# Way-Finder: Guided Tours Through Complex Walkthrough Models.

Pere-Pau Vázquez Alcocer
Universitat Politècnica de Catalunya
68 PUBLICATIONS 877 CITATIONS

Marta Fairén
Universitat Politècnica de Catalunya
19 PUBLICATIONS 190 CITATIONS

EUROGRAPHICS 2004 / M.-P. Cani and M. Slater
(Guest Editors)

*Volume 23 (2004), Number 3*

# Way-Finder: guided tours through complex walkthrough models

C. Andújar, P. Vázquez, and M. Fairén

Dept. LSI, Universitat Politècnica de Catalunya, Barcelona, Spain

**Abstract**

*The exploration of complex walkthrough models is often a difficult task due to the presence of densely occluded regions which pose a serious challenge to online navigation. In this paper we address the problem of algorithmic generation of exploration paths for complex walkthrough models. We present a characterization of suitable properties for camera paths and we discuss an efficient algorithm for computing them with little or no user intervention. Our approach is based on identifying the free-space structure of the scene (represented by a cell and portal graph) and an entropy-based measure of the relevance of a view-point. This metric is key for deciding which cells have to be visited and for computing critical way-points inside each cell. Several results on different model categories are presented and discussed.*

## 1. Introduction

Advances in modeling and acquisition technologies allow the creation of very complex walkthrough models including large ships, industrial plants and architectural models representing large buildings or even whole cities.

These often densely-occluded models present a number of problems related to wayfinding. On one hand, some interesting objects might be visible only from inside a particular bounded region and therefore they might be difficult to reach. On the other hand, walls and other occluding parts keep the user from gathering enough reference points to figure out his location during interactive navigation. This problem becomes more apparent in indoor scenes which often include closed, self-similar regions such as corridors. Finally, architectural and furniture elements can become barriers in collision-free navigation systems. For instance, smooth navigation through turning staircases or narrow passages might require advanced navigation skills.

As a consequence of the above problems, the user may wander aimlessly when attempting to find a certain place in the model, or may fail in finding again places recently visited. Sometimes the user is also unable to explore the whole model or misses relevant places.

One obvious solution to these problems is to provide the user with different navigation aids such as maps showing a sketch of the scene along with the current camera position. This solution alleviates the problem of disorientation, but still the user can miss important parts. Moreover, automatically generating illustrative maps is not an easy task. Other useful navigation aids such as somehow marking already-visited places are not enough for guaranteeing a profitable exploration.

Another solution is to explore the model following a precomputed path or a selection of precomputed viewpoints. This path can be provided by the model creator or by an experienced user who already knows the interesting regions of the scene, but this is not always feasible. Moreover, this can also become a disadvantage if we cannot express properly which are the regions that are important for us to visit. In any case, this solution also requires a noticeable user effort during the path definition.

In this paper we present an algorithm for the automatic construction of exploration paths. Given an arbitrary geometric model and a starting position, the algorithm computes a collision-free path represented by a sequence of nodes, each node having a viewpoint, a camera target and a time stamp. The algorithm proceeds through three main steps. First, a cell-and-portal detection method identifies the overall structure of the scene; second, a measurement algorithm is used to determine which cells are worth visiting, and finally, a path is built which traverses all the relevant cells.

The rest of the paper is organized as follows. Section 2 reviews previous work on automatic path generation and cell-and-portal detection. Section 3 presents a characterization of the properties that we consider suitable for a camera path. Section 4 gives an overview of our approach. Section 5 presents our algorithm for the automatic generation of cells and portals and Sections 6 and 7 explain how the more relevant cells are determined and how the exploration path is built respectively. We present some results in Section 8 and conclude our work pointing some lines of future work in Section 9.

## 2. Previous Work

Motion planning has been extensively studied in robotics, computational geometry and related areas for a long time. However, it is still considered to be a difficult problem to solve in its most basic form, e.g., to generate a collision-free path for a movable object among static obstacles. As stated by Canny [1], the best known complete algorithm for computing a collision-free path has complexity exponential in the number of degrees of freedom of the robot or the moving object. Good surveys can be found in [2] and [3].

Some approaches for motion planning present algorithms formulated in the configuration space of a robot. The configuration space (also known as *C-space*) is the set of all possible configurations of a mobile object. Isto presents two approaches, the first one [4] computes a decomposition of the *C-space* and searches the graph connecting collision-free areas of the decomposition for a correct path. The second one [5] divides the search algorithm in two levels: a global search and a local search.

Other sorts of algorithms are based on randomized motion planning. Li *et al.* [6, 7] take input from the user and predict the location where the avatar should move to. However, this approach has only been used for navigation in simple environments due to its high running time. Salomon *et al.* [8] present an interactive navigation system that uses path planing. The path is precomputed using a randomized motion planning with a reachability-based analysis. It computes a collision-free path at runtime between two specified locations. However, their system still needs more than one hour to compute a roadmap for relatively simple models (ten thousand polygons) and sometimes the results are unnatural paths. Kallmann *et al.* [9] present a new method that use motion planing algorithms to control human-like characters manipulating objects which allow up to 22 degrees of freedom.

