Environment preprocessing for real-time navigation of intelligent agents

S. Zhukov[*], A.Ions[*], G.Kronin[**]

CREAT Studio (USA, SF)
[*]Applied Mathematics Department
St.-Petersburg State Technical University (Russia),
[**]Calculus Department, St.-Petersburg State University (Russia)

{zh,iones,kronin}@creatstudio.spb.ru
www.CreatStudio.com

Abstract

The problem of intelligent agent navigation in unknown synthetic environment is a subject of a number of recent AI researches. Even though many solutions have been proposed to address it, ever-growing complexity of virtual environment and development of sophisticated characters make it necessary to further elaborate computational models used to make intelligent agents navigation more robust and well-looking.

In this paper we formally study the navigation problem and describe our navigation technique. The approach is based on the use of so-called *accessibility graphs*. We depict a general scheme applicable to control the navigation of significantly different agents in a similar fashion even in the most challenging environments. The technology is perfectly suited for the use in modern 3D video games and simulators.

Keywords: AI applications, Navigation, Path planning, Real-time AI.

1. Introduction.

Basically, a navigation problem is posed as follows. Given an environment and the set of obstacles, the initial position x of an intelligent agent and the goal position y, drive the agent from x to y so that the agent does not collide with obstacles, or establish such a path does not exist. Throughout this paper we will assume that an agent is a point. Usually it is assumed that the environment consists of a set of non-intersecting polygonal (polyhedral in 3D) obstacles and a free space. All obstacles are assumed to be open. The following basic 'motion types' can be distinguished:

- 'free' motion in the environment (2D/3D);
- motion along the surface (∂2D/∂3D).

The set of the permissible points E is specified, i.e. the set of points where an agent may be located. In case of free 2D/3D motion, all points lying outside the obstacles are considered to be permissible. In case of ∂2D/∂3D motion, only the points on the obstacle boundaries are permissible.

1.1 Previous work.

A number of researchers have proposed different methods (algorithms and data structures) to resolve many special cases of a general navigation problem. Most methods can be classified into *environment-oriented* and *agent-oriented*.

Environment-oriented methods find the path (or optimal path), knowing all the information about the environment. To find the path, one usually employs dynamic programming (breadth-first search), visibility graphs or road maps with Dijkstra algorithm used to find the optimal paths. Being theoretically sound, these methods are not perfectly suitable for real-time applications since the data they operate on are very large, comparable or even greater than the complexity of the environment itself.

Agent-oriented methods use only local information about the environment available, e.g., via collision detection or synthetic vision. Among the algorithms of this class there are left/right hand traversal, visual-based navigation [RENAU95][NOSER95] and so on.

Visibility graphs.

To resolve the navigation problem in the environment consisting of the set of polygonal obstacles one may use so called *visibility graphs* [O'ROURK95]. A *visibility graph* has nodes associated with the vertices of the polygonal obstacles and its edges correspond to the visibility lines among the nodes. Note that it is sufficient to use edges corresponding to the convex vertices only (Figure 1a). Given the initial position x and the goal position y, it is always possible to compute the optimal path using Dijkstra algorithm [O'ROURK95][BERG97][MITCH97]. A disadvantage of this method is that visibility graphs require much processing time to compute and have high storage requirements (namely, they use $O(N^2)$ memory, where N is the overall number of vertices in the given polygonal obstacles). Finally, they do not generalize in 3D, since finding optimal paths in among the polyhedral obstacles in 3D is an NP-hard problem [BERG97].

Road maps.

Another approach is based on the trapezoidalization of the environment and the construction of a *road map* (Figure 1b) [BERG97][CANNY93][SHARI97]. This method has less storage requirements and always finds the path if it exists. On the other hand it does not allow to compute the optimal paths and it's practically impossible to implement it in complex 3D environments. Both road maps and visibility graphs are hardly usable for non-trivial locomotion capabilities of an agent since these capabilities are not accounted at all.

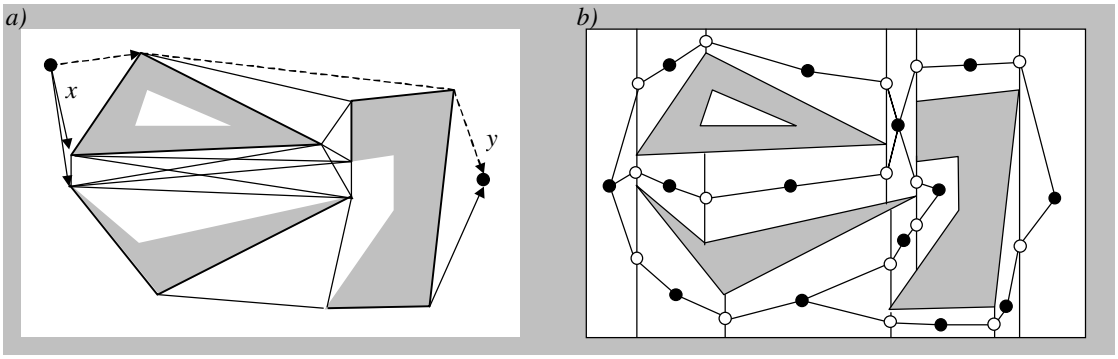

Figure 1. Visibility graph (a) and road map (b).

Graph search.

One of the most well-known contexts of the general navigation problem concerns navigation in rectilinear mazes. To find the path in such a maze the conventional breadth-first search can be used. To drive the agent, depth-first search can be performed. Alternatively, there are other maze searching algorithms based on left/right hand maze traversal and random walks with backtracking. A number of efficient path replanning methods for the navigation in dynamic environments are based on the *D** algorithm [STENZ94].

