A navigation graph for real-time crowd animation on multilayered and uneven terrain

Julien Pettre
National Institute for Research in Compute…
105 PUBLICATIONS **1,180** CITATIONS
SEE PROFILE

Jean-Paul Laumond
Laboratoire d'Analyse et d'Architecture des…
223 PUBLICATIONS **6,921** CITATIONS
SEE PROFILE

Daniel Thalmann
École Polytechnique Fédérale de Lausanne
1,001 PUBLICATIONS **15,620** CITATIONS
SEE PROFILE

A navigation graph for real-time crowd animation on multilayered and uneven terrain

Julien Pettré^1, Jean-Paul Laumond^2 and Daniel Thalmann^1

EPFL - VRlab^1
CH-1015 Lausanne, Switzerland
Telephone: +41 21 693 52 17, Fax: + 41 21 693 53 28
email: julien.pettre@epfl.ch, daniel.thalmann@epfl.ch

RIA - LAAS/CNRS^2
7,avenue du Colonel Roche, 31077 Toulouse cedex 4, France
Telephone: +33 5 61 33 63 47, Fax: +33 5 61 33 64 55
email: jpl@laas.fr

Abstract
This paper addresses real-time navigation for crowds moving on multilayered and uneven terrains. The objective is to animate thousands of characters in real-time on large-scale environments without taking care of motion details: a common goal being given to a homogeneous crowd, the proposed method automatically breaks the homogeneity of the crowd by spreading characters on various routes reaching the goal; the animation effect is displayed at a view point that does not require to address character interaction locally. Our approach is based on a space structuring technique that automatically decomposes multilayered or uneven terrains into navigation corridors giving rise to a navigation graph supporting the search.

Keywords
real-time crowds, path planning, navigation, navigation graph, uneven, multi-floored, multilayered terrain

1 Introduction

Computers power increase recently allowed to populate interactive virtual worlds with numerous inhabitants. Crowds are now common in movies and more and more in video games, especially in real-time strategy games. Simulation of real-time virtual crowds is still a difficult challenge given that available computation-time is mainly dedicated to rendering; a need for fast simulation techniques exists. In this paper, we focus on the navigation problem.

Recent approaches [SKG05, LK05] considered the navigation problem as a pure motion planning one, based respectively on Probabilistic Roadmaps [KSLO96] and on a A^*-search planner. In these approaches, a locomotion trajectory is calculated successively for each character, that is then considered as a moving obstacle for the resting planning operations (pedestrians inter-collision avoidance is thus guaranteed). These solutions are efficient and produce high-quality motions. However, they are limited by the possible size of the crowd. Indeed, planning loop is invoked for each pedestrians and the search space becomes more and more constrained given that pedestrians become progressively moving obstacles (once their own motion is planned). Our approach differs: in our point of view, pedestrians inter-collision is a *local* problem that should be solved only in the user's focus area, but elsewhere, people should be scattered in the environment to get believable crowd motion. How to get a variety of paths with similar initial and final conditions?

Previous approaches used mainly interactions between crowd characters to result in behaviors variousness. In [BG96] particle systems are used to simulate crowds. People are controlled by high-level directives while their local behaviors (obstacles and people avoidance) result from a sum of attractive or repulsive forces analogous to physical electric ones. In [HFV00] Helbing also introduces a solution based on Physics-laws. Reynolds in [Rey87] demonstrated that a simple perception model of the rounding world and a small set of local rules (match velocity, avoid collisions, move towards the center of the flock) were efficient to simulate aggregate motions of a flock of boids in a believable way. Hodgins et al. also use a perception model as inputs of the behavioral model presented in [BH97].

More elaborated behavioral models allow to simulate crowds of passengers in railway stations like described in [ST05], and more generally, pedestrians in virtual cities (note that semantic environments are required). Also, complex behavioral models are time-consuming and the possible number of people composing crowds is limited. In [UT02], Ulicny proposes a scalable behavioral model to palliate this limitation. From the geometric description of the environment, Lamarche [LD04] proposes an automatic extraction of the topology, that is stored into a graph answering to navigation queries. The method is limited to flat terrains.

Our objectives are close to the ones of this last approach. From the geometric definition of any environment, we capture automatically its topology into a graph structure. The navigation graphs solve user's queries interactively such as: selecting characters and affecting new goals, creating people flows between pairs of locations, or creating sets of attractive areas in-between which people navigate.

