Path planning in combined 3D grid and graph environment

Petr Brož[1]

Department of Computer Science and Engineering
University of West Bohemia
Pilsen, Czech Republic

Abstract

In research literature and many scientific disciplines, solution to the common problem in path planning for an autonomous robot has been extensively developed. Almost all explored techniques assume the robot has complete and detailed overview of the environment he is moving in. In addition to, many methods work over the graph representation of this environment which can be very difficult to construct or obtain in the real applications. This paper introduces a hybrid technique combining graph and grid representations of an examined space and capable of planning paths in known, partially known, unknown and dynamic environment at the price of the pseudo optimality of results.

1 Introduction

General problem of finding and planning of an optimal path is a highly explored topic in several scientific areas. There are many approaches and techniques for solving this task. They are in most cases based either on graph or grid representation of the examined environment. In other words, some algorithms for path planning demand graph-like geometric definition of the processed scene (e.g., definition of all obstacles and forbidden areas) and other algorithms assume the discrete representation of the surrounding environment is provided. Most of these techniques generally do not distinguish the dimension of examined scene - they can be used either in 2D or 3D applications without any difficult modifications. Graph based approaches usually derive special structures from the provided environment description and work with them whereas the raster based approaches usually do not need such pre-processing and search the path directly in the provided grid.

Both ways of environment representation have crucial and radical disadvantages. Graph representation of a real environment is rarely available and its construction is - if possible at all - very complicated and difficult.

On the other hand, discrete representation of the examined space is much easier accessible and measurable but the algorithm itself is in most cases (due to the amount of raster elements to inspect) very time-consuming. In addition, almost all methods for path finding and planning need either well-known or static environment which is not always available, either.

A great improvement for this type of applications can be achieved with the combination of discrete and graph environment approaches. Such a technique could use adaptive spatial structure as a graph with vertices and edges evaluated according to the values from the provided grid. Then it would be able to discover pseudo optimal path (optimal among all available transitions in the graph) and, for example, continuously adapt this spatial structure to the actual state of environment and other dynamic influences.

In this paper, we propose a possibility for path planning over the combined environment representation which eliminates (or at least reduces) the disadvantages of mentioned conventional approaches at the price of the pseudo optimality of results. The content of the paper is as follows. Section 2 explains state of the art together with the best known techniques. Section 3 describes the proposed path planning model and in the section 4, our actual solution and implementation is outlined. Section 5 shows the results gained by our solution and in section 6, the future work of the proposed path planning approach is presented.

2 State of the art

Path planning denotes a basic problem of finding an optimal path between two specified spots in an abstract environment representation. In this context, optimal path means a path satisfying one or more given objectives (the shortest, the cheapest or the fastest path, etc.). Environment can be represented in a variety of ways but the path planning algorithms are focusing mainly on evaluated graphs and grids. There are many ways these environments can be differentiated (dynamic/static or known/unknown environments, etc.) which implies a similar distinction of path planning techniques according to the types of environment they are able to work with.

[1]pebro@students.zcu.cz

First, let us introduce the approaches based on the graph representation of the surrounding environment. **Visibility graph** technique [Her87] extends the basic provided graph with edges connecting vertices that can "see" each other whereas the source and destination position is treated as an obstacle, too. New edges (together with edges defining sides of each obstacle) then represent possible transitions and through them, the optimal path can be found. Example of such pre-processing in 2D application can be seen in Figure 1: edges of all obstacles (bricks pattern filling), starting and ending position (points labelled S and E) are connected according to their mutual visibility and over possible transitions (thin lines and obstacles sides), the optimal path (dashed lines) is selected.

