Space discretization for efficient human navigation

Srikanth Bandi and Daniel Thalmann
Computer Graphics Lab
Swiss Federal Institute of Technology, CH-1015, Lausanne, SWITZERLAND
e-mail: {srik, thalmann}@lig.di.epfl.ch
{http://ligwww.epfl.ch/}srik.html,thalmann.html

Abstract
There is a large body of research on motion control of legs in human models. However, they require specification of global paths in which to move. A method for automatically computing a global motion path for a human in 3D environment of obstacles is presented. Object space is discretized into a 3D grid of uniform cells and an optimal path is generated between two points as a discrete cell path. The grid is treated as graph with orthogonal links of uniform cost. **A*** search method is applied for path finding. By considering only the cells on the upper surface of objects on which human walks, a large portion of the grid is discarded from the search space, thus boosting efficiency. This is expected to be a higher level mechanism for various local foot placement methods in human animation.

Keywords: global navigation, dynamic programming, **A*** graph search, articulated body models

1. Introduction

Human walking is a complex and well studied component of articulated body animation research. This being one of the most basic motions of a human body, several models have been developed to realistically simulate the motion on the computer. The task of making a human model move from its present location to a goal location involves two sub-tasks:

(1) Finding a path from the current location to the goal location
(2) Making the human move along the path generated.

In the second sub-task, a *walk-cycle* is implemented. Each walk cycle simulates movement of a free foot from the current position to the next. Movement along a given path is carried out by cascading several walk-cycles. A single walk-cycle movement is confined to local region of the walk path. There are several articulated body models for such local, task-specific actions. The models are by and large *dynamic* [Arm85, Isa87, Bru89, Mck90, Phi90, Rai91, Hod95] or *kinematic* [Gir85, Kor82, Bou90, Ban97]. Dynamic models apply forces and torques on the human body parts (through muscle forces) to drive the skeleton motion. These models are physically based and produce realistic animation. But the amount of control available is minimal as the model is driven by dynamics. Kinematics offer greater control by letting the animator specify the motion path of the limbs and let an inverse kinematics module compute values for skeleton joint angles to achieve the desired motion. There have been mixed approaches to this problem [Isa87, Hye96, Ove94].

Thus, there is quite extensive literature on the local control of the articulated figure in motion. However, on the first task of automatic generation of global motion paths, the literature is more modest. The freedom of movement is constrained by the presence of obstacles in the environment. The object should find an obstacle-free feasible path to a goal while satisfying certain criteria such as shortest path, least cost path etc. This problem is studied in Robotics for navigation of mobile robots. Lozano-Peréz [Loz79, Loz83] constructed configuration space of all possible configurations of the moving object, thereby reducing the problem of finding obstacle-free path of an object to that of a point (representing a configuration of the object) in a higher dimensional configuration space. The subspace in which the object intersects obstacles is termed *configuration space obstacles*. The moving point should find a path outside configuration space obstacles. Following this, Lengyel et al [Len90] discretized the configuration space obstacles (termed *C-space obstacles*) using raster hardware and generated a discrete cell path using dynamic programming. In this paper, the grid search algorithm is based on similar dynamic programming techniques. A brief video review of the results is given in [Ban98]. Configuration space approach was used by Y.Koga et al [Kog94] for arm motion. Reich et al. [Rei94] guide a human using sensor based navigation in a terrain. The obstacles are avoided using potential fields. The regions with obstacles generate high potential and those clear of obstacles have low potential. The human finds foot placements in regions of low field strength. Potential fields have a tendency to get stuck in local minima and this approach is unable to find paths even if there exists one. Michael van de Panne [Van96] simulated motion of bipeds when a track of footprints is given. The automatic generation of footprints is addressed in a limited way. If obstacles are detected, the footprint location is stochastically changed to a valid place. Noser [Nos97] also used sensor based navigation for finding a path in an environment of obstacles. A vision sensor (implemented using z-buffer) creates a map of the environment. The obstacles are avoided using potential fields. Bandi and Thalmann [Ban97] simulated human walking in an environment of obstacles using configuration spaces. However, obstacle-free global paths are still manually specified. In this work a method for automatic generation of motion paths in complex 3D environments is presented. Following are some of the features of the proposed method:

*Given a goal position, the navigation algorithm generates a discrete cell path if one exists. The human should be able to reach it on its own without interactive intervention from the animator.

* *Complex 3D environments*
The environment can be arbitrarily complex regarding navigation paths possible and obstacles encountered. The environment may represent real-life models.

**The navigation path should be optimal*
In a cell grid, distance between two cells is measured according to a distance metric (see section 3.2) which need not be same as the Euclidean distance. There may be more than one path measuring the same distance between the cells. But an optimal cell path between two cells should be no longer than the shortest Manhattan distance between the cells. Since cells are of uniform size, Manhattan distance is indicated by the number of cells in the path.

The next section describes the construction of discretized environments. Section 3 presents the method for finding optimal cell paths. Section 4 presents some results and Section 5 concludes.

2. Representation of the environment and complexity issues

The motion path is generated using an internal representation of the environment under consideration. There are basically four types of representation [PJM91]. In the first method, a polygonal (or polyhedral in 3D) representation of obstacles (*object oriented maps*) is given. This explicitly specifies regions with obstacles which should be avoided. A disadvantage is that the human moves too close by obstacles when turning at corners. In the second method, a representation of regions free of obstacles (*free-space maps*) is provided. A graph representing free spaces facilitates movement from one free space node to another. In this, the human tends to move along central regions of free spaces covering distances longer than is necessary to reach the goal. The complexity of the scene is expressed in terms of number of polygons k and number of vertices n needed to represent the scene. In the third method the space is discretized into a grid of rectangular cells (or voxels) and each cell is marked as an obstacle or a non-obstacle. The complexity of the scene is expressed by the number of cells n in the map. Since the grid map records whether a region in the environment is an obstacle or a free space, it is known as *composite-space map*. A fourth representation, *path maps*, uses a set of predefined paths for guiding a robot in a fixed environment. This model is inflexible and unsuitable for human motion.

