A unified structure for crowd simulation

Thomas Jund
tjund@unistra.fr

Pierre Kraemer
kraemer@unistra.fr

David Cazier
david.cazier@unistra.fr

Abstract

Realistic simulation of crowds is an important issue for the production of virtual worlds for games, entertainment or architectural and urban planning. Difficult issues need to be addressed such as collision avoidance and the handling of dynamic environments.

In this paper, we present a unified structure for the simulation of complex urban environments. We propose a topological multiresolution model supporting different levels of details, allowing efficient proximity querying and compatible with real-time rendering and hierarchical path planning. A fine exploitation of the multiscale aspect of the underlying model allows to achieve the same efficiency as the fastest existing methods. The generality of the approach allows the simulation to be executed on any 2-manifold and the unified approach eases the handling of dynamic environments.

1 Introduction

Human beings are naturally able to walk through a dense crowd while searching their way in complex urban environments. Simulating the behaviour of humans is a difficult challenge and a key feature for the production of accurate virtual worlds. To be realistic, an autonomous agent must adapt its trajectory all along the simulation. Basically the progression of an agent is guided by global objectives but has to be adapted frequently to take into account dynamic obstacles and other navigating agents.

To achieve this, a virtual human has to be aware of its environment at different scales. It should have an overview of the whole ground cartography in order to plan its path or to search its way if no *a priori* knowledge is assumed. Dynamic obstacles or more generally changes in the environment (*e.g.* doors opening, variations in the density of agents) should be taken into account on the fly to adjust the trajectories. At last, every virtual human must constantly be aware of the other agents operating in the vicinity to adapt its behaviour to their presence.

Thereby, obtaining a realistic simulation with thousands of autonomous agents requires to address several issues such as hierarchical path planning, proximity queries and collision avoidance. Each of these processes need specific optimizations to obtain real-time execution and rendering times. Finally, to support interactive alterations in the virtual world, the internal representations used by each process should support on the fly transformations and allow efficient propagation of the changes.

The main contribution of this paper is a unified framework for real-time simulation of the navigation of thousands of autonomous agents in complex urban environments. It includes a cellular representation of the environment that supports different levels of detail (including semantic levels), compatible with classical rendering algorithms. This same representation is used to efficiently track the agents within the environment. The environment is then adaptively subdivided depending on the crowd density to provide constant time proximity queries. The cellular decomposition of the environment is also implicitly used as a dual graph to perform hierarchical path planning.

In the following, section 2 presents related works and discusses their advantages and limitations in this context. Section 3 describes our multiresolution model and the generation of the environment. Section 4 details how efficient

proximity queries are achieved in this framework. Finally, section 5 presents experimental results and some comparisons.

2 Related Works

Many published studies have addressed the problem of modelling the behaviour of autonomous agents with the reproduction of phenomena such as the formation of queues or humans interactions in crowds [1, 2, 3]. Among them, one mainly distinguishes three kinds of approaches. They are differentiated by their ability to control the behaviour of agents or groups of agents, the number of entities that can be processed simultaneously, but also by the kind of proximity requests they require.

2.1 Collision free navigation

Potential fields based works define repulsive and attractive forces associated to agents, goals and obstacles [4, 5, 6]. These approaches are well suited for large simulations and allow the formations of lanes or flocking. Their main limitation is probably the difficulty to finely tune their parameters and the lack of tools to describe individual social behaviours.

Rule based methods have been widely used for virtual humans. In [7] a 2D regular grid is used to handle reactions of the agents and store information about probable paths or possible behaviours when reaching some destination. In [8] and [9] proximity queries are used to search the closest set of agents. Then, depending on the local configuration of agents and visible obstacles, specific rules of navigation apply. These approaches offer intuitive control and the obtained behaviours are realistic. However, the result of their combination is difficult to predict and may cause artefacts to appear.

Primarily used in the robotics field, the velocity based methods are more robust. Velocity obstacles have been introduced in [10]. They provide necessary and sufficient conditions for a robot to select a velocity avoiding collision with moving obstacles. This method have been extended to the case of multiple agents with the notion of Reciprocal Velocity Obstacle [11] and optimized in [12] to increase robustness.

