Creating High-quality Roadmaps for Motion Planning in Virtual Environments

Roland Geraerts and Mark H. Overmars
Institute of Information and Computing Sciences
Utrecht University
Utrecht, the Netherlands
Email: {roland,markov}@cs.uu.nl

***Abstract*— Our goal is to create roadmaps that are particularly suited for motion planning in virtual environments. We use our Reachability Roadmap Method to compute an initial, resolution complete roadmap. This roadmap is small which keeps query times and memory consumption low. However, for use in virtual environments, there are additional criteria that must be satisfied. In particular, we require that the roadmap contains useful cycles. These provide short paths and alternative routes which allow for variation in the routes a moving object can take. We will show how to incorporate such cycles. In addition, we provide high-clearance paths by retracting the edges of the roadmap to the medial axis. Since all operations are performed in a preprocessing phase, high-quality paths can be extracted in real-time as is required in interactive applications.**

I. Introduction

In many virtual environments, paths have to be planned for entities to traverse from a start to a goal position in the virtual world. A common way to plan the path is to use an A* algorithm on a (low-resolution) grid. This search algorithm is popular because it always finds a shortest path in the roadmap if one exists. However, as contemporary virtual worlds can be very large, storing the grid and running the algorithm may consume a huge amount of memory which is not always available, in particular on systems with constrained memory such as console systems. In addition, the algorithm may consume too much processor time, especially when many paths have to be planned simultaneously. This will lead to stalls in interactive applications. Paths resulting from A* algorithms tend to have little clearance and can be aesthetically unpleasant, so care must be taken to smooth them.

Another popular motion planning technique is the Probabilistic Roadmap Method (PRM) [1], [2]. The PRM consists of two phases: a construction and a query phase. In the construction phase, a roadmap (graph) is built, approximating the motions that can be made in the environment. In the query phase, the start and goal are connected to the graph. The path is obtained by a Dijkstra's shortest path algorithm. A drawback of the PRM is that a resulting roadmap often contains many redundant nodes and edges, in particular when the environment contains one or more narrow passages [3]. In addition, the roadmap may contain many short edges which complicates the smoothing phase that often follows a query phase [4].

Nieuwenhuisen *et al.* [4] improve a roadmap generated by the PRM such that it can be used for path planning in games. Their method guarantees that the paths are short, have enough clearance from the obstacles, and are C^1 continuous, leading to natural looking motions. Such a path can be retrieved almost instantaneously. Their method does not guarantee that a path can always be found (if one exists in the free space $\mathcal{C}_{\text{free}}$). In addition, the method is limited to two-dimensional problems.

Sometimes, roadmaps are created manually from which paths can be extracted. However, this can take many hours of precious time setting up and debugging the roadmap [5].

Our goal is to automatically create a roadmap for 2D and 3D (possibly large and complex) environments that can be used to guide the motions for entities in a virtual environment. By a careful integration of existing and new techniques, we aim at generating roadmaps with the following four properties:

1. The roadmap is *resolution complete*. This means that a valid query (which consists of a start and a goal configuration) can always be connected to the roadmap. If the start and goal belong to the same connected component of the free space, then a corresponding path can always be found (at a given resolution).
2. The roadmap is *small*. A small roadmap assures low query times and low memory consumption. A path that is extracted from a small roadmap will have reasonably long edges. These are easier to optimize. For example, Nieuwenhuisen *et al.* [4] add circular arcs to the roadmap to make the paths C^1 continuous, resulting in natural looking motions. In addition, when a roadmap must obey other criteria, a small roadmap eases manual tuning.
3. The roadmap contains *useful cycles*. These cycles provide short paths and alternative routes which allow for variation in the routes that entities take. Van den Berg *et al.* [6] exploit cycles in dynamic environments where additional obstacles might appear, and to avoid deadlock situations when multiple robots move in the same environment.
4. The roadmap provides *high-clearance paths*. By retracting the roadmap to the medial axis, paths with much clearance can be extracted in real-time. High-clearance paths work well with entities that have large widths, such as a wide formation of characters. In addition, they are perfectly suitable for guiding the motions of a group of entities [7] or for creating a useful backbone path for the animation of walking characters [8].

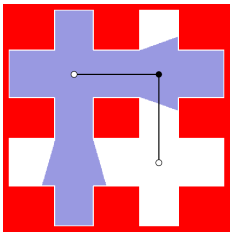


Fig. 1. The coverage and maximal connectivity criteria have been met. The reachability regions of the white nodes cover the complete free space and are connected via the black node.

