

# Environmental Modeling for Autonomous Virtual Pedestrians

Wei Shao and Demetri Terzopoulos

Media Research Lab, Courant Institute, New York University

Copyright © 2005 SAE International

## ABSTRACT

We present a technique for modeling large-scale urban environments for virtual human simulation. Our approach, which involves a set of hierarchical data structures, supports the efficient interaction between numerous autonomous pedestrians and their environment, including perceptual processing and path planning for the purposes of goal directed navigation. As a specific implementation of our approach, we develop an environmental model of a large train station and demonstrate its ability to support the real-time simulation of more than a thousand autonomous pedestrians engaged in a broad variety of individual and group behaviors appropriate to their large-scale urban environment.

## INTRODUCTION

The simulation of pedestrians is a difficult challenge that is beginning to capture the attention of researchers and practitioners in architecture and urban planning [6,12]. The field of computer graphics, in which virtual human animation has been an important research interest for decades [2,1,8], has contributed technologies fundamental to the computer-assisted visualization of planned architectural projects, including the automatic animation of pedestrians [3,7,9].

In our work, we have been developing an autonomous, self-animating model of pedestrians capable of performing a broad variety of natural activities in synthetic urban spaces. To this end, we have adopted a comprehensive artificial life approach to addressing the problem of pedestrian animation. Our approach is inspired most heavily by the work of Tu and Terzopoulos [14] on artificial animals and by Funge et al. [4] on cognitive modeling for intelligent characters.

The interaction between a pedestrian and his or her environment plays a major role in the animation of autonomous virtual humans in synthetic urban spaces. This, in turn, depends heavily on the representation and (perceptual) interpretation of the environment. Organizing the world into a simple list of geometric objects might suffice for modeling a small virtual environment with only a few environmental objects and

pedestrians, but it is grossly inadequate when attempting to synthesize a large urban space populated by numerous pedestrians, such as a busy train station.

In this paper, we develop a large-scale environment model, which includes a sophisticated set of hierarchical data structures that support the efficient interaction between numerous pedestrians and their complex virtual world through fast (perceptual) query algorithms.

In the remainder of the paper, we first describe our virtual environment model in detail. We then explain how this model is applied in the implementation of a large-scale virtual urban environment emulating the cavernous interior spaces of a computer reconstruction of New York City's original Pennsylvania Train Station, which opened to the public in 1910 and was tragically demolished a little over forty years ago.

## VIRTUAL ENVIRONMENT MODEL

In our system, we represent the virtual environment by a hierarchical collection of data structures, including a topological map, two types of maps for perception, two types of maps for path planning and a set of specialized environment objects (Fig. 1). With each of these data structures specialized to a different purpose, the combination is able to support accurate and efficient environmental information storage and retrieval.

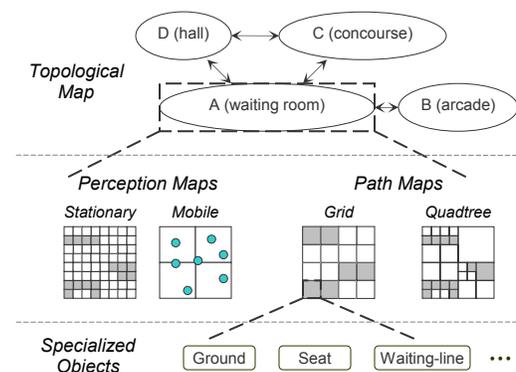


Fig. 1. Hierarchical World Model

## TOPOLOGICAL MAP

A graph serves to represent the topological relations between different parts of a virtual world. In this graph, nodes correspond to environmental regions and edges between nodes represent accessibility between regions. A region is a bounded volume in 3D-space (such as a room, a corridor, a flight of stairs or even an entire floor) together with all the objects inside that volume (for example, ground, walls, ticket booths, benches, vending machines, etc.). We assume that the walkable surface in a region may be mapped onto a horizontal plane without loss of essential geometric information, such as the distance between two locations. Consequently, a 3D-space may be adequately represented by several planar maps, thereby enhancing the simplicity and efficiency of environmental queries, as will be described momentarily.