This paper focuses on achieving efficient proximity queries in general and on dynamic environments. We thus rely on an external library to evaluate the behaviour of agents. We use the RVO2 library (Reciprocal Collision Avoidance for Real-Time Multi-Agent Simulation) [11] to compute the velocity of the agents. However our framework is generic enough to operate or support other methods for the management of agents behaviour and reactive navigation.

2.2 Proximity queries

All the mentioned methods need specific data structures and algorithms to query the agents or obstacles close to a given position. Space partitioning trees and bounding volumes hierarchies have been widely used to speed up proximity queries. Recent works [11, 6, 12] use lists of polygons and distinct kd-trees to retrieve fixed obstacles or moving agents. These kind of data structures provide efficient access, but suffer from heavy update costs when the scene changes. That makes them inappropriate for interactive or dynamic environments.

Image-based approaches exploit a rasterization of the scene in which agents register. Two kinds of grids are mostly used. Many works use static fine grid whose cell sizes are related to the thickness of agents. To test the existence of collisions with static or moving obstacles, each agent queries the grid by accessing a subset of the cells in its vicinity. In [13], only the points on the direct path of the agent are tested. In [9], all points in a region representing a cone of visibility are tested. In both cases, using dense grids allows precise computations but limits the size of the considered neighbourhood to achieve real time performances. That prevents these methods from taking into account distant obstacles.

Following the approach presented in [14], spatial hashing can be exploited to register moving agents in an implicit grid. To limit the number of queries and updates on the hash table, the cell size must be greater than the interaction distance between agents. This thereby certifies that the cells in the immediate neighbourhood of a given agent contains all possible interacting objects. This approach has shown impressive performances for collision detection between deformable objects. However, the high number of

accesses to the hash table prevents its usage in dense simulations where the concurrent memory accesses become a bottleneck.

Other approaches calculate or maintain a neighbourhood graph between agents or obstacles. In [8] and [15], a Delaunay triangulation of the agents positions is computed at each time step. This graph is filtered with visibility information before obtaining a list of the nearest neighbours of an agent. In [4], a Kinetic Data Structure is introduced. Its goal is to maintain a Delaunay triangulation between agents that is updated after moving each agent. Finally, the adaptive road map presented in [16, 17] stores a neighbouring graph whose links are updated based on physical models. These three approaches exploit the maintained graphs to achieve both collision avoidance and path planning. They are well adapted to sparse crowds, but the computations or updates of those graphs grow heavily with the density of the crowd.

2.3 Path planning

Most works in crowd simulation uses the A* graph search algorithm or variants optimized for hierarchical models. In [9] a set of hierarchical maps is used: an accessibility map associated with a perception and a path map. The latter includes a quadtree map which supports global and long-range path planning and a fine grid for short-range paths. A hierarchical path planning [18], optimized for grid-based maps is used.

The works presented in [8] and [15] use semantic road maps built from the spatial subdivision of the environment. A visibility graph is extracted from a Delaunay triangulation of the scene and simplified through abstraction of the dead end and passage paths (i.e. nodes with degree lower than 3). This graph is then used for hierarchical path planning.

Our multi-scale topological model implicitly encodes a neighbourhood graph of the scene that can be exploited in the same way to achieve hierarchical path planning. As this paper is mainly focused on proximity queries, the path planning aspect will not be detailed further.

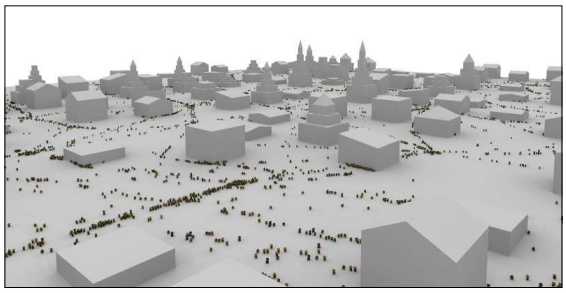

Figure 1: A procedurally generated city where 10000 autonomous agents navigate by following a series of goals.

3 A unified structure

Crowd simulation imposes the use of optimized data structures for environment representation, path planning and proximity queries. As it has been previously reminded, these structures include discrete grids, visibility graphs, road maps, hash maps and often hierarchical representations like quadtrees, BSP or kd-trees.

Using multiple data structures lower the consistency of the representation, in particular when geometrical or topological modifications of the environment occur. Moreover, the tree based structures need to be reconstructed each time their nodes are moved or the scene is transformed. Such frequent updates hinder the achievement of real-time performances for large simulations.

