EUROGRAPHICS 2004 / M.-P. Cani and M. Slater
(Guest Editors)

Volume 23 (2004), Number 3

Crowd of Virtual Humans: a New Approach for Real Time Navigation in Complex and Structured Environments

F. Lamarche[†] and S. Donikian[‡]

IRISA, Campus de Beaulieu, F-35042 Rennes, FRANCE
{fabrice.lamarche|donikian}@irisa.fr

Abstract

The navigation activity is an every day practice for any human being capable of locomotion. Our objective in this work is to reproduce this crucial human activity inside virtual environments. Putting together the high complexity of a realistic environment such as a city, a big amount of virtual humans and the real-time constraint requires to optimize each aspect of the animation process. In this paper, we present a suitable topological structuring of the geometric environment to allow fast path finding as well as an efficient reactive navigation algorithm for virtual humans evolving inside a crowd.

1. Introduction

The autonomy of a virtual human is defined by its capacity to perceive, act and decide of its actions. The behaviour is usually described through several simple skills that can be mixed to generate a more complex and credible behaviour. One of the most important skills is the ability to navigate inside a virtual environment as it is part of a large number of behaviours. Reproducing this fundamental behaviour requires to address different topics such as the topological model of the environment, path planning and collision avoidance techniques. In order to credibly animate several hundreds of pedestrians in real-time, each of these techniques should be optimized without leaving out behavioural studies. In this article, we propose a general model, inspired by studies on human behaviour to simulate the navigation process inside indoor and outdoor environments. This model is compounded of four parts:

- a spatial subdivision algorithm detecting bottlenecks inside the environment;
- a hierarchical path planning algorithm based on the abstraction and generalisation of topological properties extracted from the spatial subdivision;
- an efficient structure computing neighbourhood relations between entities;
- a general and modular algorithm which handles reactive navigation and includes visual optimization of the trajectory and collision avoidance. The human behaviour is configured through complementary modules describing rules inspired by psychological studies.

Related works are presented in the next section, including the presentation of characteristics of the pedestrian behaviour. Section 3 presents the spatial subdivision algorithm and the hierarchical path-planning algorithm. Section 4 describes the neighbourhood structure and the reactive navigation architecture based on studies on pedestrian behaviour. Finally, section 5 gives some results and benchmarks.

2. Related works

2.1. Spatial subdivision and path planning

Path planning and environment representation have been widely studied in the field of robotics where navigation is a necessary task to achieve [Lat91]. In the field of behavioural animation, similar methods are used. Three general approaches can be distinguished : roadmaps, cell decomposition and potential fields.

The **roadmap** approach consists in computing a network of standardized paths (lines, curves) passing through free

† University of Rennes I
‡ CNRS

spaces. Different approaches are used to compute roadmaps. The visibility graph [ACF01] connects together vertices of the environment if and only if they see each other. The computation of the Voronoi diagram inside free spaces allows to use generated edges to produce the roadmap. The **cell decomposition** method consists of decomposing free spaces into cells. Once this decomposition is computed, a connectivity graph can be extracted, whose nodes are cells and edges traduce cells adjacency. Two general methods can be distinguished: the exact cell decomposition consists in computing cells such as their union is exactly the free space (constrained Delaunay triangulation, convex polygons, trapezoidal), and the approximate cell decomposition consists in using predefined cell shapes, whose union is strictly included in the free space (uniform grids, quadtrees) [BT98, Kuf98]. An interesting discussion, outlining the interest of generating abstract cells on the top of uniform grids, can be found in [TB96]. This solution provides a great increase of performance for path finding computation as the number of cells is smaller. In the **potential field** method, the environment is discretized into a fine regular grid. A potential is associated to each cell which corresponds to the sum of a repulsive potential generated by the obstacles of the environment and an attractive potential generated by the goal. Thus, gradient methods can be applied to find a path to the goal. But this method is subject to local minima problems and does not necessary reach the goal. To recover from local minimas, some randomized methods have been studied [KKL96].

2.2. Reactive navigation

A spatial subdivision of the environment is not sufficient to handle navigation as several moving entities can populate the same environment. In that case, a system allowing dynamic collision avoidance is necessary to achieve consistency and realism. Several approaches can be distinguished such as particle systems, flocking and behavioural systems. Those techniques differ essentially by the number of simulated entities, their level of control and the associated collision detection method.

Particle systems are physically based simulations defining attractive and repulsive forces associated to obstacles and simulated entities. Forces applied to the entity are added in order to determine the new direction of the entity [RKBB94, HFV00, BMdOB03]. Flocks are rule based systems defining the behaviour of an entity in function of the behaviour of the nearest entities [Rey00, BLA02]. Loscos et al. [LMM03] use a fine regular grid to handle reactive navigation and to store information about pedestrian movements enabling the emergence of flows of pedestrians. Ulicny et al. [UT02] use a layered approach to model the individual behaviour inside a crowd by combining rules and finite state machines. Those types of systems raise the problem of nearest neighbour queries which is one of the bottlenecks on the number of possible simulated entities. Several approaches have been proposed to optimize those requests using spatial data structures such as bin-lattice [Rey00], K-d trees [O'H00] or Kinetic Data Structures [GKM*01]. Methods based on the exploitation of an informed environment have been developed [TD00, FBT99]. This way, some specific behaviours related to the type of the entity and the navigated area [HK02] have been modelled. Some studies have also focused on crowd simulation and its levels of autonomy [MT97] in order to provide a realistic crowd behaviour inside virtual environment. Complementary works have been performed on the optimization of the real time visualization of crowds by using hierarchical impostors [O'H02] and real-time shading of impostors [TLC02]. In order to increase the realism of animation, Ashida et al. [ALA*01] made a statistical analysis of pedestrians walking along a section of sidewalk. They exhibit subconcious actions, that they integrated into the animation system with a stochastic process to control their activation.

