Automatic Generation of Camera Motion to Track a Moving Guide

Onno Goemans and Mark Overmars
Institute of Information and Computer Sciences
University of Utrecht, The Netherlands
Email: onno,markov@cs.uu.nl.

July 20, 2004

Abstract

Manually following a moving object through a cluttered virtual environment can be challenging for the user. Instead, one would typically rather focus on watching the object and its environment. In this paper, we present an approach to automatically generate a camera motion, such that the user maintains visibility with an object moving along a known path through a virtual environment. To begin, the user specifies the camera placement at the start and end of the object path, and constraints on the camera placement relative to the object. Given this input, the system computes a smooth camera path, satisfying the constraints: Firstly, an initial camera path is generates by applying a single-shot probabilistic road map technique. We augmented this technique with several optimisations, speeding up the path generation considerably. Then, the initial path is smoothed to present the user with a pleasant camera motion. The approach has been implemented and tested: it is fast and computes paths through complicated 2D and 3D environments in less than a second.

1 Introduction

A growing public is coming into contact with virtual environments as a result of developments such as faster and cheaper 3D graphics technology and the increasing use of broadband internet connections. Crucial in all virtual environment applications is effective control of the user's viewpoint, or virtual camera. For novice users, such camera control can be challenging, and when, on top of that, the novice needs to carry out a task within a limited time frame, such a challenge can easily become overwhelming.

A typical task is camera navigating in order to follow a guide (object) moving through the virtual environment. In this paper, we present a technique to generate such camera motion automatically, under the assumption

that the guide moves along a known path. Beforehand, the user specifies both the placement of the camera at the start and end of the guide path, and constraints on the camera placement relative to the guide. The generated camera motion must satisfy the user's constraints as well as the "visibility constraint"; the latter demands that the camera maintains visibility with the guide at all times; see Fig. 1(a) and (b).

In the last decade, a lot of research has been done to assist the user in controlling the camera [9, 13, 14, 22, 26] and to control the camera automatically. The latter area of research can be roughly divided into three groups of techniques. To start, the group of techniques that plan camera motion as a sequence of shots. The shots are either based on cinematographic rules [6, 15] or calculated by solving a set of constraints [2, 8, 11]. A second group consists of techniques that generate a collision free camera motion from a start to a goal placement through a virtual environment [4, 7, 23, 24]. And lastly, the group of techniques that generate a camera motion to track a moving guide. Most of these techniques assume the path of the guide to be unknown [10, 12], and are therefore not suitable to solve our problem; that is, a smooth motion satisfying the visibility constraint would not be guaranteed.

LaValle *et al.* [18] proposed an approach that describes the guide path as a sequence of configurations in a two-dimensional environment. For each of these configurations, the approach first determines the visibility region, which are then used to calculate an optimal path. We expect that the visibility region approach is not viable in a three-dimensional environment because of large computation times.

Under the assumption of a known guide path, Li *et al.* proposed an approach to plan a camera motion through a two-dimensional environment [20, 21]. While maintaining visibility at all time, their approach optimizes several additional camera-specific criteria. Expanding on this approach, a camera configuration is described as a position relative to a guide position; furthermore, a guide position is described as a point on the predefined guide path at some moment in time. All possible camera configurations together form a continuous space, which is approximated with a uniform grid of configurations. To find a collision-free camera path in this configuration-time space, the approach applies a best-first planning algorithm. Li *et al.*'s approach is limited to two-dimensional environments that are not too complicated, as otherwise the grid becomes too large. That is why, although their approach inspired our representation for camera configuration, we use a different planning approach capable of efficiently operating in large 3D environments.

Our aim is to efficiently solve the given problem for two-dimensional as well as three-dimensional environments. The general idea is as follows: All possible placement combinations of camera and guide together form the configuration space. The relevant subspace contains all configurations satisfying

both the visibility constraint and the constraints on the camera placement relative to the guide. First, we apply a single-shot probabilistic roadmap method, or PRM [1, 3, 16, 17, 25], to construct a roadmap in this relevant subspace. Once the roadmap connects the start and goal configuration, a valid camera motion is found. We then smooth this initial motion in order to eliminate unpleasant camera movement.

The paper is structured as follows. The first four sections address the problem in a 2D workspace, starting with section 2, which provides a more detailed discussion of the guide, camera, and configuration space. Next, we present the basic approach to generate an initial camera path in section 3, along with two improvements. Section 4 then discusses the application we built to test the approach, explains the automation of a PRM-parameter, and provides test results; complex test cases are solved within one second. After this, section 5 discusses the smoothing of the initial camera path. And finally, the extension of the planner to 3D environments is presented in section 6, which is almost straight-forward and solutions are found equally fast.

2 Preliminaries

For the moment, we assume that the guide and camera move in a two-dimensional environment. We specify a guide placement g with three parameters: two for the guide position $\mathcal{W}(g)$ in the workspace, and one for the guide orientation. The guide moves along a known path $\mathcal{GP}$ through the workspace. The *length offset lo* is the normalized length along $\mathcal{GP}$ between configuration $g \in \mathcal{GP}$ and the start configuration. The guide path is defined as a mapping from length offset to guide configuration, e.g. $\mathcal{GP}(0)$ and $\mathcal{GP}(1)$ uniquely identify the start and end configuration of $\mathcal{GP}$.

