AUTOMATIC CONSTRUCTION OF ROADMAPS FOR PATH PLANNING IN GAMES

A. Kamphuis M. Mooijekind D. Nieuwenhuisen M.H. Overmars
Institute of Information and Computing Sciences
Utrecht University, The Netherlands
Email: {arnok,mmooijek,dennis,markov}@cs.uu.nl

KEYWORDS

Path planning; Roadmap; Probabilistic Roadmap Method; Games; Camera movement; Groups

ABSTRACT

Path planning plays an important role in many computer games. Currently the motion of entities is often planned using a combination of scripting, grid-search methods, and reactive approaches. In this paper we describe a new approach, based on a technique from robotics, that computes a roadmap of smooth, collision-free, high-quality paths. This roadmap can be used to obtain instantly good paths for entities. We also describe applications of the technique for planning the motion of groups of entities and for creating smooth camera movement through an environment.

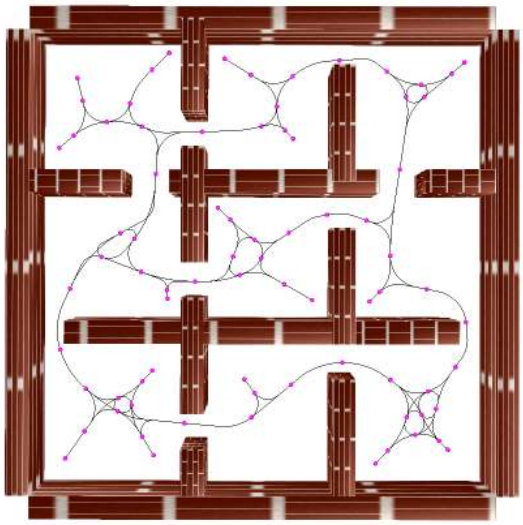


Figure 1: An example of a smooth roadmap computed by our technique.

INTRODUCTION

In many computer games, entities like enemies, NPCs, and vehicles, must plan their motions between locations in the virtual world. Currently this is typically achieved using a combination of scripting, A^*-like grid search, and local reactive methods.

In scripting the designer explicitly described the paths that can/must be followed by the entities. This is normally part of level design. Scripting is a time consuming process for the designer. Also it can lead to repetitive behavior that is easily observed by the player. Scripting gets increasingly complicated when many entities move in the same space.

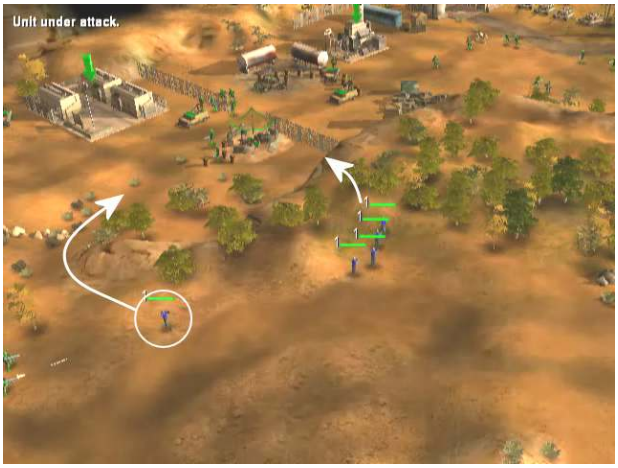
Grid based methods divide the world in a grid of cells and plan motion using an A^*-like search on the free cells (see e.g. (De-Loura 2000; Russell and Norvig 1994)). It is often used in real-time strategy games where there is a natural division of the world in cells. When the world becomes complicated and large and many entities must move around, grid based method take a large amount of computer time. Pruning the search reduces the time but might lead to wrong paths. Also, motions created by grid search tend to be unnatural, as can be observed in many RTS games.

Reactive method adapt a previously computed motion to obstacles found near the path that were not taken into account during initial planning; for example other entities or small, movable objects (see e.g. (Lamiraux and D. Bonnafous 2004; Stout 1996; Baert 2000; Pinter 2001)). Even though theoretically reactive methods can be used to compute full paths this normally leads to dead-lock situations in which the entities no longer know where to move (e.g. they get stuck in a corner of the room). Another problem with reactive methods is that it is often difficult to adapt the internal animation of the character to the motions produced.

In robotics many other path planning approaches have been developed that might be applicable to path planning in games. In robotics though the emphasis is often on the motion of a complicated robotic system in a relatively simple environment. In games the opposite is true. From a path planning perspective, the entity can often be modeled as a simple vertical cylinder, while the environment can be very complicated with tens of thousands of obstacles.

