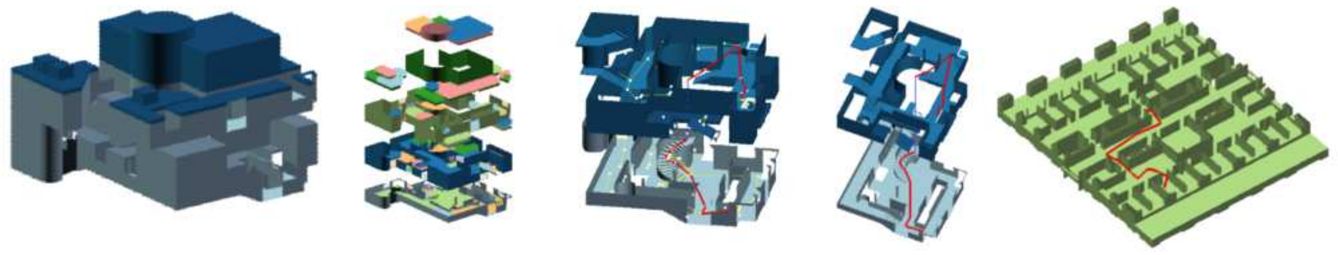
Automatically Generated Maps of 3D Environments

James F. Hamlin[*]
University of California, Berkeley



Abstract

A system for generating visualizations of routes in 3D environments is presented. From the input of a 3D triangle mesh, the approximate height of a navigating agent, and the number of floors, the system generates a segmentation of the environment into rooms and a roadmap for routing using a version of the watershed algorithm. A user may then interactively select source and destination rooms, and a shortest-path algorithm on the roadmap discovers the optimal route. Finally, the system selects viewing parameters by optimizing a score function of the view to produce a single static image of the path in an exploded view. The resulting visualization of the route displays simultaneously internal structural features of the building at each relevant floor while maintaining path visibility and clarity of important navigational cues such as changes in orientation.

Keywords: Visualization, Architecture, Mapping, Navigation Aids

1 Introduction

Computer-generated 2D maps have been popular web applications for years. The problem of automatically generating maps displaying paths between two locations easily extends to 3D environments, but little work has been done towards designing analogous systems. Just as 2D roadmaps can be invaluable to a human navigator when driving, 3D versions of such maps for architectural (or other 3D) environments may be useful for way-finding. The mock example from Tufte (Figure 1) demonstrates the possible output of such a system. Route maps of 3D environments could also be useful for other visualization tasks involving paths in 3D, just as routes on 2D maps generalize to tasks beyond mere way-finding, a paradigmatic example being Charles Joseph Minard's map of the 1812 march of Napoleon's army into Russia [Tufte 2001].

Developing a mostly-automated system for generating such maps with little user input and scarce semantic information beyond the raw geometry presents several interesting challenges: segmentation and analysis of the input, selection of a clear set of lines indicating the desired shortest path, selection and adjustment of the proper visualization parameters to produce a final static image. The system presented in this paper makes some headway towards an almost completely automated , with a framework for the generation of high-quality static images via optimization. Almost no semantic information is required beyond the environment geometry.

[*]e-mail: jfhamlin@berkeley.edu

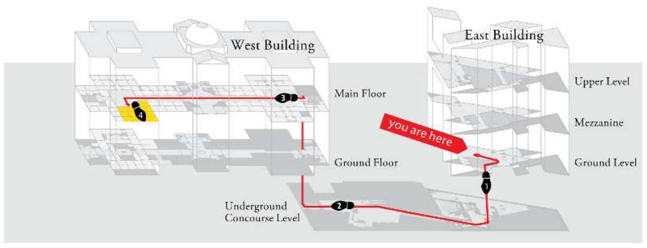


Figure 1: *Museum map kiosk example [Tufte 1997].*

2 Related Work

Recent work on navigation aids for multi-floor virtual environments [Chittaro et al. 2005] has extended third-person . Their system combines a technique for creating exploded views of architectural environments [Niederauer et al. 2003] with transparency to present local and global environment structure relevant to a navigation task in a virtual environment. The function of the third-person visualization is to provide the user with *survey knowledge*, which the authors describe as knowledge of relationships among locations in the environment, or a mental map. The authors identify four desirable properties for such navigation aids that bear on 3D environments in general:

1. Simultaneous visibility of both external and internal structure.
2. Minimization of occlusions due to multiple floors.
3. Support for third-person exploration to build up survey knowledge.
4. Easy interaction.