As proposed by Li *et al.* [20, 21], we specify the camera configuration v relative to a given guide configuration g. The *tracking distance* is the Euclidean distance between v and g in the workspace; and the *tracking angle* is the orientation of the vector $\overrightarrow{\mathcal{W}(g)\mathcal{W}(v)}$ in the workspace. We choose the camera orientation such that the camera always looks straight at the guide. Length offset, tracking distance and tracking angle together uniquely define the position and orientation of the camera. See figure 1(c).

We refer to the imaginary line segment connecting the camera and the guide as the *viewing line*. Recall that the user defines constraints on the camera placement relative to the guide. These constraints are a minimum and maximum on the tracking distance, and a minimum and maximum on the tracking angle. Note that the constraints on the tracking angle are specified relative to the guide orientation. Thus if the guide orientation changes then the allowed tracking angle range changes as well.

Let us now look at the configuration space in which we will solve our

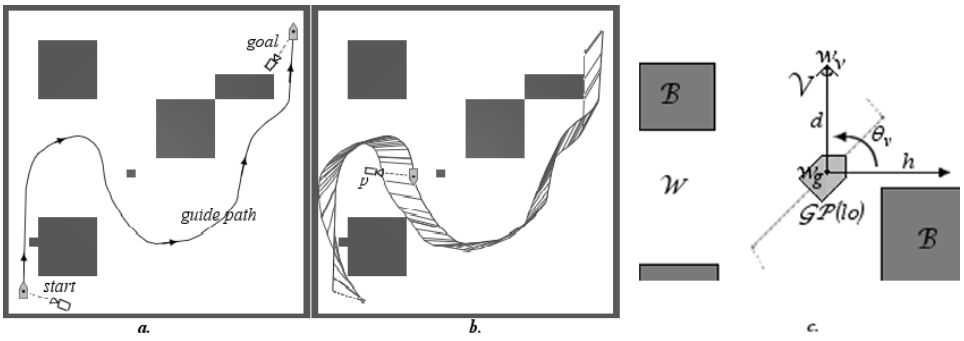

Figure 1: Figure (**a**) depicts the guide path, as well as the camera placements at the start and end of the guide path. Figure (**b**) shows a possible camera path from the start to the goal. In figure (**c**) the camera configuration v is defined relative to guide configuration $\mathcal{GP}(lo)$ by tracking distance d and tracking angle θ_v.

motion planning problem. A configuration describes a combined placement of guide and camera; that is, a guide configuration on the guide path, and a camera configuration relative to this guide configuration. We specify such a configuration with two parameters: the length offset lo, which specifies the guide configuration, and the tracking angle ta, which describes the camera configuration relative to $\mathcal{GP}(lo)$. The configuration description does not contain the tracking distance, because we fix the tracking distance at its user-defined minimum. The motivation for this choice is that if no solution exists for the minimum tracking distance, then no solution exists at all. The actual tracking distance can easily be computed during a post-processing phase. We refer to the configuration space as the *configuration-length offset space*, or $\mathcal{CLO}$-space. A path through the $\mathcal{CLO}$-space is a *camera path*.

Not all $\mathcal{CLO}$-configurations are valid, because of the visibility constraint and the user's constraints on the tracking angle. We refer to the valid part of the $\mathcal{CLO}$-space as *free* $\mathcal{CLO}$-space, or $\mathcal{CLO}_f$. See figure 2.

We assume that the guide only moves forwards along its path, and that the camera only moves when the guide moves. Consequently, a path through the $\mathcal{CLO}$-space must be increasingly monotone in the length offset direction. A camera path is free if it lies entirely in the $\mathcal{CLO}_f$ space; which means that the the viewing line never intersects any obstacles in the workspace. See figure 2.

3 Motion planner

The user specifies the start configuration, q_{start}, and goal configuration, q_{goal}, which are respectively at length offset 0 and 1. Now the goal of the motion planner is to generate a free camera path $P_{solution}$, which connects q_{start} and

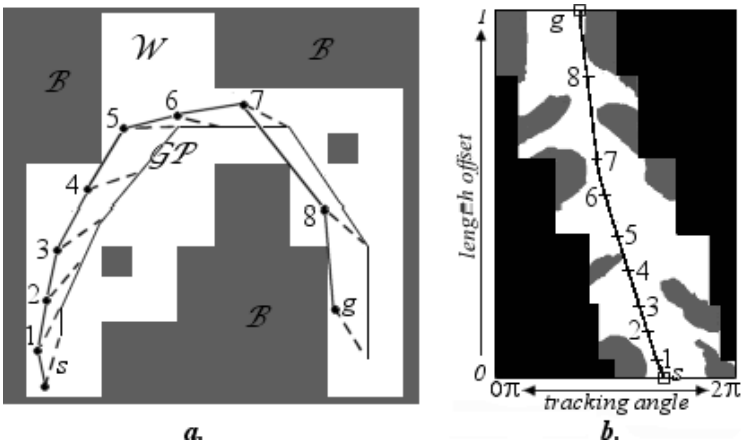

Figure 2: Figure (**a**) shows a workspace, where the sequence of line segments tagged with $\mathcal{GP}$ is the guide path. Configurations s and g are the start and goal configuration. The sequence of lines segments tagged with numbers is the camera path, where the dotted lines illustrate how the camera looks at the guide.
Figure (**b**) shows the $\mathcal{CLO}$-space. The white area is the $\mathcal{CLO}_f$-space. The gray areas illustrate the collection of occluded configurations. The black areas illustrate the collection of configurations not allowed because of the tracking angle constraints. The line through $\mathcal{CLO}_f$ is the path connecting the start and goal configuration, and corresponds to the camera path in the workspace (figure 2a)