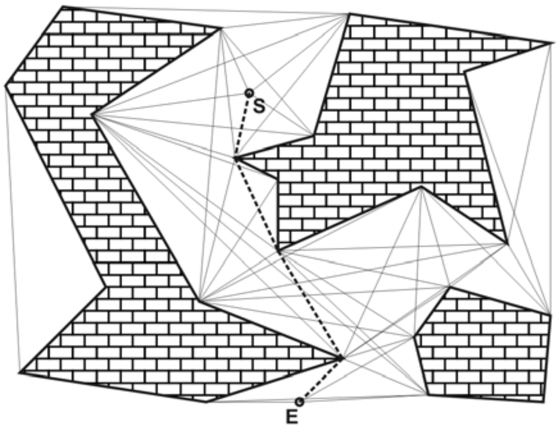

Figure 1: An example of scene processing with the "Visibility graph" technique

Minkowski sum [Ram96] is a similar approach that (unlike the previous method) considers the shape of passing object and "inflates" borders of obstacles so that the collision-free path can be solved. Example of such pre-processing is presented in Figure 2: the same obstacles as in Figure 1 are inflated with the radius of obstacle (gray areas) and the collision-free path (dashed lines) between starting and ending position (spheres labelled S and E) is selected. With the special structure prepared, both approaches can use **Dijkstra's algorithm** [DPV04] or similar to find the appropriate path.

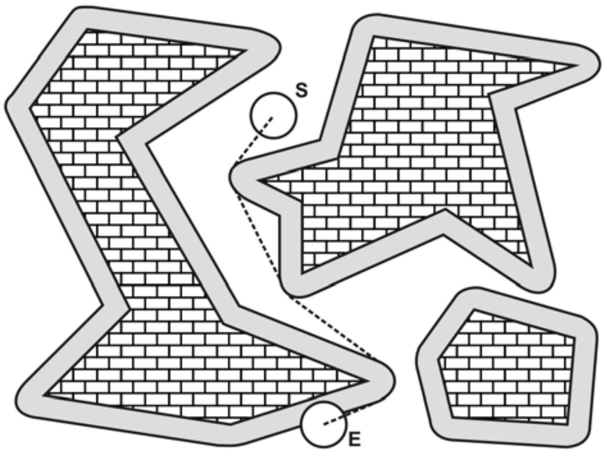

Figure 2: An example of path planning with the "Minkowski sum" method

Second, we are providing insight into the techniques based on the grid representation. Such a grid can be precomputed (if not provided) or modified at the beginning of the algorithm. In reference to the modification of the explored grid, a **potential field** model [War90] can be used for filling the grid with discrete values of a specific potential field created by all obstacles – passing through the grid elements with the lowest potential values then ensures finding the path with the maximal clearance among all obstacles. Most known techniques for searching itself are for example **A*** (for well-known environment; [Bat04]) and **D*** (for unknown, partially known or changing environment; [Ste94]) algorithms. Figure 3 documents the manner of such path finding in the grid: obstacles from Figures 1 and 2 are now splitted into the grid and in this grid, optimal path between starting and ending position (cells labelled S and E) over the grid cells is illustrated.

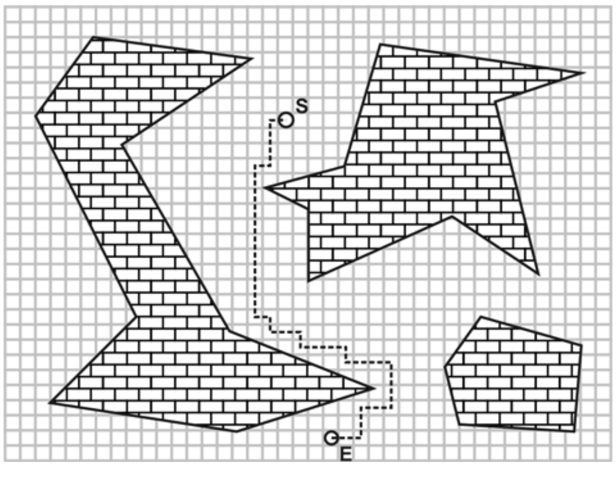

Figure 3: An example of the raster based path planning

3 The proposed solution

To provide a suitable method for the applications where the mentioned techniques fail, we are focussing on a general path planning technique that works in the known, partially known or unknown discrete environment and is designed for the virtual reality with the support of the exploring avatars.

In the proposed solution, we come out of a general idea of a fictive terrain exploration with the help of autonomous robots that are controlled from a specific kind of headquarters (HQ). These robots (also called scouts or agents) are equipped with specific sensors (dependent on the type of the application) and explore certain locations of the examined terrain according to the orders from HQ. Such headquarters keep specific „paper maps" to sketch in the discovered obstacles and other threats which are then periodically complemented and updated with actual values measured by the scouts. Agents are then guided to the unexplored locations or to the important locations according to the actual state of these maps. After certain time, the static obstacles are fully mapped throughout the explored space, the safest paths (in term of the maximal clearance among all obstacles) are known and the scouts are then guided only to locations with a suspicion of possible threats. Figure 4 represents an example of such environment exploration in 2D application: 4 agents in the terrain collect and send the information about the obstacles (bricks pattern filling) and specific kinds of threats (angry face) to the headquarters and there, the measured values are logged into the obstacles map (impassable areas) and into the threats map (a potential field of discovered threat).