Our contribution is to address at the same time:

- large-scale crowd navigation made of thousands of characters,
- unstructured environments including multilayered and uneven terrains,
- real-time computation for interactivity,

The method is based on an automatic space structuring introduced in Section 2 and giving rise to a navigation graph. The navigation graph is then used for path planning: the originality of the approach is to compute a set of non homotopic itineraries that supports character spreading (Section 3). Results and performances of the method are presented in Section 4, before Discussion and Conclusion.

2 A space structuring based navigation graph

The general idea of our approach may be introduced from the case of flat terrain navigation. In such a case the terrain is structured into obstacles to be avoided and free-space. We first compute the Voronoï diagram of the free-space. Then we build a set of collision-free convex cells along the Voronoï diagram. The adjacency graph of the cells defines the so-called navigation graph.

The idea is not new. Various algorithms exist from this simple case [Lat91]. Here the challenge is to extend the idea to multilayered and uneven terrains. In such cases terrains are no more obviously decomposed into obstacles and free-space. Some parts of the terrains should be classified into obstacles or free-space according to the slope angle. We should then use dedicated techniques for space structuring. That problem has been already addressed in the framework of motion planning for robots moving on rough terrains [HST99, Che99]. The space structuring method we use is inspired of such works. It consists in first computing an elevation map from the geometric data structures modelling the terrain. The elevation map consists in a 3D grid whose points belong to the terrain. Two adjacent points within the grid are connected if the slope of the associated segment (with respect to an horizontal plane) is less than some user-defined threshold. Such a connecting rule gives rise to a 3D navigation grid (Section 2.1). Then a distance grid is computed as well as the associated medial axis (Section 2.2). Finally navigation corridors are computed along the medial axis and they are structured into a navigation graph (Section 2.3).

2.1 Elevation map and Navigation grid

The *elevation map* is a 3D grid whose points refer to positions on the considered terrain where virtual humans are able to stand (i.e. having enough free space above). The map is computed from: the environment mesh, the maximum characters' height H_c and a user defined precision (corresponding to the grid unit). The key-idea is to use environment views from above and the OpenGL Z-Buffer to retrieve all the possible elevations of the terrain for each horizontal position. Then, we filter the elevation map to erase points where humans are not able to stand. Finally, a *navigation grid* is deduced by checking connections between adjacent elevation map points.

The grid precision is interactively defined without need for expertise: to do so, users resize the view-port of an above-view of the environment. The ideal view-port size is the smallest one allowing to display all the environment details such as narrow passages or small obstacles. The resulting view-port size is the one used for next operations.

OpengGL is initialized in an orthographic projection mode, environment being seen from above. Two horizontal clipping planes are also initialized, oppositely oriented and separated with a distance H_c. These planes allow to cut the environment into slices whose height is H_c, and to display them successively by setting the planes altitude correctly. After each slice display, drawn pixels coordinates (screen coord. and depth) are transformed into world coordinates, and stored as new elevation map points.

The resulting elevation map identifies all the possible altitudes of the terrain for any horizontal position, except some. Indeed, some areas may be occluded in slices displays. As slices height is H_c, occlusion implies that the free space above occluded areas is insufficient to allow characters to stand: occluded pixels are thus understandably ignored. However, we still have to filter superposed elevation map points having a too small vertical distance separating them at the end of the process (checking is required between points belonging to different slices).

We now check for connections between elevation map points. The elevation map points and their connectivity compose our *navigation grid*. Two elevation map points are connected if:

- adjacent (adjacency is limited to the four neighbors along the two main horizontal axis),
- the declivity between the points is limited by a user defined threshold,
- no vertical obstacle stands between the points.

Checking for vertical obstacles is required because the elevation map has been computed from above-views of the environment where vertical faces of objects are invisible. To do so, we use again orthographic views of vertical slices (whose depth is the grid unit) of the environment seen from the sides (e.g. left and front views).

If side-views do not reveal presence of obstacle and if the declivity criteria is respected, adjacent grid-points get connected. As a result, each point may have a maximum of 4 connections (one for each of its 4 neighbors). Points have less than 4 connections when close to obstacles or presenting high declivities. Such points compose the border of the navigation grid and model implicitly the non-navigable areas. From this observation, we deduce automatically *distance grid* (Section below) and next, the *navigation graph*.