Another type of connectivity information stored at each node in the graph is “path-to-via” information. Suppose  $L(A,T)$  is the length in number of edges of the shortest path from a region A to a different target region T, and  $P(A,T)$  is the set of paths from A to T of length  $L(A,T)$  and  $L(A,T) + 1$ . Then  $PT(A,T)$ , the “path-to-via” of A associated with T, is a set of pairs defined as follows:

$$PT(A,T) = \{ (region\ B, cost\ C_B) \mid \text{exists } p \text{ in } P(A,T) \ \& \ C_B = \text{length of } p \ \& \ B \text{ is next to } A \text{ on } p \}.$$

As the name suggests, if  $(B,C_B)$  is in  $PT(A,T)$ , then a *path* of length  $C_B$  from A to T via B exists. In other words,  $PT(A,T)$  answers the question “To which region shall I go, and what cost shall I expect if I am currently in A and want to reach T”? Given a graph, the “path-to-via” information is computed offline in advance using the following incremental algorithm:

Given  $G(N,E)$ , a graph with N nodes and E edges:

1. Initialization:
  - for each node A
  - for each target node T
  - if  $(A == T)$
  - then  $PT(A,T) = \{(A,0)\}$
  - else  $PT(A,T) = \{\}$
2. Collect information associated with paths of length L based on the information associated with paths of length L-1:
  - for length  $L=1$  to  $N-1$
  - for each node A
  - for each target node T
  - for every neighbor node B of A
  - if  $(X,L-1)$  is in  $PT(B,T)$  (Note: X can be any node in G.)
  - then add  $(B,L)$  in  $PT(A,T)$
3. Keep only low cost entries:
  - for each node A
  - for each target node T
  - let  $C_{min}$  be the minimal cost in  $PT(A,T)$
  - for each entry  $E(Y,C)$  in  $PT(A,T)$  (Y can be any node in G.)
  - if  $(C > C_{min} + 1)$
  - then remove E from  $PT(A,T)$

Note that after Step 3 only those entries are stored whose cost is minimal or (minimal + 1). In this way we can avoid paths with cycles. To understand this, consider  $PT(A,C)$  for the graph in Fig. 1. C is a direct

neighbor of A; so  $(C,1)$  is clearly an entry of  $PT(A,C)$ .  $(B,3)$  is another entry as A-B-A-C is also a possible path from A to C. Obviously, A-B-A-C is not desired as it contains a cycle. Such paths will automatically be removed from the “path-to-via” set after Step 3.

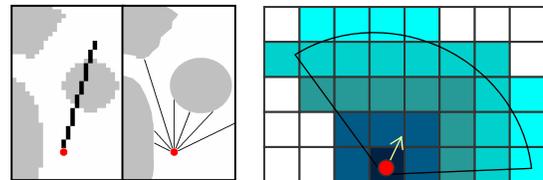
## PERCEPTION MAPS

Two types of maps support perception queries, one for stationary objects and one for mobile objects. The following table summarizes their similarities and differences, and the next two subsections present the details.

Type	#Maps	Cell size	Update cost	Query cost
Stationary	one per region	small (~0.3m)	0	constant, given the sensing range and acuity
Mobile	one per world	large (~5.0m)	linear in the number of pedestrians	constant, given the sensing fan and max number of sensed pedestrians

### Stationary Objects

Our definition of a region assumes that we can effectively map its 3D space onto a horizontal plane. By overlaying a uniform grid on that plane, we make each cell correspond to a small area of the region and store in that cell identifiers of all the objects that occupy that small area. Thus, the grid defines a rasterization of the region. This rasterized floor plan simplifies visual sensing. The sensing query shoots out a fan of line segments whose length reflects the desired perceptual range and whose density reflects the desired perceptual acuity. Each segment is rasterized onto the grid map (Fig. 2, left). Grid cells along each line are interrogated for their associated object information. This perceptual query takes time that grows linearly with the number of lines times the number of cells on each line. Most importantly, however, it does not depend on the number of objects in the virtual environment. Without the help of grid maps, the necessary line-object intersection tests would be time consuming given a large, complex virtual environment populated by numerous pedestrians. For high sensing accuracy, small sized-cells are used. In our simulations, the typical cell size of grid maps for stationary object perception is 0.2~0.3 meters.



**Fig. 2. Visual Sensing.** Left: Sensing stationary objects by examining map entries along rasterized eye rays. Right: Sensing mobile objects by examining (color-coded) tiers of the sensing fan.

## Mobile Objects

A 2D grid map is used for sensing mobile objects (typically other pedestrians). Rather than storing one map per region, this time a single global grid map suffices for the entire environment. In this map, each cell stores and updates identifiers of all the pedestrians currently within its area. The main purpose of the map is to enable the efficient query by a given pedestrian of nearby pedestrians that are within its sensing range. The sensing range here is defined by a fan as illustrated in the right part of Fig. 2. On the mobile object perception map, the set of cells wholly or partly within the fan are divided into subsets, called “tiers”, based on their distance to the pedestrian. Closer tiers will be examined earlier. Once a maximum number (currently set to 16) of nearby pedestrians are perceived, the sensing is terminated. This is intuitively inspired by the fact that usually people can pay attention at one time only to a limited number of other people, especially those in close proximity. Once the set of nearby pedestrians is sensed, further information can be obtained by referring to finer maps, by estimation, or simply by querying a pedestrian of interest. Given the sensing fan and the upper bound on the number of sensed pedestrians, this is a constant time operation.

