Adaptive grids: an image-based approach to generate navigation meshes

Ateş Akaydın
Uğur Güdükbay
Bilkent University
Department of Computer Engineering
06800 Bilkent, Ankara, Turkey
E-mail: gudukbay@cs.bilkent.edu.tr

Abstract. We propose adaptive grids, an image-based approach for constructing navigation meshes, which are used for path planning. A cellular navigation mesh, called an adaptive grid, is constructed from a top-view range image of a three-dimensional urban model. A navigation graph can then be extracted from this adaptive grid for path planning. We compare our approach with two popular navigation mesh–generation approaches and obtain promising results in terms of path accuracy and memory cost. [DOI: 10.1117/1.OE.52.2.027002]

Subject terms: image processing; crowd simulation; adaptive grids; space subdivision; path planning; navigation meshes.

Paper 121156 received Aug. 10, 2012; revised manuscript received Dec. 5, 2012; accepted for publication Dec. 20, 2012; published online Feb. 1, 2013.

1 Introduction

Path planning is a common problem in a variety of fields such as crowd simulation and robotics. Path planning is used in navigation applications for artificial objects (such as robots or virtual people). For the sake of generality, we name all such objects "agents." Algorithms developed for path planning make use of some special data structures such as road maps, Voronoi diagrams, and navigation meshes (NavMeshes). NavMeshes are a popular way among these methods to represent the navigable space of a domain with obstacles. NavMeshes allow for the geometry of a scene to be represented in a simplified mesh consisting of adjacent convex polygons. To achieve path planning, paths are mapped between polygons by crossing passages, or the edges of the neighboring polygons. A NavMesh can also be represented by a graph (commonly known as a navigation graph) where the polygons correspond to vertices, and edges correspond to graph edges connecting neighboring polygons.

This paper describes a technique called adaptive grids, an image-based approach for generating NavMeshes. The method focuses on a new way of creating a NavMesh for path planning to address challenges centered on the trade-offs between path accuracy, computational complexity, and memory usage.

As with our approach, other NavMesh approaches also decompose the navigable space into a set of convex polygons. However, there are notable differences. First, exact polygonal decomposition of the navigable space requires significant computational processing for previous approaches. On the other hand, the proposed approach has $O(n^2)$ complexity, where n is the grid dimension (i.e., resolution of the range image). Second, our iterative approach is easy to understand and implement. It does not require any significant geometric processing on the synthesized three-dimensional (3-D) model. Third, fidelity of the NavMeshes generated by our approach is controllable with a small set of parameters. For static path planning, we also demonstrate that the paths our approach generates are more accurate than the paths generated by existing methods.

The main idea behind this work is to introduce a new way to construct a NavMesh that offers advantages over the previous methods. Our contributions are as follows:

- The concept of adaptive grids is introduced; it generates a NavMesh from a range image by expanding seeds, which are initial cells used to construct the cell clusters.
- The proposed algorithm to generate adaptive grids is easy to implement and can be parallelized.
- Quality of the generated NavMeshes (in terms of resolution and path accuracy) can be controlled with a small set of parameters.
- Convex rectangular decomposition of the domain provides search efficiency for agent locations.
- Adaptive grids can easily be extended to multiple dimensions. It can be used for path planning in 3-D domains.
- Static path planning is achieved in $O(1)$ time per agent with controllable error on agent paths and minimal memory requirements.

We demonstrate the capabilities of our approach within a crowd simulation framework. We consider a low-density massive crowd with thousands of agents navigating within a virtual city model. For this specific example, we generated a low-resolution NavMesh suitable for path preprocessing in terms of memory cost. This setting enables us to perform least-complexity (linear time) path planning for the entire crowd. In this setting, only a minor fraction of the computer resources are dedicated to path planning. Therefore extensive artificial intelligence (i.e., personality) computations can be processed per agent. It is important to note that the focus of this work is on NavMesh generation. Hence, the crowd simulation application we provide depends on traditional approaches taken in the field. However, our approach is perfectly suitable for any path-planning method making use of NavMeshes.

0091-3286/2013/$25.00

The paper is organized as follows. Section 2 briefly reviews previously proposed approaches for path planning and crowd simulation. Section 3 explains the proposed approach of creating adaptive grids. Section 4 describes the evaluation metrics and parameters used to generate different types of adaptive grids and then discusses them in further detail to find a suitable configuration for the adaptive grid–generation algorithm. Section 5 provides statistical and empirical results obtained by simulations with different configurations and compares the results with the traditional methods for creating triangular NavMeshes. Section 6 concludes by discussing key results and future work.

2 Background and Related Work

Path-planning problems in virtual environments can be separated into two parts: local and global path planning. Local path planning is used to avoid or respond to contacts/collisions between agents and objects in close proximity. Global path planning is used to direct agents toward distant goals such as building entrances or specific spots on the terrain.

For local path planning, various approaches derived from the social force model of Helbing et al.[1] are proposed. The social force model applies tangential, repulsion, and attraction forces to simulate interactions between individuals and other obstacles. There are numerous extensions of the model, such as the Helbing-Molnar-Farkas-Vicsek social force model[2] and the self-organized pedestrian crowd dynamics model.[3]

For global path planning, almost all cases require some sort of high-level representation of the simulation environment to support interactive simulations. Common techniques are portal graphs,[4,5] roadmaps,[6] potential fields,[7] and NavMeshes.[8]

Since the NavMesh concept was first introduced by Snook, various researchers have proposed different ways to construct NavMeshes, which range from triangles to a mix of convex polygons.[8] In these approaches, the roadmaps are constructed from convex polygonal decompositions of the navigable space. Kallmann et al.[9] construct a NavMesh using the constrained Delaunay triangulation.[10] Tozour[11] describes a different approach using the 3-D triangle mesh of a scene to create convex polygons representing the navigable area. The polygons in the mesh can be triangles, quads, or arbitrary convex polygons. O'Neill[12] describes how to efficiently find paths on a NavMesh. Sturtevant and Geisberger[13] extensively analyze the ways of abstraction for NavMesh approaches to reduce storage requirements.

Comprehensive approaches to crowd simulation incorporate both local and global path planning methods. Sud et al.[14] propose adaptive elastic roadmaps (AERO), which can dynamically change to accommodate for itinerant obstacles and agents. Li and Gupta[15] use coordination graphs (CGs) to locally modify agent paths to avoid deadlocks in narrow passages. They parallelize the processing of CGs to achieve real-time performance. Guy et al.[16] use the principle of least effort (PLE) to assign weights to roadmap edges. This approach involves minimization of biomechanical energy along the paths.

A popular approach based on continuum dynamics, continuum crowds, is proposed by Treuille et al.[17] Their method evaluates a potential field over the simulation grid at each frame to direct the agents. Maïm et al.[18] further extend continuum crowds by integrating a navigation graph composed of circular search nodes for global path planning. In this approach, potential fields of the continuum crowds model are used for computing paths within nodes only.

3 Adaptive Grids

With the adaptive grids approach, a grid composed of adaptive cell clusters is constructed by extracting the navigable space from the virtual environment. This adaptive grid is used as a NavMesh on which different pathfinding algorithms can be applied.

We have developed a crowd-simulation application to demonstrate adaptive grids. The simulation takes place in a 3-D virtual city. The approaches we have taken for crowd simulation are based on static (preprocessed) path planning and combine both local and global path-planning methods to direct agents toward their goals. For the sake of generality, these goals are always meant to be positions in the 3-D space at terrain level. The social force model[19] is used as the base dynamics model that integrates both the global and local path-planning concepts. Local path planning considers the short-term goals and maneuvers of individuals based on immediate factors such as possible collisions. It also takes into account the physical and social forces applied on an individual agent due to its interaction with (1) neighboring agents and (2) the surrounding environment. In the social force model,[19] an equation is proposed which defines the local force applied on the agent due to its interaction with the other agents in the system. Readers may refer to Ref. 1 for additional details.

Conversely, global path planning is used to direct individuals to their long-term goals and calculate the desired direction vector of the agent. Using the social force model as a basis, global path planning can be performed independently from local path planning. In our case, global path planning is performed using a navigation graph extracted from an adaptive grid.

