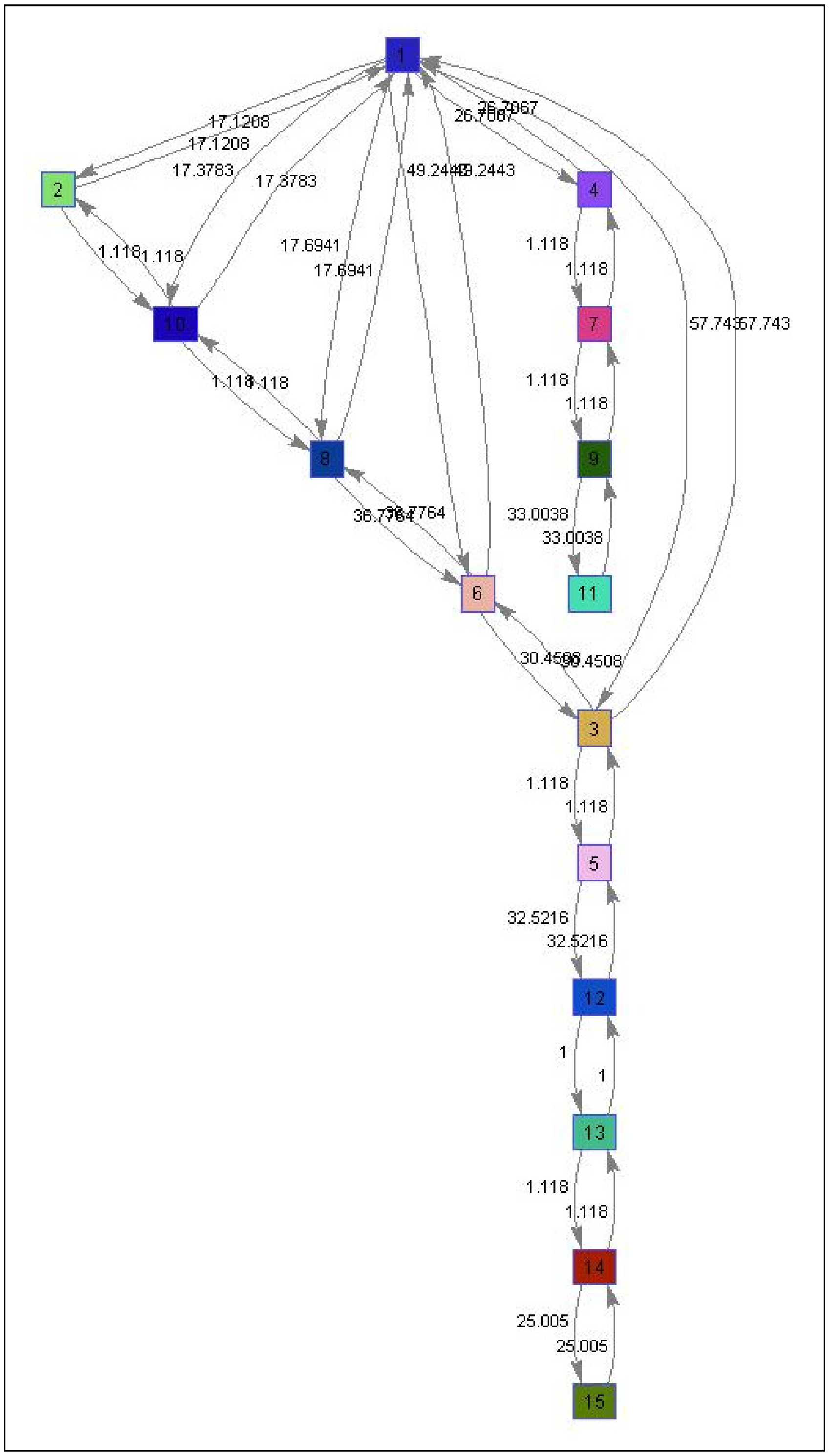


Figure 3.11: An example navigation graph generated with the Shortest Navigable Distance (SND) metric (based on seed grid in Figure 3.10).

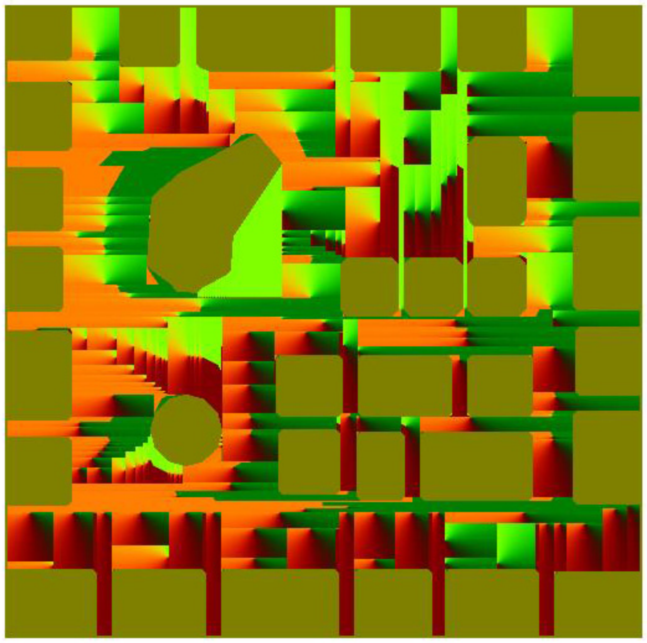


Figure 3.12: A simple two-dimensional vector field that samples directions towards the gate centers. The target point is the image center.

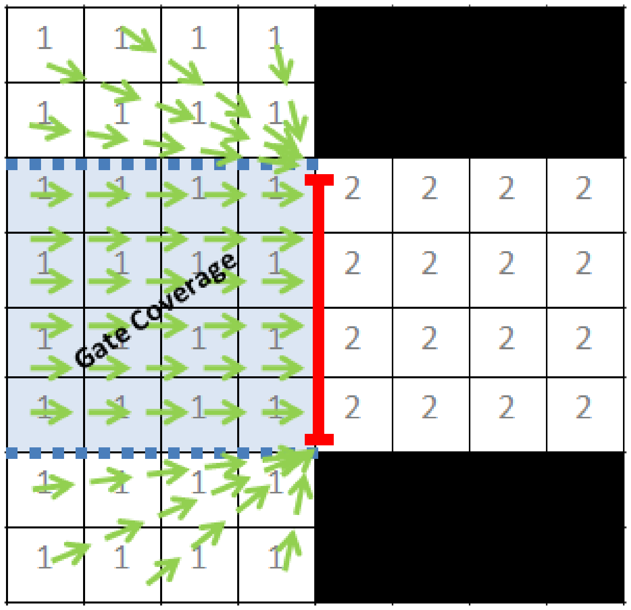


Figure 3.13: An example to show vector field formation by considering gate coverage.

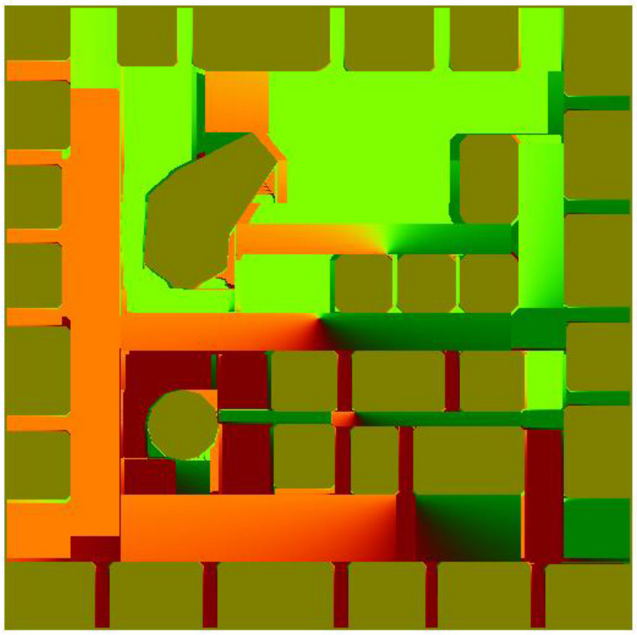


Figure 3.14: A two-dimensional vector field that samples directions within gate coverage. Points out of gate coverages are directed towards the closest point on the respective gate. The objective point is again the image center.