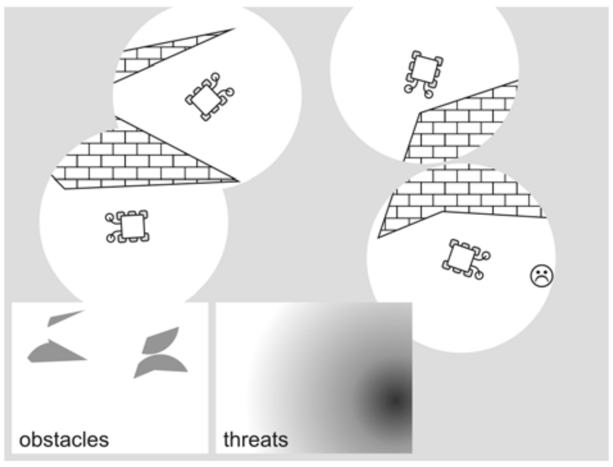

Figure 4: Preview of 2D terrain sensor-based exploration with the autonomous robots

Following the mentioned idea of the sensor based terrain exploration with the autonomous agents, we advance in the development of a general model for the real-time and adaptive path planning that was pioneered by R. A. Apu in [AG05]. The proposed model can be used for both 2D and 3D applications (the only difference lies in the undermentioned adaptive graph-like structures) and works in a complex and dynamic environment which is assumed to be provided in the raster representation and can be well known, partially known or even unknown. The described path planning system is based on three main headstones:

- A graph-like spatial structure (hereafter referred to as a **mesh**) that adapts itself to the examined environment and defines all available positions and crossings with its vertices and edges.

- A grid structure for discrete representation of certain environment hazards (hereafter referred to as a **map**), e.g., proximity to an obstacle or dynamic threats.

- An autonomous AI entity (hereafter referred to as an **agent**) for the real-time space exploration and influencing the mesh adaptation with its behaviour.

The main approach uses two separate maps of the same size for the environment description. The first one, called **obstacles map**, represents danger weights as proximities to the nearest obstacle in the mapped space and the second one, called **threats map**, represents potential fields of all located and observed threats in the space. In the following, the algorithm keeps a mesh that is „widespread over each map" and defines all available paths the agents can travel during their exploration. This mesh continuously copes with the changes in both maps and with behaviour of all agents. Such an adaptation is achieved by refinement of the mesh in the places with higher error values (calculated from the obstacles map and threats map) and by merging of the mesh in the least visited and unimportant places.

The whole algorithm is based on real-time development of the adaptive mesh in particular iterations. According to the recorded values in the maps, mesh structure is refined in the locations with a higher importance (the darker locations in the obstacles map and the threats map in the Figure 4) and it is merged in the places with a lower importance (in the least visited graph vertices). In the proposed path planning system, the adaptive mesh is used only to define the available waypoints and transitions for the movement and navigation of the agents, not for visualization. Therefore, T-vertices in the mesh do not bring any problems typical for them in the visualisation of meshes (they may cause creases in the model). Foldovers in the mesh are not possible in our case as vertices are not moved, just refined.

In the mentioned fictive application, continuous prospecting of the environment was a task of the robots but in our approach and demonstrating application, we assume the obstacles in the environment are completely explored - the obstacles map is filled with weights at the beginning of the algorithm with an IDT (Image Distance Transform) technique based on the Voronoi diagrams [Rou98]. Concretely, the elements of the obstacles map are evaluated according to their proximity to the nearest obstacle with the real value from 0 (maximal proximity) to 1 (minimal proximity or the obstacle itself). The elements of the threats map are then evaluated in a similar manner during the mesh adaptation.

One iteration of the mesh adaptation in the fictive application consists of the following general steps (similar as in [AG05]):

1. *Maps completion and updating*
   The current sensor readings are evaluated in the close neighbourhood of each agent and the corresponding map elements are updated or eventually complemented with the measured values.