2.2 Distance to border points and medial axis

Our next goal is to evaluate the clearance around each point of the previously computed navigation grid. To do so, we associate a *distance grid* to the navigation grid. The distance grid provides the distance between any navigation grid point and the nearest border grid point. Considered point and related border point should belong to the same layer. Let's take the example of a road passing under a bridge. When a point belonging to the road is considered, it's clearance is the distance to the road border and the bridge should not be taken into account (and reversely).

To compute the distance grid, we successively start from each border point and propagate to connected points. Turn-backs are forbidden, so that the propagation does not move to superposed layers. Distance between visited points and the current origin border point is calculated. Visited points store the minimal distance calculated (by comparison with successive propagations from other border points). Note that in order to increase the process efficiency, the propagation is stopped where the visited point has already a smaller minimal distance to another border point.

Finally we identify distance grid points belonging to medial axis. Medial axis are interesting in our case because they correspond to itineraries providing a maximum clearance to borders (and implicitly to obstacles and non-admissible slopes). Medial axis can be compared to the Voronoï cell borders, however slightly different in our case due to the multilayered structure of surfaces.

2.3 Navigation Graph

From the previous distance grid, we now deduce our *navigation graph*. The computed medial-axis and clearance allow to deduce a set of connected areas, which are obstacle-free and flat enough for navigation.

From each navigation grid point and the associated clearance, we can guarantee navigability into a cylindric area, whose height is H_c, and whose ray equals the clearance, and whose caps follow the terrain. Such a modeled area are our *navigation graph vertices* Figure 1, left image.

Now let us consider pairs of overlapping cylindric areas. Any point belonging to the first area can be joined to any other point belonging to the second area following a path passing by any point lying in their intersection. Consequently, *navigation graph edges* connect pairs of vertices that correspond to overlapping cylindric areas (Figure 1, right image). Geometrically, edges are represented as vertical rectangular gates. To deduce gates coordinates, we compute the intersection points of 2 circles (vertical projection of the cylinders on an horizontal plane). The edge cost is the distance between their center.

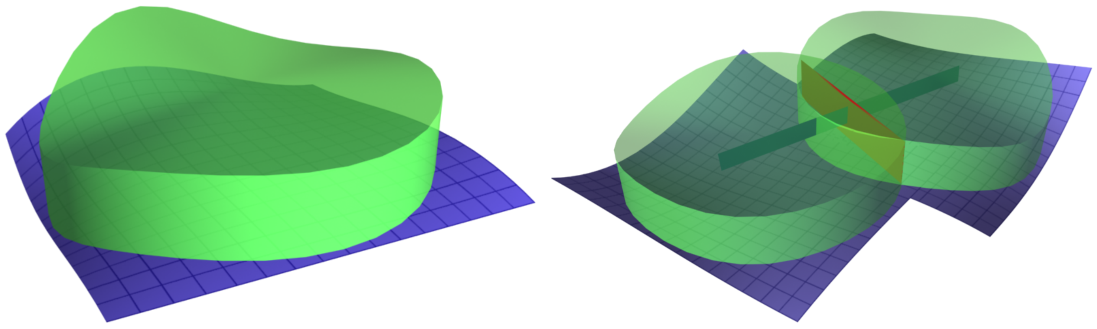

Figure 1: Representation of the Navigation Graph Vertices and Edges

As mentioned in previous Section, points belonging to medial-axis points are the most interesting to deduce graph vertices. Indeed, they all have locally a maximum clearance to borders. However, considering the totality of these points results in dense and redundant navigation graphs (lowering efficiency for solving navigation queries).

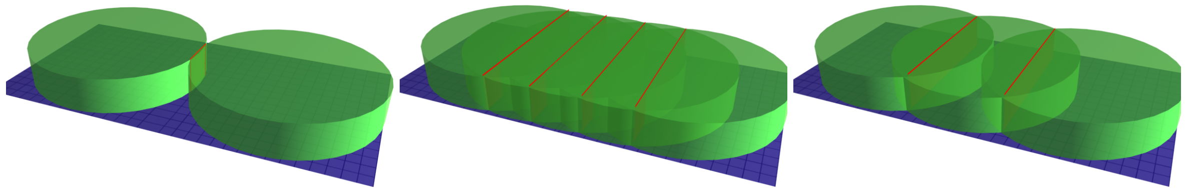

Figure 2: Vertices placement results from a tradeoff between graph complexity and coverage quality. From left to right: lowest complexity, best coverage, and chosen tradeoff