The paper is organized as follows: In Section II, we discuss the Reachability Roadmap Method which creates the initial roadmap satisfying the first and second property. In Section III, we propose an algorithm that adds useful cycles to the roadmap. We meet the fourth property in Section IV which shows how to retract a roadmap to the medial axis. We perform experiments with 2D and 3D virtual environments in Section V and conclude in Section VI that our algorithm successfully creates roadmaps satisfying these four properties.

II. Reachability Roadmap Method

In [3], we introduced the Reachability Roadmap Method (RRM) which creates small roadmaps that are resolution complete. We proved that the RRM creates a roadmap graph G satisfying the following two criteria:

Definition 1 (coverage)**.** *Graph $G = (V, E)$ covers $\mathcal{C}_{\text{free}}$ when each configuration $c \in \mathcal{C}_{\text{free}}$ can be connected using the local planner to at least one node $\nu \in V$.*

Definition 2 (maximal connectivity)**.** *Graph G is maximally connected when for all nodes $\nu', \nu'' \in V$, if there exists a path in $\mathcal{C}_{\text{free}}$ between ν' and ν'', then there exists a path in G between ν' and ν''.*

Coverage ensures that every query (which consists of a start and goal configuration) can be directly connected to the roadmap, as is required to solve the problem. If there exists a path in the free configuration space ($\mathcal{C}_{\text{free}}$) between the start and goal configuration, then maximal connectivity ensures that a path between them can be found in the roadmap graph G.

The RRM discretizes the configuration space and computes a small number of *guards* that cover the complete free space. These guards are then connected via *connectors* to fulfill the maximal connectivity criterion. The resulting roadmap is then pruned to obtain an even smaller roadmap. Fig. 1 shows an environment whose free space is covered by two (white) nodes and is connected via one extra (black) node. The reachability region for the upper left node has been drawn. Each configuration in this region can be connected with a straight-line connection to the node, and a three-node graph suffices to solve any query in this environment.

III. Adding useful cycles

In [9], we discussed three methods for decreasing the path length. Although these methods can decrease the path length considerably, they usually do not remove the detours around obstacles. These detours can be avoided by adding cycles to the roadmap. Besides obtaining shorter paths, cycles provide alternative routes for an entity.

In the following subsections, we will show how to add useful cycles to the roadmap. Our strategy is partly based on the work of Nieuwenhuisen and Overmars [10] which adds *useful edges* to the roadmap. An edge is useful if it introduces a cycle that improves the roadmap according to some criterion. As we are working with small roadmaps, unfavorably placed queries can still lead to long paths. We will show that these can be avoided by adding *useful nodes* and their corresponding edges to the initial roadmap. The final roadmap will then be composed of all nodes from the initial roadmap, as well as the added useful nodes and useful edges between those nodes.

A. Useful edges

Nieuwenhuisen and Overmars [10] propose a technique that adds useful cycles to the roadmap. The goal is to add only those edges that have a high probability of introducing a path that cannot be continuously deformed into an existing path.

A *useful edge* is defined as follows:

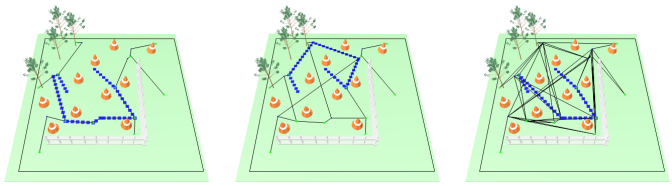
Definition 3 (Useful edge)**.** *Let ν be the node that corresponds to configuration c which has been added to the graph and V^n its set of neighbors. Let ν' be a node in V^n and $d(\nu, \nu')$ be the distance between ν and ν'. The graph distance between ν and ν' is $G(\nu, \nu')$ which is the length of the shortest path in the graph from ν to ν'. If there is no path from ν to ν', $G(\nu, \nu')$ is ∞. Then edge $\epsilon(\nu, \nu')$ is K-useful if*

$$K * d(\nu, \nu') < G(\nu, \nu').$$

This definition only adds an edge to the graph between ν and ν' if their graph distance improves by a factor K. A small value of K adds more edges than a large value of K. Fig. 2 shows that no cycles are added if K is set to ∞. If $K \leq 1$, then all collision-free edges (i.e. local paths) are allowed. The authors use a pruned version of Dijkstra's shortest path algorithm to efficiently determine whether a particular edge is useful. They also show, when time goes to infinity, that their approach will find a path with a length converging to $K * |\Pi|$, where $|\Pi|$ denotes the length of shortest possible path Π. Hence, the larger the number of nodes in the roadmap (and the smaller the value of K), the shorter the expected length of a path. As one of our criteria is obtaining a small roadmap, these two conflicting criteria (short path length and small roadmap) need to be balanced.