2.3. Pedestrian behaviour

Goffman [Gof71] describes techniques used by pedestrians to avoid bumping into each other. The social link between strangers is characterized by silence and indifference [RQ98] and to perform that, different behaviours are used. The first technique called externalization concerns the way that people are constantly making others aware of their intentions in order to minimize the interaction. Lee et al. [LW92] show that pedestrians are using social conventions such as driving rules to let other people easily predict their normal trajectory. The second technique called scanning is used by pedestrians to selectively gather externalized information from other people. The third technique is called the minimization of adjustment which expresses that people adjust their trajectory several meters before the conflict to make it perceptible early by others with the objective to reduce interaction and avoid coordination. Goffman introduces the notion of the oval security region whose front distance corresponds to an anticipation area depending on the pedestrian speed, while the width is the accepted gap to pass beside a person or an obstacle or to follow a wall. He defines also the law of minimal change which means that a pedestrian will try in its journey to reduce the amount and the amplitude of turns.

Hillier et al. [HPH*93] show that the majority of human-pedestrian movement occurs along lines of sight, that they named as axial lines. A. Turner et al. [TP02] propose the EVA system based on a visibility graph, compare results of this agent-based simulator with real data on the Tate Britain Gallery and conclude that they were able to reproduce the aggregate movement with a good correlation. M. Relieu [RQ98] introduces the notion of urban discrimination which means that the pedestrian focuses his attention inside his current region to select pertinent information relevant of the activity he is engaged in.

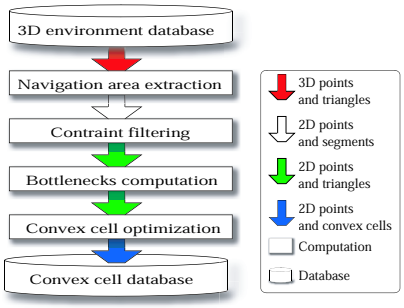

Figure 1: *Computation steps of spatial subdivision.*

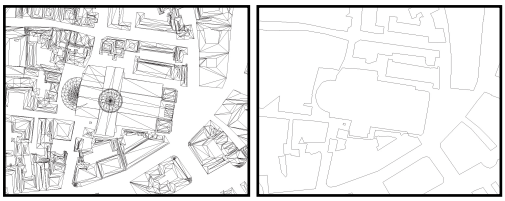

Figure 2: *Example of a map extraction from 3D database.*

3. From environment to path finding

3.1. Spatial subdivision

Our spatial subdivision model is presented in fig. 1. It is compounded of several computation steps starting from the 3D geometric database and generating a 2D spatial subdivision using convex cells and identifying bottlenecks.

2D map extraction

The first step converts the 3D geometric database of the environment into a 2D map containing all constraints delimiting obstacles under the assumption that the environment is flat. It consists in cutting the database with two parallel planes corresponding to the floor and a cutting plane whose distance to the floor is generally equal to the height of a humanoid. This extracts all geometry belonging to the navigation area. This geometry is then projected on the XY plane in order to compute a 2D map representing the environment. In order to organize this information and to simplify the constraints, a constrained Delaunay triangulation is computed [KBT03], resulting in a first spatial subdivision using triangular cells. A connectivity graph is extracted from this triangulation and a transitive closure is computed starting from a user selected cell in order to extract the navigation area. Constraints are then filtered in order to extract those delimiting this area while removing the other ones and filtered in order to merge colinear segments (with a given threshold). The example of fig. 2 presents different steps of the 2D map extraction on a part of the 3D model of a city: the projection of the geometry on the XY plane and the map extracted after constraint filtering and simplification.

Minimal distance between corners and walls

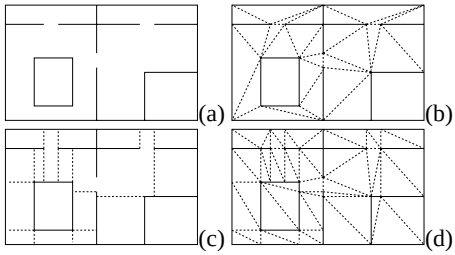

Figure 3: *(a) 2d map of environment. (b) Original constrained Delaunay triangulation. (c) Computed shortest distances between corners and walls. (d) Constrained Delaunay triangulation with shortest distances.*

The constraints previously extracted delimit the navigation area. But an information is still missing for the navigation inside the environment: bottlenecks. Those bottlenecks characterize the minimal distance between corners and walls. Without this information, it is difficult to ensure that a given humanoid can navigate from one point to another. To detect bottlenecks, we propose an algorithm based on a modification of the constrained Delaunay triangulation algorithm. In the following, we will consider two types of segments and two types of points inside the triangulation:

- $\mathcal{C}_s$: is the set of constrained segments.
- $\mathcal{F}_s$: is the set of free segments.
- $\mathcal{C}_p$: is the set of points extracted from the environment
- $\mathcal{D}_p$: is the set of points generated for the purpose of minimal distance computation between walls and corners.

First, a constrained Delaunay triangulation of the segments in $\mathcal{C}_s$ is computed. Then, for each triangle (A,B,C) of the triangulation, if $(BC \in \mathcal{C}_s) \wedge (AC \in \mathcal{F}_s) \wedge (AB \in \mathcal{C}_s \cup \mathcal{F}_s) \wedge (A \in \mathcal{C}_p)$ and if the orthogonal projection P_A of A lies on segment BC then the segment BC is removed and replaced by segments BP_A and CP_A which are added in $\mathcal{C}_s$ and P_A is added in $\mathcal{D}_p$. Triangulation is locally recomputed in order to take those modifications into account. This process is repeated until no more triangle satisfies the condition of the rule. An example of generated shortest distances is shown in fig. 3(c). The fact that points belonging to $\mathcal{D}_p$ can not be reprojected ensures the convergence of the algorithm. This computation generates a triangulation containing shortest distances between corners and walls (Cf. fig. 3(d)). This identification of bottlenecks ensures that if the width of an entity is smaller than the length of a free segment, the entity can pass through the segment without colliding with walls.