q_{goal}. We apply a single-shot motion planner based on the PRM-approach to construct a graph $G = (V, E)$, or roadmap, in the $\mathcal{CLO}_f$-space. The graph nodes V correspond to free configurations, and edges E to free paths. Now, recall that a path in the $\mathcal{CLO}$-space must be monotone in length offset. This restriction imposes two limitations on the graph: G is a directed graph, and for all edges the length offset of the begin configuration must be smaller than the length offset of the end configuration.

The start and goal configuration are the first two nodes added to the graph. Next, the motion planner chooses a random free configuration q_{new} and adds it to the graph. Then, the motion planner tries to connect q_{new} to the graph by attempting to connect q_{new} to nearby nodes (*neighbors*). And finally, the motion planner checks whether the graph connects q_{start} and q_{goal}. These last three steps are repeated in a loop, referred to as the PRM-loop, until $P_{solution}$ is found.

Algorithm 1 *Overview Motion Planner*

1. $E \leftarrow \emptyset$ *[edges]*
2. $V \leftarrow \{q_{start}, q_{goal}\}$ *[nodes]*
3. ***loop***
4. $\quad q \leftarrow$ *randomly chosen configuration* $\in \mathcal{CLO}_f$ *[Sampling]*
5. $\quad N_q \leftarrow$ *set of neighbors of* q *chosen from* V *[Neighbor set]*
6. $\quad$ *Attempt to connect* q *to its neighbours* *[Connection]*
7. $\quad$ ***if*** G *connects* q_{start} *and* q_{goal} ***then*** *return path* *[Termination]*
8. $\quad$ ***else*** *continue loop*

3.1 Basic motion planner

We need to fill in some details in the basic planner:

Sampling: The motion planner picks a random configuration q_{new} from $\mathcal{CLO}$. The motion planner adds q_{new} to V if the random configuration is free; otherwise, the motion planner repeats the sampling.

Neighbor set: The planner selects the nodes $\in V$ that are within a certain length offset range from q_{new}'s length offset. This length offset range is referred to as the *neighbor set distance*. Subsection 4.2 further discusses the choice of neighbor set distance.

Connection: Next the algorithm attempts to connect q_{new} to each of its neighbors with a *local path*, which is computed by the *local planner*. A local path is the shortest possible path between two configurations in the $\mathcal{CLO}$-space. If a local path is occlusion free, then it is added as an edge to the roadmap.

Termination: The algorithm first checks if q_{start} and q_{goal} are in the same connected graph component. If this test is positive, the algorithm then

queries the directed graph to determine whether G connects q_{start} and q_{goal}. If so, the algorithm returns $P_{solution}$; otherwise, the PRM-loop is repeated.

3.2 "Push" sampling

A narrow passage in the $\mathcal{CLO}_f$-space is a length offset interval for which a relatively small part of the tracking angle interval consists of free configurations. The chance that the sampler randomly generates a free sample in an area of the $\mathcal{CLO}_f$-space is determined by the size of the area. Eventually, enough samples will be generated in a narrow passage to enable the generation of a path though the passage; however, this process is rather time-expensive.

To increase the sample generation in narrow passages, the "push" motion planner pushes a new sample out of the obstacle in case of occlusion as has also been suggested for PRM application in other motion planning problems [1, 25]. The pushing is done by changing the tracking angle value while maintaining the same length offset. Thus the narrower a passage, the higher the expected number of nodes in the passage.

3.3 "Useful" nodes

Occlusion checks require collision tests, which takes up more than 90% of the motion planner's execution time. Hence, the motion planner should avoid occlusion checks whenever possible. This subsection discusses the concept of usefulness in the framework of our study. We first experimented with growing trees, a technique discussed in [19], to reduce useless roadmap growth; however, the gain was only marginal. Due to the monotonicity property, the roadmap graph G is a directed acyclic graph. We exploit this roadmap property; as a result the performance of the motion planner improves drastically.

Let v be a node in G. We refer to the nodes that connect to v as the *ancestors* of v, and the nodes to which v connects as *descendants* of v. We define the length of a path P as the difference in length offset between P's start and goal configuration. Now let av be the ancestor of v with the smallest length offset, and let dv be the descendant of v with the largest length offset. The camera path $P(av, dv)$ is the longest path of which v is a node. If v is not connected to any ancestors and descendants, then the longest path consists only of node v. We denote the longest path containing v by $\widehat{P}(v)$. See figure 3(a).

For each graph-node v we store a $\widehat{P}$-record, which describes $\widehat{P}(v)$. The record stores the length offsets of $\widehat{P}(v)$'s begin and end configuration, which are respectively referred to as $\widehat{P}^0(v)$ and $\widehat{P}^1(v)$. Thus for example $\widehat{P}^0(v)$ in figure 3(a) is the length offset of node a_2.