## PATH MAPS

Goal-directed navigation is one of the most important abilities of a pedestrian, and path planning enables a pedestrian to navigate a complex environment in a sensible manner. To facilitate fast and accurate online path planning, we use two types of maps with different data structures—grid maps and quadtree maps.

### Grid Map

Grid maps, which are useful in visual sensing, are also very useful for path planning. Using the well-known A\* graph search algorithm [10,13], we can always find a shortest path on a grid map if one exists. In our system, grid path maps are used whenever a detailed path is needed. Suppose  $D$  is the direct distance between pedestrian  $H$  and his target  $T$ . Then a detailed path is needed for  $H$  if  $D$  is smaller than a user-defined constant  $D_{\max}$  and there are obstacles between  $H$  and  $T$ . This occurs, for instance, when one wants to move from behind a chair to its front and sit on it. Clearly, the accuracy of the path in this instance depends on the size of the cells in the grid path maps. A small cell size results in a large search space and, likely, low performance. However, detailed paths are usually not needed unless the target is very close to the starting point. Therefore, chances are that paths are found quickly, after the search covers only a small portion of the entire search space. Roughly speaking, the space that must be searched is bounded by  $4D_{\max}^2 / \text{cellsize}$  in most cases, and the typical values for those constants in our current system are  $D_{\max} = 5$  meters and  $\text{cellsize} = 0.2\sim 0.3$  meters.

## Quadtree Map

The quadtree map supports fast online path planning. Each quadtree map comprises a list of nodes, the number of different node cell sizes appearing in the map, and a pointer to an associated grid map with small cell sizes. Each node of the quadtree [11] stores information about its level in the quadtree, the position in the world of the region represented by the node, the occupancy type (ground, obstacle, seat, etc.), and pointers to neighboring nodes, as well as information for use during path planning, such as a distance variable (indicating how far the region represented by the node is from a given start point) and a congestion factor (the portion of the region of the node that is occupied by pedestrians).

As Fig. 3 illustrates, the algorithm for constructing the quadtree map first builds the list of map levels containing nodes representing increasing cell sizes, where the cell size of an upper level node is twice as large as that of lower level nodes. Higher level nodes, which aggregate lower level nodes, are created so long as the associated cells are of the same occupancy type, until a level is reached where no more cells can be aggregated. Usually quadtree maps contain a large number of lower level nodes which cover only a small portion of the entire region. Such nodes significantly increase the search space for path planning. Thus, in the final stage of construction, these nodes will be excluded from the set of nodes that will participate in online path planning. As the area that they cover is small, their exclusion does not cause significant accuracy loss. However, in occasions when path planning fails because of this exclusion, grid maps will be used to find the path, as will be described next.

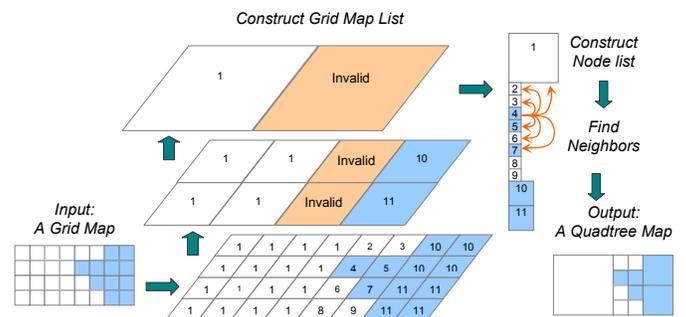


Fig. 3. Constructing a quadtree map.

### Combining the Path Maps

When used for path planning, quadtree maps have the advantage of quick response, while grid path maps are good for detailed plans. So quadtree maps are always used when detailed paths are not required. Sometimes, however, quadtree maps may fail to find a path even though one exists. This is because low level nodes on a quadtree map are ignored during path search in order to decrease the search space. Therefore, paths that must

go through some low level nodes cannot be found on a quadtree map. To resolve this, we turn to grid path maps whenever search fails on a quadtree map. For efficiency reasons, a list of grid maps with decreasing cell sizes is kept for each region. Gridmap path search starts at the coarsest level and progresses down the hierarchy so long as the search fails on coarser levels. Although grid maps with large cells are likely to merge separate objects due to aliasing, and thus cause the search to fail, the multiresolution search ascertains that, as long as the path exists, the path will always be found on maps with smaller cells. The following table compares the average performance between the two types of path maps, as measured in our experiments.