As illustrated on the bottom images of Figure 2, the subset of points from which graph vertices are deduced results from a tradeoff between graph complexity and coverage quality (the chosen tradeoff is illustrated on the right image). We use the following algorithm to extract automatically a subset of grid-points from which navigation graph vertices are built:

1. select the distance grid point having maximum clearance (this point implicitly belong to medial-axis),
2. create a vertex from the selected point,
3. disable points englobed by the resulting cylinder,
4. choose the next available medial-axis point having the maximum clearance, and loop to step 2.

Using the property enounced above in this section, people can navigate from gate to gate by with respect to the graph structure, i.e. by following contiguous set of edges. Thus, gates delimit navigable corridors in the environment. Next section describes how navigation graphs, gates and corridors are used to answer to user's queries with a variety of solutions.

3 Planning for crowds based on navigation graphs

Path planning based on navigation graphs is reduced to a graph search problem. The solution for navigating from an area to another is captured by the graph (if existing) and can be found using any graph search algorithm. The resulting solution path, that is a succession of edges to follow, can be transformed into an itinerary by selecting way points in the resulting corridors. However, our goal is to bring solution variety to the path planning problem, so that people sharing the same goal get spread on individual itineraries.

Solution variety is obtained at two distinct levels: first, a specific graph search method provides several solution paths belonging to different homotopy classes (Section 3.1). Then, each of these paths can be continuously declined in different sets of way-points (Section 3.2), corresponding to different itineraries belonging to a same homotopy class.

Roughly speaking, in our case, two itineraries belong to the same homotopy class if they can be continuously transformed one into another without crossing obstacles or high declivities (transformation space is limited to

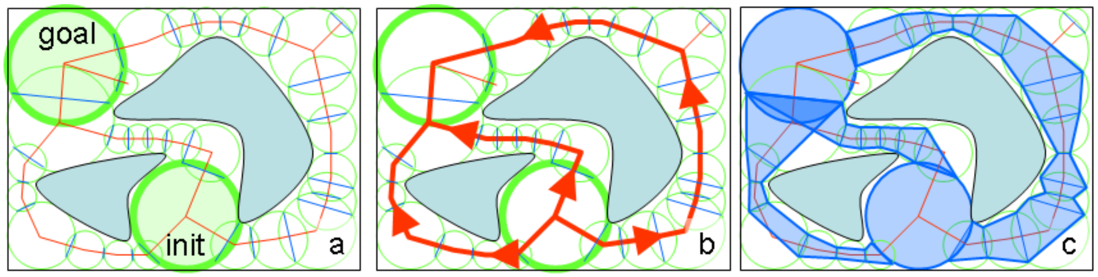

Figure 3: The solution paths variety is obtained in 2 steps: first, non-homotopic paths are found, second, way-points may be selected anywhere in the resulting corridors

navigable surfaces). For example, in a virtual city, itineraries in identical streets belong to the same homotopy class, and at the opposite, if a building separates them, they belong to different homotopy classes.

The two next Sections describe first, how non-homotopic paths are found, and second, how each is declined into homotopic itineraries.

3.1 Dijkstra based path search

Given an initial and a final area to join, our objective is to retrieve several solution paths from the navigation graph. To do so, we use an iterative Dijkstra-based path search technique. The principle is the following: first, the shortest path search is found. Second, some of the edges composing the solution path are (temporarily) removed from the graph. Finally, the processus is reiterated and stops when the initial and the final vertices get disconnected.

Parameters control the edge removal stage and allow to drive the process. We experimentally observed that:

- playing with the percentage of edges to remove controls the variousness between to successive paths found (intuitive parameter),
- playing with the percentage of edges to keep at path extremities avoid quick disconnections and influences the total number of paths found,
- furthering removal of edges linking vertices having few connections also preserve connectivity,
- furthering removal of edges linking vertices having numerous connections increases path spread, but reduces the number of paths found.

As mentioned before, each path among the several ones found is a succession of edges, i.e. a succession of rectangular gates to cross, that delimit corridors as illustrated on Figure 3. As obstacles separate each set of corridors, paths belong to different homotopy classes. Now, the corridors width is exploited to compute individualized way-points sets for each pedestrian.