Moreover, when working with complex environments that may contain bridges, tunnels or multiple floors, the representation of the ground by 2D grids is no longer possible. In approaches like the one presented in [5], the environment has to be cut in several planar parts linked together in a graph.

We propose a more generic representation of the environment based on a multiresolution model. This structure supports any 2-manifold environment and provides all the needed information for rendering, path planning and proximity querying with multi-scale capabilities.

Our main structure is encoded by a multiresolution 2-map [19] which is a multiresolution extension of the well-known half-edge data structure. The latter is a topological model that can represent the cellular decomposition of any ori-

entable 2-manifold. It proposes a light weight data structure along with optimal neighbourhood queries between the cells.

The multiresolution extension inherits its flexibility and efficiency. Any polygonal mesh can be represented on every resolution levels of the hierarchy. Each level is a 2-map and neighbourhood queries are resolved in optimal time, whatever the considered resolution level. We use the CGoGN library [20] that provides an efficient implementation of these models, tools to manipulate the topology of meshes and an attribute manager.

An important property of combinatorial maps is that they simultaneously encodes meshes and their duals. Thus, for each level, a map describes the faces it contains, their adjacency relations and therefore, implicitly, their neighbourhood graph. An edge between two faces is also a link between two nodes of this graph. Thus, implicitly, when an edge of the environment is marked as an obstacle, the corresponding link cannot be followed by the path planning algorithm.

When the environment is modified, the topological neighbourhoods are processed simultaneously with the geometry without requiring specific updates. Modifications include for instance the subdivision or merging of faces or any other manipulation with an interaction tool. In other words, a multiresolution map defines a hierarchy of graphs that naturally supports dynamic transformations.

In our simulations, the environment is composed of open spaces and fixed obstacles that are partitioned into vertices, edges and faces. Figure 1 shows a typical simulation of a city where the buildings are obstacles and the agents have to find their ways in the streets. All cells, including the buildings walls and roofs, are encoded in the global manifold mesh of the scene.

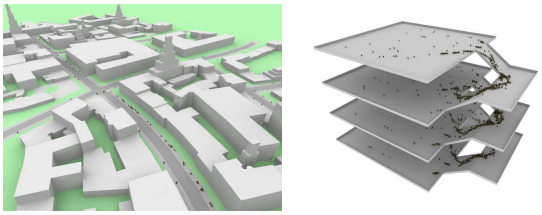

Figure 2: Left: a urban environment generated from *OpenStreetMap* data. Right: multiple superimposed floors.

The topology of these meshes can be arbitrary. They may contain holes (whose edges must be marked as obstacles) and be non-planar, *i.e.* contain bridges, tunnels or more generally be any 2-manifold. The agents can navigate through all the faces and edges of the scene, except those that were explicitly marked as obstacles (walls, roofs, doors, parts of the scenery, etc.). As shown in figures 2 and 3 our system supports the existence of superimposed floors, can import data from geographic databases or simulate a crowd walking on any kind of surfaces.

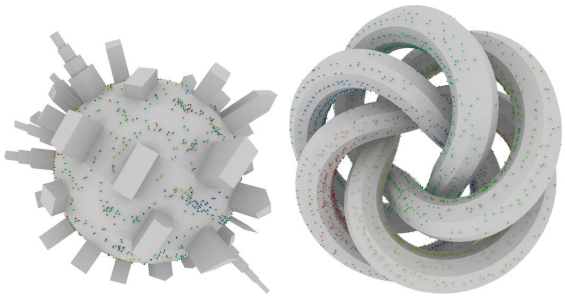

Figure 3: Left: 1500 agents navigating on a planet. Right: 6000 agents navigating on a torus knot.

Non-flat environment requires to adapt the behaviour algorithm, namely in our case the RVO2 library. Indeed, the computation of an agent new velocity at each time step is intrinsically solved as a 2D problem and the implicit parametrization given by a flat environment is no more available.

We use a local parametrization to flatten the neighbourhood of an agent with respect to its viewing distance. As shown in the following, this neighbouring region can be limited to the faces incident to the one that contains the agent.

4 Proximity querying

At each time step of the simulation, each agent has to sense the environment for nearby neighbours and obstacles. This process is a critical point for the global efficiency of the simulation and is generally not addressed in papers dealing

with crowd simulation systems that mainly focus on collision avoidance algorithms.

