ELSEVIER

Computational Geometry 15 (2000) 103–127

Computational Geometry
Theory and Applications

www. elsevier.nl/locate/comgeo

Path finding for human motion in virtual environments

Srikanth Bandi [a,*], Daniel Thalmann [b,1]

[a] *Electronic Imaging and Media Communications, University of Bradford, Bradford, West Yorkshire, BD7 1DP, United Kingdom*

[b] *Computer Graphics Lab, Swiss Federal Institute of Technology, CH-1015, Lausanne, Switzerland*

Abstract

This paper presents an efficient and robust technique for generating global motion paths for a human model in virtual environments. Initially, a scene is discretized using raster hardware to generate an environment map. An obstacle-free cell path sub-optimal according to Manhattan metric is generated between any two cells. Unlike 2D techniques present in literature, the proposed algorithm works for complex 3D environments suitable for video games and architectural walk-throughs. For obstacle avoidance, the algorithm considers both physical dimensions of the human and actions such as jumping, bending, etc. Path smoothening is carried out to keep the cell path as closely as possible to Euclidean straight-line paths.

Keywords: Path planning; Obstacle avoidance; Cellular paths; Virtual walk-throughs; Heuristic search

1. Introduction

In computer simulation of *goal-directed motion*, human models are made to move towards various goal locations in virtual environments. The motion is either *exploratory* or along a predefined path. In exploratory motion, movements of a human model are reactive, i.e., actions are responses to the environment it encounters. Goal directed motion is subject to several constraints of which the most important is obstacle avoidance. Another constraint is reaching the goal in an optimal manner meaning shortest path, least cost path, least dangerous path and so on. Generating motion paths that meet these constraints is useful in simulating human motion in virtual environments and video games [8]. Exploratory motion, by its very nature is reactive and suitable when the environment is not completely known. This method however performs only local search for obstacle avoidance and is unsuitable for producing globally optimal paths. The optimal paths can be procedurally or manually defined to make

* Corresponding author. Currently with Rushes Postproduction, 66 Old Compton St, London W1V 5PA, UK. Http://ligwww.epfl.ch/srik.html.

E-mail addresses: srik@rushesfx.co.uk (S. Bandi), thalmann@lig.di.epfl.ch (D. Thalman).

[1] Http://ligwww.epfl.ch/thalmann.html.

0925-7721/00/$ – see front matter
PII: S0925-7721(99)00046-2

the human reach the goal. Naturally this is very cumbersome and often it is not clear which path turns out to be optimal except for very simple scenes. For very complex environments automatic generation of paths is very desirable, but the presence of various types of obstacles makes the task non-trivial. Many existing techniques are therefore limited to 2D environments and may not be robust. They are also very inefficient when applied to 3D scenarios. This paper presents an efficient and robust technique for automatic generation of paths for goal directed motion of human models in completely known virtual environments.

Section 2 presents a review of path finding techniques. Sections 3 and 4 deal with environmental representation and a simple path finding technique in 2D. In Section 5 this technique is extended to more complex environments. Section 6 applies the algorithm for human motion. 3D environments are treated in Section 7. Section 8 deals with semantics of the environment, action encoding and moving obstacles. Section 9 concludes with some examples.

2. Literature review

Several sophisticated walking models have been developed by researchers using dynamic and/or inverse kinematics techniques over years for computer animation [7,12]. A predefined global path is necessary for applying these models. However, literature on automatic generation of global paths for human motion, in which the walking models could be realized, is inadequate. The problem of path finding is primarily studied in Robotics and Artificial Intelligence (AI) literature. One of the important techniques is *configuration space* approach [16,17]. All possible configurations of a moving object (position and orientation) are considered. Configurations which involve the object intersecting with any other objects in the environment are termed *configuration space obstacles*. A collision free path is computed by searching the regions of configuration space free of *configuration space obstacles*. The problem is, therefore, reduced to generating a collision-free path for a point object in a higher dimensional configuration space. This concept was used by Koga et al. [13] for arm motion. Bandi and Thalmann [4] simulate leg motion of human for walking in environments of obstacles. These techniques could be directly applied to any moving entities including human, but configuration space approaches are too slow for real-time applications. Lengyel et al. [15] discretize the configuration space obstacles (termed *C-space obstacles*) using raster hardware and generate discrete cell paths using dynamic programming techniques. Their technique works in real-time and is well suited for path finding in connected planar regions. Kuffner's approach [14] is along the similar lines but applies Dijkstra's graph search algorithm for finding optimal paths.

Human motion is simulated by Reich et al. [22] using sensor based navigation on terrains. The obstacles are avoided using potential fields: regions with obstacles generate high potential and others low potential. The human finds foot placements in regions of low potential field strength. Noser [21] used vision sensor based on Z-buffer for detecting obstacles in environments. A main disadvantage with potential field approach is that the human may get stuck in a local minima and may not reach the goal even if a path exists. Michael van de Panne [25] simulates motion of bipeds on a given track of foot prints. Obstacle avoidance is limited to stochastically changing foot locations to valid places upon detecting obstacles. Bandi and Thalmann [1,5,6] extend 2D discrete grid method to generate robust motion paths in 3D scenes for human navigation. Their approach addresses the problem of discontinuous regions and takes physical dimensions of human model into account for overcoming obstacles. Objects are, however,

merely classified as obstacles and non-obstacles without semantic information of the environment that may have bearing on the path generation. They also restrict the motion to walking only.

This paper presents a technique based on discrete 3D grid of object space. An efficient way of discretizing the space and path finding is described. The path is generated as a sequence of discrete cells and smoothened to 'mitigate' jaggedness of the cell path. Further, ways of incorporating semantics of the environment are considered and human actions such as jumping, bending, running, etc. are taken into account. The paper concludes with some examples and suggests plausible areas of improvement to make the algorithm more comprehensive.

3. Representation of the environment

Path finding in a scene requires a model or representation of the environment known as *environment map*. In Robotics literature, there are four basic types environment maps [18]: *path maps, free-space maps, object-oriented maps* and *composite space maps*. The algorithm and complexity of path planning is dependent on the type of the map. Each representation has its own advantages and disadvantages. Path maps specify a network of predetermined paths for movement of robots on a shop floor. Different sets of paths are required for different environments. Such a system is equivalent to manual path specification and is obviously very inflexible for human navigation. All other representations normally convert the environment into a scene graph. Free-space maps supply a list of obstacle-free regions represented as nodes in the scene graph. The paths generated tend to move along central regions of free spaces covering distances greater than required for reaching a goal. Object-oriented maps explicitly specify polygonal (or polyhedral in 3D) representation of obstacles that should be avoided. The main disadvantage is paths move too close by obstacles while turning at the corners. The scene complexity in both types of maps is expressed in terms of k and n representing number of polygons and vertices, respectively. Object-oriented maps are partial in that they record information either on free spaces or obstacles, but not both. Composite space maps, on the other hand, preserve both types of information. In these maps, the object space is subdivided into voxels in the form of a hierarchical tree or a uniform cell grid. If a voxel contains an object, it is marked occupied otherwise empty. In the oct-tree subdivision, each voxel is recursively subdivided upto a predefined depth, or until the voxel is completely occupied or empty. The tree structure can be skewed or balanced depending on the distribution of objects in the scene. In a nearly balanced oct-tree, depth of the tree represents the scene complexity. In uniform cell grid the scene complexity is represented by n, number of cells.

3.1. Choice of environment map for human navigation

Polygonal methods accurately model environments, but are computationally expensive for path finding. There are methods for finding shortest paths in $O(\log n)$ or in linear time, but pre-processing polygonal objects to achieve this performance runs into polynomial in n [19]. One common method is construction of *visibility graph* from polygonal models. The nodes of the graph are vertices of objects and edges are connections between pairs of vertices that can "see" each other without being obstructed by the other objects in the scene. Visibility graph can be searched for a shortest Euclidean path using A^* in $O(K + n \log n)$, K being number of edges in the graph. The worst case time complexity of construction

of visibility graph for a 2D scene is $O(n^2)$. In 3D the visibility graph construction runs into exponential time [24]. These considerations lead to the choice of composite space maps in the work presented here.

A hierarchical composite-space map is organized as a tree structure without any adjacency links between nodes. Reconstruction of adjacency graph structure during A* execution could be time consuming process. A uniform grid voxelizes the object space and further implicitly serves as an adjacency graph. Creation of uniform grid based environment map is therefore equivalent to discretizing the object space. The worst case time complexity of path finding in uniform grid is $O(n \log n)$. In general, an m-dimensional grid with r cells on each side takes $O(mr^m \log r)$ time [10]. The grid search is prohibitively expensive for dimensions greater than four, but is more efficient at lower dimensions. Using dynamic programming principle the grid search time can be reduced to $O(mr^m)$. That is, in a 2D grid with $n = r \times r$ cells, path finding takes only $O(n)$ time. For 3D environments, the search performance runs close to 2D levels since *surface cells* only are considered (see Section 3.5).