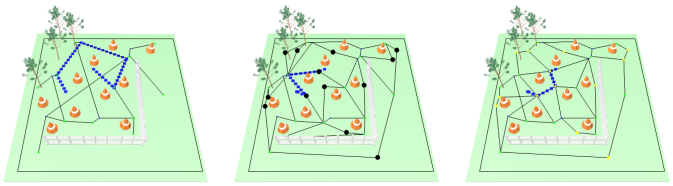
B. Adding useful nodes

In Fig. 2, unfavorably placed start and goal positions were added and connected to the roadmap. As few nodes were placed in the middle of the environment, such a small roadmap can yield long paths, making detours around the obstacles. We handle this problem by adding *useful nodes* (and useful edges) to the roadmap, while attempting to keep it small. Accordingly, we define a useful node as follows:



(a) $K = \infty$ (b) $K = 1.5$ (c) $K = 1$

Fig. 2. A roadmap that contains few nodes can lead to long paths for unfavorably placed queries. Even when parameter K is set to one, i.e. when all collision-free connections are added as edges to the roadmap, the extracted path can be long compared to the optimal path.



(a) Useful cycles: length=63.8 (b) Useful nodes: length=25.9 (c) Useful nodes, useful cycles, edges reconnected: length=18.3

Fig. 3. Path lengths of an unfavorably placed query. The factor K for adding useful cycles is 1.5.

Definition 4 (Useful node)**.** *Let $c \in \mathcal{C}_{\text{free}}$ be a configuration and ν be its representing node. Let $\{\nu', \nu''\}$ be its two closest neighbors in the graph to which a collision-free connection exists, i.e. edge $\epsilon(\nu, \nu') \in \mathcal{C}_{\text{free}}$ and edge $\epsilon(\nu, \nu'') \in \mathcal{C}_{\text{free}}$. Let Π be the shortest path in G between ν' and ν''. If no path exists, then $\Pi = \emptyset$. Then node ν is useful if:*

$$\exists \nu_i \in \Pi : \epsilon(\nu, \nu_i) \notin \mathcal{C}_{\text{free}}.$$

As one of our goals is to obtain high-clearance paths, we only select candidate useful nodes that lie on the medial axis. Definition 4 says that a node ν is useful if ν can be connected to two neighbors and the new cycle guides the entity around an obstacle, i.e. there must be at least one connection from node ν to the nodes describing the shortest path Π which causes the moving object to collide with an obstacle. We limit ourselves to checking connections between nodes because checking all connections from node ν to each configuration on path Π will consume too much time.

Algorithm 1 ADDUSEFULNODES(graph $G(V, E)$)

```
MA ← list of configurations, sampled on the medial axis
for all c ∈ MA do
    ν ← node that represents configuration c
    ν', ν'' ← the two closest neighbors of the input graph
    to ν for which edges ε(ν, ν') and ε(ν, ν'') ∈ C_free
    Π ← shortest path in G from ν' to ν''
    for all ν_i ∈ Π do
        if ε(ν, ν_i) ∉ C_free then
            V ← V ∪ ν
            E ← E ∪ ε(ν, ν')
            E ← E ∪ ε(ν, ν'')
            break
```

As a clarification, we apply Algorithm 1 on our running example. Fig. 3(b) shows the resulting roadmap. (The black discs represent the useful nodes.) It can be noticed that the useful nodes have a tendency to spread. This can be explained as follows. Suppose that a candidate node ν is very close to a previously added useful node ν' which has already been added and connected to the roadmap. Then it is likely that ν has the same two neighbors as ν' to which free connections exist. (Node ν will not be connected to ν' as useful nodes are only connected to nodes from the initial roadmap to keep the roadmap small.) As their edges will lie close together, it is unlikely that ν is part of a connection that collides with an obstacle. As a result, node ν is not labeled useful according to Definition 4.

C. Reconnecting the edges

The roadmap quality can be further improved by rearranging its edges. Our approach creates a graph G' that consists of all nodes from graph G. Then, for each pair of nodes, we check if the connection between them is collision-free. Such a connection is added as an edge to G' if the edge is K-useful. The approach is outlined in Algorithm 2. First, we create a priority queue (sorted on increasing edge length) and fill it with all collision-free connections between pairs of nodes from graph G. The improved graph G' will have the same nodes as graph G. The edges of G' consist of edges extracted from the queue if the two nodes do not belong to the same connected component or if the edge introduces a K-useful cycle. Due to the reconnection of the edges, it can occur that a (useful) node is no longer part of a cycle. If such a node has degree one, than this node is removed as our goal is to keep the roadmap small. Fig. 3(c) shows the roadmap whose edges have been reconnected.