For human navigation grid based approach is adapted due to its amenability for interactive applications. Polygonal methods can accurately model objects but are computationally expensive. There are methods that run in linear or O(log *n*) time, but preprocessing of polygons to achieve this performance runs into polynomial time. Hence for complex 3D environments polygonal methods are unsuitable. One common polygonal representation uses *visibility graph*. In a visibility graph, the nodes are vertices of obstacles and edges are connections between pairs of vertices which can "see" each other without being obstructed by the obstacles. The graph can be searched for a shortest Euclidean path using A* algorithm in O(K + *n* log *n*) time where K is the number of edges in the graph [Mit88]. In 2D, the worst case time complexity for the construction of the visibility graph is $O(n^2)$. Schwartz and Sharir report [Sch88] that in 3D this runs into exponential time. In uniform grid method, the preprocessing corresponds to discretizing the object space into cells. Using frame buffer hardware, the discretization can be achieved in linear time and performed very efficiently. In general an *m* dimensional grid with *r* cells on each side takes $O(mr^m \log r)$ time [Dea91]. For dimensions greater than 4 the grid method is prohibitively expensive, but is more efficient than the polygonal methods at lower dimensions. If dynamic programming is used, it is possible to construct optimum paths from any cell to a goal cell in $O(mr^m)$. That is, the 2D grid search with *n* cells works in O(*n*) time. As will be seen later, in 3D the performance is close to 2D levels as only the grid cells lying on the surface of the objects are considered.

2.1. Discretization of the environment

Discretization of object space is the bottleneck in the algorithm, but performed only once for static objects. After discretization each cell in the grid possesses a binary information to indicate whether it is occupied by an obstacle or not. One way of efficiently accomplishing this is through integer arithmetic of DDAs used for collision detection [Ban95]. In this, polygons are covered by equidistant lines and each line is rasterized using DDA [Rog85]. However, such methods require geometric specification of the objects in a form understood by the navigation algorithm. This may be difficult if the environment description comes from third party sources in other forms such as Inventor or DXF files.

An efficient method for discretization that keeps navigation task independent of geometry is by making use of framebuffer hardware for display. As a convention Y axis is assumed to be the vertical axis and Z-X plane as the horizontal plane. The scene is enclosed in a bounding box and sliced into several horizontal portions along the Y axis. With each slice treated as a clipping volume the scene is displayed in a window of a resolution equal to the required resolution of the grid in the Z-X plane. The number of slices along Y equal the resolution along the axis. The pixels drawn inside the window correspond to occupied cells in the horizontal portion of the cell grid under consideration. They are read from the framebuffer and the cell grid is marked accordingly to indicate the occupancy status.

The Z, X components of the cell dimensions are set equal to the distance between the feet when human stands in a normal upright position. This is deemed to be the minimum distance the foot should move in a single step. Thus, the Z,X resolution of the object space is fixed by the bounding box size and the distance between the feet. In contrast, vertical cell size can vary. Here is a trade-off : Higher vertical resolution helps more accurate foot placement, but it takes longer time to discretize the scene.

The only requirement for separating the navigation task from the discretization is the availability of a drawing function for the scene. The user is expected to provide a method of drawing the customized scene descriptions provided by him/her. As graphic oriented displays perform drawing operations in hardware, discretization is very efficient. Once the discretization is complete, the navigation algorithm is invoked for generating a cell path between the current and goal locations.

3. Finding the motion path

The uniform grid can be thought of as a graph with rectilinear connections. Every cell (except border cells) has four orthogonal neighbors. There are many algorithms for searching such graphs for optimal paths. A survey of such techniques is given in

[Mit88]. In Artificial Intelligence (AI) literature many graph search algorithms are variants of *branch and bounds search.* One of the common variants is A^* (read “A-star”) search method [Nil80]. The node representing the starting cell has zero cost. This cell is expanded to generate all non-empty neighboring orthogonal cells. The new cells are appended to an initially empty queue and their costs are incremented by one. Previously generated cells are not enumerated again. The cost of a cell gives the number of cells in the path from the first cell and as such constitutes a shortest path from it. This is dynamic programming mechanism. All enumerated cells have their shortest paths computed. At every iteration first cell is de-queued and expanded. The cells are enumerated in a breadth-first fashion starting from the first cell. The generated cells spread like a *bush-fire*. This process continues until the goal cell is reached. Breadth-first search (BFS) is a special case of A* method and is guaranteed to find an optimal path. This is also a technique for flood-filling regions with closed boundaries using the first cell as the *seed cell*. Hence this is also known as *floodfill* algorithm. In order to reduce number of cells enumerated a heuristic is employed to assign weights to the cells generated. Following is the evaluation function used:

$$f(n) = g(n) + h(n),\ g(n),h(n) >= 0$$

g(n) is the cost of reaching the node from the starting node, *h(n)* is the heuristic which is an estimated cost from node *n* to the goal and *f(n)* is the overall cost. In the search tree the node with the least *f(n)* is selected for the next expansion. Thus nodes with promising optimal cost get searched on a priority basis leading to faster termination than in the BFS method. The heuristic used here is the ‘shortest distance’ as defined in Section 3.2.

Once the goal cell is reached, the path is generated by tracing all parent cells backward up to the starting cell. A parent cell is simply the neighbor with the lowest cost. A parent cell can be one of the 4 orthogonal neighbors (generates 4-path) or one of the 8 neighbors (generates 8-path).