Convex cell optimisation

The constrained Delaunay triangulation computed during the previous step constitutes a first spatial subdivision using triangular cells. In order to simplify this subdivision and to minimize the number of cells, an algorithm merges triangles in order to generate convex cells while locally conserving bottleneck information. This algorithm first sorts all free segments, based on their decreasing length. The sorted

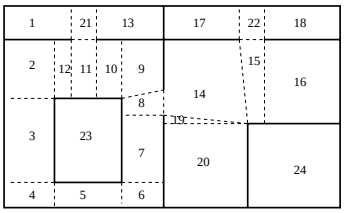

Figure 4: *The subdivision of the environment of fig. 3*

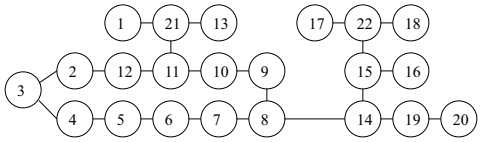

Figure 5: *Graph extracted from spatial subdivision of fig. 4.*

list is iterated and the two cells sharing the current segment are merged if and only if the resulting cell is convex and the length of the shared free segment is greater than the length of all free segments delimiting the resulting cell. The resulting subdivision is compounded of convex cells and identifies bottlenecks. An example of convex cell optimisation can be found in fig. 4. This spatial subdivision accurately maps the environment geometry and the extraction of bottlenecks automatically identifies the most constrained parts of this environment. Thanks to this information, accessibility between adjacent cells can be filtered before any path planning computation by using the humanoid width.

3.2. Topology and abstraction

Once the convex cell subdivision is computed, a graph containing topological relations is extracted. A node of this graph is a convex cell and an edge represents a free segment shared by to adjacent cells with a length greater than the width of the humanoid. The figure 5 represents the graph extracted from the spatial subdivision presented in figure 4. Each node c of the graph can be topologically qualified according to the number of connected edges given by the $arity(c)$ function:

- if $arity(c) = 0$ then c is a **closed** cell.
- if $arity(c) = 1$ then c is a **dead end** cell.
- if $arity(c) = 2$ then c is a **passage** cell.
- if $arity(c) > 2$ then c is a **crossroads** cell.

This information enables the topological abstraction of the environment. For example, a sequence of **passage** cells can be interpreted at a certain level of abstraction as a unique passage. Thus, when planning, the geometric information related to the low level cells can be omitted and summarized in a higher abstract level. The main idea of the abstraction algorithm is to generate an abstraction tree by merging interconnected cells while trying to preserve topological properties. When merging several cells into a single one, the composition of cells is stored in a tree structure in order to generate the abstraction tree. Before explaining the algorithm, some functions have to be defined. Let note $p(c) = arity(c) * (arity(c) - 1)$ the number of paths traversing the cell c. Let note $add(c_1, c_2) = p(c_1 \cup c_2) - p(c_1) - p(c_2)$ the number of added paths when merging cells c_1 and c_2 and removing all shared boundaries. The abstract levels are computed as follow:

1. All **dead end** and **passage** cells are extracted. Then all sequences of interconnected cells are abstracted in order to generate a balanced binary tree. Thus, each sequence of cells is reduced to a unique abstract cell.
2. All **dead end** cells are merged with their adjacent **crossroads** cells. If this pass generates new **dead end** cells, the algorithm returns to step 1.
3. All **passage** cells are merged with one of their adjacent **crossroads** cells.
4. For all pairs (c_1, c_2) of **crossroads** cells, the number of added paths $add(c_1, c_2)$ is computed. Let m be the minimum number of added paths. All pairs of cells such as $add(c_1, c_2) = m$ are merged to create a new abstract cell. If this step generates new **dead end** or **passage** cells, the algorithm returns to step 1. Otherwise, this step is repeated until the obtention of a unique **closed** cell.

The proposed algorithm removes dead ends and linear paths. This suppression of dead ends is a very good property as they are often responsible for the worst computation time. Moreover, the use of the *add* function during step 4 tends to reduce the number of traversals for a given abstract cell compounded of two **crossroads** cells.

3.3. Hierarchical path planning

Thanks to the spatial subdivision, it is possible to automatically generate a roadmap enabling path-planning. But inside large and complex environments, this roadmap can also be large and then reduce the performances of path finding algorithms. But the parallel exploitation of the roadmap and the topological abstraction enables a drastic reduction of the path planning graph size, resulting in real-time path finding computation inside large and complex environments.

Roadmap generation and abstraction

As cells are convex, there always exists linear paths traversing each cell and connecting all free segments belonging to the boundaries. For each cell, key points are generated on free edges and are connected with linear paths (Cf. fig. 6). Let suppose that k key points are generated on each of the n free segments belonging to the boundaries of a given cell c. The number of generated paths inside this cell is $n(n-1)k^2 = p(c)k^2$. Each cell is then informed with its associated paths. Moreover, in accordance with the topological abstraction, all paths traversing abstract cells are precomputed and stored using references to the sub-precomputed paths in order to limit the amount of needed memory.