In our approach, the configuration space depends on the spatial structure of the scene and we want to explore it by means of cells and portals, so the graph we need is completely different, we need a *cell-and-portal* graph.

A cell-and-portal graph (CPG) is a structure that encodes the visibility of the scene, where nodes are cells, usually rooms, and edges are portals which represent the openings (doors or windows) that connect the cells. The construction of a CPG is commonly done by hand, so it is a very time consuming task as the models become more and more large and complex. The automatic generation of portals and cells is therefore a very important issue. There are few papers that refer to the automatic determination of portal-and-cell graphs, and most of them work under important restrictions. Teller and Séquin [10] have developed a visibility preprocessing suitable for axis-aligned architectural models. Hong *et al.* [11] take advantage of the tubular nature of the colon to automatically build a cell graph by using a simple subdivision method based on the center-line (or skeleton) of the colon. To determine the center-line, they use the distance field from the colonic surface. Haumont *et al.* [12] present a method that adapts the 3D watershed transform, computed on a distance-to-geometry sampled field. However, their method only works on cells free of objects, and therefore these have to be removed previously by hand.

## 3. Camera path characterization

Given a geometric model, there is an infinite number of paths exploring it. In order to compute paths algorithmically we have to identify which are the properties that distinguish a suitable path from non-useful ones. The following list presents the main properties users might expect from a camera path.

- *Collision-free*
  Ideally, a camera path should be free from collisions with scene objects. However, this is not always feasible since the input scene might contain interesting parts bounded by closed surfaces which will be impossible to reach using this criterion strictly. Therefore we require our paths to not cross any wall unless it is the only way to enter a cell bounded by a closed surface.
- *Relevant*
  A good path must show the user the most *relevant* parts of the model while skipping non relevant or repetitive parts. Relevance is a subjective quality that depends on user interests, but requiring the user to identify and mark relevant objects would compromise the scalability of our approach. As a consequence, a metric for estimating relevance is required. One contribution of this paper is the use of entropy-based measurements for quantifying the relevance of a given viewpoint.
- *Non-redundant*
  Ideally, a camera following the path should visit each place only once. Again, this is often not possible e.g. traversing the same corridor many times can be the only way to visit all relevant rooms. We therefore require our algorithm to avoid already visited places whenever possible.

