Interactive Navigation in Complex Environments Using Path Planning

Brian Salomon Maxim Garber Ming C. Lin Dinesh Manocha
Department of Computer Science
University of North Carolina at Chapel Hill
{salomon, garber,lin,dm}@cs.unc.edu
http://www.cs.unc.edu/gamma/Navigation

Abstract

We present a novel approach for interactive navigation in complex 3D synthetic environments using path planning. Our algorithm precomputes a global roadmap of the environment by using a variant of randomized motion planning algorithm along with a reachability-based analysis. At runtime, our algorithm performs graph searching and automatically computes a collision-free and constrained path between two user specified locations. It also enables local user-steered exploration, subject to motion constraints and integrates these capabilities in the control loop of 3D interaction. Our algorithm only requires the scene geometry, avatar orientation, and parameters relating the avatar size to the model size. The performance of the preprocessing algorithm grows as a linear function of the model size. We demonstrate its performance on two large environments: a power plant and a factory room.

CR Categories and Subject Descriptors: I.3.6 [**Computer Graphics**]: Methodology and Techniques - Interaction techniques; 1.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism and framebuffer operations

Keywords: navigation, motion planning, collision detection, interaction, large models

1 Introduction

Virtual environment technology has been increasingly used for design and evaluation of man-made structures and complex engineering products. It has the potential to be used as an aid to designers of complex spatial arrangements, such as the interiors of submarines, cargo ships, power plants, oil platforms, airplanes, etc. The multi-disciplinary design review of such structures benefits greatly from user-steered interactive walkthroughs. This requires the ability to render the model at interactive rates, and also to easily navigate the environment using appropriate interaction modes.

It is common for a model of a large structure to be composed of thousands of objects, and to be composed of millions of primitives. This complexity makes it difficult to both render such a model at interactive rates on current graphics systems, and to interact with it in an intuitive manner. Typically, a user navigates through such a model using a driving mode, where the camera position is continuously changing based on mouse, joybox, or arrow-key inputs. Often, this input is used to control the motion of an unconstrained camera that can be positioned and oriented in the environment in an arbitrary manner. Such interaction methods have several limitations. Because the environment may be cluttered with many objects, tight spaces, corners, and stairways, a free flying camera can easily penetrate portions of the model and provide the user with confusing views of the geometry. An unconstrained camera also fails to provide the user with a natural understanding of how a human would move around objects, or navigate between different locations in the environment. Thus, the combination of model complexity, limited visibility, and restricted interfaces makes it difficult for a user to interact with a complex model in an intuitive manner.

In this paper, we address issues related to constrained camera motion and automatic path computation to assist 3D navigation of complex synthetic environments. The presented approach plans collision-free paths for an avatar constrained to the walkable surfaces of the environment. Most of the earlier work in this area has either been limited to local navigation modes based on driving, flying, or real-walking, or has dealt with relatively small environments. Many algorithms based on force-field methods have been proposed for local navigation. No automatic techniques are known for path computation in complex environments. Our approach combines results from robot motion planning and driving interaction methods to facilitate planned, constrained motion in complex environments for an avatar.

1.1 Main Results

We present a novel approach for interactive navigation in large environments using path planning. Our algorithm automatically computes a collision-free and constrained path from one part of the environment to another, and also enables local user-steered exploration. It uses a variant of the probabilistic roadmap planning algorithm, along with a reachability-based analysis, to precompute a navigation roadmap of the environment. The algorithm requires only a few parameters and automatically generates the global roadmap by performing a sequence of sampling, local planning and reachability computations.

The runtime algorithm searches the roadmap graph for a path between user specified start and goal positions. It also performs collision detection, using bounding volume hierarchies, and path smoothing to interactively guide the avatar along this path through the environment. The runtime algorithm allows the user to interactively steer the avatar through the model in a driving mode, while imposing natural constraints on the avatar's movement to ensure walk-like motion, across the floor, stairs, or other walkable surfaces in the model. We integrate these capabilities and constraints in the control loop of a 3D interactive application.

The overall path computation algorithm offers a number of benefits for navigating in large environments. The preprocessing and runtime steps are completely automatic. This is an important feature for handling very large synthetic environments. Moreover, neither the path computation nor local navigation expects any explicit walkable surface specification from the model. This information is inferred from the avatar dimensions and orientation with respect to the scene. Finally, the complexity of the algorithm grows at most linearly with the model size.

We have used our approach to navigate through two complex environments. One is a cluttered factory room model composed of over $10,000$ triangles. The second is a complete power plant model composed of more than 12 million triangles and 1200 objects. We used our path planning algorithm to compute collision-free paths between different locations in these environments, as well as to navigate through them in a driving mode. The resulting system can also be applied to many domains that involve exploring, or training in unfamiliar synthetic environments.

1.2 Organization

The rest of the paper is organized in the following manner. We give a brief survey of previous work in navigation and motion planning in Section 2. Section 3 gives an overview of our approach, while Section 4 presents the roadmap computation algorithm. We describe the path computation and navigation algorithms in Section 5. In Section 6 we discuss the implementation and performance of our system on the example environments. Section 7 discusses some of the limitations of our approach and presents future research directions.

2 Related Work

In this section, we give a brief overview of related work in navigation of virtual environments and path planning.

2.1 Navigation in Virtual Environments

There is a vast body of work related to 3D navigation in virtual environments. It can be classified into two main categories: work on understanding the cognitive principles behind navigation, and work on developing navigation techniques for specific tasks and applications. A task-based taxonomy of different navigation techniques is presented in [Tan et al. 2001]. Darken and Sibert [Darken and Sibert 1996] have explored cognitive and design principles as they are applied to large virtual environments. There has also been considerable work in designing intelligent user interfaces for improved navigation [Shneiderman and Maes 1997; Li and Ting 2000].

Most of the prior work on navigation for walkthrough applications has focused on developing body-centered interaction methods, including devices such as treadmills, or on evaluating the differences between various interaction techniques, such as walking and joystick based flying, [Mackinlay et al. 1990; Robinett and Holloway 1992; Slater et al. 1994; Usoh et al. 1999]. There has been less work on automatic computation of navigation paths in complex environments. Slater et al. [Slater et al. 1994] have presented an interaction technique based on body gestures for walking and ascending, or descending, steps and ladders in virtual environments. These techniques were applied to architectural walkthrough environments. Igarashi et al. [Igarashi et al. 1998] have presented a simple interaction technique for walkthroughs in which the user draws the intended path directly on the scene, and the avatar automatically moves along the path. Many of the current computer games also offer effective means of navigation. However, it is not clear that these approaches can be extended to navigation and automatic path computation in general massive environments.

