Path planning in dynamic environment using an adaptive mesh

Petr Brož[1,3]
pebro@students.zcu.cz

Ivana Kolingerová[1]
kolinger@kiv.zcu.cz

Russel Ahmed Apu[2]
apu@cpsc.ucalgary.ca

Marina Gavrilova[2]
marina@cpsc.ucalgary.ca

Přemysl Zítka[1]
premysl@students.zcu.cz

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

[2]Department of Computer Science
University of Calgary
Calgary, Canada

ABSTRACT

Solutions for the common problem of path planning in an abstract environment have been extensively developed in many scientific disciplines. However, almost all explored techniques assume the environment does not change and that there is a complete and detailed overview of this examined space. In addition to, many methods for the path planning need to derive a specific graph structure from the environment representation and it can be often very difficult to construct or obtain this structure in some real applications.

In our proposal, we introduce a general model for the real-time path planning in a known, partially known, unknown and dynamic environment. We provide a hybrid technique that combines a graph and grid representation of the examined space and that is trying to combine the advantages from both types of the environment representation. The proposed path planning method uses an adaptive mesh for its graph part to provide the capability of the assimilation to the changing environment.

The presented method offers faster times for the path retrieval then the classical raster based approaches and works in a dynamic environment where the conventional graph based techniques fail. On the other hand, there are still some higher memory requirements of the proposed solution due to the necessary raster representation of the examined environment.

Categories and Subject Descriptors

D.3.2 [**Language Classifications**]: C# 2.0, D.3.3 [**Programming Languages**]: Language Constructs and Features – *abstract data types, genericity.*

General Terms

Algorithms, Measurement, Performance, Design, Experimentation, Theory.

Keywords

Path planning, Motion planning, Adaptivity, Level of Detail (LOD), Virtual reality, Computer graphics.

1. INTRODUCTION

The first methods for finding an optimal path in an abstract environment were developed even before the information science appeared. Therefore, there are many fast and efficient techniques for solving this general task. These conventional methods are in most cases subdivided according to the representation of the examined environment they are able to work with. Some path planning algorithms demand a graph-like definition of the processed scene (e.g., geometrical definition of all obstacles and other forbidden areas) and other algorithms assume the discrete representation of the surrounding environment is available. The graph based approaches often need to derive special graph-like structures from the provided environment description and work as lately as with this structure prepared whereas the raster based approaches usually do not need such preprocessing and search the optimal path directly in the provided grid representation of the environment.

The path planning algorithms based on both types of the environment representation have crucial and radical disadvantages when they are to be used in the real applications. The graph representation of the real environment is rarely available and its construction is – if possible at all – very complicated and difficult. On the other hand, the discrete representation of the examined space is much easier accessible and primarily measurable but the algorithm itself is in most cases (due to the amount of the raster elements to inspect) very time-consuming. In addition, almost all methods for the path finding and planning need well-known or static environment which is not often available, either.

A great improvement for the mentioned applications of these path

[3] Supported by the Ministry of Education of the Czech Republic – project No. LC 06008

planning systems can be achieved with the combination of the discrete and graph environment approaches. Such a technique could use an adaptive spatial structure as a graph with its vertices and edges evaluated according to the specific values from a collaborative grid structure. It would be then able to discover a pseudo optimal path (an optimal path among all available transitions in the graph but not among the grid values) and, for example, continuously adapt this spatial structure to the actual state of the environment and other dynamic influences.

In this paper, we propose a possibility for the path planning over the combined environment representation which reduces (or eliminates) the main disadvantages of the mentioned conventional approaches. We use an adaptive mesh to define all available waypoints and transitions for searched paths and a simple 3D matrix to store all raster based values. As the result, we provide a path planning technique that is faster than the raster based approaches and suitable for the applications in the dynamic environment (preliminary version of our method, without the adaptivity, has been published in [Bro06]).

2. STATE OF THE ART

Path planning in general denotes a basic problem of finding an optimal path between two specified spots in an abstract environment representation. In this context, the optimal path means the path satisfying one or more given objectives (the shortest, the cheapest or the fastest path, the path with the maximal clearance among all surrounding obstacles, etc.). The mentioned abstract environment can be represented in a variety of ways but the path planning algorithms are focusing mainly on evaluated graphs and 2D or 3D grids. There are many ways these environments can be differentiated (dynamic/static, known/unknown environments, etc.) which implies a similar distinction of the path planning techniques according to the types of the environment they are able to work with.