The work presented here demonstrates the first three properties, and while the primary interest has been to produce static images of routes in 3D environments, the methods used can easily be applied to the development of a system with the explicit goal of allowing a user to interactively develop survey knowledge.

The work presented bears similarity to a recently developed system for generating interactive maps of buildings from their AutoCAD descriptions. It uses an exploded view of floor plans to display routes in architectural environments [Sadhal]. A main drawback of this system is that it requires significant user input; the roadmap graph for each floor must be created by hand. Moreover, the resulting visualization can be overly schematic for some purposes, as the internal structure of the building can only be represented as 2D lines on an array of flat planes. Nonetheless, the current project shares a motivating example (the map in Figure 1), and takes several cues from this previous work.

3 Approach

I begin with a high-level overview of the approach used to generate route maps of 3D environments. Figure 2 shows the different stages of the system. The input includes the up vector, the height of a navigating agent, a description of the environment geometry in the form of a triangle mesh, and the number of floors in the structure. There are two subsystems that feed their output to the renderer to produce the final visualization, which will be described briefly now.

One subsystem segments the space of the environment into rooms to produce a roadmap graph, and it is inspired by a volumetric approach to cell-and-portal generation for visibility determination [Haumont et al. 2003]. The environment geometry, up vector, and agent height are used to construct a volumetric description of the environment. From this description, a topologically 2D set of voxels is extracted that spans the space accessible to a floor-bound navigator. A version of the 2D distance transform is applied, followed by the watershed transform, segmenting the space and geometry into rooms. The markers of the watershed transform and the interfaces between rooms (which form the *geodesic skeleton by zones of influence* (SKIZ)) become the nodes of the roadmap graph used for path-finding. The room geometry is used to allow a user to click on a rendered image to select source and destination rooms. The chosen rooms are then used along with the roadmap graph to find the shortest route between them by Dijkstra's algorithm [Dijkstra 1959]. This route is then provided as input to the visualization renderer.

The other subsystem finds the heights at which to split the environment to produce the exploded view used in the final visualization. In addition to the input to the previously described subsystem, the number of floors must also be provided. Given this, a technique for generating exploded views of architectural environments is employed, yielding the heights used by the renderer in displaying the exploded view.

Once we have a segmentation of the environment into floors and a 3D route, the final task is view optimization. The parameter space of view direction and separation distance of the floors is searched to minimize a score function to avoid occlusions of the route and improve the clarity of the final visualization.

3.1 Room Segmentation

The segmentation of the model into rooms serves two purposes: associating geometry with rooms, and extracting a roadmap. The approach is inspired by the technique of automatically generating cell and portal graphs for visibility determination in [Haumont et al. 2003].

3.1.1 Voxelization

Room segmentation begins by constructing a voxel representation of the environment. A uniform 3D voxel grid is layed across the bounding box of the input geometry, and each voxel is marked as solid if it is intersected. My system uses a naive algorithm, testing each voxel explicitly for intersection, but performance and quality might be improved by employing techniques from the 3D rasterization literature [Kaufman et al. 1993]. Nonetheless, both performance and quality of the naive voxelization method have proved acceptable for this application.