3.2. Discretization of environment

In real world, the entities that move, for example vehicles and people, are usually small and occupy small volumes compared to the whole scene. Objects such as buildings and furniture are static and comprise bulk of the scene. Discretization, the bottleneck of the algorithm, is therefore performed only once per scene. Dynamic moving obstacles are assumed to be rare and treated later in Section 8. One way of discretizing a 3D scene is by making use of 3DDDA originally developed for ray tracing [11]. 3DDDA is 3D extension of *Digital Differential Analyzer* (DDA) used in line drawing on raster screens [23]. Bandi and Thalmann [3] apply this technique for collision detection in identifying cells occupied by multiple objects in a uniform 3D grid. The technique involves rapid discretization of polyhedral objects in a scene using 3DDDA. In 3DDDA based discretization complete scene geometry is required. This may not always be available, if the scene is designed using various platforms such as Inventor, DXF, etc. Usually this is the case if complex scenes happen to be created by specialist designers who need not be aware of the internal requirements of the path finding algorithm. In order to keep the algorithm independent of details of scene construction, a frame buffer based discretization has been adapted. This is slightly different from the discretization method of Lengyel et al. [15] who apply this to C-space obstacles rather than to the obstacles in the environment directly.

In this paper, Y-axis is assumed to be the vertical axis and $Z-X$ plane the horizontal plane. The scene is enclosed in a tight-fitting bounding box and partitioned into various horizontal slices. Each slice is drawn on the screen using orthographic projection onto a window of resolution equal to the required resolution of the 3D grid in $Z-X$ plane. The number of slices represent the resolution along Y. The pixels drawn in the window indicate occupied cells in that slice. The pixel values are read from the framebuffer and corresponding occupied cells in the grid are marked.

The only requirement of separating path finding algorithm from scene discretization is the availability of a drawing function for the scene. On Silicon Graphics, for example, IRIS Performer graphics library can interpret various types of formats. High level functionality of providing a drawing function for discretizing scenes offers flexibility in designing them in various formats. On machines with dedicated graphics hardware, the discretization is very efficient. The drawing time increases with additional objects only linearly. This contrasts with quadratic or even polynomial complexity of building environmental maps in polygonal methods.

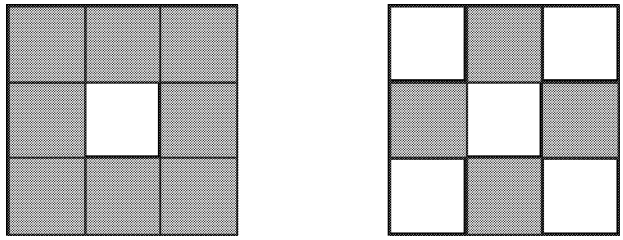


Fig. 1. 8-neighbors refer to all cells surrounding a central cell (left). 4-neighbors refer only to 4 orthogonal neighbors (right).

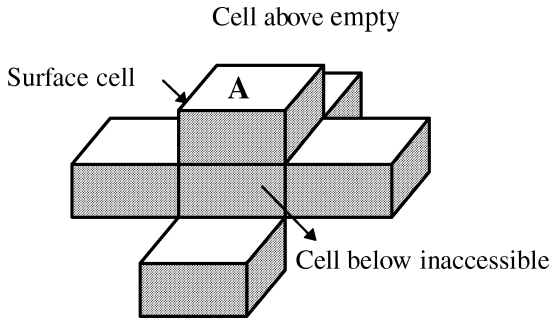


Fig. 2. Similar to 2D, a surface cell in 3D scene has a maximum of 8 immediate surface cell neighbors. In the figure only orthogonal neighbors are shown.

3.3. Cell dimensions

The cell dimensions along Z, X axes are same as the distance between two feet of a human standing upright in the normal position. The cell size is not a rigid requirement, but convenient for the purpose of avoiding obstacles using *border regions*. The vertical dimension along Y can vary according to requirement. There are as many horizontal slices as the Y resolution. Since discretization of a scene requires displaying slice by slice, very high Y resolution slows down the discretization process.

3.4. 4-neighbors and 8-neighbors of a cell

The central cell in Fig. 1 has eight neighbors: four orthogonal and four diagonal. In this paper, 4-neighbors refer to a cell's orthogonal neighbors only. The cell is said to be 8-connected when one can move to any of the 8-neighbors. Alternately, the cell is 4-connected if the movement is restricted to 4-neighbors. The connectivity influences the shape of the paths generated. A cell is assumed to be connected to any of its 8-neighbors by an arc of unit length. This means distances from diagonal and orthogonal neighbors are considered identical. This deviates from Euclidean metric, but makes path finding efficient. Section 7.4 presents techniques for rectifying distortions in paths resulting from these assumptions.

3.5. Surface cells

An occupied cell with an empty cell directly above is a surface cell as shown in Fig. 2. A surface cell comprises a maximum of 8 surface cell neighbors, which is same as in 2D. The figure shows a surface cell A and its orthogonal surface cell neighbors. The cell immediately below a surface cell remains inaccessible and can be discarded from the path search. A large number of interior 3D cells are thus eliminated making the algorithm very efficient.

3.6. Issues in path planning for human models

The path finding algorithm is executed using the grid structure discussed in the previous sections. Some of the features of the proposed method are:

- *Environments can be arbitrarily complex*. The environment can be in 3D and arbitrarily complex. As in many video games, all regions between which a human moves need not be physically connected (see Fig. 9). Real-life models can be used as typical scenes making the application suitable for virtual environments.
- *Robustness*. The algorithm is guaranteed to find a path, if one exists. The path is generated as a sequence of cells.
- *Optimal paths*. Paths generated between two cells measure no longer than the distance according to L_1 Manhattan metric and are often shorter. The cell paths are subjected to post processing to smoothen them in order to bring them closer to Euclidean straight line paths.
- *Encoding path cells*. The human model is expected to move on the path generated. Each cell in the path is encoded with an action the human should take upon reaching it: continue walking, jumping, bending, etc. This makes the human model adapt to specific local requirements of the environment.

4. Path finding

4.1. Simple bushfire path finding algorithm

Path finding between two cells in a discrete grid can be achieved using dynamic programming technique described by Lengyel et al. [15]. A is the start cell and B is the goal (Fig. 3). The discrete grid contains a hole region shown as the dark square. The algorithm works in two stages. In the first stage a *navigation front* is generated as diagonally aligned squares centered at A. This is achieved by generating all four orthogonal neighbors of A and entering into a queue. They form the first level of expanded cells and their enumeration levels are set to 1. Each time a cell is de-queued and expanded into its four neighbors. If these new cells are not enumerated in earlier iterations then they are appended to the queue and their enumeration levels are set to be 1 more than that of their parent cell. The cells lying in the hole region are discarded and never entered in the queue. The cells are expanded in breadth-first manner as a series of concentric squares navigating outwards from A. Each square passes through the cells at the same level of enumeration. Each layer in the navigation front is therefore numbered in increasing order as shown in the figure. The iteration continues until the queue is empty or the goal cell is reached. This method of enumerating cells is a computer graphics technique used in filling closed regions with irregular non-polygonal boundaries and holes and is also known as floodfill algorithm as the region is flooded with enumerated pixels.

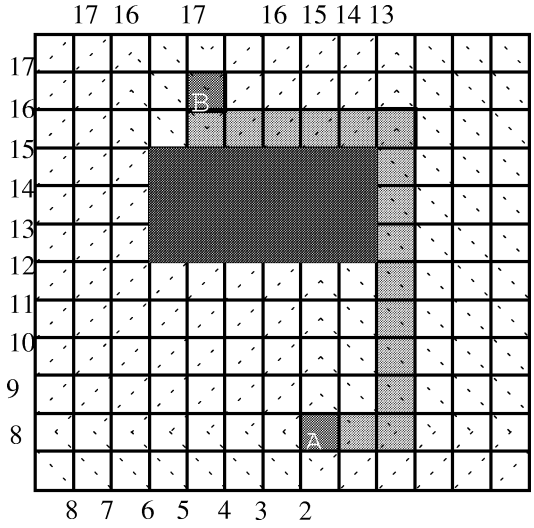


Fig. 3. Application of simple bushfire technique for path finding. Navigation front expands gradually from cell *A* in concentric layers of squares aligned diagonally. The numbers indicate the iteration in which cells on a layer are enumerated.

In the second stage, a path is retraced from the goal cell *B* to *A*. The cell *B* is entered in the path list. Then the enumeration levels of all its four neighbors are examined. The cell with the lowest level is selected. If there are multiple cells at the same level, a cell is randomly chosen and entered in the path list. For example, cell *B* has left, right and lower cells at the same enumeration level 15. The path generation selected the lower cell in this case. At any instance the four neighbors of the last cell in the path are examined and the navigation front is traced backwards. Since the navigation front expanded outward starting at *A*, the first cell will be eventually reached forming the complete path. The path generated is shown in Fig. 3. The navigation front reaches closer cells earlier than the farther ones. This means, once a cell is enumerated, its shortest Manhattan path from the start cell is automatically defined.

4.2. Drawbacks of the technique

The simple path generation technique suffers from several shortcomings. The navigation front takes into consideration only orthogonal neighbors generating Manhattan paths, which are unnatural for human motion. Kuffner's [14] method improves upon this by permitting diagonal search and considering all 8-neighbors while expanding a cell. The algorithm however uses Dijkstra's graph search which requires quadratic time compared to linear time in Lengyel's method. These techniques are also limited to planar environments, making them unsuitable for realistic 3D scenes. The simple bushfire enumerates a large number of cells before reaching the goal thereby reducing the over all speed. The regions occupied by start and goal cells need to be connected by contiguous cells in order to generate navigation front between them using bushfire technique. These restrictions limit the scope and application of the technique to human motion in virtual environments. In this paper several improvements to the basic algorithm are proposed before applying it for human motion.