2.1 Graph based methods

First, let us introduce the most known approaches based on the graph representation of the examined space. The **Visibility graph** [Her87] technique extends the basic provided geometrical definition of the environment with the edges connecting the points that can "see" each other whereas the source and destination position is treated as an obstacle, too. The new edges (together with the edges defining the sides of each obstacle) then represent the possible transitions and through them, the optimal path can be found. An example of such preprocessing in a 2D application can be seen in Figure 1: the edges of all obstacles (filled with the bricks pattern), the starting and ending position (the points labelled with the letters S and E) are connected according to their mutual visibility and over the possible transitions (the thin lines together with the obstacles sides), the optimal path (the dashed lines) is found.

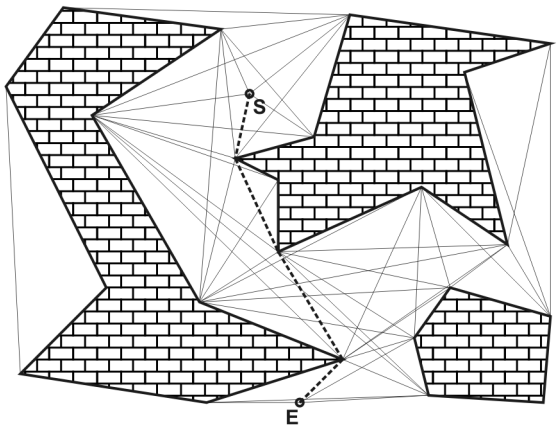

Figure 1: An example of the scene processing with the **Visibility graph** technique

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

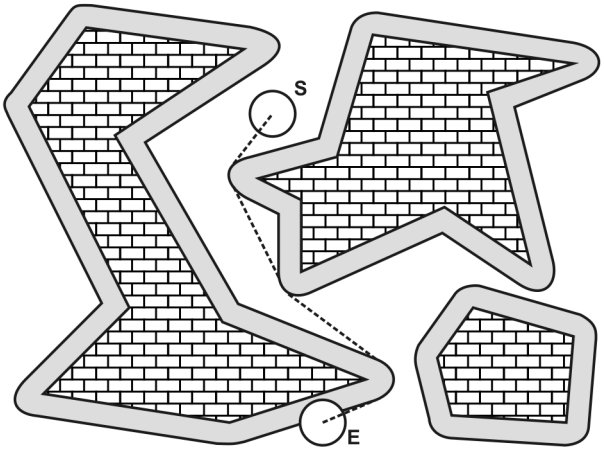

Figure 2: An example of the path planning with the **Minkowski sum** method

2.2 Raster based methods

After the short introduction into the graph-based methods for the path planning, we are providing insight into the techniques based on the grid representation. Such a grid can be precomputed (in case it is not provided) or modified at the beginning of the

algorithm. In reference to the modification of the explored grid, the **potential field** model [War90] can be used for filling the grid with the discrete values of a specific potential field generated 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. The most known techniques for searching itself are for example the **A*** (for the well-known environment; [Bat04]) and the **D*** (for the unknown, partially known or changing environment; [Ste94]) algorithms. Figure 3 shows the manner of such path finding in the grid: the obstacles from Figure 1 and 2 are now splitted into the raster and in this raster, the optimal path between the starting and ending position (the cells labelled with the letters S and E) over the grid cells is outlined.

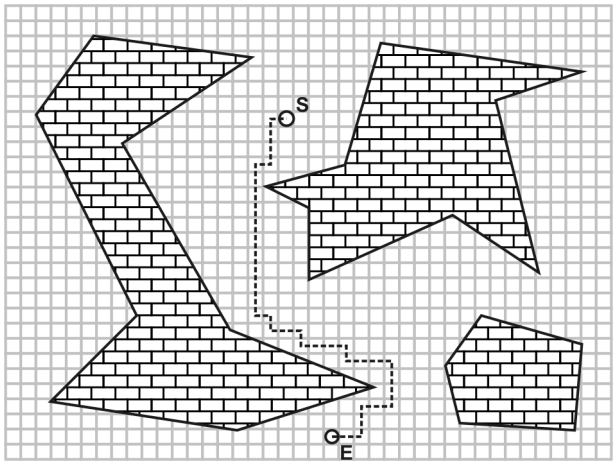

Figure 3: An example of the raster based path planning

3. THE PROPOSED MODEL

To provide a suitable path planning model for the mentioned applications where the conventional approaches fail, we are focussing on a general path planning technique that is able to work in the known, partially known or unknown discrete environment and that is designed for use in the virtual reality with the possible 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 (depending on the application they are used for) and survey certain locations of the examined terrain according to the orders from the HQ. Such headquarters then keep specific "paper maps" to sketch in the discovered obstacles and other threats which are then periodically complemented and updated with the actual values measured by the scouts. The idle 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 terms of the maximal clearance among all obstacles) are known and the scouts are then guided only to the locations with a suspicion of the possible threats. Figure 4 shows an example of such environment exploration in a 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 stored into the obstacles map (impassable areas) and into the threats map (a potential field of discovered threats).