For handling very large environments, Wilson et al. [Wilson et al. 1999] have presented fast algorithms for collision detection and distance computation between the avatar, or other moving objects, and the rest of the environment. Hubbold et al. [Hubbold and Keates 2000] have presented techniques based on force-field methods for local navigation as well as collision detection algorithms for parameterically-defined models. The resulting system works well in terms of computing collision-free paths in the localized neighborhood of the avatar, but cannot be used to compute a global path between arbitrary initial and final positions in a complex environment.

Some commercial products like *Walkinside* [Walkinside 2002] offer support for viewing detailed information related to all the elements in industrial and architectural environments.

2.2 Motion Planning and Path Computation

Motion planning has been extensively studied in robotics, computational geometry and related areas for more than three decades. However, it is still considered to be a difficult problem to solve in its most basic form, e.g., to find a collision-free path for a rigid object among static obstacles. The best known complete algorithm for computing a collision-free path has complexity exponential in the number of degrees of freedom (dof) of the robot or the moving object [Canny 1988]. Such planners, also called *criticality-based* planners, rely on an explicit, global geometric analysis to generate a provably complete representation of the configuration space for the moving object so that it can be effectively searched for a path. In practice, these planners are challenging to implement and slow during execution, especially in massive environments, which contain a very large number of primitives.

The theoretical and practical complexity of complete planners has motivated the development of planners that rely on approximate or heuristic methods [Latombe 1991]. Such planners are relatively simple to implement. However, they are not always guaranteed to find a path, if one exists. Examples of such planners include those based on potential field, or force field, methods [Khatib 1986]. These are relatively fast in practice and work well for local path planning. Potential field method have also been used for motion planning, movement control in animation [Egbert and Winkler 1996], camera viewpoint control for virtual colonoscopy [Hong et al. 1997], navigation in large and complex geometric environment [Hubbold and Keates 2000] and real-time motion planning in complex environments [Hoff et al. 2000]. The resulting planning, or navigation, algorithms have the limitation that they can become trapped in local minima of the potential function and so may not work well in dense environments composed of a large number of objects, especially when a long indirect path is required to move the avatar to a faraway goal location.

Some of the practical algorithms for global motion planning are based on taking random samples in the robot's configuration space [Kavraki and Latombe 1994; Overmars and Švestka 1995]. These approaches build a *probabilistic roadmap* (PRM) in the configuration space of the robot. For such systems a key concern is how to generate sufficient samples of the environment for the roadmap to capture the topology and connectivity of the configuration space. Their application to large and complex environments has been limited. A particularly relevant recent roadmap planning approach is the Visibility Based PRM system (VisPRM) described in [Simeon et al. 2000]. The VisPRM algorithm provides a method for rejecting many of the randomly generated configurations used to build the PRM roadmap, to produce a roadmap with far fewer nodes that retains the same global path planning information.

Motion planning algorithms have also been used for designing better user interfaces and providing better navigation techniques for animated characters. Li et al. [Li et al. 1999; Li and Ting 2000] have presented algorithms, based on randomized motion planning, that take mouse input from the user and predict the location in the environment that the user would like the avatar to move to. Next they compute a collision-free path to the predicted goal position. However, the running time of a randomized planner can vary considerably and this system has only been used for local navigation in relatively simple environments. Kuffner [Kuffner 1998] has presented a navigation algorithm for computing collision-free motion for animated human characters using graphics hardware. It uses 2D motion planning navigate characters through environments of moderate complexity at interactive rates.

Nieuwenhuisen and Overmars [2003] present a method for navigating through a model based on the PRM algorithm and cinematography principles. Their approach maintains desired aesthetic qualities over a path through the environment by controlling the camera roll, distance to obstacles, smoothness of turns, and continuity of the path traversal speed.

3 Overview

In this section, we give an overview of our approach and introduce different modes of navigation.

3.1 Assumptions

Our approach is designed to allow a user to navigate in any large environment. Aside from model geometry, the only necessary input to the preprocess are the dimensions of the avatar and its natural orientation in the scene. The environment is specified in terms of geometric primitives and their connectivity.

We assume that the user is steering through the environment in a driving mode. The avatar can move along three degrees of freedom, including translation along the plane corresponding to a floor or rotating about an axis orthogonal to that plane. Moreover, the motion of the avatar is constrained to lie on a surface such as a floor or stairway. We allow the avatar to navigate in the environment in two modes:

- **Global:** In this case, the user specifies an initial and final position in the environment. The algorithm automatically computes a collision-free path that satisfies the motion constraints.

- **Local:** The user is allowed to explore the environment in a driving mode, where the avatar responds to translational and rotational inputs from the user. The algorithm automatically performs collision checks with the environment and constrains the avatar to walkable surfaces in the environment.

The module controlling the local movement is used during roadmap preprocessing, local navigation, and path following as shown in Figure 1 and Figure 2.

3.2 Preprocessing

During the pre-processing stage, we compute a global roadmap of the synthetic environment (details given in Section 4). We achieve this by randomly generating collision-free configurations of the avatar in the scene. Samples are connected using a local planner that enforces our constraints including collision detection using a bounding box hierarchy [Larsen et al. 1999]. We also prune the graph to reduce the redundancy of coverage or reachability over the entire walkable space. The resulting roadmap is a global data structure that is used at runtime to navigate the user through the environment.

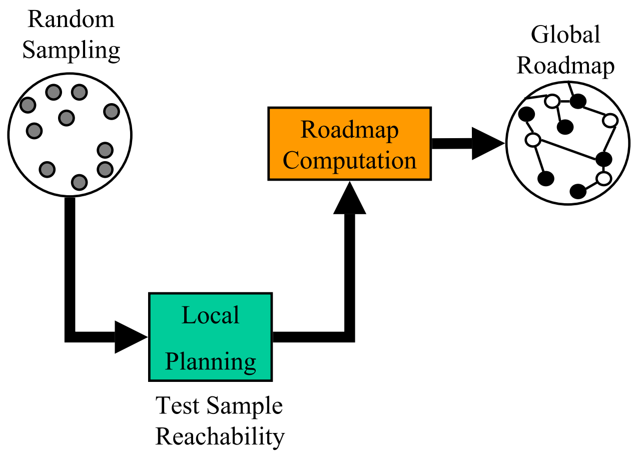

Figure 1: *Preprocessing phase*

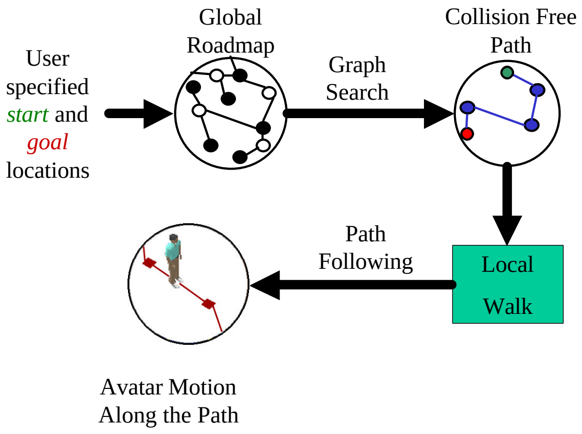

Figure 2: *Runtime algorithm*

3.3 Runtime Algorithm

At runtime, the user specifies the initial and goal position. The algorithm searches the roadmap graph using a variation of the IDA^* search algorithm [Korf 1985] and performs connected component analysis to improve its performance.