Potential minimization methods.

In this method, a *potential field* $P_{X,Y}$ is defined [BADL95]. To make the step being in some point x_i, the agent selects one of the points x_{i+1} such that the function $P_{X,Y}(x_{i+1}) + d(x_i, x_{i+1})$ is minimal (here $d(x_i, x_{i+1})$ denotes the cost of stepping from x_i to x_{i+1}).

Potential minimization method is easy to implement and it is not restricted to be used in environments with polygonal obstacles only. The drawback of this method is that the agent may get stuck in the local minimum and the path toward the goal may be missed while it actually exists. Theoretically, it is possible to construct potential functions without local minimum [ZELEK], but these methods require too much processing power and are not practically implementable.

On Figure 2 we illustrate potential minimization method. Moving from the initial position *x* toward the goals *y* and *y''* the agent finds reasonable paths that carefully avoid obstacles. The path from *x* to *y'* is farther from optimal – the agent tried to bypass the obstacle, deviated too far from the goal and performed a step over the obstacle – it was cheaper to step over immediately when the obstacle was encountered. Finally, heading to the goal *z*, the agent got stuck in local minimum.

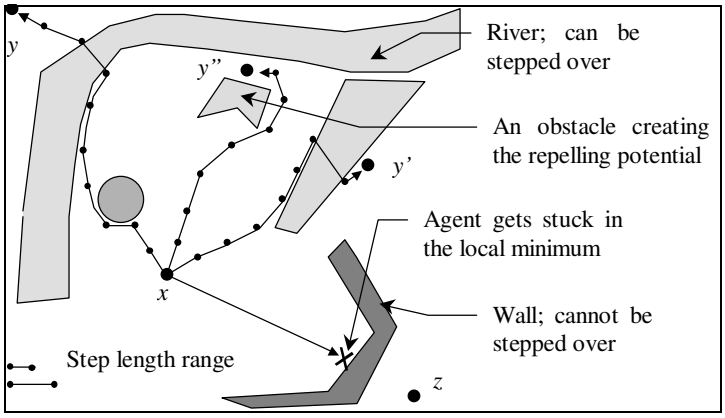

Figure 2. Illustration to the potential minimization method. Grayed areas are obstacles.

1.2 Accessibility graphs.

Studying the general navigation problem, we aimed at creating a solution that would possess the following properties:

- applicable to different types of agent locomotion capabilities;
- works well in 2D and 3D;
- does not require much processing power on real-time path planning and does not rely on the underlying hardware;
- always finds the path if it exists.

Our method is based on the *accessibility graphs*. Heuristic methods of accessibility graph creation were described in [ZHUKO98]. To create such graphs, the environment is subdivided into the zones such that within each zone the navigation problem can be resolved using some simple algorithm. Each node in the accessibility graph corresponds to one zone, and an edge between the nodes is established if the agent may move from one zone to the other. Clearly, the graph intrinsically depends on the agent's locomotion capabilities.

2. Navigation problem – formal statement.

2.1 Different types of a general navigation problem.

The general intelligent agent navigation problem can be formulated in three different ways:

- navigation in the unknown or partially known environment (exploratory navigation);
- off-line map building for navigation;
- navigation in the known environment using maps.

Navigation in the unknown environment (NU)

Problem: given the environment E, initial position x and the goal position y, drive the agent from x to y. The navigation algorithm does not possess any kind of global information about the environment. It must use only local information available via collision detection or synthetic vision.

To find the path in the unknown environment, the agent usually builds a map on-line. To create such a map, the agent needs to extract and store the information on cross-accessibility among different parts of the environment and memorize some key points. For example, doing depth-first search for navigation in mazes the agent performs backtracking to the previously known key points (junctions). These key points implicitly constitute a map. The task of navigation in the complex unknown environment is a complicated, especially in the real-time systems. Practically, this problem is subdivided into the exploration and map building problem, and to navigation in the already known environment using the created maps.

Let's assume that the agent already *knows* the environment. It means that the agent possesses the information on the cross-accessibility among the different parts of the environment and can *associate* itself with these areas. Clearly, the information must be stored in compact form and must be easily retrievable.

We call the low-level algorithm invoked by higher level modules to navigate between key points as *walk algorithm W*. As stated above, walk algorithm operates only on local information such as collision detection or synthetic vision data. We assume as well that for a *well-built map* the walk algorithm does not need to create its own map by using, for example, recursive goal selection and backtracking. Formally speaking, algorithm W belongs to the CONST-SPACE class, i.e. it uses only some constant memory no matter how complex the environment is. For navigation among polygonal obstacles in 2D, there exists the universal walk algorithm W_U that fully resolves the NU problem, i.e. it drives the agent to the goal, if the path exists, and effectively detect that it's impossible [ZHUKO98a].

Thus, the navigation problem is subdivided into the map creation, global navigation using map and local navigation using walk algorithm. The navigation problem using maps can be defined as follows.

Navigation using maps (NM)

Problem: given the environment E, map M, walk algorithm W, the initial position x and the goal position y, drive the agent from x to y, or detect it's impossible.

As stated above, the map of the environment must be built according to the agent's capabilities, i.e. depending on the walk algorithm W. The map is built *well* if the agent moves satisfactory in the environment. It means that for any initial position x and goal position y the path from x to y, obtained by navigation using map and walk algorithm W does not differ much from the optimal path. The map building problem is a problem of compact knowledge representation about the cross-accessibility among different parts of the environment. We define the map building problem as follows.