3.2 Individualized way-points

For each character, one of the previously computed path is selected and transformed into a way-points sequence. As many way-points as the number of gates to cross are generated. An individual parameter p ranging from 0 to 1 determines whether a character crosses gates on the left ($p \in [0, 0.33]$), on the middle ($p \in [0.33, 0.66]$) or on the right ($p \in [0.66, 1]$). Doing so, we avoid homotopic itineraries intercrosses.

Consequently, we get a continuous declination of paths into individualized way-points sequences. For a given path, this declination generates itineraries belonging to a unique homotopy class (because no obstacle separates them). An example of such a declination is illustrated on the right image of Figure 3.

Steering method allowing to join successive way-points is not a main issue here, however, let us mention that we use a set of different methods to ensure scalability. Depending on the distance between the virtual character and the user's point of view, we synthesize linear paths or smooth paths [Bou05]. We do not synthesize paths for invisible characters, but maintain a progression parameter to ensure crowd coherency. Each character parameterize steering methods with an individual set of values (e.g. maximum locomotion velocity).

Next Section illustrates some navigation examples.

4 Results

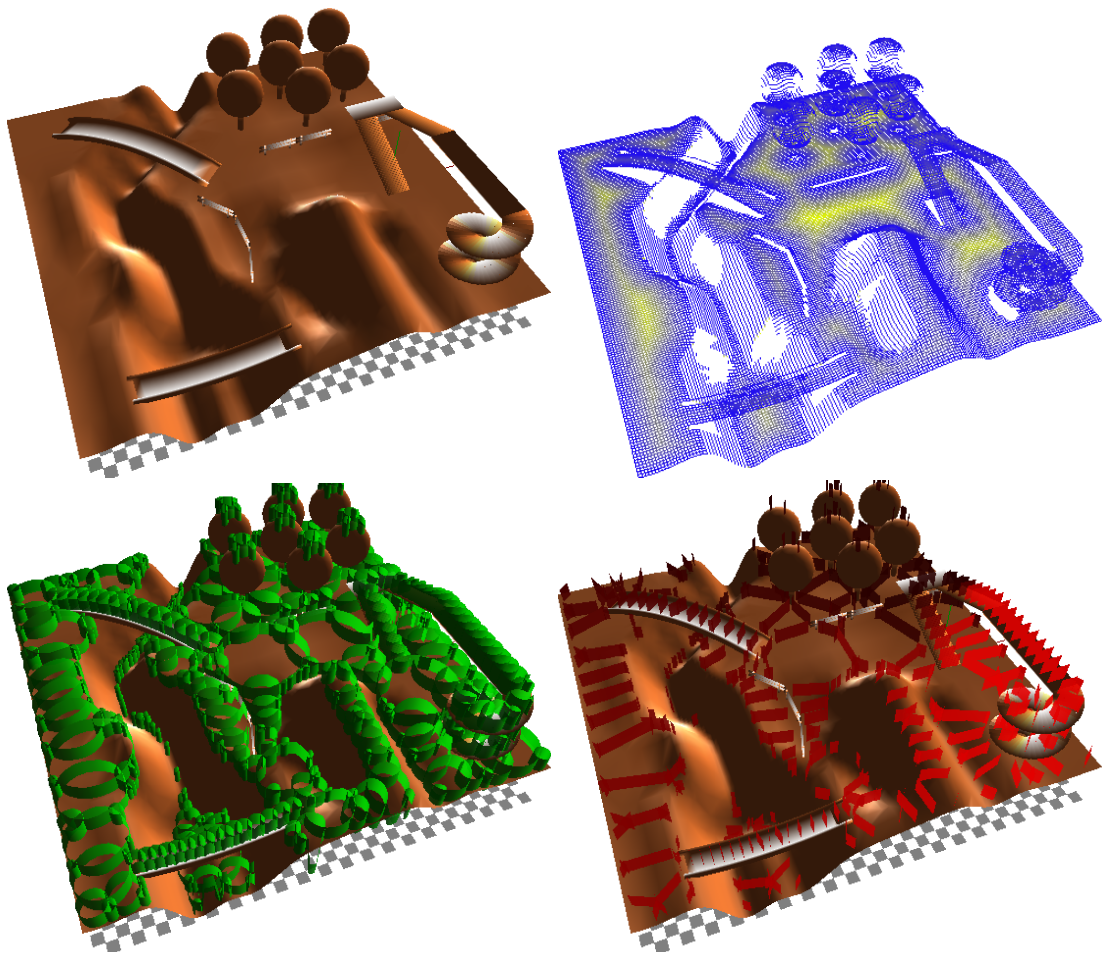

Figure 4: Considered environment(top-left), resulting navigation and distance grids (top-right), and deduced navigation graph (bottom)

