Computers & Graphics 34 (2010) 136–144

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Emergency crowd simulation for outdoor environments

Oğuzcan Oğuz, Ateş Akaydın, Türker Yılmaz, Uğur Güdükbay *

Bilkent University, Department of Computer Engineering, Bilkent, 06800 Ankara, Turkey

ARTICLE INFO

Article history:
Received 1 July 2009
Received in revised form
12 October 2009
Accepted 16 December 2009

Keywords:
Emergency
Crowd simulation
Occlusion culling
Outdoor environments

ABSTRACT

We simulate virtual crowds in emergency situations caused by an incident, such as a fire, an explosion, or a terrorist attack. We use a continuum dynamics-based approach to simulate the escaping crowd, which produces more efficient simulations than the agent-based approaches. Only the close proximity of the incident region, which includes the crowd affected by the incident, is simulated. We use a model-based rendering approach where a polygonal mesh is rendered for each agent according to the agent's skeletal motion. To speed up the animation and visualization, we employ an offline occlusion culling technique. We animate and render a pedestrian model only if it is visible according to the static visibility information computed. In the pre-processing stage, the navigable area is decomposed into a grid of cells and the from-region visibility of these cells is computed with the help of hardware occlusion queries.

1. Introduction

Crowd animation is a crucial problem in computer graphics since the crowds are a major component of a virtual scene. It is so important that without crowd animation or with an improper application, a virtual scene would not be realistic at all, regardless of the qualities of the other components. Many applications of computer graphics, such as computer games, virtual reality applications and animated films, require high quality crowd animation. Physically correct crowd simulations also have other application areas, such as psychology, transportation research, and architecture.

The requirements of a crowd simulation might be application-specific. For instance, computer games require a crowd simulated in real time, and so they would sacrifice some of the characteristics of the crowd animation that affects realism, whereas the film industry uses sophisticated and computationally-costly techniques to produce more realistic animations. In both of these diverse areas, crowd animations need to be controllable: animated crowds should exhibit the intended behavior. Other types of information may also be incorporated like statistical data, in order to realistically perform simulations as in transportation research.

Almost all the techniques to animate crowds are agent-based. In agent-based approaches, every single agent has its own computation of future behavior. Path planning and collision avoidance are performed for each agent in the scene. This approach is the most natural one since it is the way that real crowds behave: each human makes her/his own motion decisions according to the information s/he has, such as visibility, information about the destination, and proximity. However, this approach has the disadvantage that when animating a large group of people, it requires large computational time. Agent-based models have the flexibility to add any intended variation to the animated crowd, since each agent can be modeled differently but it needs expertize to model every agent consistently.

Recently, another approach to crowd animation problem has been proposed that is inspired by continuum mechanics. This approach treats the crowd as a continuum and animates the crowd flow by the help of a set of equations tailored to simulate crowd motion realistically. Continuum perspective unifies global path planning and collision avoidance since the continuum equations take the goals, obstacles and other agents into account when predicting the motion of an agent.

One of the application areas of crowd simulation is simulating emergency behavior of crowds. Emergency situations frequently arise in cities. Incidents, such as fires, explosions, or terrorist attacks, can stagger emergency situations. Training animations, video games, and animated movies can make use of emergency simulations. If the emergency simulations are realistic and flexible enough, then foreseeing the possible problems that may arise in an emergency situation will be possible.

For the case of outdoor environments of a city, an emergency situation causes the nearby crowd to panic and behave different than normal. During an emergency situation, decisions of people are mostly reflex-based and do not vary greatly from a person to another; most people try to run away and hide in reaction to the incidents. Thus, in contrast to the simulation of a normal crowd

* Corresponding author. Tel.: +90 312 290 1386; fax: +90 312 266 4047.
E-mail addresses: oguzcan@cs.bilkent.edu.tr (O. Oğuz), akaydin@cs.bilkent.edu.tr (A. Akaydın), turker.yilmaz.trk@gmail.com (T. Yılmaz), gudukbay@cs.bilkent.edu.tr (U. Güdükbay).