The proposed approach constructs an underlying grid, which can be partitioned into a set of clusters. A navigation graph is then formed from the set of clusters. All paths are computed on this navigation graph as a preprocessing step to make constant-time static path planning possible.

The solution to the crowd-simulation problem with static path planning involves five steps (cf. Fig. 1). In step 1, we generate a Boolean navigable grid from a city model. In step 2, an adaptive grid is formed. In step 3, its respective navigation graph is created. In step 4, the Dijkstra or Floyd Warshall algorithm is applied on the navigation graph to find and store the shortest paths. Step 5 generates the vector field to direct agents toward their global goals in constant time. Agents query the vector fields at their positions to determine their global paths. Steps 1, 2, and 3 clarify the primary contribution (adaptive grids) of this work. Steps 4 and 5 can be customized with respect to the needs of other similar applications. These five steps are discussed in the following sections.

3.1 *Navigable Space Extraction*

A range image is taken from an axis-aligned top-view of a city representing the height map of an outdoor city environment. For virtual cities (and synthetic 3-D models), this range image corresponds to the z-buffer image. The range

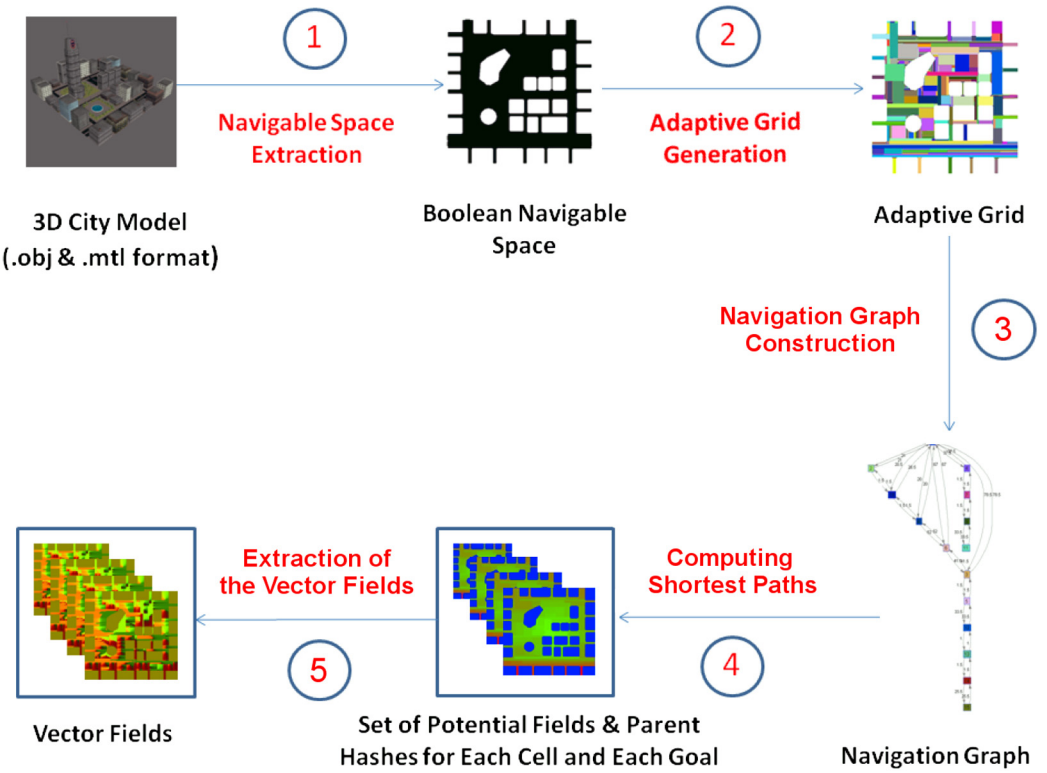


Fig. 1 Flow diagram of the proposed solution to crowd simulation.

image does not need to be taken from a synthesized 3-D model as well. It can also be taken from real sources. For instance, range images of real cities can be taken by using laser imaging detection and ranging (LIDAR) technologies. Our approach is strictly image-based compared to the existing geometric NavMesh generation methods.

For a virtual city, small objects that may obstruct this height map (e.g., trees, traffic lights, and benches) are excluded, leaving only buildings and the terrain. For a real city where the 3-D model is not known beforehand, removal of such small details from the range image may be carried out using image-processing methods.

A variety of filters (such as Gaussian and median filters) may be applied to the range image to reduce the effects of noise. This first step is especially necessary for range images belonging to real sources. In the second step, the range image is filtered by Sobel filters in different directions to detect edges. A fixed threshold is applied to each of these Sobel-filtered images to obtain their corresponding Boolean images. Using the logical *and* operator, these images are combined into a single image. Gaps may occur on this combined image, which may lead to incorrect topological information. We use morphological operators such as *closing* to close these gaps. In the last step, connected component analysis is applied on this closed image. Navigable components are hand-picked, and the desired navigable space is extracted by assigning a logical *one* value to all pixels belonging to the navigable components. The rest of the pixels are assigned a logical *zero* value.

3.2 *Adaptive Grid Generation*

The navigable-space image includes information regarding the topology of the virtual urban environment. Using this information, it is possible to construct the adaptive grid structure.

Our method partitions the navigable space image into a number of convex regions, called cell clusters, including one or multiple cells using an expansion based approach. We use the term "seed" for a single cell, which is expanded into a cell cluster.

Figure 2 gives the adaptive grid–formation and navigation graph–construction algorithm. In line 2 of the algorithm, the *genInitialSeeds* function is used to select a set of initial seeds from the navigable space image and add them to the *InitSeeds* data structure. The initial seeds cover only one pixel, and they form the first set of clusters to be expanded. The initial seed selection determines the result of the algorithm in terms of cell-cluster size and count; we propose and evaluate different initial seed-selection methods.

In line 9 of the algorithm, the *expand* function is used to expand a single given seed. A seed can only expand to navigable and unoccupied cells. The expand function returns false if the given seed cannot be expanded; this result usually happens when it is surrounded by non-navigable cells or all cells toward the expansion direction are already occupied by cell clusters generated from other seeds. If a given seed cannot be expanded, it is removed from the queue, as it does not require any further processing. Otherwise, the seed is inserted back into the queue. The behavior of the expansion method can significantly affect the results in terms of cluster size, shape, and count.

The four different characteristics of the seed expansion method are (1) operating dimensions, (2) expansion ordering, (3) memory property, and (4) restrictiveness. The operating dimensions specify the dimension of expansion. For example, a one-dimensional algorithm expands in a single

Algorithm: 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 Cluster-Ids.
Result: Adapts the given regular connectivity grid *NavGrid* and stores adaptive seed information on regular grid *SGrid* and the relevant navigation graph in *SGraph*.

```
1  begin
2      InitSeeds ← genInitialSeeds(NavGrid);          /* Choose initial seeds */
3      foreach Seed s ∈ InitSeeds do
4          Queue.push(s);                          /* Push seeds to the queue */
5          SGrid.set(s.row, s.col, s.id);
6          SGraph.addVertex(s);                     /* Add seeds to the graph */
7      while Queue ≠ ∅ do
8          s ← Queue.pop();
9          if expand(s, SGrid, NavGrid) then              /* Expand the seed */
10             Queue.push(s);
11         else                    /* If cannot expand, generate new seeds */
12             genSeedsFromSeed(s, SGrid, NavGrid, SGraph, Queue);
13     foreach s ∈ SGraph.V do              /* Construct navigation graph */
14         connectSeedToNeighbors(s, SGrid, SGraph);
```

Fig. 2 Adaptive grid formation and navigation graph construction algorithm.