Chapter 4

Evaluation of the Adaptive Grid Formation

In this chapter, we first discuss a number of parameters that affect the outcome of the adaptive grid formation algorithm in detail. Then, the performance metrics that are used to evaluate the results of the algorithm are introduced.

4.1 Adaptive Grid Parameters

Adaptive grid formation parameters are summarized below:

Distance metric: This parameter determines the criterion that is used to measure the distance between two cells in an adaptive grid. Possible values of this parameter are *(i)* Manhattan distance, *(ii)* Euclidean distance, and *(iii)* Shortest navigable distance. Shortest navigable distance considers the navigable/non-navigable distance due to the buildings.

Data structure: This parameter determines the order in which the cells are considered during the adaptive grid formation process. We consider *(i)* queue, *(ii)* stack, *(iii)* min-heap, and *(iv)* max-heap as possible data structures.

For min-heap and max-heap data structures, the multiplication of the number of rows by the number of columns in a cell is used as the key associated with that cell.

Initial seed generation method: This method is invoked before the adaptive grid formation algorithm (see Algorithm 1). It determines the seeds (*i.e.,* the initial set of cells the adaptive grid formation algorithm visits when it is first initialized) and adds them to the *Queue* object, which is an input to the adaptive grid formation algorithm. Possible seed generation methods are as follows:

- *Single:* Starting from the top left corner, the navigable grid is swept in a *row-first* manner. A new seed covering a single cell is initialized every time a new navigable cell is encountered and added to the *Queue*.
- *Border:* On each non-initialized, navigable cell on the borders of the navigable grid (in each direction: east, west, south, north), a new seed is initialized and added to the queue.
- *Random:* The seeds that are added to the *Queue* are selected randomly from non-initialized, navigable cells. The terminating condition (*i.e.,* the number of seeds to be generated) is given as a parameter.
- *Sampled:* This method requires the sampling period to be given as a parameter. Given the sampling period, the cells whose locations (either row-wise or column-wise) are multiples of the sampling period are initialized as seeds and added to the *Queue*.
- *Seed expansion method:* This parameter determines the behavior of the seed expander which is explained in detail in Chapter 3. The seed expander may choose to expand towards one or two dimensions at a time. It can restrict the seed size and also it may choose to expand using a depth-first strategy (*i.e.,* memoryless) or a breadth-first strategy.

4.2 Evaluation Metrics

The grids obtained via using different values of the parameters discussed in Section 4.1 are evaluated using these metrics:

Seed count: The total number of seeds in the constructed adaptive grid. Using higher number of seeds reduces the accumulated error on path costs at the cost of exponentially-increasing memory and preprocessing time.

Average cell size: The size of a cell is calculated as the number of rows times the number of columns in that cell. This metric considers the average size of all cells in the constructed adaptive grid. Smaller average cell sizes usually indicate high number of seeds as well as reduced error on path costs.

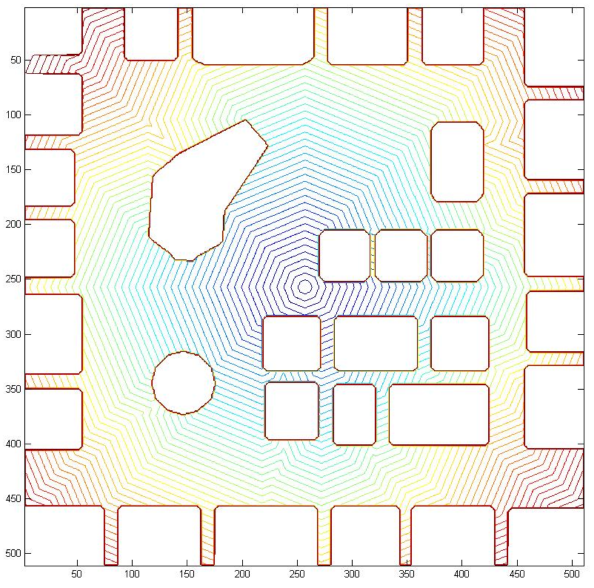
Average aspect ratio: The aspect ratio of a cell i is calculated as $max(row_i, column_i)/min(row_i, column_i)$ where row_i and $column_i$ represent the number of rows and columns in cell i, respectively. The average aspect ratio is calculated as the arithmetic mean of the aspect ratios of all cells. It is desired that this number is as close to 1 as possible since highly varying aspect ratios in general have a negative effect on the error introduced.

Mean Square Error (MSE): For a particular adaptive grid, MSE is approximated with respect to the difference in distances to the same location on both regular grid and adaptive grid. The MSE is calculated as follows:

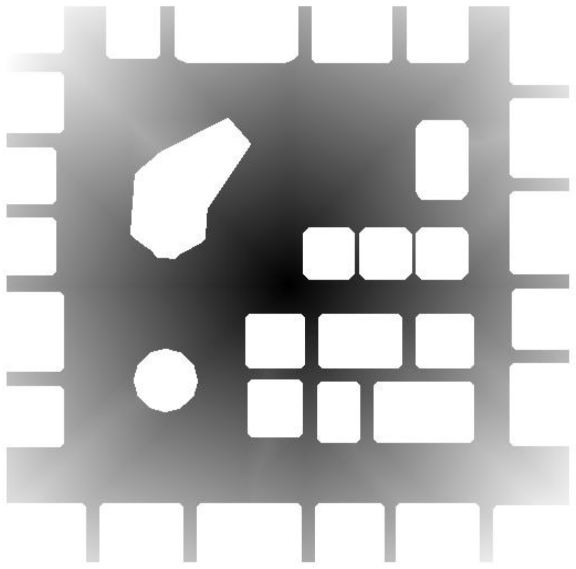
$$MSE(X,R) = \frac{1}{row_{size} \times col_{size}} A \tag{4.1}$$

$$A = \sum_{row=1}^{row_{size}} \sum_{col=1}^{col_{size}} (X(row,col) - R(row,col))^2, \tag{4.2}$$

In Eq (4.1), R represents the shortest distance of each pixel to the center of the adaptive grid (using the navigable area) while X represents the distances based on the navigation graph constructed on the adaptive grid. Figure 4.1 shows the pixel distances to the center of the navigation grid and Figure 4.2 presents the corresponding distances in an adaptive grid.

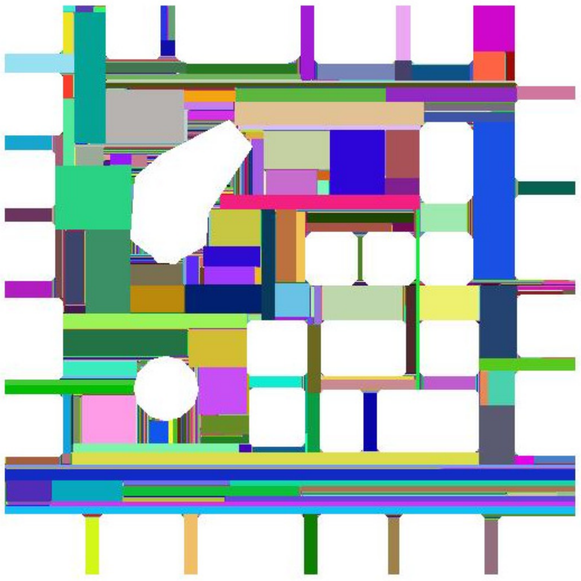


(a)

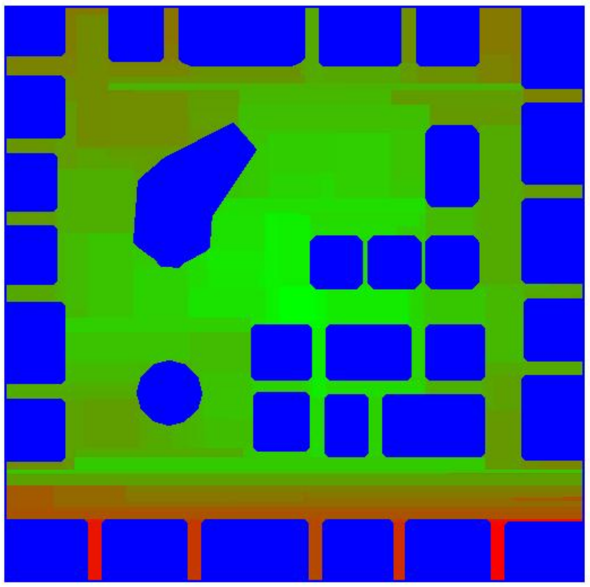


(b)

Figure 4.1: Pixel distances on the regular navigation grid. (a) Pixel distance contours; (b) Pixel distances using a gray scale image.