Algorithm 2 RECONNECTEDGES(graph $G(V, E)$, factor K)

Output: graph $G'(V', E')$

```
PriorityQueue Q {sorted on increasing edge length}
for all pair of nodes ν', ν'' ∈ V : ν' ≠ ν'' do
    if edge ε(ν', ν'') ∈ C_free then Q.push(ε(ν', ν''))
V' ← V
E' ← ∅
while not Q.empty() do
    edge ε(ν', ν'') ← Q.top()
    Q.pop()
    if K * d(ν', ν'') < G(ν', ν'') then E' ← E' ∪ ε
Remove all useful nodes from V' with degree 1
```

Fig. 3 gives an indication of changes in path length of the unfavorably placed query in several phases of the running example. In each picture, the same initial roadmap was used (which was produced by the Reachability Roadmap Method). Fig. 3(a) shows the path for the roadmap to which useful cycles were added. This path is rather long. A shorter path was found

in (b), which shows the roadmap after adding useful nodes and their corresponding connections. Yet a shorter path was found after reconnecting the edges, which is shown in (c).

IV. Providing high-clearance paths

The fourth criterion which a roadmap must obey is that high-clearance paths can be obtained in real-time. A path has a high clearance if it follows the medial axis of the free configuration space. We use our retraction algorithm from [9] to retract the paths to the medial axis.

Rather than retracting paths, our goal now is to retract the entire roadmap to the medial axis. To ensure that the complete roadmap will be retracted to the medial axis, we require that its nodes initially lie on the medial axis. The Reachability Roadmap Method already satisfies this requirement. For each edge ϵ, let Π be the local path that corresponds to the edge. We apply the retraction algorithm to each local path Π.

The result of this approach can be viewed in Fig. 5. The input graph was created in three steps. A small covering roadmap was produced by the Reachability Roadmap Method. Then useful nodes and edges were added. These edges were then successfully retracted by the appropriate algorithm. Note that some retracted edges have overlapping parts. We do not attempt to merge them as this will increase the number of nodes in the roadmap.

V. Experiments

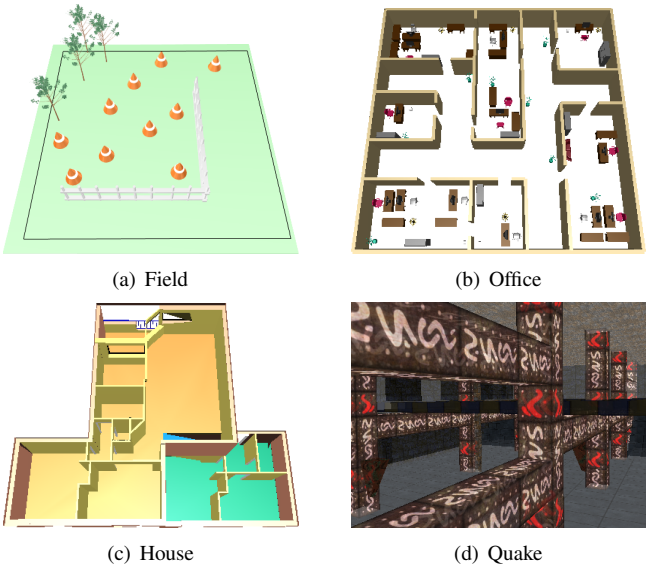
In this section, we test our approach on four virtual environments. In the first part of the experiments, we compare the roadmaps produced by the following three algorithms. The first algorithm, which we refer to as the *Grid Roadmap Method* (GRM), creates a grid. The GRM is constructed by placing a node on each corner of each free grid cell. Edges are added for each boundary and each diagonal of a free cell. This algorithm is often used in the game community. The second algorithm is the Reachability Roadmap Method (RRM) from [3]. (See this paper for a comparison between the RRM and the PRM.) The third algorithm (RRM*) uses the RRM as input and adds useful nodes and edges. Its edges are then rearranged.

In the second part of the experiments, we retract the edges of the roadmaps produced by the RRM* to the medial axis of the free space to obtain high-clearance paths. We refer to this combination of methods as the Retract RRM* (RRRM).

A. Experimental setup

We integrated the algorithms in a single motion planning system called SAMPLE (System for Advanced Motion PLanning Experiments), which we implemented in Visual C++ under Windows XP. SAMPLE automates conducting experiments, i.e. statistics are automatically generated and processed. All experiments were run on a 3 GHz Pentium 4 processor with 1 GB memory. We used Solid for collision checking [11].