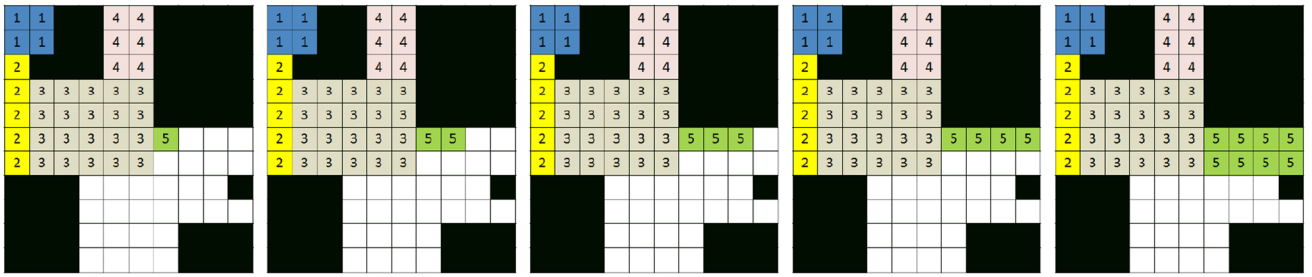
direction (east, south, west, or north), whereas a multidimensional algorithm may expand toward all directions at once or toward intermediate directions. Expansion ordering specifies direction priorities. For instance, a clockwise ordering may cause the algorithm to expand the seed along east, south, west, and north directions. An expansion method with the memory property remembers the last direction it expanded to and continues from the successive directions in a breadth-first manner. A greedy method (i.e., memoryless) will exhaustively expand toward the same direction before trying other directions. Cell-cluster 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 clusters are restricted by a maximum size. Although larger clusters lead to a lower cell count, they also increase the error made in path costs on the navigation graph. Restricting cluster size may keep that error within an acceptable range. Figure 3 shows the behavior of two different expansion methods for adaptive grid construction. Results are collected after running the algorithm for five iterations, starting from the seed with id = 5.

For each cluster 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 areas on the adaptive grid (*SGrid*). The default algorithm (*genSeedsFromSeed*) takes a fully expanded seed and operates clockwise, evaluating immediate neighbors of the expanded seed, excluding corners. Each unoccupied navigable grid element, following a series of occupied or unnavigable cells, is marked. New seeds are generated on the marked cells and added as vertices to the seed graph. They are also inserted into the seed queue to continue the adaptive grid formation process. The seed generation process, which is a stage of the adaptive grid construction, is illustrated in Fig. 4. An example of an adaptive grid of size 156×156 cells is given in Fig. 5(a).

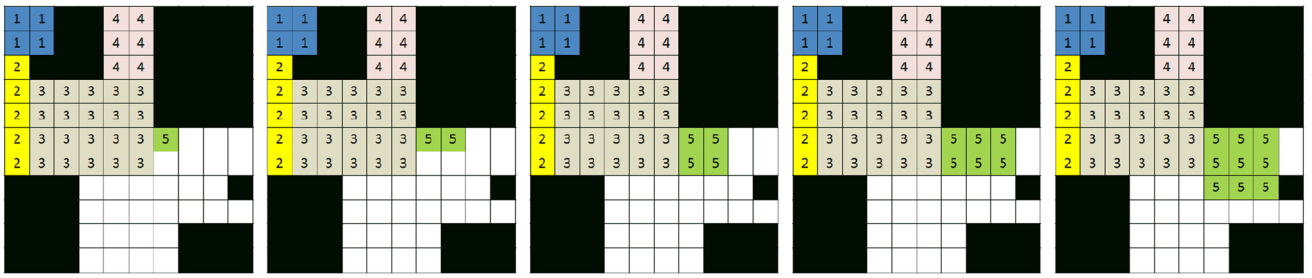
3.3 *Navigation Graph Construction*

When all of the seeds within the queue are evaluated and all expansions are completed, the graph construction step begins (*connectSeedToNeighbors* in line 14 of algorithm in Fig. 2). For each cell cluster, we identify immediate neighbors on the adaptive grid (*SGrid*) and introduce an edge connecting neighboring cluster pairs with an attached cost, which can be calculated using (1) Manhattan (block), (2) Euclidean, or (3) shortest navigable distance (SND) metrics.

SND is the shortest navigable distance connecting two cluster centers that does not intersect with unnavigable cells. If the cluster centers are in sight of each other, then the SND is equal to the Euclidean distance. Otherwise, the SND is the sum of the separate Euclidean distances between the cluster centers and the shared unnavigable neighbor corners. Figure 6 shows the Manhattan, Euclidean, and SND metrics for evaluating edge costs on the grid. Using one of these



(a) Clockwise memoryless 1D expansion



(b) Clockwise breadth-first (with memory) 2D expansion

Fig. 3 Seed expansions for five consecutive calls using different expansion methods.

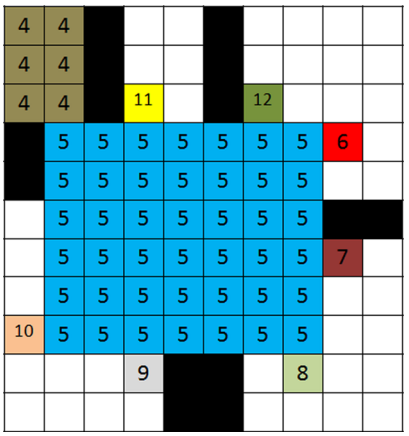
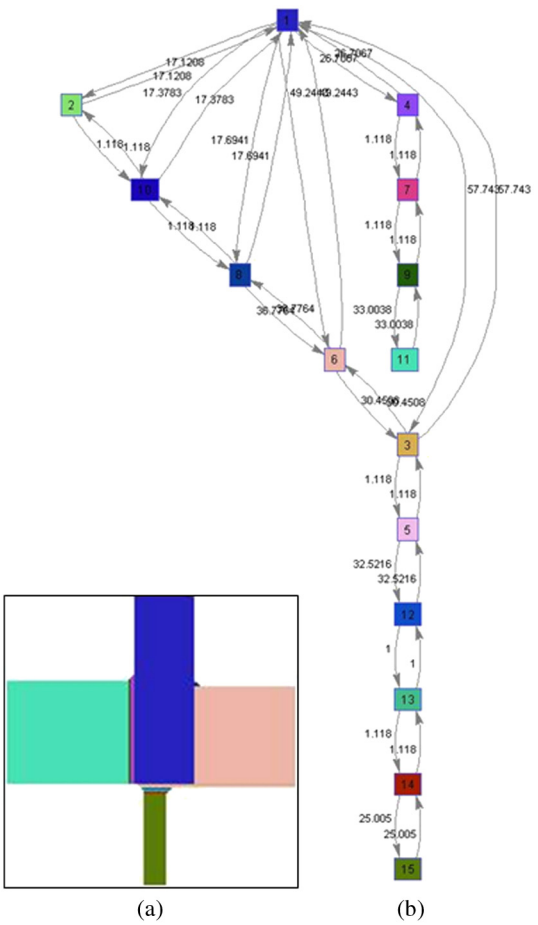


Fig. 4 Seed generation starting from a fully expanded cluster with cluster id = 5. Seeds with IDs 6 to 12 represent newly generated seeds in clockwise direction.

methods, edge costs are calculated for each cluster pair by the adaptive grid formation algorithm. and a navigation graph is formed.

3.4 *Computing Shortest Paths*

Navigable space extraction, adaptive grid formation, and navigation graph construction explained in the previous steps are the primary focus of our approach. The constructed navigation graph is suitable for path planning using both dynamic (i.e., A^* search) and static methods. To demonstrate adaptive grids and to compare it with the existing *NavMesh* generation techniques, we've proposed a crowd-simulation application using static path planning. Adaptive grids is particularly powerful for this form of path planning because it can narrow down the search space. The memory cost of storing the navigation graph is close to minimum, as the



(a) (b)

Fig. 5 (a) A simple grid instance. (b) The corresponding navigation graph generated with the shortest navigable distance metric. The same colored vertices on the navigation graph (b) correspond to the clusters with the same color in the grid instance (a).