(a)



(b)

Figure 4.2: Pixel distances on the adaptive grid. (a) An adaptive grid formation with max-heap priority queue, 100 random initial seeds and shortest navigable distance (SND) distance metric. A memoryless 1d clock-wise seed expander is used. (b) Distances of each pixel to the center of the adaptive grid. Colors turn from green to red as the distance increases. Blue color denotes buildings.

Chapter 5

Results

This chapter summarizes the results of the proposed approach. In Section 5.1, we report statistics regarding the behavior of the proposed adaptive grid formation algorithm for a number of different parameter-value combinations. These statistics are presented along with a brief discussion about their impacts. In Section 5.2, results obtained from the realized crowd simulation application for different scenarios are given. Section 5.4 provides detailed information about the hardware configuration that is used to collect the final simulation results along with the development environment. Finally, in Section 5.3, some optimization techniques to support real-time simulation are briefly explained.

5.1 Forming Adaptive Grids

Figures 5.1, . . ., 5.6 present statistics for 132 different adaptive grids that are constructed with different combinations of parameters. These statistics compare the results obtained using different evaluation metrics given in Chapter 4.

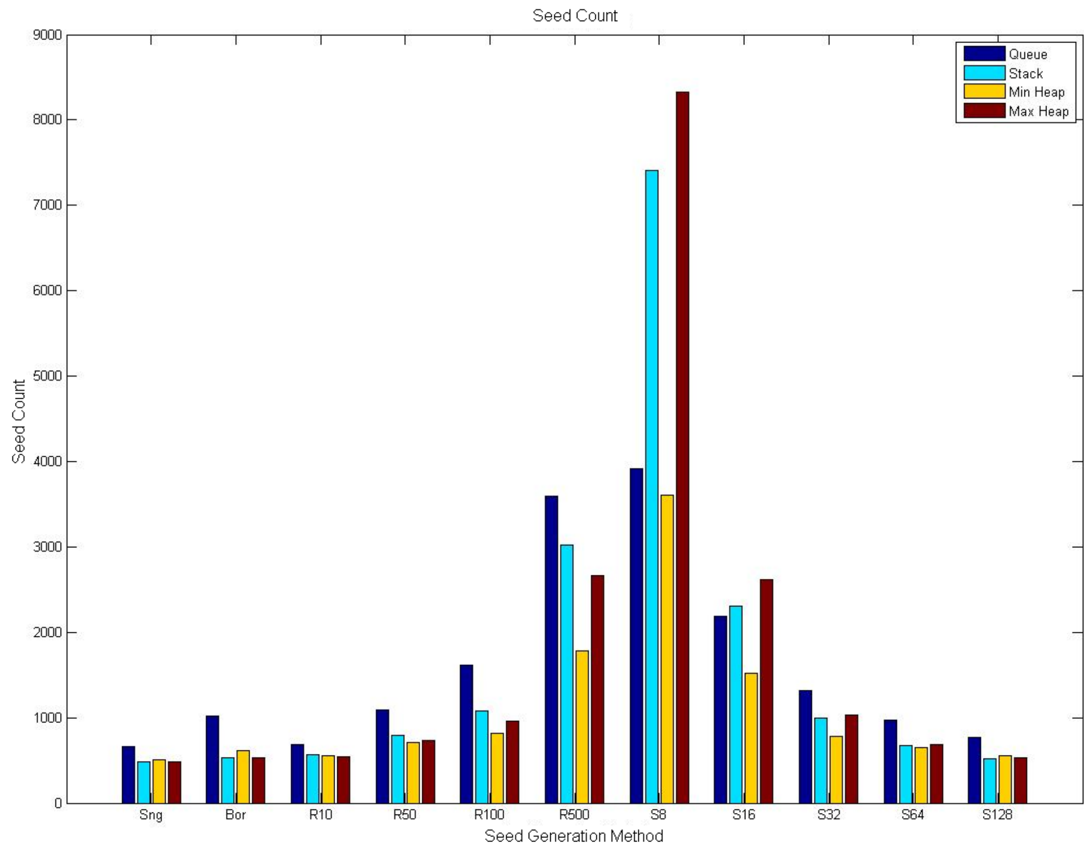


Figure 5.1: Seed counts for adaptive grids with different parameters. Seed initialization methods are Single (Sng); Border (Bor); Random 10 (R10), 50 (R50), 100 (R100), 500 (R500); Sampled 8 (S8), 16 (S16), 32 (S32), 64 (S64), 128 (S128), respectively.

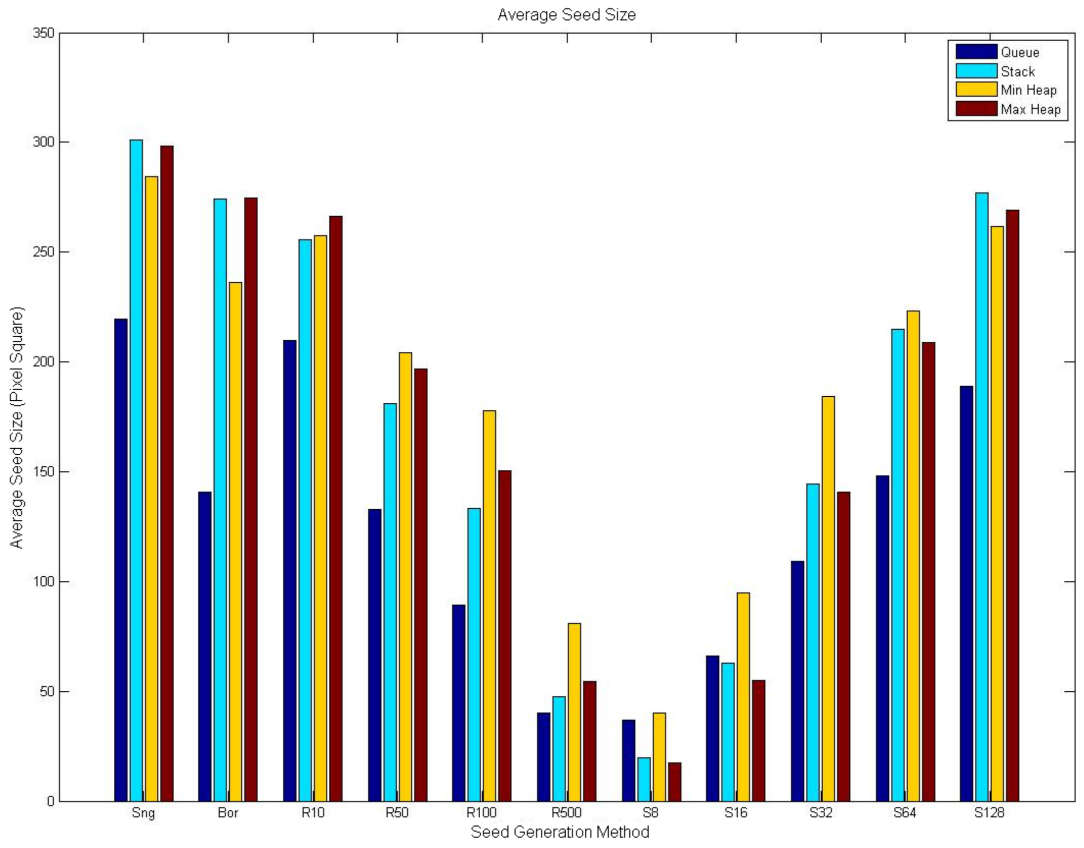


Figure 5.2: Seed sizes for adaptive grids with different parameters. Seed initialization methods are the same as those in Figure 5.1.