5. Rectifying bushfire technique

5.1. Multiple goals

The basic algorithm can be easily extended to accept multiple goal cells. The navigation front terminates after reaching any of the goal cells and the path is retraced. In this case, a path to the closest goal cell is generated. If paths to all the goal cells are needed, navigation front continues until all the goal cells are enumerated. Multiple goal cells are useful if the human goal is a region spanning several cells, rather than a point location in a single cell.

5.2. Distance metrics and path characteristics

The Euclidean distance between two points $P(px, py)$ and $Q(qx, qy)$ in a Cartesian plane is given by L_2 metric:

$$\text{Euclidean-distance}(P, Q) = \sqrt[2]{(px - qx)^2 + (py - qy)^2}.$$

If the locations are two cells from a uniform grid, then the distance according to L_1 or Manhattan metric is

$$\text{Manhattan-distance}(P, Q) = |px - qx| + |py - qy|.$$

Let $dx = |px - qx|$ and $dy = |py - qy|$. Then the number of cells in the shortest path between P and Q is given by

$$d_4(P, Q) = dx + dy + 1.$$

The equation computes a 4-path, meaning the path has only orthogonal moves along 4-neighbors. The 4-path need not be unique. The number of shortest 4-paths is given by [9]

$$\frac{(dx + dy)!}{dx!dy!}.$$

Fig. 4 shows locus of equidistant cells according to Manhattan (left) and Euclidean (right) distance metrics. The former produces diagonally aligned concentric squares and the latter produces concentric circles.

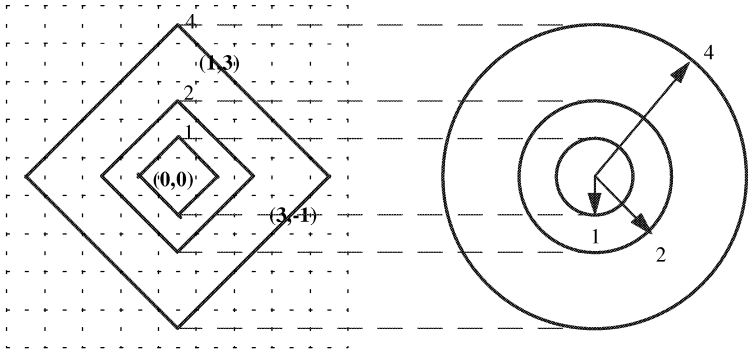
If 8-neighbor moves are allowed, an 8-path of following length will be generated:

$$d_8(P, Q) = \text{maximum}(dx, dy) + 1.$$

If $dx \geqslant dy$ and the 8-path is confined to a rectangular box with P and Q on diagonally opposite corners, then the number of shortest 8-paths is

$$\frac{dx!}{(dx - dy)!dy!}.$$

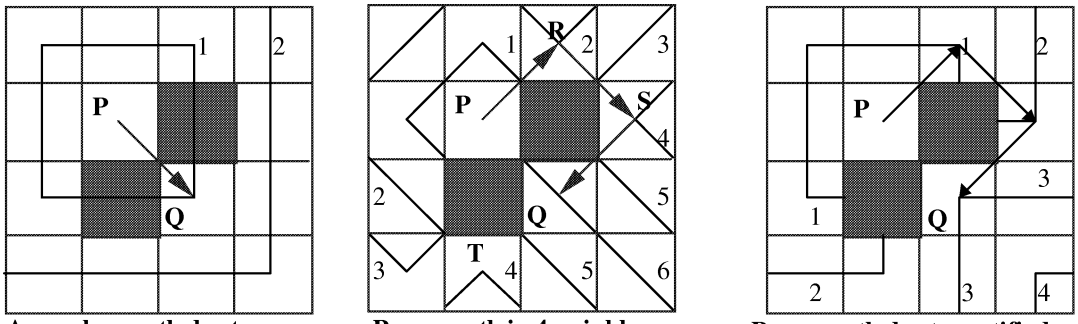
The roles of dx and dy are reversed if $dx < dy$. The locus of equidistant cells also form concentric squares, but are aligned along coordinate axes. It is to be noted that $d_8(P, Q) \leqslant d_4(P, Q)$, i.e., 8-path is the shortest path between the cells. They can be produced by generating 8-neighbor navigation front followed by tracing back the path across all 8-neighbors.



***L1* Manhattan distance: Locus of equidistant cells lie along concentric squares**

***L2* Euclidean distance: Locus of equidistant points lie on concentric circles**

Fig. 4. Distance metrics.



Anomalous path due to 8-neighbor graph connectivity

Proper path in 4-neighbor graph connectivity

Proper path due to rectified 8-neighbor graph connectivity

Fig. 5. Paths in 4 and 8 neighbor connectivity. In the left most picture path from P to Q passes through the obstacles. Corrections can be made using 4-neighbor (middle) or modified 8-neighbor navigation fronts (right).

8-paths, however, could produce illegal paths as shown in Fig. 5. The navigation front starts from P and enumerates cell Q at level 1 (left picture). The path is traced back from P to Q going in between the obstacles shown as dark cells. Often this results in collision with the obstacles except for point objects. One solution to this problem is to generate a 4-neighbor navigation front and then apply 4-neighbor or 8-neighbor back tracing of the path. In 8-neighbor back tracing, the obstacles should be taken into account while moving diagonally in order to prevent movement from Q to P. The navigation front enumerates the cell Q at 6th level resulting in the correct path as shown in the middle picture. 8-neighbor navigation front could still be used if diagonal obstacles are identified during generation of navigation front itself. If the line from current cell (P) to a diagonal 8-neighbor (Q) is flanked by cells with obstacles as shown in this figure, then the 8-neighbor (Q) is not enumerated from the cell. The tracing back of the path using 8-neighbors generates collision free path.

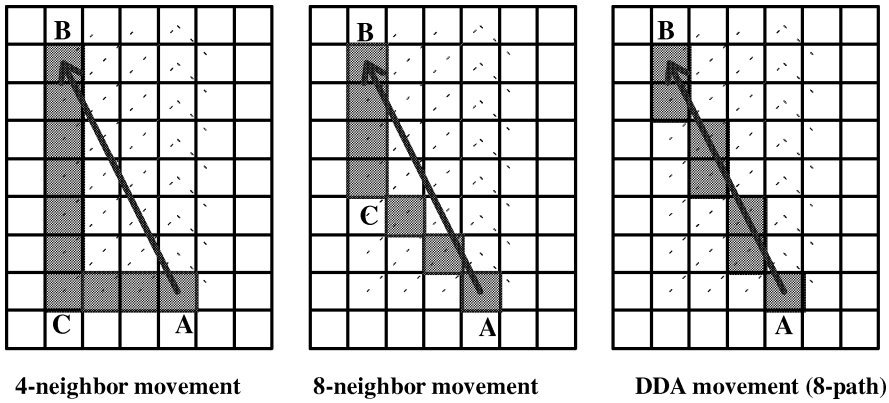


Fig. 6. Path generation between two cells in various methods. DDA path is an 8-path generated along the Euclidean straight line path.

5.3. Examples of 4-path, 8-path, DDA path

Fig. 6 shows an example of cell paths constructed using 4-neighbors (left picture) and 8-neighbors (middle picture) between cells A and B. It is clear from the figure that 4-distance and 8-distance metrics are at variance with the Euclidean metric between the cells. Even if the cell path is optimal according to L_1 (Manhattan) metric, it falls short of being one according to L_2 (Euclidean) metric. This anomaly creates potential distortions in the cell path, that is, unnatural paths are created for human motion. One solution to this problem is to select the cell path closest to Euclidean straight line. *Digital Differential Analyzer* (DDA) enumerates all cells lying on a straight line connecting two cells. Right picture shows cells enumerated along line AB using DDA. This generates shortest cell path that lies closest to Euclidean path between any two given cells. The example here shows an 8-path but the DDA implementation can be altered to generate 4-paths as well.