In our system, the environment partition itself is used as an accelerating structure for proximity queries. Every face maintains the set of the agents it contains and of nearby obstacles. The agents can thus scan their vicinity by traversing the topological structure starting from their face and accessing the sets in adjacent faces. Maintaining this information is a fundamental aspect of our framework and requires fine optimizations. As an autonomous entity, each agent is given the task to register in the cell it lies in. For this purpose, each agent is tracked in the partition using the algorithm described in [21].

4.1 Optimizing neighbourhood traversals

Computing the new direction and velocity of an agent to avoid collisions requires to gather all the agents lying within a given distance. This set is obtained by querying the environment for the agents lying in the neighbouring faces. Even if the underlying combinatorial model provides efficient queries, naively gathering all these agents requires a breadth-first traversal of regions included within a defined visibility distance around each agent.

To optimize these proximity queries, we limit this search to the one-ring of the face that contains the agent – *i.e.* the faces that share at least one vertex with it (figure 4). If we assume an upper bound to the degree of faces, it ensures that the number of traversed faces is kept within a constant limit and thus improves the overall complexity of the algorithm.

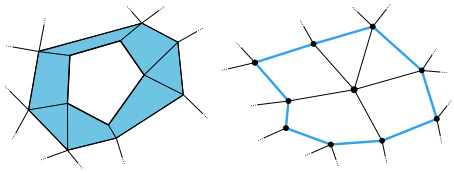

Figure 4: Face one-ring and vertex one-ring border.

An agent can lie anywhere in its face. To ensure that this optimization does not lead to miss any close agent, all the visible neighbours of an agent must lie in the one-ring of its face. This leads to the following shape constraints that all faces have to respect.

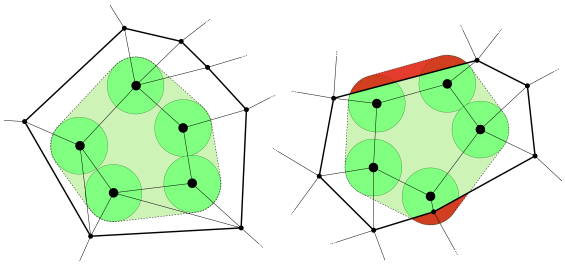

Figure 5: Left configuration is valid: all the possible viewing area of any contained agent is in the one-ring of the central face. Right configuration is not valid: parts of the possible viewing areas are outside the one-ring of the face.

For a face F to be valid, the Minkowski sum of F and the viewing area of agents must not get beyond the one-ring of F (figure 5). This constraint can be expressed in a vertex-centered way. Let the border of the one-ring of a vertex be the set of the edges of the faces incident to the vertex minus the set of edges directly incident to the vertex (figure 4). For each vertex, the point-segment distance to each edge of its one-ring border has to be greater than the agents viewing distance.

4.2 Optimizing memory access

At each time step, the set of agents contained in a face is accessed not only by the agents it contains but also by all the agents contained in the faces of its one-ring. At the same time, this set is only updated when an agent leaves or enters the face. As we measured in real simulations, these events are an order of magnitude less frequent than the read accesses. Moreover, all the agents of a given face perform at each time step the exact same neighbourhood traversal and memory access.

In order to reduce the number of access to the data structure and at the expense of some memory space, we chose to duplicate some information. Each face of the environment stores not only the set of contained agents, but also the set of agents contained in its one-ring. The agents will then find, with only one read access, all the

information they need to explore their neighbour configuration. The effect is a dramatical reduction of the number of access to the data structure that would otherwise become a bottleneck for the successive treatment of thousands of agents.

The maintenance cost is slightly increased but updates only occur when agents cross edges. When entering a new face F_n, an agent first unregisters as *contained* from its source face F_s and as *neighbour* from its one-ring F_s^{adj}. Then it registers in the reached face as *contained* and in its one-ring F_n^{adj} as *neighbour*, as illustrated in figure 6. An agent usually enters a new face after crossing a single edge. In this case, the agent only has to unregister from the faces belonging to $F_s^{adj} - F_n^{adj}$, and to register in the faces belonging to $F_n^{adj} - F_s^{adj}$. The tracking system is able to discriminate between single or multiple edge crossing.