Type	Accuracy (smallest searchable node size)	Ground area covered	Typical search space size	% of all paths planned	% time used in a simulation
Quadtree	0.8m	85%	~500	94%	8%
Grid	0.5-1.0m	100%	~5000	6%	2%

## SPECIALIZED ENVIRONMENT OBJECTS

Many of the environment objects are specialized to support quick perceptual queries. For instance, every ground object contains an altitude function which responds to ground height sensing queries. A bench object keeps track of how many people are sitting on it and where they sit. By querying nearby bench objects, weary virtual pedestrians are able to determine the available seat positions and decide where to sit without further reference to perceptual maps. Other types of specialized objects include queues (where pedestrians wait in line), purchase points (where pedestrians can make a purchase), entrances/exits, etc. In short, these objects provide a higher level interpretation of the world that would be awkward to implement with perception maps, and this simplifies situation analysis for pedestrians when they perform autonomous behaviors.

## PATH PLANNING

Autonomous virtual pedestrians are capable of automatically planning paths around static and dynamic obstacles in the environment. When creating grid maps, special care must be taken to facilitate efficient updates and queries. Polygonal bounding boxes of obstacle objects represented on grid maps are enlarged by half of the size of a pedestrian's bounding circle. If the center of a pedestrian never enters this "buffer" area, collisions will be avoided. This enables us to simplify the representation of a virtual pedestrian to a single point, which makes most queries simpler. Each pedestrian executes a path planning algorithm whose main phases are as follows: 1) insert a target into the quadtree map and expand the target if necessary; 2) from a given start node, try to find any node of the expanded target using one of several available search schemes; 3) if the search reaches an expanded target node, then back-track through the visited nodes to construct an initial path back to the start node; 4) post-process this initial

path to compute the final path. In the case that a path cannot be found on a quadtree map, the standard A\* algorithm [10, 13] will be used to find a path on one of the grid maps, starting at the coarsest level and progressing down the grid map hierarchy, as we discussed in the previous section.

For quadtree maps, our search schemes employ several variations of the A\* search algorithm. In the conventional A\* algorithm, the search procedure iteratively gets an unvisited (ground) node from a queue, visits it, marks it as visited, adds its neighbors to the queue, and repeats until the target is reached or the algorithm fails to reach the target. As the algorithm progresses, it updates a distance variable in each node which indicates the approximate distance of the node from the start point. After the search succeeds, the distance tags of all the visited nodes form a distance field, which back-tracking uses to find a shortest path along the distance gradient from the target point back to the start point.

The A\* algorithm can be applied straightforwardly to plan paths on uniform grids. The situation with a quadtree map is more complex, however, as the nodes of a quadtree map can represent different sized regions. In updating the distance variables, we therefore employ the actual distances between the centers of adjacent nodes. The algorithm can update the distance variables efficiently since the internodal distances increase or decrease by a factor of 2 from level to level in the quadtree,.

Note that the centers of regions associated with large ground nodes in the quadtree are generally further from obstacles than those of the small ground nodes. When planning paths, virtual pedestrians can keep further away from obstacles simply by favoring bigger nodes in their path searches. The search algorithm does this by appropriately weighting the internodal distances from level to level in the quadtree.

The search algorithm also takes the congestion of an region into consideration during path planning. The congestion factor variables of adjacent nodes are incremented and decremented when a pedestrian transitions between the regions covered by these nodes. If the congestion factor for the node exceeds a preset congestion threshold, the internodal distances from that node are weighted with a weighting factor greater than unity. This associates a greater cost for congested nodes, in terms of their distance from a start node, and the search backtracking process will avoid that route.

We have designed several variants of the A\* algorithm, each with its own emphasis. Some variants aim at high speed while others always attempt to find shortest paths. The virtual pedestrians use all of the variants in order to increase the variety of their motions. Our A\* variants differ in how they maintain the unvisited nodes in the queue(s). Variant "SingleQ" is the standard A\* algorithm for uniform grid maps. It does not find shortest paths in quadtrees because it expands quadtree nodes non-

isotropically. “SortedQ” maintains a sorted queue, and thus it first visits the frontier node (frontier nodes are nodes in the boundary between visited and unvisited regions) which is closest to the start node. The expansion is more isotropic, but maintaining a sorted queue is relatively expensive. SingleQ tends to visit larger quantities of smaller nodes in the quadtree, since they are more plentiful; hence, the visited region tends to grow rather slowly. “MultiQ” maintains a separate queue for each level in the quadtree and it visits every level in turn; hence, more large nodes are visited sooner than in the SingleQ search, improving the search speed. “PmultiQ”, is a prioritized MultiQ scheme, which tries to visit the largest frontier node first and therefore exhibits the fastest growth speed among the four methods. If we consider the size of a node to be analogous to the size of a road, the PmultiQ scheme finds a path by searching along interstate highways first, then all state highways, then local roads, etc., until it finds the target. However, the path found will not necessarily be optimal. Thus, in the order in which the four schemes were presented (“SortedQ”, “SingleQ”, “MultiQ”, “PmultiQ”), the length of the paths that they generate are increasingly less optimal, but the searches are increasingly more efficient.