3.1. Multiple goals

There can also be a multitude of goal cells for the path. When the floodfill reaches one of the goal cells, the search is terminated. Thus, the path is generated from the start cell to the closest goal cell. Now, consider a simpler case of navigation in 2D planar environments. Most of the concepts of 3D navigation are extended from the 2D case.

3.2. Global navigation in 2D

In two dimensions, the surface is a flat terrain which is discretized into occupied/empty cells. The terrain can contain hole regions (unoccupied regions) which are prohibited. No cell in the generated path can fall inside the hole region. Subject to this constraint the algorithm generates a path no longer than the shortest Manhattan distance.

Shortest distances

For the navigation algorithm, the distance between two cells P(px,py) and Q(qx,qy) in a discrete plane, where (px,py) and (qx,qy) represent integer cell coordinates, is *defined* as below.

$$d_4(P,Q) = |px-qx| + |py-qy| \quad \ldots\ldots\ldots\ldots\ldots (1)$$

d_4, which represents Manhattan distance metric, is the smallest number of cells lying between P and Q if only 4-neighbor (orthogonal neighbors) moves are allowed. The shortest 4-path need not be unique. If $dx=|px-qx|$ and $dy=|py-qy|$, then the number of shortest 4-paths is given by [Cof95]

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

If 8-neighbor (both orthogonal and diagonal neighbors) movement is allowed, then the shortest 8-path between P and Q is

$$d_8(P,Q) = \max(dx=|px-qx|, dy=|py-qy|) \ldots\ldots (2)$$

If x > y, and if all 8-paths are confined to the rectangular cell box formed by P and Q, then the number of 8-paths is

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

Note that $d_8 <= d_4$ always. So the 8-path is the shortest path between two cells. However, 8-neighbor connectivity is known to produce anomalous paths in case of obstacles as shown in Figure 1. There are two cells with obstacles (shown in dark). The path from P to Q passes in between two obstacles which is illegal. This is due to 8-neighbor connectivity of the cell grid graph. This error could be rectified using 4-neighbor connectivity (right picture). The strategy followed here is like this: construct the graph with 4-neighbor

connectivity to avoid anomalous paths and then generate a d_8 path using 8-neighbor movement to optimize the path; during diagonal motion (8-path motion) check for obstacles at the crossing point. The heuristic cost function for A* search is taken as the 4-distance:
$h(n) = d_4(n,g) = |gx-nx| + |gy-ny|$, where *g* is the goal cell.

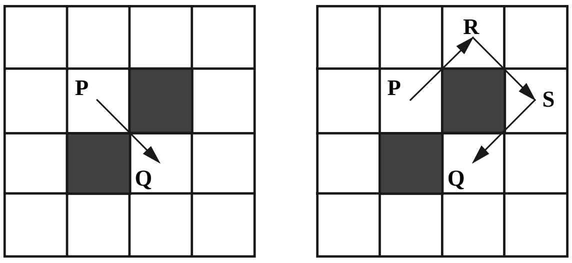


Figure 1. ***(left)*** **Anomalous path due to 8-neighbor graph connectivity,** ***(right)*** **Proper path in 4-neighbor connectivity**

An Example

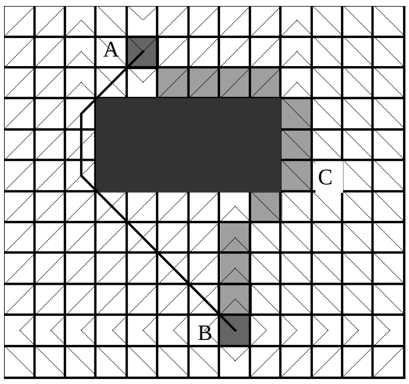


Figure 2. Cell enumeration from B to A

Figure 2 shows a rectangular region with a hole (dark-shaded) inside. There are two cells A and B between which a shortest cell-path is enumerated (light-shaded). For illustration, the cells are enumerated without the heuristic. The generated cells expand from cell B in waves or layers of concentric squares. All cells in the same layer are equidistant from the starting cell B. They represent leaf nodes of the **A*** graph search tree to be expanded in the next round. The path generation starts from cell A and propagates backwards towards cell B. The path moves from cells of higher layers to that of lower layers until the 0th layer consisting of only the single cell, B, is reached.

Alternate shorter path

If all eight neighbors are generated for navigation front, a path passing by the left side of the hole region may be obtained. This path has fewer cells than the path on the right, even though they have identical Manhattan distances of 17 cells. 4-neighbor wave-front is slightly faster to generate and has fewer twists in the final path for smoothening[1].

Application to human model

This is a path for a point moving one cell at a time. However, for a real human this kind of movement is inadequate. For example, if the hole is small enough, the human can walk over skipping several cells, instead of walking around it.

Human Dimensions

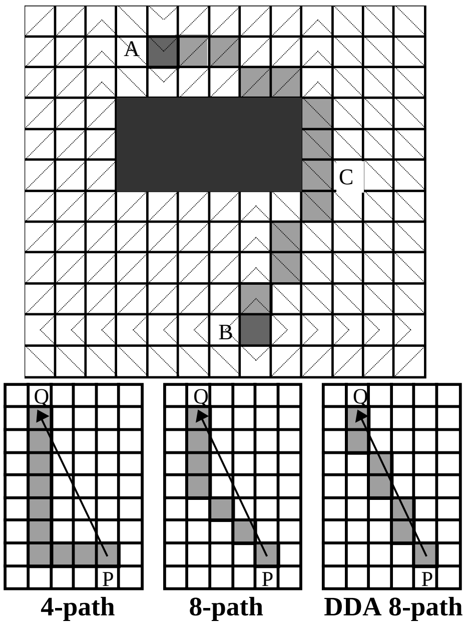
Whether an obstacle can be overcome or not depends on the human dimensions. A child can not cross long gaps as an adult can. On the other hand a child can pass through under low roofs an adult can't. The basic human dimension is *height* which is expressed in terms of cells it occupies along the vertical Y axis. *Horizontal foot span* is defined as the maximum number of cells in Z and X directions that can be crossed by the foot in a single step. Similarly *vertical foot span* can be defined as the maximum number of cells a foot can ascend or descend in a single step. Both these parameters are made functions of height. In doing so, the primary concern is to reduce the number of parameters in the system.