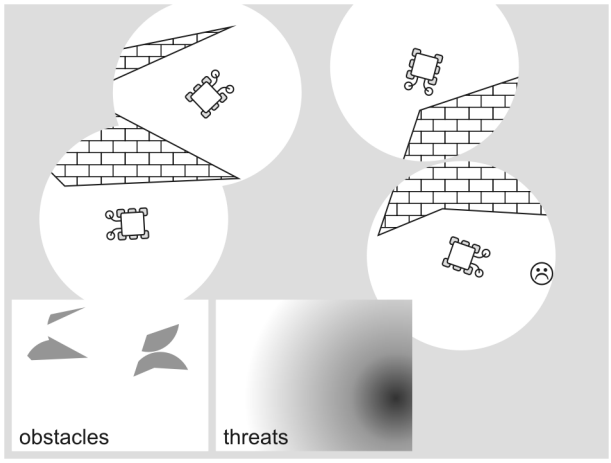

Figure 4: A preview of the 2D terrain sensor-based exploration with the autonomous robots

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 reachable positions and transitions with its vertices and edges.
- A grid structure for the discrete representation of certain environment hazards (hereafter referred to as a **map**), e.g., the proximity to an obstacle or the dynamic threats.
- An autonomous AI entity (hereafter referred to as an **agent**) for the real-time survey of the explored space 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 the danger weights as the proximities to the nearest obstacle in the mapped space and the second one, called **threats map**, represents the potential field generated by all located and observed threats in the examined terrain. In the following, the proposed path planning model keeps a mesh that is „widespread"

over the examined space covered also by the mentioned mapping structures. This mesh then defines all available waypoints and transitions the agents can travel during their exploration and continuously copes with the changes in both mapping structures and with the 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 areas.

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

In the mentioned fictive terrain exploration, the continuous prospecting of the environment was a task for the robots but in our current solution and demonstrating application, we assume the obstacles in the environment are already completely explored – that the obstacles map is filled with the 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 (minimal proximity) to 1 (maximal proximity to the nearest obstacle or the obstacle itself). The elements of the threats map are then evaluated in a similar manner during the mesh adaptation.

Single iteration of the mesh adaptation in the proposed path planning system 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 (but in the demonstrating application, the environment surveying is accomplished at the beginning of the program, as mentioned above).

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 (this ensures the mesh will „remember“ the important locations of the examined space; the least visited places – the vertices with the lower influence – are not important for the exploring and that is why the adaptive mesh does not need to be refined around them).

3. *Error function evaluation and refinement*

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

4. *Orders execution*

   Each agent executes its orders – he finds an optimal path to the given location with the provided cost function or follows its already computed waypoints (if the path cannot be travelled due to the refinements of the mesh, it is recomputed to the first existing waypoint).

5. *Exploration*

   If all user's objectives and goals are reached, the agents ensure the exploration of the unvisited locations in the examined space – they automatically plan the path to the vertices with no values recorded. Steps 3, 4 and 5 are not accomplished in our current solution as the demonstrating application prepares and fills the mapping structures at its beginning and so there are no unvisited locations that should by explored by the agents.

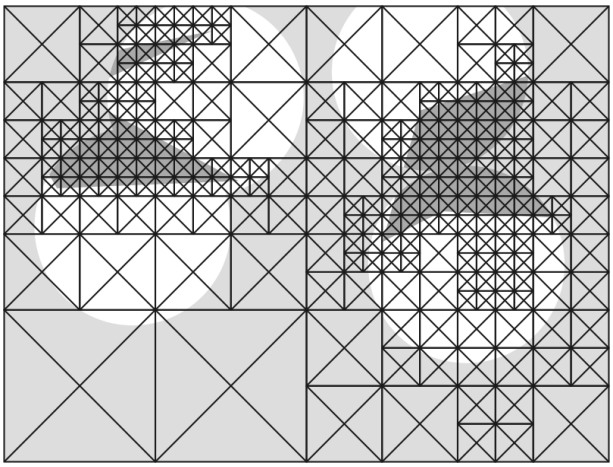

Figure 5: A single stage of the adaptive mesh for the same type of the explored terrain as in Figure 4

After a certain time, the mesh is fully adapted to the static obstacles and copes only with the dynamic influences – with the threats. A pseudo optimal path for the user can then be computed using Dijkstra's algorithm with the specific cost function (there are many ways to specify the cost functions and in addition to, they can differ according to the type of the application). We are

not focusing on the definition of these cost functions but an example of that function (concretely the example of the function used by the robots during their exploration) can be found again in the [AG05].