To make the search even faster, every target is expanded on a quadtree map until it touches any node at a certain level  $L_t$ . The value of  $L_t$  is a trade-off between accuracy and efficiency and it is automatically determined during map construction. Nodes on and above this level will cover a large portion of the entire region and there will be significantly fewer of them. Target expansion will likely shorten the time needed in the search step as nodes at levels lower than  $L_t$  probably do not have to be visited before the expanded target is reached (especially in the case of “PmultiQ”). “SortedQ” is used in order for the expansion to be nearly isotropic.

The search step successfully completes its task once it has found a node  $T_e$  that belongs to the expanded target. Using the aforementioned back-tracking method, the path planning algorithm constructs a path from the starting point to  $T_e$  within the searched area and a path from  $T_e$  to a real target node within the expanded target area. By linking these two paths together, we obtain a complete path to the target. Finally, the complete path will be refined for use by the virtual pedestrian.

## PERCEPTUAL PROCESSING

Our environment model efficiently provides accurate perceptual data in response to the perceptual queries that may be issued by an autonomous pedestrian. Typical queries are explained next in order of increasing abstractness.

**Sensing ground height:** To ensure that a virtual pedestrian’s feet touch the ground in a natural manner, especially when climbing stairs or locomoting on uneven ground, the pedestrian must query the environment model in order to sense the local ground height so that

the feet can be planted appropriately. Each grid map cell contains the height functions of sometimes a few (most often a single) ground objects. The greatest height at the desired foot location is returned in constant time.

**Visual sensing:** As stated earlier (see also Fig. 2), our data structures dramatically increase the efficiency of the sensing process when a pedestrian must perceive static obstacles and nearby pedestrians, which is a crucial component of obstacle avoidance. On the perception map for static objects, rasterized eye rays are used to detect static obstacles. On the perception map for dynamic objects, a constant number of neighbor cells are examined to identify nearby pedestrians. Both of the algorithms are localized and do not depend on the size of the world, the number of objects or pedestrians, or anything else that increases with world size.

**Planning paths between regions:** Here, the “path-to-via” information is useful in identifying intermediate regions that lead to the target region. Any intermediate region can be picked as the next region and, by applying the path-searching scheme, a path can be planned from the current location to the boundary between the current region and that next region. The process is repeated in the next region, and so on, until it can take place in the target region to terminate at the target location. Although the extent of the path is global, the processing is local.

**Locating an object:** Given a location identifier (say, “Track 9”), a search at the object level can find the virtual object. This is accomplished in constant time using a hash map with location names as keys. As the virtual object has an upward reference to its region, it can be located quickly (say, under the lower concourse), as can nearby objects in that region (say, “Platform 9” and “Platform 10”).

## PENN STATION MODEL

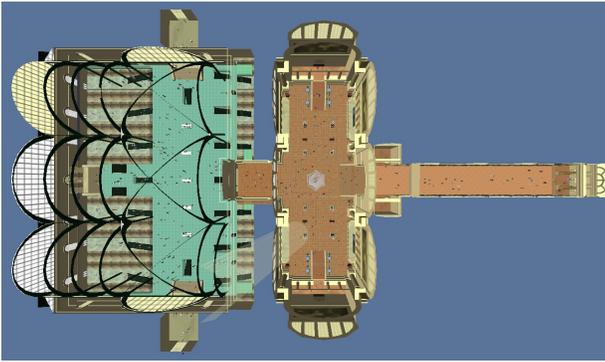


Fig. 4. Original Penn Station, NYC

Waiting Room Interior

We have applied the environmental modeling techniques described in the previous section to modeling the original Pennsylvania Station of New York City, as was mentioned in the introduction. Fig. 4 shows historical photographs of the exterior and interior of the station. The raw geometric model (with associated textures) of Penn Station is available in MultiGen-Paradigm Inc.’s OpenFlight file format. The model was distributed to us by Boston Dynamics, Inc.

Fig. 5 shows a roofless plan view of the rendered Penn Station geometric model with the two-level Concourse at the left, the main Waiting Room at the center, and the long Arcade at the right.



**Fig. 5.** Plan view of Penn Station, showing the concourse (left), the main waiting room (center), and the arcade (right). Note that the station model has a complete ceiling, which has not been rendered in this view.