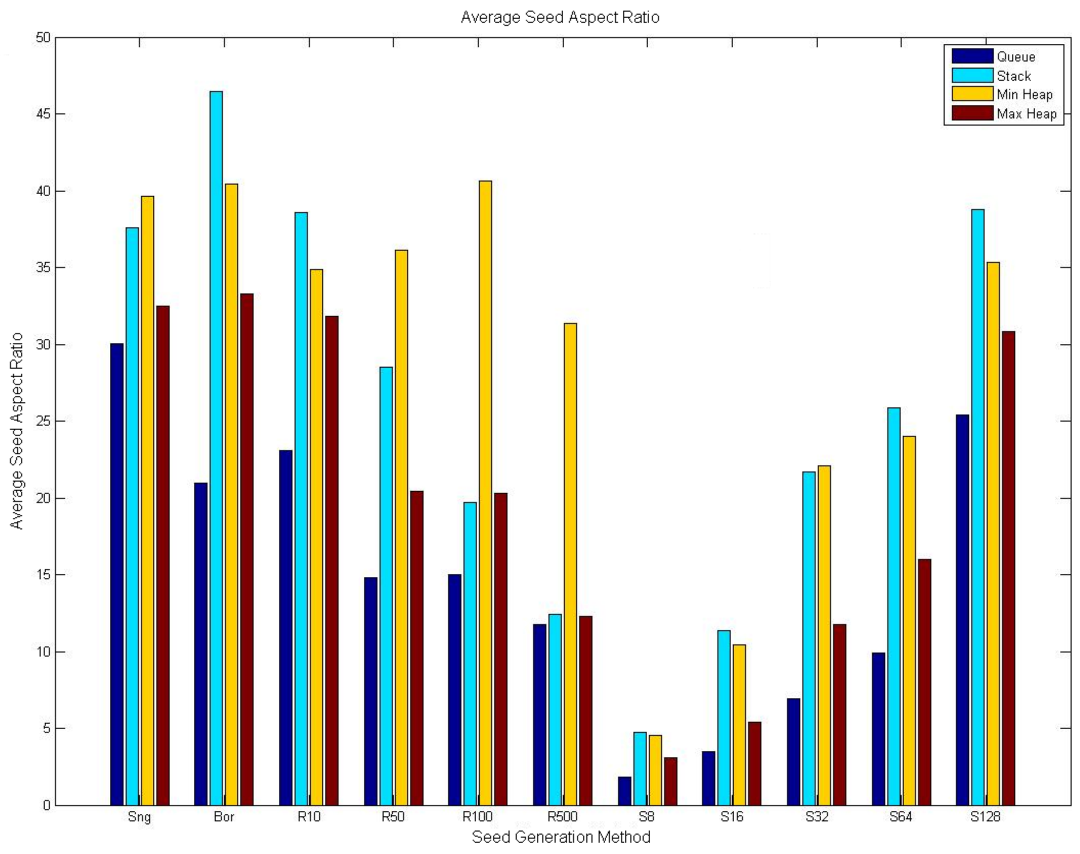


Figure 5.3: Seed aspect ratios for adaptive grids with different parameters. Seed initialization methods are the same as those in Figure 5.1.

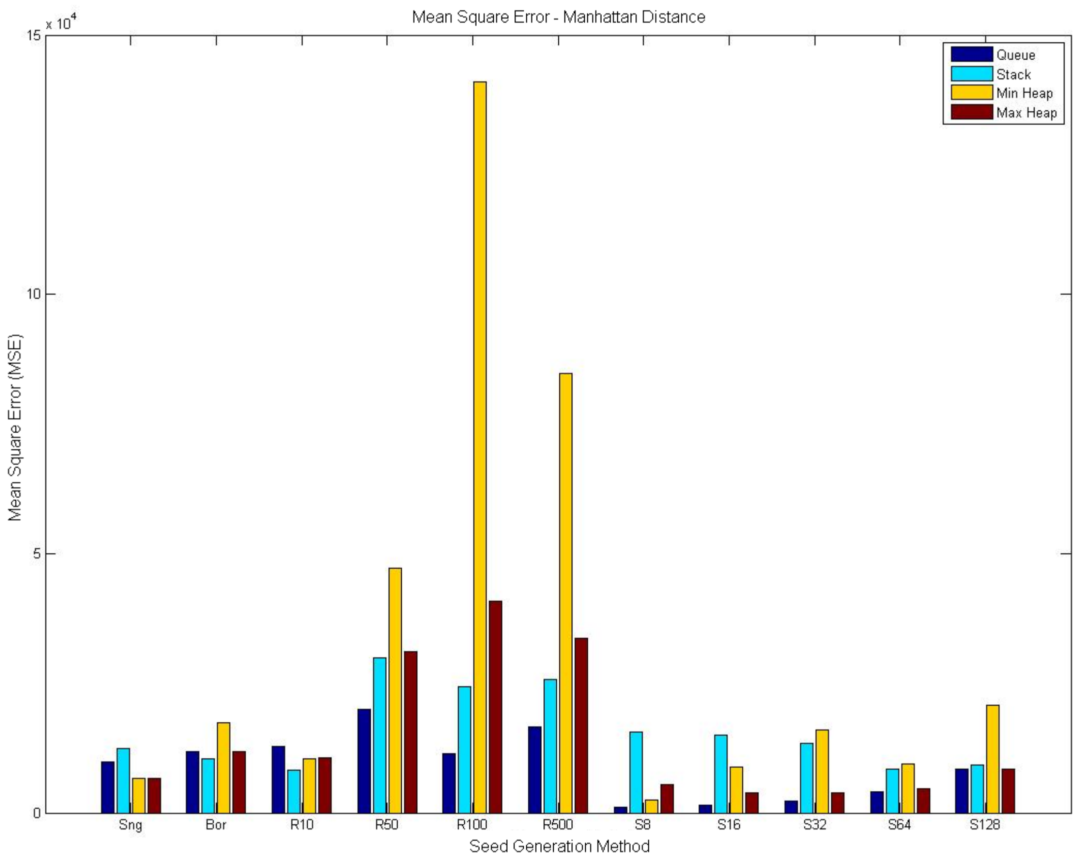


Figure 5.4: MSE for Manhattan distance metric for adaptive grids with different parameters. Seed initialization methods are the same as those in Figure 5.1.

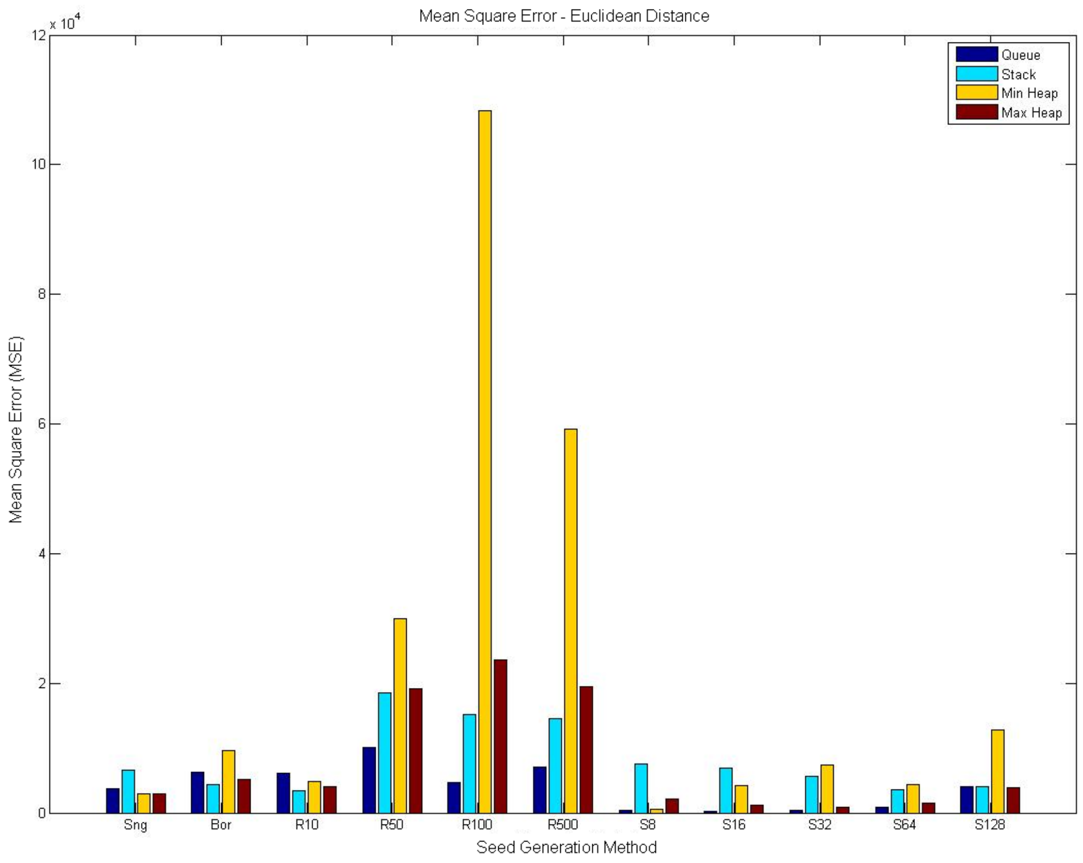


Figure 5.5: MSE for Euclidean distance metric for adaptive grids with different parameters. Seed initialization methods are the same as those in Figure 5.1.