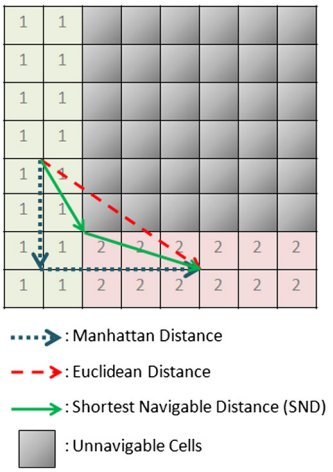


Fig. 6 Different distance metrics for evaluating edge costs visualized on navigation graph.

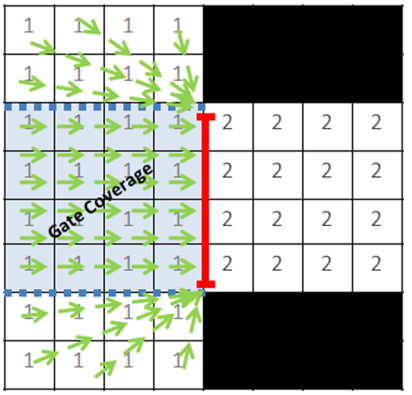


Fig. 7 Example of vector field formation by considering passage coverage. The numbers show the Cluster-IDs for the pixels; i.e., to which cluster the cell belongs. The distances between clusters 1 and 2 can be calculated using different distance metrics given in Fig. 6.

approach reduces the number of vertices significantly. The following sections will explain the static (preprocessed) path-planning approach.

For static, preprocessed path planning, it is necessary to find and store paths from each cluster to every other cluster on the grid. To this end, the Floyd–Warshall algorithm or the Dijkstra algorithm can be applied on the navigation graph for all clusters to compute all-pair shortest paths. *Cluster-IDs* are given in consecutive order starting from 1 on the grid; 0 denotes unnavigable cells. An array indexed by target *Cluster-IDs* can store the immediate links lying on the shortest path from the current cluster toward the target cluster. At any time, a query for a particular target cluster from a root cluster can be answered with $O(1)$ time. An example of a navigation graph corresponding to the adaptive grid given in Fig. 5(a) is shown in Fig. 5(b). This example uses the SND metric.

Queue type, distance metric, seed expansion method, and initial seed generation method are configurable parameters of the adaptive grids algorithm. These parameters are configured on demand to bound path error and memory. In Secs. 4 and 5, the error made on path costs is defined and comparative results obtained by using different parameters are provided. The targets picked by the agents may not necessarily be clusters; they can also be points in the clusters. Because all clusters are convex (rectangular), it is guaranteed that the paths within clusters are free of static obstacles.

3.5 *Extraction of the Vector Fields*

Having computed and stored all shortest paths, a vector field should be generated on all points within the grid to smoothly direct the agents toward their global goals. Such a vector field may be generated using many different methods. The simplest one is to direct each agent toward the center of the common edge that is shared by the agent's current cluster and the next cluster, which leads to the agent's global target. Such edges may be named "passages." But this approach is not preferable, as it will lead to aligning of agents who share a common goal as they progress through their path, because all such agents are aiming for a single point at each cluster.

A better approach would be to direct all agents that reside within the passage's coverage toward the passage. Passage coverage defines the rectangular zone within the cluster where moving toward the passage would be enough to exit the cluster on the desired path. Agents that are out of the passage's coverage can pick the passage's closest end point as their primary target and move toward this point before exiting the passage. Figure 7 shows an example of the passage-coverage approach. Passage coverage prevents the lining up of agents at common passage centers, as the whole passage is now regarded as an exit.

4 Evaluation of Grid Adaptation

We first discuss in detail a number of parameters that affect the outcome of the adaptive grid formation algorithm. Then, the performance metrics used to evaluate the results are described.

4.1 *Adaptive Grid Parameters*

Four types of parameters are used in the formation of an adaptive grid. These parameters are (1) distance metric, (2) data structure, (3) initial seed-generation method, and (4) seed-expansion method.

The distance metric determines the way of measuring the distance between two cell clusters in an adaptive grid. It can be chosen as Manhattan (block) distance, Euclidean distance, or SND. SND takes into account the navigable/non-navigable distance due to the buildings.

The data structure determines the order in which clusters are processed during the adaptive grid formation. We consider (1) queue, (2) stack, (3) min-heap, and (4) max-heap as possible data structures. For min-heap and max-heap data structures, cluster size is considered as a key.

We employ four different methods for initial seed generation: (1) single, (2) border, (3) random, and (4) sampled. The single method uses a single navigable cell at the top left corner of the grid as the initial seed. The border method picks navigable cells at the borders of the regular grid as the initial seeds. The random method randomly draws a number of the navigable cells as the initial seeds. In the sampled method, a set of initial navigable cells is chosen as seeds by sampling the regular grid with a pixel period in two dimensions.

The last parameter, the seed-expansion method, determines the behavior of the seed expander explained in detail

in Sec. 3. The seed expander may choose to expand in one or two dimensions at a time. It can restrict cluster size and may choose to expand using a depth-first strategy (i.e., memoryless) or a breadth-first strategy.

4.2 *Evaluation Metrics*

The adaptive grids obtained using different parameter values are evaluated using four different metrics, which are (1) cluster count, (2) average cluster size, (3) average aspect ratio, and (4) mean square error (MSE).

Cluster count is the total number of clusters in the constructed adaptive grid. Using more clusters reduces the accumulated error on path costs at the cost of exponentially increasing memory and preprocessing time. The average cluster size corresponds to the average size of all clusters in the constructed adaptive grid. Smaller average cluster sizes usually indicate a higher number of clusters as well as reduced error on path costs. Aspect ratio is the ratio of the width of a seed to its height. The average aspect ratio is calculated as the mean of the aspect ratios of all clusters. For static path planning, this number should be as close to 1 as possible, since highly varying aspect ratios have a negative effect on the error introduced. The MSE is calculated with respect to the difference in distances to the same location on both the regular and adaptive grids.

$$\mathrm{MSE}(X,R) = \frac{1}{(m \times n)} \sum_{i=1}^{m} \sum_{j=1}^{n} [X(i,j) - R(i,j)]^2. \quad (1)$$

In Eq. (1), m is the number of rows and n is the number of columns of the original regular grid (range image) and also the corresponding adaptive grid. R represents the shortest distance of each pixel to the center of the original grid (using the navigable area), whereas X represents the distances on the navigation graph constructed using the adaptive grid. Figure 8 shows the pixel distances to the center of the regular grid, and Fig. 9 presents the corresponding distances in an adaptive grid.

5 Results

5.1 *Forming Adaptive Grids*

Figures 10 and 11 present statistics for 132 different adaptive grids constructed with different combinations of parameters. These statistics compare the results obtained using the different evaluation metrics given in Sec. 4.2.

As can be observed from Figs. 10 and 11, although an increase in the cluster count notably reduces MSE (i.e., S8 and S16), it significantly increases the overhead in memory requirements and preprocessing time. For adaptive grids that have few clusters (i.e., those with single [Sng] and border [Bor] initialization) and for grid adaptations that have a large number of clusters (like those with sampled 8 [S8] and random 500 [R500] initialization), the MSE values obtained are lower than those with medium amounts of clusters (such as R100 and S128 initialization). Especially for clusters with high and varying aspect ratios, the accumulated error on the paths grows as the number of clusters on the paths increases. However, an increase in cluster count generally causes a decrease in cluster size and restricts the aspect ratio; after some point, the accumulated error starts to decrease. Hence, the most successful grid adaptations are achieved with R100 and S128 grid initialization schemes and by using a common, round-robin queue.