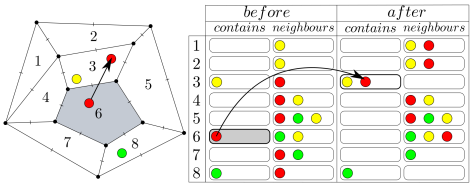

Figure 6: Updating neighbourhoods: an agent registered in face 6 and neighbour faces $(3, 4, 5, 7, 8)$ moves into face 3. It unregisters from face 6 and as neighbour of faces $F_6^{adj} - F_3^{adj}$ $(3, 7, 8)$ and registers in face 3 and as neighbour of faces $F_3^{adj} - F_6^{adj}$ $(1, 2, 6)$.

4.3 Bounding neighbours number

For each agent, among all the neighbours lying in its face and its one-ring, only those included in its viewing distance are considered by the collision avoidance system. Filtering this set of neighbours is usually done by sorting the candidates by increasing distance and pruning the result to keep only the ones that satisfy the viewing condition. Moreover, avoidance algorithms usually consider only a limited number of these selected neighbours. As the number of agents contained in each face – and therefore in their one-ring – increases, so does the complexity of this filtering process. To support real time performances even in presence of dense situations, this number should be bounded.

Representing the environment with small faces limits the number of potential neighbours. As a negative result, the memory cost would be greatly increased and the probability for an agent to cross an edge while moving would be higher, leading to more information updates.

We propose an adaptive routine that subdivides high density regions and simplifies them back when the density declines (figure 7). As shown in the following experimentations, this algorithm shows good performances while exhibiting a limited memory overcost.

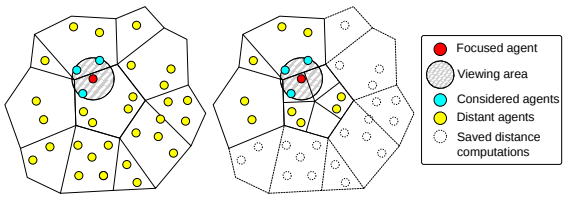

Figure 7: The subdivision of densely filled cells reduces the number of distance computations each agent has to perform.

When the number of agents registered in a face and its one-ring exceeds a given number D_h, the cell is subdivided and the registrations of the agents it contains are updated. As part of the multiresolution topological model, this subdivision process can be repeated as many time as needed by the simulation.

The shape constraint exhibited in section 4.1 can be ignored in this case. Indeed, this constraint is only meaningful in the coarsest mesh to ensure that all neighbours within the interaction distance are considered. When a face is subdivided, the resulting faces are included in it. If D_h is chosen in concordance with the number of neighbours considered by the behaviour algorithm, then no close agent can be missed.

When the number of agents registered in a face and its one-ring becomes smaller than a given number D_l, a simplification is performed. The face gets back its previous shape and the contained agents update their registrations. This simplification process keeps the memory cost of the environment in control and reduces the probability for agents to cross edges.

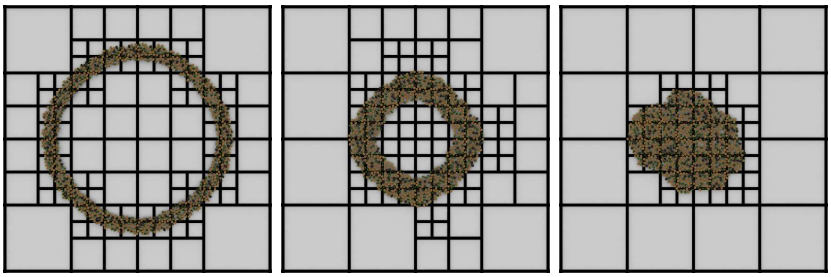

Figure 8: Circle-1000 scenario. With our method the mesh automatically gets refined and coarsened depending on the density of agents; any subdivision model can be used.

5 Experimentations

In this section, we present experimentations demonstrating the performance of our system compared to other approaches used for collision avoidance. We set up several scenarios to test our crowd simulation system in different configurations.

In the Circle-N example (figure 8), N agents are regularly distributed along a circle and aim at their diametrically opposed position. This scene implies a high density of agents near the center. In the Crossing-N example, N agents navigate, half of them walking from the left to the right of the scene, the other in the opposite direction. Here again high densities of agents are obtained in the middle of the scene. In the City-N example, N agents cross a city-like environment (figure 1) following random paths. Our framework is able to run this simulation for 10000 agents at 30 fps on a standard desktop computer using only one core.