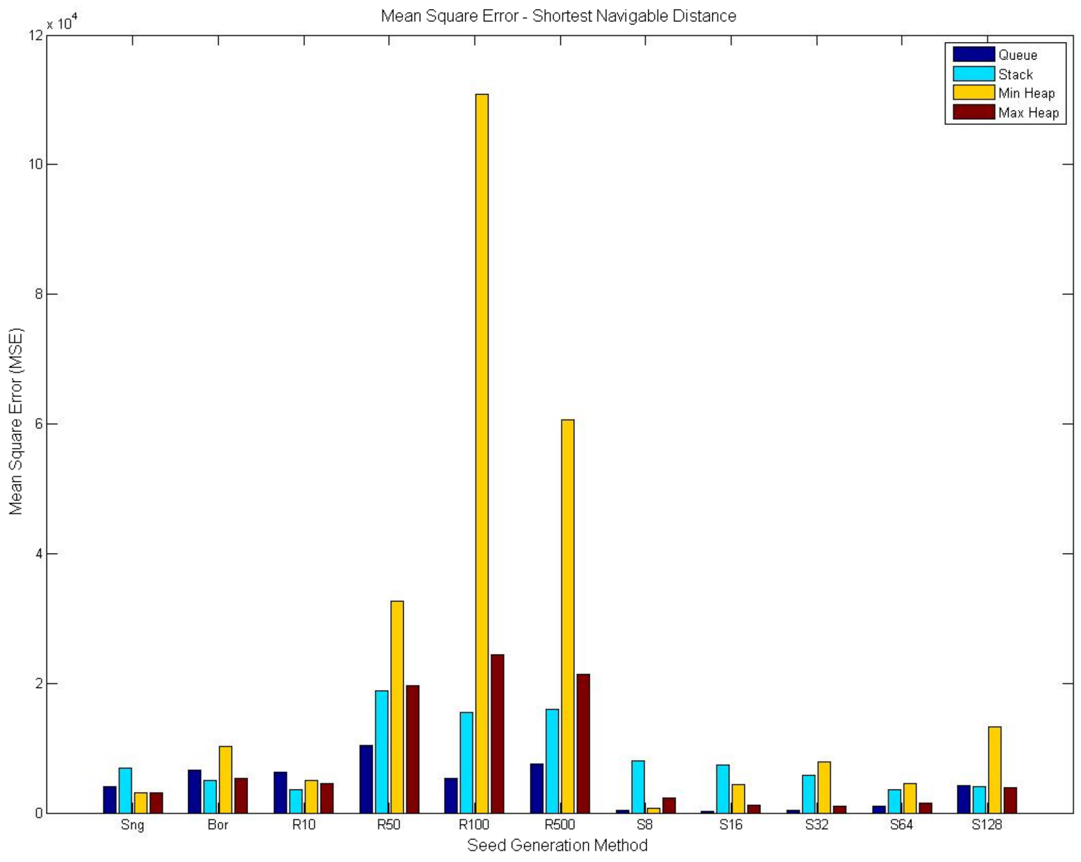


Figure 5.6: MSE for shortest navigable distance (SND) metric. Seed initialization methods are the same as those in Figure 5.1.

As it can be observed from Figures 5.1, . . ., 5.6, although an increase in the seed count notably reduces mean square error (*i.e.*, S8 and S16), it significantly increases the overhead in memory requirements and preprocessing time as well. For grid adaptations which has few amounts of seeds (*i.e.*, those with Single (Sng) and Border (Bor) initialization) and for those adaptations which has high amounts of seeds (like those Sampled 8 (S8) and Random 500 (R500) initialization) the mean square error (MSE) values are obtained lower than those which has medium amount of seeds (such as R100 and S128 initialization). Especially for the cells that have high and varying aspect ratios, the accumulated error on the paths grows as the number of seeds on the paths increase. However, an increase in seed count generally causes a decrease in seed size and restricts the aspect ratio; after some point, the accumulated error starts to decrease. Hence, the most successful grid adaptations are achieved with Random 100 (R100) and Sampled 128 (S128) grid initialization schemes and by using a common, round-robin queue.

5.2 Crowd Simulation Application

This section presents the results for the execution of simulation scenarios which include 500, 1000, 2000 and 4000 agents respectively. Agents are periodically created on the pre-specified entry points within the virtual city and are assigned global tasks to reach a random entry/exit point. These enty/exit points are assigned to the actual building entrances(doors) and street exits. Snapshots from 2000-agent scenario are illustrated in Figures 5.7, 5.8 and 5.9 for visualization purposes.

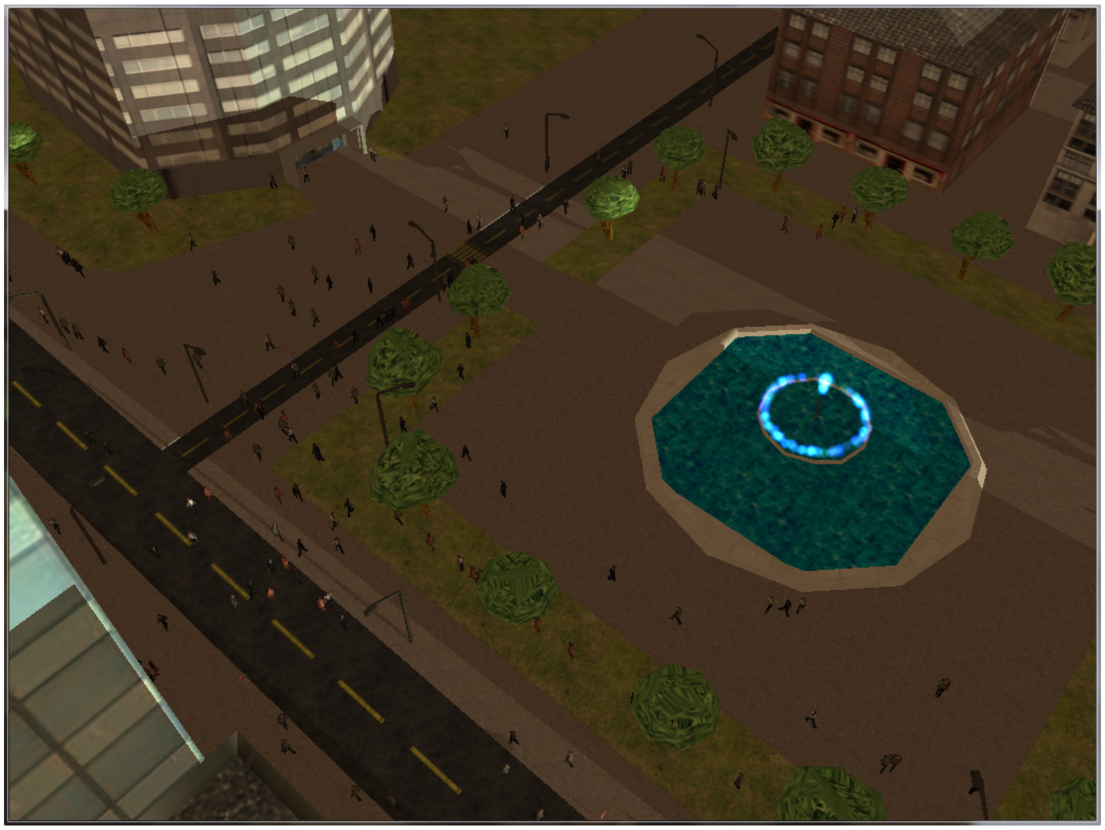


Figure 5.7: A snapshot from the 2000-agent scenario showing the top view of the park area.