5.2 *Comparing Adaptive Grids with Other Approaches*

Traditional approaches to creating NavMeshes involve creating a triangular mesh based on the geometry of a scene. We compare the adaptive grid approach with a Delaunay triangulation of the scene, which is similar to Kallmann's use of the constrained Delaunay triangulation. We also compare adaptive grids with various triangular meshes constructed using the recast toolset.[20]

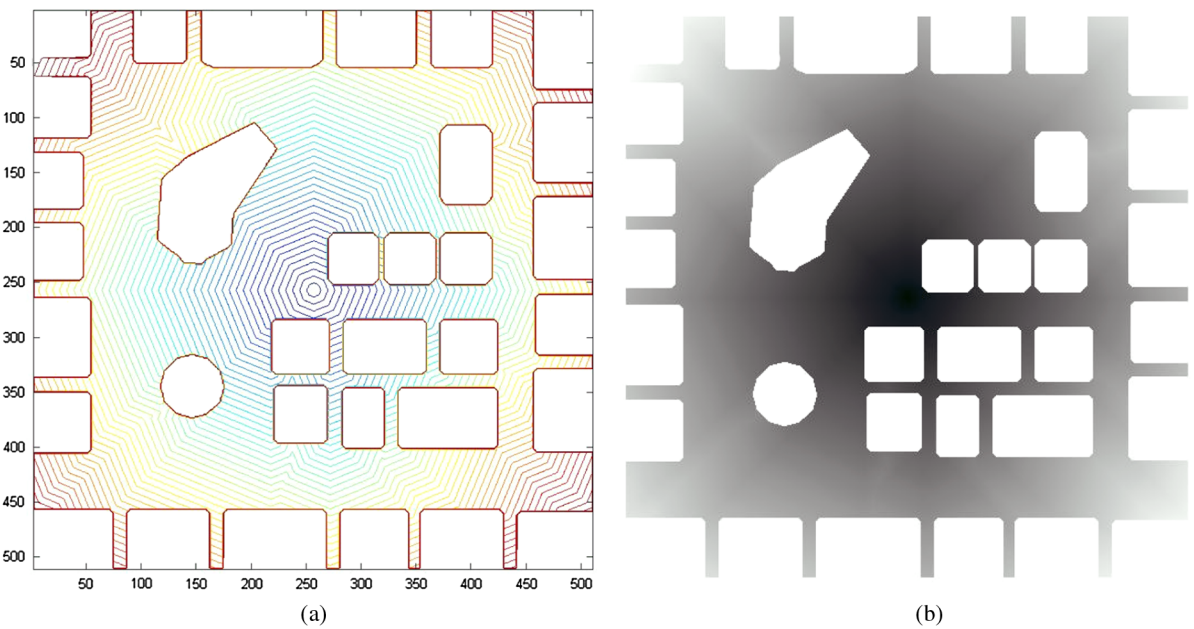
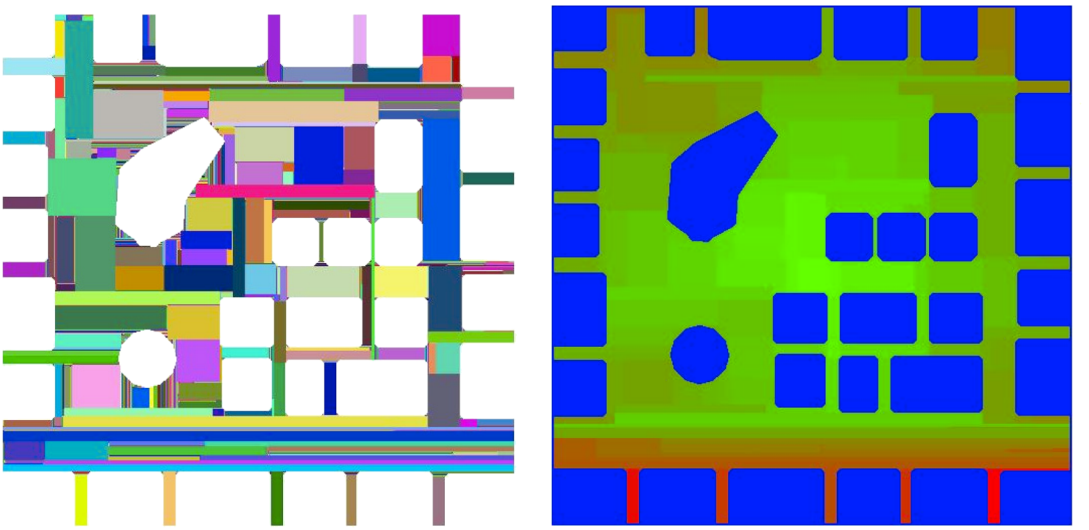
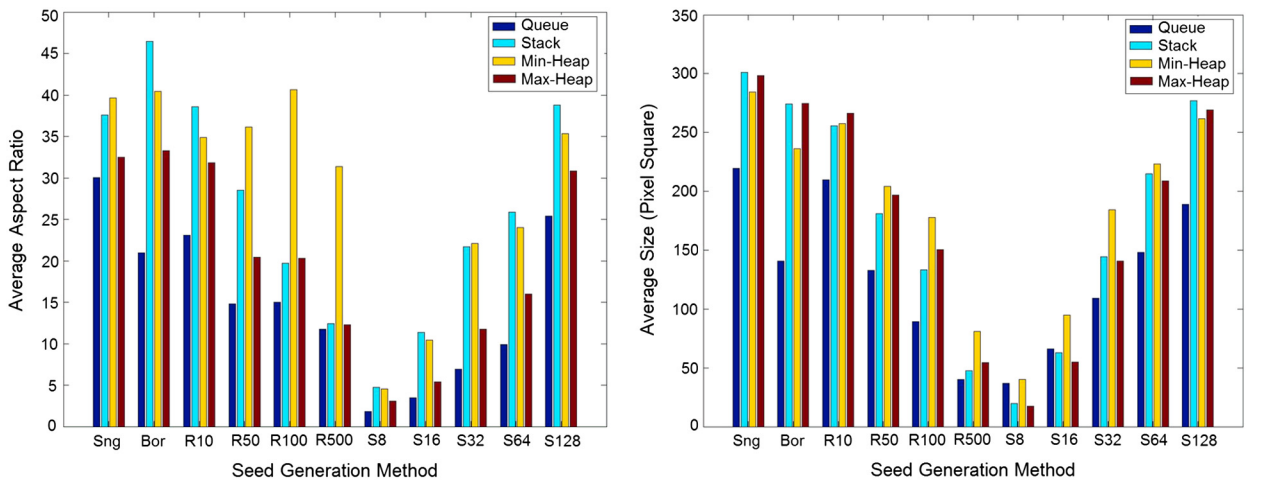


Fig. 8 Pixel distances on the regular navigation grid.



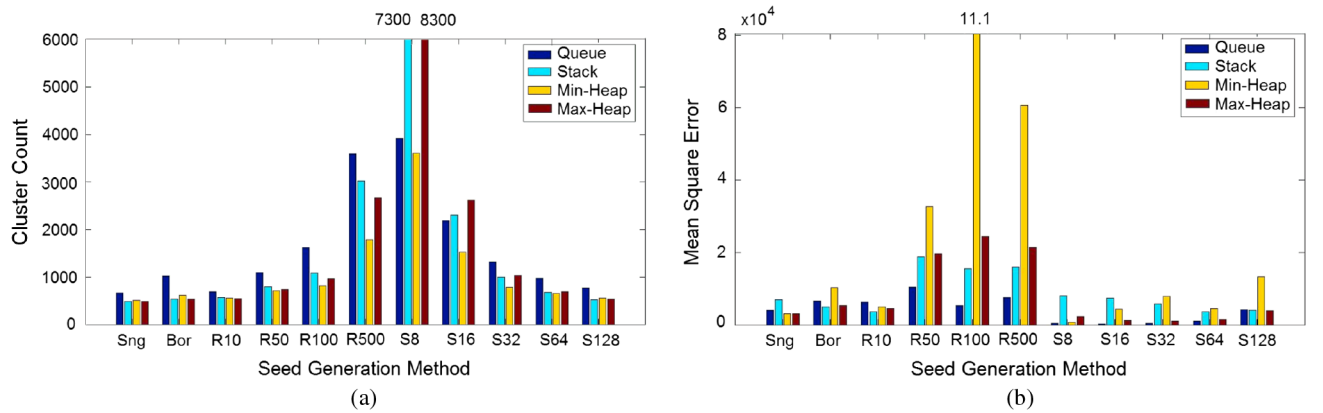
(a) (b)

Fig. 9 Pixel distances on the adaptive grid.