2. *Influence depletion and replenishment*
   An importance of the recorded values (so called **influence**) of each vertex in the adaptive mesh is partially depleted and then again partially replenished according to the count and distance of the agents near this vertex. The more agents are in the proximity of the vertex, the bigger is the amount of the influence replenishment.

3. *Error function evaluation and refinement*
   The specific error function with the values from the obstacles map, threats map and influences is evaluated for each block (in [AG05], the blocks are called **engrams** in a specific spatial structure ASM – Adaptive spatial mesh) of the adaptive mesh and according to the result, the blocks are merged, splitted or left. Figure 5 shows a single stage of the adaptive mesh for the 2D scene presented in the Figure 4: the mesh is refined in important regions (above the obstacles and nearby the threat) and coarsened in less important or unexplored regions.

4. *Orders execution*
   Each agent executes its orders – he finds an optimal path to the goal position with the provided cost function or follows already computed waypoints (if the path cannot be travelled due to the refinements of the mesh, the path is recomputed to the last waypoint).

5. *Exploration*
   If all goals are reached, agents ensure an exploration of unvisited locations in the examined space – they automatically plan the path to the vertices with no values recorded.

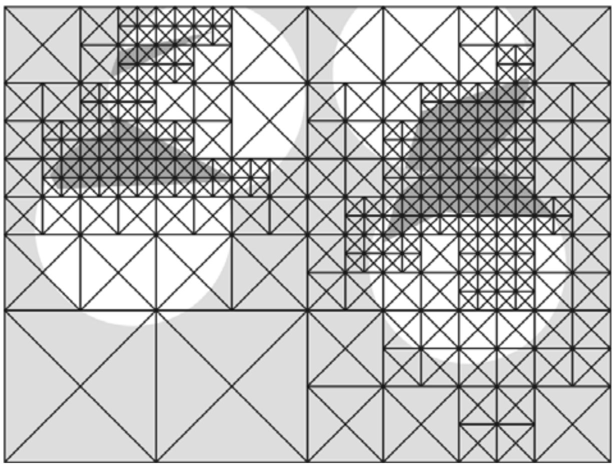
Figure 5: Example of the stage of the adaptive mesh for the same type of the explored terrain

With this approach, after a certain time, the mesh is fully adapted to the static obstacles and copes only with the dynamic influences – threats. A pseudo optimal path for the user can then be computed using Dijkstra's algorithm with the cost function similar to the function used by robots during their exploration in [AG05].

4 Our solution & implementation

This section provides an overview of our implementation of the path planning model and closely describes the implementation details. Therefore, the readers interested in the algorithm only can skip it. At this moment, our solution does not fulfil the first mentioned requirement – use of an adaptive spatial structure for the graph part – and so it is degraded to the basic type of raster-based path planning methods. Implementation fundamentals for this structure have been prepared and we will mske this generalization in the near future.

The main implementation of path planner is realized in C# language and the whole proposal is designed for providing high-level modularity – for each structure required by the proposed algorithm an interface is prepared. Each interface defines basic operations the concrete implemented structure must provide. Figure 6 shows basic elements of scene mapping part: **IRasterMap** interface must be implemented by every mapping structure used in the path planning algorithm. In compliance with the interface definition, such mapping structure must be able to provide weight of mapped space in a certain area or position. Classes **ThreatsMap** and **ObstaclesMap** implement this interface while work in different way. **ThreatsMap** keeps only a list of threats (instances of class that implements **IThreat** interface) and **ObstaclesMap** class keeps 3D array for whole mapped space.

Basic elements of adaptive mesh part are shown in Figure 7: Generic class **Mesh** implements **IGraph** interface (presented in Figure 8) and so provides basic operations for passing the graph such as passing all vertices or passing the descendants of the specified vertex. **Engram** class is for internal use of the **Mesh** class and the delegate labelled **BlockWeight** defines the only way for adaptation of the mesh - **BlockWeight** is a specific kind of a safe pointer to the function and requires method that is able to compute weight of certain area of mapped space. Genericity of the **Mesh** class ensures that the class works with any type of vertices. Only condition for each class of vertex is implementation of the **IVertex** interface presented in Figure 8.