Since we must ensure that this discretized representation maintains all portals through which a navigating agent could pass with at least one hollow voxel, the voxel size is set to a value proportional to an input value *agent height*, set interactively by the user through the display of an indicator with alternating black and white segments (Figure 3) with lengths equal to the current *agent height*. The indicator is rendered atop the environment geometry in an orthographic view with an ArcBall interface [Shoemake 1992]. A user may then rotate the view as desired to compare the height of a segment to features of the environment. As this height need only be approximate, this input method and feedback representation suffice. In the implementation, the voxels are cubes with edges of length (0.2 * *agent height*), ensuring that any portals greater than $0.2 * \sqrt{2} * (agentheight)$ across appear as hollow cells in the voxel grid. Such coefficients are used in several other places in the system, and though they might be exposed to the user as further input values, in these cases reasonable values have been supplied.

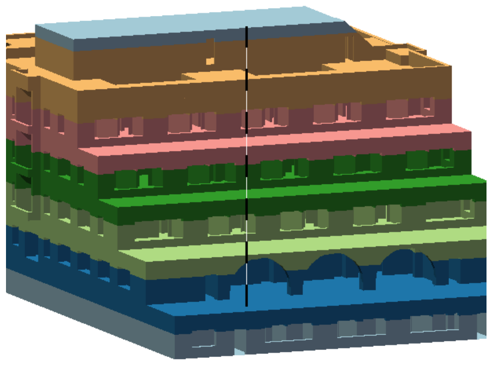


Figure 3: *A vertical line with alternating white and black segments shows the current setting of the* agent height *input variable relative to the environment.*

3.1.2 Search Space

The segmentation process proceeds by identifying a set of 'walkable' voxels, that is, for our purposes, a set of voxels that could be occupied by the center of an agent of height *agent height*. This condition is approximated by two requirements, called *support* and *clearance*. Support requires that such a voxel be hollow, have exactly one hollow voxel immediately beneath it, and a solid voxel immediately beneath this hollow one. Clearance requires that the voxel have a hollow voxel immediately above it. Beginning at the highest plane of voxels, the algorithm iterates until a voxel is found that satisfies both the support and clearance criteria and has not yet been designated walkable. Every time such a voxel is encountered, a flooding algorithm marks further walkable voxels using a lateral 4-connectivity. The flooding continues to neighbors that satisfy the above walkability criteria.

To account for negotiable changes in floor height, if a neighbor voxel satisfies the clearance criterion but not the support criterion, the voxel immediately beneath this neighbor is then checked for support. If this cell satisfies the support criterion, then the neighbor and this cell are marked as walkable, and flooding continues through the lower cell. In this way, the flooding may cascade down stairs and inclines. Figure 4 demonstrates the flooding algorithm.