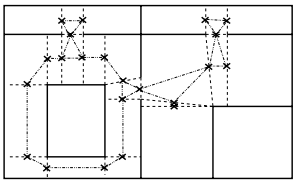

Figure 6: *A roadmap generated using 1 key point by free segment in the environment of fig. 4*

Path Finding

During the previous step, a data structure containing the topological abstraction and the associated roadmap abstraction has been generated. It is used to compute the minimal topological graph needed to extract the minimal roadmap enabling path planning. Let note C_e the cell containing the entity, C_g the cell containing the goal, C_{top} the top level abstract cell containing C_e and C_g. Before explaining the algorithm, two functions need to be introduced:

- $pickElement(S)$: it removes an element from the set S and returns this element.
- $split(C)$: it returns a set containing the lower level cells or abstract cells compounding the abstract cell C.

In the following algorithm, S^{result} will contain all cells necessary for path planning computation:

$S^{result} = S^{result}_{tmp} = \emptyset$; $S^{explore} = C_{top}$
while $S^{explore} \neq \emptyset$ **do**
  while $S^{explore} \neq \emptyset$ **do**
    $c = pickElement(S^{explore})$
    if $(C_e \in c \wedge C_e \neq c) \vee (C_g \in c \wedge C_g \neq c)$ **then**
      $S^{explore}_{tmp} = S^{explore}_{tmp} \cup split(c)$
    else
      $S^{result} = S^{result} \cup \{c\}$
    end if
  end while
  $S^{explore} = S^{explore}_{tmp}$; $S^{explore}_{tmp} = \emptyset$
end while

The set S^{result} contains cells C_e, C_g and several abstract cells corresponding to abstract paths connecting those two cells. All precomputed paths associated to the cells of S^{result} are extracted and used to compute the path-planning graph in which nodes are key points and edges are paths linking those key points. In order to compute the path, start and goal points are added in the graph and connected to all key points generated on the boundaries of their respective cells. The generated path, that is partially abstract, is then materialized using stored information about the composition of precomputed paths. The resulting path is then furnished as a sequence of free segments instead of a sequence of key points. In the next section, we will show that this feature enables visual path optimization that is a characteristic of human navigation.

The graph is minimal in respect with the topological abstraction and is drastically smaller than the original non abstracted roadmap. It contains a maximum of two dead ends (if C_g and C_e belongs to a dead end) ; thus it removes the worst case for path-planning algorithms. Moreover, the number of available paths to the goal is reduced thanks to the use of precomputed paths associated to abstract cells. The impact on path finding computation time and complexity is logarithmic (see the result section for benchmarks). This property enables real-time path finding for several virtual humans within large environments, enabling this key feature for real-time applications.

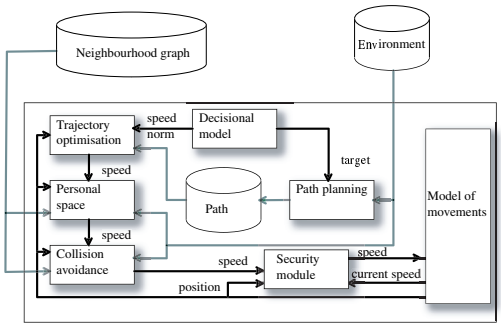

Figure 7: *The reactive navigation model.*

4. Navigation

Each virtual human is now able to plan its own path to reach its goal. The next step is to follow the path while avoiding collisions with other humanoids and with the environment. The architecture of our reactive navigation model is summarized in fig. 7. The first part is the computation of the neighbourhood graph. The second part is the reactive navigation modular algorithm filtering the optimal speed computed to follow the path in order to predict and avoid collisions while generating a human like behaviour.

Notations. By now, the humanoid H is represented with the following parameters: $P(H)$ is its position, $W(H)$ is its width and $S(H)$ is its speed.

4.1. Neighbourhood graph

All collision prediction algorithms are based on neighbourhood computation. When dealing with this sort of computation, two aspects need a particular attention:

1. the construction complexity of neighbourhood relations,
2. the possible relation between the computation complexity and the prediction distance.

The neighbourhood graph is a compromise between those two aspects. It creates long distance neighbourhood relations in sparse crowds and short distance relation in dense crowds without impact on computational cost. It is based on a two dimensional Delaunay triangulation of the humanoid's positions filtered with visibility. This triangulation has a construction cost of $O(n \ln n)$ [BY98] for n humanoids that en-

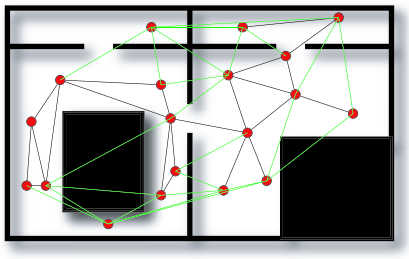

Figure 8: *Filtered Delaunay triangulation between entities.*

ables its computation for a large number of pedestrians. It generates a linear number of neighbourhood relations (edges of the triangulation) with an upper bound of $3n-3$ relations. This results in an average number of direct neighbours lower than 6. Those relations ensure the prediction of all collisions, because one of the Delaunay triangulation properties is to link each point (humanoid) to its nearest neighbour. Moreover, this triangulation generates the crowd topological structure with a computational cost independent of the distance between entities. The neighbourhood relation length is correlated to the local density inside the crowd allowing far collision prediction in sparse crowds and near prediction in dense crowds.

As the triangulation is not correlated with the environment geometry, neighbourhood relations are filtered using ray casting inside the convex cell structure. The number of rays is lower than $3n-3$ and the associated computational cost is optimized thanks to the underlying subdivision. This filtering process generates the neighbourhood graph in which each edge represents a neighbourhood and a visibility relation. An example of such a graph is given in fig. 8; green edges are removed during the filtering process. Once computed, this graph enables a direct access to visible neighbours of a given humanoid while automatically adapting the prediction distance to the crowd density. In the following, the term of **direct neighbourhood** will refer to the set of neighbours connected with an edge to the humanoid.