One popular path planning technique in robotics is the Probabilistic Roadmap Method (PRM). It has been studied by many authors, see e.g. (Kavraki et al. 1996; Švestka and Overmars 1998; Kavraki and Latombe 1994; Holleman and Kavraki 2000). In a preprocessing phase this method builds a roadmap of possible motions of the robot through the environment. When a particular path planning query must be solved a path is retrieved from this roadmap using a simple and fast graph search. The PRM approach is suited for very complicated environments. Unfortunately though the roadmap produced by

(a) A group of five characters should attack the site pointed to by the arrow.



(b) The group inappropriately splits up, loosing some of its troops.

Figure 2: One of the problems with the current techniques for motion planning for multiple units is that the group splits up to reach the goal. This scene was taken from Command and Conquer: Generals from EA Games.

the method can be rather wild, leading to ugly paths that require a lot of time-consuming smoothing to be useful for gaming applications.

In this paper we will describe a new path planning approach, building on the PRM method, that can be effectively applied in gaming applications. The approach also constructs a roadmap of possible motions but guarantees that the paths are short, have enough clearance from the obstacles, and are C^1 continuous, leading to natural looking motions. See Figure 1 for an example of a roadmap computed with our approach. After building the roadmap, which can be done as a preprocessing phase, paths can be retrieved almost instantaneously, and do not require any postprocessing.

Besides the standard application, in which the roadmap is used for planning the paths for individual entities, we describe two additional applications. First, we consider the motion of groups of entities. In games this problem is often solved using a combination of grid-based planning and flocking (Reynolds 1987; Reynolds 1999). Unfortunately, this can lead to unwanted behavior where the group of entities splits up (see Figure 2 for an example). We will use the smooth paths computed by the new planning approach as a backbone path and then use a social potential field approach to guide the flock through a corridor around this path (extending our earlier work in (Kamphuis and Overmars 2004)). This results in a naturally looking motion in which the group is guaranteed to stay together.

In games also the virtual camera, through which we observe the world, moves through the environment. Currently the camera is often under direct control of the user (in first person games) or under indirect control of the user (in third person games). Direct camera control is difficult, easily leads to motion sickness due to redundant motions, and is often not required. Building on our earlier work in (Nieuwenhuisen and Overmars 2004a) we describe a method in which the user only specifies positions of interest and the camera automatically moves to such positions using a smooth, collision free motion. For computing the camera path we will use the new planning approach described. This is then combined with techniques to control the view direction and the speed of the camera to obtain a camera motion that is pleasant to watch.

ROADMAP GENERATION

In this section we will describe how, in a preprocessing phase, a roadmap of possible motions for the entities can be computed. A roadmap is normally represented as a graph in which the nodes correspond to placements of the entity and the edges represent collision-free paths between these placements. A standard technique for automatic roadmap creation is the probabilistic roadmap method (PRM).

Unfortunately the PRM method leads to low quality roadmaps that can take long detours. This is due to the random nature of the PRM method. Techniques exist to improve paths in a post-processing stage but this is time-consuming and can still lead to long detours. Here we present a variant of the PRM method that leads to short, smooth and high quality roadmaps. These roadmaps can then during the game be used to solve path planning queries almost instantaneous using a simple shortest path graph search algorithm (for example Dijkstra's shortest path algorithm).

In the preprocessing phase we create the roadmap graph, consisting of vertices (V) and edges (E). For the placement of the entity we only take its position (x, y) into account since these are the only parameters that are important for planning the path. Later, the other parameters (such as orientation) can be added depending on the application. The edges of the graph will represent straight line and circular paths between the vertices. Only vertices and edges that are collision free are allowed in the graph.

In order to be able to use the graph for as many different

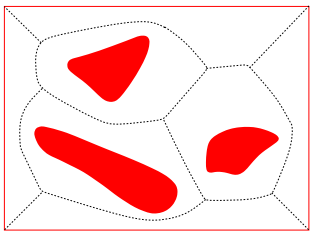


Figure 3: An example of a Voronoi diagram. We treat all four boundaries of the workspace as separate obstacles.

queries as possible, we need a good coverage of the space. Many improvements for the PRM method have been proposed in order to achieve this (see e.g. (Bohlin and Kavraki 2000; Boor et al. 1999; Nissoux et al. 1999; Wilmarth et al. 1999; Hsu et al. 2003; Branicky et al. 2001; Isto 2002)), but all are based on the same underlying concept and lead to roadmaps consisting of straight line segments only that result in low path quality.