The user is navigated along the computed path. Given the collision-free and constrained path, a few viewing options including path alignment are available to the user. More detail is given in Section 5. The combination of these techniques enables the user to automatically navigate through the complex environment.

The algorithm also allows the user to navigate any part of the environment in a driving mode. The user can also stop at any point of interest and control his movement to inspect a particular location within the environment. Our algorithm provides the user with both local control and global navigation in a large environment.

4 Global Roadmap Precomputation

In this section we describe the pre-processing phase of the path planning algorithm. In this phase the system extracts geometric information from the virtual environment, which is later used to aid the user in interacting with the environment. In particular, it automatically computes a graph, or roadmap, that captures the connectivity of the portions of the model that are accessible to the user's avatar. This roadmap is later used to generate walkable paths that let the user navigate between arbitrary locations in the environment.

During this preprocess phase we randomly generate valid configurations for the avatar in the virtual environment. They are linked to form a graph by planning simple paths between them by using a local planner. During the query phase, when a user is interacting with environment, the resulting roadmap is quickly searched for a path that can take the avatar between the specified start and goal configurations.

Next we define some key roadmap planning concepts, and later apply them to our problem.

4.1 Configuration Space

We use the motion planning *configuration space* formulation [Lozano-Pérez and Wesley 1979]. In this formulation, the avatar is represented as a point in a higher dimensional space, called the configuration space, $\mathcal{C}$. In this manner any planning task can be interpreted in terms of a point robot, so that the problem of characterizing the constraints on the avatar's motion is isolated from that of finding a path that satisfies those constraints. In our application, the user's avatar is constrained to the walkable surfaces. The configuration space is decomposed into two sets, $\mathcal{C}_{free}$ and $\mathcal{C}_{blocked}$. $\mathcal{C}_{free}$, often called the free space or the workspace, is the set of all configurations in $\mathcal{C}$, for which the avatar is not in collision with any obstacle in the environment. $\mathcal{C}_{blocked}$ is the set difference $\mathcal{C}\backslash\mathcal{C}_{free}$, or the set of all configurations in which the avatar is in collision with at least one obstacle.

4.2 Local Planning Method

Given two configurations $\mathbf{c}_1$ and $\mathbf{c}_2$ in $\mathcal{C}_{free}$, the local method, $\mathcal{L}$, computes a path $\mathcal{L}(\mathbf{c}_1, \mathbf{c}_2)$ connecting the two configurations, which does not collide with any obstacles. In roadmap planning approaches the local method that is used is simple, quick to evaluate, and easy to reproduce during the query phase, even if it does not always succeed in finding a path, if one exists. The local planner should be efficient because it is invoked many times during the construction of the roadmap. A very common approach is to use a local planner that is restricted to only finding simple straight line paths between two configurations.

In our scenario, we use the analog of a straight-line path as it applies to our particular configuration space and restricted local walk algorithm, described in Section 5.1. Given two configurations, $\mathbf{c}_1$ and $\mathbf{c}_2$ that we would like to connect, the local planner places the avatar in $\mathbf{c}_1$ and performs the local-walk step repeatedly in the direction of $\mathbf{c}_2$ until either the configuration $\mathbf{c}_2$ is reached or the planner determines there is no direct path to $\mathbf{c}_2$. The details of the local planner are presented in Section 5.2.

Intuitively paths in this local planning method consist of only a single rotation followed by a continuous, collision free, walk, in the forward direction until the goal is reached. Note that such paths need not be straight lines, as the avatar may move across several disjoint walkable surfaces (triangles), perhaps even with different orientations, as in the case of walking up or down stairs.

4.3 Roadmap Computation

A roadmap of the environment is defined as a graph in which nodes represent configurations in $\mathcal{C}_{free}$, and edges are used to connect nodes for which the corresponding configurations can be connected by the local planning method, described above. In the general case this graph is a directed graph, but, since in our implementation we have chosen the parameters of the avatar so that any path can be followed in either direction, in practice the roadmap graphs we produce are undirected.

Our roadmap computation algorithm is a variation of the VisPRM algorithm presented in [Simeon et al. 2000]. We use the term *reachability* as a more general term for visibility. In order to ensure a maximum distance between graph nodes in the environment, we limit the reachable region of a node by a radius. Our algorithm allows a "connector" node to connect more than two "guard" nodes, and adds "connector pruning" to eliminate redundant connectors. This section presents the details of our algorithm.

4.3.1 Reachability

Given a local planning method $\mathcal{L}$, the reachability domain of a configuration $\mathbf{c}$ and a radius r, is defined as the set:

$$Reach(\mathbf{c}, r) = \left\{ \begin{array}{l} \mathbf{c}^{'} \in C_{free} \text{ such that } dist(\mathbf{c}, \mathbf{c}^{'}) \leq r \\ \text{and } \mathcal{L}(\mathbf{c}, \mathbf{c}^{'}) \in C_{free}\} \end{array} \right\}$$

where *dist* is the Euclidean distance function. Intuitively, the reachability domain of a configuration is the portion of a neighborhood around the configuration that can be reached using the local planner. For the reachability domain, $Reach(\mathbf{c}, r)$, the configuration, $\mathbf{c}$, around which domain is centered is defined as the guard of $Reach(\mathbf{c}, r)$.

Our algorithm tests whether a configuration $\mathbf{c}_1$ is in $Reach(\mathbf{c}_2, r)$ for some other configuration $\mathbf{c}_2$, by first testing the distance between the two locations to ensure that it is less than or equal to r. If this is true, then we use the local planning method described in Section 4.2 to test if $\mathbf{c}_2$ is reachable from $\mathbf{c}_1$.

4.3.2 Free Space Coverage

A set of guards, G, is a coverage of $\mathcal{C}_{free}$ for a given reachability radius r, if for any $\mathbf{c}$ in $\mathcal{C}_{free}$, there exists a g in G such that $\mathbf{c} \in \text{Reach}(g, r)$. Since our synthetic environment is bounded, our configuration space is a subset of the union of a finite number of triangles, and hence has finite area, it is guaranteed that a finite coverage always exists. The size of the coverage is proportional to the size of the domain, in relation to the radius r, and the shape of the domain. In our application we do not consider the problem of finding an optimum coverage for the domain, where optimum refers to the smallest possible number of guards. To find such a coverage would require solving an instance of the well know art gallery problem, which is NP-hard [O'Rourke 1997].