0097-8493/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2009.12.004

behavior, the simulations aiming emergency situations require an approach that directly supports the simulation of homogeneous behavior.

In this work, we take a continuum perspective to simulate the crowds during emergency situations. In a large virtual city, we only simulate agents that are in the region of interest defined around the viewer's location. When there is no incident in the scene, crowd is simulated to exhibit normal behavior: agents walk and try to reach their goals. When an incident is introduced, the agents that are aware of the emergency situation switch to the panic behavior: they try to run away from the incident. The agents that are not aware of the emergency situation still try to walk their goals but they avoid incident region this time. An agent becomes aware of the emergency when the information radiating from the incident reaches at the position of the agent or when the agent is caused to panic by another agent that is already aware of the emergency situation. We represent the incidents as discomfort regions with discomfort values decreasing as the distance of the cells to the incidents increases.

We model each pedestrian as an articulated body with links and joints and the whole body is a polygonal model. Each pedestrian is capable of making realistic motions. This increases the realism of the rendered crowds but a lower number of pedestrians could be rendered with this approach as compared to the state-of-the-art approaches using impostors.

We employ an occlusion culling technique to cull the occluded agents. Culling the occluded agents is essential for speeding up the simulations in virtual cities since there would be a large number of agent models to draw at each step of an animation and most of the agents would not be visible to the view-point. The details of the occlusion culling technique used can be found in [1]. By extending the occlusion culling to navigable areas, we became able to render the agents only if they are actually visible with respect to the contemporary view-cell of the user. Static cell-to-cell visibility is computed in the preprocessing phase by shrinking the occluders and computing visibility information using hardware occlusion queries. The resulting system is able to realistically simulate a significant number of agents in emergency situations with high frame rates. The main contributions of this work can be summarized as follows.

- A continuum dynamics-based crowd simulation framework that is specifically designed to simulate and visualize crowds for emergency situations in outdoor urban environments.
- An offline occlusion culling technique to cull the occluded agents. Cell-to-cell visibility is computed in the preprocessing phase using occluder shrinking and hardware occlusion queries.

2. Related work

Most of the work about crowd animation is agent-based in which each agent plans its motion individually. The agent-based approaches could get quite complex, and thus computationally demanding, when one wants to consider cognitive aspects, such as knowledge, learning and emotional states [2–4]. The visibility and path planning is added to the Funge's work by Terzopoulos and Shao [5]. Massive Software, a production quality tool, gives the animator full responsibility to define each agent's behavior [6]. However, Massive Software requires considerable effort to come up with sufficiently realistic simulations of large groups of people. Legion [7] and Simulex [8] software aim analysis and design of crowd dynamics, and require expertize to come up with crowd simulations.

Luo et al. propose a human behavior modeling framework that naturally reflects human decision-making process [9]. The proposed framework adopts a layered architecture to imitate a person's awareness of the situation and consequent changes on the internal attributes defining individual and crowd behaviors. The framework is generic and shown to realistically simulate a small crowd of people under user defined conditions.

Pettré et al. represent navigable regions for crowd simulation with circles of variable radius [10]. Agents are assigned various paths between two distant circles, where each path consists of a series of connected circles. With the help of multiple simulation and visualization levels, on not too complex environments, this framework is able to simulate and visualize thousands of agents. In contrast to our work, agents are assigned static paths in their work.

Pelechano et al. simulate the agents in a continuous space with a forces model; the movement of the agents are driven by a set of attractors while the agents avoid the obstacles and the other agents in the scene [11]. In their model, agents may have varying personalities and roles, and the communication between the agents provide information sharing about the hazards and exit routes in the building. Their work is mainly developed for indoor emergency evacuation scenarios.

Particle-based approaches are more suitable for simulating large crowds because the computational cost for each individual is much less than the one in agent-based approaches. Helbing simulates the crowd as a self-driven many-particle system [12]. In this model, crowd dynamics of pedestrians are driven by a mixture of psychological and physical forces.