5.4. A heuristic reduction of number of enumerated cells

The bushfire algorithm presented in Section 4 enumerates all the cells in the planar grid. A* graph search algorithm can be applied to reduce number of enumerated cells [20]. The uniform grid can be thought of as a special graph with rectilinear connections. A* algorithm sorts the cells to be expanded in a queue according to the following cost function and selects the best cell:

$$f(m) = g(m) + h(m), \quad g(m), h(m) \geqslant 0,$$

where $f(m)$ denotes the cost of cell m. The cost has two components. g is minimum cost of reaching m from the starting cell so far and h is the heuristic cost of reaching the goal from m. The cost function therefore gives an optimum estimate of reaching the goal from the starting cell via cell m. In each iteration, A* expands the cell with the least f value into its neighbors. If h is a lower bound on the actual cost between m and the goal, then the algorithm expands fewer cells than the breadth-first simple bushfire technique, but produces an optimal path. The heuristic can be different for 4-neighbor and 8-neighbor navigation fronts. The lowest cost between cells cannot be less than the 4-distance in case of the former,

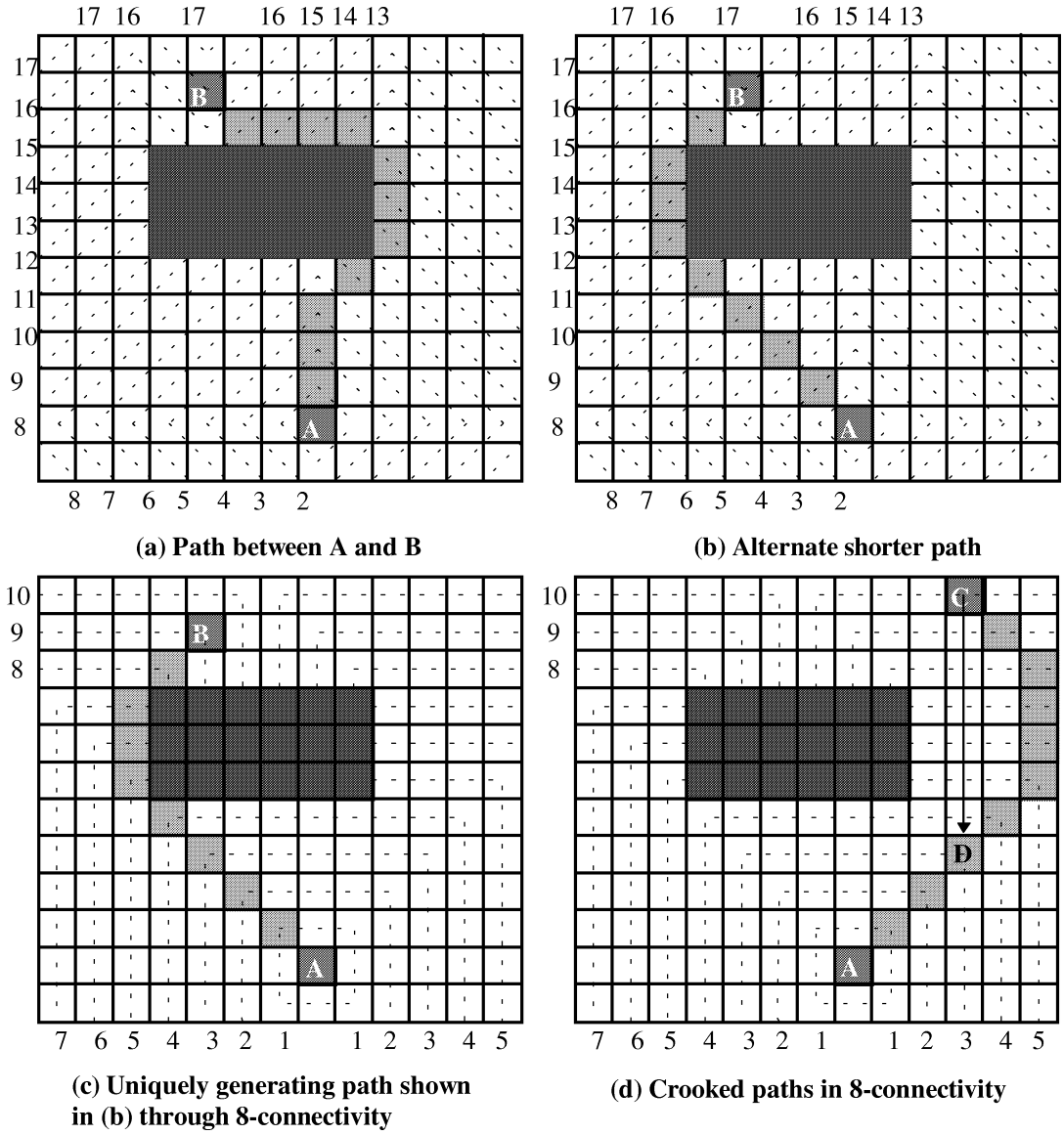


Fig. 7. Paths generated in 4-neighbor navigation fronts (top) and 8-neighbor navigation fronts (bottom).

and 8-distance in case of the latter. Application of the heuristic substantially reduces the number of cells expanded.

5.5. 8-neighbor, 4-neighbor navigation fronts, rectifying crooked paths

As mentioned earlier, 8-paths result is shortest paths in terms of number of cells. They are produced by generating 4-neighbor or 8-neighbor navigation fronts and then tracing back across 8-neighbors.

Fig. 7 exposes shortcomings of the method. An optimal path is required between cells A and B in a uniform grid containing a hole region (dark cells). The top left picture shows an 8-path after generating 4-neighbor navigation front. This is produced by starting at B, selecting its lower right neighbor (at level 14) as the next cell and then proceeding until A by the right side of the hole. The path measures 13 cells. This is, however, not the shortest path. On selecting the lower left neighbor (also at level 14) of B as its next cell, the path in the upper right picture will be generated. The path lies to the left of the hole and measures only 10 cells which happens to be the shortest distance between A and B. Optimal Manhattan 4-paths, if generated, would measure 17 cells irrespective of the direction taken at B. For 8-paths, however, the choice of neighbor turns out to be crucial in producing optimal paths. This dependency can be apparently removed by generating 8-neighbor navigation fronts for 8-paths as shown in the lower left picture. This correctly identifies cells in the shortest path. 8-neighbor navigation fronts however are not immune from the choice of neighbors either. Consider an optimal path between cells A and C in the lower right picture. The path generated after tracing back the navigation front from C to A is shown in the picture. At each cell, the lower right diagonal cell is tested first for lower levels of navigation front. As a result, a crooked path, which is an optimal according to d_8 metric, is generated. This would deviate substantially from a plausible Euclidean shortest path between A and C.

The choice of cell neighbors in 8-neighbor navigation fronts can be resolved in the following way: in case of a tie between diagonal and orthogonal neighbors (i.e., at the same level of navigation front), the latter should be given priority while tracing back the path. In the example cited above, at cell C, the lower orthogonal cells are continuously selected until the path reaches D from where it heads towards A in diagonal fashion. 8-neighbor navigation front combined with back tracing with orthogonal priority ensures optimal paths.

6. Application to human model

6.1. Human dimensions

In path finding, physical dimensions of human model are important parameters. Whether the model can or cannot overcome an obstacle depends on its physical capabilities. For example, a child or a short human model cannot cross wide gaps an adult can. A short model, however, can pass through low elevations, for example, tunnels, more easily than a tall one. *Horizontal foot span* defines maximum number of cells a human could cross in a single step along Z or X axis both of which form the horizontal plane. Similarly, *vertical foot span* can be defined as the maximum number of cells human model could ascend or descend in a single step along Y-axis. These parameters can take higher values if the human is allowed to jump, run, etc., i.e., actions required to be performed under different circumstances or environments. Another parameter *height* could be expressed in terms of number of cells human occupies in vertical direction. *Height* also can take minimum and maximum values if the human is allowed to bend or jump according to situation.

6.2. Reachability and extended neighbourhood

A typical 3D scene can have staircases, obstacles, gaps that should be taken into consideration during path planning. As mentioned earlier, the simple bushfire algorithm works in two stages (see Section 4).

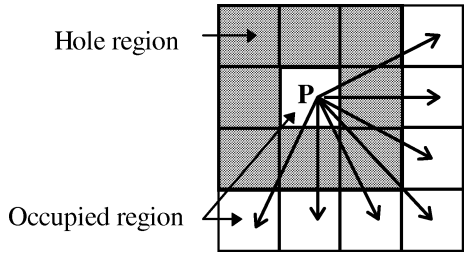


Fig. 8. No path exists between P and rest of the occupied region when simple bushfire is applied. Arrows point to extended neighbors.

A bushfire is applied to enumerate cells avoiding hole regions. One can consider holes as *unreachable* regions and the rest as *reachable*. In the simple algorithm reachability is equivalent to *contiguity*. This means that a path between two cells is made up of contiguous cells. A path does not exist between regions separated by a gap. Such a restriction prevents human characters in computer games jumping from one platform to another separated by a distance. Consider cell P in Fig. 8. A hole region (dark cells) separates the cell from the other occupied cells (white). The simple algorithm fails to establish a path between P and rest of the occupied region.

Extended neighbourhood

If a cell grid contains hole regions, some cells will have a few or all of their immediate 8-neighbors missing as shown in case of the cell P in Fig. 8. P is completely surrounded by a layer of hole cells. By extending neighbourhood along the directions of missing cells, new occupied cells could be found. The arrows in the figure show extensions of neighbourhood beyond immediate vicinity of P in order to find new neighbors. The concept of extended neighbourhood is applied in generating navigation fronts between completely disconnected regions, provided a few cells from the regions are reachable from each other (see Section 7.1).

Reachability

Humans are usually capable of manoeuvring limited discontinuities in any environment. A human placed at P, for example, may step across the hole region to reach the other side. In such cases the human should be able move by more than a single cell at a time. For larger holes crossing may prove difficult or impossible. Reachability, therefore, depends on both contiguity of the region and human capability to surmount physical obstacles. Overcoming obstacles takes the form of stepping across gaps as well as jumping, bending, etc. The reachability should comprise all these abilities of which crossing contiguous cells is only a special case.

In the context of human motion, two occupied cells $\boldsymbol{p}$ and $\boldsymbol{q}$ are reachable from each other if 4-distance d_4 between them is less than or equal to *horizontal foot span*. If the cells are separated by a hole region wider than *horizontal foot span*, they become unreachable from each other. In case of 3D scenes, apart from satisfying the 2D criterion, reachable cells should not be separated along Y direction by more than *vertical foot span*. Any reachable cell must also have enough vertical room for the human to stand upright or bend.