4.3.3 Reachability Roadmap

The reachability roadmap is a roadmap of the workspace that is a bipartite graph containing the following node types:

- **Guard Nodes:** Given a fixed radius r_g, called the *guard reachability radius*, each guard node g has the property that no other guard node g' is in the reachability domain $Reach(g, r_g)$. Intuitively this means that all of the guards are mutually unreachable for this given radius.

- **Connector Nodes:** For a given distance, r_c, called the *connector reachability radius*, a connector node represents a configuration, $\mathcal{C}$, such that there are two or more

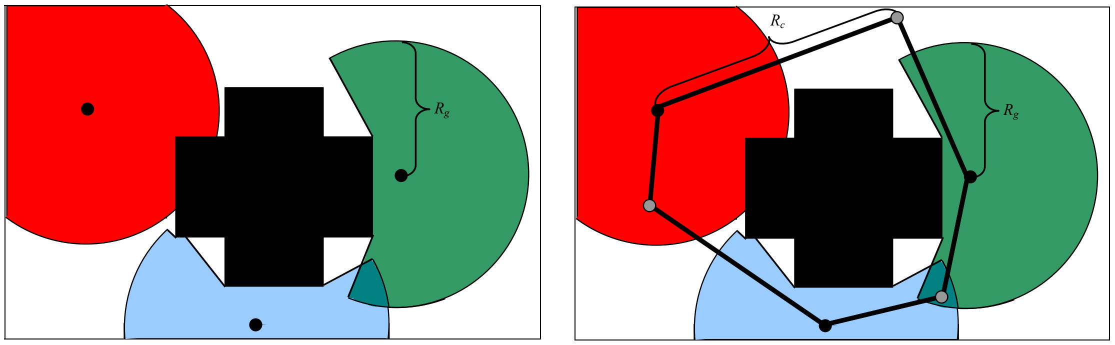

Figure 3: *Left: The reachability domain of three configurations in a simple domain. Right: A roadmap, consisting of three guards (black) and three connectors (grey), which partially covers the domain.*

guard nodes in the reachability domain $Reach(\mathbf{c}, r_c)$. For each connector node, we add edges in the graph between the connector and all guard nodes in this reachability domain.

The guards nodes define a set of reachability domains that, once complete, should cover as much of the free space of the virtual environment as possible. Connector nodes act as links between two or more guards to produce a well-connected graph that is used for path planning. Two parameters control the characteristics of the reachability roadmap. The guard reachability radius, r_g, controls the density of guards in the roadmap. As r_g decreases, more guards are needed to cover the domain. The second parameter, the connector reachability radius, r_c, influences the connectivity of the roadmap by defining the maximum edge length in the graph. If $r_c = r_g$ it will take many samples to connect two guards with approximate separation distance r_g. For this reason we used $r_c = 2r_g$.

4.3.4 Probabilistic Construction of the Roadmap

We construct the roadmap incrementally by randomly sampling the workspace, $\mathcal{C}_{free}$. For each random configuration, $\mathbf{c}$, we search the roadmap for all guards that contain $\mathbf{c}$ in their associated reachability domains, with each of the two radii r_c and r_g. If there are no such reachable guards, $\mathbf{c}$ is added as a guard node. If there are two or more, $\mathbf{c}$ is added as a connector. With only one reachable guard node, $\mathbf{c}$ is discarded.

The reachability roadmap algorithm is given in Algorithm 4.1. Notice that in the construction of the graph we choose to resolve situations where $\mathbf{c}$ can be added as both a guard and a connector, by adding $\mathbf{c}$ as a connector, so that we increase the number of edges in our graph making it more useful for path planning.

4.3.5 Pruning Connectors

As a further optimization to the roadmap algorithm, before any configuration $\mathbf{c}$ is added as a connector to the roadmap, we compare it to all reachable connectors already in the graph. If we find a connector already in the graph, which is reachable from $\mathbf{c}$, that joins all of the same guards as $\mathbf{c}$, then we do not need to add $\mathbf{c}$ to the graph. Any previously added connectors that connect a proper subset of the guard nodes connected by $\mathbf{c}$ are redundant if reachable from $\mathbf{c}$. Such connectors are removed after adding $\mathbf{c}$. This ensures that we only keep the connectors with the highest number of linked nodes, whenever possible.

4.3.6 Sampling

An important aspect of any randomized roadmap planner is the technique used to generate samples in the configuration space. The goals of the sampling strategy are:

```
Build_Roadmap

Input  The model geometry and the desired percentage coverage $p_{cover}$.

Output  A roadmap $R$ that is estimated to cover at least $p_{cover}$ percent of the model.

Repeat {
        Let $\mathbf{c} \in C_{free} \leftarrow$ a random configuration.
        Let $S_{r_c} \leftarrow$ the set of all guards in $Reach(\mathbf{c}, r_c)$.
        if $|S_{r_c}| > 1$
        {
            Add $\mathbf{c}$ to $R$ as a connector.
            Add edges between $\mathbf{c}$ and each
            guard $g \in S_{r_c}$.
        {
        else
        {
            $S_{r_g} \leftarrow$ the set of all guards in $Reach(\mathbf{c}, r_g)$.
            if $|S_{r_g}| = 0$
            {
              Add $\mathbf{c}$ to $R$ as a guard.
              Add edges between $\mathbf{c}$ and each connector
              $c \in Reach(\mathbf{c}, r_c)$.
            }
            else
            {
              Reject $\mathbf{c}$
            }
        }
}until until estimated coverage of $\mathcal{C}_{free}$ by $R$ exceeds $p_{cover}$
return The roadmap $R$.
```

ALGORITHM 4.1: Build Roadmap

- To sample the configuration space uniformly by area
- To distribute the samples uniformly in the environment
- To have, at all stages of the sampling, an upper bound on the worst case distance between the samples

While our sampling strategy should be automatic and make no assumptions about the environment, it is possible to take advantage of the constraints imposed on the avatar's motion and satisfy the stated goals. The key observation is that the direction of the gravity vector in the scene plays a large role in determining the walkable portions of the model. Since our constrained movement model limits the gradient of surfaces which the avatar can ascend or descend, the walkable triangles are nearly perpendicular to gravity. Thus,

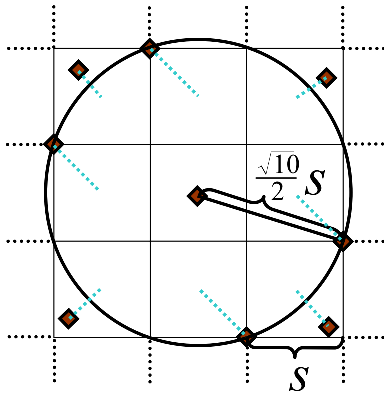

Figure 4: *An example of the largest radius around a sample that can be empty of any other samples, in our sampling scheme. S is the length of the grid cells in the finest resolution grid that has been fully sampled. The light dotted lines indicate the center of the cell from which each sample is taken*

we can sample the space by shooting random rays through the model, along the direction of gravity, and generate sampling that is efficient and also very closely approximates a sampling that is uniform by area. We intersect these rays with polygons in the model and create samples wherever the avatar is able to stand on the intersection point.