For a roadmap that is used to steer entities in games, we can formulate the following criteria:

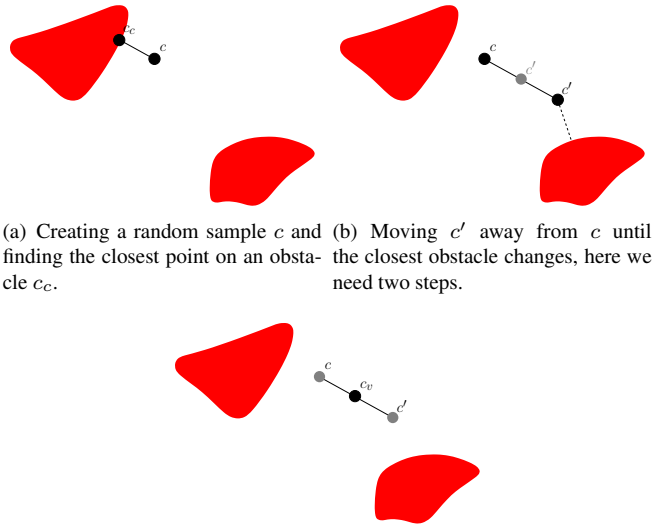
- The paths generated by the roadmap should always keep some minimum amount of clearance from the obstacles in the scene.
- The paths should be smooth i.e. it should be C^1 continuous.
- A path needs to be created very fast (not delaying the motion) and should be short, not taking any detours.

Creating Samples on the Voronoi Diagram

The Voronoi diagram of a scene defines for every obstacle a set of points in the free space that are closer to this obstacle than to any other obstacle in the scene; together these points form the Voronoi diagram (see Figure 3 for an example).

Here, we propose a new variant of the PRM method that uses the Voronoi diagram as a guide. The method works as follows. In every iteration of the algorithm we randomly pick a sample (placement) of the entity. Then we check whether this sample is collision free for the entity. If this is the case, we continue by retracting it to the Voronoi diagram using the following procedure. We calculate the closest point on an obstacle from c, we call this point c_c (Figure 4(a)). We now move another sample c' from c in the opposite direction of c_c using as a step size the distance between c and c_c (Figure 4(b)). We proceed until the closest obstacle to c' changes. We now have two samples c and c', both having another closest obstacle. The above procedure guarantees that the Voronoi diagram passes through a point between c and c'.

We continue by using binary search between c and c', with precision ϵ until we have found a sample that has two obstacles at the same distance. This sample, called c_v, is at most a



(a) Creating a random sample c and finding the closest point on an obstacle c_c.

(b) Moving c' away from c until the closest obstacle changes, here we need two steps.

(c) Finding the sample c_v that is on the Voronoi diagram using binary interpolation.

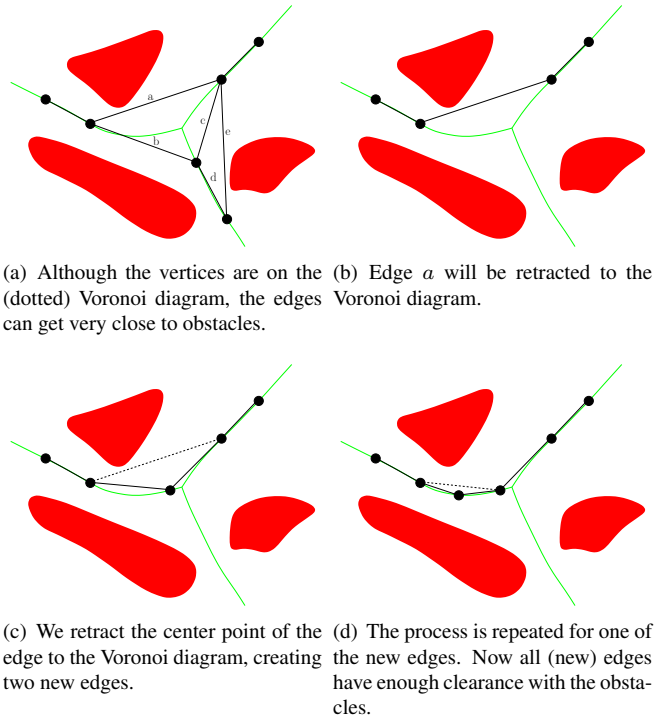
Figure 4: Retracting samples to the Voronoi diagram.

distance ϵ away from the Voronoi diagram (Figure 4(c)). Now, we add c_v to the list of vertices V in the roadmap graph.