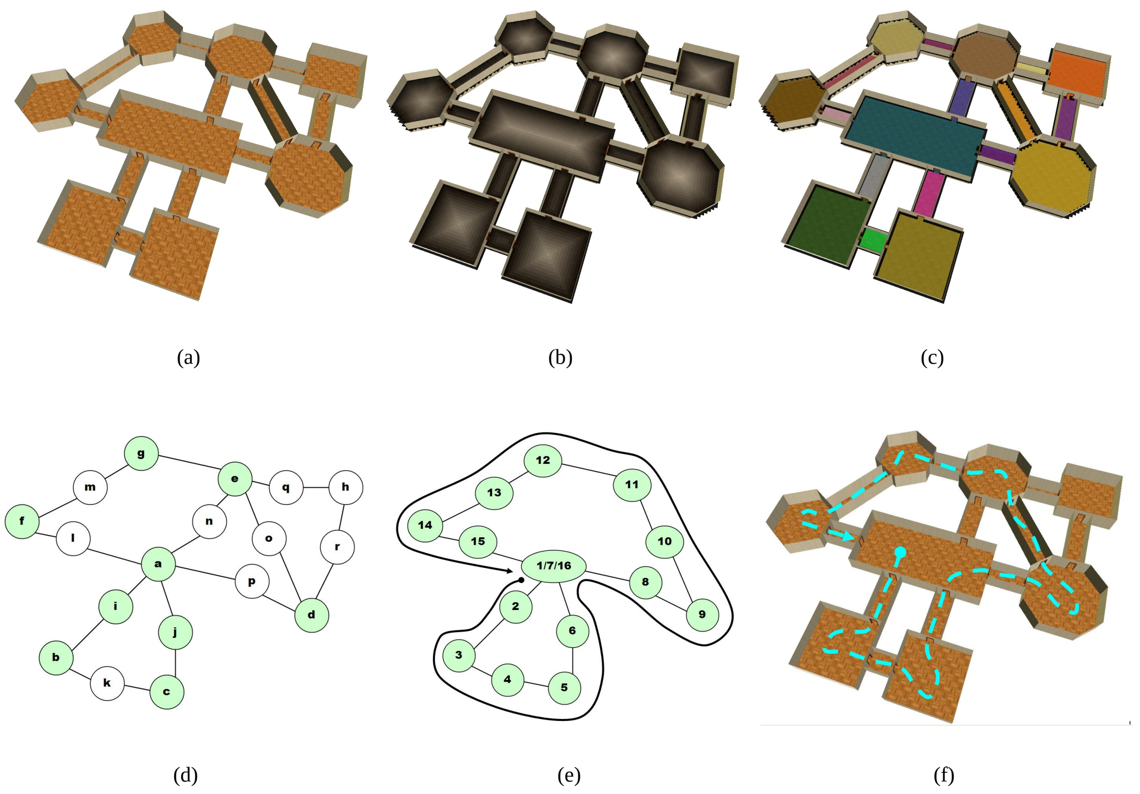

**Figure 1:** *Overview of our approach. (a) Input scene (furniture and ceiling not shown); (b) Distance-to-geometry field computed over the 3D grid (only one slice is shown); (c) Cells detected with random color (note that corridors are also identified as cells); (d) Cell-and-portal graph embedded in the model space; cells are labeled according to relevance measure; (e) High-level path computed as a sequence of cells; visited cells is a superset of relevant ones; (f) Final path after smoothing (camera target not shown).*

- *Uncoupled target*
  In most online navigation systems, the camera target is defined in accordance with the forward direction of the viewpoint as this facilitates the camera control. However, precomputed paths do not benefit from this limitation. Uncoupling the camera target from the advance direction is often desirable because it allows the user e.g. to watch the paintings on the ceiling of a room while crossing it.
- *Ordered*
  This property is closely related to the non-redundancy criterion. The path should not leave a room unless all the relevant details it contains have been visited.
- *Self-adjusting speed*
  In addition to let the user modify the camera speed during the reproduction of the path, it is also convenient to define the path so that the speed is defined in accordance with the relevance of the part of scene being seen. This implies that the speed increases while traversing open spaces with distant details or when walking through already visited places. Similarly, the speed decreases while approaching relevant objects.
- *Smooth*
  The path creator should try to avoid abrupt changes in speed, camera position and camera target.

## 4. Algorithm overview

Our algorithm receives as input an arbitrary walkthrough model and a starting camera position and computes a collision-free path represented by a sequence of nodes, each node having a viewpoint, a camera target and a time stamp.

The algorithm proceeds through three main steps (see Figure 1).

First, we identify the free-space structure of the scene by computing a cell and portal graph $G = (V, E)$ over a grid decomposition (Section 5). Our cell and portal graph differs from the ones used for visibility computation in that we do not need to classify the scene geometry against the cells nor do we need to compute the exact shape of the portals. In fact, our cells are simply represented by a collection of voxels and for each portal we just need a single way-point. This cell decomposition allows the algorithm to produce paths with minimum redundancy where cells are visited in a natural way, the portals being suitable waypoints.

In a second step (Section 6) we use an entropy-based measurement algorithm to identify the cells in $V$ that are worth visiting (*relevant* cells). This step filters out non-interesting cells and also ensures the robustness of the algorithm against an over-decomposition of the scene into cells due to geometric noise.

The last step builds a camera path which traverses all the relevant cells and visits the more interesting places inside each cell (Section 7). This task is accomplished at two levels. We first decide *in which order the relevant cells should be visited* by computing a path $H$ over the cell-and-portal graph. We call $H$ the *high-level* path, which is just an ordered sequence of cell identifiers and portals connecting adjacent cells. For this task the algorithm must find the shortest path traversing all the relevant cells while minimizing the traversal of non relevant cells and repeating cells. At this point our path contains only a few waypoints which correspond to the portals connecting adjacent cells on the high-level path. The next task is to decide how to refine the path *inside each cell*. This is accomplished by computing a sequence of waypoints for visiting each cell from an entry-point to an exit-point. Again the entropy-based measure is used for deciding both the waypoints and the best camera target at each viewpoint. Note that both entry and exit points are just the center of the portals connecting the current cell with the previous cell and the next cell respectively. Finally, a simple postprocess smoothes the path and adjusts the speed in accordance to the precomputed relevance of the viewpoints.

## 5. Automatic portal and cell detection

The creation of the cell-and-portal graph pursues two aims. On the one hand, the cell decomposition provides a high-level unit for evaluating the relevance of a region and for deciding whether this region should be visited or not. Moreover, this decomposition allows for solving the problem of finding collision-free paths considering only one cell at a time. On the other hand, the portal detection provides a first insight into the final path because portals are natural waypoints.

Our approach for computing the cell-and-portal graph is based on conquering quasi-monotonically decreasing regions on a distance field computed on a grid. The cell detection is organized in successive stages explained in detail below. First, we build a binary grid separating empty voxels from non-empty ones. Next we approximate the distance field using a matrix-based distance transform. Then we start an iterative conquering process starting from the voxel having the maximum distance among the remaining voxels. During this process, all conquered voxels are assigned the same cell ID. A final merge step eliminates small cells produced by geometric noise. Finally, faces shared by voxels with different cell ID's are detected and portals are created at their centers.

### 5.1. Distance field computation

The first step converts the input model into a voxel representation encoded as a 3D array of real values. Voxels traversed by the boundary of the scene objects are assigned a zero value whereas empty voxels are assigned a $+\infty$ value. This conversion can be achieved either by a 3D rasterization of the input model or by a simultaneous space subdivision and clipping process supported by an intermediate octree [13].