5.1 Time performance

We present here the performance of our method compared to the kD-tree based method used in the RVO2 library [11] and to a static regular grid at a coarse and a fine resolution using our proximity queries algorithm. Figure 9 shows on the left the computation time of the Circle-N simulation with 500 and 1000 agents and on the right the one of the City-N simulation with a group of 900, 1444 and 2025 agents. For these first benchmarks, we enable the face shape constraint on our method.

As we can see, in both cases our method needs about 5 to 6 less time to play the scenario

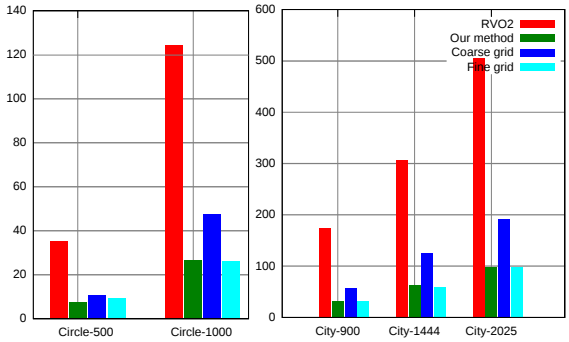

Figure 9: Time needed for the Circle-N and City-N scenarios including proximity queries and velocities updates.

than the kD-tree based method. The coarse regular grid needs here about twice the time of the adaptive method. It is still more efficient than the kD-tree based algorithms, but the size of the coarse element has been here completely arbitrarily chosen and a coarser resolution may lead to lower performances. The fine grid resolution corresponds here to the finest achievable resolution with respect to the face shape constraint. The time needed to complete the scenario is here equivalent to the adaptive method.

The speed-up compared to kD-tree based methods mainly comes from the need to rebuild the trees at each time step of the simulation. Our method is based on the usage of a single structure used as environment representation as well as acceleration structure for proximity queries. This allows us to reduce the number of structures to maintain and to minimize the amount of information to compute at each time step by exploiting the spatial and temporal consistencies of the displacements of the agents.

In terms of memory cost, our adaptive approach usually stands in the same order of magnitude than a coarse grid while presenting the same performance than the usage of a fine grid.

To be fair in the comparison, let us compare with a spatial hashing technique based on the approach presented in [14]. The agents register here in an implicit fine grid whose resolution also corresponds to the finest achievable resolution with respect to the face shape constraint. The non-empty cells are stored in a hash table. Its memory cost is chosen to be equivalent to that of our structure.

Figure 10 shows some results for the Crossing-5000 scenario. The bumps in the middle of the curves correspond to the meeting of the two opposite groups of agents that creates a high density region. Map-20 curve corresponds to a configuration where the size of the cells generated by the adaptive process is strictly limited by the face shape constraint.

The two methods are quite equivalent here. The bumps appear earlier in our approach as the environment has first to pay the cost of the subdivision before absorbing the high density. The bump for the HashTable-20 curve is due to the increasing memory access required to filter the dense neighbours lists. The Map-10 (respectively Map-5) curve shows a configuration where the face shape constraint is relaxed to consider half (respectively quarter) of the agents viewing distance. Here our dynamically subdivided model performs clearly better as the number of queried neighbouring agents is limited.

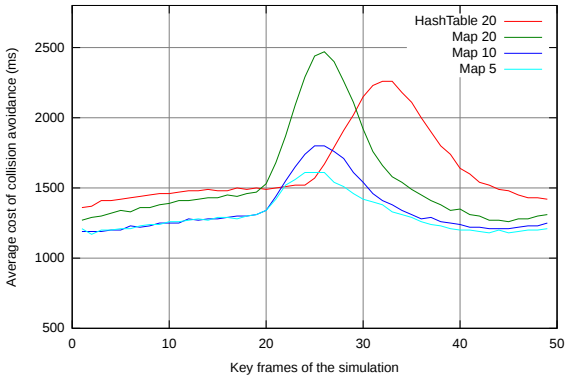

Figure 10: Time (in ms) for the Crossing-5000 scenario in different configurations.