It may appear that such a sampling scheme is biased because samples are correlated in the direction of gravity. But since the avatar is not able to fly, or to walk up along the vertical direction, samples that are geometrically correlated in the vertical direction are not correlated in the configuration space of the avatar. This is because the avatar can only move up or down indirectly, as a result of horizontal motion, such as in the example of walking up or down stairs.

To distribute the samples in the environment and obtain a bound on the worst case spacing between samples we shoot the rays through the environment from a plane positioned outside the model and oriented perpendicular to the gravity vector, and partition this plane with a two dimensional grid. For each cell in the grid, a sampling ray is originated at a random point within the cell. After a ray is shot for each cell, we divide each cell into four smaller cells in the manner of a quad-tree. Initially, the cell length is chosen to be twice the guard reachability radius, r_g, so that the sampling starts coarsely, with few samples within reachable distance of each other, and becomes more refined as more samples are taken. At each stage of the sampling, the length of the grid cells in the finest resolution grid that has been fully sampled, S, gives a limit on the radius of the neighborhood around a sample that does not contain any other samples. Figure 4 shows an example of the largest empty region around a sample. The radius of this region can be at most $\frac{\sqrt{10}}{2}S$ units. In Figure 4, the light dotted lines indicate the center of the cell from which each sample is taken.

4.4 Analysis of the Roadmap

In this section, we analyze the size and coverage of the roadmap computed by our algorithm.

4.4.1 Size of Roadmap

The size of the roadmap is determined by the guard reachability radius and the size and complexity of the environment. The number of guards is limited by the maximum number of guards that can be placed in the environment without mutual reachability. Our connector pruning strategy ensures that the number of connectors is also limited by mutual visibility, with a strong tendency to create the smallest number of connectors possible to link any given set of guards. In practice we have observed that roadmaps for our environments typically have less than one connector for every guard.

4.4.2 Estimating Coverage

We terminate the roadmap construction when we have achieved the desired coverage of our environment by the union of the reachability regions of the guards. To obtain a probabilistic estimate on the ratio $A_{reachable}/A$, of guard reachable area to total area of the model, we use the fact the ratio is equivalent to the probability, $p_{reachable}$, that a random sample taken in the workspace will fall in an area reachable by the guards. To provide a probabilistic bound on the value of $p_{reachable}$, for a given roadmap, we could take N random samples of the workspace and record the number, $N_{reachable}$, of those samples that land in reachable space. As N grows large, approaching infinity, the ratio $N_{reachable}/N$ converges to $p_{reachable} = A_{reachable}/A$. But we desire an estimate on the coverage as the graph is being constructed. For efficiency it is important to have a bound on this ratio that does not require additional samples to be generated just to estimate the bound. So as we build the roadmap, we maintain a tally of the number of samples that are reachable from at least one guard. Since we never remove guards from the roadmap, any sample that is visible by at least one guard in the partially constructed roadmap will also be visible by at least one guard when the roadmap construction algorithm is terminated. Moreover, any configuration that is not visible by at least one guard will be added to the roadmap as a guard, and so may increase the amount of reachable space. Thus, the ratio of reachable samples to total samples as the roadmap is being constructed is a lower bound on the ratio $N_{reachable}/N$ from a complete graph. This tally is easy to maintain, and provides a lower bound probabilistic estimate of the probability $p_{reachable}$. Therefore, it provides a conservative estimate on the quality of the roadmap that can be used as a termination condition for our algorithm.

Note that for any target coverage of the configuration space the roadmap algorithm is guaranteed to terminate in the asymptotic sense, provided that the sampling is not biased. This is because our configuration space is finite, and every new guard added must increase the coverage towards 100%. Also, in the limit our sampling strategy will sample the entire C_{free}.

5 Local-Walk and Navigation

In this section, we present a local navigation mode, which we call *local-walk*.

Given, as input, some desired motion for the avatar, the local-walk system converts it into a motion that satisfies the following constraints:

- The avatar must never penetrate objects in the environment.

- The avatar must always be on a walkable surface.

A surface is considered *walkable* if the surface normal is within a tolerance angle of the up vector defined for the environment. This criterion prevents the avatar from walking up or down unreasonably steep slopes. The two constraints on the avatar's motion are enforced to provide a realistic movement model that is similar to the motion of a human walking through the environment.

The local-walk algorithm plays three roles in our system. First, in the interactive mode, it converts user input directly into valid avatar motion, allowing the user to drive the avatar

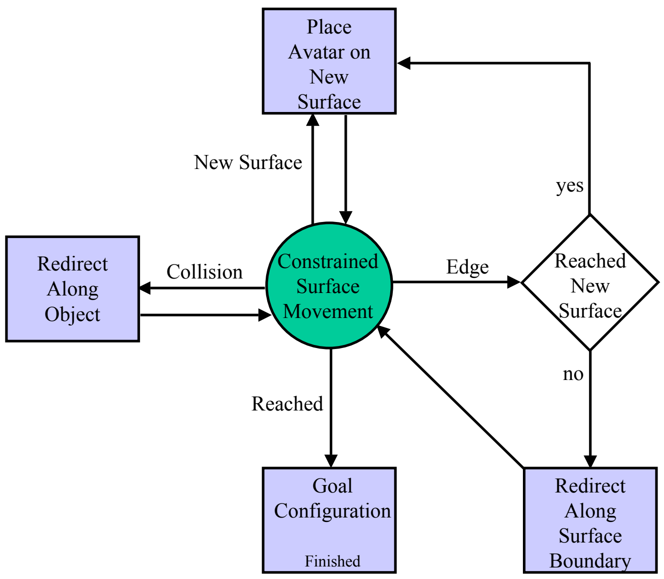

Figure 5: *Overview of the local-walk algorithm: The circle represents the default walking mode. Different events are represented by directed edges out from the circle and the boxes represent the responses to the events.*

through the environment. Second, the roadmap computation algorithm presented in Section 4 uses local-walk as part of local planner when testing the reachability of samples. Finally, local-walk is used during the path following stage to automatically drive the avatar is along the computed path.