An additional test for support is to test the surface normal of the

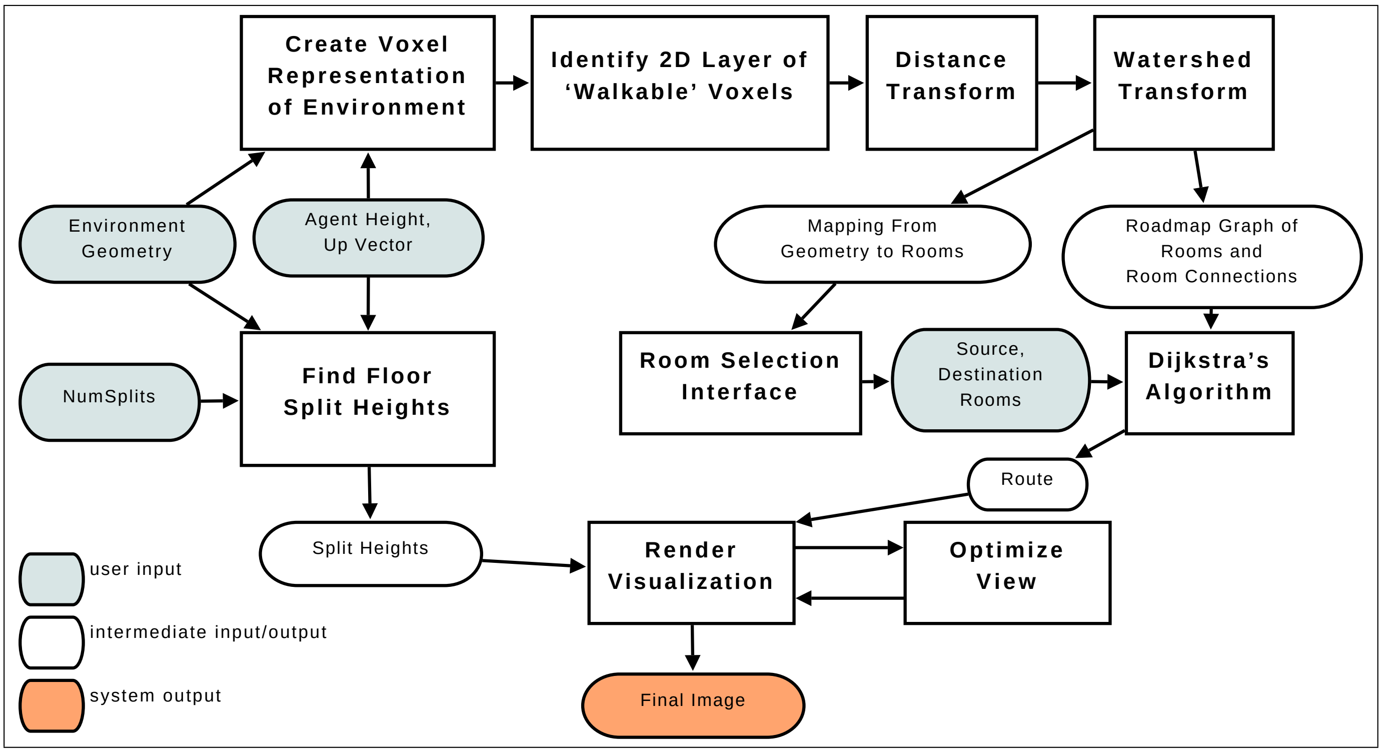


Figure 2: *An overview of the presented system. Boxes represent operations, and capsules represent important input and output data. Arrows denote the flow of data between operations.*

surface in the lower solid voxel, failing if the normal deviates by some threshold amount from the up vector. Since most real-world architectural environments do not have impassably steep surfaces adjacent to walkways without some barrier, this has been left unimplemented.

The voxels in the output set of walkable voxels may be understood as representing one or more two-manifolds lying just above the upward-facing surfaces of the environment. It is in the space spanned by these manifolds that we perform room segmentation. At this point the volumetric representation might be discarded and a 2D discrete element mesh extracted from the walkable voxels used for all further processing. But since room geometry is desired in addition to mere segmentation, the volumetric representation is kept.

3.1.3 Distance Transform

As in the volumetric approach to cell and portal generation [2003], the next stage of room segmentation is the application of the distance transform, which computes for each voxel its minimum Manhattan distance to the environment geometry. While room segmentation for the visibility determination problem requires that one consider the full 3D space, the space of an environment relevant to navigation by a human-like navigator is merely the upward-oriented 2D surfaces, which are homeomorphic to disks with zero or more holes. Thus, the 2D distance transform is applied to the discrete voxel representation of the walkable two-manifolds extracted by the previous step.

The distance transform is an efficient algorithm for computing a distance field over a discrete, n-dimensional domain [Rosenfeld and Pfaltz 1966]. The result is a distance map, where for each voxel we have the Manhattan distance to the nearest object - in this case, the environment geometry.