After adding a sample as a vertex to the graph, we determine its neighbor vertices. The set of neighbor vertices of vertex v, called N_v is defined as all vertices V that are closer to v than some choosen maximum neighbor distance. For each vertex v_n in V_n we test whether the straight line connection between v and v_n is collision free. If this is the case, then we add the connection (v, v_n) as an edge to the set of edges E of the graph. If two vertices are already connected in the graph (via other vertices), then we only add the new edge if the path between v and v_n is shortened considerably by at least some constant K (for more details see (Nieuwenhuisen and Overmars 2004b)).

Retracting Edges

We have retracted the nodes of the graph to the Voronoi diagram (within a certain boundary ϵ) but when connecting the samples with edges, these edges are usually not on the Voronoi diagram and can get very close to obstacles (Figure 5(a)). In order to solve this problem, edges are retracted to the Voronoi diagram until every part of the edge is at least some prespecified distance away from the obstacles. We achieve this by proceeding in the following manner: if (a part of) an edge is too close to an obstacle, this edge is split in two equal length parts and the middle point is retracted to the Voronoi diagram using the same procedure as described above. This procedure is recursively repeated for the two new edges until every edge has enough clearance with the obstacles. An example of this procedure is shown in Figure 5. In some cases (when the edge goes through a very narrow corridor) the clearance threshold will never be reached and the edge will be split an infinite number of times. In order to prevent this, we stop retracting the edge if its length is shorter than some predefined value.



(a) Although the vertices are on the (dotted) Voronoi diagram, the edges can get very close to obstacles.

(b) Edge a will be retracted to the Voronoi diagram.

(c) We retract the center point of the edge to the Voronoi diagram, creating two new edges.

(d) The process is repeated for one of the new edges. Now all (new) edges have enough clearance with the obstacles.

Figure 5: Retracting an edge to the Voronoi diagram.

Retracting edges to the Voronoi diagram may result in some edges overlapping each other. For example in Figure 5(a) if edges c and d are already retracted,‘ then retracting e will result in overlap. Fortunately, detecting overlapping edges is easy. For every pair of edges e_i and e_j, we check how far their endpoints are away from the other edge. If this distance is smaller than some predefined distance, then we try to project the vertices of e_i on e_j and vice versa. If at least one of these projections is successful we call the two edges overlapping and we can join them. We can distinguish four different kinds of overlapping edges. In Figure 6 these are shown together with the situation after removing the overlap.

Improving the Roadmap

The roadmap graph obtained has enough clearance from the obstacles. It can though be improved further. In particular we want to make sure that it has paths running through all the corridors between obstacles. On one hand we can apply some of the known techniques for finding paths in narrow passages (see e.g. (Geraerts and Overmars 2004; Boor et al. 1999; Hsu et al. 2003)). We can also use the special structure of our roadmaps by trying to connect nodes with degree 1 to other nodes, using techniques as described in (Nieuwenhuisen and Overmars 2004b).

Another problem with our graph is that is contains many degree two vertices. This can easily be remedied by removing such vertices if the merged edge is collision free and has enough clearance.

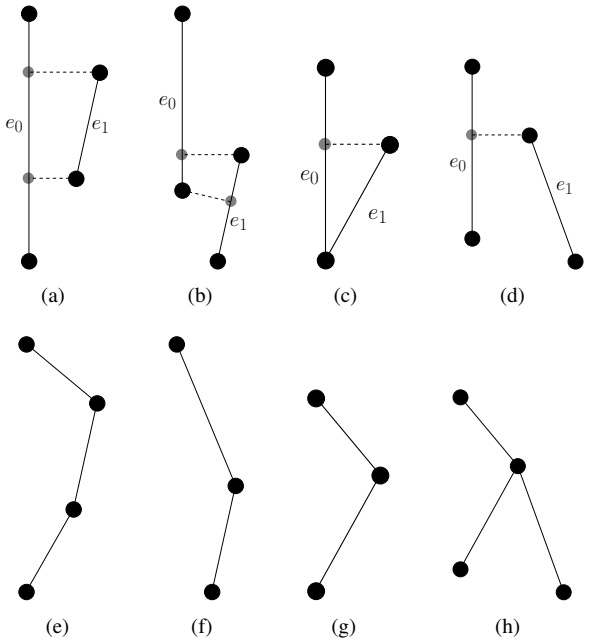


Figure 6: Merging two overlapping edges e_0 and e_1. Four different cases can be distinguished (a..d). The results after merging are shown in (e..h).

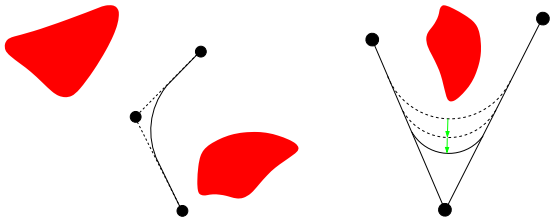
Circular Blends

After retracting the vertices of the graph to the Voronoi diagram and after adding some minimal amount of clearance to the edges we still end up with a graph that consists of straight line segments. Following such a path will have C^1 discontinuities at the vertices that cause sudden directional changes to an object that follows the path. In order to solve this problem we will replace parts of the straight line edges by circular blends.