The next step involves the computation of a distance field (Figure 1-b). The distance field of an object is a 3D array of values, each value being the minimum distance to the encoded object [14]. Distance fields have been used successfully in generating cell-and-portal graph decompositions [12]. The distance field we consider here is unsigned. Distance fields can be computed in a variety of ways (for a survey see [14]). We approximate the distance field using a *distance transform*. Distance transforms can be implemented through successive dilations of the non-empty voxels and more efficiently by a two-pass process. The Chamfer distance transform [14] performs two passes through each voxel in a certain order and direction according to a distance matrix. The local distance is propagated by the addition of known neighborhood values provided by the distance matrix. In our implementation we use the 5x5x5 quasi-euclidean chamfer distance matrix discussed in [14]. Indeed, our experiments show that computing the distance field on a horizontal slice of the voxelization (using the central 5x5 submatrix) leads to better cell decompositions as it limits the influence of the floor and ceiling and it is less sensitive to geometric noise caused e.g. by furniture. Note that the maxima of the distance transform (white voxels in Figure 1-b) can be seen as an approximation of the Medial-Axis Transform.

### 5.2. Cell generation

The cell decomposition algorithm visits each voxel of the distance field and replaces its unsigned distance value by a cell ID. We use negative values for cell ID's to distinguish visited voxels from unvisited ones. The order in which voxels are visited is key as it completely determines the shape and location of the resulting cells.

The order we propose for labeling cells relies on a conquering process starting from the voxel having the maximum distance among the remaining unvisited voxels. This local maximum initiates a new cell whose ID is propagated using a breadth-first traversal according to the following propagation rule. Let $v$ be the voxel being visited, and let $D_v$ be the

```
procedure cell_decomposition
    cellID = -1
    S = sort_voxels()
    while not_empty(S) do
        (i,j,k) = pop_maximum(S)
        if grid[i,j,k]>0 then
            expand_voxel(i,j,k,cellID)
        end
        cellID = cellID - 1
    end
end
```

**Figure 2:** *Cell decomposition algorithm*

distance value at voxel $v$. The current cell ID is propagated from $v$ to a face-connected neighbor $v'$ if $0 < D_{v'} \leq D_v$, i.e. the distance value at $v'$ is positive but less or equal than the distance at $v$. The propagation of the cell ID continues until the whole cell is bounded by voxels having either negative distance (meaning already visited voxels), zero distance (non-empty voxels) or positive distance greater than the voxels at the cell boundary. Then, a new unvisited maximum is computed and the previous steps are repeated until all non-zero voxels have been assigned to some cell (Figure 2).

Furniture and other scene objects might exert a strong influence on the distance field, causing many local maxima to appear and therefore producing an over-segmentation of the cell decomposition. A straightforward solution could be to remove by hand all furniture elements before the model is converted into a voxelization, which is the solution adopted in [12]. The solution we propose is to relax the propagation process by including a decreasing tolerance value in the propagation rule: the ID is propagated from $v$ to $v'$ if $0 < D_{v'} \leq D_v + \varepsilon$, where $\varepsilon$ vanishes to zero as the cell grows. The consequence of this aging tolerance is that small variations of the distance field near the cell origin do not impact their propagation. This variation is less sensitive to noise than a watershed transform considering simultaneously all local maxima [12]. The connectivity used during the propagation process is 4-connectivity in 2D and 6-connectivity in 3D. A two-dimensional propagation suffices e.g. when the camera height (with respect to the floor) remains constant during the path.

## 5.3. Cell merging and portal detection

A cell merging process further improves the cell decomposition by merging uninteresting cells. Let $|A|$ be the size of cell $A$, measured as the number of voxels, and let $Portal(A,B)$ be the number of voxels shared by cells $A$ and $B$. We use the following merging rules: (a) if $|A|$ is smaller than a given minimum size then the cell is merged with the cell sharing the large number of boundary faces with $A$; if no such a cell exist (i.e. $A$ is bounded by 0-distance voxels) then the cell $A$ is removed; (b) if $Portal(A,B)$ is greater than a maximum portal size, then cells $A$ and $B$ are merged into a single cell. The results shown in Section 8 have been computed using only the first rule.

The graph nodes in $V$ correspond to the identified cells and the graph edges in $E$ correspond to links between adjacent cells. Besides the graph connectivity, each cell is represented by a collection of face-connected empty voxels and a graph edge connecting two cells is represented by the collection of portals shared by the two cells. Portal detection is straightforward and requires a single traversal of the voxels identifying faces shared by voxels with different IDs. Portals correspond to connected components of shared voxels. Each portal is assigned a single point that can be computed as the portal center. An alternative which works better for non-planar portals consists in keeping the distance field values during the cell generation process (instead of re-using these values for storing the cell IDs) and compute the portal representant as the voxel with the highest value on the distance field (i.e. the point on the portal farthest from the nearby geometry). These points are candidates for waypoints in case the path has to cross the portal for going from one cell to another.