To describe the usefulness-criterion, we first introduce an alternative

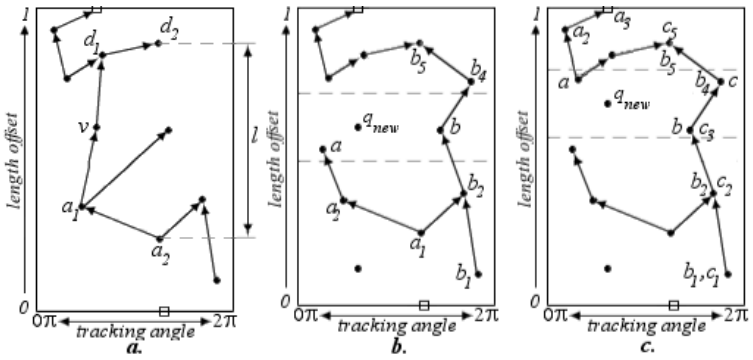

Figure 3: Figure (**a**): The set of ancestors and descendants of node v are respectively $\{a1, a2\}$ and $\{d1, d2\}$. The longest path $\widehat{P}(v)$ is $P(a2, d2)$ with length l.

description of the motion planner's goal. Recall that the motion planner expands the roadmap until a path is found. The length offsets of q_{start} and q_{goal} are respectively 0 and 1, hence the length of $P_{solution}$ is 1. All other paths of the roadmap are shorter. Thus we can redefine the goal of the motion planner as the creation a path of length 1. In other words, the motion planner expands the roadmap until $\widehat{P}$ of one of its nodes is of length 1.

Now let us return to the criterion of usefulness. Recall that the PRM-loop starts with the generation of a new sample q_{new}. The next step is the generation of the neighbor set containing all nodes close to q_{new}. One of the three following scenarios occurs after the neighbor set generation: In the first scenario the neighbor set is empty. In the second scenario, the motion planner can connect q_{new} to the roadmap such that $\widehat{P}(q_{new})$ is potentially longer than the longest path of any of its neighbors; see figure 3(c): if q_{new} is successfully connected to neighbors a and b, then the longest path $\widehat{P}(q_{new})$ is longer than $\widehat{P}(a)$ and $\widehat{P}(b)$. In the third scenario $\widehat{P}(q_{new})$ does not exceed the length of the longest path of one of its neighbors; see figure 3(b): even if q_{new} is successfully connected to neighbors a and b, the longest path $\widehat{P}(q_{new})$ can clearly not become longer than $\widehat{P}(b)$. The sample q_{new} is called useful when one of the first two scenarios occur. We will ony add useful nodes. It can be proven that this approach is probabilistically complete.

Once it is decided that the addition of q_{new} is useful, the next step is to connect q_{new} to the roadmap by adding edges. The addition of an edge is only useful, when it increases the length of the current $\widehat{P}(q_{new})$. Let us fill in the details:

Sampling: We can use the basic sampler or the push sampler to generate q_{new}. The difference is that the motion planner does not perform an occlusion check q_{new} before q_{new} is found to be useful.

Neighbor set generation: The motion planner again compiles the neighbor set N_q by taking all nodes $\in V$ within a certain length offset range from q_{new}. We refer to neighbors with a smaller length offset than q_{new} as *backward neighbors*; and neighbors with a larger length offset than q_{new} as *forward neighbors*. We now determine whether q_{new} is a potentially useful node. If not, q_{new} is discarded and the PRM-loop is repeated.

A longest path of roadmap node v can be divided in two sub paths. The first sub path consists of v and the ancestors of v; and the second sub path consists of v and descendants of v. With this division in mind, we can break the construction of $\widehat{P}(q_{new})$ in two parts as well: the connection of a backward neighbor bn to q_{new}, where $\widehat{P}^0(bn)$ should be as small as possible; and the connection of q_{new} to a forward neighbor fn, where $\widehat{P}^1(fn)$ should be as large as possible.

The motion planner splits the neighbor set N_q into the set of backward neighbors BN_q, and the set of forward neighbors FN_q. The nodes of the first set are sorted on increasing $\widehat{P}_0$ value, and the elements of the latter on decreasing $\widehat{P}^1$ value. See figure 4. So we first try the connections that will lead to the longest paths.

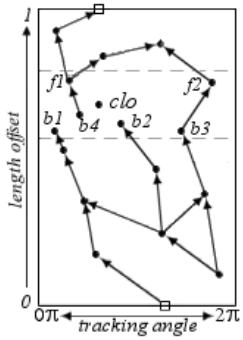

Figure 4: The sorted backward neighbors set BN_q is $\{b1, b2, b3, b4\}$ and the sorted forward neighbour set is $\{f1, f2\}$.

Connection: The motion planner attempts to connect the backward neighbors to q_{new}, starting with the first element of BN_q. This connection process continuous until the local planner generates a free local path, or until all backward neighbors are tested and the corresponding local paths are all found occluded. In the first case the motion planner sets $\widehat{P}^0(q_{new})$ as $\widehat{P}^0(nb)$; otherwise $\widehat{P}^0(q_{new})$ is set to q_{new}. Similarly, the motion planner attempts to connect q_{new} to a forward neighbor, and sets $\widehat{P}^1(q_{new})$ accordingly. Again in case of failure $\widehat{P}^1(q_{new})$ is q_{new}. This approach reduces the number of attempted connections considerably.