The degree of a vertex is defined as the number of edges that is connected to this vertex. If a vertex has degree 1, it is an endpoint of a path segment, and no circular blend needs to be added. If a vertex has degree 2, the addition of the circular blend is straightforward. We create a circle arc that touches both edges and use this to replace a part of the path (see Figure 7(a)). If the degree of a vertex v is higher than 2, we need to add blends between every pair of edges. Let $e_1, ..., e_k$ be the incoming edges for the vertex v. We add a vertex v_i at the middle of each edge e_i. Now for each pair of new vertices v_i, v_j we replace the path $v_i v v_j$ with a circular blend. Finally we remove v. It is easy to show that this makes every path in the roadmap graph C^1 continous.

In the previous section we retracted the edges of the graph to the Voronoi until they had at least some predefined clearance. Adding circular blends may decrease this clearance. Since we do not want the clearance to be lower than some predefined value, we check the minimum clearance of each circular blend. If it is too low, then we replace the blend by another blend that has a smaller radius. We repeat this until the blend has enough clearance. This procedure is shown in Figure 7(b).

See Figure 1 for an example of a typical roadmap graph created with this method. Computing this roadmap took about



(a) Creating a circular blend between two edges.

(b) Moving the circular blend to increase clearance.

Figure 7: Creating the circular blends.

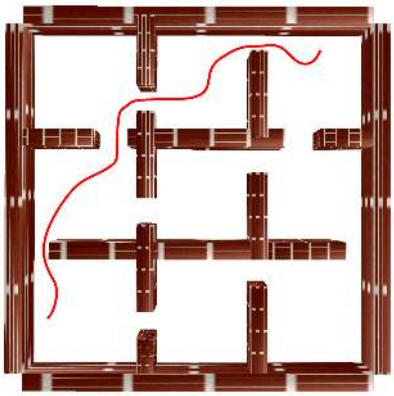


Figure 8: An example of a resulting path.

1 second on a Pentium IV 2.4 GHz. Realize though that roadmaps can be computed during the creation of the scene and can easily be stored with it.

Answering Path Queries

During the game, whenever an entity has to move to a new location, it can search the graph to plan its route through the environment. Because of the properties of the roadmap, paths will be short, have enough clearance from the obstacles, and are smooth. To obtain a path we first need to connect the current position of the entity to the graph. This can easily be done by finding the closest vertex in the graph and connecting the current placement to this vertex using circular blends. We proceed in the same way for the goal placement of the entity. Now we can use a shortest path algorithm in the graph to find the path between the start and goal positions. See Figure 8 for an example of such a path.

Computing paths during game play is extremely fast. Even for a large roadmap graph consisting of 1000 vertices and 3000 edges, the calculation of the shortest path takes less than 10ms on a Pentium IV 2.4GHz.

PATH PLANNING FOR GROUPS

Games are often populated with a large number of moving entities. The entities should often behave as a coherent group rather than as individuals. For example, one needs to simulate the behavior of whole army divisions. Current games solve the problem of path finding on the entity level, i.e. they plan the motion of individual entities, using techniques like flocking to keep the entities together. However, in cluttered environments this often leads to non-coherent groups. There is no guarantee that the entities will stay together, albeit that staying together is not well defined. Even though the entities all have a similar goal, they try to reach this goal without real coherence. This results in groups splitting up and taking different paths to the goal, for example as in Figure 2.

We will briefly describe a novel technique in which groups do stay together. More details can be found in (Kamphuis and Overmars 2004). So we are given a game level in which a group of entities must move from a given start to a given goal position. The entities must avoid collisions with the environment and with each other, and should stay together as one group. The entities are modeled as discs (or cylinders) and are assumed to move on a plane or terrain. Later, the resulting paths for the cylinders can be used to animate avatars, e.g. sprites or motion captured human-like avatars.

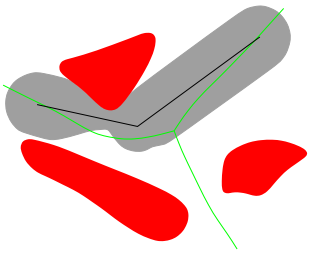
The method works as follows: First, a so-called *backbone path* for a single entity is computed. This path defines the homotopic class used by all entities. Two paths P_0 and P_1 are said to be in the same homotopic class only if P_0 can be continuously deformed into P_1 without intersecting the obstacles. Next, a *corridor* is defined around the backbone path in which all entities must stay. Finally, the movement of the entities is generated using force fields with attraction points on the backbone path. By limiting the distance between the attraction points for the different entities, coherence of the group is guaranteed.