Map building problem (MBP):

Problem: given the environment E and walk algorithm W, create the map M possessing the following. properties:

- $|M| << |E|$
- for any initial position x and any goal position y the path found using the navigation with maps and walk algorithm W does not differ much from the optimal one.
- the complexity of a single step performing is relatively low.

We assume that the overall path of the agent can be represented as a number of subsequent runs of walk algorithm W from one subgoal to another. This is illustrated on Figure 3. Moving from x to y, walk algorithm builds the path $l_W(x, y)$ that gets stuck (Figure 3.a) and does not reach the goal ($L_W(x, y) = +\infty$). The approximation of the optimal path with 5 subsequent runs of walk algorithm W is shown on Figure 3.b.

Now let's define the steps needed to create a map of the environment:

1. The environment E must be subdivided into the *zones* Z_i: $E = \cup Z_i$. Each zone is a set of points grouped together according to some accessibility criterion (depending on algorithm W). Note, that the zones may intersect.
2. Create the nodes of the oriented *accessibility graph* $G_M = (V,A)$, each node v_i corresponds to a single zone Z_i.
3. Create and weight the oriented edges among the pairs of nodes reflecting the accessibility property for the respective zones. An edge between v_i and v_j is established if the agent may move from Z_i to Z_j using W.

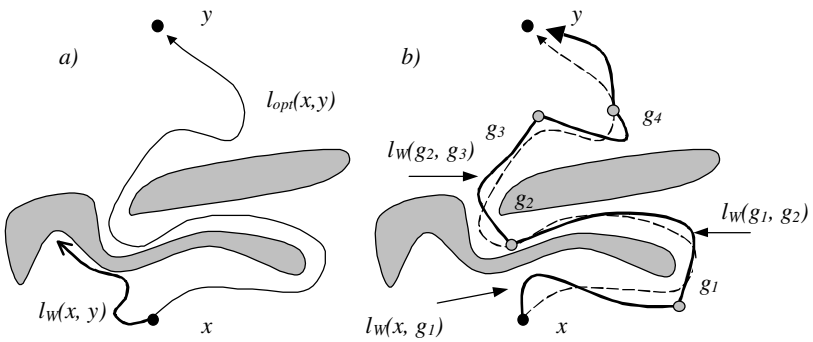

Figure 3. Approximation of the optimal path using the multiple runs of the walk algorithm

Once the zones are constructed, the navigation algorithm will drive the agent from the initial position x to the goal y by moving from zone to zone, until the zone containing the goal is not reached.

2.2 Descriptive and metric map building problem

The general map building problem can be stated as *descriptive* or *metric* one. *Descriptive* problem addresses the task of building the map that preserves the accessibility relations between the points in the environment. That is, it answers the question: is there *any* path from point x to point y?

On the other hand, *metric* problem solves the task of building the map such that for any points $x, y \in E$ the path found from x to y using the map M and walk algorithm W does not differ much from the optimal one: $L_{W,M}(x, y) \le C * L_{opt}(x,y)$, where C is some constant, $1 \le C < +\infty$.

From here and on we will use the following notations:

- $L_W(x, y)$ - the length of the path from x to y obtained using algorithm W without a map.
- $L_{W, M}(x, y)$ - the length of the path from x to y obtained using algorithm W with map M.
- $L_{OPT}(x, y)$ - the length of the optimal path from x to y .

If the path from x to y cannot be found or does not exist, we will write $L_W(x, y) = +\infty$, $L_{W, M}(x, y) = +\infty$ or $L_{OPT}(x, y) = +\infty$, respectively.

3. Navigation subproblems.

3.1 Association problem.

Once the environment is subdivided into the zones, our navigation algorithm must be able to associate any given point with them. Note that since the zones can intersect each other, a given point can be associated with many zones.

Definition. Let $E = \cup Z_i$ be a given environment subdivided into zones. The *association function* is a boolean function $\mathbf{A}(x, i)$ such that:

$$A(x,i) = \begin{cases} \text{TRUE, if } x \in Z_i \\ \text{FALSE, otherwise} \end{cases}$$

Definition. *Association algorithm* is an algorithm $A(x) \rightarrow N^+$, N^+ denotes a sequence of integer zone indices.

Clearly, the number of zones to which the environment is subdivided must be relatively small. Besides, each zone must be compactly specified so that the association algorithm will work fast and the map would not consume too much memory.

To define a zone, one may simply specify the association algorithm. So defined, a zone Z_i will contain such points x for which $\mathbf{A}(x, i) = TRUE$. Practically, the association algorithm exploits three properties of the points:

i) containment in a bounding volume (Figure 4.a);
ii) visibility of a key point (beacon) (Figure 4.b);
iii) containment in a single connected component (with respect to the obstacles); this can be effectively simulated by visibility of a number of beacons (Figure 4.c).

The properties exploited by the association algorithm depend heavily on the type of environment and walk algorithm. For example, if the association algorithm uses properties i) and ii), then zone Z_i can be defined by a pair (V,B): $Z_i = \{x \mid x \in V \ \& \ Visibility(x, B)\}$, where B is a beacon and V is a bounding volume.

We may note that the selected method of zone specification resembles the one used by humans walking in the environment. Usually, we memorize the environment by memorizing some number of key points. For example, we know that we are in our boss' room if we know we are on the second floor (bounding volume) and see the boss' arm-chair (key point).