We conduct experiments with the environments depicted in Fig. 4. Information on the environments and robots is listed in Table I. The environments have the following properties:



(a) Field (b) Office

(c) House (d) Quake

Fig. 4. The four test environments.

TABLE I

Information on the environments and robots.

	Bounding boxes			
	environment	**robot**	**grid resolution**	**# objects**
Field	47×47	1×1	94×94	16,000
Office	80×80	$1 \times 1 \times 4$	160×160	79,000
House	$57 \times 20 \times 40$	$3 \times 3 \times 3$	$57 \times 20 \times 40$	1,000
Quake	$130 \times 25 \times 80$	$1 \times 1 \times 1$	$130 \times 25 \times 80$	4,000

Field This small environment contains ten cones, two fences and four trees. These obstacles are cluttered in a large part of the environment. The other part is rather empty. There are many alternative routes. We will test whether our algorithm can capture most of them.

Office This large environment with more than 80 pieces of furniture (79,000 geometrical objects) has a rather non-uniform distribution. There are large open spaces and many narrow passages, requiring a large grid to capture the connectivity of $\mathcal{C}_{\text{free}}$. Also this environment contains many alternative routes. The results will show whether our algorithm can capture them. We also investigate how well our algorithm deals with large environments containing many obstacles.

House This environment has twelve rooms. There are few alternative routes from one room to another room. Hence, we expect that few cycles will be added to the reachability roadmap. Each edge in the roadmap will be retracted to the medial axis of the environment. We will investigate how much the clearance improves along the roadmap.

Quake This environment has been converted from a level from the game Quake. (See www.quake.com). There are many alternative routes. We will investigate how much the average path length decreases when we add useful cycles. Furthermore, we will test whether the clearance can be added successfully to roadmaps of problems involving three DOFs.

When we add cycles to a roadmap, we need a way to measure the improvement. For this purpose we define the *Shortest path factor* (SPF):

Definition 5 (Shortest path factor)**.** *Let $G' = (V', E')$ be the graph created by the* GRM*. Let $G = (V, E)$ be a graph for which $V \subseteq V'$. Furthermore, let Ψ be the set of shortest paths in G between each pair of nodes in V, n be the number of paths in Ψ and Ψ' be the set of shortest paths in G' between each pair of nodes in V. Finally, let $|\cdot|$ denote the length of a path. Then the* Shortest path factor *is defined as follows:*

$$\text{SPF} = \sum_{i=1}^{n} |\Psi_i| / \sum_{i=1}^{n} |\Psi'_i|.$$

This definition provides a factor that describes how much longer the expected length of a path (between two nodes of graph G) is compared to the optimal path length in the grid. If the factor equals one, then each extracted path will be the shortest one in the grid. The larger this factor, the larger the detour made by the moving object.

For all environments and techniques, we define 100 random queries and report how many seconds it takes to solve them. Regarding the Grid Roadmap Method, finding the closest free neighbor can be done in $O(1)$ time. Hence, we only report the time needed for running Dijkstra's shortest path algorithm. In contrast, the other two methods require finding the closest free neighbors *and* running Dijkstra's algorithm.

We also keep track of the sizes of the roadmaps, the construction times, the query times and clearance information (i.e. minimum, average and maximum clearance of configurations in the roadmap). The clearance is measured as the Euclidean distance between the pair of closest points on the moving object and obstacles.

B. Experimental results

In the following paragraphs, we will describe the results of adding useful cycles and nodes, and adding clearance to the roadmap. See Fig. 5 for visualizations of these results.

1) Adding useful cycles and nodes: For each environment, we created a roadmap by applying the Grid Roadmap Method (GRM), the Reachability Roadmap Method (RRM) and the modified RRM (RRM*) method. We set parameter K to 1.5. The results are stated in Table II and are discussed below:

Field Even for this relatively small environment, the GRM produced a huge roadmap. As a result, computing 100 random queries took 1.36 seconds. These running times may be acceptable for real-time behavior. An advantage of the method is that a shortest path in the grid will always be found, but as indicated above, the path lacks clearance as it runs very close to obstacles. The RRM created a small roadmap consisting of 29 nodes and 18 edges. As the graph was small, connecting a query to the roadmap and running Dijkstra's algorithm on average took 4.3 ms. Hence, this roadmap can be used in real-time situations. However, due to the small size of the roadmap, the off-center placement of the nodes in the environment, and the fact that the roadmap does not have cycles, the shortest path factor is rather high (1.57). This means that an extracted path between two nodes in the roadmap is on average 57% larger than the corresponding shortest path in the grid of the GRM roadmap. The RRM* added 14 useful nodes and 29 useful edges to the roadmap. As a result, the SPF decreased to 1.137 which means that an extracted path will be much shorter than an extracted path in the RRM roadmap. This did not have a negative impact on the extraction times of the queries.