Backbone Path Planning and the Corridor

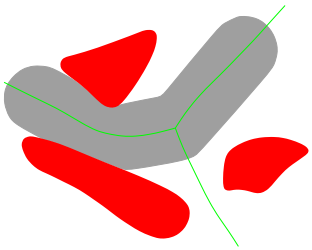
The first phase of the approach consists of finding the backbone path. Since every entity should be able to traverse the path, the clearance on the path should be bounded by a minimum value, namely the radius of the enclosing circle/cylinder of the largest entity. The backbone path can thus be defined as follows: A backbone path is a path in the 2D workspace, where the clearance at every point on the path is at least the radius of the enclosing circle/cylinder.

Although finding a path with a minimum clearance of the radius of the enclosing circle is required, we prefer a larger clearance, since a larger clearance leads to more coherent behavior. Also we prefer the paths to be smooth and short. Hence, the paths in the roadmap created with the method described above are very well suited for this application.

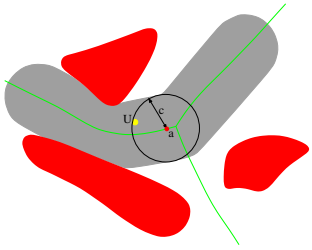
From the backbone path, a corridor is created. For this we use the clearance along the path. On every point along the path the clearance is defined as the radius of the largest circle around the point that does not intersect with the environment. The value of the clearance is upper bound by the maximum group width, i.e. the clearance used can never exceed the maximum group width. The union of all the upper-bounded clearance circles form a corridor around the backbone path. Figure 9(a) shows a path generated with a technique that produces paths



(a) A backbone path and corridor generated from a non-optimized roadmap. The path is too close to the obstacles, creating an artificial narrow passage.



(b) A backbone path and corridor generated from a roadmap generated on the medial axis. Clearly the path is optimal in the sense of clearance.



(c) The attraction point a of an entity U when point a has a clearance c.

(d) A real corridor.

Figure 9: The corridor

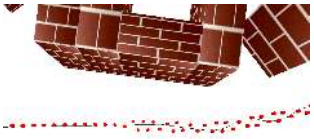
too close to obstacles, resulting in artificial narrow passages. In contrast, the path in Figure 9(b) was generated with the roadmap approach described above, and lies far from obstacles. The resulting corridor is much more natural.

Generating the Motion inside the Corridor

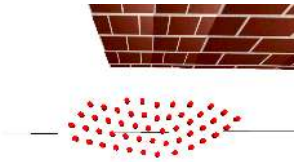
Once the corridor is created, we need to use it to generate the motion of the individual entities. The approach used is an artificial force field technique. Forces are defined that act on the entities and influence their movement. Every entity in the group has a corresponding attraction point on the backbone path. This attraction point is selected as the maximum advanced point p along the backbone such that the entity is still inside the circle centered at that point p with radius equal to the clearance at p (see Figure 9(c)). The attraction points make the entities move forward and keep the entities inside the corridor. The entities also repulse each other to avoid collisions between them. Additional forces could be incorporated, for example to accomplish formations.

Keeping Coherence in the Group

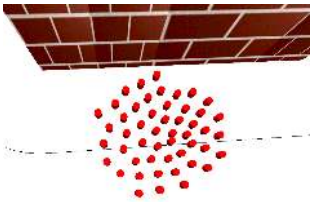
In order to keep the group coherent the dispersion should be upper bounded. Due to the manner in which the corridor is constructed, the lateral dispersion (dispersion perpendicular to the backbone path) is automatically upper bounded by the group width. However, the longitudinal dispersion (in the direction of the backbone path) is not yet bounded in this approach. To achieve this, the distance along the path from the least advanced attraction point to the most advanced attraction point is limited. This results in the entities in front waiting for the entities at the back.



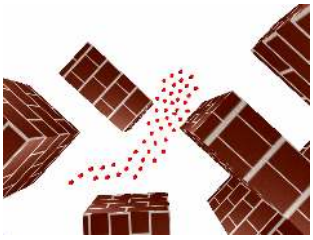
(a) The group with high longitudinal dispersion and very low lateral dispersion.



(b) The group with medium longitudinal dispersion and medium lateral dispersion.



(c) The group with low longitudinal dispersion and larger lateral dispersion.