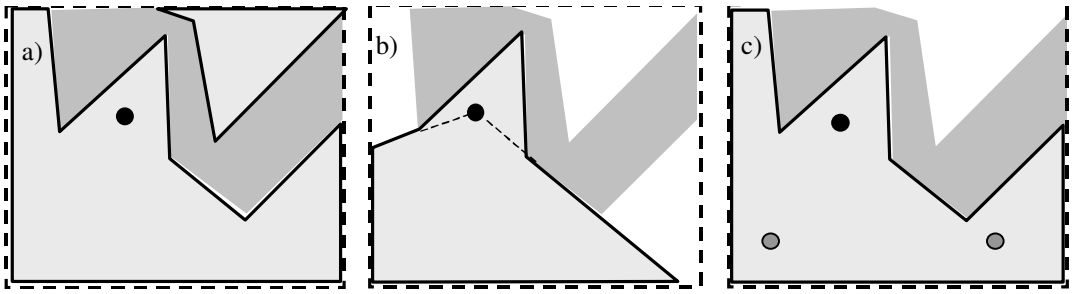

Figure 4. Illustration to the properties used by an association algorithm.
Dotted line – zone's bounding volume boundary. Dark gray – obstacles. Light gray – points associated with the zone. White – points within the volume that do not belong to the zone. Black points – zone centers. Light gray points - beacons.

The proposed method of zone specification is quite compact and practical. For instance, to specify such a complex zone as the zone Z_A (shown on Figure 5) and distinguish it from zone Z_B, it is still enough to specify just one bounding volume V and a few beacons. Obviously, it would be much more memory consuming to store the polygonal boundary of the zone, and this would cause the association algorithm to perform slowly.

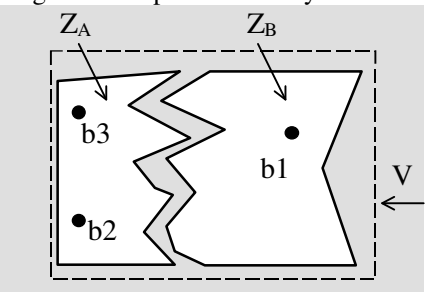

Figure 5. The power of the zone specification method using bounding volumes and beacons.

3.2 Zone types.

Definition. The *kernel* of the zone Z_i is the set K_i, such that $\forall\, k \in K_i,\ \forall\, x \in Z_i\ \boldsymbol{L_W}(x,k) + \boldsymbol{L_W}(k,x) < +\infty$.
Definition. Zone Z_i is a *star*, if its kernel is not empty: $K_i \neq \varnothing$.
Definition. Zone Z_i is *complete* if for all points $x,\ y \in Z_i$: $L_W(x,y) + L_W(y,x) < +\infty$.

Some examples of different zone types are shown on Figure 6. As a running example we will use two types of walk algorithms W:

- GoStraight algorithm drives the agent directly from x to y; if y is not visible from x, the algorithm does not find the path;
- GoCDT algorithm drives the agent from x to y repelling from obstacles (Figure 9) .

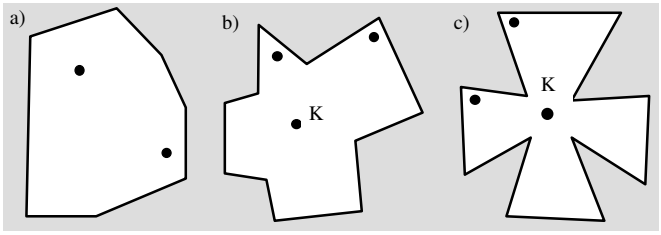

Figure 6. Different zone type examples.
a) – complete zone for W = GoStraight algorithm
b) – star zone for W = GoStraight; complete zone for W = GoCDT
c) – star zone for W = GoStraight and W = GoCDT

3.3 Zone connectivity.

Definition. Point x *is connected with* the point y by algorithm W (denoted as $x \rightarrow_W y$), if $L_W(x, y) < +\infty$.
Definition. Points x and y *are connected* by algorithm W (denoted as $x \leftrightarrow_W y$), if $x \rightarrow_W y$ and $y \rightarrow_W x$.

We will assume that walk algorithm W is fixed, so W subscript will be skipped.
Definition. Zone Z_i *is strongly connected with* zone Z_j (denoted as $Z_i \Rightarrow Z_j$), if $\forall\, x \in Z_i,\ \forall\, y \in Z_j$: $L_W(x, y) < +\infty$.
Definition. Zones Z_i and Z_j *are strongly connected* (denoted as $Z_i \Leftrightarrow Z_j$), if $Z_i \Rightarrow Z_j$ и $Z_j \Rightarrow Z_i$.
Definition. Zone Z_i *is weakly connected with* zone Z_j (denoted as $Z_i \rightarrow Z_j$), if $c_i \rightarrow c_j$.

Here and henceforth c_i denotes the *center* of the zone Z_i, i.e. any fixed point attached to Z_i.

Definition. Zones Z_i and Z_j *are weakly connected* (denoted as $Z_i \leftrightarrow Z_j$), if $Z_i \rightarrow Z_j$ and $Z_j \rightarrow Z_i$.

Some examples on environment subdivision into complete connected zones are shown on the figures below.

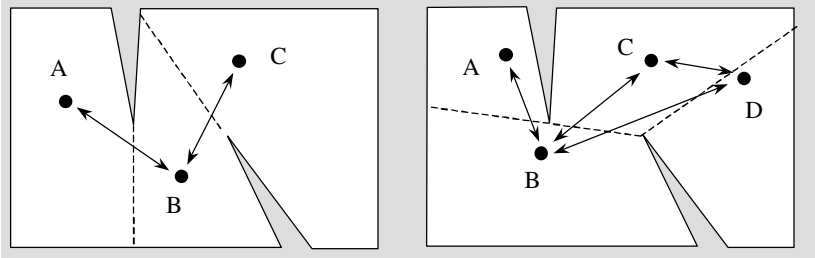

Figure 7. Two examples of environment subdivision into complete, weakly connected zones for W = GoStraight algorithm.