4. EXPERIMENTS & RESULTS

4.1 Demonstrating application

We have implemented a simple application (in the C# language with the Direct3D libraries) to provide the results and comparisons of our algorithm. In this demonstrating application, we generate a certain number of obstacles formed by the solid spheres with different radii and we also add some abstract threats represented by the small red cubes. During the program run, the threats are directed to the random locations of the examined space and the adaptive mesh is refined in each iteration. The obstacles map is precomputed and filled with the corresponding values before the main loop of the program. Therefore, the complexity of this structure and its creation does not affect the qualities of the real-time method itself. To demonstrate the adaptivity of the algorithm, the optimal path is recomputed after each refinement of the mesh.

An example of such a testing scene with 16 obstacles and 2 threats is shown in the following images: Figure 6 shows the basic scene only with the found path between two opposite corners of the examined space, Figure 7 then shows the same scene together with the actual state of the adaptive mesh (with the maximum level of the mesh division equal to 4) and Figure 8 shows again the same scene but this time with the weight values from the obstacles map (the darker cubes define safer locations and the lighter cubes define more dangerous locations).

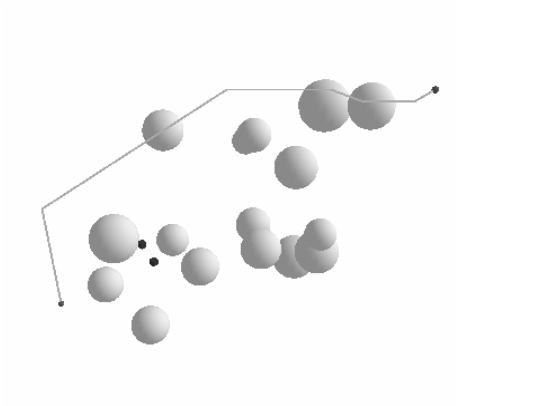

Figure 6: An example of the random scene in the demonstrating application

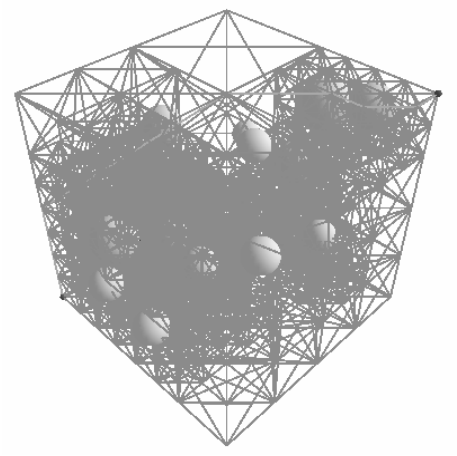

Figure 7: A snapshot of the scene from Figure 6 with the adaptive mesh

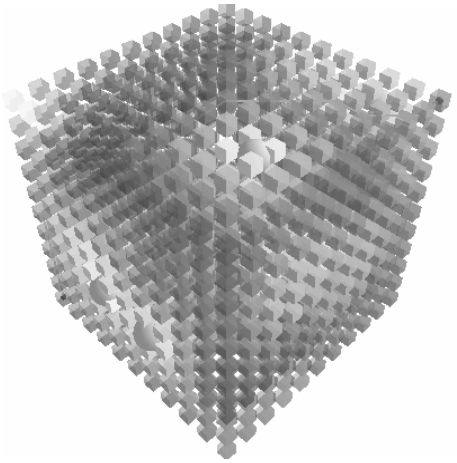

Figure 8: A snapshot of the scene from Figure 6 with the values from the obstacles map presented

4.2 Survey of the tests

For our survey, we have selected and measured the following variables as the most characteristic and important parameters of the proposed method:

- **Clusters amount** defines the count of the atomic elements in the adaptive structure (in Figure 7, these are the smallest cubes with all corners connected to each other).
- **Adaptation time** determines the time needed for the single iteration of the adaptive structure progression.
- **Allocated memory** defines the memory requirements of our C# implementation (debug version).
- **Path finding time** denotes the time needed for finding the optimal path between two constant spots in the opposite corners of the explored space.