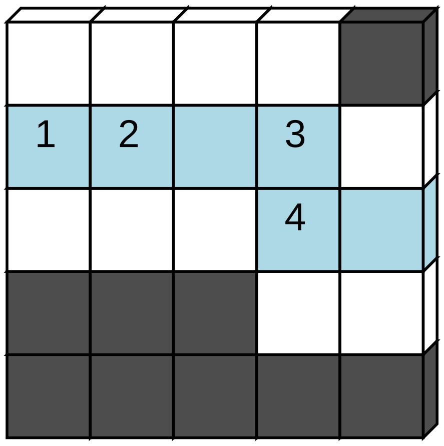


Figure 4: *A vertical 2D slice of the volumetric representation of the environment. Shaded cells represent solid voxels, white cells represent hollow voxels, and blue cells represent walkable voxels. Flooding begins at the cell marked '1.' The flooding continues to cell 2, since they are adjacent laterally and cell 2 satisfies both of the support and clearance criteria. Cell 3 fails the support criterion, but since the cell beneath it, 4, satisfies it, cell 3 is designated as walkable and flooding continues at cell 4.*

3.1.4 Watershed Transform

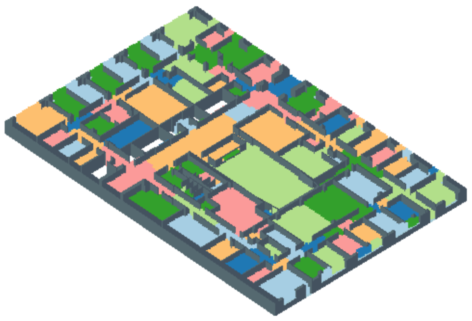


Figure 5: *A floor of the Soda Hall model with rooms 'flooded' by the watershed algorithm. The regional maximum of the distance field in each room and the points at which the medial axis meets the interface of pairs of catchment basins become the nodes of the roadmap graph.*

Once the distance map has been computed, the watershed transform is applied. This proceeds almost exactly as in [2003], with the following changes:

- Catchment basins are restricted to the voxels designated walkable.
- Instead of performing portal placement when two catchment basins meet, a roadmap node connecting adjacent rooms is placed as described below.

3.1.5 Roadmap Extraction

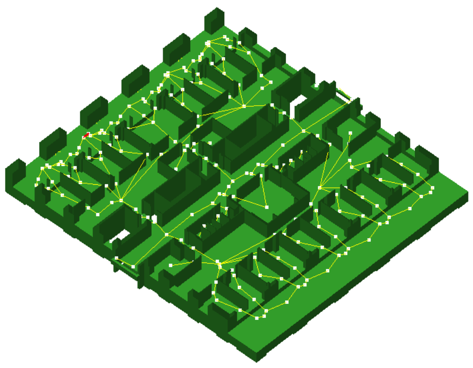


Figure 6: *The roadmap graph of a floor of the Soda Hall model. White squares denote nodes of the graph, yellow lines denote edges.*

The roadmap is extracted as the watershed transform proceeds. Room nodes are placed at the location of a single voxel in each regional minimum of the negated distance map. A node is also placed when two catchment basins (representing rooms) of the watershed algorithm merge. These nodes are then connected to the room nodes of each of the meeting catchment basins.

3.2 Rendering

In this section, the design choices for the final rendering are described. Given the view direction, floor separation distance, and the route, a bounding box is constructed around the floors through which the route passes. The projection matrix is set to fit this bounding box tightly. Each floor is rendered by setting the OpenGL clipping planes, adding a vertical translation to the modelview matrix, and rendering the environment geometry. Finally, the segments of the route that pass through the floor are rendered.

3.2.1 Axonometric Projection

To eliminate perspective distortions, an axonometric projection is used [Niederauer et al. 2003].

3.2.2 Exploded View

The primary visualization tool used to produce output images is the exploded view [2003]. Exploded views are a means of simultaneously conveying both internal and external structure. In visualizing a path through a 3D environment, the virtues of the exploded view are exploited in several ways. First, since a single path may pass through any number of floors, the exploded view allows the entire path to be visible from one view. Additionally, floors that are irrelevant to the path may be ommitted from the view.