4.2. Reactive navigation

The reactive navigation process is described through a pipe filtering the speed vector of the entity (Cf. fig. 7). First, the planned path is analyzed in order to provide an ideal speed to adopt. This speed is filtered by the personal space module in charge of respecting a given minimal distance to humanoids and obstacles. Then the collision avoidance module modifies this speed in order to avoid collisions. This part of the algorithm is parameterized in order to copy out human like navigation rules. Finally, a security module verifies this speed in order to take into account the inertia of the humanoid.

Visual trajectory optimization

As described in the section 3.3, the path is furnished as a sequence of segments (or portals) to pass, in order to reach the goal. In this path, two consecutive segments are belonging to the same convex cell. A characteristic of pedestrian behaviour is to use visual optimization of the path. In order to do so, we propose a simple and fast visual optimization algorithm exploiting properties of the convex cell subdivision.

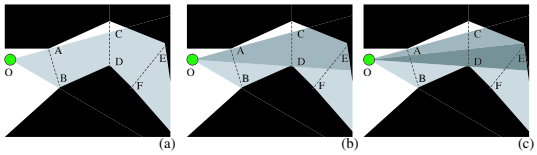

Figure 9: *Visual trajectory optimization.*

The visual optimization algorithm consists of sequentially intersecting visibility cones defined thanks to the path segments. This computation stops if the cone is empty or if the angle of the cone in O is lower than a certain threshold. If one of those two constraints is violated, the last valid cone is selected by the algorithm. As cells are convex, this cone defines an obstacle free region in which the humanoid can navigate. In figure 9, the cones OAB, OCD and OFE are sequentially intersected to compute the speed chosen collinear to the bisecting line. The second constraint influences the trajectory taken by the pedestrian: if the angle has a high value, the humanoid will pass through the center of the segments, if this value is low, the humanoid will skim along obstacles. This parameter, defined for each humanoid, enables the configuration of the pedestrian trajectory. This algorithm reproduces a well known characteristic of the pedestrian behaviour and rapidly computes an optimized and realistic trajectory continually refreshed in function of the pedestrian location relatively to the segments compounding its path.

Personal space

The personal space is a social rule defining a distance to respect between navigating pedestrians (d_N) and between pedestrians and obstacles (d_W). At least three situations can justify the violation of this distance: the crowd density is too high, the humanoid navigates inside a group, the humanoid is overtaking one of its neighbours. This rule is weak and do not have to be always respected. It is modeled as a repulsion force modifying the orientation (but not the norm) of the speed vector. Let $\mathcal{S}_N$ be a set containing the positions of the direct neighbours and $\mathcal{S}_W$ be a set containing the projections of the pedestrian position on the constraints delimiting its current cell. The output speed S_O is computed from the input speed S_I as follow:

$$\begin{cases} R(p,\mathcal{S},d) &= \sum_{x\in\mathcal{S}} \frac{-||x-p||+d}{d}(x-p) \\ S_O &= ||S_I|| \frac{R(p(H),\mathcal{S}_N,d_N)+R(p(H),\mathcal{S}_W,d_W)+S_I}{||R(p(H),\mathcal{S}_N,d_N)+R(p(H),\mathcal{S}_W,d_W)+S_I||} \end{cases}$$

If the sum of repulsive forces and S_I is null, this equation is not valid. In this case, S_O is equal to S_I. This equation only modifies the direction of the speed vector but not the norm in order to deform the trajectory to respect the personal space constraint.

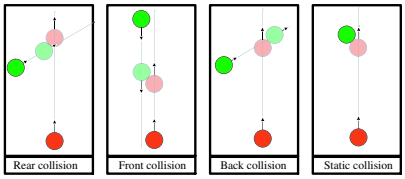

Figure 10: *The four collision types.*

Collision avoidance

The collision avoidance algorithm uses a linear trajectory extrapolation for collision prediction and a local optimization algorithm for the computation of a new speed avoiding collisions. This algorithm is configured with collision reaction modules describing possible typologies of reaction.

Collision prediction

Let S_I be the proposed speed for humanoid H and N be the tested neighbour. $P_r = P(H) - P(N)$ is the location of H relatively to N. $S_r = S(N) - S_I$ is the speed of H relatively to N. Solutions of the following equation are the possible collision times between humanoids H and N:

$$P_r^2 + (P_r.S_r)t + S_r^2 t = (W(N) + W(H) + \varepsilon)^2$$

This equation expresses the evolution of the distance during time between humanoids N and H. ε is a minimal security distance which can, eventually, vary over time. If this equation has no solution or a unique solution, there is no predicted collision. If there are two solutions t_1 and t_2, with $t_1 < t_2$, three cases can arise:

- $t_2 \leq 0$: this is a past collision, so there is no possible collision in the future.
- $t_1 < 0 \wedge t_2 > 0$: this is a collision, repulsive forces must be generated in order to correct the situation.
- $t_1 \geq 0$: a collision will arise at time t_1.

This information is also used to qualify the type of the collision. Let t_c be the computed collision time, $C_H = P(H) + S_I.t_c$ be the location of H at t_c and $C_N = P(N) + S(N).t_c$ be the location of N at t_c:

- if $(C_N - C_H).S_I < 0$ the collision is a **rear collision**,
- if $(C_N - C_H).S_I > 0 \wedge S_I.S(N) < 0$ the collision is a **front collision**,
- if $(C_N - C_H).S_I > 0 \wedge S_I.S(N) \leq 0$ the collision if a **back collision**,
- if $||S(N)|| = 0$ the collision is a **static collision**.

Those types of collision are summarized in fig. 10. They are used to configure the local avoidance algorithm with a subscription of collision avoidance to different types of collision.