5.1 Local-Walk

We assume that the avatar starts in a valid configuration. The local-walk algorithm receives a goal position and moves the avatar along the floor surface towards the goal. This default walking mode can be interrupted by different events and the algorithm responds to those events. The events are listed below and also highlighted in Figure 5.

Goal Reached The goal position is reached and the avatar stops moving.

Collision If the avatar encounters an obstacle in the scene, the avatar's motion is projected along the obstacle edge. This is illustrated in Figure 6(a).

New Surface As the avatar moves it may come into contact with a surface above the current surface as in Figure 6(b). The avatar responds by transitioning to the new surface and continuing along the desired motion vector.

Edge If the boundary of the current surface is encountered, there are two possible responses. The algorithm looks for a surface along the motion vector that the avatar may step onto (collision free). The avatar has a limited distance it may step up, down, or outward. If such a surface is found, it is handled exactly as in the *New Surface* event. Otherwise, the avatar's motion is projected along the boundary as in the *Collision* event. In Figure 6(c) the latter case is highlighted.

Collisions below a certain height with surfaces that are deemed non-walkable based on their normals are permitted so that the avatar is able to step over low obstacles. The avatar is redirected by Edge and Collisions events, but never allowed to move in a direction that makes an angle greater than 90 degrees with the original intended direction.

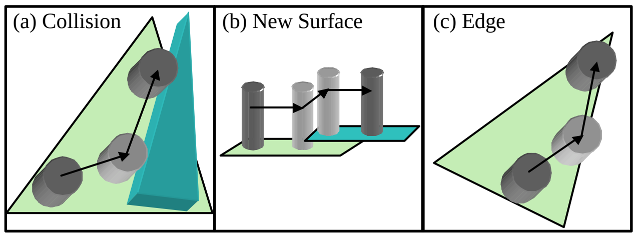

Figure 6: *Three possible events: (a) When the avatar reaches the boundary of the surface and there is no reachable surface to which to step, the avatar's motion is aligned to move along the surface boundary. (b) Similarly, to avoid obstacle penetration, the avatar's motion is projected along the edge of obstacles. (c) If the avatar encounters a surface above the current surface, a surface transition is made.*

5.2 Local-walk as a Local Planner

The local-walk algorithm is modified slightly when used as a local planner. The input is specified to move the avatar along the vector between the start position and the goal position in the plane of the surface of the start configuration.

If a collision occurs or an edge event occurs and no surface can be transitioned to, the local planner determines that the goal position is not reachable from the start position in a single local plan step. Additionally, the avatar is reoriented towards the goal position after transitioning between walkable surfaces.

This method of moving the avatar has a potential pitfall. While crossing a small gap when transitioning surfaces, the avatar may pass directly over or under the goal position. The local planner will continually oscillate the avatar between the sides of the gap. To correct this, if the avatar passes above or below the target configuration, the local planner determines that it cannot connect the configurations.

5.3 Path Searching

Once we have generated a roadmap for the configuration space that provides the desired amount of coverage, we use it to generate paths between some given start and goal locations. We first link the start and goal configurations to the roadmap. In particular, we add the start and goal configurations to our graph as query nodes and create edges between these nodes and all other nodes reachable from them in less than the connector link distance, as described in Section 4.

Once these nodes have been added we can execute any graph search to find a path from the start to the goal. In our approach, we use the well-known IDA* algorithm [Korf 1985], which uses iterative depth first searches with an increasing distance cutoff.

5.4 Following Paths

Once a path has been found between the desired start and goal configurations, we render the path for the user and also provide an automatic navigation function that propels the avatar along that path. This automatic navigation function walks the avatar along the path in the same manner as the local-walk described in Section 5.1. In addition to following the path, our algorithm also performs trimming, which cuts redundant corners in the path. It checks for these corners in real-time as the avatar is traversing the path. Specifically, it checks whether any nodes in the path ahead of the avatar are reachable, in the sense of our local planning method, from the current position. If so, the avatar is redirected to this node from its current position to smooth the path.

6 Implementation and Performance

In this section, we describe the implementation of our algorithm and highlight its performance in two environments.

6.1 Implementation

Our system is implemented in C++ and runs on the Windows PC platform. We use a public domain collision detection library, Proximity Query Package (PQP) [Larsen et al. 1999] for collision checking. We have also used several techniques to accelerate the performance of our system.

6.1.1 Spatial Partitions

To take advantage of the spatial coherence, we use a regular grid spatial data structure to accelerate the local-walk algorithm, roadmap construction, and rendering.

We use bounding volume hierarchies for collision detection. They are computed for each cell. This localizes the collision detection that is performed when invoking local-walk during the preprocess and at runtime. As to be explained in Section 6.2, the cells are also used in memory management.

To accelerate the rendering of complex environments, we combine the spatial grid along with a hardware supported occlusion query, GL_NV_occlusion_query, available on recent NVIDIA GPUs [Hillesl et al. 2002]. The goal is to render the grid cells in a front to back order from the viewpoint, and use the query to cull away cells that are occluded. We combine it with view frustum culling and are able to obtain reasonable frame rates, $8 - 15$ frames per second (on the power plant), in most regions of the environment.

The same cell grid is also used in the preprocessing phase for roadmap computation. Because nodes have a limited reachability radius, when searching for reachable guards or connectors from a sample we need only attempt local planning steps to nodes in nearby cells.

6.1.2 Graph Search Acceleration

As described in Section 5.3, we use the IDA* algorithm [Korf 1985] to quickly search our roadmap for paths between the user specified start and end goals. We accelerate the IDA* algorithm by using a transposition table that maintains a cache of history records for the nodes traversed in an iteration, and prevents the algorithm from traversing the nodes more than once when it doesn't improve the search result.

A second acceleration technique that we use is performing connected component analysis of the roadmap. Connected component analysis of a graph decomposes a given graph into strongly connected components, such that all nodes within a component are reachable from each other. By labelling each node in the graph with a component index we can easily test if there exists a path between any given pair of start and end nodes. Without this information the path-planning algorithm will have to traverse the entire graph before it would be possible to conclude that no path exists between the nodes.

6.2 Memory Management

Memory management is an important issue in handling massive environments. Our system's most significant memory requirement (aside from the given model geometry) arises from PQP bounding volume hierarchies (BVHs) for collision detecting. The roadmap graph contains on the order of thousands of nodes and can be stored in less than 1MB.

If the bounding volume information is stored for the entire model geometry it would at least double the memory usage. For applications in which the memory constraints do not permit the retention of PQP representations of the entire model, we use employ a cache for managing BVHs. When the avatar encounters a cell whose BVH is not in the cache it is added to the cache. If the cache was already full a least-recently-used policy is used to delete a BVH.