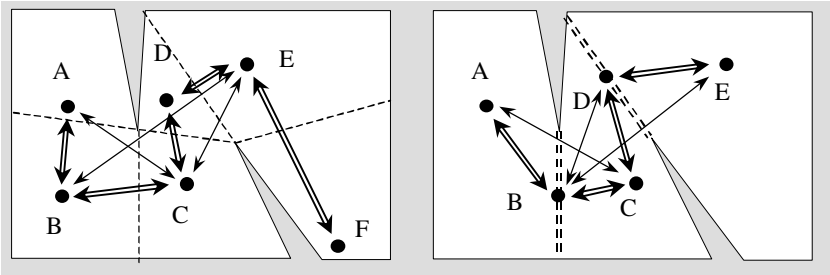

Figure 8. Different environment subdivisions into complete zones for W = GoStraight. The neighbor zones are strongly connected. On the right, some complete zone are degenerated to the boundary zones.

Note, that for *W = GoStraight* algorithm the relations 'weakly/strong connected' between zones and 'connected' for points are symmetric. For other walk algorithms (e.g., for *GoCDT*) this may not be true (Figure 9).

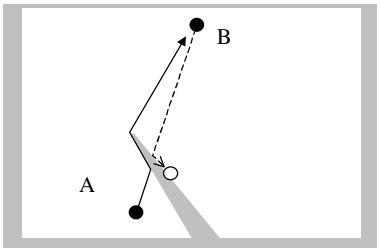

Figure 9. 'Connected' relation is not symmetric for W = GoCDT algorithm for points A and B.

In the examples above (Figure 7-Figure 9), for the universal walk algorithm W_U there is no need to subdivide the environment into zones, so the whole environment can be represented as a single node.

3.4 Navigation using maps

The upper part of the navigation system is a path planner (*PP*). This module works using the information contained in the map only. Its work is similar to finding the optimal path in the visibility graph [O'ROUR95]. Given the initial point *x* and the goal point *y*, path planner firstly invokes the association algorithm to associate *x* and *y* with the zones. Then, two additional nodes corresponding to x and y are added to accessibility graph G_M and the edges are established between them and the associated zones. Then, the optimal path is computed from *x* to *y* in the modified graph using Dijkstra algorithm.

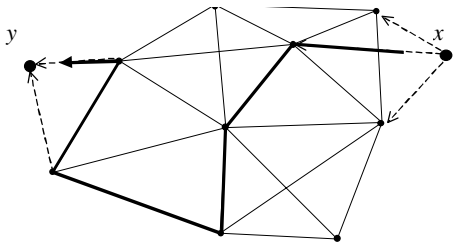

Figure 10. The scheme of the path planning algorithm

Once the path *l* from *x* to *y* is found on the map: $l = x, v_{q1}, v_{q2}, \ldots v_{qk}, y$, it is passed to the walk system (WS).

3.5 Walk system.

The walk system subsequently selects the subgoals from the path computed by the path planner and establishes them for the walk algorithm *W*. An important part of the walk system is the condition *F* that determines when it is time to select the next subgoal on the path *l*. This condition depends on the zone types and walk algorithm *W*. Clearly, if the subgoal is reached, the subgoal must be changed. Besides, the subgoal can be changed if:

- some more distant subgoal on the constructed path is reached;
- some more distant subgoal on the constructed path is visible (if for the given W visibility is equivalent to accessibility);
- agent enters the zone associated with some distant subgoal on the constructed path.

When the condition *F* evaluates to TRUE, path planner is invoked to replan the path according to the changed associations.

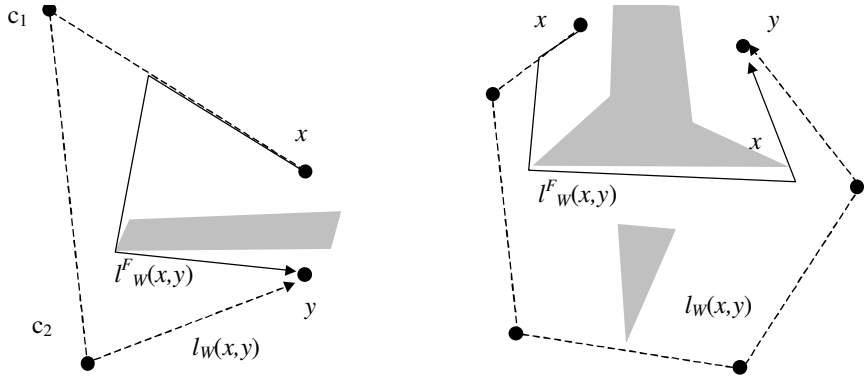

Figure 11. Illustration to the 'condition F'. W = GoStraight, visibility ⇔ accessibility. Here l_W – the planned path. The use of condition F enables to make the constructed path $l^F{}_W$ much shorter.

3.6 Maps of the environment

Definition. *Connectivity matrix (connectivity graph)* $\mathcal{M}$ of the zones $Z_1 \ldots Z_n$ is a matrix [1..n]*[1..n], where m_{ij} shows the strongest type of connectivity of zones Z_i and Z_j.

Definition. *Map of environment* is a set $M = \{\mathcal{M}, A, \{Z_i\}, PP, F, W\}$, where $\mathcal{M}, A, \{Z_i\}, PP, F, W$ are as introduced above.