Chenney defines flow tiles for representing and designing velocity fields easily [13]. Once the flow tiles are defined between the buildings, congestion avoidance can be achieved easily since the flow tiles are divergence free. However, the flow tiles approach does not address all the concerns of a crowd animation; for example, we cannot assign goals to a single pedestrian or a group of pedestrians.

Hughes is the first one to view a crowd as a continuum and derive the set of equations to simulate large crowds [14]. Hughes defines the crowd as a density field and uses differential equations to derive the motion of the crowd. The density field is driven towards the goal by the help of a potential function; density follows the direction of gradient vector of the potential function. The model proposed by Hughes is confirmed with real crowd data [15].

An inspiration from Hughes' model resulted in continuum crowds [16]. Continuum crowds approach makes the simulation of crowd flows possible by transforming Hughes' continuous crowd field into a particle representation. Treuille et al. make numerous improvements to Hughes' model to make the simulation able to exhibit a number of visually interesting and empirically proven behavior.

AMD Froblins demo demonstrates a different use of continuum crowds approach [17]. Global path planning is done by the continuum approach in coarse resolution and local path planning is handled differently. Their justification for this combination is that solving the Eikonal equation at a high resolution to perform local and global path planning for large numbers of agents is prohibitively expensive for a real time application. So they augment the global Eikonal solution with a local avoidance model that resolves the fine-grained obstacles. Local path planning is typically handled by a continuous cycle of examining the nearby environment and reacting based on the discovered information. The system is implemented on CPU and GPU. The parallelization enabled by the GPU performs very well: nearly 65 000 agents are simulated and rendered in real time with simple cylindrical agent models. The main bottleneck is said to be rendering. However, the

performance is higly dependent on the GPU power and the GPU implementation is not very straightforward.

Tecchia et al. propose a framework for the simulation of virtual crowds with an emphasis on visualization [18]. Crowd behavior is simulated by placing a 2D grid containing four layered behavioral information on the environment and moving the agents according to the behavior data stored on the grid. Complex behaviors can be achieved but the resulting behaviors are static for a specific simulation. They use an image-based rendering approach where a set of pre-computed textures are displayed on impostors according to the viewpoint and the frame of animation.

Social psychologists have carried out extensive research on pedestrian behavior in reaction to an emergency situation [19,20]. Panicking people have found to show maladaptive escaping behaviors different than normal socially-controlled behaviors [21]. In the case of bottlenecks, such as doors and exits for a building, panic behaviors cause jamming and overcrowding [20].

3. Emergency crowd simulation for outdoor urban environments

The agent-based approach to crowd simulation is an approximation to the real life crowds and so it is a more natural way of simulating a crowd than particle-based approach. In an agent-based crowd simulation, each agent can behave and react uniquely as its artificial intelligence model may be driven by its unique parameters. Each agent can have its own decisions and reactions, pursue its goals and interact with the other agents. However, in an emergency situation, people generally show more homogeneous behaviors: they all try to escape some way. There would be fewer interactions between the agents if the incident occurs outside, as in a building, people may show a more organized behavior due to the past evacuation practices. For these reasons, we take a continuum dynamics-based approach [16]. In a 2D grid, the agents are considered to be particles and their flow is driven by dynamic vector fields. Crowd can be divided into a small number of behavior groups; the vector fields of each group are computed based on minimizing a potential function at every time step. Computational cost is mainly dependent on the grid size and the number of groups. Since we compute a vector field for each group at each time step and move all the agents in the group accordingly, the cost per agent is amortized.

The crushing of people during emergency situations is less crucial in outdoor environments than it is in indoor environments; it does not severely affect the way individuals move. To this end, we do not compute any interaction forces between individuals. We do not perform any collision detection since the continuum approach takes care of collisions of agents as far as the resolution of the grid permits.