When a user is interacting with the environment, the constrained movement model ensures that the avatar's motion has high spatial coherence. Thus, a small cache for storing the BVHs can provide interactive performance in most cases. Additionally, because sampling is performed by iterating through a 2D grid, local planning steps in the preprocess also have high spatial coherence.

The number of cells that need to be cached depends on the size of the model, the resolution of the partitioning grid, and the system memory.

6.3 Interaction

In our current implementation, interaction is facilitated by the use of the keyboard arrow keys and the mouse. Our interface has been motivated by first-person computer games, where the arrow keys are used for translation and the mouse is used to turn and tilt. There is a limit to how far the user may tilt the camera up or down to account for the fact that a user can only tilt his head by a limited angle. The arrow key directions up, down, left, right are mapped respectively to walk forward, backward, left, right. Eight possible walking directions are available by pressing a combination of arrow keys. Pressing two opposite keys causes no effect.

6.4 Performance

We have tested our implementation on two synthetic environments, a modest factory room consisting of over ten thousand triangles, and a power plant model composed of over twelve million triangles.

6.4.1 Roadmap Computation

The navigation roadmap computed by our system for the factory room model is shown in Plate 1. Plate 2 shows a small portion of the roadmap generated for the power plant model. This image illustrates how our roadmap spans not only the traditional walking surfaces of the model, such as hallways and stairs, but also portions of the model, such as beams and trusses that are not traditional walking surfaces, but still satisfy our local driving model constraints. The size of the roadmap generated for the environment, its estimated coverage, and the preprocess time taken to compute the roadmap are shown in Table 1.

For the factory room environment, the growth of the roadmap, as random samples are taken, is shown in Figure 7. The change in estimated convergence as samples are added is also shown in Figure 7. Notice that in the early stages of the roadmap algorithm, most samples explore new reachable space, and hence are added to the roadmap. These early samples quickly cover the open areas of the environment with their reachability regions. As more samples are taken, the unexplored portion of the model becomes smaller and more fragmented. Thus, as the sampling grows more dense, more and more samples are in the already reachable space and so are found to be redundant. When this happens, both the size of the roadmap and its estimated coverage increase much more slowly.

6.4.2 Path Query

Our path query system is guaranteed to find a path between any user specified start and end locations in the model, provided that such a path exists in our roadmap. Thus, the answer to the question of whether a path query will be successful is strongly coupled to the degree to which the roadmap represents the connectivity of the free space. Even for a roadmap that covers a high percentage of the environment, the accuracy with which this roadmap captures the

Scene	Polygons	**Samples**	**Nodes**	**Coverage**	**Time**
(1) Factory Room	10143	5000	71	99.2%	1.3 hours
(2) Power Plant	12541083	30000	5304	88.03%	13.7 hours

Table 1: Roadmap statistics for two environments. **Samples:** The number of random samples taken. **Nodes:** The total number of nodes in the included in the roadmap. **Coverage:** The roadmap's estimated coverage of the environment. **Time:** The total time taken for the roadmap construction preprocess.

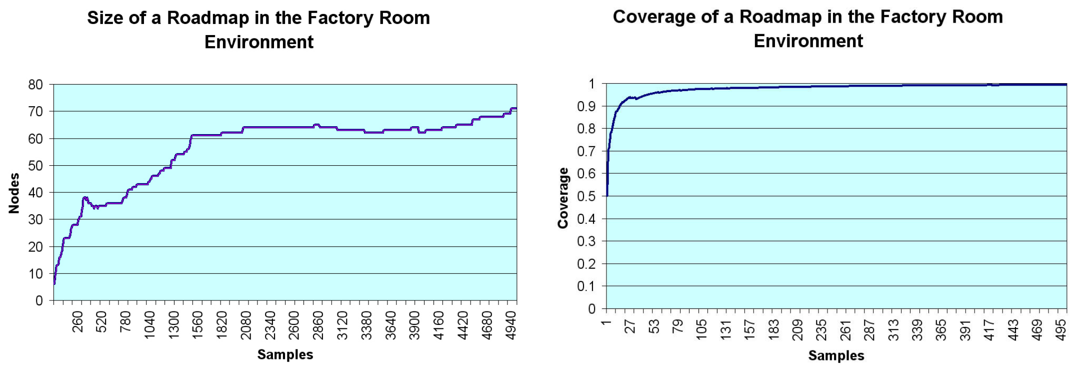

Figure 7: *Left: A graph of the roadmap size as a function of the number of samples takes, for the factory room scene. Right: A graph of the roadmap coverage as a function of the number of samples takes, for the factory room scene.*

connectivity of the model depends highly on the complexity of the model's free space. This is often refereed to as the *Narrow Passage* problem in the motion planning literature. It is very difficult for a probabilistic algorithm to link areas of the environment that are separated by a very narrow passage, because it is very unlikely that a sample will be randomly generated in such an area. Many techniques have been proposed to help deal with the problem of narrow passages. One possibility is to use medial-axis based sampling [Foskey et al. 2001].

Both of the test environments contain passageways that are narrow with respect to the avatar. We have found that when the roadmap has achieved high coverage of the environment (around 90%), the path finding algorithm is able to find paths between almost all connected model regions. Moreover, due to the acceleration techniques used to speed up the path query, described in Section 6.1.2, the path query time is on the order of one or two seconds. This makes the path query model a fast and easy tool to aid the user in exploring the model in real-time.

6.4.3 Example Paths

Our color plates show four examples of paths that have been automatically computed by our system. In each image we show several configurations of the avatar as it traverses the path. Plate 3 shows a portion of a path through the hallways and stairways in the power plant model. Plate 6 shows another portion of a path in the power plant. In this case, the avatar walks along the beams at the top of the power plant in order to achieve a short path to the goal, again illustrating how our algorithm explores all walkable surfaces.

Plate 4 shows a complete path from a start location, denoted by the green umbrella at the bottom on the image, to a goal location denoted by the red umbrella at the top of the image. This model, which consists of many disjoint platforms, conveyer belts, and other pieces of machinery, illustrates that our algorithm is able to plan paths in an environment that lacks a clear decomposition into functional units, such as walkways, floors, or stairways.

Plate 5 shows the avatar traversing another portion of a path in the power plant model. As the avatar follows the path, portions of the path are skipped, and replaced with straight line segments by a look-ahead computation, whenever this can be done in a manner that shortens the path while still maintaining the constraints of the movement model.

7 Conclusions

We have presented an approach for navigating complex environment based on path planning algorithms. It includes a novel approach to precompute the roadmap of a complex environment that uses a combination of randomized sampling, collision checking, local planning and reachability constraints. At runtime our algorithm searches the roadmap graph to compute a path between two locations in the environment and steers the user along the path. It also enables local user-steered exploration of the scene and imposes constraints on the motion avoiding collision and moving along surfaces. We have applied it to two complex environments and the initial results are promising.