(a) (b)

Fig. 10 (a) Cluster aspect ratios for adaptive grids with different parameters. Seed initialization methods are single (Sng); border (Bor); random 10 (R10), R50, R100, R500, sampled 8 (S8), S16, S32, S64, and S128, respectively. (b) Cluster sizes for adaptive grids with different parameters.



(a) (b)

Fig. 11 (a) Cluster counts for adaptive grids with different parameters. (b) Mean square error (MSE) for shortest navigable distance metric.

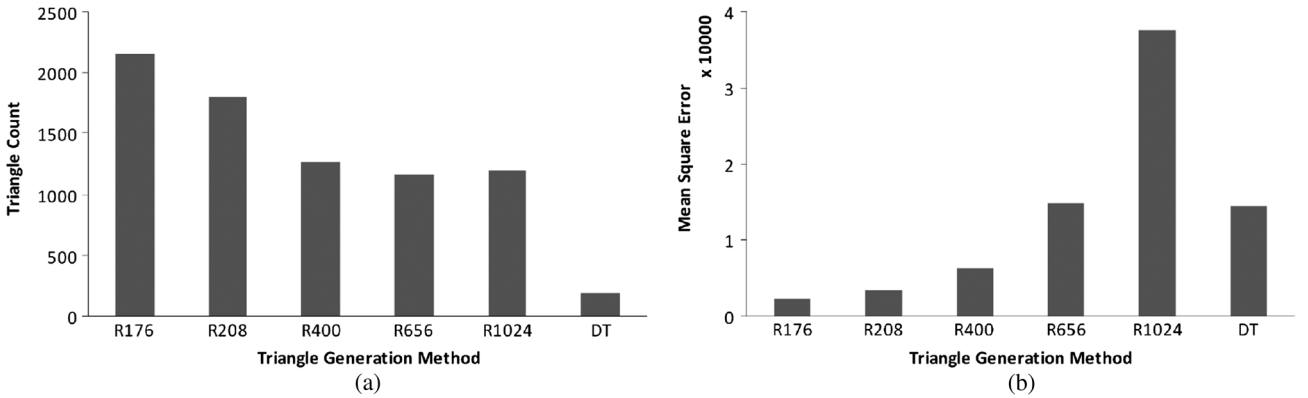


Fig. 12 (a) Triangle counts for different triangular NavMeshes. Triangle generation methods are recast using different number of tiles and the Delaunay triangulation. (b) MSE for triangle meshes.

Using the triangular meshes, the navigation graph construction algorithm is applied to compute the MSE. The results for various meshes are shown in Fig. 12. The Delaunay triangulation, which consists of 192 triangles, results in an MSE of 14,365 and has the smallest average triangle aspect ratio. The recast mesh with the smallest number of triangles (1160 triangles) has an MSE of 14,732. Of the meshes produced by recast, the mesh with the greatest number of triangles results in the smallest MSE. When the number of triangles increases, the MSE drops dramatically. The mesh with 2154 triangles has an MSE of 2259.

In most cases, the adaptive grid approach outperforms the triangle meshes. For instance, the S16 queue generation method results in 2190 clusters and an MSE of 208. When comparing the S16 queue with the triangle mesh having approximately the same number of triangles as clusters, the adaptive grid outperforms the triangle mesh 10-fold. The most similar adaptive grid example to Delaunay triangulation (in terms of number of clusters) uses single initial seed and stack data structure and has 479 clusters and an MSE of 6,958. The adaptive grid method produces much less error than triangular meshes that have approximately the same number of triangles as clusters.

5.3 *Application of Adaptive Grids on LIDAR Images*

Adaptive grids can also be applied on range images captured from real-world sources using technologies such as LIDAR. Figure 13(a) shows an example LIDAR image taken from Seattle, Washington.

Processing a LIDAR range image to generate a binary navigation grid that represents the navigable space is slightly different than processing depth images of virtual cities. First of all, LIDAR imaging is especially sensitive to refractive surfaces, such as glass. Range values may not be acquired from such surfaces, and hence, they are represented as black pixels in the image. As a preprocessing step, we use median filtering and morphological closing operations on the LIDAR image to remove these black pixels. Second, the distribution of range values over the image is not uniform. We use adaptive thresholding with a fixed window to separate roads from buildings. To clear away any small zones that are misclassified as buildings, we use morphological opening. The rest of the preprocessing follows the steps discussed in Sec. 3.1. Figure 13(b) depicts the navigable space obtained after these operations.

Finally, we apply the proposed algorithm on the binary navigation grid to obtain a rectangular partitioning of the navigable space. This partitioning has 757 clusters and is

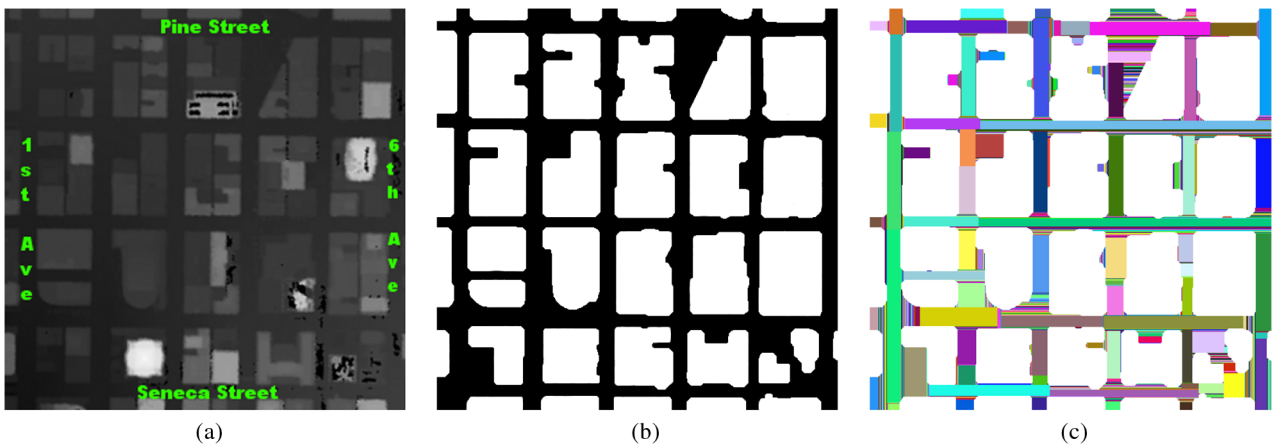


Fig. 13 Application of our approach on LIDAR images. (a) LIDAR image of the city of Seattle. Street and avenue names are overlaid to locate the blocks in the city that are captured in this image. (b) The navigable space obtained from the LIDAR image. (c) The generated adaptive partitioning. Different cell clusters are rendered in unique colors.

generated from a single initial seed using a one-dimensional, depth-first expander. We also construct the associated navigation graph. Figure 13(c) shows the generated partitioning. It is suitable for many applications of path planning, including traffic and pedestrian simulations.

5.4 *Crowd-Simulation Application*

Simulation results are obtained for an adaptive grid that has 784 clusters. This adaptive grid is generated from uniformly sampled initial seeds with a period of 32 pixels in both directions, and a depth-first, one-dimensional expander is used. The total memory consumed by the adaptive grid is 7.06 megabytes. This memory cost is derived for all possible 784×784 pairs. The memory cost can be significantly reduced by computing only the relevant paths. The average error on paths for this adaptive grid is 89 pixels for a grid of 512×512 pixel dimensions. These results include 2400, 4800, 9600, and 19,200 agents. Agents are periodically created on prespecified entry points within a virtual city and are assigned global tasks to reach a random entry/exit point. These entry/exit points are assigned to the building entrances and street exits. Still frames from an animation with 2000 agents are shown in Fig. 14.