We tested our method on the environment introduced by Figure 4, top-left image. Environment size is 50×50 $units$ and is composed of an uneven surface, 7 trees, wooden barriers, 2 bent bridges over a small canyon, and a planar footbridge linked to the ground by a spiral stair and a L-stair.

The top-right image of Figure 4 displays the computed navigation grid. We used a 0.2 $unit$ precision and a 2 $units$ maximum character's height (H_c) in this example. Navigation grid identifies the too sloping zones: in the canyon or on the sides of the hill, one can observe that grid points have missing connections to neighbors. Also, under bridges or stairs, where pedestrians cannot stand, no point has been inserted into the grid. Distance to the nearest border point is color-coded: blue is for small distances whereas yellow points ar farer to obstacles and slopes.

The two bottom images of Figure 4 represent vertices (left) and edges (right) of the deduced navigation graph. Vertices are represented as cylinders whereas edges are represented as vertical gates as explained before in Section 2.3. The graph is made of 474 vertices and 582 edges, distributed in several connected components. The largest component covers correctly the area where pedestrians are expected to walk. The navigation graph is computed from the environment in a minute on a common Pentium IV 1.5 GHz, 1 Mo RAM desktop station.

Figure 5 illustrates two examples of navigation plans. In the first example, a crowd is asked to go from the left side of the environment to the planar footbridge. In the second example, the crowd moves down from the footbridge up to the top of the hill.

Top-left image of Figure 5 illustrates the solution paths to the first problem: different successions of gates and corridors leading to the goal are displayed. Several solutions are available: first, pedestrian may choose between one of the two possible bridges to move towards the center of the environment. Then, pedestrians may use the L-stair as well as the spiral stair to get upon the footbridge. The multilayered structure of the spiral stairs (parts of the solution path are superposed) is perfectly handled.