If a pair of cells lie within horizontal and vertical foot spans of each other they are *immediately reachable*. Suppose $\boldsymbol{p}$ is immediately reachable from $\boldsymbol{q}$ and $\boldsymbol{q}$ from $\boldsymbol{r}$. $\boldsymbol{p}$ is then considered reachable

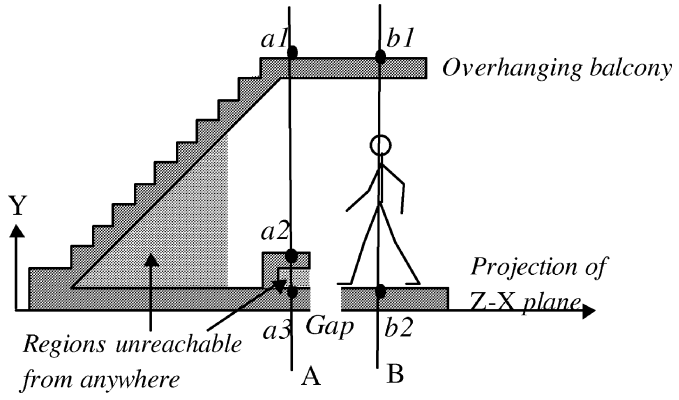


Fig. 9. A 3D scene with reachable and unreachable regions. A gap separates $a2$ and $b2$.

from $\boldsymbol{r}$ even though it need not be immediately reachable. The closure of all reachable cells forms a *reachable region*. Several chunks of disconnected regions can merge into a single reachable region if the cells satisfy the reachability criterion. There exists a path between any two cells within a reachable region. The human cannot reach any location outside this region.

7. Extension to 3D scenes

In 2D environments, the foot location in vertical direction is a single valued function of horizontal plane. For each cell there is a unique set of 8-neighbors which simplifies expansion of the cell into its neighbors. In 3D scenes, a vertical line through a point on horizontal plane can cut through multiple locations. Fig. 9 shows a human near a staircase connected to an overhanging balcony. Such overhangs are not permitted in 2D terrains. Line A intersects three surface cells at $a1$, $a2$ and $a3$ and line B intersects at $b1$ and $b2$. A human model positioned at $b2$ and moving forward therefore needs to choose from one of the three cells along the line A. The separation between $a1$ and $b2$ exceeds vertical foot span and at $a3$ there is no sufficient standing room for human. Cell $a2$ is separated from $b2$ by a gap but is within horizontal and vertical foot spans. The cell at $a2$ is therefore immediately reachable but those at $a1$ and $a3$ are not. It should be noted that even though $a1$ is not immediately reachable from $b2$, it is eventually reachable via staircase. In this case both the cells are in the same reachable region. Since moving from $b2$ to $a2$ involves crossing a gap, neighbourhood of cell at $b2$ should be extended. The region under the staircase cannot accommodate human (assuming bending is not permitted) and is considered unreachable.

7.1. Path finding in 3D

By using the concepts of extended neighbourhood and reachability, the simple bushfire algorithm can be applied to complex 3D scenes. The two stages of the algorithm need to be executed in the modified version. In the first stage, a navigation front consisting of only *reachable* cells is generated. The start cell is entered into a queue. In each iteration first cell is de-queued and expanded into eight reachable neighbors. Reachable cells of a current cell are normally its contiguous neighbors. In their absence,

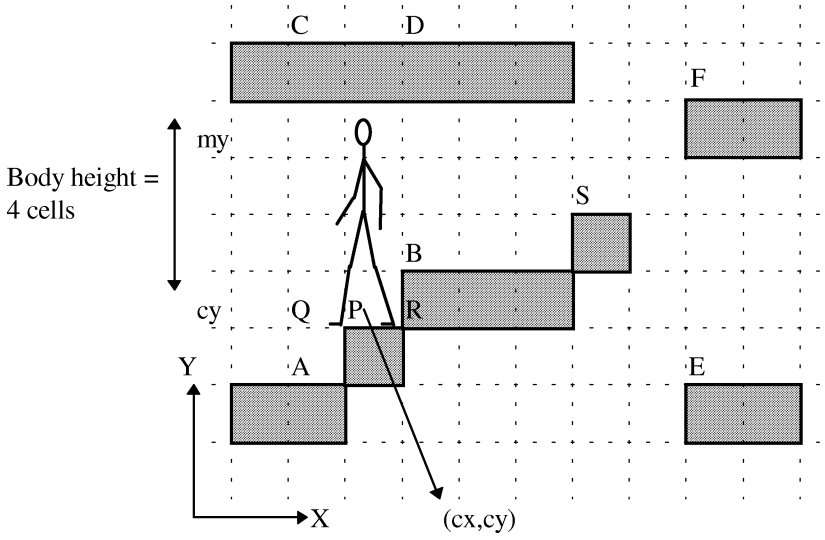


Fig. 10. Computing extended neighbors.

extended neighbors, if any, are considered and tested for reachability in order to eliminate cells which are too far away or too tall for human model to overcome. This method expands each cell into only the first layer of reachable cells surrounding it, but not all the reachable cells. The problems arising from this approach are discussed in the next subsection.

In the second stage, path is traced backwards across the navigation front. In the simple bushfire technique, navigation front spreads in layers of contiguous cells. In the modified version, contiguity is possibly lost and is replaced by reachability. The path is traced from one cell to another cell, which is on the next lower layer as well as reachable.

7.1.1. Computing extended neighbors

Consider a case as illustrated in Fig. 10. The navigation front has reached cell P which has to be expanded into its neighbors. The figure shows a planar projection onto XY plane. For illustration, human model of height 4 cells is placed at P. Let both its horizontal and vertical foot spans be 2 cells. The contiguous neighbors of P along X-axis are Q and R, both of which are yet to be enumerated. Q, however, does not have a supporting object to step on (i.e., lower cell is empty) and R is not a surface cell. The surface cells C and D are directly above Q and R, respectively, but are immediately unreachable from P. The immediately reachable cell neighbors happen to be the extended neighbors A and B located below Q and above R, respectively. The following pseudo-code fragment identifies extended neighbor of cell $P(cx, cy)$ along positive X-axis.

```
x = cx + 1;
while (x < max_X_resolution) {
  y = cy + body_height_in_cells - 1;
  while (y >= 0) { /*0 is the lowest y unit in the grid */
    if (cell(x,y) is occupied)
      return success;/* cell (x,y+1) becomes a valid neighbor */
```

```
    y = y - 1;
  }
  x = x + 1;
}
return failure;
```

In order to identify the nearest surface cell neighbor all columns on either side of P are scanned. In each column, starting from the cell with Y component $my = (cy + \text{body height in cells} - 1)$, downward search is performed for the first occupied cell. In the absence of such cell, the next column is tested. The first column with an obstacle provides X component of the (extended) neighbor. The procedure is repeated along all directions around P which returns A and B as the orthogonal extended neighbors of P. Cell B however fails the reachability test as clearing height above it is only 3 cells which is insufficient for 4-cell tall human model. This leaves cell A as the unique extended neighbor of P along X-axis.

The code identifies only one surface cell in a column. For example, if the human model is moving in right direction from S, the cells E and F could be considered as the potential neighbors of S. Since the search is carried out at 4 cells above Y component of P, the only identifiable cell would be F. For normal walking situations, single surface cell per column is sufficient. If the human however is endowed with other actions such as bend, jump, run, etc. multiple surface cells should be identified as extended neighbors. If the human is prohibited from reaching F (for example, because of some danger), then cell E located downwards in the column should be considered. Multiple cells therefore need to be tested for reachability and appropriate unique cell should be entered into the navigation front. The action required (for example, reaching E may require a jump from S) should be encoded at the current cell (see Section 8.2).

7.1.2. Efficiency considerations

The 3D path for human motion is constrained to lie on surface region with solid support underneath to walk on. All empty or unoccupied cells are ignored. The navigation front is therefore generated using surface cells only. Since a surface cell has maximum of eight surface cell neighbors (see Section 3.5), which is same as in 2D navigation front is generated almost with the same efficiency as in 2D. In addition, tracing back of the path is also as fast as in 2D case.

7.2. Drawbacks

Consider the example shown in Fig. 11. Cells B and C are assumed to be immediately reachable from A. Suppose the vertical distance between B and C exceeds vertical foot span making the latter unreachable from the former. If A is expanded into its reachable neighbors, cell B would be enumerated in the next step. The navigation front stops at B as cell C is unreachable from B. This is, clearly, an anomaly as C can be immediately reachable from A due to only a small difference between their elevations. The solution is to enumerate *all* reachable neighbors of A, instead of only the first layer of reachable cells surrounding it.

7.3. Border cells

Optimal paths created using object-oriented maps make the human move very close by obstacles. In real life, humans do not walk close to walls, pillars, etc. while approaching a goal, even if such paths are

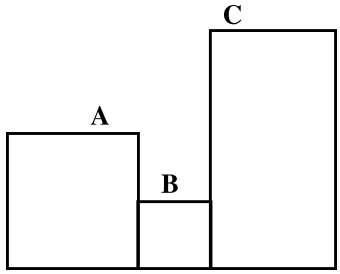


Fig. 11. B is reachable from A. C is not reachable from B because the former is at too high an elevation. A human starting at A will not be able to reach C via B. C however is reachable from A directly due to lower difference in elevation. If A enumerates only its closest reachable neighbors, the navigation front stops at B resulting in failure. If it enumerates all its reachable neighbors, C will be generated creating a path from A to C.