Office Again, the GRM created a huge roadmap. Finding a shortest path between existing nodes in this roadmap took 34 ms on average. The RRM created a small roadmap. The RRM* added 15 nodes and 33 edges to this roadmap, improving the SPF with 53 percent points. Hence, an extracted path will make less detours. In addition, a close inspection of this roadmap in Fig. 5 shows that many alternative routes have been introduced. This did not result in longer query times. The RRM and RRM* methods function well in environments with a lot of detail and many obstacles as the three methods needed only a few seconds.

House A roadmap produced by the GRM for a 3D grid will inevitably become huge, even for a relatively small environment such as this one. Running a query on average took 135 ms, which is far too long in real-time situations. It was surprising to observe that the RRM only needed 34 nodes and 33 edges to get the free space covered and connected. Apparently, the nodes were properly placed in each room, as the RRM* did not add any useful node to the roadmap. Only one edge was added. Running a query took on average 8 ms. An extracted path will be short, because, on average, it will be only 23% larger than the corresponding optimal path in the grid.

Quake The roadmap created by the GRM became huge (> 1.5 million edges). As a result, extracting a query took 0.64 seconds on average. The RRM created a small roadmap. This sparse roadmap lead to a high SPF. By adding 61 nodes and 151 edges, the RRM* reduced this factor substantially. Extracting a query took on average 42 ms, which may be too high in a real-time situation. Connecting the query to the graph consumed relatively much time as many connections had to be checked for collisions.

In conclusion, roadmaps created by the GRM rapidly become huge, especially when 3D grids are required, resulting in query times that are too high for real-time performance. In practice, much smaller grids are used but this can be problematic when there are narrow passages. In contrast, the RRM creates small roadmaps that completely cover the free space and have low query times. However, the extracted paths can be long. The RRM* combines the advantages of GRM and RRM, i.e. reasonably short paths are produced while the extraction times are kept relatively low. The resulting paths were on average 14 to 23 percents larger than the optimal paths in a corresponding grid. Since the RRM* placed the nodes on the medial axis, a large clearance caused the nodes to lie far away from the obstacles, increasing the path length.

TABLE II

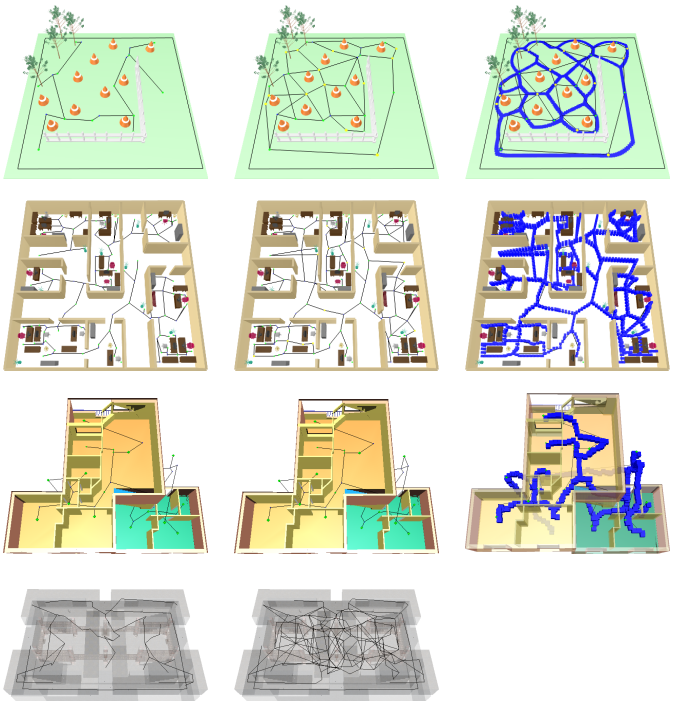
ROADMAP STATISTICS FOR THE FOUR ENVIRONMENTS. THREE METHODS WERE COMPARED. WE COLLECTED THE FOLLOWING STATISTICAL DATA: THE CONSTRUCTION TIME OF THE ROADMAP, ITS NUMBER OF NODES $|V|$ AND EDGES $|E|$. THEN WE MENTION THE SHORTEST PATH FACTOR (SPF) AND THE SUMMED RUNNING TIME OF 100 RANDOM QUERIES.