According to definition, the map *M* depends on the agent locomotion capabilities (algorithm *W*, condition *F*, association algorithm *A*) and topological structure of the environment ($\{Z_i\}, \mathcal{M}$). Clearly, these properties depend on the agent's capabilities as well.

4. Construction of the maps of the environments

4.1 Basic considerations

We impose the following basic requirements on the map construction algorithm:

- algorithm must be suitable for different agent movement types (2D, ∂2D, 3D, ∂3D);
- algorithm must be independent on the actual walk algorithm *W* (*W* must be invoked as a callback);
- zones must be constructed according to the selected association algorithm.

Two different strategies of map construction can be considered. They are *analytical* and *discrete*.

To create the map analytically, the environment is subdivided into the zones geometrically. For example, triangulation, trapezoidalization, subdivision into convex or star polygons in 2D and tetrahedralization in 3D, Voronoi diagrams in 2D/3D belong to this type of algorithms of map creation. This strategy works well only when it is possible to determine analytically the structure of the zones generated using *W* and *A* algorithms.

Discrete strategy is useful if the structure of the zone defined by algorithms *W* and *A* cannot be trivially described. This always happens for powerful (non trivial) algorithms *W*. In this case it is infeasible to create a geometrical figure corresponding to a zone. Therefore, practically a zone can be determined only by the association algorithm. To make sure the zone possesses the required properties (e.g., it is complete), we discretize the environment using the grid and check these properties for the discrete points (that's why the strategy is called *discrete*). This strategy is especially effective for creation the metric maps for complex 3D environments.

4.2 Descriptive environment map construction

To create the descriptive maps of the environment, we use an incremental algorithm. Its scheme is outlined below.

Incremental descriptive map construction algorithm (IDMC)

Select any point *x* from *E* not yet covered by any zone. Here the new connected component starts to grow.

1. Using the association algorithm *A*, 'grow' the zone *Z* starting from *x* (Figure 12). Zone *Z* must satisfy some required

property (e.g., completeness) according to the walk algorithm *W*.

2. On the boundaries of the constructed zones $\{Z_i\}$, select a point *x* that at least one neighbor *y* not yet covered by any zone, and such that $L_W(x, y) < +\infty$.
3. If such a point exists, goto 2.
4. If the environment is not fully covered by the zones, goto 1.

To prevent the algorithm from cycling, the zone *Z* generated on step 2 should include at least one point not yet covered by any other zone. Clearly, IDMC algorithm constructs a descriptive map of the environment: if there exists a path from point *x* to point *y*, then this path will be found by the navigation system. The work of IDMC algorithm is illustrated on Figure 13.

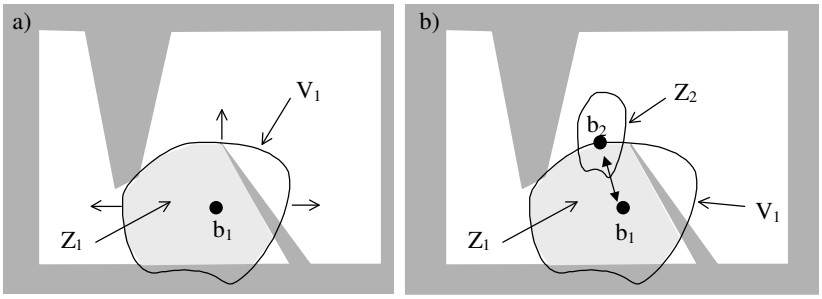

Figure 12. Illustration of the zone growth process

Usually, the growth of the zone is achieved by the growth of the bounding volume *V*. Initially, volume *V* coincides with the starting point *x* and defines the complete zone with respect to any walk algorithm *W*. Usually, volume *V* specifies the minimal bounding volume that defines the maximal complete (star) zone with the beacon in point *x*. The minimality condition ensures that the volume does not contain unnecessary points.

Obviously, the selection of the start point *x* influences the final subdivision of the environment into zones, so to create optimal maps it is important to use some heuristics. On the other hand, the proposed method for *x* selection ensures the construction of descriptive environment map.

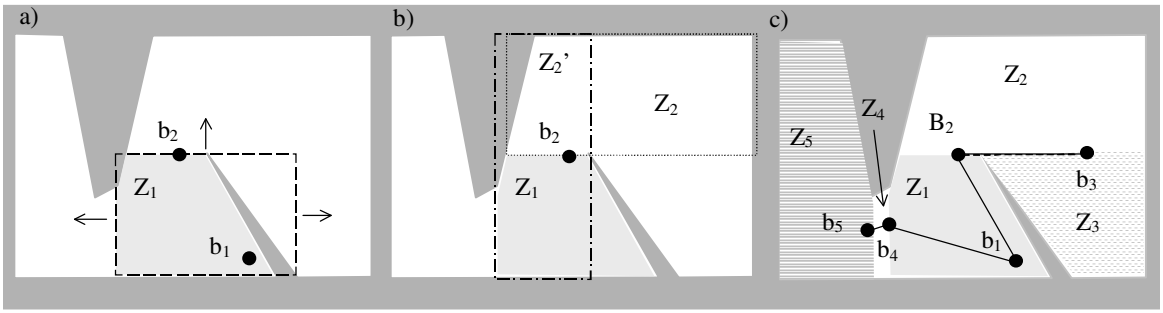

Figure 13. Illustration of the work of IDMC algorithm. On fig. b) it's possible to grow two different zones from b_2.