3.1. Navigable space extraction

To employ continuum approach on a city environment, we need a distinction between the navigable and the non-navigable grid cells. Non-navigable cells can be the cells occupied by a building or a road. Agents cannot penetrate to non-navigable cells so we do not need potential functions to be computed for non-navigable cells. By excluding non-navigable cells from computation of potential functions, we prevent continuum flow from going through the boundary cells; this saves the need for collision detection against static structures.

Navigable space needs to be extracted as accurate as possible. However, the accuracy of extraction is determined by the resolution of the grid since we must define a cell either as navigable or not [22]. To get the ground level cells that intersect with a building, we test each cell against all the primitives of the buildings. Redundant tests are avoided by testing a cell only against the buildings whose bounding boxes intersect the tested cell. After these tests are performed, only the cells that intersect the primitives defining the surfaces of the buildings are extracted.

3.2. Local continua using active grid

In the continuum crowds approach, the cost of computing a potential function that is specific to a group of agents is only dependent on the resolution of the 2D regular grid. The potential function is computed everywhere on the grid, and all of the agents are moved according to the gradient of the potential function. If we have a large grid that covers the entire city area or a large portion of the city area, then it would be too costly to compute potential functions for all the groups across the grid. In an urban environment, a viewer can only see a tiny portion of the whole city due to the occlusion caused by the nearby buildings. Thus, computing the potential functions everywhere would be redundant. The viewer would not see the simulated agents out of interest area. To avoid high computational cost and redundancy to simulate crowds in the whole city area, we only care about the grid that covers the interest area. That is, the potential functions are computed and the agents are simulated only in a small part which we call the active grid.

Due to aligned streets in a city, the grid portion that covers all the regions that can be seen from a view point can be huge; so, the active grid cannot be defined to cover all the visible regions. Instead, we define the active grid as a fixed-size rectangular grid that has the viewpoint at its center. This definition has some drawbacks. Not simulating the agents that are out of the active grid degrades the continuity: the user cannot follow and view an agent if the agent leaves the active grid. Since we aim to simulate and demonstrate the behavior of the crowd during emergency situations, the travel of a single agent is not crucial. Another drawback is that we can only animate and render the simulated agents so the agents that enter or leave the active grid would pop in or pop out. If the viewer is located on the ground, the viewer would be surrounded by agents and if the active grid is large enough, the user would not notice popping artifacts since they would be far and occluded. In the case of fly-through scenarios, the active grid needs to be defined large enough to cover whole region that can be seen.

3.3. Normal crowd behavior before the incident

To demonstrate the effect of an incident on the surrounding crowd, we need to simulate the normal crowd behavior before the incident. This simulation may span a small period of time and the crowd switches to the emergency behavior when the incident occurs. Since we have an active grid of fixed size and we only simulate the agents in it, we need to maintain a pedestrian flow inside the active grid. New agents are continuously added to the active grid to prevent the grid from becoming empty as some agents leave the grid. We set the goals for an agent as the cells that are distant to the cells to which the agent is added, so that the agent travels through most of the active grid. To minimize pop-in and pop-out artifacts, we add new agents at the border cells of the grid and direct them to some distant border cells.

3.4. Emergency behavior

In our system, agents try to get away from the incident region in reaction to the incident. Since the active grid can be placed

anywhere on the city, the orientation of the navigable and non-navigable cells inside the active grid tends to differ greatly. This structure inside the active grid affects the behavior of the people. For instance, if we place the active grid in an open space in the city, everywhere would be navigable and people could run to any direction. In contrast, if the incident occurs in a street, people would try to get away by running to either ends of the street. To adapt to different structural variations, we place the goals for the escaping groups only at the navigable border cells of the active grid. In this way, the agents try to find their way out of the active grid. However, this behavior is not sufficient since an agent may run through the incident region to minimize the potential function when it is less costly to take a path going through the incident region (see Fig. 1(a)). This configuration only makes sense if the incident is placed at the center of the active grid. For this reason, we take the position of the incident into account and assign the cells surrounding the incidents with appropriate discomfort values. Assigned discomfort value for a cell decreases as the distance of the cell to the incident location increases. So we define an incident point as a local maximum inside the active grid. In Fig. 1(b), we want the agents to follow the vectors; moving outwards from the incident. The discomfort field for each incident i is defined as