Five character models (three male and two female) with varying numbers of polygonal complexity (between 2000 and 4000) are randomly assigned to agents upon instantiation. All agents have a wide set of skeletal animations (including idle, walking, running, talking, etc.) for realizing more interactive scenarios. During animation switches, all animations are linearly blended to support smooth transitions. Agents are removed from the simulation once they achieve their goals. As the successful agents are removed, the system maintains the maximum count of agents in the scene by injecting new agents. The removed agents are not destroyed but inserted into a pool (queue) so that they can be used for new injections. This agent-pooling approach

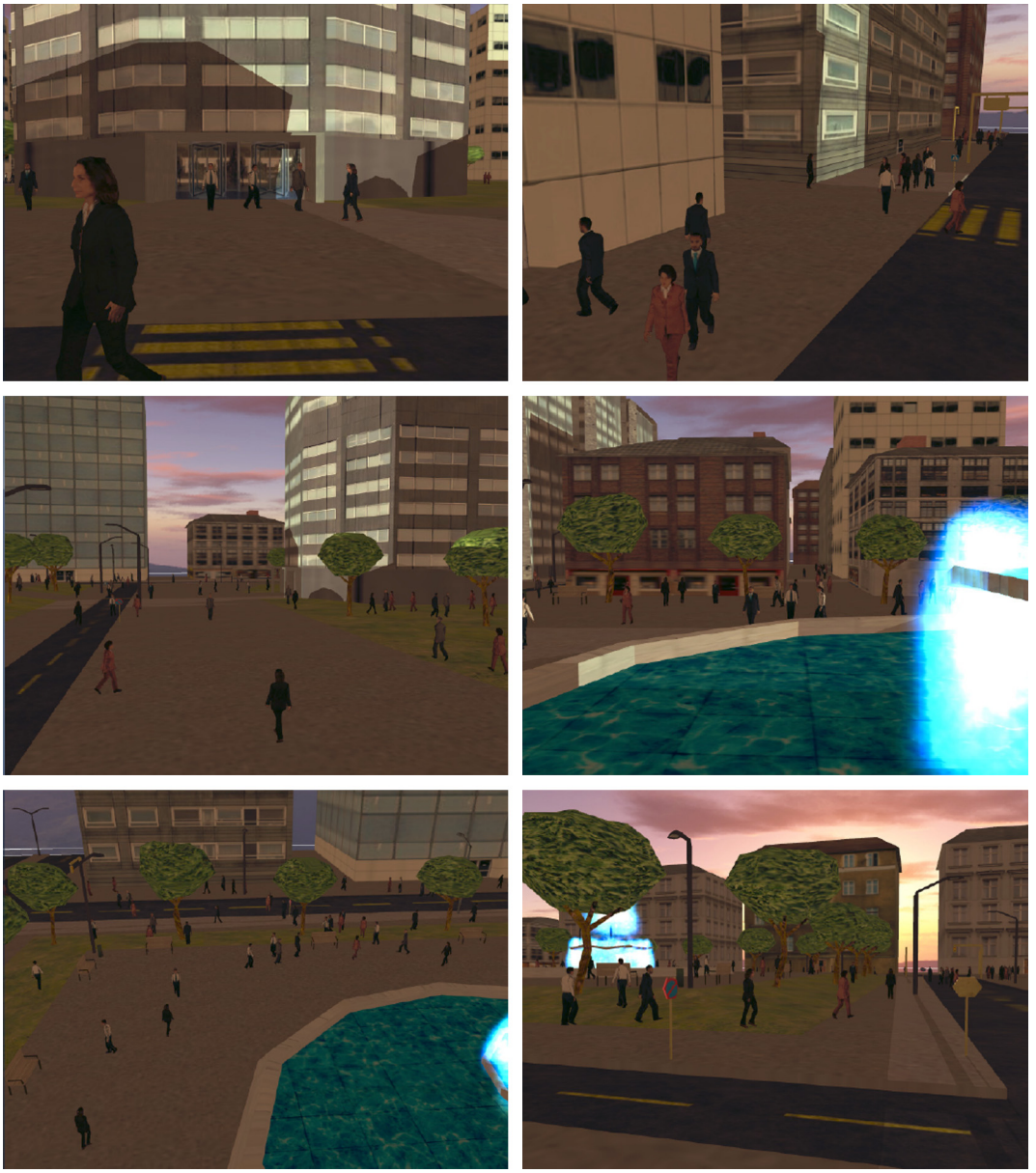


Fig. 14 Still frames from a crowd simulation in a virtual environment (Video 1, WMV, 14.3 MB) [DOI: http://dx.doi.org/10.1117/1.OE.52.2.027002].

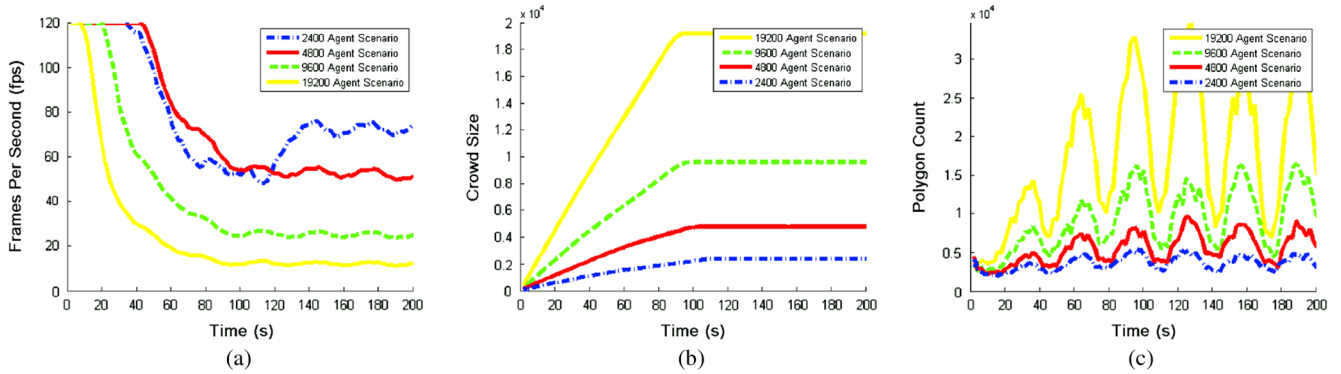


Fig. 15 Visualization statistics for different simulation scenarios.

significantly reduces the computational overhead of the new agent instantiation process. In this way, all agents within the system are instantiated only once.

The virtual city block is composed of 33 buildings and a number of other environmental objects, such as city lights, traffic signs, trees, and benches. These objects are recognized as obstacles by the agents and hence taken into account during local path planning for collision avoidance. The complete geometric model of the virtual city includes 18,342 polygons.

The tests are performed on a machine with an Intel Core i7 920 (8 MB cache, 2.67 GHz clock) processor, 6 GB RAM, 2 × ATI Radeon HD4890 graphics processing unit. Simulation statistics are evaluated for the four different scenarios, 2400-agent, 4800-agent, 9600-agent, and 19,200-agent. Imposter models and low-resolution textures are used for testing massive crowds. These scenarios are defined with respect to the maximum crowd size they permit. The statistics for each scenario include (1) the amount of polygons rendered, (2) average frames per second (fps), and (3) crowd size with respect to simulation time. All statistics are collected within a time period of 200 s. In all scenarios, maximum crowd size is achieved after 80 s. The results are given in Fig. 15.

Table 1 Average frame rates for different scenarios.

Agents (n)	2400	4800	9600	19,200
Frame rate (fps)	65.86	54.01	25.50	12.26

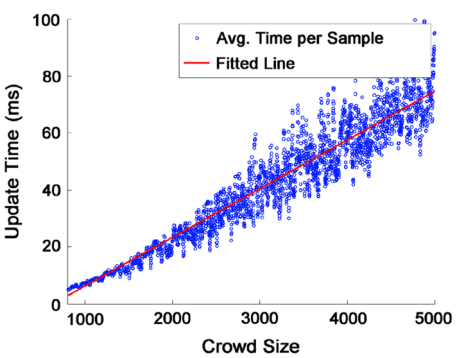


Fig. 16 Computational cost of agent updates.

The average frame rates for these scenarios are provided in Table 1, where it can be seen that the system operates at real-time rates for up to 9600 agents. It should be noted that, in the presented results, optimizations such as crowd-based occlusion culling are not applied.