3.2.3 Line Styles

The route is rendered as a solid red line. Where the route is occluded, the line width is slighly thinner and a dashed style is used. To maintain route continuity between floors, vertical lines, drawn thinner than the route and in blue, connect the route where it meets the split planes.

3.2.4 Surface Properties

Each floor of the environment is rendered in a different color. Colors were chosen from the ColorBrewer utility [Brewer and Harrower]. A simple Lambertian shader is used. A directional light points downward to illuminate the floors, and two other lights provide fill. The result is that wallls appear significatnly darker than the floors, with particular shading depending on orientation.

3.3 View Optimization

The final task is to set the view parameters to best depict the route. The view parameters include view direction (azimuth, elevation), and the floor separation distance. The goal is to minimize route occlusion, self-intersections, visualization area, and to maximize the visibility of changes in route orientation.

Simulated annealing is an algorithm for non-linear optimization that has been successfully applied to several visualization problems including map layout, label placement, and route simplification [Agrawala and Stolte 2001]. It is used here to set the view parameters. Simulated annealing requires two problem-specific functions: a neighbor function for generating new states from old ones, and a score function that gives a score to a given state.

3.3.1 Neighbor Function

The neighbor function generates a new parameter vector by randomly perturbing the current parameter values. First, one of the parameters is chosen at random with uniform probability. An offset o is then generated from a geometric distribution. The parameter space is discretized, so the current value of the selected parameter is randomly either incremented or decremented by o discrete steps.

3.3.2 Score Function

Here the score function used for view optimization is described in detail. The score function $S(p)$ of the parameter vector p is a weighted sum of four terms

$$S(p) = w_0 * O(p) + w_1 * I(p) + w_2 * A(p) + w_3 * p_s \qquad (1)$$

where O, I, and A are functions of the parameters, and p_s is simply the separation distance component of the parameter vector.

- **Path Occlusion**

  The function $O(p)$ gives the ratio of occluded route pixels to the total number of route pixels. If no route pixels are drawn at all, the function returns 1.

$$O(p) = \begin{cases} 1 & \text{if } RoutePixels(p) = 0 \\ \frac{OccludedRoutePixels(p)}{RoutePixels(p)} & \text{otherwise} \end{cases}$$

  The system calculates $RoutePixels(p)$ and $OccludedRoutePixels(p)$ using the OpenGL occlusion query extension, which reports the number of fragments drawn by a set of primitives. First, two OpenGL occlusion query objects are generated. The framebuffer is cleared, the modelview and projection matrices set according to the parameters, and the first query is begun. In this query, only the path is rendered. The query is then ended. Next, the environment geometry is rendered, preparing the depth buffer for the next query. The depth function, which specifies what comparison test must be passed to allow a fragment to be written, is set to `GL_GEQUAL`. Thus, only occluded fragments will be drawn. The second query is begun, the path drawn a second time, and the query ended.

  As a potential optimization, these queries are initiated at the start of the evaluation of the score function, but the results are not requested from OpenGL until all other components of the score function have been calculated. After the queries are made, `glFlush()` is called to force the execution of any buffered commands. `glFlush()` does not block, but tells the driver to execute buffered commands in finite time, allowing for as much CPU/GPU concurrency as possible between issuing the queries and requesting their results.

- **Path Self-Intersection**

  The function $I(p)$ returns the number of intersections that occur in the projection of the route segments in image space. The view and projection matrices for the parameter-prescribed view are constructed and used to map the segments into image space. Intersections are counted by an efficient sweep line algorithm [Bentley and Ottmann 1979].

- **Angle Deviation**

  The function $A(p)$ returns the average squared difference between the angles adjacent route segments make in 3D and the angles they make in their 2D projections in image space:

$$A(p) = \frac{1}{n-2}\sum_{i=1}^{n-1}[\arccos(\frac{pv_{i-1} - pv_i}{\|pv_{i-1} - pv_i\|} \cdot \frac{pv_{i+1} - pv_i}{\|pv_{i+1} - pv_i\|}) - \arccos(\frac{v_{i-1} - v_i}{\|v_{i-1} - v_i\|} \cdot \frac{v_{i+1} - v_i}{\|v_{i+1} - v_i\|})]^2$$

  where the v_i are the vertices of the route segments and the pv_i are the v_i projected into image space. This measure could be improved by weighting the error terms by the lengths of the adjoining segments.