Collision reaction modules

Reactions adopted when avoiding collisions can be classified in two (non-exclusive) categories: speed modification and direction modification. In order to describe the navigation behaviour, we introduce the notion of collision reaction module. Its role is to compute a new speed S_O for humanoid H with an actual proposed speed S_I that avoids the collision with humanoid N. The position of H relatively to N has the following expression: $P_r(t) = P(H) - P(N) + (S_I - S(N))t$

Avoiding the collision is equivalent to finding a new relative speed $S'_r = S_O - S(N)$ such as the distance between the straight line defined by $P(H) - P(N) + S'_r t$ and point $(0,0)$ is greater than $W(H) + W(N) + \varepsilon$, where ε is a security distance. The problem can be reformulated as the research of S_O such as $P(H) - P(N) + S'_r t$ is a tangent to the circle $\mathcal{C}$ centered in $(0,0)$ with a radius equal to $W(H) + W(N) + \varepsilon$. Let T_l and T_r be the two points lying on $\mathcal{C}$ and defining tangents passing through P_r. $D_l = T_l - (P(H) - P(N))$ and $D_r = T_r - (P(H) - P(N))$ with $D_l.((S_I - S(N)) \times R(\frac{\pi}{2})) > 0$ and $D_r.((S_I - S(N)) \times R(\frac{\pi}{2})) < 0$ defines the left and right relative avoidance directions in which $R(\alpha)$ stands for a rotation of angle α. The collision avoidance speed S_O is a solution of the following system, assuming that $D = D_l$ for a left avoidance and $D = D_r$ for a right avoidance:

$$\left\{ \begin{array}{lcl} \alpha D & = & S_O - S(N) \\ \alpha & > & 0 \end{array} \right. \quad (1)$$

By using this system, we defined four modules: left avoidance, right avoidance, accelerate and decelerate. Each of those modules find a new speed and adds new constraints:

- left and right avoidance: the constraint $||S_I|| = ||S_O||$ is added in order to maintain the norm of the speed.
- deceleration module: the constraint $S_O = \beta S_I$ with $0 \leq \beta < 1$ in order to maintain the speed direction.
- acceleration module: the constraint $S_O = \beta S_I$ with $\beta > 1$ in order to maintain the speed direction.

Once all collision reaction modules are described, they can be used to define the navigation behaviour of an entity. For special purposes, an acceptation module has been defined in order to accept a given speed even if it results in a collision.

As we use a local optimization algorithm to find the best speed to adopt, each solution computed by a collision reaction module has to be evaluated and rated. If S_I is the proposed speed resulting in a collision and S_O is the output speed avoiding this collision, following formulas are used to rate the proposition:

$$\begin{array}{lcl} c_d(S_O, S_I, \beta) & = & ((1 + \frac{S_O + S_I}{||S_O|| \times ||S_I||}) * 0.5)^\beta \times 2 - 1 \\ c_n(S_O, S_I, \beta) & = & (\frac{min(||S_O||, ||S_I||)}{max(||S_O||, ||S_I||)})^\beta \\ n(S_O, S_I, \beta_d, \beta_n) & = & c_d(S_O, S_I, \beta_d) * c_n(S_O, S_I, \beta_n) \end{array}$$

Those three functions compute a factor in the interval $[0;1]$ evaluating the quality of the speed S_O relatively to the speed S_I. The function c_d evaluates the direction difference whereas the function c_n evaluates the speed norm difference. Finally the function n evaluates the overall quality of the computed speed S_O. This function accepts two parameters β_d and β_n allowing to penalize high speed norm variations and/or high speed direction variations.

Driving rule model.

	rear	front	back	static
RA		$n(I,O,1,1)$		$n(I,O,1,1)$
LA			$n(I,O,1,1)$	$n(I,O,1,1)$
D		$n(I,O,1,1)$	$n(I,O,1,1)$	
A		$n(I,O,1,5)$	$n(I,O,1,5)$	
AC	X			

Minimal adjustment model.

	rear	front	back	static
RA		$n(I,O,1,1)$	$n(I,O,1,1)$	$n(I,O,1,1)$
LA		$n(I,O,1,1)$	$n(I,O,1,1)$	$n(I,O,1,1)$
D		$n(I,O,1,1)$	$n(I,O,1,1)$	
A		$n(I,O,1,5)$	$n(I,O,1,5)$	
AC	X			

Figure 11: *Different configurations of the navigation model.*

Local optimization algorithm

The local optimization algorithm is configured with instances of collision reaction modules. A list of collision reaction modules with a partially instantiated evaluation function n (only β_d and β_n are defined) is associated to each collision type (rear, back, front and static). The figure 11 presents different configurations of the navigation model: the driving rule model and the minimal adjustment model. In this figure, RA stands for right avoidance, LA for left avoidance, D for decelerate, A for accelerate and AC for acceptation.

Once the model is configured, this information is used by the local optimization algorithm. Each module is specialised for the computation of a speed avoiding a collision with one entity. Thus the computed speed can result in a new collision with an other neighbour. The role of the algorithm is to find a combination of reactions producing an "optimal" speed.

1. The algorithm uses a sorted list L containing data of the form (q,s,t) sorted decreasingly in function of $q \times t$. (q,s,t) is a speed proposition in which q stands of the quality of the speed s proposed in reaction to a collision occurring at time t. This list is initialised with $(1,S_I,\infty)$.
2. If L is empty, no solution have been found so the resulting speed S_I is null. Otherwise, (q,v,t) is extracted from the head of L.
3. A collision prediction is computed using v as the humanoid speed. This computation searches the earliest occurring collision. Two cases can arise:
   a. A collision is predicted at time t_c but $t_c \geq t$ or no collision is detected. The actual speed v is selected because it is the best compromise between the quality of reactions to collisions and predicted time without collision. The first condition $t_c \geq t$ ensures the convergence of the algorithm by assuming that this predicted collision will be avoided later.
   b. A collision is detected and $t_c < t$. The type of the collision is determined (back, rear, front, static) and each corresponding collision reaction module is consulted. If one of those modules is an acceptation module, the current speed is selected. Otherwise, for each module, its proposed speed v' and the speed quality q' are computed and (q',v',t_c) is inserted in L. The algorithm returns in step 2.