## 6. Identifying relevant cells

Once we have determined the graph of portals and cells, the following step is to determine which are the cells that are worth visiting in the model.

The data structure built to determine the cell-and-portal graph is also useful to give a set of points inside each cell. Given this set of points, we can determine if the cell is relevant by measuring the amount of information that can be seen from each point of the set. In order to compute the amount of information we use an entropy-based measure, dubbed viewpoint entropy, which has been successfully applied to determine the best view of objects and scenes [15]. We measure and store the point of maximum entropy for each cell and then choose those cells that have a higher relevance. These selected cells will be the *relevant cells* to be visited.

### 6.1. Viewpoint Entropy

Viewpoint entropy is a measure based on Shannon entropy [16]. It uses the projected areas of the visible faces on a bounding sphere of the viewpoint to evaluate how much of the visible information can be seen from the point. For a single view, we only need to render the scene into an item buffer from the viewpoint. Then, the buffer is read back and we sum the area of each visible polygon (actually as we should render to a sphere, we weigh each pixel by its subtended solid angle, to calculate entropy properly). Then, the relevance is computed using the following formula:

$$H_p(X) = -\sum_{i=0}^{N_f} \frac{A_i}{A_t} \log \frac{A_i}{A_t},$$

where $N_f$ is the number of faces of the scene, $A_i$ is the projected area of face $i$, $A_t$ is the total area covered over the sphere, and $A_0$ represents the projected area of background in open scenes. In a closed scene, or if the point does not *see* the background, the whole sphere is covered by the projected areas and consequently $A_0 = 0$. The maximum entropy is obtained when a certain point can *see* all the visible faces with the same relative projected area $A_i/A_t$. To cover all the surrounding space we need six projections (similarly to the cube map construction process).

### 6.2. Relevance cell determination

To detect the important cells, we select a set of viewpoints inside each cell. The candidate points are given by the cell detection algorithm (for example one per voxel if voxel size is small enough). The viewpoint entropy of each candidate point is evaluated and stored in order to select the best one, whose entropy will indicate the relevance of the cell. Usually, the cells detected in the first step will be relatively free of objects, and large occluders will naturally determine new portals and cells. Throughout the process of determination of the relevance of a cell, we can store the visible projected areas of each face for each evaluated viewpoint. Then, we can determine the best set of views by iteratively selecting the best one, marking the already visited faces, and recomputing the entropy values for the rest of the views only taking into account the not yet visited faces [15]. If almost all the visible faces were visible from the best view, this means that there are no large occluders in our cell. Otherwise we select more than one important point in the cell for a future visit. Note that the example in Section 8 only yields one viewpoint per cell, as the selected points are placed relatively close to the center and therefore they capture much information. Notice that if the discretization is roughly the same, the camera will be *attracted* by regions of high number of polygons. Otherwise, we can set an importance value to the polygons we consider interesting (such as the ones belonging to statues). In the examples presented here we have not considered texturing, but this can be addressed using a region growing segmentation and posterior color coding of the regions to include the resulting texture in our measure as different polygons, as detailed in Vázquez *et al.* [15]. Viewpoint entropy has also been used for automatic interactive navigation in indoor scenes [17]. Unfortunately, the lacking of knowledge of the general structure of the model makes it difficult to ensure that the camera will pass through all the relevant cells. An entropy-based measure has also been presented and used to automatically place light sources [18].

## 7. Path construction

With the information collected from the previous two steps, we can build a minimal length path through the graph that visits all relevant cells. Our objective is not only to determine the path that covers all interesting cells but to determine at each moment which is the suitable camera position in order to see the highest amount of information of the scene.

### 7.1. *High-level* path

The first step on the path construction is to decide in which order the path will visit the relevant cells. We compute a high-level path $H$ over the cell-and-portal graph which is the shortest path traversing all the relevant cells from the initial point given by the user.

Given the set of relevant cells to visit and the initial cell, the problem of finding the shortest path traversing all the relevant cells is similar, but not equal, to the traveling salesman problem (TSP) which is an NP-complete problem [19]. We use a backtracking algorithm optimized by discarding partial solutions when they are longer than an already found solution. When the search is finished, we have a group of solutions that are minimal on its length (number of nodes traversed), and we choose the one with minimal node repetition. The cost of this algorithm is not enlarging the total cost of the approach much since the models usually don't have more than 50 cells. Nevertheless the cell merging process in phase 1 can be adjusted to limit the number of cells.

### 7.2. *Low-level* path

The *low-level* path can be computed just after the high-level path generation. Once we know which cells have to be visited and which is the ordering, we get an entrance and an exitance point for each cell. This, together with the best viewpoint (or viewpoints) of each cell allows us to build a smooth path. To build this path we perform two steps:

1. Path detection
2. Path approximation

As we do not know in advance which is the geometry of the cell and we only get a set of points inside it, the determination of a free-from-obstacles path must be done accurately. Given the set of points corresponding to voxels inside a cell, we build a graph where the nodes are the points and the edges the connectivity of the points to the neighbors. This can be carried out very fast. Then, we apply an A* algorithm [20] to search the best path from the entrance point to the best point of the cell, and we apply it again to reach the exit from the best point. For complex cells (i.e. the cells that generate more than one best points) this is applied iteratively until reaching the exit. In these cases, the iteration is also applied from the exit point to the entrance point, generating an alternative path which could be shorter than the previous one. If this *backwards* path is shorter, we will choose it reverting the point order.