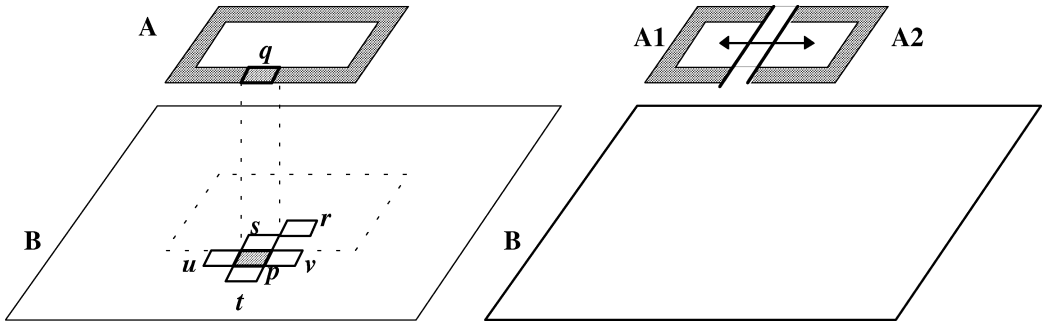


Fig. 12. Left: $\boldsymbol{p}$, $\boldsymbol{q}$ are unreachable from each other; $\boldsymbol{q}$ is the only border cell in the pair. Right: $A1$ and $A2$ are separated but reachable from each other; consequently, there exists no border between them. Both cells are, however, unreachable from B creating an overall border.

geometrically optimal. One way is to geometrically expand the obstacles by certain amount so that the path computed passes nearby leaving a safe margin. This is costly and many of the objects, for example, a ceiling or decorative lamps, do not require the expansion as they may not always lie in the human path. Further, according to an assumption made earlier (see Section 3.2), the geometrical information of the scene may not be available in the first place. Consequently, in this paper, the concept of border cells is used to produce a safe margin between obstacles and paths using the following definition:

- If a cell has no immediately reachable neighbor from any of its 4 sides, then it is a border cell.

The definition tests only 4-neighbors for border cells. This can be easily extended to 8-neighbors by testing for reachability across diagonal cells. It should be noted that if $\boldsymbol{p}$ is unreachable from $\boldsymbol{q}$, the latter is a border cell while the former *need not be*. The reason is that the extended neighbourhood, as computed in Section 7.1, is not symmetric. If $\boldsymbol{p}$ is a contiguous neighbor to $\boldsymbol{q}$, so is $\boldsymbol{q}$ to $\boldsymbol{p}$. On the other hand, if $\boldsymbol{p}$ is an extended neighbor of $\boldsymbol{q}$, the latter need not be an extended neighbor of the former. Consider the example in Fig. 12. A small object A is located high above platform B. As a result, the platform is unreachable from cells in A. This creates border cells (e.g., cell $\boldsymbol{q}$) around A shown as the dark region. For cell $\boldsymbol{p}$ on B, all its four orthogonal neighbors $\boldsymbol{s}$, $\boldsymbol{t}$, $\boldsymbol{u}$ and $\boldsymbol{v}$ are reachable and does not constitute a border cell even though cell $\boldsymbol{q}$ is not immediately reachable from $\boldsymbol{p}$. This is consistent with the principle

that reachability is local in nature. If this was not the case a border region would have been created on B around the projection of object A which would further prevent human models from reaching an interior cell, for example, $\boldsymbol{r}$ from outside.

The right picture in Fig. 12 illustrates another property of border regions. Suppose platform A is replaced by two segments $A1$ and $A2$. Both the regions are close enough to be reachable from each other and there is no border region that separates them. Both the regions then merge into a single reachable region. Consequently both $A1$ and $A2$ are unreachable from B which creates a border around the single reachable region.

Borders are generated by carrying out an initial bushfire without A* optimization to go through all the cells. At each step, a cell is expanded into its reachable 4-neighbors (or 8-neighbors if corner cells are allowed). If any neighbor is unreachable then the current cell is marked as a border cell. Since a border cell's $Z-X$ dimension is same as the distance between feet (Section 3.3), the border region is usually sufficient for avoiding obstacles by a safe margin. If further clearance is needed, another layer of border cells may be generated, treating previous border cells as unreachable regions. Multiple border layers are useful if small cell size is used in discretization.

7.4. Path smoothening

The path generated is sub-optimal according to Manhattan metric, but it still produces unnaturally jagged cell paths. This creates large variations between Euclidean paths and cell paths. In Fig. 13, the picture on the left shows a path generated after considering border regions (compare with top left picture in Fig. 7). The path between C and A contains an unwanted bend deviating from Euclidean path CA. A realistic human path makes a direct connection between these two cells along CA as shown in the picture on the right. This is created by applying DDA between *junction cells* which consist of the two terminal cells in a path and intermediate cells at which the path changes its direction. In the figure A, B

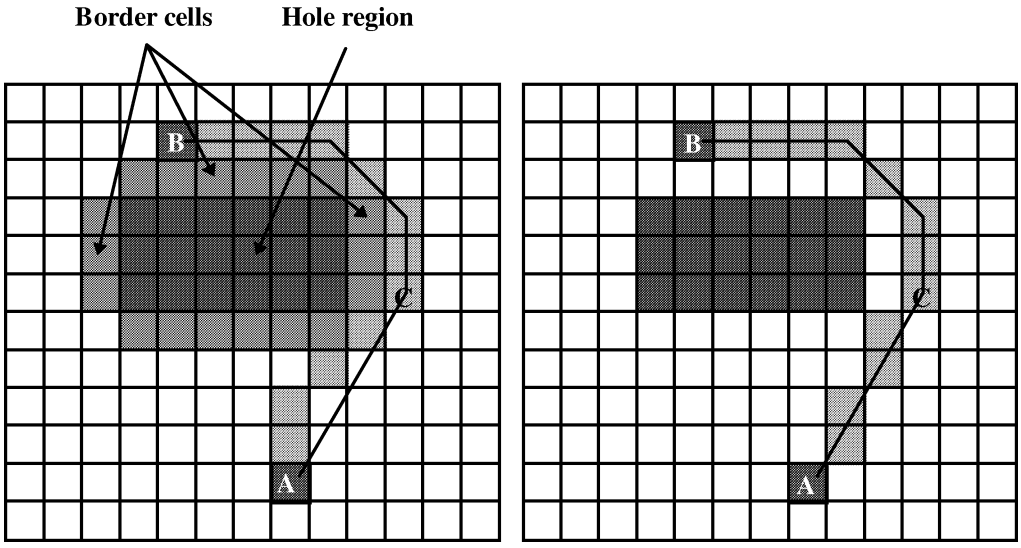


Fig. 13. Border cells (left) and path smoothening using DDA (right).

and *C* are some of the junction cells. As shown in the figure, enumeration of cells between *C* and *A* using DDA generates a smoother path that is closest to the Euclidean straight line path. The original path is retained if cells generated by DDA turn out to be unreachable. This technique produces only partial smoothening. For comprehensive smoothening of the path, all cells in the path should be treated as junction cells. Real-time requirements however compel a compromise and only a few cells as selected above are considered as junction cells. The method produced reasonably smooth paths for most of the examples tested.

8. Special features

8.1. Semantics of environment

The discretization process hitherto described partitions object space into a discrete grid of 3D cells. Each cell is marked as occupied or empty. The occupancy status however does not contain any information about the type of objects. All objects are classified as either obstacles or supports for walking. In a 3D scene, walls, pillars, etc. are treated as obstacles and are avoided. Floors, staircases are used as supports over which paths are generated for walking. Consider an object such as a table. If this is tall, it is treated as an obstacle and avoided when navigation front reaches this region. If this is a low-level table, a path may well be generated requiring the human to step on it in order to reach the goal. Since this is not usually the case in real life, these actions should be avoided. The reason is that objects in every day life are endowed with meaning constituting *semantics* of the environment. This prevents people from walking on tables, chairs, decorative items, etc. which are termed *forbidden* objects.

Consideration of semantics limits regions for navigation. Forbidden objects should be discarded by the navigation front. One way of achieving this is to mark the cells occupied by such objects as hole cells. Following this, the navigation path circumvents the hole region thereby guaranteeing that the human model avoids these objects. One disadvantage of this technique is that a solid object may lose vertical dimension when converted to a hole as illustrated in Fig. 14.

Fig. 14 shows a discretized object space with a floor and an obstacle. The lower portion of the figure represents the object space when the obstacles are converted to hole regions during discretization. Since holes are only empty spaces embedded within occupied regions, vertical dimension is lost when obstacles

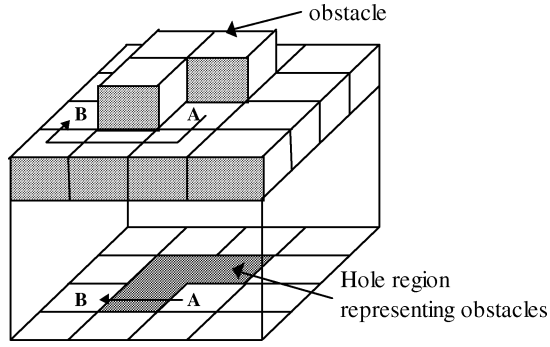


Fig. 14. A solid obstacle may lose dimension when represented as a hole.

are represented as holes. If the obstacles are forbidden, a path from cell A to cell B, should go *around* the obstacles without either stepping on/over them as shown in the upper picture. With a hole region, however, a direct path may be generated from A to B as shown in the lower picture. The human model then walks over the obstacles possibly intersecting them during animation.