The mentioned parameters have been measured with the following testing datasets to present the qualities and possible weak points of our implementation:

- **Dataset #1** describes the algorithm in the environment consisting of 8 solid spheres (these obstacles fill about 3% in the volume of the examined space). The rank of the grid used in this preset is equal to 32 (that means the obstacles map contains 32 768 elements) and the maximum level of the mesh division is set to 4 (the smallest cluster in the mesh has one sixteenth of the original width).
- **Dataset #2** defines the same adjustment of the algorithm as the dataset #1 (8 solid spheres with different radii randomly dislocated in the examined environment and filling again about 3% of the space, 32x32x32 elements in the grid) except that the maximum division level is equal to 5.
- **Dataset #3** specifies the configuration with 16 obstacles in the examined space (filling about 6% of the explored environment), with the rank of the obstacles map equal to 64 and the maximum level of the mesh division set to 4.
- **Dataset #4** again defines the same adjustment of the previous dataset (16 obstacles filling about 6% of the examined environment, 64x64x64 elements in the grid used for the obstacles map) except that the maximum division level is set to 5.

First of all, we present the functional dependence of the clusters amount on the particular iterations of the program (for all mentioned datasets) in Figure 9. At the very beginning of the main loop, we can see a rapid growth of the clusters amount as the adaptive mesh refines itself around the obstacles. The remaining behaviour of this dependence highly varies due to the movement of the threats in the scene. When the threat shifts off from the near obstacles, it raises the weight value of the previously unimportant locations and so evokes a new refinement of the mesh around these locations. Such situations then evoke the peeks of the clusters amount that are visible in all mentioned dependencies in Figure 9.

The environment of the testing application changes itself in an absolutely random way as the threats are directed to the random locations in it. Though, it is possible to discover some events in the application from the presented dependencies. We can take a look for example at the graph for the dataset 2 in Figure 9: the growths in the behaviour (during the iterations 150-200, 350-500, 600-700, 780-800 and 920-980) are owing to the threats that smooth away from the nearest obstacles and so invoke the mesh refinement in the previously unimportant locations. In Figure 9, we can also see the datasets 2 and 4 are much more varying than the datasets 1 and 3 but there is a clear explanation for it. With the higher level of maximum mesh division, there are more clusters reacting on the threats movement.

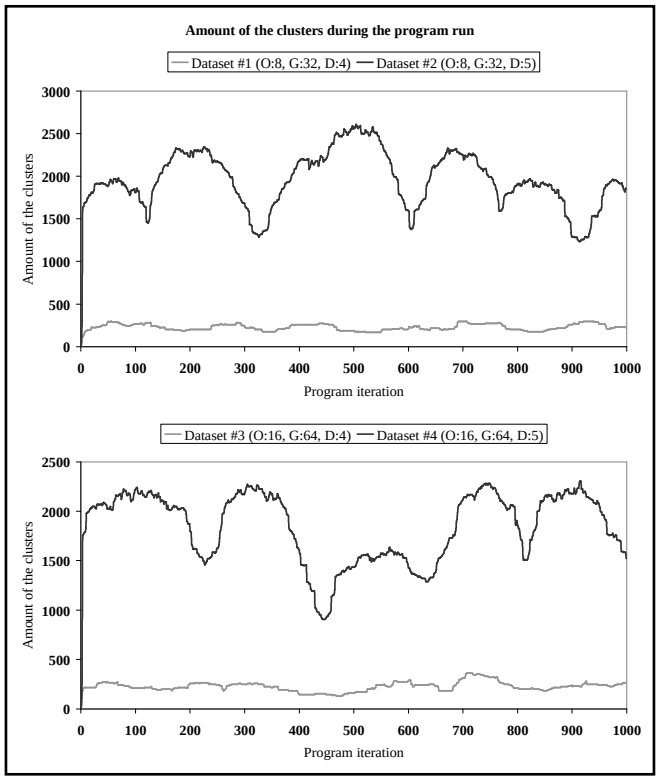

Figure 9: A clusters count dependent on the program iterations

Figure 10 shows the time requirements of each single mesh adaptation. There are presented the moving average (8 periods) values instead of the original values. Apparently, the time complexity of the adaptation depends mainly on the maximum level of the mesh division. Also in these graphs, the peeks can be explained with the behaviour of the threats. When the threats shift away from the near obstacles, they fill more elements of the grid with higher values and so there are more areas for the refinement.