$$d_i(x) = \begin{cases} d_{i,\max} - |d_{i,\max} - d_{i,\min}| \times \left(\frac{\|x-p_i\|}{r_i}\right)^k & \text{if } \|x-p_i\| < r_i \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where p_i is the position of the incident i, r_i is radius of the region that can be affected by the incident and, $d_{i,\max}$ and $d_{i,\min}$ are the maximum and minimum discomfort values caused by incident i, respectively, and k is a positive constant. The total amount of discomfort value at a cell is computed by summing discomfort contributions of all the incidents:

$$d(x) = \sum_i d_i(x). \quad (2)$$

When an emergency situation arises, the maximum traveling speed ($f_{\max}$) of the agents increases and their tolerance to discomfort is increased by scaling γ in the unit cost definition [16].

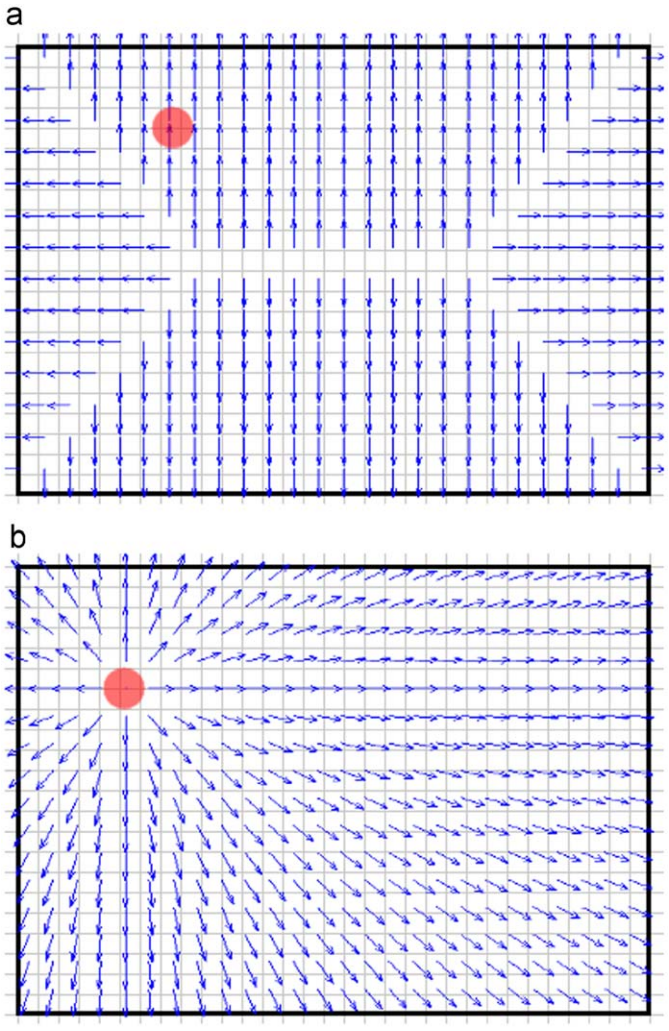

Fig. 1. (a) The vector field that the agents would follow if the border cells are defined to be goals. Some agents would go through the incident region (filled circle) in order to take the shortest path to a border cell. (b) The ideal vector field that the agents should follow when an incident occurs at the filled circular region.

3.4.1. Smoothing agent paths

Computation of the potential fields for the agent groups is the most costly operation in the simulation process [16]. This cost increases dramatically as the number of agent groups within the system increases. Therefore, to achieve real-time simulation rates, we decouple the renderer from the dynamic computation of group potential fields. In our current implementation, we have four directional agent groups (north, south, east, west) to populate the area specified by the active grid. We also have a fifth emergency group, which is populated when an emerging incident is introduced within the system. Thus, the simulator has five group steps plus one short idle step for handling minor operations, like agent exchanges between groups.