- **Separation Distance**

  The floor separation distance is minimized as a proxy for visualization area.

4 Results and Discussion

Figure 7 shows two maps generated by the system.

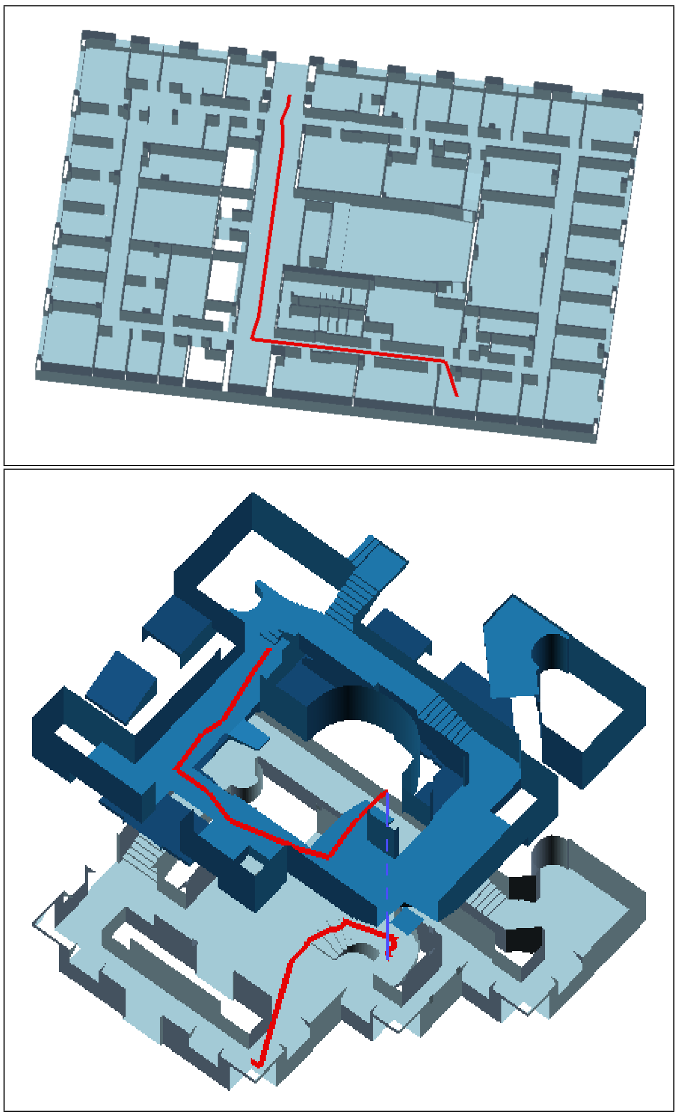


Figure 7: *Maps of routes rendered by the system. A route on a floor of Soda Hall (top). A route between two floors of a Quake 3 map (bottom).*

The approach is largely successful - rooms are identified well, especially in traditional architectural environments such as Soda Hall. The optimization scheme almost always eliminates any occlusions of the route, and keeps large turns visible.

The main flaw in this approach is the direct use of the roadmap graph extracted during the watershed algorithm for the rendered route. Figure 8 shows how the edges of the graph may intersect environment geometry. Future work will address this issue with a more in-depth look at route-generation.

A second issue is color choice. While floors are distinguishable, the color of the vertical connector between routes on different floors can easily be confused with the floor surfaces. Additionally, parts of floors distant from the route clutter the view and contribute to confusions about depth and height.