Extended cell neighbors

If a cell grid contains holes, then cells may not lie in contiguous locations. As a result a cell may not have all 8-neighbors. However, by extending the search for neighbors along the direction of the missing cells, one could find closest neighbors in that direction. They are termed *extended neighbors.* In 3D, a cell has 26 neighbors. However, since walking takes

[1] 8-neighbor wave-fronts produce paths usually wander away from straight line paths connecting start and goal cells. This could be mitigated by generating the path with 4-neighbor priority, i.e. in case of diagonal and orthogonal cells with identical costs, the latter should be selected. See [PJM91] for related discussion.

place on the surface, reachable cells lie on the surface. Thus, for a *surface cell*, the cell directly below is inaccessible and the cell above does not exist. So extended neighbors could be along the direction of any of the remaining 24 neighbors, possibly on a different plane. Again, considering only one cell is selected in vertical direction for foot placement, effective neighbors will be reduced to 8 just like in the 2D case. Henceforth the term 'neighbor' includes extended neighbors as well unless otherwise specified. More details on extended 3D neighbors are given in section 3.3.1 in the context of 3D obstacles.



4-path 8-path DDA 8-path

Figure 3. DDA refinement (above) Path generation between two cells in various methods (below)

Reachable cells in 2D, hole cells

Two occupied cells ***p*** and ***q*** are said to be reachable from each other if the 4-distance d_4 between them is less than or equal to the horizontal foot span, i.e. the human can step from ***p*** to ***q*** or vice-versa. For example if ***p*** and ***q*** are separated by a hole region larger than the horizontal foot span, then they become unreachable from each other.

Hole cell: *A cell from a hole region is reachable by none of its (immediate or extended) 4-neighbors.*

Smoothening the path

Though, the path generated is optimal according to Manhattan distance metric, the algorithm enumerates 8-neighbors of a cell producing unnatural paths. The more realistic path makes direct motion in the lower and upper regions as shown in Figure 3. The path is smoothened by removing jaggedness using Digital Differential Analyzer (DDA). DDA is an algorithm for enumerating all cells closest to a straight line connecting two cell centers [Rog85]. This is a special case of 8-path in which the enumerated cells lie along the straight line connecting the cell centers. For example the enumerated cells between **C** and **B** in Figure 2 are replaced by the cells shown in Figure 3. Bottom picture in Figure 3 shows path generation between cells P and Q using 4-neighbor movement, 8-neighbor movement and the DDA technique. The 4-path diverges greatly from the line connecting end cells. The middle and right most paths are 8-paths and have the same number of cells. Note that DDA path is along the shortest Euclidean path connecting P and Q. By applying DDA, one can obtain an 8-path that closely matches the Euclidean path. DDA refinement is carried out as below:

The cells where path changes direction are identified. These cells along with the first and the last are termed *node cells*. The direct path connects all alternating nodes using DDA. Some times, new cells generated by DDA may fall inside invalid regions such as holes. In that case original path segments are retained. However, DDA refinement between node cells does not completely eliminate jaggedness. To achieve maximum smoothness, every cell has to be treated as a node cell and refined. In practice the former alternative produces realistic paths at interactive rates.

3.3. Global navigation in 3D

In 2D environments, the foot location is a single valued function of the horizontal plane. This means, on a 2D surface there is only one valid location for foot placement for any given point in the horizontal plane. In 3D space the function can have multiple values. Figure 4 illustrates this. The human figure can legally take two vertical values for its current Z-X value: its current location *b2* and the location *b1* on the balcony directly above. But *b1* is inaccessible from the current location. Point *a1* is a valid location where as *a2* is not. There are many such invalid locations. The human can not walk underneath a portion of the stair case marked as 'forbidden region', but the region above remains valid. Various scenarios in the environment are encapsulated through *obstacle types*.

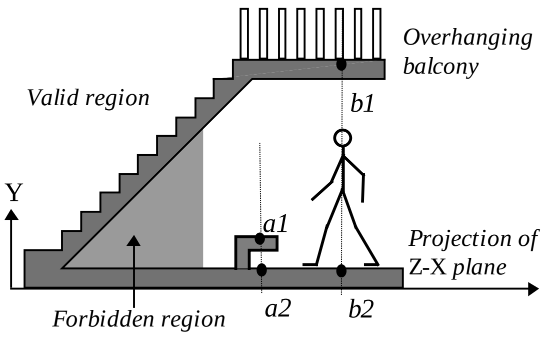


Figure 4. A 3D environment with multiple locations along the vertical axis

3.3.1. Obstacle types

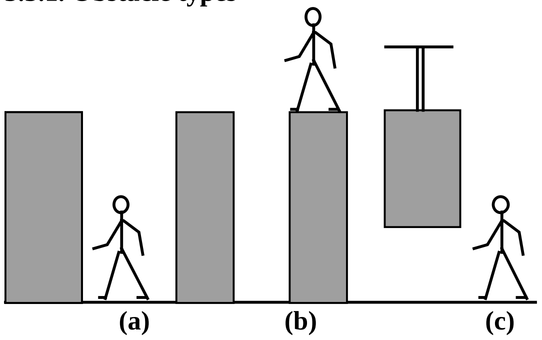


Figure 5a. Insurmountable obstacles: the human can not (a) climb (b) cross (c) pass through obstacles