Once q_{new} is connected to the graph, the $\widehat{P}$-records of q_{new}'s ancestors and descendants must be updated: $\widehat{P}^0(q_{new})$ is propagated through q_{new}'s ancestors, and $\widehat{P}^1(q_{new})$ is propagated through q_{new}'s descendants.

Termination criterion: The goal of the motion planner is to construct a longest path of length 1. This is the case once the length of $\widehat{P}(q_{new})$ is 1.

4 Experimental results

We implemented the techniques described above in C++. As collision check package we used Solid [5]. All tests are run on a computer with a 2.4 GHz Pentium 4 processor, 512 MB of memory and Windows XP operating system.

4.1 Results

Figure 5, 6 and 7 illustrate the test cases. Each left picture depicts the workspace containing the guide path, and the start and goal camera configuration at respectively the start and end of the guide path. Each right picture shows a possible camera path. In all experiments we use a maximal tracking angle of $\pi/2$; that is, the camera must stay behind the guide.

We use test case 1 to test the motion planner in an environment with a rather uniform obstacle distribution, and enough room for the camera to maneuver. The purpose of test case 2 is to determine the effect of the number of guide path segments on the planner's efficiency. The path is short, but consists of a large number of small segments, increasing the cost of the local planner. Case 3 tests the planner's capability to operate in cluttered environments with a challenging guide path. Below you find the results of the basic planner, the "pusher" planner and the "useful" planner equipped with the push sampling. Recall that the tracking distance is fixed at the most challenging value. The results, expressed in seconds, are averages over 250 runs.

	Test case 1	**Test case 2**	**Test case 3**
Basic planner	1.29	1.02	60+
Push planner	0.91	0.31	45.96
Push + Useful planner	0.22	0.02	0.60

The basic motion planner solves test case 1 and 2 in about one second. However the basic planner is not able to solve the complex test case 3 within a minute. To give an impression of the roadmap size: for test case 3, the roadmap on average consisted of about 60,000 edges and 1,600 nodes.

The "pusher" motion planner clearly performs better overall, although its performance in test case 3 is still not satisfying. For test case 3, the roadmap now consisted of about 43,000 edges and 1,400 nodes.

Adding the usefulness test solved each test case well within a second. The notion of usefulness enables the motion planner to quickly expand the

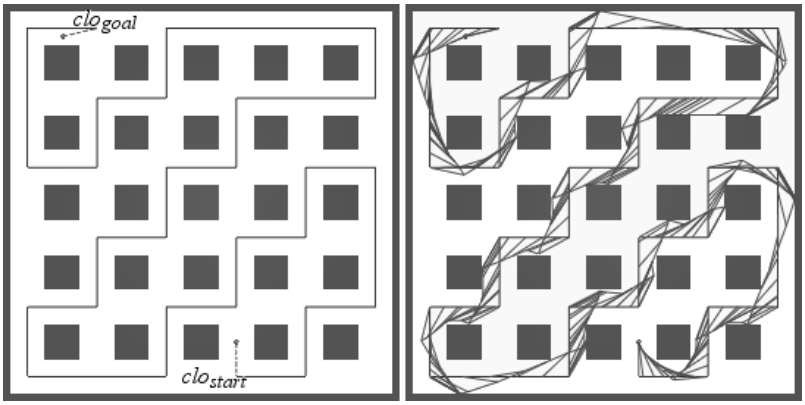

Figure 5: Test case 1

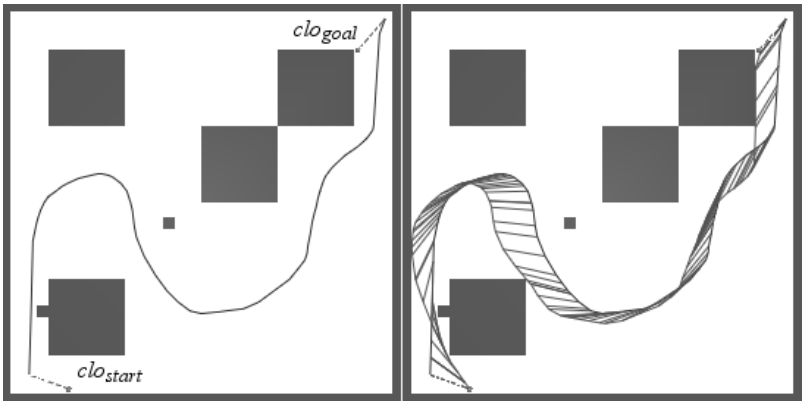

Figure 6: Test case 2

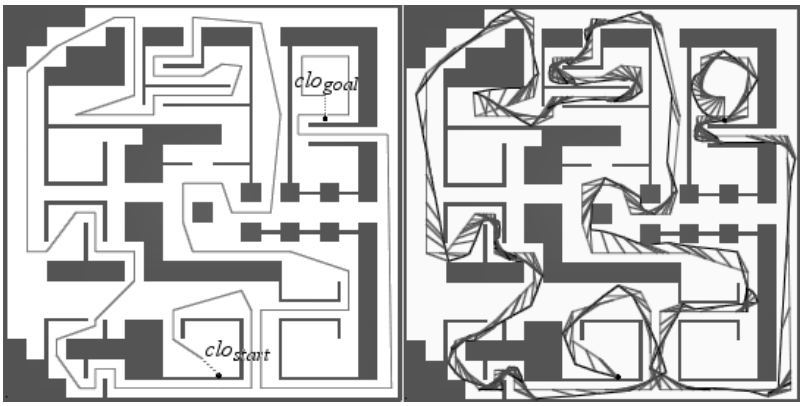

Figure 7: Test case 3

roadmap in uncluttered areas of the $\mathcal{CLO}$-space; and then focus on the cluttered areas of the $\mathcal{CLO}$-space, while discarding the already covered easy areas. The roadmap of test case 3 now consists on average of only about 600 edges and 400 nodes.