## MAP INITIALIZATION AND UPDATE

The construction of the hierarchical map of a virtual environment requires first a manual division of the 3D space into regions. The initial graph map for the Penn Station model is constructed according to this division and the “path-to-via” information of every region is computed automatically. For each region, all the objects are automatically loaded and abstracted into the appropriate maps. For an obstacle, a polygonal bounding box is included, while for a ground object, a sampled height function is stored. Orientation is specified for objects such as chairs, newsstands, and vending machines. Special labels are stored for platforms/tracks and for trains to different destinations. A registered object also stores reference pointers to associated representations in coarser-resolution maps.

Mobile objects and stationary objects are stored in two separate grids, which form a composite grid map. Hence, objects that never need updating persist after the initialization step and more freedom is afforded to the mobile object (usually virtual pedestrian) update process during simulation steps. When a pedestrian in motion transits from one cell to another, the update process removes information from the previous cell and adds it to the new cell on the grid map. Since we have simplified a pedestrian’s representation to a single point, the update takes negligible time.

Fig. 6 presents a visualization of the quadtree map of the Concourse area of the station. The lower part of the figure also compares paths computed by the four search variants of the path planning algorithm discussed in the previous section.

## AUTONOMOUS PEDESTRIAN ANIMATION

Analogous to real humans, our synthetic pedestrians are fully autonomous. They perceive the virtual environment around them, analyze environmental situations, make decisions and behave naturally. Our autonomous human characters are structured in accordance with a hierarchical character model which, progressing through levels of abstraction up the “modeling pyramid”, incorporates appearance, motor, perception, behavior, and cognition sub-models [4]. The details of the perception, behavior, and cognition submodels are beyond the scope of this paper.

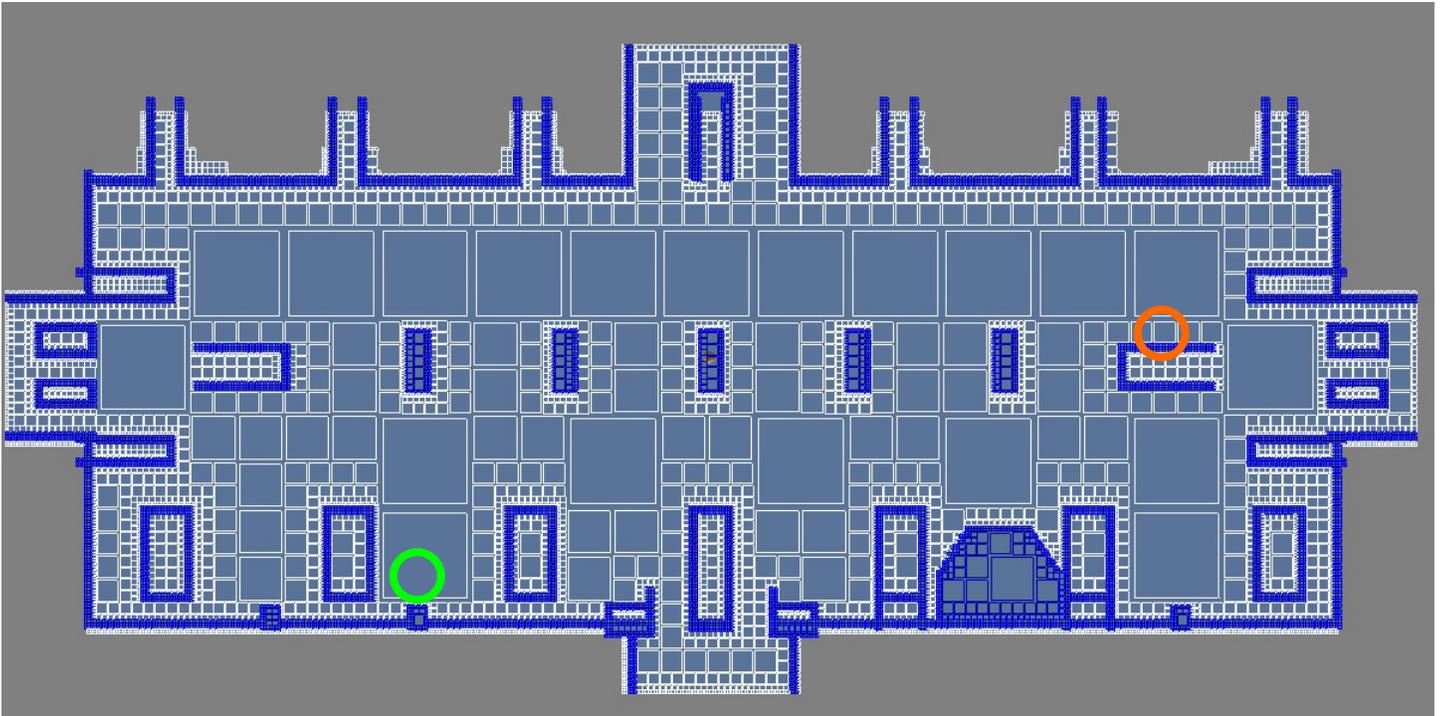
To implement the low-level appearance and motor levels, we employ a human animation package called *DI-Guy*, which is commercially available from *Boston Dynamics Inc.* [5]. *DI-Guy* provides a variety of textured geometric models of various people. These character models are capable of basic motor skills, such as walking, jogging, sitting, etc., implemented using IK and motion capture techniques. *DI-Guy* was intended as an application that enables users to script the actions of human characters manually in space and time. To facilitate manual scripting, it provides an interactive scripting interface called *DI-Guy Scenario*, which we do *not* use. However, it also provides an SDK that enables each character’s motor repertoire to be controlled by external user-specified C/C++ programs.