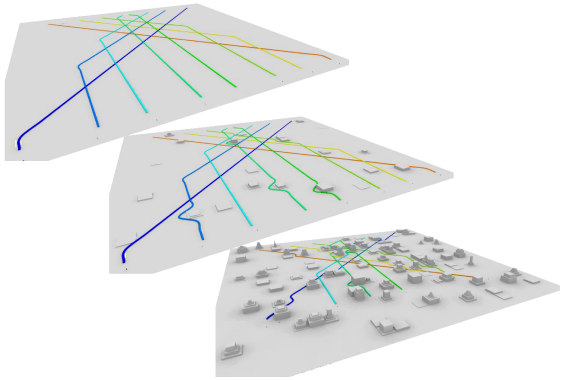

Figure 11: When the environment changes (here new buildings are constructed), agents paths are recomputed dynamically during the simulation.

5.2 Dynamic environments

As we already stated above, our system is well suited to dynamic environments. The multiresolution topological model provides efficient local operators for the modification of the topology of the environment partition. The agents tracking method is also robust to dynamic changes in the topology of the underlying mesh.

In the scene illustrated in figure 11, several agents plan a path to cross an empty area. As they start to walk along this path, new buildings arise from the ground. These new obstacles register themselves in the environment cells. As soon as they appear, they are taken into account by the collision avoidance system when the agents pass by. Each time a new obstacle appears in the path of an agent, it is updated to avoid this new obstacle.

6 Limitations and perspectives

We have presented a complete framework from environment generation to crowd simulation. Several aspects can still be improved to extend the scope of our method.

6.1 Agents visibility

In our system, all the agents that lie in the viewing area of a given agent are considered as neighbours for the collision avoidance system. This can include for example some agents that

lie behind a building corner which should actually not be considered as visible. To avoid this, the particle tracking system could be used by throwing a virtual particle from the agent to its neighbours – as done in [22] for edge collision detection. The tracking algorithm will then detect if an obstacle is in the way between the two agents and the given neighbour will not be considered for collision avoidance.

The viewing area of an agent is considered here as a disc around its position. This is inherited from the RVO2 collision avoidance library we use. Other works suggest the use of more complex shapes like cones. Our registration algorithm should be adapted to handle such sophisticated approaches. The registration principle we use is generic enough to be adapted this way, for instance with agents registering only in faces in front of their trajectories. The updates of the registration data in case of subdivision or simplification of faces would however be more complex.

6.2 Moving obstacles

In our examples the environment is only composed of fixed obstacles like buildings. To obtain a more realistic simulation, moving obstacles like vehicles should be considered.

Instead of representing obstacles as edges of the environment partition, obstacles can be represented by ”floating” closed polygons. A particle can be associated with each vertex of such a moving obstacle and tracked using the same particle tracking system as for the agents. Edges of these polygons can then be registered in the cells of the environment just like the fixed obstacles. This way, moving obstacles are considered by the agents in their velocity update algorithm.

6.3 Parallelism

Parallel computation can be achieved for all the agents update process. After querying information about its neighbouring configuration, each agent can autonomously compute its new direction and velocity. We did not investigate deeply the speed-up that such parallel computation could bring to the overall simulation performance, but our framework does not hinder such improvement.

7 Conclusion

We have presented a crowd simulation system based on the intensive usage of a unique topological model. It is used for the representation of the environment, path planning and for proximity querying algorithms. Its multiresolution nature allows us to implement interesting features for all these tasks.

Details of the modelling of the environment can be added on finer levels enabling the usage of view-dependent rendering algorithms. Path planning can take advantage of a multi-scale structure in hierarchical algorithms. Proximity queries take advantage of the underlying efficient topological structure and are speeded up by the dynamic adaptive refinement of the environment partition. Moreover, this unified approach allows the simulation to take place on any 2-manifold environment and eases the management of dynamic scenes.

We showed the efficiency of our approach through a comparison with a well established library (RVO2) and spatial hashing techniques. All our example use the RVO2 library to compute the agents velocities. We want to explore the use of other approaches and especially rule based ones that could take advantage of our agent centered multiresolution model.

Finally, the combinatorial maps model we use is defined in any dimension and may thus encode multiresolution volumetric meshes. The particle tracking system presented in [21] is defined in 3D as well. We are convinced that good results could be obtained for 3D navigation as required for flocking simulations of birds or fishes.