The generated path is a polyline that is not necessarily smooth. To avoid sharp moves through the exploration process, we relax the path by using an interpolation based on Hermite curves [21]. We set a control point every two points of the path and build a smooth path that goes from the entrance to the exit. At each control point we use as tangent direction the vector joining the current point with the next path point. Note that it is better not to enforce $C^1$ continuity on the path at the best view point. The best point is supposed to show a high amount of interesting information, so the walkthrough will stop there and the camera will rotate to allow the user to see all the important information. In Figure 3 we can see an example of a path inside a cell. As the set of camera positions is very dense, we have only drawn one out of five camera positions. The yellow line indicates the orientation of the camera at these positions.

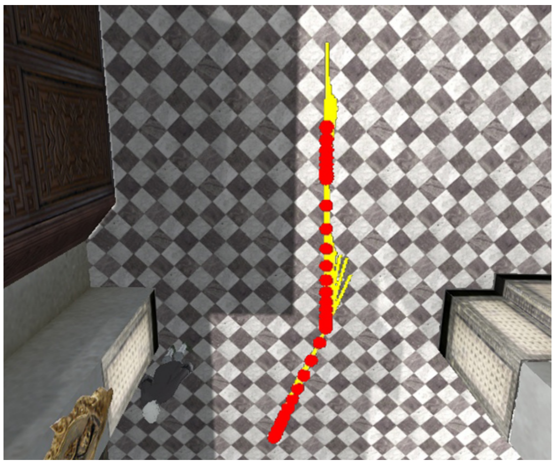

**Figure 3:** *An example path inside a cell. Camera positions are shown in red and the orientations appear as yellow lines.*

### 7.3. Complete Walkthrough

After the construction of the path, we want to determine which are the camera orientations that better show the scene during the navigation.

In a similar process than the one that determines the best point inside each cell, we place a camera at each point of the path and, for each point, we evaluate the amount of information that can be seen at different orientations and choose the orientations that will yield better results. This is computed almost interactively. We set some reasonable constrains to camera moves in order to build a smooth path.

- Limited rotation: The camera must be oriented forward, we do not allow rotations of more than 30-40 degrees from the walking direction in order to maintain a normal movement sensation during the walkthrough. If the camera were allowed to point backwards, the user could feel uncomfortable.
- Correct orientation at endpoint. People usually look forward when traversing a portal. We simulate this by limiting the rotation of the camera when it is reaching the exit point. When the camera is close to the way out, it smoothly starts turning back to the walking direction, and we ensure that it is at the correct orientation before crossing the portal.

For each cell the path is built in the following way. We place the camera at the entrance point of the cell and pointing towards inside the cell. Then, the best new camera orientation is computed by evaluating the possible new orientations (these measures are calculated on the back buffer), we allow only small rotations in order to make the movements smooth. For a given point and camera orientation, five different views are inspected, as depicted in Figure 4.

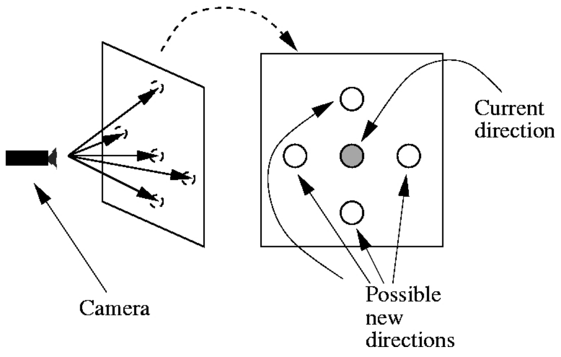

**Figure 4:** *The possible changes of orientation of the camera at each step*

To decide which of the orientations is the best, we take into account the amount of information visible from each camera configuration, as well as the history of the visited regions. This can lead to a problem when there is a very complex region in a cell, because the camera would be always pointing there. In order to avoid this, we keep track of the visited faces. For each view, when we analyze the amount of visible information we only take into account the faces that have not been visited yet. However, as the path will contain one or two hundred different positions at each cell, this could be a problem if we considered a polygon as visited if it had appeared in a single view, because it could cause the camera to change the orientation continuously. As we want the user to be able to see the environment properly, we have implemented a pseudo aging policy. We only consider a face visited when it has been seen at least in 20 different views, then it is marked and it will not be considered again. This strategy ensures that the regions of higher information will be visited and that the user will be able to stare at them calmly.