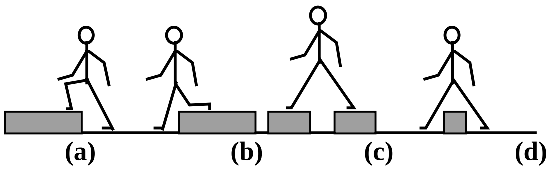


Figure 5b. Surmountable obstacles: the human can (a) step on (b) step down (c) cross (d) step over obstacles

Obstacles are of two types: *surmountable* and *insurmountable* obstacles. For this classification it is assumed that the human always walks upright and *does not* bend. This prevents human from going underneath low elevations such as 'forbidden regions' in Figure 4. Figure 5 shows some obstacles of both the types. Insurmountable obstacles, as the name indicates consist of objects the human can not step on or step over (such as walls, pillars), gaps too large to cross, elevations of roof too low to let a human of certain height pass through. Very narrow regions through which human can not pass through also constitute such obstacles. Surmountable obstacles are of the opposite kind. These are negotiable by the human (for example climbing stair cases). Whether an obstacle is surmountable or not is decided by the reachability criteria:

Reachable Cells (3D), border cells

Following condition is added to the 2D criteria for reachability between ***p*** and ***q***:
p and ***q*** should be separated by no more than the *vertical foot span* of the human (to permit ascending or descending obstacles) and the cells above these two should be free of obstacles at least up to the height of human (to provide enough room for standing upright).

Border cell: *A cell that has at least one unreachable (immediate or extended) 4-neighbor.*

It can be seen that reachability is a *local* concept. The point *b1* in Figure becomes reachable once the human reaches the balcony. The closure of all reachable cells forms a reachable region. Thus, formerly insurmountable obstacles might become reachable if they fall inside this region.

Extended cell neighbors in 3D

In 2D, when neighboring cells are missing, extended neighbors are found on the same plane. In 3D, extended neighbors may be found on a different plane, if immediate neighbors do not exist. The neighbor should be such that when the human is crossing over to it, no obstacle is present in between. Let the human is standing on its current cell with its head occupying a cell with Y component *hy*. Let the prospective neighboring cell be (*nx,ny,nz)* with *hy*>*ny*. If all the cells in the vertical column between (*nx,ny,nz)* and *(nx,hy,nz)* are free of obstacles, then it is a proper neighbor cell. This cell still has to meet the reachability criteria before the human can step on that. The navigation front enumerates only reachable cells. Thus the reachable cell region lies only on the surfaces of navigable objects reducing the large search space of cells.

Graph connectivity for surmountable and insurmountable obstacles

All cells on the insurmountable obstacles are locally unreachable and are equivalent to holes as shown in the left picture in Figure 6. Any feasible path for the human should completely avoid insurmountable obstacles, i.e. holes generated by them. For example, the path from A to B (Figure 6, left) is a projected as a path from A' to B' in 2D. A direct path is blocked by the hole region in the dark shade. Once the hole regions for the 3D case are defined, it can be handled by the conventional 2D algorithm.

Surmountable obstacles such as stairs bridge motion from one planar region to the other paving way for three dimensional movement. The regions with insurmountable obstacles can be locally viewed as normal 2D regions with holes. The surmountable obstacles can be viewed as connections between those regions as shown in Figure 6. Even though it is shown as a sequential list, the nodes may be interconnected like in a graph.

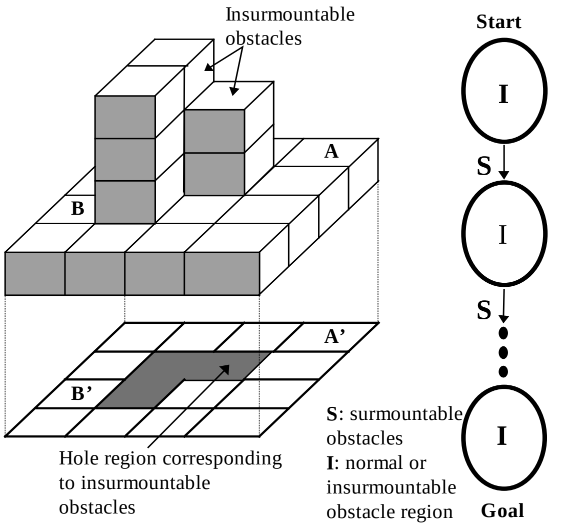

Insurmountable obstacles
A
B
A'
B'
Hole region corresponding to insurmountable obstacles
Start
I
S
I
S
I
Goal
S: surmountable obstacles
I: normal or insurmountable obstacle region

Figure 6. Insurmountable obstacles in 3D as holes in 2D (left), over all connectivity in the search graph (right)

3.3.2. Border Cells

The hole regions projected by insurmountable obstacles such as walls, pillars etc. are inaccessible from the surrounding regions and acquire border cells around. Border cells are avoided, even though they are technically valid locations for the human to occupy. For example a human does not walk too close by the walls even if such a path guarantees a shortest path.