It can be recalled that unreachable regions are identified during generation of navigation front. The forbidden objects are treated at a higher level by explicitly marking the cells occupied by them as unreachable even before applying the navigation front. This is done during discretization of the scene. These regions are then discarded from the navigation front making the paths generated conforming to the semantics of the environment.

Marking objects to be avoided

Since discretization uses frame buffer technique, there are several options for identifying forbidden objects. They ultimately require cells occupied by such objects to be explicitly marked. One way is to colour code them and tag the cells occupied by them during discretization. Another possibility is to discretize normal and semantic objects separately and mark the cells accordingly. Binding certain objects with a predefined colour scheme makes designing of large scenes inflexible. Further, if colour information is omitted for making discretization more efficient, colour coding fails to work. As a result in the current implementation the second option is adapted.

8.2. Action encoding

If the human is permitted to perform actions beyond normal walking, this information should be encoded during the generation of navigation front. Since navigation front depends on reachability criterion, actions like jumping, bending, etc. introduce additional reachability to human and consequently greater choice of paths. If several alternate optimal paths are possible from different actions, ties are resolved by prioritizing actions. The generated path should preserve actions encoded at each cell. The human performs these actions upon reaching the cells while walking along the path.

8.3. Dynamic obstacles

Until now, the environment is assumed to be static. In real world much of the environment remains stationary, but moving or dynamic objects are common. For example, motor vehicles or people themselves. This requires changing occupancy status of grid cells whenever the scene changes. If the whole scene has to be discretized afresh, it will be very time consuming. The dynamic content of the scene is however assumed to be small compared to the total scene. The moving obstacles are therefore locally discretized and corresponding portion of the 3D grid is updated. If an obstacle obstructs a previously generated path, the path is locally rectified to make it obstacle-free. The path is segmented into three pieces: first one connecting the start cell, second one connecting the goal and the last one – which lies between the previous two – ruptured by the obstacle. The third segment has to be re-routed.

Fig. 15 shows a cell path from starting cell A to goal cell B obstructed by a moving obstacle (dark shaded). The obstacle is assumed to cover any portion of the cell path except the start and the goal cells. The human is also assumed to be on a cell not covered by the new obstacle. In the example, the cells from P to Q are completely covered by the obstacle. The cell segments $A-C$ and $D-B$ are left intact. If the human happens to be on the segment $D-B$ which is free of obstacles, no action is necessary as B

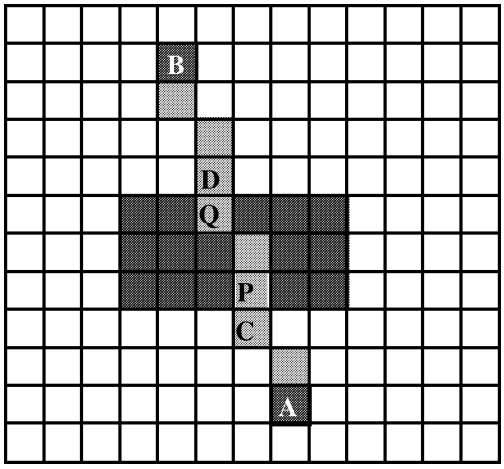


Fig. 15. Path obstructed by an obstacle; re-routing needed between C and D.

is the goal cell and human can continue walking along $D-B$. If the human is on the segment $A-C$, then it is not possible to follow the existing path to the goal. A new path has to be generated so that it reaches the goal cell B. The cell C is considered as the starting cell and all other path cells in the segment $D-B$ are considered as goal cells. Multiple goal floodfill is used to generate a path from C to $D-B$ segment. The resulting path connects C to a cell on the segment $D-B$ establishing new path between C and B. As described in Section 7.4, the resulting path is smoothened using DDA.

9. Examples and conclusion

9.1. Examples

This section provides some examples for the purpose of illustrating the path finding technique. Many scenes are taken from already available sources in order to demonstrate the applicability of the technique on various scenes.

The labyrinth scene (Fig. 16) shows top view of the path generation. At point A there exists enough space for cell path, but not enough room for border region which might result in collision of human with objects. The path takes an alternate longer route through point B.

The stair case with other obstacles is procedurally defined and drawn (Fig. 17). The top picture shows a human and an initial cell path to a goal. This is an 8-path which requires smoothening. After applying DDA, the path as shown in the picture on the lower left is generated. Before reaching the goal, another obstacle appears blocking the path as shown in the picture on the lower right. An alternate path is generated to re-route the ruptured segment. The DDA smoothening results in the final path.

Super market scene shows entering it from outside (Fig. 18, left picture). In the factory scene (courtesy http://www.3dcafe.com), the human has to avoid the overhanging obstacle underneath which it cannot stand upright. A border is generated on the floor to keep the human from entering this region. The path generated goes past this to reach a goal located at the furthest corner.

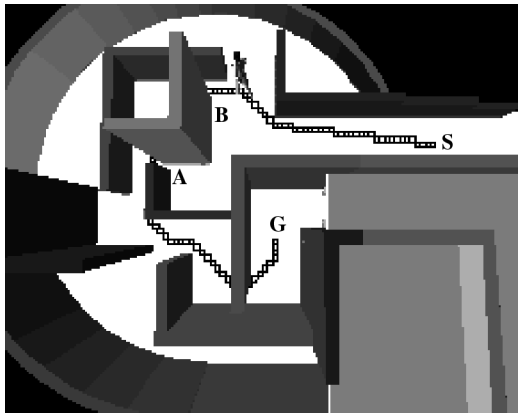


Fig. 16. Labyrinth (Inventor file, 640 K).

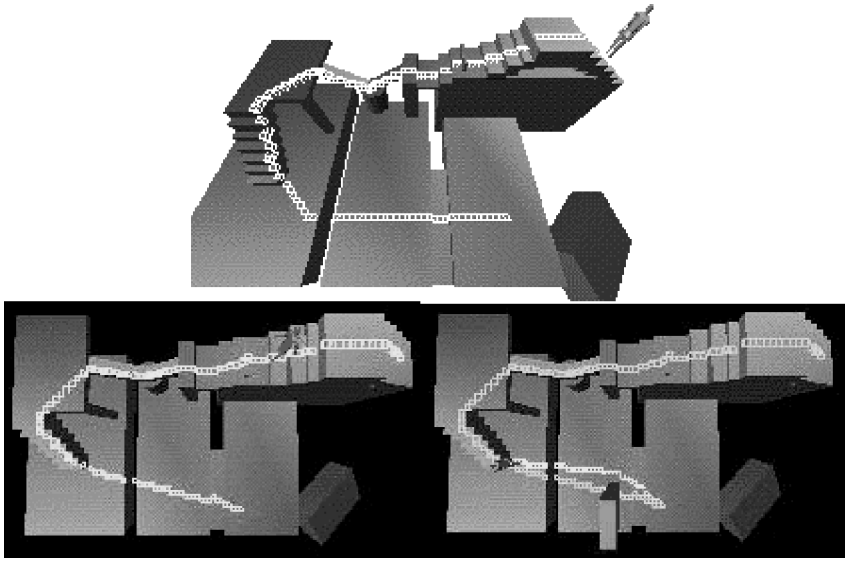


Fig. 17. Stair case (procedurally generated).

Table 1
Statistics for path finding examples (platform: Silicon Graphics Indigo2 Impact with R10000 CPU, 128 Mb)

	File size (Kilobytes)	Grid size ($x \times y \times z$)	Discretization, border generation times (seconds)
Labyrinth	443	$72 \times 23 \times 93$	3, 6
Stairs	procedurally defined	$53 \times 102 \times 91$	5, 10
Factory	609	$69 \times 95 \times 69$	40, 8
Super market	1,500	$162 \times 90 \times 236$	45, 5