Upcoming position and velocity vectors for the agents will only be available (to be used in interpolation) after the six steps in the simulator algorithm are completed. At each step for a group, only the potential field of this subject group is evaluated. In addition to this, the actual positions and velocities of all agents in the system are evaluated at each simulator step by interpolating the previous and the current navigational information available. At the same time, the upcoming navigational information is computed using the current values. We use Hermite cubic spline interpolation scheme to compute the inbetween values of the agent positions and spherical linear interpolation to compute the inbetween agent directions at each step. In this way, we also smoothen the agent paths in a less costly manner.

3.4.2. Propagation of panic

When an incident occurs, all of the surrounding agents would not be aware of the emergency situation instantly. We define an awareness regions around the incident regions that get larger with time; and so, it takes longer to farther agents switch to emergency behavior. However, the agents that cannot see the incident, are not affected by this awareness region. To check that if an agent is able to see the incident, we use the precomputed visibility information, which is explained in the next section. In addition to the awareness region, we also use a panic field to model the behavior of agents being affected by scared agents nearby. At each frame, each agent in the emergency group applies a panic value to the neighboring grid cells with respect to the agents' position on the grid. For a particular grid cell, the panic value may stack if there are several agents nearby. And if an agent behaving normally happens to enter one of these cells with applied panic (exceeding a specified threshold), then this agent will join the emergency group as well. It is natural though for this panic field to dissipate over time. At each frame we dampen out the remaining panic in unpopulated cells by multiplying the panic

```
For each timestep:
  1. If the view points distance to active grids center is above some threshold:
    1.1 Reposition the active grid according to the view point.
    1.2 For each group:
      1.2.1 Clear all the goals.
      1.2.2 Update the goals with the navigable border cells of the active grid.
  2. If any new incident is to be added, add it.
  3. For each incident:
    3.1 If the incident's lifetime is over, remove the incident and clear its effects.
    3.2 Dissipate awareness area caused by the incident.
    3.3 Dissipate discomfort values caused by the incident.
  4. For the emergency group, dissipate panic field caused by the panicking agents.
  5. For each of the normal behavior groups:
    5.1  Add new agents at the navigable border cells of the active grid.
    5.2  Move the agents that switch to emergency behavior, to the emergency group.
  6. Convert the crowd to a density field.
  7. For each group:
    7.1 Construct the unit cost field.
    7.2 Construct the potential field and its gradient.
    7.3 Update the agents locations.
```

Fig. 2. The simulation algorithm executed for each time step.

value in these cells with a specified constant between 0 and 1. The panic in these cells are set to 0 when they become lower than a specified low-threshold. Only the cells within the active grid are applied panic dampening to reduce the computational cost. The emergency crowd simulation algorithm is given in Fig. 2.

4. Crowd rendering

In our system, we use Cal3D skeletal animation library [23] to animate the agents. For each agent to be drawn, a pose is computed and the resulting mesh is rendered. However, not all the agents are seen by the user: a simulated agent may be occluded by buildings or it may be out of the view frustum. At a frame, if an agent is not visible to the view point, we do not only avoid the rendering computation but also the pose update computation.

4.1. Occlusion culling

The simulation happens in a city so there would be lots of buildings that may cause a great amount of occlusion, especially at the ground level viewing positions. In contrast, the occlusion of an agent by the crowd is not crucial except for very dense regions; thus, we only care about the buildings as occluders. Since all the buildings are static, their occlusion effect can be computed offline. Offline computation of visibility requires a from-region visibility approach in which the scene is decomposed into a number of view cells at which the view point can be located. Since it is more like an open space, we prefer a uniform grid of view cells at the outdoor environments, whereas it is more appropriate to define the rooms as the view cells connected to each other with portals in a building. The visibility information for each navigable view cell is pre-computed; that is, all the objects seen at every point in the view cell is extracted and stored [1]. However, we aim to compute visibility of the agents that are dynamic and their position cannot be known for sure. Unlike what a regular occlusion culling algorithm does, we choose to cull other navigable cells so as to determine the cell to cell visibility for the outdoor environment. We define a uniform grid of target cells, which are essentially 3D boxes with the height of an agent and placed at the ground level. Visibility information of a target cell for a view cell can be used as follows: the skeleton pose is computed for an agent and the resulting mesh is rendered, only if the agent is in a target cell that is visible to the view cell in which the view point resides.