Figure 8 documents top-level elements of proposed path planning algorithm. Main class **Dispatcher** requires above all instance of adaptive mesh class implementing the generic interface **IGraph** and list of instances for mapping the examined space (instances of a class implementing the **IRasterMap** interface).

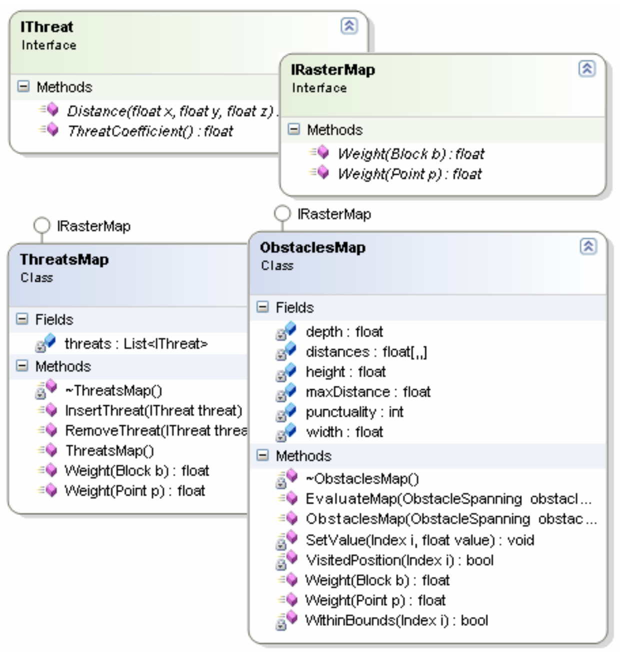

Figure 6: Class diagram of interfaces and classes in the mapping part of path planner

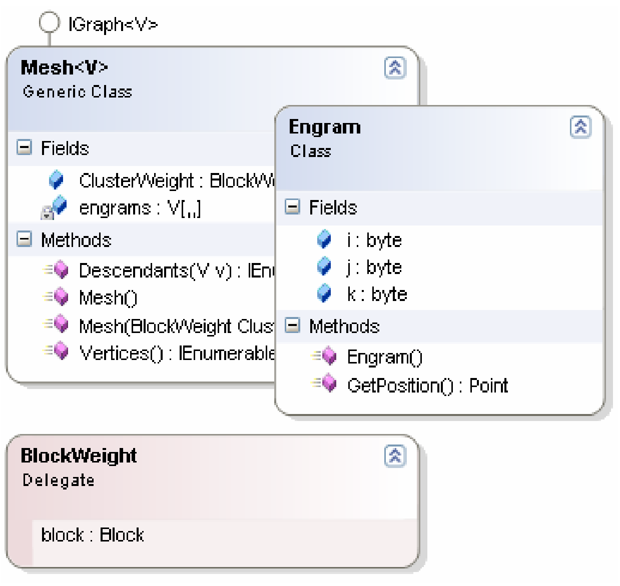

Figure 7: Class diagram of interfaces and classes in the part for adaptive mesh

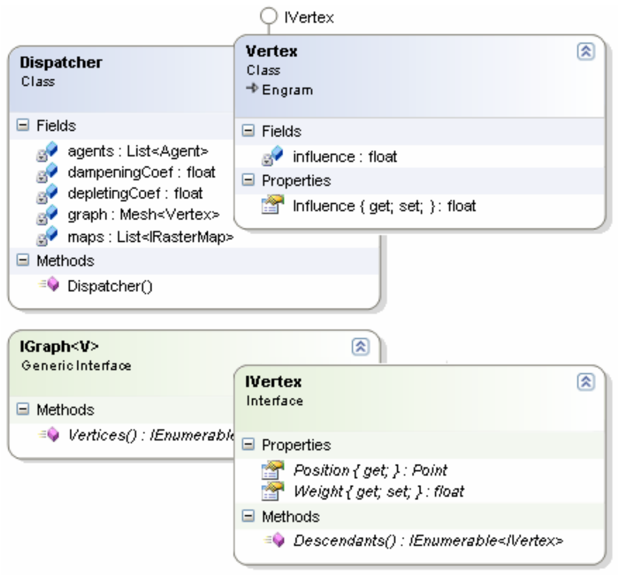

Figure 8: Class diagram of interfaces and classes in the main path planner

The proposed model provides solution for applications in known, partially known or unknown discretized environment at the price of the pseudo optimality of the final path. It comes to this, that this consequent path is optimal “only” with respect to the adaptive mesh.