Five different character models (three male and two female) with varying numbers of polygonal complexity (between 2000-4000) are randomly assigned to agents upon instantiation. All agents have a wide set of skeletal animations including idle, walking, running, talking, etc. for realization of more interactive scenarios. During animation switches, all animations are linearly blended together to support smooth transitions. Agents are removed from the simulation once they achieve their goals. As the successful agents get removed, the system maintains the maximum count of agents in the scene by injecting new agents periodically. The instances of removed agents do not get destroyed completely but rather inserted into a pool (queue) so that they can be reused for newer injections. This agent-pooling approach significantly reduces the computational overhead of the new agent instantiation process. In this way, all agents within the system are instantiated only once.

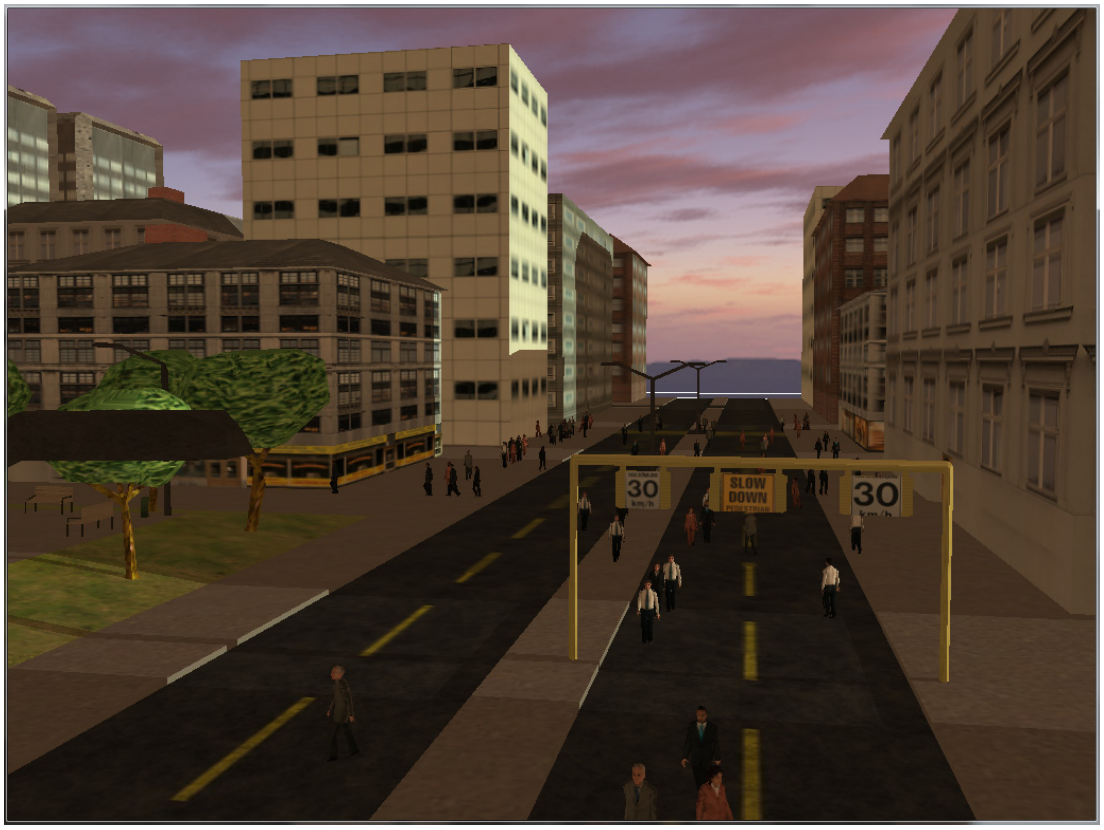


Figure 5.8: A snapshot from the 2000-agent scenario showing a view of the main street.

The virtual city block is composed of 33 buildings and a number of other environmental objects, such as city lights, traffic signs, trees, banks. These objects are also recognized as obstacles by the crowd. Hence, they are also taken into account during local path planning for collision avoidance. The complete geometric model of the virtual city includes 18342 polygons. To have a lively illustration of the city, a number of special effects such as particle systems for the fountain within the pool and camera lens flare effects are added. Stencil shadows are also applied with respect to the light position to increase realism.

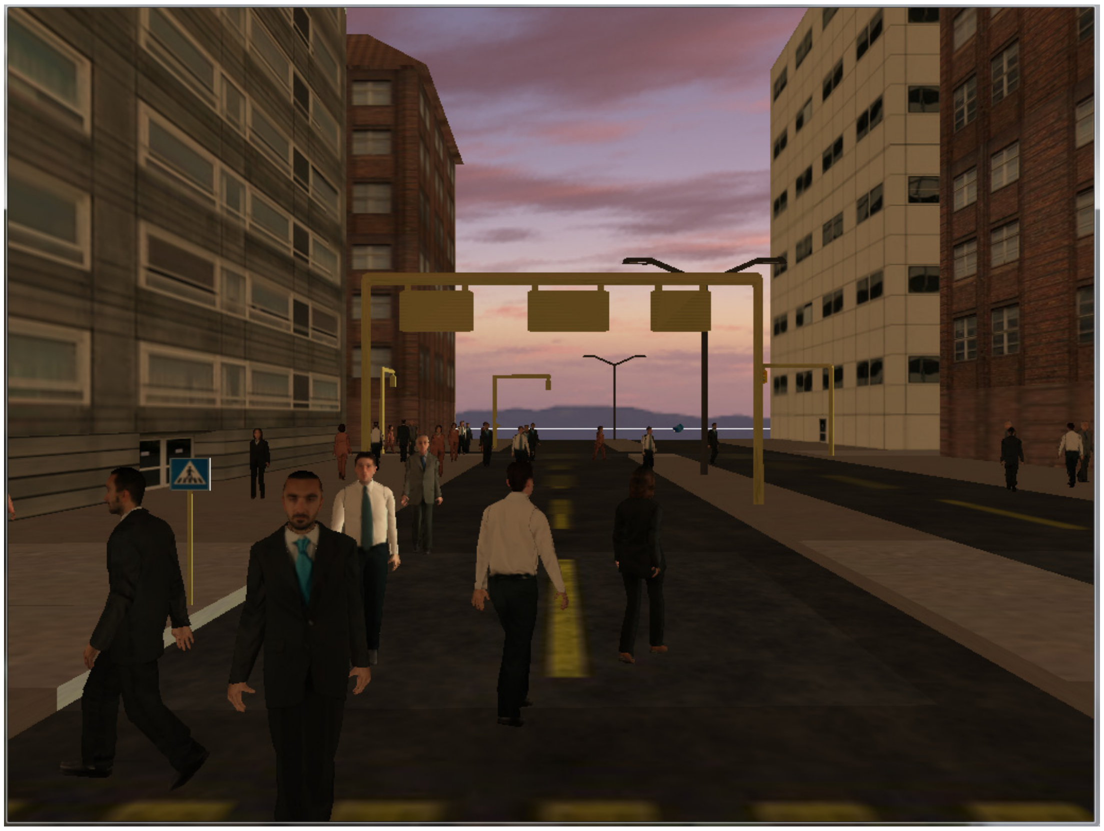


Figure 5.9: A snapshot from the 2000-agent scenario showing the ground level view of the main street.

Simulation statistics are evaluated for four different scenarios which are 500-agent, 1000-agent, 2000-agent and 4000-agent scenarios. These scenarios are defined with respect to the maximum crowd size they permit. These statistics include *(i)* the amount of polygons rendered, *(ii)* average frames per second (fps), and *(iii)* crowd size with respect to simulation time. All statistics are collected within a time period of 200 seconds. In Figure 5.10, these results are illustrated.