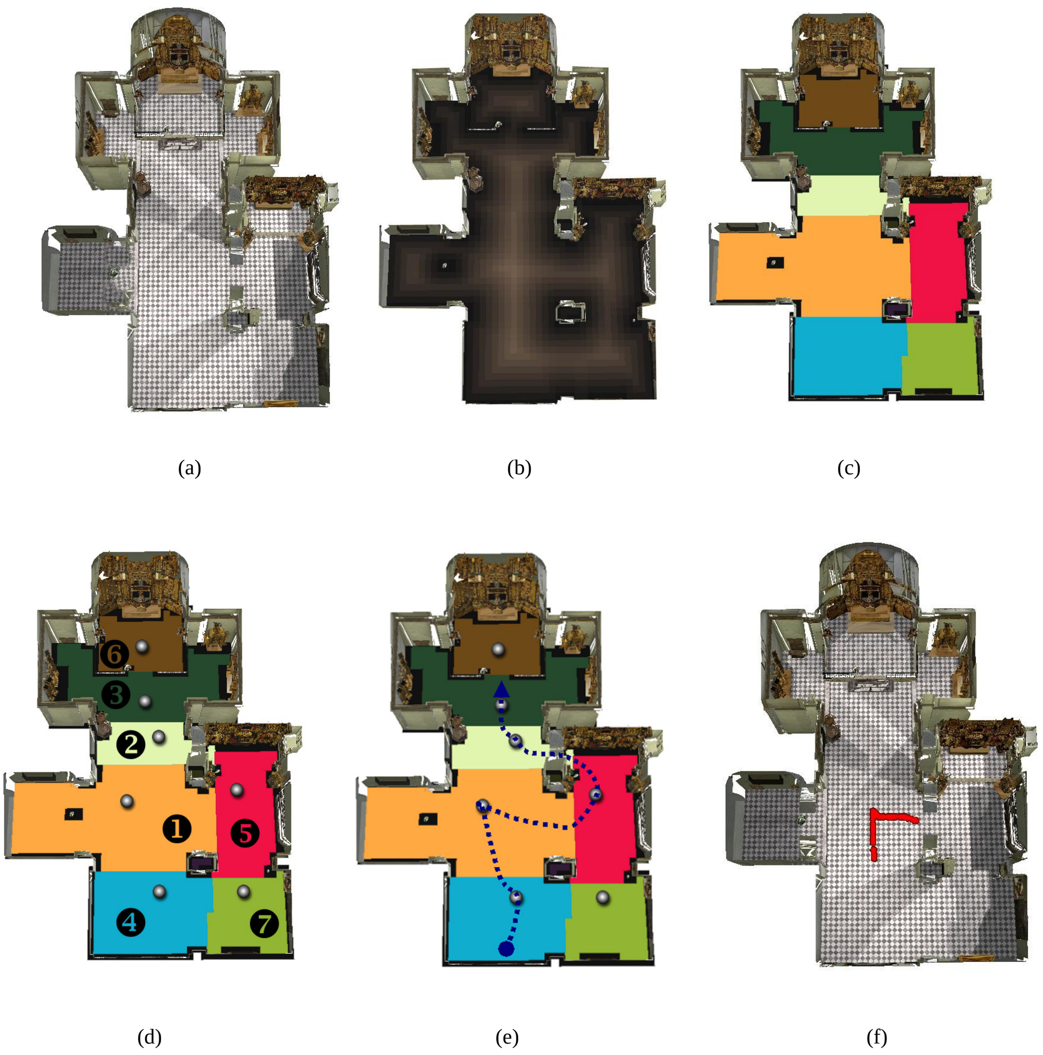

**Figure 5:** *Results in the church model. (a) Top view of the original model. (b) Computed distance field on a 128x128x128 grid. (c) Final cells detected after the merging process. (d) Ordered cells based on their entropy. (e) High level path through the 5 most interesting cells. (f) Computed low-level path for the most interesting cell.*

## 8. Results

We have presented a complete approach for automatically generating guided tours through complex walkthrough models. In contrast to other approaches, our method is completely automatic, the only input really required is an initial point.

Images of the whole process appear in Figure 5. In Figure 5-a we show the plan of the original model. The computation of the distance field map appears in Figure 5-b. After the distance field computation, the cell and portal generation detects a set of cells that are then refined through the merge process. The result of the merge for this example is shown in Figure 5-c. With this information we can proceed to compute the relevance of each cell. This generates an ordering between the cells that is depicted in Figure 5-d. After the cell evaluation, we choose a subset of cells with high entropy. In this case the threshold chosen selects cells 1 to 5, and the starting point of the path is at cell 4, the entrance of the church. Then, a high level path is calculated using the backtracking method explained in Section 7.1. The computed high level path is shown in Figure 5-e. For each cell, a low level path is computed from the entrance point to the best view of the model. As an example, we show the path corresponding to the most important cell in Figure 5-f. Note that the generated path has almost 200 camera positions, so we have only shown one out of 5 camera positions (with their corresponding orientations) for the sake of clarity. As we have commented, we do not force continuity on the best viewpoint, as at this point, the camera rotates to show the information all around that was not already seen during the previous walkthrough positions.