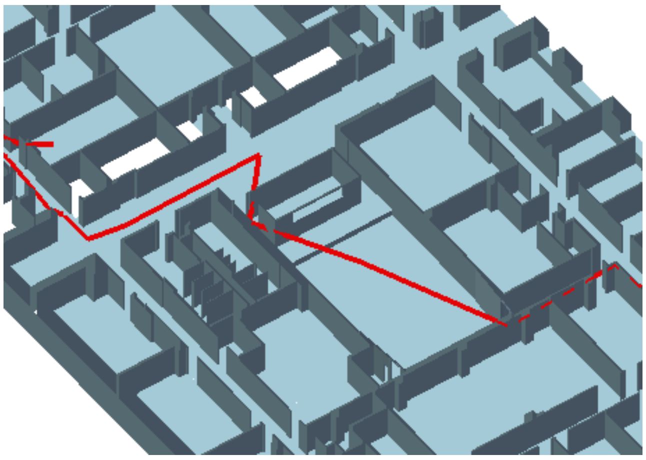


Figure 8: *Because the roadmap graph is not guaranteed to avoid obstructions, intersections with the environment geometry can occur when it is used directly for route rendering.*

5 Future Work

Given the breadth of the system, there is much room for enhancement. Of high importance is further improvement of the generated roadmaps, which have no guaranteed quality. While the currently produced graph accurately represents room connectivity, its edges do not clearly express navigational cues. Worse, they may intersect environmnet geometry, since there is only one node per room. Further research into this problem will begin by formalizing the roadmap problem for 3D environments.

Research into rendering styles that both improve performance of the cognitive tasks associated with reading routes is required. Currently, the segmentation of geometry into rooms is only used to allow picking in the user interface. It may also be used to separate not only floors in the exploded view, but rooms as well. Different rendering styles might be used for rooms depending on their relation to the route or other properties. Finally, additional view parameters might be exposed to the optimization scheme and the energy function refined.

Finally, integrating minimal semantic information could greatly improve the results of the system. Future work will include using floor maps to match semantic objects with locations in the 3D model to improve room segmentation and add automatic support for elevators.

References

AGRAWALA, M., AND STOLTE, C. 2001. Rendering effective route maps: improving usability through generalization. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 241–249.

BENTLEY, J. L., AND OTTMANN, T. A. 1979. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput. 28*, 9, 643–647.

BREWER, C., AND HARROWER, M. ColorBrewer.

CHITTARO, L., GATLA, V. K., AND VENKATARAMAN, S. 2005. The interactive 3d breakaway map: A navigation and examination aid for multi-floor 3d worlds. In *CW '05: Proceedings of the 2005 International Conference on Cyberworlds*, IEEE Computer Society, Washington, DC, USA, 59–66.

DIJKSTRA, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 1 (December), 269–271.

HAUMONT, D., DEBEIR, O., AND SILLION, F. 2003. Volumetric cell-and-portal generation. In *Computer Graphics Forum*, Blackwell Publishers, vol. 3-22 of *EUROGRAPHICS Conference Proceedings*.

KAUFMAN, A., COHEN, D., AND YAGEL, R. 1993. Volume graphics. *Computer 26*, 7, 51–64.

NIEDERAUER, C., HOUSTON, M., AGRAWALA, M., AND HUMPHREYS, G. 2003. Non-invasive interactive visualization of dynamic architectural environments. In *I3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 55–58.

ROSENFELD, A., AND PFALTZ, J. L. 1966. Sequential operations in digital picture processing. *J. ACM 13*, 4, 471–494.

SADHAL, N. Master's thesis.

SHOEMAKE, K. 1992. Arcball: a user interface for specifying three-dimensional orientation using a mouse. In *Proceedings of the conference on Graphics interface '92*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 151–156.

TUFTE, E. R. 1997. *Visual Explanations: Images and Quantities, Evidence and Narrative*. Graphics Press, February.

TUFTE, E. R. 2001. *The Visual Display of Quantitative Information*. Graphics Press, May.