The resulting speed S_O computed by this algorithm is a compromise between the quality of the reactions and the predicted time without collision.

Security module

As the speed computed by the local optimization algorithm can not avoid all collisions, a security module as been added. This module reduces the norm of the speed S_I in order to maintain a given reaction time (t_{react}) with the next collision. The next collision is predicted with direct neighbours and with obstacles (thanks to ray casting in the convex cell subdivision) using the speed $S(H)$ (and not S_I) in order to take into account the inertia of the pedestrian. Let t_c be the moment of the predicted collision. If $t_c < t_{react}$ then $S_O = S_I \frac{t_c}{t_{react}}$. This speed modification ensures wall collision avoidance and reduces the speed in case of possible collision with neighbours.

The proposed architecture includes a wide variety of pedestrian characteristics and is configurable. It enables the reproduction of a large number of navigation behaviours inspired by psychological studies without leaving out real-time constraints as shown in the result section. But when controlling animated characters using a combination of motion capture and inverse kinematic for legs motion, the algorithm can cause jerky motions due to a noisy absolute position. This position needs to be filtered in order to solve this problem.

5. Results

The performances of the path planning algorithm have been tested on a database representing the center of a city. The three dimensional model represents about 2600 buildings on a surface of $1.3 \times 1.3 km^2$. The resulting spatial subdivision is compounded of 8165 convex cells containing 8005 constrained segments and 10439 free segments. The figure 12 presents a comparison between A^* path planning in the full graph and path planning with A^* in the abstract graph. This figure exhibits a logarithmic gain on path planning time. In the worst case, A^* running on the full graph computes a path in 45ms whereas A^* using our algorithm computes a path in 2.5ms. Our system is thus able to plan a path in real-time in large environments, enabling this feature in interactive applications.

Thanks to the precise representation of the environment

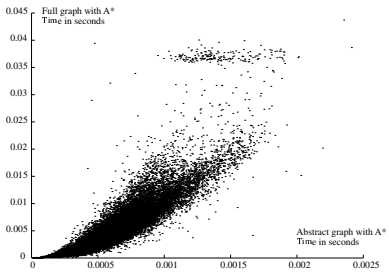

Figure 12: *Path planning benchmark.*

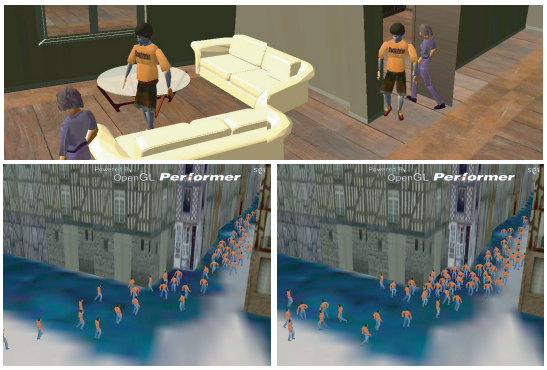

Figure 13: *Indoor and outdoor navigation.*

through the convex cell subdivision, humanoids can navigate in indoor and outdoor environments (Cf. fig. 13). The proposed architecture of navigation is rich. It includes characteristics of pedestrian behaviour such as visual trajectory optimisation and personal space rule. Moreover, the collision avoidance algorithm enables the description of a wide variety of pedestrian behaviour through the translation of social rules (driving rules, minimal adjustments...). But it is also efficient. The figure 14 presents some benchmarks (realised on an athlon XP 1800+). This figure traduces the evolution of the computation time (neighbourhood graph construction and reactive navigation algorithm) in function of the number of pedestrians. Whereas the computation complexity of the Delaunay triangulation is $O(n \ln n)$, the evolution looks linear. The algorithm is able to simulate about 2000 pedestrians at a frequency of 10Hz with about 20-25% of computation time dedicated to the computation of the neighbourhood graph. Other tests have been made with pedestrians evolving inside the 3D database of a city. We are able to simulate about 400 fully animated pedestrians on a XEON 3GHz with a quadro FX graphics card.

6. Conclusion and future work

The approach presented in this paper enables the real-time animation of several hundreds of pedestrians, populating large and complex indoor and outdoor environments. An accurate hierarchical topological structure is built from the

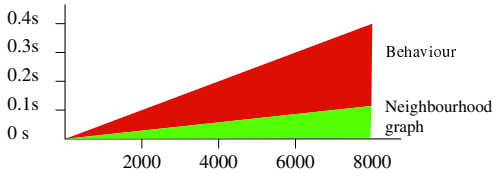

Figure 14: *Evolution of the computation time in function of the number of pedestrians (Athlon XP 1800+).*

geometric database of a virtual environment. Based on this structure an optimized path planning algorithm has been built. Moreover, it also handles visibility computation between different entities. The neighbourhood graph filtered with visibility, allows to bound the complexity of collision detection to $O(n)$ while offering a rich topological information on crowds through neighbourhood relationships. Moreover, this structure, thanks to the underlying Delaunay triangulation, automatically adapts to the density of the population, allowing near collision avoidance in dense crowds and far collision avoidance in sparse crowds with the same computational cost. Finally, the proposed reactive navigation architecture is configurable and inspired by psychological studies on pedestrian behaviour without leaving out real-time constraints.

Future work will focus on two points. The first one concerns the extension of the hierarchical topological structure of the environment to allow the automatic generation of informed and structured environments and thus more complex behaviours. The second one is the use of the neighbourhood graph for automatic group detection as it contains sufficient information to allow real time classification of entities. This classification should improve the realism of the simulation by performing group avoidance instead of pedestrian avoidance and will allow to model both individual and group behaviour and to combine them.