4.2 Neighbor Set Distance

There is one parameter to fill in; the neighbor set distance (nsd). For practical use it is important that the system determines this value automatically. This subsection describes a technique we refer to as NSD_{auto}.

Let us first define a few roadmap properties. The percentage of all local paths that is tested occlusion free, is referred to as the *success ratio*. Furthermore, the percentage of the length offset interval covered by the roadmap G is denoted as the *coverage ratio*.

NSD_{auto} consists of two phases. In the first phase, the learning phase, the roadmap provides too little and too unreliable information to base nsd on. Hence in the learning phase nsd is a constant value. Experiments in a variety of scenes show that the constant nsd is rather scene independent; somewhere in the range from 0.05 to 0.10 is optimal. The NSD_{auto} goes into the next phase, when the coverage ratio becomes larger than the success ratio. This transition assures on the one hand that the motion planner "learns" enough about the environment, and on the other that the motion planner's efficiency does not decrease because of the relatively large constant nsd. During the second phase, we base nsd on the average length of the last c free local paths. Experiments show that the results are optimal when c is between about 25 and 100. For test cases of average complexity, NSD_{auto} spends most of its time in the second phase.

The NSD_{auto} as described above causes two undesired side effects, which result in a decrease of the motion planner's performance. Both problems are caused by the decrease of nsd over time and can easily be solved. One problem occurs when the neighbor set of a new sample q_{new} is empty. Figure 8 illustrates the problem: The neighbor set N of the newly generated sample q is empty; thus the motion planner would mark q as useful, and add q to the roadmap. However the addition of q is not useful, since the graph already spans the $\mathcal{CLO}$-space covered by the neighbor set N. Note that this situation does not occur when we use a fixed nsd, since in that case either node v or w would be in N. To eliminate this problem, the motion planner determines whether the graph already covers the neighbor set length offset interval. If so, then the addition of q_{new} is considered not useful.

The second problem occurs when nsd has become small, while a relatively long local path is needed to bridge a gap in the roadmap. We solve this problem by setting a lower boundary for nsd; which we base on the tracking distance.

Experiments show that the motion planner performs as well with NSD_{auto}

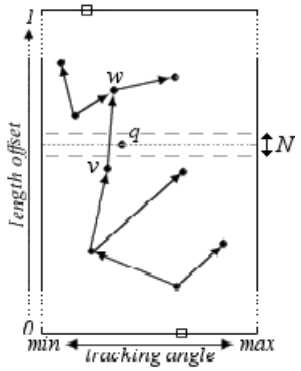

Figure 8: Illustration of problem occurring because of decreasing nsd

as when the neighbor set distance is manually optimized. The results in the previous section were obtained using NSD_{auto}.

5 Smoothing

Path quality indicates how pleasant the camera images are to watch for a user. The motion planner as described so far disregards this aspect. See figure 9(a). Experiments show that the path quality is inversely related to the total tracking angle change. Hence to improve the path quality, the motion planner has to minimize the total tracking angle change. This smoothing process consists of two phases.

The first phase is active while the motion planner generates the initial camera path: The weight of each roadmap edge e is the tracking angle difference between e's begin and end configuration. When q_{start} and q_{goal} are graph connected, the path with the smallest total weight is taken as initial solution $P_{solution}$.

The second phase applies a standard smoothing approach [24]. The motion planner repeatedly replaces random path segments of $P_{solution}$ with new path segments of higher quality using the local planner; and in doing so increases the overall quality of $P_{solution}$.

The longer this is repeated, the smoother $P_{solution}$ becomes. Figure 9(b) illustrates a smoothed path for the example test case. Without smoothing the motion planner generates $P_{solution}$ for the example test case on average in 0.1 second. The (close to) optimally smoothed path as illustrated in figure 9 is on average generated in 0.5 second. Experiments show however that optimal smoothing is not necessary to ensure a satisfying path quality: the motion planner generates a satisfying smooth $P_{solution}$ in on average 0.2 second. The three test cases described in the previous section are all solved and smoothed within a second. We expect that the smoothing phase can easily be improved to reduce its execution time even further.

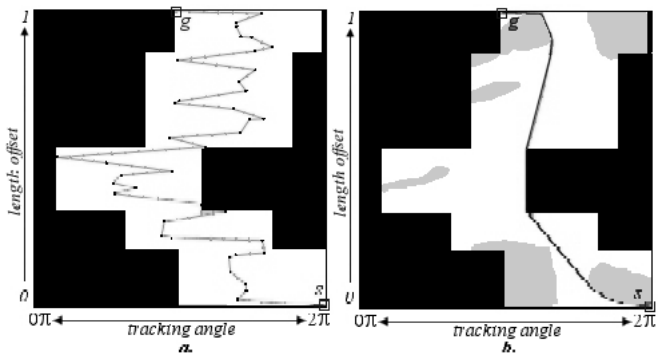

Figure 9: Figure (**a**): A typical example of an unsmooth camera path in the configuration space. The resulting camera images are unpleasant to watch, because the camera changes its tracking angle frequently. Figure (**b**): The white area is an approximation of $\mathcal{CLO}_f$. The path is an example of a smoothed path.