5 Experiments & results

To survey our current solution, we have prepared a test application that creates a space with obstacles and uses the proposed technique for finding the path between two constant positions. Figure 9 shows the discovered pseudo optimal path in the space with the obstacles in the form of spheres with different radii. In the zoom, the same scene snapshot with weight markers is presented (lighter markers represent a low level of danger and darker markers represent a high level of danger).

The obstacles map is precomputed at the beginning of the algorithm and so the rank of this map (a count of the grid elements in each dimension) does not affect the time needed for the path planning itself. On the other hand, rank of (for the present) non-adaptive mesh has a great impact on this time. Functional dependence of the elapsed time on the mesh rank is evaluated and presented in the figure 10. The considered example finds a path in the same scene as in Figure 9 with obstacles map rank equal to 64. The computer configuration is disposed with AMD Athlon XP 1.67 GHz and 512 MB DDR RAM.

Figure 11 demonstrates using of the proposed path planning system in 2D applications and indicates the dependence of the path optimality on the rank of the mesh. Optimality of the path increases with the punctuality of the mesh (Figure 11 presents examples for the ranks 24, 36 and 48). Increase of the rank implicates the growth of the memory usage.

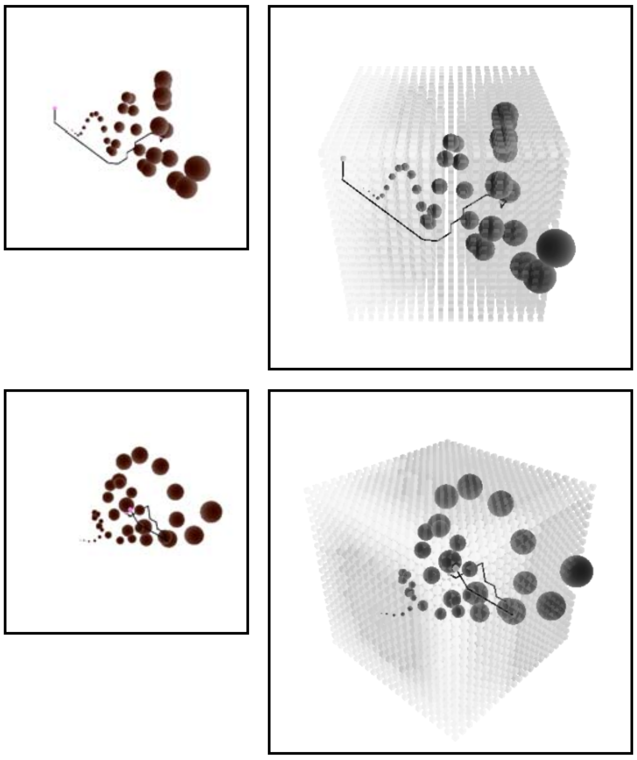

Figure 9: Mapped space with the obstacles and found path (in the zoomed snapshot with weight markers)

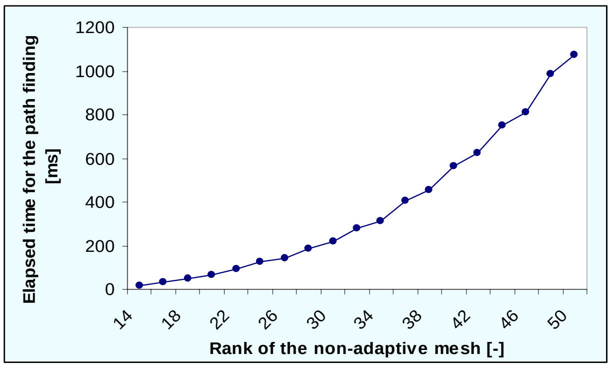

Figure 10: Elapsed time for the path finding dependent on the rank of the used non-adaptive mesh