4.3 Metric environment map construction

In the metric environment map construction problem, a positive constant *C* is defined, specifying how worse the path found by the navigation system can be compared to the optimal path:

$$L_{W,M}(x,y) \le C * L_{opt}(x,y). \qquad (*)$$

An obvious approach to the metric environment map creation is based on the refinement of the descriptive map obtained by the IDMC algorithm. The zones created by IDMC are subdivided and new zones are added to enable the construction of better paths ensuring the satisfaction of the requirement (∗). Unfortunately, the map created by IDMC does not reflect properly the 'metric characteristics' of the environment, so some better approaches are to be sought.

The scheme of the IMMC (incremental metric map construction) algorithm is outlined below. We implement this algorithm on a discretized environment.

Incremental metric map construction algorithm (IMMC)

1. $G_M = \{\varnothing, \varnothing\}$. Initially, the map graph is empty.
2. Find points *x*, *y* such that $L^F_{W,M}(x,y) > C * L^D_{OPT}(x,y)$. If such a pair does not exist, stop. Here we build the optimal path l^D_{OPT} from *x* to *y* over the discrete grid using Dijkstra algorithm.
3. Add some points from the optimal path l^D_{OPT} to G_M (see below).
4. goto 2.

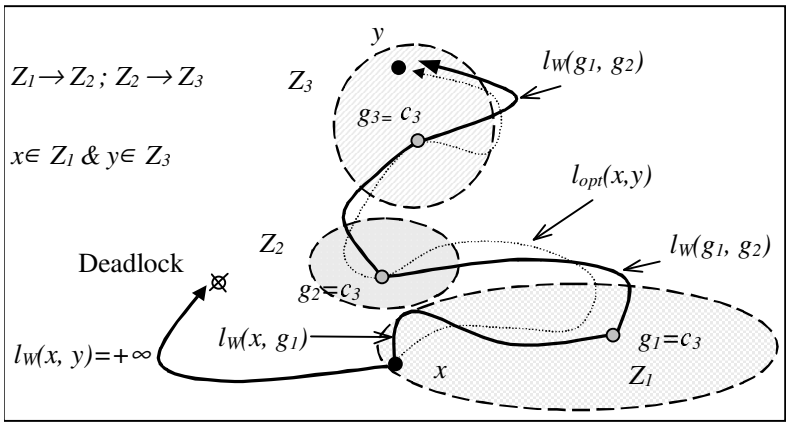

Figure 14. Illustration to the IMMC algorithm.

For example, on Figure 14 there were no zones $Z_{1\text{-}3}$ and the path from x to y was too lengthy compared to the optimal one. IMMC algorithm finds the optimal path $l^D_{OPT}(x,y)$ (dotted line), subsequently selects the points $c_{1\text{-}3}$ (centers of the future zones) near $l^D_{OPT}(x,y)$ and grows up 3 zones $Z_{1\text{-}3}$ ensuring that they are (at least) weakly connected. For actual real-time navigation, the centers of the zones become the subsequent subgoals $c_{1\text{-}3}$ when moving from x to y. Thus the optimal path is approximated by three sub-paths generated by walk algorithm W between the centers of the zones.

Conclusions.

In this paper we briefly introduced the technology of intelligent agent navigation in the complex environment. Strict mathematical treatment of the navigation problem allowed us to study the underlying problems and to develop efficient solutions.

All parts of the described 3D navigation system are practically implemented in a 3D action game currently being developed at Creat Studio. In the game, a large number of characters such as spiders, butterflies, hoppers and humans inhabit the 3D environments and successfully navigate performing the tasks of the AI system. Below we show two actual screenshots of different environments where the created maps are visualized (the maps are created for two character with quite different locomotion capabilities).

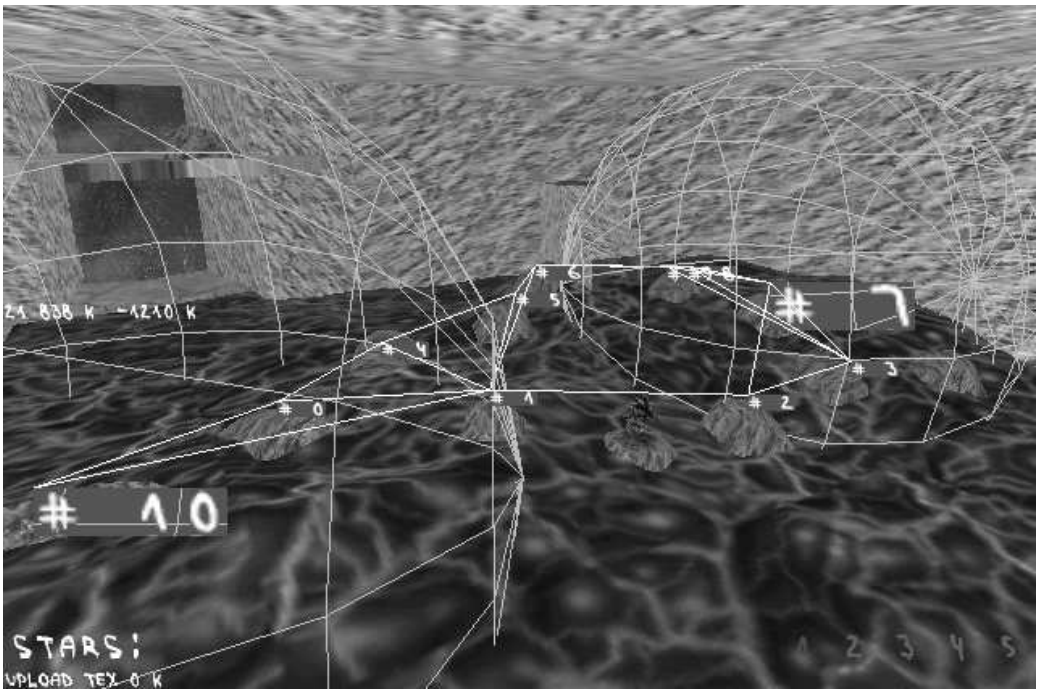

Figure 15. A map created for a 'hopper' character (W is very complicated). For zones Z_7, Z_{10} bounding volumes are shown.