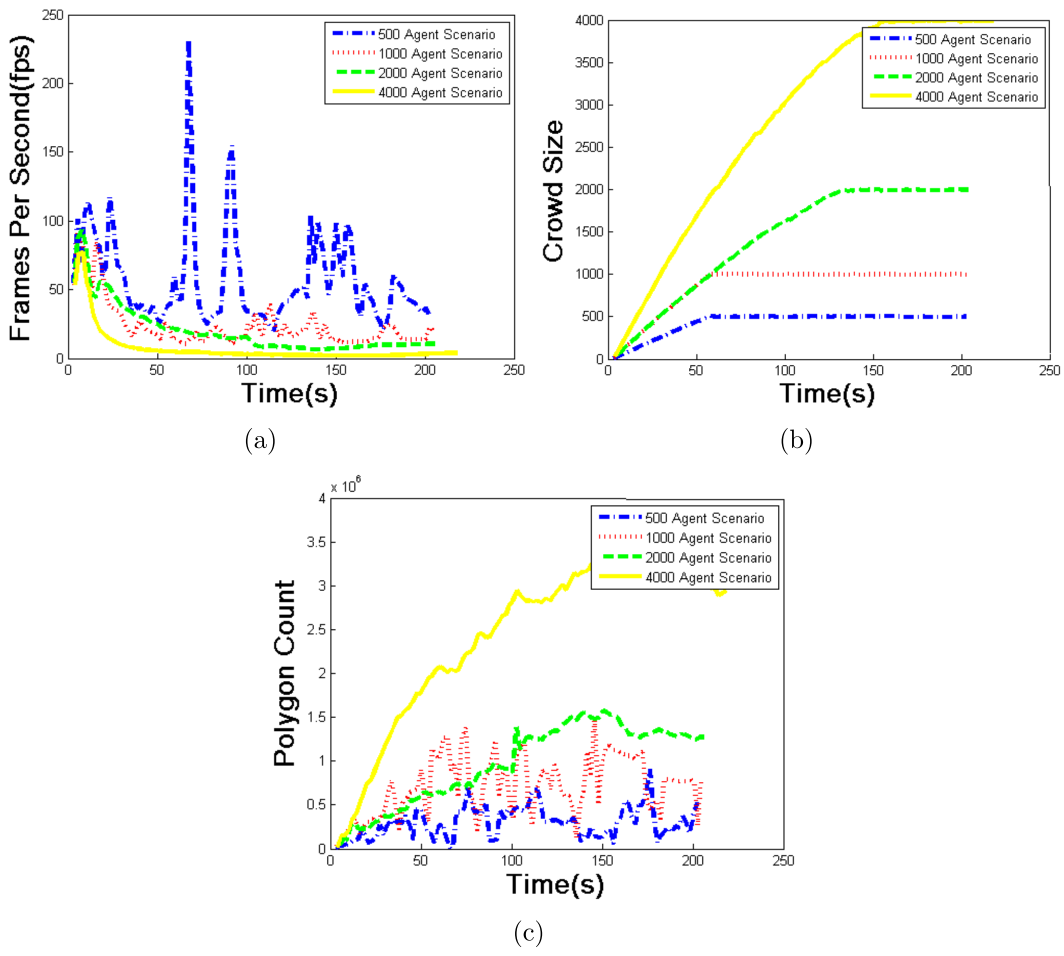


Figure 5.10: Visualization statistics for different simulation scenarios; (a) frames per second (fps) vs time; (b) crowd size vs. time; (c) polygon count vs. time.

Table 5.1: Average frames per second (fps) for different scenarios.

500-Agent	1000-Agent	2000-Agent	4000-Agent
56.08	23.68	20.01	7.59

The average frames-per-second (fps) values for these scenarios are provided in Table 5.1. As seen in Table 5.1, the system operates at near real-time rates for up to 2000 agents. However, it should be noted that in the presented results, optimizations such as crowd based occlusion culling [19] are not applied and the polygonal complexity of the agents is very high for visual quality purposes. For the 4000-agent scenario, as seen in 5.10(c), the peak polygon count for rendering hits up to 3.5 million polygons. Optimizations to reduce polygon count for achieving real-time performance are discussed further in Section 5.3.

To be able to observe the constant time behavior of per agent updates for global path planning, performance measured at every agent update is logged. For each frame, the accumulated computational cost for all agent updates are calculated with respect to varying crowd sizes. These values are expected to follow linear scaling as the simulated crowd size is increased. In Figure 5.11 the accumulated performance of all crowd updates is illustrated and a line is fit to the graph. Computational costs which fall below floating point precision may be rounded to zero especially for crowds which are very small in size.

5.3 Optimizations for Real-Time Performance

There are several methods which are used to lower the count of polygons to be rendered at GPU side. Most common methods in computer graphics are View Frustum and Occlusion Culling methods. View Frustum Culling is the elimination of any polygonal mesh which reside outside of the camera's current view frustum. Similarly Occlusion Culling is the elimination of objects which are not visible as they are occluded by some other objects that are closer to the camera. Some form of spatial partitioning is necessary to calculate both occluded and out of view frustum objects. Common approaches include Octrees and Binary

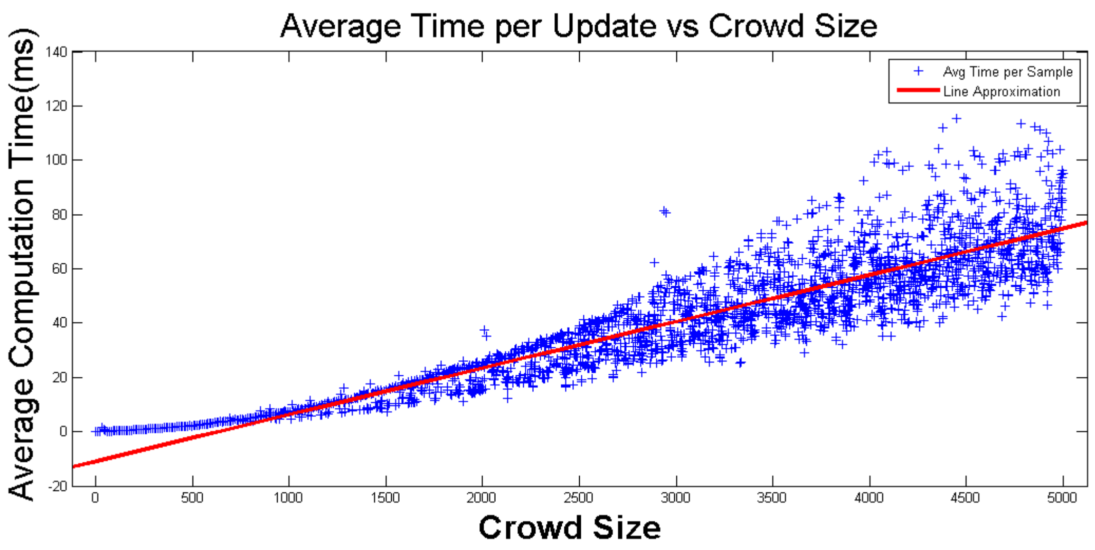


Figure 5.11: Accumulated performance (ms) vs. crowd size for 5000-agent scenario

Space Partitioning methods (BSPs) to partition the space for view frustum and occlusion queries. The underlying application uses BSP tree approach which eliminates out of frustum and occluded objects before a rendering request. The underlying OGRE framework [18] uses axis aligned bounding boxes to speed up view frustum queries [1].

Another approach to lower polygon count is to use Level Of Detail (LOD) [14]. This approach is based on progressively lowering the polygonal detail of an object with respect to the distance from the viewport. LOD technique may both be discrete and continuous. For the continuous case simplification of the original model is computed at run-time. Discrete LOD on the other hand simply replaces the high detail geometric meshes of an object with simplified ones at the set threshold distances. Adaptive Grids application uses four levels of discrete Level Of Detail. For each consecutive level a simpler mesh is replaced which has approximately 30 % less polygons with respect to the previous (high detail) one.