7.1 Limitations

There are several limitations of our approach. Because the roadmap graph is fairly sparse and the avatar follows the path in linear segments, paths may sometimes look unnatural, especially when compared to a hand-selected or user-steered path. Our look-ahead technique for smoothing the path can trim over-shot corners but still uses linear path segments. Using a path traversal similar to that in [Nieuwenhuisen and Overmars 2003] may lead to better results.

The precomputation process is time consuming. It precludes interactively recomputing the graph for a dynamic environment. It may be useful to alter the environment and note how this alters walkable pathways between various locations.

7.2 Future Work

There are many avenues for future work. We would like to apply our algorithm to more complex environments and also like to perform a formal user-study to evaluate the benefits of our navigation algorithm. We would also like to combine it with interactive rendering algorithms to handle even more complex environments, where frame rate must be more carefully managed by LODs [Govindaraju et al. 2003]. We currently use a simple mouse-based 2D interface for input and it may be useful to explore other input devices or even integrate our algorithm with an immersive environment where the user performs real-walking in the environment and uses body gestures. Additionally, it may be possible to perform local graph recomputation to handle moderate changes in the environment.

7.3 Acknowledgements

This work has been supported in part by ARO Contract DAAD19-99-1-0162, NSF awards ACI 9876914, IIS-982167, ACI 0118743, ONR Contracts N00014-01-1-0067 and N00014-01-1-0496, a DOE ASCI grant, and by Intel Corporation. The Power Plant model is courtesy of an anonymous donor. We would like to thank the members of UNC Walkthrough group for useful discussions and support, in particular Karl Hillesland and William Baxter.

References

Canny, J. 1988. *The Complexity of Robot Motion Planning*. ACM Doctoral Dissertation Award. MIT Press.

Darken, R., and Sibert, J. 1996. Navigating in large virtual worlds. *The International Journal of Human-Computer Interaction 8*, 1, 49–72.

Egbert, P. K., and Winkler, S. H. 1996. Collision-free object movement using vector fields. *IEEE Computer Graphics and Applications 16*, 4 (July 1996), 18–24. ISSN 0272-1716.

Foskey, M., Garber, M., Lin, M., and Manocha, D. 2001. A voronoi-based hybrid planner. *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*.

Govindaraju, N., Sud, A., Yoon, S., and Manocha, D. 2003. Interactive visibility culling in complex environments with occlusion-switches. *Proc. of ACM Symposium on Interactive 3D Graphics*.

Hillesland, K., Salomon, B., Lastra, A., and Manocha, D. 2002. Fast and simple occlusion culling using hardware-based depth queries. Tech. Rep. TR02-039, Department of Computer Science, University of North Carolina.

Hoff, K., Culver, T., Keyser, J., Lin, M., and Manocha, D. 2000. Interactive motion planning using hardware accelerated computation of generalized voronoi diagrams. *Proceedings of IEEE Conference of Robotics and Automation*.

Hong, L., Muraki, S., Kaufman, A., Bartz, D., and He, T. 1997. Virtual voyage: Interactive navigation in the human colon. *Proc. of ACM SIGGRAPH*, 27–34.

Hubbold, R., and Keates, M. 2000. Real-time simulation of a stretcher evacuation in a large-scale virtual environment. *Computer Graphics Forum 19*.

Igarashi, T., Kadobayashi, R., Mase, K., and Tanaka, H. 1998. Path drawing for 3d walkthrough. In *Proc. of UIST*, 173–174.

Kavraki, L., and Latombe, J. C. 1994. Randomized preprocessing of configuration space for fast path planning. *IEEE Conference on Robotics and Automation*, 2138–2145.

Khatib, O. 1986. Real-time obstable avoidance for manipulators and mobile robots. *IJRR 5*, 1, 90–98.

Korf, R. E. 1985. Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence 27*, 97–109.

Kuffner, J. 1998. Goal-directed navigation for animated characters using real-time path planning and control. *Proc. of CAPTECH: Workshop on Modeling and Motion Capture Techniques for Virtual Environments*.

Larsen, E., Gottschalk, S., Lin, M., and Manocha, D. 1999. Fast proximity queries with swept sphere volumes. Tech. Rep. TR99-018, Department of Computer Science, University of North Carolina.

Latombe, J. 1991. *Robot Motion Planning*. Kluwer Academic Publishers.

Li, T.-Y., and Ting, H.-K. 2000. An intelligent user interface with motion planning with 3d navigation. *Proc. of IEEE VR*.

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

Lozano-Pérez, T., and Wesley, M. 1979. An algorithm for planning collision-free paths among polyhedral obstacles. *Comm. ACM 22*, 10, 560–570.

Mackinlay, J. D., Card, S. K., and Robertson, G. G. 1990. Rapid controlled movement through a virtual 3D workspace. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, F. Baskett, Ed., vol. 24, 171–176.

Nieuwenhuisen, D., and Overmars, M. 2003. Motion planning for camera movements in virtual environments. Tech. Rep. UU-CS-2003-004, Utrecht University, Department of Information and Computing Sciences.

O'Rourke, J. 1997. Visibility. In *Handbook of Discrete and Computational Geometry*, CRC Press LLC, Boca Raton, FL, J. E. Goodman and J. O'Rourke, Eds., 467–480.

Overmars, M. H., and Švestka, P. 1995. A probabilistic learning approach to motion planning. In *Algorithmic Foundations of Robotics*. A. K. Peters, Wellesley, MA.

Robinett, W., and Holloway, R. 1992. Implementation of flying, scaling, and grabbing in virtual worlds. 189–192.

Shneiderman, B., and Maes, P. 1997. Direct manipulation vs. interface agents. *Interactions 4*, 6, 42–61.

Simeon, T., Laumond, J. P., and Nissoux, C. 2000. Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics Journal 14*, 6.

Slater, M., Usoh, M., and Steed, A. 1994. Steps and ladders in virtual reality. In *ACM Proceedings of VRST*, 45–54.

Tan, D. S., Robertson, G., and Czerwinski, M. 2001. Exploring 3d navigation: Combining speed-coupled flying with orbiting. In *Proceedings of CHI*, 418–425.

Usoh, M., Arthur, K., Whitton, M., Bastos, R., Steed, A., Slater, M., and Brooks, F. 1999. Walking ¿ walking-in-place ¿ flying in virtual environments. In *Proc. of ACM SIGGRAPH*, 359–364.

Walkinside. 2002. http://www.walkinside.com.

Wilson, A., Larsen, E., Manocha, D., and Lin, M. C. 1999. Partitioning and handling massive models for interactive collision detection. *Computer Graphics Forum (Proc. of Eurographics) 18*, 3, 319–329.