Field	Graph statistics			Path statistics	
	time (s)	$\|V\|$	$\|E\|$	**SPF**	**100 queries (s)**
GRM	0.88	7,350	27,741	1.000	1.36
RRM	0.75	29	18	1.570	0.43
RRM*	0.91	43	47	1.137	0.23

Office	Graph statistics			Path statistics	
	time (s)	$\|V\|$	$\|E\|$	**SPF**	**100 queries (s)**
GRM	0.89	16,917	62,297	1.000	3.42
RRM	1.10	154	147	1.812	0.38
RRM*	2.70	167	180	1.181	0.36

House	Graph statistics			Path statistics	
	time (s)	$\|V\|$	$\|E\|$	**SPF**	**100 queries (s)**
GRM	12.91	40,088	454,250	1.000	13.48
RRM	11.67	34	33	1.225	0.82
RRM*	18.68	34	34	1.224	0.82

Quake	Graph statistics			Path statistics	
	time (s)	$\|V\|$	$\|E\|$	**SPF**	**queries (s)**
GRM	210.16	134,492	1,511,241	1.000	63.54
RRM	306.44	71	65	2.068	2.71
RRM*	384.90	132	216	1.194	4.15



(a) RRM (b) RRM* (c) RRRM

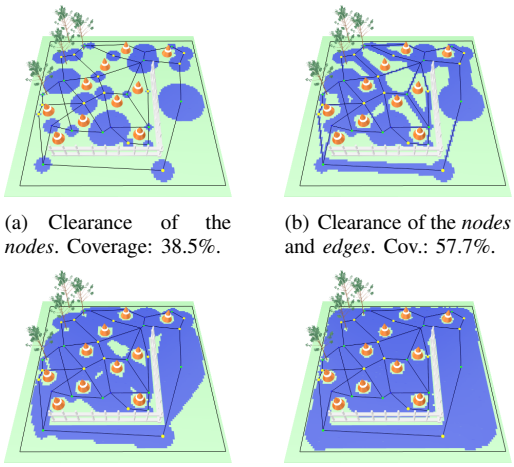
Fig. 5. The results for the Field, Office, House and Quake environments. The left column shows the initial roadmaps created by the Reachability Roadmap Method. The middle column shows the roadmaps to which useful nodes and cycles were added (RRM*). The right column shows the roadmaps to which clearance was added (RRRM). Due to space limitations, the retracted roadmap for the Quake environment could not be displayed effectively here.

2) Improving query performance: The largest portion of the query time for RRM and RRM* is occupied by checks for collision-free connections from the query position to the roadmap. We could remove the need for collision checks by using the clearance information of the roadmap. For example, let c be a configuration that has to be added as a node to the roadmap and c' be a configuration located on the roadmap. If distance $d(c, c') <$ Clearance(c'), then we know that c can be connected to c'. Since distance calculations between configurations are much faster than collision checks, using clearance information, gathered in the preprocessing of roadmaps, decreases query time significantly.

As an example, we apply this idea to the Field environment (see Fig. 6). First, we only use the clearance information of the nodes in the roadmap. Fig. 6(a) shows a collection of discs centered at the nodes. The radius of a disc equals the amount of clearance of the moving object represented by the node. A query does not collide with the obstacles when it lies in these discs. The total coverage of the discs is 38.5% of the free space. If we also use the minimum amount of clearance along the paths that correspond to the edges and place a disc along each configuration on the paths, the percentage improves to 57.7 which is shown in Fig. 6(b). The amount of coverage can be further improved by considering all configurations on all local paths of the roadmap. Fig. 6(c) shows that this improves the coverage to 80.1%. The highest amount of coverage can be obtained by using each configuration in a *retracted* roadmap. This leads to a coverage of 98.5% of the free space. In this case, the retracted roadmap consists of 870 configurations. Hence, at most 870 distance calculations are needed to check if a configuration can be connected to the roadmap. Checking whether one configuration can be connected takes 0.073 ms which makes the approach suitable for real-time planning. Even when the roadmap is larger (which is the case in the other environments), connecting a query by using clearance information turned out to be efficient, i.e. the times were below 1 ms. Since running Dijkstra's shortest path algorithm was far below 1 ms for the roadmaps produced by the RRM and RRM*, we conclude that this technique enables real-time extraction of queries in all tested environments.

A big advantage of using clearance information is that motion planning can be performed without expensive collision-checking operations. In particular, no geometrical data has to be stored for motion planning purposes which saves memory. Since we created small roadmaps, storing clearance information is feasible.

3) Adding clearance: As indicated above, high-clearance paths are often preferred. Such paths can be obtained by retracting each of the four roadmaps created by the RRM* to the medial axis. We refer to this method as the Retracted RRM* (RRRM). We compare the clearance of roadmaps produced



(a) Clearance of the *nodes*. Coverage: 38.5%.