Smaller or thin objects usually fade out of the view faster as they get far away from the viewer's viewpoint. Therefore just by looking at the distance from the camera, small objects can be culled independent of frustum and occlusion. In Adaptive Grids Application each detail object has a distance culling attribute associated with it to fulfill this purpose. This attribute is usually determined with respect to the relative dimensions of the object.

One relatively obvious way of increasing performance is to make crowd updates run in parallel in multiple cores. Crowd update routines input only read-only data which is updated only once at the start and at the end of the simulation frame. Therefore parallelizing the routines is relatively easy. Adaptive grids application benefits from such parallelization via the Open Multi Processing (OMP) library.

5.4 System and Development Environment Configuration

The application is developed with the C++ programming language. Microsoft Visual Studio 2005(SP1) is used as the Integrated Developments Environment. Along with the C++ Standard Template Library, several other libraries and frameworks are used. The most important one was the Ogre 3D Rendering Engine Framework (1.6.4) [18]. In addition Open Multi Processing library (OMP) is used for parallelization support. The tools provided by The Mathworks MatLab R2010A application[17] is used for performing image processing on the Z-Buffer image to obtain the navigation (occlusion) map. MatLab is also used for processing the application data to generate the graphs and figures which are provided in this thesis.

Characters and city geometry, including all the detail objects are hand modeled with Autodesk 3d Studio Max application [15]. Character animations and city Z-Buffer pictures are also obtained with 3d Studio Max. All tests and executions of the application are done on a machine with Intel i7 920 (8Mb Cache, 2.67Ghz Clock) Processor, 6Gb Ram, ATI Radeon HD4790 Graphics Processing Unit (GPU).

Chapter 6

Conclusion

This thesis proposes *Adaptive Grids*, an approach to perform constant time global path planning for simulated virtual crowds. *Adaptive Grids* addresses shortcomings of the previously proposed crowd simulation approaches in terms of scalability, performance, and realism. The proposed approach is easy to implement and can be adapted to any application where path planning is of primary concern.

Considering our simulation results, constant time path planning is feasible in terms of realism, performance and scalability. Many approaches in the literature restrict the behaviors of the agents either by performing path planning for only agent groups (instead of individuals) or by enforcing the agents to follow strict navigation paths. Our approach provides complete freedom to agents in selecting destination points. Hence, *Adaptive Grids* is more appropriate for crowd simulation applications that spend most of the execution time on extensive Artificial Intelligence(AI) routines per agent.

There are several drawbacks of *Adaptive Grids* as well. However, most of them can be considered negligible. For instance, memory requirements of the application totally depends on the size of the navigation graph. Therefore, for different map topologies different space costs may be achieved and it is hard to estimate the cost for a given map topology. Several seed expansion approaches may need to be evaluated to determine the one that fits the underlying topology the best. In addition, the adaptive grid is not uniform and therefore generated paths will most likely to be not optimized in terms of distance and travel times. However, path costs in between irregular cells on the graph still approximate the actual path costs to a great degree. The lowest estimated cost value Manhattan Distance and Shortest Navigable Distance metrics can provide for the distance between two consecutive cell centers corresponds to the actual distance in between these cells (*i.e.* the Euclidean distance between two consecutive cell centers). This also implies that the path costs Manhattan Distance and Shortest Navigable Distance metrics provide never underestimate the actual costs between irregular cells, satisfying the heuristic selection requirements. Therefore, the path costs between irregular cells act as a good heuristic in searching the actual target. Increasing the number of seeds usually leads to better path approximations, while significantly increasing the preprocessing times and the memory costs.

As one last point, there are several potential extensions/future work this approach may benefit from. Firstly, the restriction of rectangular based irregular cells may prevent size of the navigation graph from being reduced to its minimum. A potential extension for Adaptive Grids could be partitioning the regular grid in a way which ensures that all cells are convex and maximized in size (with the least number of cells) would be a good.

Secondly, the potential fields generated for adaptive grids are not continuous on cell borders. This leads to stacking of the agents which share the same path for a common goal over time. To be able to address this problem and to provide smoother movement behavior without stacking, the potential field connecting a pair of irregular cells may be defined as a deformed vector field which has control points on the consecutive cell intersections and cell corners. By deforming a uniform vector field (similar to Free Form Deformations (FFD) [24]) to match the shape of the neighboring cell pairs, it may be possible to derive a deformed vector which would point towards a particular goal position on the grid.

Bibliography

[1] U. Assarsson and T. Moller. Optimized view frustum culling algorithms for bounding boxes. *Journal of Graphics Tools*, 5(1):9–22, 2000.

[2] O. Bayazit, J. Lien, and N. Amato. Roadmap-based flocking for complex environments. In *Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, page 104, 2002.

[3] A. Braun, S. Musse, L. de Oliveira, and B. Bodmann. Modeling individual behaviors in crowd simulation. In *Proceedings of the 16th International Conference on Computer Animation and Social Agents (CASA)*, pages 143–148, 2003.

[4] S. Chenney. Flow tiles. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pages 233–242, 2004.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2001.

[6] R. C. Gonzalez and R. E. Woods. *Digital Image Processing.* Prentice Hall, Upper Saddle River, N.J., 2008.

[7] D. Helbing, L. Buzna, A. Johansson, and T. Werner. Self-organized pedestrian crowd dynamics: Experiments, simulations, and design solutions. *Transportation Science*, 39(1):1, 2005.

[8] D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature*, 407(6803):487–490, 2000.

[9] D. Helbing and P. Molnar. Social force model for pedestrian dynamics. *Physical Review E*, 51(5):4282–4286, 1995.

[10] J. Hodgins and D. Brogan. Robot herds: Group behaviors for systems with significant dynamics. In *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 319–330, 1994.

[11] T. Lakoba, D. Kaup, and N. Finkelstein. Modifications of the Helbing-Molnar-Farkas-Vicsek social force model for pedestrian evolution. *Simulation*, 81(5):339, 2005.

[12] A. Lerner, Y. Chrysanthou, and D. Cohen-Or. Efficient cells-and-portals partitioning. *Computer Animation and Virtual Worlds*, 17(1):21–40, 2006.

[13] C. Loscos, D. Marchal, and A. Meyer. Intuitive crowd behaviour in dense urban environments using local laws. In *Proceedings of the Theory and Practice of Computer Graphics*, pages 122–129, 2003.

[14] D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics*. Morgan-Kaufmann Publishers, San Francisco, 2002.

[15] Autodesk, Inc. 3D Studio Max. `http://www.autodesk.com`, Accessed at July 2010.

[16] J. Maïm, B. Yersin, and D. Thalmann. Real-time crowds: architecture, variety, and motion planning. In *ACM SIGGRAPH ASIA Courses, Course no. 56*, 2008.

[17] The MathWorks Inc. MatLab Computer Algebra System. `http://www.mathworks.com`, Accessed at July 2010.

[18] Torus Knot Software, Ltd. Ogre 3D Rendering Engine Framework. `http://www.ogre3d.org`, Accessed at July 2010.

[19] O. Oğuz, A. Akaydın, T. Yılmaz, and U. Güdükbay. Emergency crowd simulation for outdoor environments. *Computers & Graphics*, 34(2):136–144, 2010.

[20] N. Pelechano, J. Allbeck, and N. Badler. Controlling individual agents in high-density crowd simulation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 99–108, 2007.

[21] J. Pettre, J. Laumond, and D. Thalmann. A navigation graph for real-time crowd animation on multilayered and uneven terrain. In *First International Workshop on Crowd Simulation*, pages 81–90, 2005.

[22] C. Reynolds. Flocks, herds and schools: A distributed behavioral model. *ACM Computer Graphics (Proceedings of ACM SIGGRAPH)*, 21(4):25–34, 1987.

[23] C. Reynolds. Steering behaviors for autonomous characters. In *Proceedings of Game Developers Conference*, 1999.

[24] T. Sederberg and S. Parry. Free-form deformation of solid geometric models. *ACM Computer Graphics (Proceedings of ACM SIGGRAPH)*, 20(4):151–160, 1986.

[25] W. Shao and D. Terzopoulos. Autonomous pedestrians. *Graphical Models*, 69(5-6):246–274, 2007.

[26] A. Treuille, S. Cooper, and Z. Popović. Continuum crowds. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 25(3):1160–1168, 2006.