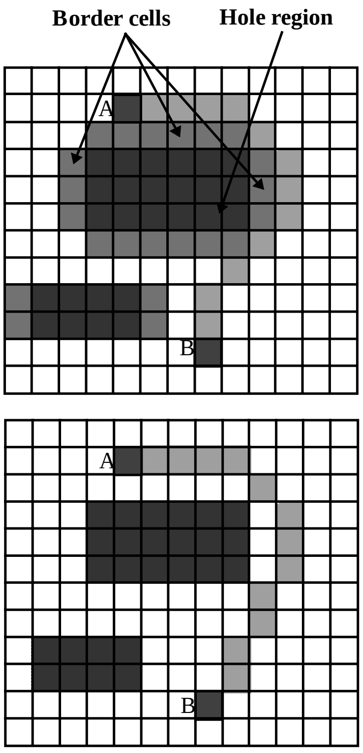


Figure 7. Avoiding border regions in the navigation path

Figure 7 shows the previous picture with an additional hole region. If the horizontal foot span is three cells, then all the cells abutting the bigger hole region become border cells as they can not be reached from the side adjacent to the hole. The smaller hole has border cells only on its shorter side. For human navigation, this is not strictly a hole region. The navigation path that avoids border region is shown in the top picture. After the DDA refinement, the direct path is generated as shown in the lower picture. Note that the cells adjacent to four corners of the hole are not considered border cells as all of their 4-neighbors are in non-hole region and hence reachable. If the definition includes all 8 (extended) neighbors, then even the corner cells would become unreachable as one of the 8-neighbors comes from the hole region.

3.3.2.1. Generating Border Cells

Border cells are identified by making an initial run of unoptimized floodfill on the discretized scene. At each step, a cell is expanded into all reachable 4-neighbors (or 8-neighbors if diagonal border cells are allowed). If any neighboring cell is unreachable (due to an insurmountable obstacle), then the cell is marked as a border cell. This continuous until all the cells have been generated. The actual path generation takes place as before with another run of floodfill. This cell-path avoids all border cells, resulting in a border-free optimal navigation path.

3.3.3. DDA Refinement of the path

In 3D case, the algorithm works exactly the same with some modification to the vertical Y component. The Y component of any newly enumerated cell is taken to be that of the immediate predecessor cell. If the previous cell and the new cell generated are unreachable, then the DDA path is deemed to have failed. The original path segments are retained.

3.3.4 Walking

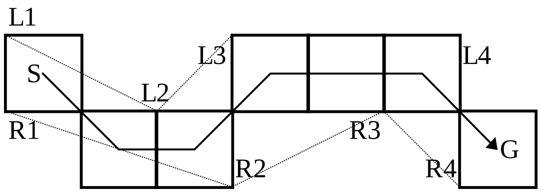


Figure 8. Next placement of the right foot

Once the path is generated, the human should step through the sequence of cells in the cell path. Various schemes are possible to simulate the motion, but in the present implementation, the direction of movement is maintained to be along the vector connecting the center of the current cell with that of the next. The projection of the center of the human should follow the line segments from S to G connecting cell centers, with foot locations on either side of it (Figure 8). The moving foot is placed at one of the corners of the next cell that satisfies these constraints. The figure shows a typical path sequence for left and right leg movements. The motion sequence is: L1L2-R1R2-L2L3-R2R3-L3L4-R3R4 in that order, where L and R are alternating left and right leg movements respectively. This placement technique somewhat mitigates abrupt orthogonal turns compelled by the discrete path. Though, this illustration shows movement from one cell to the next, the next cell movement can be more than unity if bigger strides are needed. The local movement of the foot from one location to the next is described in detail elsewhere [Ban97].

4. Results

labyrinth

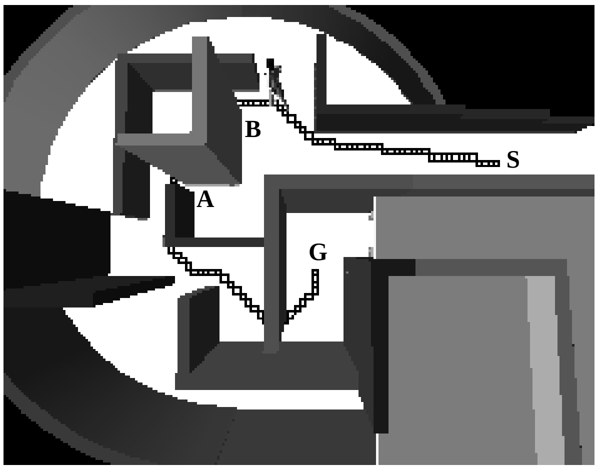


Figure 9. Labyrinth

Figure 9 shows a human placed in a labyrinth kind of structure. The scene is an Inventor file. The entire scene is discretized in the beginning and the occupancy status is stored in the memory. A 3D cursor (a straight line intersecting objects) facilitates interactive selection of the goal location **G** in the scene. The path starts at the location **S**. The human is unable to cross very narrow corridors even if its physical dimensions can be accommodated. This is due to cell size. The corridors less than one cell wide can not accommodate cell paths. However a cell path is guaranteed when the corridor width is at least twice the cell size. The widths between one and two cells may or may not have paths depending on the location of cells. In the figure the location **A** has a very narrow corridor while location **B** has a wider corridor and selected by the navigation algorithm. The path is smoothened using DDA.

Stairs and obstacles

Figure 10 shows a scene with several obstacles and a staircase. The scene is defined inside the program and unlike in the case of Inventor scenes, the geometry is completely known. However, the scene map is constructed in the same way as for Inventor files by rasterizing them.