(b) Clearance of the *nodes* and *edges*. Cov.: 57.7%.

(c) Clearance of each *configuration*. Cov.: 80.1%.

(d) Clearance of *retracted configs*. Cov: 98.5%.

Fig. 6. Using clearance information of the roadmap.

TABLE III

CLEARANCE AND TIME STATISTICS FOR RRM* AND RETRACTED RRM* (RRRM) ROADMAPS. STATISTICS CORRESPONDING TO THE RRRM ARE THE AVERAGES OVER 100 INDEPENDENT RUNS.

Field	Clearance			Time
	min	**avg**	**max**	**s**
RRM*	0.03	2.71	6.44	
RRRM	0.34	3.08	6.46	24

Office	Clearance			Time
	min	**avg**	**max**	**s**
RRM*	0.00	1.60	6.82	
RRRM	0.01	1.77	7.53	320

House	Clearance			Time
	min	**avg**	**max**	**s**
RRM*	0.00	2.17	5.64	
RRRM	0.13	3.33	10.41	49

Quake	Clearance			Time
	min	**avg**	**max**	**s**
RRM*	0.00	2.90	9.45	
RRRM	0.05	3.28	9.75	343

by the RRM* and RRRM. Table III shows the corresponding statistics. In all retracted roadmaps, the (minimum, maximum and average) clearance was improved. The times needed for the retraction were reasonably fast for the Field and House environments. The roadmaps for Office and Quake environments were retracted within six minutes. The main reason for this difference is that the latter two roadmaps are considerably larger than the other two. We expect that the running times of the retraction algorithms can be dramatically decreased by incorporating learning techniques.

VI. CONCLUSION

We presented a method that automatically computes a roadmap for 2D and 3D virtual environments. The method creates roadmaps having advantages over roadmaps created with e.g. a PRM or GRM. We used the Reachability Roadmap Method to generate resolution complete small roadmaps. As paths, extracted from this roadmap, can make long detours around obstacles, we provided a method that adds useful cycles to the roadmap. Experiments showed that alternative and reasonably short paths can be extracted from the enhanced roadmap. The query times on this roadmap are low which enables real-time extraction of paths. Another criterion the roadmap should satisfy is that high-clearance paths can be extracted without extra computation time in the query phase. This criterion was met by retracting the edges of the roadmap to the medial axis of the free space. While this may take a reasonable amount of time, we believe that the running times of the retraction algorithm can be improved dramatically by incorporating learning techniques.

In future work, the method could be extended to incorporate extra constraints that level designers put on the roadmaps. For example, one could incorporate non-holonomic constraints, walkable surfaces, take special care of staircases and incorporate tactical information in the roadmap.

ACKNOWLEDGMENT

Part of this research has been funded by the Dutch BSIK/BRICKS Project.

REFERENCES

[1] L. Kavraki, P. Švestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 566–580, 1996.

[2] J. Barraquand, L. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan, "A random sampling scheme for path planning," *International Journal of Robotics Research*, vol. 16, pp. 759–744, 1997.

[3] R. Geraerts and M. Overmars, "Creating small roadmaps for solving motion planning problems," in *IEEE International Conference on Methods and Models in Automation and Robotics*, 2005, pp. 531–536.

[4] D. Nieuwenhuisen, A. Kamphuis, M. Mooijekind, and M. Overmars, "Automatic construction of roadmaps for path planning in games," in *International Conference on Computer Games: Artificial Intelligence, Design and Education*, 2004, pp. 285–292.

[5] G. Alt, "The suffering: A game AI case study," in *Challenges in Game AI workshop, Nineteenth national conference on Artificial Intelligence*, 2004, pp. 134–138.

[6] J. Berg, D. Nieuwenhuisen, L. Jaillet, and M. Overmars, "Creating robust roadmaps for motion planning in changing environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 2515–2521.

[7] A. Kamphuis and M. Overmars, "Finding paths for coherent groups using clearance," in *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, 2004, pp. 19–28.

[8] A. Kamphuis, J. Pettre, M. Overmars, and J.-P. Laumond, "Path finding for the animation of walking characters," in *Poster proceedings of Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, 2005, pp. 8–9.

[9] R. Geraerts and M. Overmars, "Clearance based path optimization for motion planning," in *IEEE International Conference on Robotics and Automation*, 2004, pp. 2386–2392.

[10] D. Nieuwenhuisen and M. Overmars, "Useful cycles in probabilistic roadmap graphs," in *IEEE International Conference on Robotics and Automation*, 2004, pp. 446–452.

[11] G. Bergen, *Collision Detection in Interactive 3D Environments*. Morgan Kaufmann, 2003.