(d) The group moves through a narrow passage, making the first wait for the last ones (movement is from the lower-left to the upper-right).

Figure 10: A group of 50 entities moving in a virtual world. These paths and behaviors are created with the same approach, only by varying the parameters.

Results

The behavior of the group can be controlled by adjusting the coherences parameters, *lateral dispersion* and *longitudinal dispersion*. Figures 10(a) to 10(c) show a group of 50 entities moving through an environment. In these pictures the lateral and longitudinal dispersion is varied, resulting in a longer, more stretched group (10(a)) or more compact group (10(c)).

Figure 10(d) shows the same group moving through the environment from the left lower corner to the right upper corner. The most advanced entities, i.e. the entities that passed the narrow passage earliest, wait for the last entity to pass the passage.

We tested the performance of the approach to show that the technique is usable in real-time applications as computer games. For this, we developed a typical implementation. In this implementation we created numerous paths. The processor usage to create the paths was very minimal. For groups of 50 to 100 entities the processor usage did not exceed 1 percent. More efficient implementations could further decrease the processor usage.

PLANNING CAMERA MOTIONS

Every game has a camera through which the player views the world. Usually this camera moves depending on user input and place of action. A camera motion directs the camera from one position to another while controlling camera speed and view direction. There are many situations in which an automatic camera motion in a game could be of great use. Think for example about an RPG where the user has almost completed a level, but wants to get back to the start of that level to pick something up she forgot. Without automatic camera motions, the attention of the user is needed to walk the whole way back just to get one item. Also in many adventures it would be nice to be able to specify a location in the interface and have the game create a fast and efficient motion to that location without warping directly to it (causing the user to get disoriented).

The roadmap from the previous section can easily be used to steer a camera. Using this roadmap, the camera is guaranteed to keep a certain amount of clearance from the obstacles and the circular arcs make sure that the camera motion is gentle. The roadmap alone however is not enough to create a smooth camera motion. Camera theory (Millerson 1973; Wayne 1997) shows that we need to take care of two more variables. First the speed of the camera should be adapted according to the curvature of the path. Otherwise, objects will move too fast through the view. Secondly, the user should get cues about where the camera is going. In particular the viewer should be able to anticipate a camera rotation. We will resolve these issues in the next two sections.

Adapting the Camera Speed

Smoothness of the path is not enough for a smooth camera motion. The speed of the camera along the path should be adapted according to the curvature of the path. Also there should be a maximum acceleration and deceleration for the camera in order to prevent too abrupt speed changes.

Since our path consists of straight lines and circle arcs, we can adapt the speed of our camera by making use of the radius of the arcs. The smaller the radius, the lower the camera speed. When the camera leaves an arc with a small radius, we accelerate until we have reached the maximum speed of the current arc or straight line. If, on the other hand, the next arc requires a lower speed than the current camera speed, we must start decelerating before we reach this arc such that when we reach the next arc, our speed is sufficiently low. A speed diagram can be computed efficiently that satisfies both the constraints on the maximal speed for each arc and the bounds on acceleration and deceleration. See Figure 11 for an example of such a speed diagram for a simple path.

Smoothing the Viewing Direction

Intuitively one might think that the viewing direction should be equal to the direction of the camera motion. As stated before however, the user should be given cues about where the camera is heading to. We can achieve this by always looking at the position the camera will be in a short time. Experiments show that about 1 second is the right amount. Note that, as we fix the time we look ahead, the distance we look ahead changes depending on the speed of the camera. This is exactly what we want to achieve as in sharp turns we want to look at a nearer point than in wide turns. Looking ahead has another important effect. If we would look in the direction of motion and the camera reaches a circular arc, then it suddenly starts rotating at the start of the arc. Stated more formally, the rotation of the camera is only C^0 continuous. It can be proved that looking ahead solves this issue by making the camera rotation C^1 continuous.

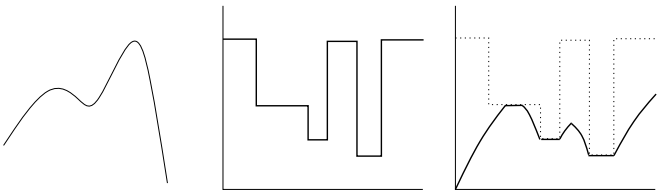


Figure 11: A speed diagram. The left image shows the path, the middle image shows the maximal speed allowed at each point. The right image shows the actual speed, taking acceleration and deceleration bounds into account.

Figure 12: An implementation of the techniques. The user can click on a location in the map at the left top and the program creates a smooth camera motion to that location.