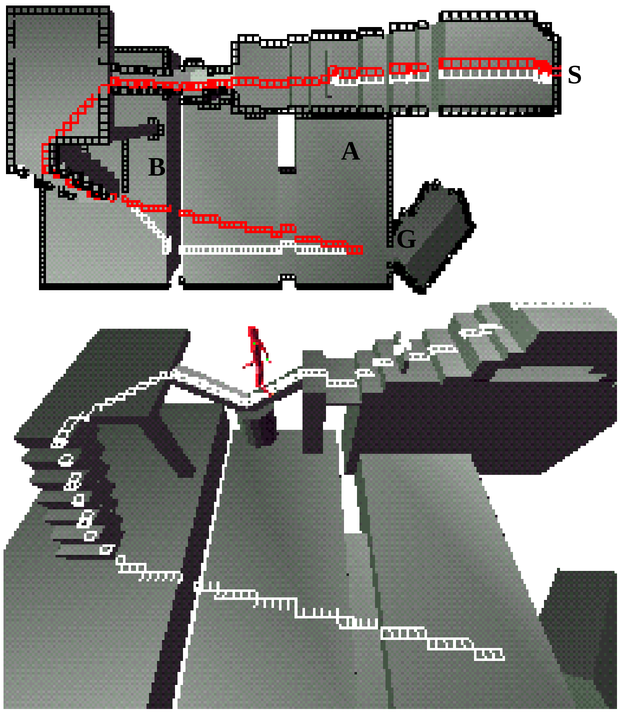


Figure 10. Walking on stairs

Figure 10 (top picture) shows top view of the scene. The dark cells represent the border regions which are off-limits to the human. There are also some hole regions. A border region is abutting every hole region. However not all sides of hole regions have borders (for example at location **A**; compare with smaller hole in Figure 7). Also notice a border below the staircase at **B**. The human can not stand upright within this region without colliding with the staircase (compare with forbidden region in Figure 4). The white cells are generated in the original path from **S** to **G**. After smoothening, it is converted into the path in the bottom picture, which shows a snapshot from the navigation along the final path enumerated.

Walking inside a factory

The factory floor scene is a public domain Inventor file from *3dcafe*[2]. The process of path finding is illustrated in Figure 11. The top picture shows creation of border regions using the floodfill algorithm. The white cells are reachable regions human can walk on. The region below the overhanging structure on the right can not accommodate the human standing upright. This is reflected as a hole in the 2D grid. Bottom picture shows the path generated using another round of floodfill.

[2] http://www.3dcafe.com

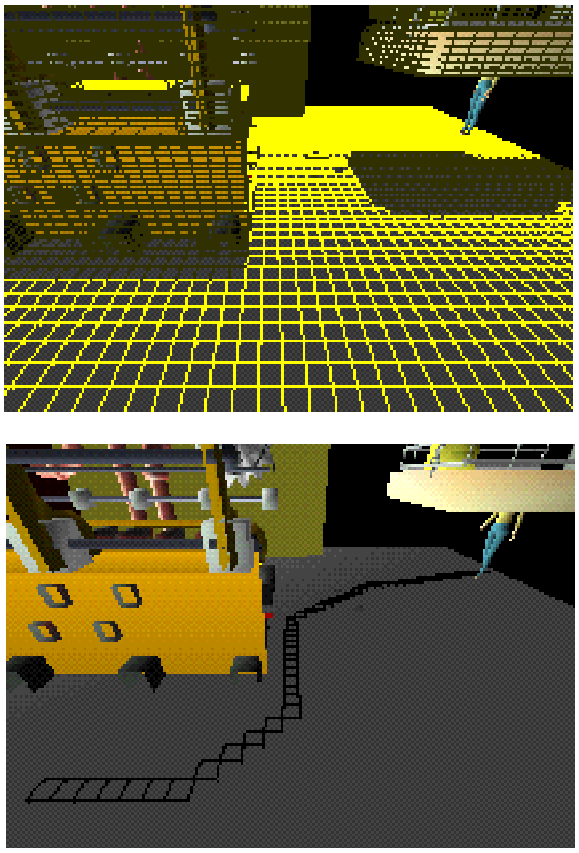


Figure 11.Factory floor: floodfill for borer cells (above), path generation (below)

Memory requirements and timing statistics

Table 1 lists timing statistics for the examples shown. Timings are as measured by a wall-clock. For files of similar sizes the discretization time is dependent on the vertical resolution. The border generation spans all reachable regions from a given seed cell, usually the goal cell. Factory floor has less space to walk on than stairs and takes up much less time for border generation. After these two steps, the path finding is almost instantaneous taking at most a second.

Each grid cell takes up 2 bytes, one for indicating occupancy status and other indicating if it is a border cell. Another grid stores costs generated in the navigation front. Each cell requires one byte for indicating whether it is already enumerated and a 4-byte integer quantity for cost of the cell. Thus, each cell needs a total of 7 bytes. A 100x100x100 grid requires a minimum of 7 Mb. With suitable coding scheme a single byte may be shared to indicate various status flags for reducing run-time memory. The heuristic search needs dynamic memory allotment that varies with the distance between start and goal cells.

	file size (Kb)	Grid size x,y,z	discretization, border generation
labyrinth	443	72,23,93	3, 6 secs
stairs	--	53,102,91	5, 10 secs
factory	609	69,95,69	40, 8 secs

Table 1. Timing statistics
(Executed on SGI Indigo2 Impact,R10000 CPU)

5. Conclusion

A method for global navigation of human in a 3D environment of obstacles is presented. The method is efficient and simple to implement. This can serve as a higher level mechanism for local foot placement methods. Some important aspects are yet to be explored: presently the semantics of object space is ignored. The model treats every object as an obstacle. For example a low level table and a step on the staircase are identical and human might walk on them. Resolving such common sense issues involves assigning meaning to objects and should be done at a higher level. Another drawback is with curved objects. The discretization is coarse and stepping on them does not look very natural. Assigning weight to objects based on amount of occupancy is being explored using super sampling of each grid cell. The cells with low weights are discarded during the path generation. A chief extension planned for the future is simulating motion of multiple humans with out colliding with each other. Another direction of further research involves environment with moving obstacles.

Acknowledgments

This research was partly sponsored by Swiss National Research Foundation. The authors would like to thank Olivier Aune for designing some of the scenes.