References

[ACF01] ARIKAN O., CHENNEY S., FORSYTH D. A.: Efficient multi-agent path planning. In *Computer Animation and Simulation '01* (2001), Springer-Verlag, pp. 151–162. 2

[ALA*01] ASHIDA K., LEE S., ALLBECK J., SUN H., BADLER N., METAXAS D.: Pedestrians: Creating agent behaviors through statistical analysis of observation data. In *Computer Animation* (2001), Society I. C., (Ed.). 2

[BLA02] BAYAZIT O. B., LIEN J.-M., AMATO N. M.: Roadmap-based flocking for complex environments. In *Pacific Conference on Computer Graphics and Applications* (2002). 2

[BMdOB03] BRAUN A., MUSSE S., DE OLIVEIRA L., BODMANN B.: Modeling individual behav-

iors in crowd simulation. In *16th International Conference on Computer Animation and Social Agents* (2003), IEEE. 2

[BT98] BANDI S., THALMANN D.: Space discretization for efficient human navigation. *Computer Graphics Forum 17*, 3 (1998). 2

[BY98] BOISSONNAT J.-D., YVINEC M.: *Algorithmic Geometry*. Cambridge University Press, 1998. 5

[FBT99] FARENC N., BOULIC R., THALMANN D.: An informed environment dedicated to the simulation of virtual humans in urban context. In *Computer Graphics Forum* (1999), vol. 18(3), pp. 309–318. 2

[GKM*01] GOLDENSTEIN S., KARAVELAS M., METAXAS D., GUIBAS L., AARON E., GOSWAMI A.: Scalable nonlinear dynamical systems for agent steering and crowd simulation. *Computers and Graphics 25*, 6 (2001). 2

[Gof71] GOFFMAN E.: *Relations in public : microstudies of the public order*. New York : Basic books, 1971. 2

[HFV00] HELBING D., FARKAS I., VICSEK T.: Simulating dynamical features of escape panic. *Nature 407* (2000), 487–490. 2

[HK02] HOSTETLER T. R., KEARNEY J. K.: Strolling down the avenue with a few close friends. In *Third Irish Workshop on Computer graphics* (2002), pp. 7–14. 2

[HPH*93] HILLIER B., PENN A., HANSON J., GRAJEWSKI T., XU J.: Natural movement: or, configuration and attraction in urban pedestrian movement. *Environment and Planning B: Planning and Design 20* (1993), 29–66. 2

[KBT03] KALLMANN M., BIERI H., THALMANN D.: Fully dynamic constrained delaunay triangulations. *Geometric Modelling for Scientific Visualization* (2003). 3

[KKL96] KAVRAKI L., KOLOUNTZAKIS M., LATOMBE J.: Analysis of probabilistic roadmaps for path planning. In *Proc. IEEE ICRA* (1996), vol. 4, pp. 3020–3025. 2

[Kuf98] KUFFNER J. J.: Goal-directed navigation for animated characters using real-time path planning and control. *Lecture Notes in Computer Science 1537* (1998), 171–179. 2

[Lat91] LATOMBE J.-C.: *Robot Motion Planning*. Boston: Kluwer Academic Publishers, Boston, 1991. 1

[LMM03] LOSCOS C., MARCHAL D., MEYER A.: Intuitive behavior in dense urban environments using local laws. In *Theory and Pratice of Computer Graphics (TPCG'03)* (Birmingham, UK, Juin 2003). 2

[LW92] LEE J. R. E., WATSON R.: *Regards et habitudes des passants*, vol. 57-58 of *Les annales de la recherche urbaine*. 1992. 2

[MT97] MUSSE S., THALMANN D.: A model of human crowd behavior : Group interrelationship and collision detection analysis. In *Computer Animation and Simulation '97* (1997), Springer Verlag, pp. 39–51. 2

[O'H00] O'HARA N.: K-d tree neighbour finding for the flocking algorithm. In *Pacific Graphics* (2000). 2

[O'H02] O'HARA N.: Hierarchical impostors for the flocking algorithm in 3d. *Computer Graphics Forum 21*, 4 (2002), 723–731. 2

[Rey00] REYNOLDS C. W.: Interaction with groups of autonomous characters. In *Game Developers Conference 2000* (2000). 2

[RKBB94] REICH B., KO H., BECKET W., BADLER N. I.: Terrain reasoning for human locomotion. In *Computer Animation* (1994), IEEE. 2

[RQ98] RELIEU M., QUÉRÉ L.: *Les risques urbains : Acteurs, systèmes de prévention*. Anthropos : Collection ville. 1998, ch. Mobilité, perception et sécurité dans les espaces urbains. 2

[TB96] THRUN S., BÜCKEN A.: Integrating grid-based and topological maps for mobile robot navigation. In *Proc. of the AAAI Thirteenth National Conference on Artificial Intelligence* (1996), AAAI Press / MIT Press. 2

[TD00] THOMAS G., DONIKIAN S.: Virtual humans animation in informed urban environments. In *Proc. of the Conference on Computer Animation* (2000), IEEE. 2

[TLC02] TECCHIA F., LOSCOS C., CHRYSANTHOU Y.: Visualizing crowds in real-time. *Computer Graphics Forum 21*, 4 (2002), 753–765. 2

[TP02] TURNER A., PENN A.: Encoding natural movement as an agent-based system: an investigation into human pedestrian behaviour in the built environment. *Environment and planning B: Planning and Design 29* (2002). 2

[UT02] ULICNY B., THALMANN D.: Towards inter-

active real-time crowd behavior simulation. *Computer Graphics Forum 21*, 4 (2002). 2