By supplementing *DI-Guy* with our own control programs at the perceptual, behavioral, and cognitive levels, we have successfully developed a fully self-animating virtual human model that is capable of synthesizing a relatively rich variety of *autonomous* behaviors and actions associated with pedestrians in urban environments.

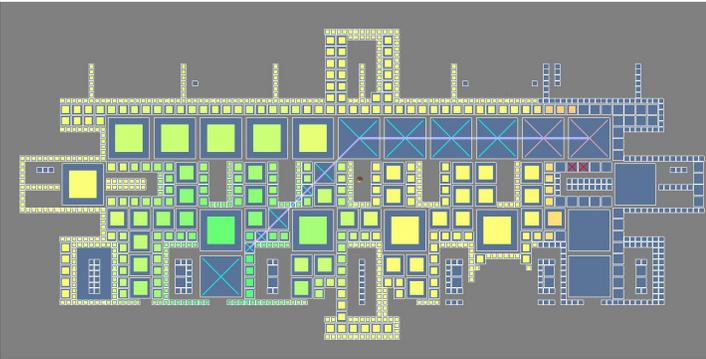
Fig. 7 illustrates frames from large-scale simulations of pedestrian behaviors in the main waiting room, concourse, and arcade areas of the Penn Station synthetic environment. Our simulations demonstrate not only conventional crowd behaviors, in our case involving over a thousand pedestrians, but various individual activities that are typical for pedestrians in a train station. These include pedestrians entering the station, navigating through portals, queuing at the ticket booths to purchase tickets and at vending machines when hungry/thirsty, sitting on chairs or benches when tired, congregating to watch a dance performance when they are interested in some amusement, and eventually proceeding to the train platforms down some rather narrow stairways.

## CONCLUSION

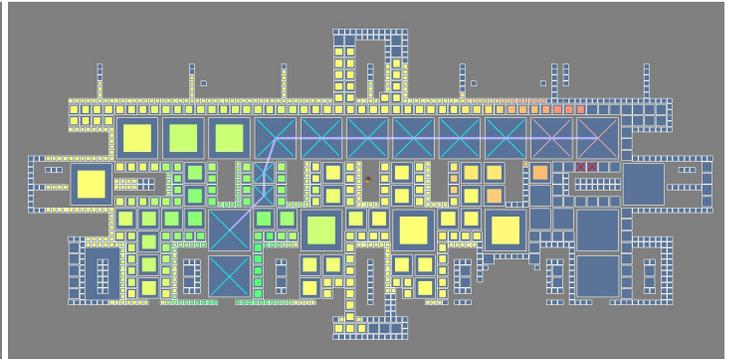
We have presented a methodology for modeling large-scale urban environments that facilitates the animation of numerous autonomous virtual pedestrians. In this paper, we have focused on the representation of indoor urban environments in terms of hierarchical data structures—a topological map, grid maps and quadtree maps—that support efficient path planning for pedestrian



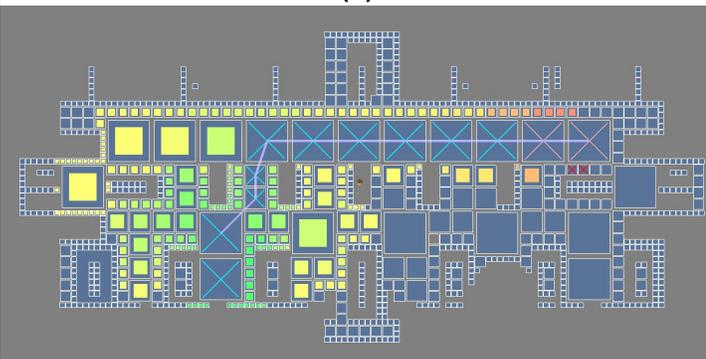
(a)