Results

We implemented this approach in a walk-through system for virtual worlds. See Figure 12 for a screenshot. Rather than letting the user steer the camera directly, we display a map in the top left corner. By clicking on the map the user indicates the position she wants to move to. A smooth camera motion is then calculated in the way described above. The processor time required for this is minimal. Experiments indicate that this is a pleasant way to inspect the environment.

CONCLUSIONS

In this paper we have described a new technique for automatically constructing high-quality roadmaps in virtual environments and we have shown how these can be used the plan the motion for individual entities, groups of entities, and the camera through which we observe the world.

We described our method as a 2-dimensional approach in which entities move on a ground plane. It is though easy to extend it to e.g. terrains and even movement in buildings in which the roadmap would automatically follow the corridors and stairs.

Roadmap construction is best seen as being part of the construction of the virtual world. It is easy to incorporate special requirements from the level designer. For example, the designer can add fake obstacles to force the path to e.g. stay on the sidewalks of a road. Also the designer can easily manipulate the roadmap graph by manually adding, changing, or removing nodes. Moreover, weights can be added to the graph to e.g. indicate preferred routes.

ACKNOWLEDGEMENTS

This research was supported by the Dutch Organization for Scientific Research (N.W.O.). This research was also supported by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2001-39250 (MOVIE - Motion Planning in Virtual Environments).

REFERENCES

Baert, S. (2000). Motion planning using potential fields. *gamedev.net*. url: http://www.gamedev.net.

Bohlin, R. and L. Kavraki (2000). Path planning using lazy prm. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 521–528.

Boor, V., M. Overmars, and A. van der Stappen (1999). The gaussian sampling strategy for probabilistic roadmap planners. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 1018–1023.

Branicky, M., S. Lavalle, K. Olson, and L. Yang (2001). Quasi randomized path planning. In *Proc. IEEE Int. Conf. on Robotics and Automation*.

DeLoura, M. (Ed.) (2000). *Game Programming Gems 1*. Charles River Media.

Geraerts, R. and M. Overmars (2004). A comparative study of probabilistic roadmap planners. In *Algorithmic Foundations of Robotics V, Springer Tracts in Advanced Robotics 7*, pp. 43–57. Springer-Verlag Berlin Heidelberg.

Holleman, C. and L. Kavraki (2000). A framework for using the workspace medial axis in prm planners. In *Proc. IEEE Int. Conf. on Robotics and Automation*, Volume 2, pp. 1408–1413.

Hsu, D., T. Jiang, J. Reif, and Z. Sun (2003). The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Proc. IEEE Int. Conf. on Robotics and Automation*.

Isto, P. (2002). Constructing probabilistic roadmaps with powerful local planning and path optimization. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 2323–2328.

Kamphuis, A. and M. H. Overmars (2004, August). Finding paths for coherent groups using clearance. In *Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2004)*, pp. to appear.

Kavraki, L. and J.-C. Latombe (1994). Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 2138–2139. IEEE Press, San Diego, CA.

Kavraki, L., P. Švestka, J.-C. Latombe, and M. Overmars (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation 12*, 556–580.

Lamiraux, F. and O. L. D. Bonnafous (2004). Reactive path deformation for nonholonomic mobile robots. In *IEEE Transactions on Robotics*, pp. to appear.

Millerson, G. (1973). *TV Camera Operation*. Focal Press, London. ISBN: 0-24050-850-5.

Nieuwenhuisen, D. and M. Overmars (2004a). Motion planning for camera movements. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 3870–3876. IEEE Press, San Diego, CA.

Nieuwenhuisen, D. and M. Overmars (2004b). Useful cycles in probabilistic roadmap graphs. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 446–452. IEEE Press, San Diego, CA.

Nissoux, C., T. Siméon, and J.-P. Laumond (1999). Visibility based probabilistic roadmaps. In *Proc. IEEE Int. Conf. on Intelligent Robots and Systems*, pp. 1316–1321.

Pinter, M. (2001, March). Toward more realistic pathfinding. *gamasutra.com*. url: http://www.gamasutra.com.

Reynolds, C. (1987). Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics 21*(4), 25–34.

Reynolds, C. (1999). Steering behaviors for autonomous characters. In *Game Developers Conference*.

Russell, S. and P. Norvig (1994). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Stout, W. (1996, October). Smart moves: Intelligent pathfinding. *Game Developer*.

Švestka, P. and M. Overmars (1998). Coordinated path planning for multiple robots. *Robotics and Autonomous Systems 23*, 125–152.

Wayne, M. (1997). *Theorising Video Practice*. Lawrence and Wishart, London. ISBN: 0-85315-827-4.

Wilmarth, S., N. Amato, and P. Stiller (1999). Maprm: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 1024–1031.