For each frame, the accumulated computational cost for all agent updates is calculated with respect to varying crowd sizes. These values are expected to follow linear scaling as the simulated crowd size increases. Figure 16 gives the computational cost of agent updates, and a line is fitted to the graph.

6 Conclusions

We propose adaptive grids, a new image-based approach to construct a gridlike NavMesh, and an algorithm to construct the corresponding navigation graph, which facilitates constant-time global path planning for simulated virtual crowds. Adaptive grids addresses the shortcomings of previously proposed NavMesh construction algorithms. Adaptive grids can be configured to generate NavMeshes for both static and dynamic path planning, and it can be adapted to any application where path planning is of primary concern.

Considering the results when comparing adaptive grids with other triangular NavMeshes, adaptive grids offers better performance for static path planning. Adaptive grids also provides the following benefits over the traditional triangular meshes. When searching for an agent's cell location, adaptive grids quickly finds the cell by querying the grid, whereas bounding boxes and inclusion tests are needed for triangular NavMeshes. Adaptive grids is much simpler to implement compared to geometric NavMesh approaches. Adaptive grids is an image-based approach; hence, parallel computation of an adaptive grid is rather trivial. Graphics processing unit (GPU) implementations are possible, further improving the preprocessing time. Existing NavMesh approaches, such as the one proposed in Ref. 8, simplify the navigable geometry of the 3-D model (i.e., the surfaces with up-normals) until they achieve a minimal NavMesh. Processing time and performance of such approaches strictly depend on the capabilities of the artist who created the 3-D model (i.e., game levels or city models). Our approach, on the other hand, is image-based, where the axis-aligned range image of the model is independent from the model's topology. Thus, adaptive grids is a computationally cheap and easy-to-use approach when constructing a NavMesh.

Considering our simulation results, constant-time static path planning using the adaptive grids approach is feasible

in terms of realism, performance, and scalability. Many approaches in current literature restrict the behaviors of agents by either performing path planning only for agent groups (instead of individuals) or forcing the agents to follow strict navigation paths. Using adaptive grids for static path planning 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 routines per agent.

In addition, the proposed system need not store all pairs of shortest paths. In real-life scenarios, the preferred global goals of the pedestrians are within a finite domain. They are usually building entrances or specific locations, such as cafés, parks, etc. This observation justifies precomputation and storage of paths toward goals (in a finite domain) to support for $O(1)$ time complexity. The majority of the existing path-planning approaches in literature use graph search algorithms such as A^* to compute paths dynamically at run-time with greater precision and cost. Hence, dynamic global planning support versus computational complexity is one trade-off, which is open to debate, especially for massive crowds.

Although path planning on a two-dimensional terrain is demonstrated, our algorithm can easily be modified to handle 3-D spaces as well. In this case, the adaptive grid will be formed by 3-D axis-aligned rectangular prisms, and our algorithm should be extended to take into account up and down directions (in addition to east, south, west, and north) for the seed expansion process.

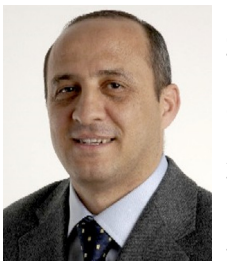
Acknowledgments

We are grateful to Miray Kaş and Rana Nelson for proofreading and suggestions. This research is supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) under Grant No. EEE-AG 112E110.

References

1. D. Helbing, I. Farkas, and T. Vicsek, "Simulating dynamical features of escape panic," *Nature* **407**(6803), 487–490 (2000).
2. T. I. Lakoba, D. J. Kaup, and N. M. Finkelstein, "Modifications of the Helbing-Molnar-Farkas-Vicsek social force model for pedestrian evolution," *Simulation* **81**(5), 339–352 (2005).
3. D. Helbing et al., "Self-organized pedestrian crowd dynamics: experiments, simulations, and design solutions," *Transport. Sci.* **39**(1), 1–24 (2005).
4. A. Lerner, Y. Chrysanthou, and D. Cohen-Or, "Efficient cells-and-portals partitioning," *Comp. Anim. Virt. Worlds* **17**(1), 21–40 (2006).
5. J. Pettre, J. P. Laumond, and D. Thalmann, "A navigation graph for real-time crowd animation on multilayered and uneven terrain," in *First International Workshop on Crowd Simulation*, pp. 81–90, Pergamon Press, Elmsford, New York (2005).
6. O. B. Bayazit, J. M. Lien, and N. M. Amato, "Roadmap-based flocking for complex environments," in *Proc. IEEE 10th Pacific Conf. on Comput. Graphics and Applications*, pp. 104–113, IEEE Computer Society, Washington, DC (2002).
7. S. Chenney, "Flow tiles," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 233–242, Eurographics Association, Geneve, Switzerland (2004).
8. G. Snook, "Simplified 3D movement and pathfinding using navigation meshes," in *Game Programm. Gems*, M. DeLoura, Ed., pp. 288–304, Charles River Media, Newton, Massachusetts (2000).
9. M. Kallmann, "Navigation queries from triangular meshes," in *Proc. 3rd Int0 Conf. on Motion in Games (MIG)*, pp. 230–241, Springer-Verlag, Heidelberg, Germany (2010).
10. M. Kallmann, H. Bieri, and D. Thalmann, "Fully dynamic constrained Delaunay triangulations," in *Geometric Modeling for Scientific Visualization*, pp. 241–257, Springer-Verlag, Heidelberg, Germany (2003).
11. P. Tozour, "Building a near-optimal navigation mesh," in *AI Game Program. Wisdom*, S. Rabin, Ed., pp. 298–304, Charles River Media, Newton, Massachusetts (2002).
12. J.C. O'Neill, "Efficient navigation mesh implementation," *J. Game Dev.* **1**(1), 71–90 (2004).
13. N. R. Sturtevant and R. Geisberger, "A comparison of high-level approaches for speeding up pathfinding," in *Proc. 6th Artificial Intell. and Interactive Digital Entertainment Conf.*, AAAI Press, Stanford, California (2010).
14. A. Sud et al., "Real-time navigation of independent agents using adaptive roadmaps," in *Pro. ACM Symposium on Virtual Reality Software and Technology*, pp. 99–106, ACM, New York (2007).
15. Y. Li and K. Gupta, "Motion planning of multiple agents in virtual environments on parallel architectures," in *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 1009–1014, IEEE, Piscataway, New Jersey (2007).
16. S. J. Guy et al., "PLEdestrians: a least-effort approach to crowd simulation," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 119–128, Eurographics Association, Geneve, Switzerland (2010).
17. A. Treuille, S. Cooper, and Z. Popović, "Continuum crowds," *ACM Transact. Graph. (Proc. SIGGRAPH)* **25**(3), 1160–1168 (2006).
18. J. Maïm, B. Yersin, and D. Thalmann, "Real-time crowds: architecture, variety, and motion planning," in *ACM SIGGRAPH ASIA Courses*, Course No. 56, IEEE, New York (2008).
19. D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Phys. Rev. E* **51**(5), 4282–4286 (1995).
20. M. Mononen, "Recast: navigation-mesh construction toolset for games," http://code.google.com/p/recastnavigation/ (January 2013).

Ateş Akaydın received his BS and MS degrees in computer engineering from Bilkent University, Ankara, Turkey, in 2007 and 2010, respectively. He is currently pursuing his PhD study at the Department of Computer Engineering, Bilkent University. His research interests include various aspects of computer graphics, including human modeling and animation, crowd simulation, visualization of complex graphical environments, virtual, and augmented reality.



Uğur Güdükbay received a BSc degree in computer engineering from Middle East Technical University, Ankara, Turkey, in 1987. He received his MSc and PhD degrees, both in computer engineering and information science, from Bilkent University, Ankara, Turkey, in 1989 and 1994, respectively. Then, he conducted research as a postdoctoral fellow at the University of Pennsylvania, Human Modeling and Simulation Laboratory. Currently, he is an associate professor at Bilkent University, Department of Computer Engineering. His research interests include various aspects of computer graphics, including human modeling and animation, crowd simulation, visualization of complex graphical environments, virtual and augmented reality. He is a senior member of both IEEE and ACM.