References

[Arm85] Armstrong, W.W., Green, M.W., The Dynamics of Articulated Rigid Bodies for Purposes of Animation, *The Visual Computer* , Springer-Verlag (1985) 1: 231-240

[Ban95] Srikanth Bandi, Daniel Thalmann, An Adaptive Spatial Subdivision of the Object Space for Fast Collision Detection of Animated Rigid Bodies, Eurographics '95 Conference Proceedings, Computer Graphics Forum, Vol.14, No.3, pp.C259-C270 The Eurographics Association, 1993

[Ban97] Srikanth Bandi, Daniel Thalmann,"A Configuration Approach for Efficient Animation of Human Figures", IEEE Non-Rigid and Articulated Motion Workshop, Puerto-Rico, June 15, 1997

[Ban98] Srikanth Bandi, Daniel Thalmann "The Use of Space Discretization for Autonomous Virtual Humans" Video Paper, Second International Conference on AUTONOMOUS AGENTS, Minneapolis/St. Paul, May 10-13, 1998)

[Bou90] Boulic Ronan, Thalmann Daniel, A global Walking Model with Real-time Kinematic Personification, *Visual Computer*, Springer-Verlag (1990) 6:344-358

[Bru89] A.Bruderlin, T.W.Calvert, Goal Directed, Dynamic Animation of HumanWalking, *Computer Graphics Proceedings*, Annual Conference Series (August, 1989), ACM SIGGRAPH, 233-242

[Cof95] Juditha Cofman, "Numbers and Shapes Revisited", Oxford University Press (1995), pp.83,245,246.

[Dea91] T.L.Dean, M.P.Wellman, Planning and Control, Morgan Kaufman Publishers 1991

[Gir85] Girard Michael, Maciejewski, A.A., Computational Modeling for the Computer

Animation of Legged Figures, *Computer Graphics*, 19, 3 (July, 1985), 263-270

[Hod95] J.K.Hodgins et al. “Animating Human Athletics”, *Computer Graphics Proceedings*, Annual Conference Series (August, 1995), ACM SIGGRAPH, 71-78

[Hye96] Hyeongseok Ko, N.I.Badler, “Animating Human Locomotion with Inverse Dynamics”, *IEEE CG&A*, March 1996, 50-59

[Isa87] Isaacs, P.M., Cohen, M.F., “Controlling Dynamic Simulation with Kinematic Constraints, Behavior Functions and Inverse Dynamics”, *Computer Graphics*, 21, 4 (July, 1987), 215-224

[Kog94] Yoshihito Koga et al., Planning Motion with Intentions, *Computer Graphics Proceedings*, Annual Conference Series (July, 1994), ACM SIGGRAPH, 395-408

[Kor82] Korein, J.U., Badler, Norman I., Techniques for Generating the Goal- Directed Motion of Articulated Structures, *IEEE CG & A*, (November, 1982) 71-81

[Len90] Jed Lengyel et al., Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware, *Computer Graphics Proceedings*, Annual Conference Series (August, 1990), ACM SIGGRAPH, 327-335

[Loz79] Tomas Lozano-Pérez, Michael A. Wesley, “An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles”, *Communications of the ACM*, Vol.22, No.10, October, 1979, pp.560-570

[Loz83] Tomas Lozano-Pérez, “Spatial Planning: A Configuration Space Approach”, *IEEE Transactions on Computers*, Vol.C-32, No.2, February, 1983, pp.108-120

[McK90] McKenna, Michael, Zeltzer David, Dynamic Simulation of Autonomous Legged Locomotion,*Computer Graphics*, 24, 4 (August, 1990), 29-38

[Mit88] Joseph S.B.Mitchell, “An Algorithmic Approach to Some Problems in Terrain Navigation”, in *Geometric Reasoning*, Deepak Kapur, Joseph L. Mundy (editors), Elsevier Science Publishers B.V, 1988

[Nil80] Nilsson, N.J, “Principles of Artificial Intelligence”, Tioga, Palo Alto, CA, 1980

[Nos97] Hansrudi Noser, “A behavioral Animation System Based on L-Systems and Synthetic Sensors for Actors”, *PhD Thesis*, Thesis No.1609, 1997, Ecole Polytechnique Federale de Lausanne, Swtizerland.

[Ove97] C.W.A.M. Van Overveld, H.Ko, Small Steps for Mankind: Towards a Kinematically Driven Dynamic Simulation of Curved Path Walking, *The Journal of Visualization and Computer Animation*, Vol.5:143-165 (1994)

[Phi90] Philip Lee et al. Strength Guided Motion, *Computer Graphics Proceedings*, Annual Conference Series (August, 1990), ACM SIGGRAPH, 253-262

[PJM91]Phillip John McKerrow, “Introduction to Robotics”, Addison-Wesley, 1991.

[Rai91] M.H.Raibert, J.K.Hodgins, Animation of Dynamic Legged Locomotion, *Computer Graphics Proceedings*, Annual Conference Series (July, 1991) ACM SIGGRAPH, 349-358

[Rei94] B.D. Reich, H.Ko, W.Becket, N.I.Badler, Terrain Reasoning for Human Locomotion, *Computer Animation ’94*, pp.76-82

[Rog85] David F.Rogers, "Procedural elements for computer graphics", McGraw-Hill, pp.30-39, 1985.

[Sch88] J.T.Schwartz, M.Sharir “A Survey of Motion Planning and Related Geometric Algorithms”, in *Geometric Reasoning*, Deepak Kapur, Joseph L. Mundy (editors), Elsevier Science Publishers B.V, 1988, pp.157-169

[Van96] M. Van de Panne, From Footprints to Animation, *Computer Graphics Forum*, Vol. 16(1997), No.4, The Eurographics Association, pp.211-223.