References

[1] A. Braun, S. Musse, L. de Oliveira, and B. Bodmann. Modeling individual behaviors in crowd simulation. In *CASA*, page 143, 2003.

[2] D. Thalmann, C. O’Sullivan, P. Ciechomski, and S. Dobbyn. Populating virtual environments with crowds. In *Eurographics 2006 Tutorial Notes*, 2006.

[3] N. Badler, J. Allbeck, and N. Pelechano. *Virtual Crowds: Methods, Simulation, and*

Control (Synthesis Lectures on Computer Graphics and Animation). Morgan and Claypool Publishers, 2008.

[4] S. Goldenstein, M. Karavelas, D. Metaxas, L. Guibas, E. Aaron, and A. Goswami. Scalable nonlinear dynamical systems for agent steering and crowd simulation. *Computers & Graphics*, 25(6):983–998, 2001.

[5] Hao Jiang, Wenbin Xu, Tianlu Mao, Chunpeng Li, Shihong Xia, and Zhaoqi Wang. Continuum crowd simulation in complex environments. *Computers & Graphics*, 34(5):537–544, 2010.

[6] S. J. Guy, J. Chhugani, S. Curtis, P. Dubey, M. Lin, and D. Manocha. Pledestrians: a least-effort approach to crowd simulation. In *EG SCA*, pages 119–128, 2010.

[7] C. Loscos, D. Marchal, and A. Meyer. Intuitive crowd behaviour in dense urban environments using local laws. In *TPCG*, page 122, 2003.

[8] F. Lamarche and S. Donikian. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum*, 23(3):509–518, 2004.

[9] W. Shao and D. Terzopoulos. Autonomous pedestrians. In *EG SCA*, pages 19–28, 2005.

[10] P. Fiorini and Z. Shillert. Motion planning in dynamic environments using velocity obstacles. *Int. Journal of Robotics Research*, 17:760–772, 1998.

[11] S. Guy, J. Chhugani, C. Kim, N. Satish, M. Lin, D. Manocha, and P. Dubey. Clearpath: highly parallel collision avoidance for multi-agent simulation. In *EG SCA*, pages 177–187, 2009.

[12] Jur van den Berg, S. Guy, M. Lin, and D. Manocha. Reciprocal n-body collision avoidance. In *Robotics Research*, volume 70 of *Springer Tracts in Advanced Robotics*, pages 3–19. Springer, 2011.

[13] F. Tecchia, C. Loscos, and Y. Chrysanthou. Visualizing crowds in real-time. *Computer Graphics Forum*, 21(4):753–765, 2002.

[14] M. Teschner, B. Heidelberger, M. Müller, D. Pomerantes, and M. H. Gross. Optimized spatial hashing for collision detection of deformable objects. In *VMV*, pages 47–54, 2003.

[15] S. Paris, S. Donikian, and N. Bonvalet. Environmental abstraction and path planning techniques for realistic crowd simulation. *Computer Animation and Virtual Worlds*, 17(3-4):325–335, 2006.

[16] A. Sud, R. Gayle, E. Andersen, S. Guy, M. Lin, and D. Manocha. Real-time navigation of independent agents using adaptive roadmaps. In *VRST*, pages 99–106, 2007.

[17] R. Gayle, A. Sud, E. Andersen, S.J. Guy, M.C. Lin, and D. Manocha. Interactive navigation of heterogeneous agents using adaptive roadmaps. *IEEE TVCG*, 15(1):34–48, 2009.

[18] A. Botea, M. Müller, and J. Schaeffer. Near optimal hierarchical path-finding. *Journal of Game Development*, 1:7–28, 2004.

[19] P. Kraemer, D. Cazier, and D. Bechmann. Extension of half-edges for the representation of multiresolution subdivision surfaces. *The Visual Computer*, 25(2):149–163, 2009.

[20] CGoGN. Combinatorial & geometric modeling with generic N-d maps. `http://cgogn.u-strasbg.fr`.

[21] T. Jund, D. Cazier, and J.-F. Dufourd. Particle-based forecast mechanism for continuous collision detection in deformable environments. In *SIAM/ACM GDSPM*, pages 147–158, 2009.

[22] T. Jund, D. Cazier, and J.-F. Dufourd. Edge collision detection in complex deformable environments. In *VRIPHYS*, pages 69–78, 2010.