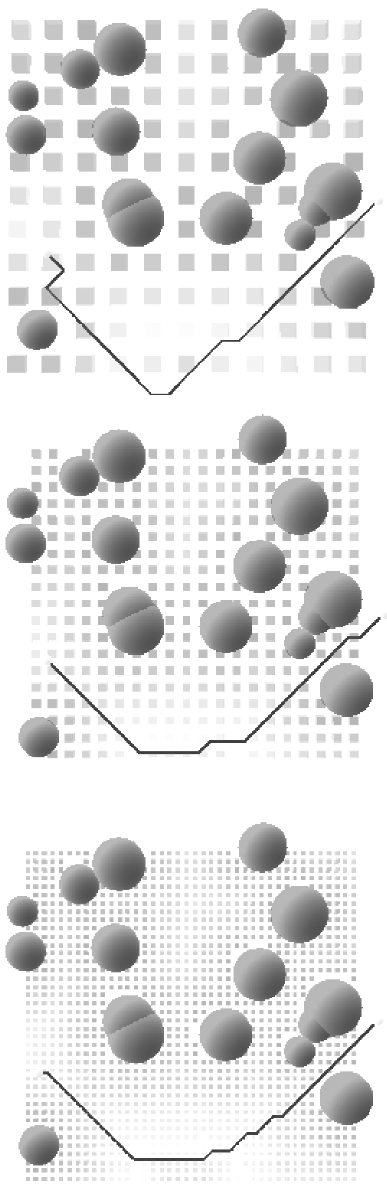

Figure 11: The discovered path for the different ranks of the adaptive mesh

6 Conclusion & future work

We have proposed hybrid and real-time path planning technique for real applications with the ability to read information about environment through the specific devices equipped with sensors. The provided method can be used in both 2D and 3D applications and works in known, partially known or unknown environment. We cooperate on research with authors of this technique in the team from the University of Calgary.

In the future, we are going to improve several parts of the algorithm solution:

- Enhancement of a method for filling the obstacles map according to the scene description: In our solution, we use unoptimized code for filling the obstacles map. If we want our technique to be usable in computer defined and abstract environment, we must assume the scene will be provided in one of the description formats. So it would be better to improve the way the map grid is created and filled from this scene representation.

- Enhancement of the adaptive structure: The way the adaptive structure copes with changes in mapped space is another great factor of algorithm performance. Topology and adaptation behaviour of this structure are main ways we want to focus on.

- Interleaving the waypoints of the found path with a specific curve is another step to make the path planning results look more human-like. While creating this curve, we must keep the collision-free property of found path and that will be another way of development we are going to take.

Acknowledgements

This work was done in cooperation with the University of Calgary in Canada. I would like to thank to Dr. M. Gavrilova and Mr. R. A. Apu for providing me the starting knowledge, 2D implementation of their solution and advices to continue in this project. I would also like to thank to Dr. I. Kolingerova from the University of West Bohemia, Pilsen, Czech Republic, for supervision and help with the paper preparation.

References

[AG05] R.A. Apu, M. Gavrilova. *Adaptive Spatial Memory Representation for Real-Time Motion Planning*. Proceedings of the 8th International Conference on Computer Graphics and Artificial Intelligence, 2005.

[Her87] John Hershberger. *Finding the Visibility Graph of a Simple Polygon in Time Proportional to its Size*. Annual Symposium on Computational Geometry, Proceedings of the third annual symposium on Computational geometry, 1987.

[Ram96] G.D. Ramkumar. *An Algorithm to Compute the Minkowski Sum Outer-face of Two Simple Polygons*. Annual Symposium on Computational Geometry, Proceedings of the twelfth annual symposium on Computational geometry, 1996.

[War90] C.W. Warren. *Multiple Path Coordination using Artificial Potential Fields*. Proceedings of the IEEE International Conference on Robotics and Automation, 1990.

[Ste94] Anthony Stentz. *Optimal and Efficient Path Planning for Partially-Known Environments*. Proceedings of the IEEE International Conference on Robotics and Automation, 1994.

[Rou98] J. O'Rourke. *Computational geometry in C (2. edition)* (http://maven.smith.edu/ ~orourke/books/compgeom.html). Cambridge University Press, 1998.

[DPV04] S. Dasgupta, C.H. Papadimitriou, U.V. Vazirani. *Paths in graphs* (http://inst.cs.berkeley.edu/~cs170/sp04/notes/ dijkstra.pdf).

[Bat04] Ch. Batten. *Algorithms for Optimal Assembly* (http://www.mit.edu/~cbatten/work/ssbc04/ optassembly-ssbc04.pdf).