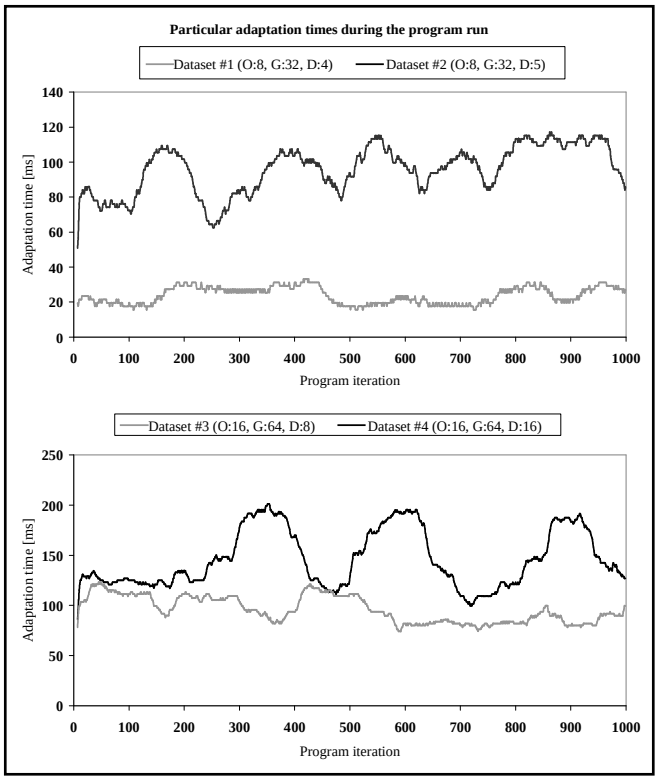

Figure 10: A time complexity of the mesh adaptation dependent on the program iterations

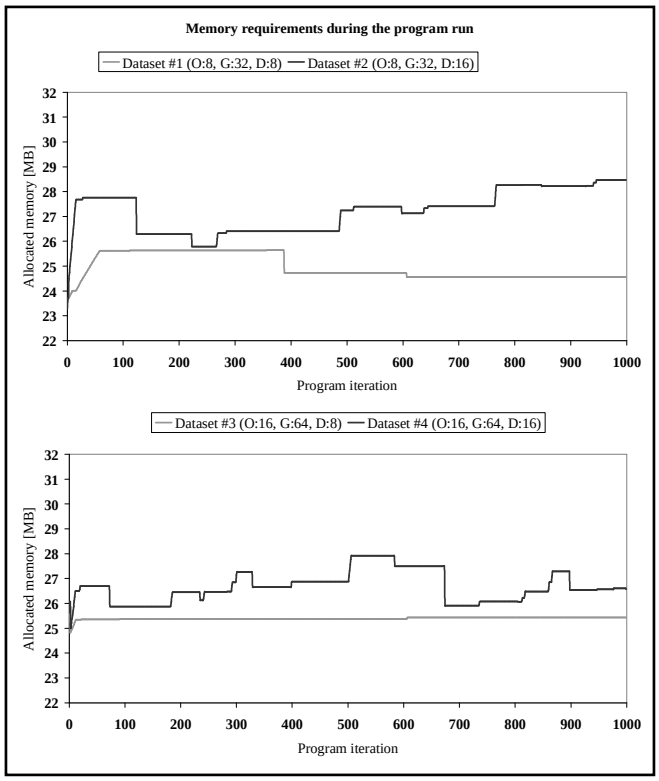

Figure 11: Memory requirements of our implementation dependent on the particular iterations

Finally, Figure 11 provides common insight into the memory requirements of our implementation. The measured values are approximate due to the garbage collector used in the C# programs. High and constant amount of the memory is required for the grid structure (for the rank of the grid equal to 32, program must allocate the memory for the 32x32x32 matrix consisting of the float values) and so the step changes in the dependencies are caused only by the mesh adaptation itself.

4.3 Obstacles count independence

The demonstrating application was also used to survey the algorithm independence of the obstacles amount. We have defined other adjustments of the algorithm similar to the presets from the subsection 4.2 and we have measured the properties of our solution for 256, 512, 1024 and 2048 obstacles randomly dislocated throughout the space. The concrete values are again defined in the parentheses behind each preset in the legends – 'O' stands for the obstacles count, 'G' stands for the rank of the grid and 'D' stands for the maximum level of the mesh division.