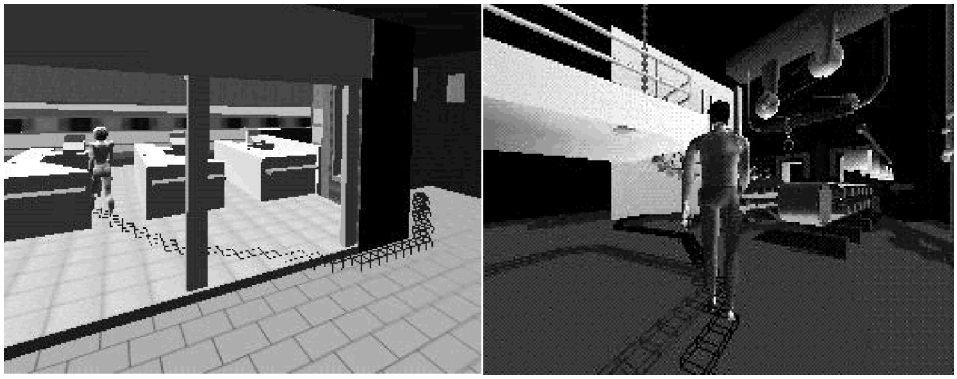
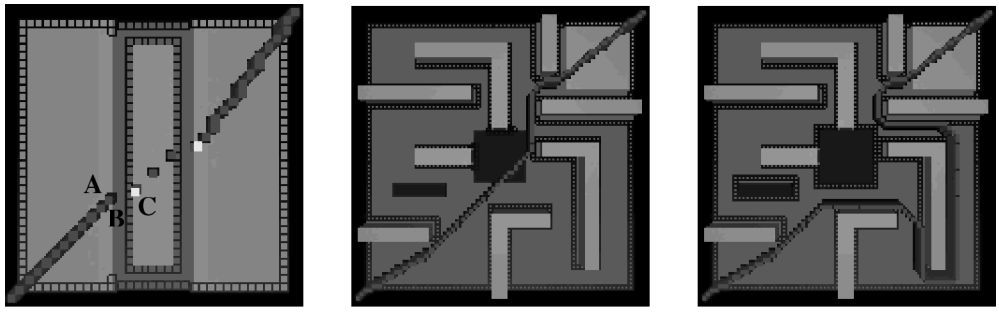


Fig. 18. Super market (left, Inventor file, 1.5 MB); factory scene (right, Inventor file, 600 K).



(a) Path generation by enumerating all reachable cells (b) Forbidden regions ignored (c) Forbidden regions considered

Fig. 19. Path generation by considering all reachable neighbors in navigation front expansion (a); taking semantics of environment into account ((b) and (c)).

The table lists timing statistics for the examples. It is clear from the table that the discretization and the border generation are time consuming processes of the algorithm. The path generation is almost instantaneous, taking up around a second even for complex scenes. The time for discretization increases with vertical resolution which defines number of slices to be drawn. Each time entire scene is drawn despite the fact that most of it would be clipped within the slice. This could be reduced by more sophisticated scene processing. The drawing function at present uses all colours in drawing which is rather unnecessary for finding occupancy status. This could be optimized by using only a bi-level monochrome framebuffer for drawing.

Considering all reachable cells, action encoding and semantics of environment

Fig. 19(a) illustrates the path generation technique after rectifying the drawback mentioned in Section 7.2. The picture shows three blocks on a flat floor. The small middle block is at a higher elevation compared to the larger blocks on its either side. The cells A, B and C shown here have similar properties as described in that section (see Fig. 11). A is on the left block, B is on the flat floor, and C is on the

middle block. A can reach both B and C, but B cannot reach C. A navigation front expanding by one cell layer at a time therefore stops after reaching B from A. In order to rectify this, A has to enumerate all reachable neighbors which includes cell C. While tracing back across the navigation front for the path generation, the next cell with lower cost need not be contiguous to the current cell. This is illustrated in Fig. 19(a). The lighter cells in the path encode actions indicating that a jump is needed at those particular cells to move to the next cell. Fig. 19(b) shows an object at the center of the scene. This is semantically forbidden object. If the semantics are not considered the resulting path is as shown in Fig. 19(b). The corrected path is shown in Fig. 19(c).

Super sampling

The cell size on $Z-X$ plane is set to be distance between the feet of human standing in normal position (Section 3.3). Some times, with this cell size, resulting scene discretization may be very coarse. Occupied cells may contain only very small objects with large empty spaces. The path containing such cells makes human unnaturally step on small supports. One way of reducing this effect is to super sample each cell with several cells and set an occupancy weight derived from occupancy status of sub-cells. The cells with very low weight could then be eliminated by the navigation front cell enumeration. In a simpler alternative (implemented in the current version) navigation front and path are generated as before. The cells in the path are tested for occupancy weight and only the cells with high occupancy are stepped on. The number of discarded cells in a single stretch however cannot exceed *horizontal foot span*.

9.2. Conclusion

In summary, this paper presents a discrete grid based path finding algorithm for complex 3D scenes. Once a discrete grid map is prepared, the path finding is very efficient for real-time applications like video games. The algorithm handles complex 3D scenes and provides a robust mechanism for incorporating various types of actions. It also suggests a way of adding semantic information to the scene. Dynamic obstacles could be avoided by alternate path generation on-the-fly. The smoothening techniques generate natural looking motion paths for humans.

The limitations of algorithm at present include inability to handle multiple humans. The treatment of dynamic obstacles could be slow if the obstacles are very large. The moving target or goal location is dealt in straightforward manner. A path is generated again from the current human position if the goal location moves. This could be costly if the target is changing continuously after an initial path is generated. In the case of dynamic obstacles, this is optimized by preserving most of the original path. A similar approach should be planned in this situation. In the on going research, secondary heuristics are applied to take factors other than geometric optimality into account [2] thereby subjecting the path generation to additional constraints. Further research is needed in this direction.

Demo videos, pictures

http://ligwww.epfl.ch/~srik/path.html

Acknowledgements

The authors would like to thank Paolo Baerloch and Ms. Sahithi P.S. for painstakingly going through the draft and making many useful suggestions.

References

[1] S. Bandi, Discrete object space methods for computer animation, Ph.D. Thesis, Thesis No. 1802, Ecole Polytechnique Federale de Lausanne, Switzerland, 1998.

[2] S. Bandi, M. Cavazza, Integrating world semantics into path planning heuristics for virtual agents, in: Workshop on Intelligent Virtual Agents (Virtual Agents 99), Salford, United Kingdom, 13 September 1999.

[3] S. Bandi, D. Thalmann, An adaptive spatial subdivision of the object space for fast collision detection of animated rigid bodies, Computer Graphics Forum 14 (3) (1993) C259–C270.

[4] S. Bandi, D. Thalmann, A configuration approach for efficient animation of human figures, in: IEEE Non-Rigid and Articulated Motion Workshop, Puerto-Rico, 15 June 1997.

[5] S. Bandi, D. Thalmann, The use of space discretization for autonomous virtual humans, Video Paper, in: Second International Conference on Autonomous Agents, Minneapolis/St. Paul, 10–13 May 1998.

[6] S. Bandi, D. Thalmann, Space discretization for efficient human navigation, Computer Graphics Forum 17 (3) C195–C206.

[7] R. Boulic, D. Thalmann, A global walking model with real-time kinematic personification, Visual Comput. 6 (1990) 344–358.

[8] M. Cavazza, S. Bandi, I. Palmer, "Situated AI" in video games: Integrating NLP, path planning and 3D graphics, in: Symposium on Artificial Intelligence and Compute Games, AAAI 99 Spring Symposium, 22–24 March 1999.

[9] J. Cofman, Numbers and Shapes Revisited, Oxford University Press, 1995, pp. 83, 245, 246.

[10] T.L. Dean, M.P. Wellman, Planning and Control, Morgan Kaufmann, San Mateo, CA, 1991.

[11] A. Fujimoto, T. Tanaka, K. Iwata, ARTS: Accelerated Ray Tracing System, IEEE CG&A 6 (4) (1986) 16–26.

[12] P.M. Isaacs, M.F. Cohen, Controlling dynamic simulation with kinematic constraints, behavior functions, inverse dynamics, Comput. Graphics 21 (4) (1987) 215–224.

[13] Y. Koga et al., Planning motion with intentions, in: Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, July 1994, pp. 395–408.

[14] J.J. Kuffner, Jr., Goal-directed navigation for animated characters using real-time path planning and control, in: International Workshop, CAPTECH'98, Geneva, Springer, 1998, pp. 171–186.

[15] J. Lengyel et al., Real-time robot motion planning using rasterizing computer graphics hardware, in: Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, August 1990, pp. 327–335.

[16] T. Lozano-Pérez, M.A. Wesley, An algorithm for planning collision-free paths among polyhedral obstacles, Comm. ACM 22 (10) (1979) 560–570.

[17] T. Lozano-Pérez, Spatial planning: A configuration space approach, IEEE Trans. Comput. 32 (2) (1983) 108–120.

[18] P.J. McKerrow, Introduction to Robotics, Addison-Wesley, Reading, MA, 1991.

[19] J.S.B. Mitchell, An algorithmic approach to some problems in terrain navigation, in: D. Kapur, J.L. Mundy (Eds.), Geometric Reasoning, Elsevier, Amsterdam, 1988.

[20] N.J. Nilsson, Principles of Artificial Intelligence, Tioga, Palo Alto, CA, 1980.

[21] H. Noser, A behavioral animation system based on l-systems and synthetic sensors for actors, Ph.D. Thesis, Thesis No. 1609, Ecole Polytechnique Federale de Lausanne, Switzerland, 1997.

[22] B.D. Reich, H. Ko, W. Becket, N.I. Badler, Terrain reasoning for human locomotion, in: Computer Animation'94, pp. 76–82.

[23] D.F. Rogers, Procedural Elements for Computer Graphics, McGraw-Hill, 1985, pp. 30–39.

[24] J.T. Schwartz, M. Sharir, A survey of motion planning and related geometric algorithms, in: D. Kapur and J.L. Mundy (Eds.), Geometric Reasoning, Elsevier, Amsterdam, 1988, pp. 157–169.

[25] M. Van de Panne, From footprints to animation, Computer Graphics Forum 16 (4) (1997) 211–223.