A view cell theoretically includes infinite number of view points so it is impossible to sample the visibility at every point inside a view cell. There has been several geometric and image-based solutions proposed for this [24,25]. The approach we take is based on the notion of occluder shrinking [26]. By shrinking the occluders present in the scene and sampling the visibility at the center of the view cells, conservative occlusion culling can be achieved. Once the shrunk version of every occluder in the scene is computed, we need to check the visibility of every navigable target cell for each navigable view cell. In this process, we make use of hardware occlusion queries. For a navigable view cell, a target cell is tested against all the buildings by drawing the shrunk versions of the buildings first and then issuing an occlusion query for the target cell. Since all the shrunk buildings are drawn prior to the occlusion query, occluder fusion is achieved. Visibility information calculated by using hardware occlusion queries is certainly conservative for the configuration in which hardware occlusion query is issued. However, hardware occlusion queries are dependent on the limited viewport resolution and prone to sampling and precision errors. The calculated visibility can be erroneous for a different configuration in which the clipping window covers a smaller area of the scene. For a different configuration, a far away target cell that did not generate a fragment before may generate a fragment, and so, it may become visible. Additionally, due to sampling and the limited viewport resolution, a target cell that was completely occluded before may become visible. In order to get a visibility information as precise as possible, the viewing parameters are adjusted during the visibility computation for a target cell so that the target cell is zoomed to the maximum extent to create a greater resolution during occlusion culling and prevent errors.

Due to the structure of a city, nearby buildings would occlude most of the target cells for a view cell. In the light of this fact, the number of occlusion queries issued for a view cell can be reduced by computing visibility information for a coarser-grained grid first. We make use of region quadtrees for this purpose.

Once the visibility information is computed, the extracted information need to be stored in main memory during runtime. If we store the visibility of every target cell for every view cell, the memory space needed to store the visibility information would be on the order of $\Theta(v^2 \times t^2)$, where v is the number of view cells and t is the number of target cells. Even though it is enough to store a byte for each view cell-target cell pair and we only store visibility information for navigable view cells, the required memory space can be huge, depending on the resolutions of the view cells and target cells grids. For this reason, we store the quadtree, which is used in the visibility calculation, for each navigable view cell, and we store the visibility information for target cells in the visible leaves of the quadtree. We also perform view-frustum culling. If an agent is visible, then its bounding box is tested against the view frustum. For every agent, three levels of detail of the mesh geometry are pre-computed and stored [23]. Depending on the distance between the camera and an agent, one of the three levels of detail is used to render the agent.