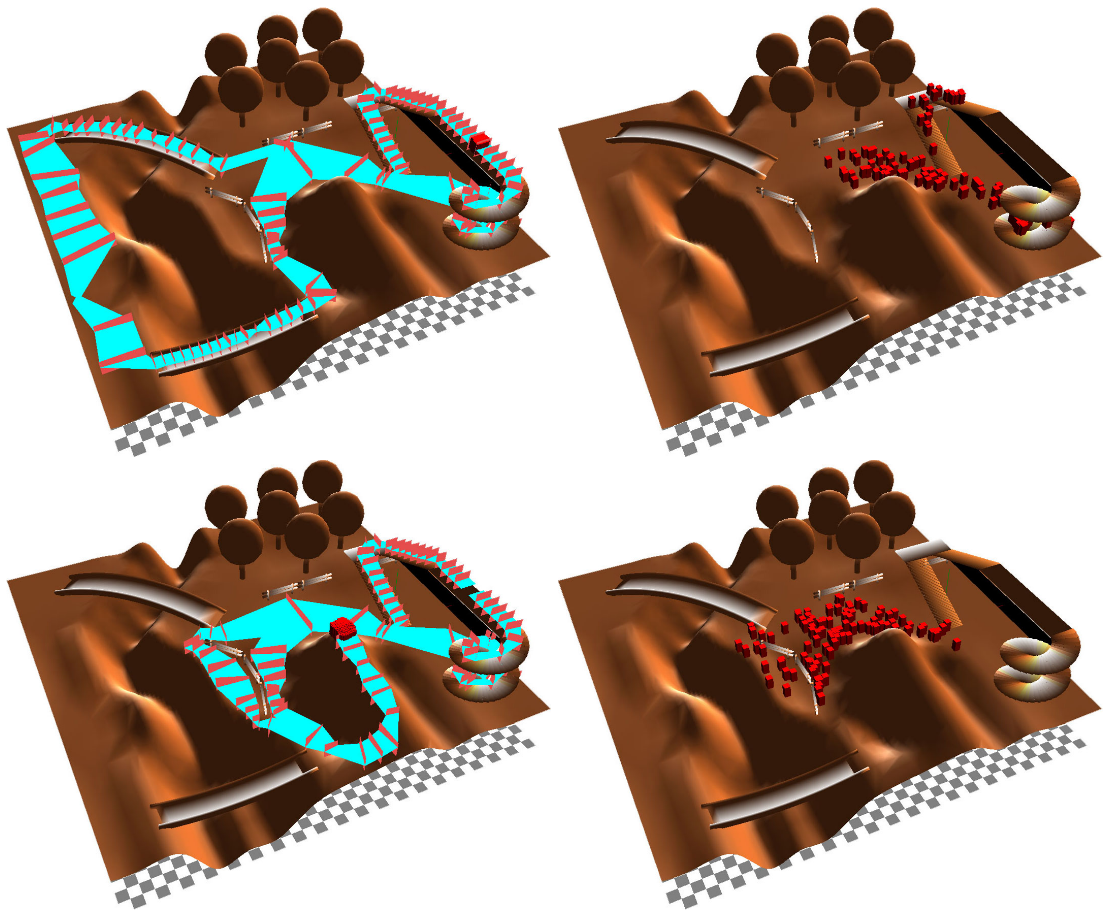
Figure 5: Navigation results

Pedestrians are allowed to walk anywhere inside corridors. On the top-right image, we can see a 100 pedestrians crowd following the navigation plan. Each pedestrian individually chooses one of the available paths, and transforms it into a way-points sequence. As a result, the crowd get spread both on paths and inside corridors.

The second navigation plan (Figure 5, bottom-left) illustrates the solution ability to handle uneven terrains. The declivity of the right side of the hill is too high and people have to skirt round to get on the top. To get there, pedestrians may choose to go down from the footbridge by one of the 2 available stairs, and to pass on the left or right side of the barrier nearby the hill.

In these examples, the path search computation time is $3ms$ per navigation query, and paths transformation into way-points sequences consumes $0.36ms$ per character. The path search complexity is $O(PElogV)$, where P is the number of non-homotopic paths found, E the number of edges and V the number of vertices. Transformation of paths into way-points is linear with the number of characters in the crowd and the number of edges composing solution paths. The memory usage is low (the graph occupies $1MB$ in our case, whereas the whole data structure, i.e. crowd agents, environment geometry, paths, way-points, etc... occupies $10MB$).

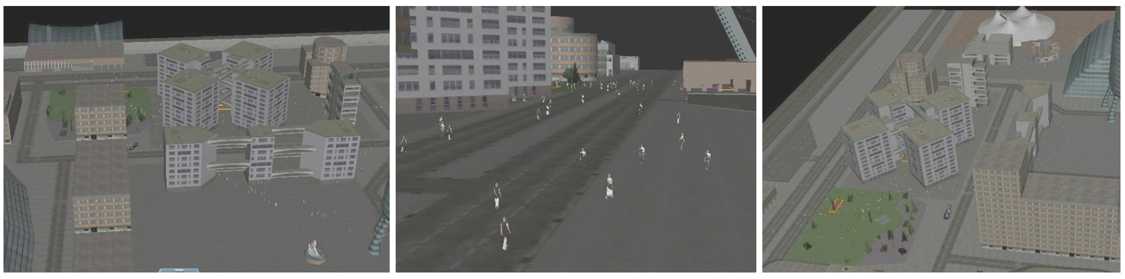
Figure 6: Navigation in a virtual city

Technically, we developed a stand-alone program for computing navigation graphs, whereas the navigation functions are gathered in a C-library. As a result, our navigation solution can be invoked by other software platforms. Figure 6 illustrates our solution operating in VHD++, a virtual reality software platform [PPM$^+$03]. A crowd is moving in a town where several attractive places are defined: a railway station, an hotel, a circus, etc... Pedestrians move randomly between attractive areas.

5 Discussion

Compared to previous approaches, the main originality of our solution is the absence of interactions between pedestrians to drive their navigation. Collision detection and avoidance (between pedestrians) was probably the main bottleneck in crowds simulation. However, this stage was necessary to obtain behaviors variety and to get believable motions, as mentioned in Introduction. We started from the observation that solving collisions is required by some applications, such as safety-related ones, but is not a matter for many others (such as entertainment).