Figure 12 shows the functional dependence of the clusters amount and the particular iterations of the program for these new configurations. The presented graphs indicate that the clusters count is not directly dependent upon the obstacles amount. The differences in these graphs are caused by the topology of all obstacles in the scene that is randomly generated for each dataset. The same situation of this separateness occurs in Figure 13 with the functional dependence of the mesh adaptation times and in Figure 14 with the dependence of the time for finding the path. Figures 13 and 14 are again presented in the form of the moving averages due to the oscillating measured values. The considerated obstacles are preprocessed in the grid and that is the only part of our solution that is affected by the count of the obstacles.

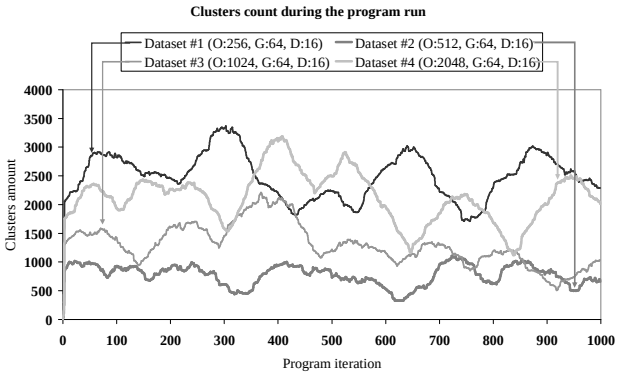

Figure 12: Clusters counts in the particular iterations of the program

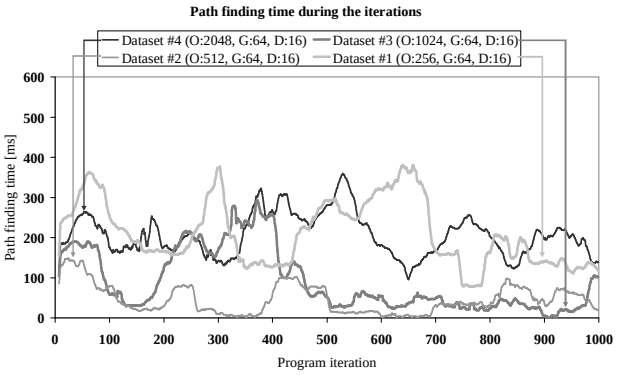

Figure 14: Path finding times dependent on the particular iterations of the program

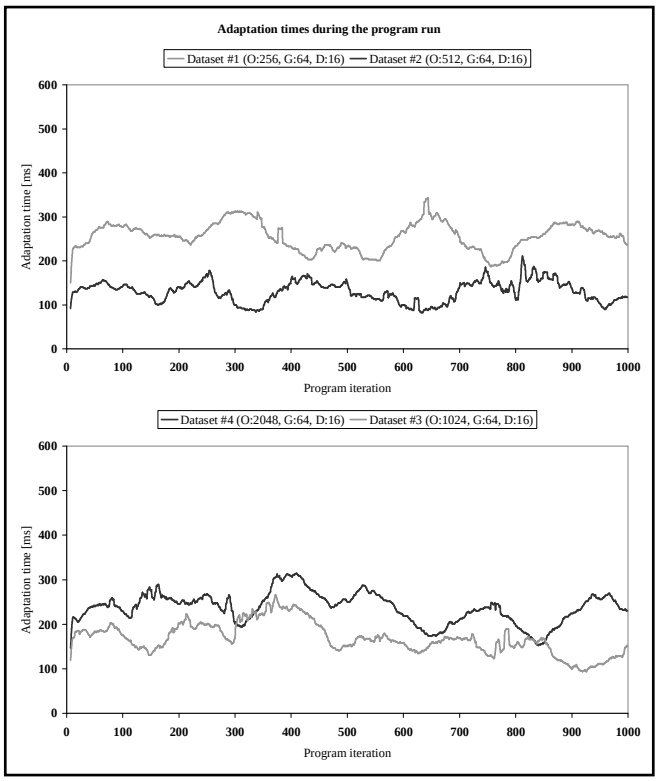

Figure 13: Mesh adaptation times dependent on the particular iterations of the program

4.4 Non-adaptive mesh comparison

We also own an implementation of the proposed path planning technique with the regular mesh - it provided us the opportunity to compare this old solution to our current implementation.