6 3D Motion Planner

We so far discussed camera motion generation in a two-dimensional workspace. A virtual environment however is three-dimensional and the motions of the guide and camera are not always reducible to a two-dimensional plane. The motion planner described in the previous sections though is easily extendable to operate efficiently in a three-dimensional workspace. We briefly describe the required changes here.

The guide path remains a map from length offset to guide configuration, where the guide configurations are now described by six parameters. Recall that the camera position is specified relative to a guide configuration. In the two-dimensional workspace we needed the tracking distance and the tracking angle to describe this relative position. In the three-dimensional situation we add a second tracking angle to describe the relative vertical angle. The camera orientation is again discarded: the camera will always look straight at the guide; furthermore, the camera's up-vector is kept equal to the guide's up-vector. The corresponding $\mathcal{CLO}$-space is a three-dimensional space spanned by both tracking angles and the length offset. The motion planner works without adaptation on the expanded $\mathcal{CLO}$-space. We implemented and tested the three-dimensional motion planner. The "push" and "usefulness" techniques again result in a drastic performance improvement.

Figure 10 shows a typical three-dimensional test case with open spaces, narrow passages and height variations. As for the two-dimensional test cases, the motion planner generates a smooth camera path in less than a second. The resulting roadmap again is relatively small: on average it contains about 150 edges and 100 nodes.

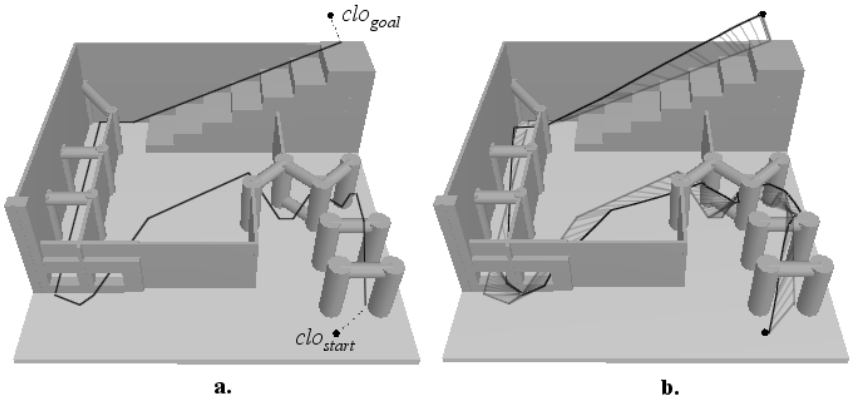

Figure 10: Figure (**a**) shows the guide path, and the start and goal configurations of the camera as q_{start} and q_{goal}. Figure (**b**) shows the guide path in gray and camera path in black. The short lines between the guide and camera path illustrate viewing directions of the camera.

7 Conclusion

We presented a technique to generate a camera motion such that the camera tracks a guide moving through a known environment along a known path. The motion planner applies a single-shot PRM approach to construct a graph in the free configuration space. A substantial performance gain is accomplished by a new technique, which determines whether a node or edge should be added to the roadmap based on its usefulness. The resulting camera path is smoothed to improve the path quality. Experiments show that the planner can operate in three-dimensional environments, and provide the user with a fast responds as is needed for interactive environments.

A number of extensions are possible. Firstly, when the guide is an object rather than a point, we might want to enforce the camera to see the complete guide (and even some part around it). This can easily be achieved by replacing the occlusion check by an intersection test with a visibility cone. Secondly, we can also easily deal with preferred tracking angles by changing the sampling distribution and the smoothing process. And lastly, preferred tracking distance can be achieved in a post processing stage. Finally, additional camera motion properties such as camera speed, camera acceleration, changes in tracking distance and changes in the camera orientation can be incorporated using techniques similar to those proposed in [24].

Acknowledgement This work was partly supported by the Netherlands organization for Scientific Research (NWO), and by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2001-39250 (MOVIE - Motion Planning in Virtual Environments).

References

[1] N. Amato, Y. Wu (1996). *"A randomized roadmap method for path and manipulation planning"*. Proc. IEEE Int. Conf. on Robotics and Automation, 1996, pp. 113-120.

[2] W.H. Bares, S. Thainimit, S. McDermott (2000). *"A model for constraint-based camera planning"*. Smart Graphics. Papers from the 2000 AAAI Spring Symposium (Stanford, March 20-22, 2000), pages 84-91, Menlo Park, 2000. AAAI Press.

[3] J. Barraquand, L. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, P. Raghavan (1997). *"A random sampling scheme for path planning"*. Int. Journal of Robotics Research 16 (1997), pp. 759-774.

[4] S. Beckhaus, F. Ritter, T. Strothotte (2000). *"Cubicalpath – Dynamic Potential Fields for Guided Exploration in Virtual Environments"*. roceedings Pacific Graphics 2000 (Hong Kong, China, October 2000), pages 387-395, Los Alamitos, 2000. IEEE Computer Society.