Our solution provides eye-believable motions without interaction, and we observed that collisions occurring in our simulations are undetectable when far from the point of view. We agree that collisions should be solved locally, in the spectator's focus area, for a reduced number of pedestrians. Elsewhere, in the background, the aspect of the crowd motion is what the spectator *expects* to see. Populating virtual places with large-size crowds now becomes feasible using our solution and above all, crowds are interactive and controllable.

Navigation graph structures the environment but also the crowd. Indeed, each pedestrian knows in which graph vertex he is actually moving, and reversely each vertex references which pedestrians are actually navigating in the corresponding area. These relationships between environment and crowd provided by the graph are yet unexploited in our simulations, but work is in progress.

For example, Figure 5 displays only a 100 pedestrians crowd moving along the planned paths, so that the people spread remains visible on figures. Results are still believable with a 1000 pedestrians crowd, and solution remains interactive for tenth of thousands, but pedestrians continue to move ignoring each others in areas that become overcrowded. To palliate this limitation, instantaneous local population density can be computed efficiently for each area. Local population density then influences pedestrians' velocity and distribution on paths. First results are promising.

Finally, future works directions concern simulation scalability. As mentioned in Section 3.2, the steering method for reaching way-points is already scalable. Each stage of our solution should also scalable. Depending on the visibility of the considered pedestrians, navigation queries should be solved with a different accuracy. This should allow us to address larger and more complex environment, as well as larger crowds.

6 Conclusion

We presented a versatile, robust and efficient solution for real-time crowds navigation in environments combining uneven and multilayered surfaces. Our solution provides variety of solutions to the path planning problem so that each character can reach goals in a unique way, and crowds are spread on the whole environment while moving. The solution fits many applications, especially for the entertainment industry where absence of collisions check is not necessarily a matter.

Firsts results are promising, and we aim at using the developed method as a basis for a more elaborated crowd simulation architecture. Our future directions are first to equip the method with inter-collisions avoidance technique, and to exploit available implicit information to implement smart navigation strategies, to simulate formation and aggregation of people flows, and to influence individual behaviors.

References

[BG96] E. Bouvier and P. Guilloteau. Crowd simulation in immersive space management. In *Proc. of the Eurographics workshop on Virtual environments and scientific visualization '96*, pages 104–110, 1996.

[BH97] D. Brogan and J. Hodgins. Group behaviors for systems with significant dynamics. In *Autonomous Robots*, volume 4, pages 137–153, 1997.

[Bou05] R. Boulic. Proactive steering toward oriented targets. *Eurographics Short Papers*, 2005.

[Che99] M. Cherif. Motion planning for all-terrain vehicles: a physical modeling approach for coping with dynamic and contact interaction constaints. *IEEE transactions on Robotics and Automation (ICRA)*, 1999.

[HFV00] D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature*, 407:487–490, 2000.

[HST99] A. Hait, T. Siméon, and M. Taïx. Algorithms for rough terrain trajectory planning. *Advanced Robotics*, 14(6), 1999.

[KSLO96] L.E. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Tr. on Robotics and Automation*, 12(4):566–580, 1996.

[Lat91] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[LD04] F. Lamarche and S. Donikian. Crowds of virtual humans : a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum*, 2004.

[LK05] M. Lau and J. Kuffner. Behavior planning for character animation. In *ACM SIGGRAPH / EUROGRAPHICS Symposium on Computer Animation (SCA'05)*, 2005.

[PPM+03] M. Ponder, G. Papagiannakis, T. Molet, N. Magnenat-Thalmann, and D. Thalmann. Vhd++ development framework: Towards extendible, component based vr/ar simulation engine featuring advanced virtual character technologies. *Comupter Graphics International (CGI)*, 2003.

[Rey87] C.W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *ACM Computer Graphics*, 21(4):25–34, jul 1987.

[SKG05] M. Sung, L. Kovar, and M. Gleicher. Fast and accurate goal-directed motion synthesis for crowds. In *ACM SIGGRAPH / EUROGRAPHICS Symposium on Computer Animation (SCA'05)*, 2005.

[ST05] W. Shao and D. Terzopulos. Autonomous pedestrians. In *ACM SIGGRAPH / EUROGRAPHICS Symposium on Computer Animation (SCA'05)*, 2005.

[UT02] B. Ulicny and D. Thalmann. Towards interactive real-time crowd behavior simulation. *Computer Graphics Forum*, 21(4):767–775, dec 2002.