Figure 15 shows two selected configurations from the subsection 4.3 together with the results from the two equivalent adjustments of the old algorithm with the regular mesh (in this case, 'G:64' means the regular mesh consisting of 64x64x64 clusters). The times for the regular mesh stay around the value 400ms whereas the times for finding the path in the adaptive mesh strongly vary (average time for the preset #2 is about 45ms and for the preset #4 about 200ms) but they never achieve the time needed by the old algorithm. Figure 16 then presents the values measured simultaneously with the path planning times for Figure 15 (values are interpolated by the polynomial regression of the third grade) and demonstrates the identical results in the path optimality for these techniques. In this context, the path optimality is evaluated according to the highest danger weight in the waypoints on the found path where this optimality grows with the descending maximal weight (Figure 16 shows this maximal weight during the program run).

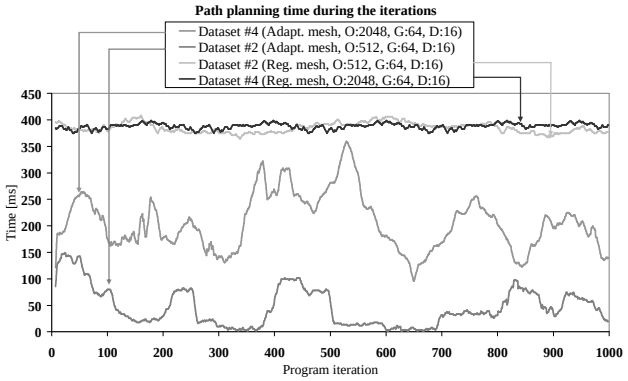

Figure 15: Path planning times dependent on the particular iterations of the program

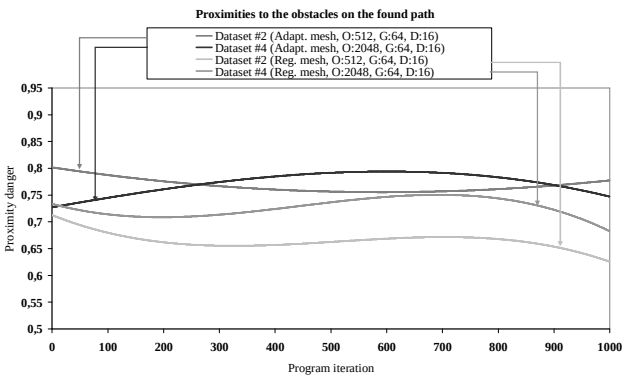

Figure 16: Proximities to the nearest obstacles on found paths during the program iteration

The measurements show the proposed path planning method is a suitable alternative for the path planning in dynamic environments: it is faster than the raster based approach and it is usable in the applications where the graph based techniques fail. On the other hand, there are still high memory requirements due to the 3D matrix for the grid used in our solution.

5. CONCLUSION & FUTURE WORK

We have outlined the possible model for the path planning system that eliminates the described disadvantages of the conventional approaches applied in the virtual reality. We have measured the most important properties of our implementation and provided the gained dependencies, some of them compared to the results of the conventional approach. The provided method can be used in 2D and 3D applications and works in the known, partially known or unknown and dynamic environment. In comparison with the regular mesh, the method with the adaptive mesh needs only about 10-50% of the original time for finding the optimal path (on equal optimality results).

5.1 Possible trends

The proposed solution is still under development and there are many possible ways to improve this model for the real-time path planning. In a few following points, we are denoting the most important and the most interesting ones:

- An enhancement of the method for filling the obstacles map according to the scene description – In our solution, we use an unoptimized code for filling the obstacles map. If we want our solution to be usable in computer defined and abstract environment, we must assume the scene will be provided in one of the common description formats. Then it would be better to improve the way the mapping structure is created and filled from this scene representation.
- An enhancement of the adaptive structure: the way the adaptive structure copes with the changes in the mapped space is an important factor of the algorithm's performance. In the measured dependencies from the section 4 it is obvious that the main effect of our solution appears from the mesh adaptation process. The topology and adaptation behaviour of this structure are the main points we want to focus on in the future.
- Interleaving the waypoints of the found path with a specific curve is another step to make the path planning results look more human-like and so to make them more useful. While creating this curve, we must keep the collision-free property of the found path and that will be another way of the development we are going to take.

6. REFERENCES

[Bro06] P. Brož. *Path Planning in Combined 3D Grid and Graph Environment*. Proceedings of the 10th Central European Seminar on Computer Graphics, 2006.

[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] J. 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 12th 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] A. 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 (2nd 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).