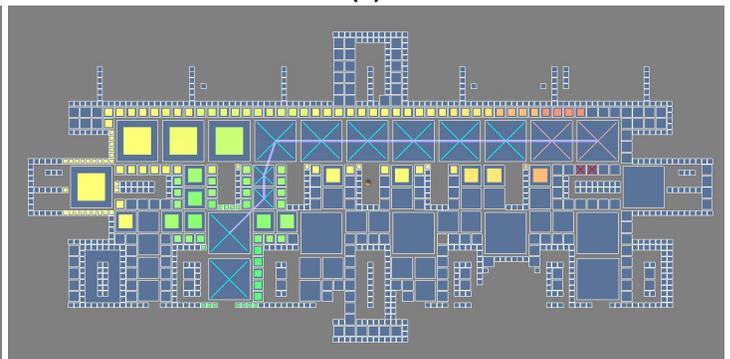
(b)



(c)



(d)



(e)

**Fig. 6.** (a) Visualization of the quad-tree map of the concourse's upper level in the Penn Station environment model. The white quads denote ground nodes and the blue ones denote obstacles. The green circle is the start point and the orange circle is the target. (b-e) Comparison of path planning algorithms on quad-tree maps. Paths planned by using (b) SortedQ, (c) SingleQ, (d) MultiQ, and (e) PmultiQ. The search space is color coded with the distance variable values increasing from green to orange. Note that, although the four paths are similar, the sizes of search space are different. (For clarity, obstacle quads are not shown in the lower images.)



Fig. 7. Waiting Room

Concourse / Platforms

Arcade

navigation guided by equally efficient perceptual processing.

Our simulations of autonomous pedestrians in a reconstruction of New York City's original Pennsylvania Station have included well over one thousand virtual pedestrians performing a reasonably broad repertoire of activities as autonomous *individuals*, not as "crowds".

It is important to note that our high-level autonomous control models are designed to be more or less independent of the lower-level human modeling/animation/rendering levels; hence, in principle, any other suitable low-level API can serve as a replacement for Boston Dynamics' DI-Guy API in our future work.

## ACKNOWLEDGMENTS

The research reported herein was supported in part by grants from the Defense Advanced Research Projects Agency (DARPA) of the Department of Defense and from the National Science Foundation (NSF). We thank Dr. Thomas Strat of DARPA for his generous support and encouragement. We also thank Mauricio Plaza-Villegas for his invaluable contributions to the implementation and visualization of the Penn Station environmental model and its integration with the DI-Guy software.

## REFERENCES

1. K. ASHIDA, S. LEE, J. ALLBECK, H. SUN, N. BADLER, AND D. METAXAS. 2001. Pedestrians: Creating agent behaviors through statistical analysis of observation data. In *Proceedings of Computer Animation*.
2. N.I. BADLER, C.B. PHILLIPS, AND B.L. WEBBER. 1993. *Simulating Humans: Computer Graphics, Animation, and Control*, Oxford University Press.
3. N. FARENC, S. MUSSE, E. SCHWEISS, M. KALLMANN, O. AUNE, R. BOULIC, AND D. THALMANN. 2000. A paradigm for controlling virtual humans in urban environment simulations. *Applied Artificial Intelligence*, 14(1):69–91.
4. J. FUNGE, X. TU, AND D. TERZOPOULOS. 1999. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. *Proceedings of SIGGRAPH 99*, pg 29–38.
5. J. KOECHLING AND A. CRANE AND M. RAIBERT. 1998. Applications of realistic human entities using DI-Guy. *Proc. of Spring Simulation Interoperability Workshop*, Orlando, FL.
6. G. G. LOVAS. 1993. Modeling and simulation of pedestrian traffic flow. In *Modeling and Simulation: Proceedings of 1993 European Simulation Multiconference*
7. R. METOYER AND J. HODGINS. 2003. Reactive pedestrian path following from examples. *Computer Animation and Social Agents*, pg 149-156.
8. S. MUSSE AND D. THALMANN. 2001. Hierarchical model for real time simulation of virtual human crowds. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):152–164.
9. H. NOSER, O. RENAULT, D. THALMANN, AND N. MAGNENAT-THALMANN. 1995. Navigation for digital actors based on synthetic vision, memory and learning. *Computers and Graphics*, 19(1)
10. S. RABIN. 2000 A\* Speed Optimizations. *Game Programming Gems*, Charles River Media. Hingham, USA. pp. 272-287.
11. H. SAMET. 1989. *Spatial Data Structures*. Addison-Wesley.
12. M. SCHRECKENBERG AND S.D. SHARMA (EDS.). 2001. *Pedestrian and Evacuation Dynamics*, Springer-Verlag.
13. B. STOUT. 2000. The Basics of A\* for Path Planning. *Game Programming Gems*, Charles River Media. Hingham, USA. pp. 254-263.
14. X. TU AND D. TERZOPOULOS. 1994. Artificial fishes: Physics, locomotion, perception, behavior. *Proceedings of SIGGRAPH 94*, pg 43–50, July.