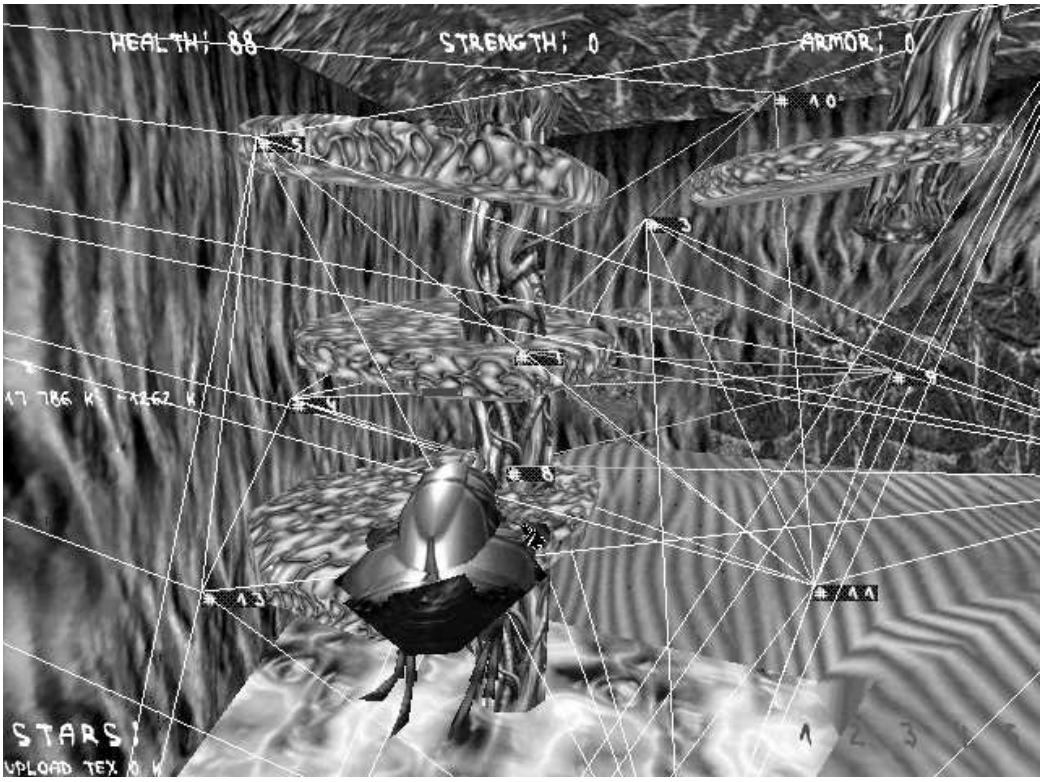

Figure 16. A part of the map created for the flying character (W = GoStraight). The discrete grid for this scene contained 672 nodes. Final graph contains 26 nodes (star zones) and 154 edges. The graph was computed in 3 minutes on Pentium-200 computer. Here, $\forall x, y: L_{W,M}(x,y) \leq 2 * L_{OPT}(x,y)$.

References.

[BRUM92] B. Brumitt, R. Coulter, A. Stentz, "Dynamic trajectory planning for a cross-country navigator", Robotics Institute, Carnegie Mellon University, 1992

[CANNY93] J. Canny, M. Lin, "An opportunistic global path planner", 1993

[SHARI97] M. Sharir: "Algorithmic motion planning", in "Handbook of Discrete and computational Geometry", Ed. by J.E. Goodman, J. O'Rourke, CRC Press, 1997.

[MITCH97] J.S.B. Mitchell, "Shortest Paths and Networks", in "Handbook of Discrete and computational Geometry", Ed. by J.E. Goodman, J. O'Rourke, CRC Press, 1997.

[RENAU95] O. Renault, N. Thalmann, D. Thalmann. "A Vision-Based Approach To Behavioral Animation." SIGGRAPH'95 Course Notes (7), pp. 5.14-5.22

[NOSER95] H. Noser, O. Renault, D. Thalmann. "Navigation for Digital Actors based on Synthetic Vision, Memory and Learning." SIGGRAPH'95 Course Notes (7), pp. 5.23-5.32

[BADL95] N. I. Badler, "Dynamic Behaviors for Real-Time Synthetic Humans", SIGGRAPH'95 Course Notes (11)

[O'ROURK95] J. O'Rourke "Computational geometry in C", Cambridge University Press, 1995

[BERG97] M. de Berg, M. van Kreveld, M. Overmars, O. Shwarzkopf, "Computational Geometry. Algorithms and Applications", Springer-Verlag, Berlin

[ZHUKO98] S. Zhukov, A.Iones, G. Kronin, "Navigation of intelligent characters in complex 3D synthetic environment in real-time applications", Proc. of 8th Central European Conference on Computer Graphics and Visualization'98, pp. 456-463 (Plzen, CZ).

[ZHUKO98a] S. Zhukov "Navigation of intelligent agents", PhD thesis, St.-Petersburg State Tech. University, 1998

[ZELEK] J. Zelek "Dynamic path planning", McGill Univ., Canada

[STENZ94] A. Stentz, "Optimal and efficient path planning for partially-known environments", Proc. of IEEE on Robotics and Automation, 1994