[5] G. van den Bergen (2003), *"Collision Detection in Interactive 3D Environments"*. Morgan Kaufmann 2003.

[6] D. Christianson, S. Anderson, L. He, D. Salesin, D. Weld, M. Cohen (1996). *"Declarative camera control for automatic cinematography"*. Thirteenth National Conference on Artificial Intelligence, 148-155.

[7] N. Courty, E. Marchand (2001). *"Computer animation: a new application for image-based visual servoing"*. IEEE Int. Conf. on Robotics and Automation, Volume 1, pages 223–228, Seoul, South Korea, Mai 2001.

[8] S.M. Drucker, D. Zeltzer (1995). *"CamDroid: A system for implementing intelligent camera control"*.P. Hanrahan and J.Winget, editors, 1995 Symposium on Interactive 3D Graphics, pages 139-144. ACM SIGGRAPH, Apr. 1995. ISBN 0-89791-736-7.

[9] M. Gleicher, A. Witken (1992). *"Through-the-Lens Camera Control"*. Computer Graphics, 26(2): 331-340.

[10] H.H. Gonzalez-Banos, C.Y. Lee, and J.-C. Latombe (2002). *"Real-Time Combinatorial Tracking of a Target Moving Unpredictably Among Obstacles"*. Proc. IEEE Int. Conf. on Robotics and Automation, 1683-1690, Washington D.C., May 2002.

[11] N. Halper, P. Olivier (2000). *"Camplan: A Camera Planning Agent"*. Smart Graphics. Papers from the 2000 AAAI Spring Symposium (Stanford, March 20-22, 2000), pages 92-100, Menlo Park, 2000. AAAI Press.

[12] N. Halper, R. Helbing, T.Strothotte (2001). *"Computer Games: A Camera Engine for Computer Games"*. Computer Graphics Forum Volume 20, Issue 3.

[13] A. Hanson, E. Wernert (1997). *"Constrained 3D Navigation with 2D Controllers"*. IEEE Visualisation, pages 175-182.

[14] Hanson, A., E. Wernert, S. Hughes (1999). *"Constrained navigation environments"*. Scientific Visualization: Dagstuhl '97 Proceedings, H. Hagen, G.M. Nielson, and F. Post, Editors. 1999, IEEE Computer Society Press. p. 95-104.

[15] L.-W He, M.F. Cohen, D.H. Salesin (1996). *"The Virtual Cinematographer: a Paradigm for Automatic Real-Time Camera Control and Directing"*. Proceedings of ACM SIGGRAPH'96, pp. 217-224, 1996.

[16] L. Kavraki, J.C. Latombe (1994). *"Randomized preprocessing of configuration space for fast path planning"*. Proc. IEEE Int. Conf. on Robotics and Automation, 1994, pp. 2138-2145.

[17] L. Kavraki, P. Švestka, J. Latombe, and M. Overmars. (1996). *"Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces"*. IEEE Transactions on Robotics and Automation, vol. 12, no. 4, pp. 566-580.

[18] S.M. LaValle, H.H. Gonzalez-Banos, C. Becker, J.-C. Latombe (1997). *"Motion Strategies for Maintaining Visibility of a Moving Target"*. Proceedings of the 1997 IEEE International Conference on Robotics and Automation, pages 731-736.

[19] S.LaValle, J. Kufner (2001). *"Rapidly-exploring random trees: Progress and prospects"*, in Algorithmic and Computational Robotics: New Directions (B.Donald, K.Lynch, and D.Rus, eds.), pp. 45-59, A.K. Peters, 2001.

[20] T.Y. Li (1999). *"An Auto-Navigation System for Virtual Factories"*. Proceedings of 1999 Automation Technology Conference, Chia-I, Taiwan.

[21] T.Y. Li, J.M. Lien, S.Y. Chiu, T.H. Yu (1999). *"Automatically Generating Virtual Guided Tours"*. Proceedings of the Computer Animation '99 Conference, Geneva, Switzerland, pp. 99-106, May 1999.

[22] J. Mackinlay, S. Card, G. Robertson (1990). *"Rapid controlled movement through a virtual 3d workspace"*. Computer Graphics (SIGGRAPH '90 Proceedings), pages 171-176, July 1990.

[23] E. Marchand, N. Courty (2000). *"Image-b̃ased virtual camera motion strategies"*. Graphics Interface Conference, GI2000, pages 69-76, 2000.

[24] D. Nieuwenhuisen, M.H. Overmars (2003). *"Motion Planning for Camera Movements in Virtual Environments"*. Proc. IEEE Int. Conf. on Robotics and Automation 2004, to appear.

[25] M.H. Overmars (1992). *"A random approach to motion planning"*. Technical Report RUU-CS-92- 32, Dept. Comput. Sci., Utrecht Univ., Utrecht, the Netherlands, 1992.

[26] C.B. Phillips, N.I. Badler, J. Granieri (1992). *"Automatic viewing control for 3D direct manipulation"*. M. Levoy and E. E. Catmull, editors, Proceedings of the 1992 Symposium on Interactive 3D Graphics, pages 71-74, Cambridge, MA, Mar.-Apr. 1992. ACM Press.