The total computation time was 10 minutes and 40 seconds on a 2GHz PIV with a GeForce Ti graphics card and 512Mb of memory with a model (the church model) of 63312 polygons. The bottleneck is the cell relevance evaluation process and the low-level path calculation because both require rendering the scene multiple times, which could benefit from the portal-and-cell graph if we used this for portal culling. More results can be found in *http://www.lsi.upc.es/˜virtual/EG2004.html*

In our current implementation the output of our algorithm can be used in several ways. The *full-auto* mode consists in the reproduction of the path by letting the camera follow the precomputed viewpoints and targets. The *guided-tour* variation would let the user look around during the navigation by allowing him or her to control the target but not the viewpoint. Finally, in the *free* mode the path nodes are rendered as arrows oriented along the direction of the next waypoint.

## 9. Conclusions and Future Work

We have presented a fully automatic system for the generation of walkthroughs inside closed environments that can be segmented using a cell-and-portal approach. The method can be useful as a way to automatically create visits of monuments or presentations of buildings, and can also be a good tool in the context of interactive systems as a first constrained path to help the interactive user navigate an environment.

As future work we want to introduce a hierarchical structure, concretely an octree representation to optimize the cell detection process. Moreover, we would like to further limit the effects of the furniture in the cell detection algorithm. An interesting issue would be to be able to compute cells in outdoor sparse scenes.

## Acknowledgements

This work has been partially funded by the Spanish Ministry of Science and Technology under grants TIC2001-2226 and TIC2001-2416.

## References

[1] John F. Canny. *The complexity of robot motion planning.* MIT Press, 1988. 2

[2] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991. 2

[3] Y. K. Hwang and N. Ahuja. Gross motion planning – a survey. *ACM Computing Surveys*, 24(3):219–291, September 1992. 2

[4] P. Isto. Path planning by multiheuristic search via subgoals. In *Proceedings of the 27th International Symposium on Industrial Robots*, pages 721–726, 1996. 2

[5] P. Isto. A two-level search algorithm for motion planning. In *Proc. IEEE International Conference on Robotics*, pages 2025–2031, Aut 1997. 2

[6] T. Y. Li, J. M. Lien, S. Y. Chiu, and T. H. Yu. Automatically generating virtual guided tours. In *Proc. of Computer Animation*, pages 99–106, 1999. 2

[7] T. Y. Li and H. K. Ting. An inteligent user interface with motion planning with 3d navigation. In *Proc. IEEE VR*, 2000. 2

[8] Brian Salomon, Maxim Garber, Ming C. Lin, and Dinesh Manocha. Interactive navigation in complex environments using path planning. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 41–50. ACM Press, 2003. 2

[9] Marcelo Kallmann, Amaury Aubel, Tolga Abaci, and Daniel Thalmann. Planning collision-free reaching motions for interactive object manipulation and grasping. *Computer Graphics Forum*, 22(3), 2003. 2

[10] Seth J. Teller and Carlo H. Séquin. Visibility preprocessing for interactive walkthroughs. *Computer Graphics*, 25(4):61–68, 1991. 2

[11] Lichan Hong, Shigeru Muraki, Arie Kaufman, Dirk Bartz, and Taosong He. Virtual voyage: Interactive navigation in the human colon. *Computer Graphics*, 31(Annual Conference Series):27–34, 1997. 2

[12] Denis Haumont, Olivier Debeir, and François Sillion. Volumetric cell-and-portal generation. *Computer Graphics Forum*, 22(3):303–312, September 2003. 2, 4, 5

[13] Carlos Andujar, Pere Brunet, and Dolors Ayala. Topology-reducing surface simplification using a discrete solid representation. *ACM Transactions on Graphics*, 21(2):88–105, 2002. 4

[14] M. Jones and R. Satherley. Using distance fields for object representation and rendering. In *Proc. of the 19th annual Conference of Eurographics (UK chapter), London*, pages 37–44, 2001. 4

[15] P.-P. Vázquez, M.Feixas, M.Sbert, and W.Heidrich. Automatic view selection using viewpoint entropy and its application to image-based modeling. *Computer Graphics Forum*, 22(4):689–700, Dec 2003. 5, 6

[16] E.C. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423,623–656, July-October 1948. 5

[17] P.-P. Vázquez and M. Sbert. Automatic indoor scene exploration. In *International Conference on Artificial Intelligence and Computer Graphics, 3IA*, Limoges, May 2003. 6

[18] S. Gumhold. Maximum entropy light source placement. In *Proceedings of the Visualization 2002 Conference*, pages 275–282. IEEE Computer Society Press, October 2002. 6

[19] K. Thulasiraman and M. N. S. Swamy. *Graphs: theory and algorithms*. John Wiley & Sons, Inc., 1992. 6

[20] Steve Rabin. *"AI Game Programming Wisdom"*. Charles River Media, March 2002. 6

[21] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics. Principles and Practice. Second Edition*. Addison-Wesley, 1990. 7