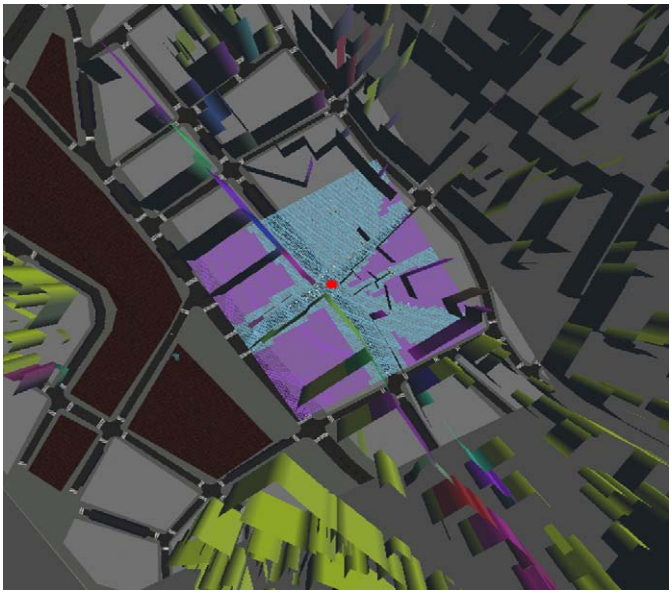

Fig. 3. Top view of the scene. The viewer is located at the red colored region. The portions of the active grid that are occluded by the buildings are colored in purple. The agents in the purple colored regions are culled. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

5. Results

The proposed algorithms were implemented using C++ Programming Language. The simulated crowd and the city environment are visualized using OpenGL. Hardware-based occlusion culling is performed with the help of *GL_NV_occlusion_query* extension of OpenGL, provided by NVIDIA Corporation [27]. Skeletal animation of the simulated pedestrians are computed using Cal3D. In order to get a higher performance for the priority queue used in the potential function computation, we use the *p_queue* structure of LEDA [28]. The test platform is a personal computer with Intel 2 GHz Centrino Duo processor, 2 GB RAM, and an NVidia GeForce Go 7400 graphics card. The city model used is the Vienna2000 Model. The city model is composed of 805 buildings and a total of 23 K triangles. The extracted navigable space at the ground level is defined on a grid of 1000×817 cells.

The size of the view cells to be used in from-region occlusion culling is not dependent on the cell size of the simulation grid. The view cell size need to be defined according to the size of the occluders in the scene. If the view cell is set to be too large, then the occluders would be shrunk extensively and their occlusion effect would be mostly lost. On the other hand, if we set the view cell size to be too small, then the preprocessing to compute from-region visibility information would take too long, and the memory space requirement to store the visibility information would be too high.

The size of the target cells should be defined according to the structure of the urban environment. We either cull or process all the agents inside a target cell according to the visibility of the target cell. Large target cells would preclude fine occlusion culling. On the other hand, too small target cells would result in a higher number of target cells, and thus requiring longer preprocessing time and larger memory space requirement. Making the target cells too small would also waste the benefits of spatial coherence of visibility. In our tests, the resolutions of the view cell grid and the target cell grid are both set to 170×130. Cell-to-cell visibility computation for this configuration take less than 20 min. The memory space required to store the visibility information is 48 MB. The use of calculated visibility information is depicted in Fig. 3.

We ran a series of simulations with varying active grid size. The simulations are initiated without any emergency situations. Until an emergency situation is introduced, normal crowd behavior is simulated. When an emergency situation happens, the agents that become aware of the incident switch to emergency behavior. Fig. 4 illustrates the agents seen before and after the occurrence of the incident.

Fig. 5 shows stacked graphs of number of agents during the simulations. All the simulations are run with the same camera paths. The total number of agents may change during a simulation since some agents may leave the active grid; the graphs show the total number of agents in the active grid. The similarity of the graphs is an indication of the scalability of the proposed occlusion culling technique. Table 1 presents the average frame rates for crowd simulations, which are run with and without occlusion culling, with different number of agents. The speed-up due to occlusion culling increases with the crowd size since the ratio of the occluded agents to the visible agents would be higher in large crowds. This indicates that the occlusion culling for crowd rendering is more effective and essential when visualizing large number of agents. The highest level of detail for the male and female models are composed of 424 and 350 polygons, respectively. The medium and the lowest levels of detail contain approximately 65% and 45% of the polygons in the highest level of detail version for both male and female models. The frame rates change according to the number of agents rendered for a particular viewpoint; the frame rates are higher at the regions where more agents are occluded.

The number of frames are not equal to the crowd simulation steps taken, since we decouple the renderer from the crowd simulator. The renderer interpolates the pose and the velocity of the agents between the two adjacent simulation steps. For each simulation step, six interpolated frames are drawn. It should be noted that the frame rates for the simulations are affected by the time required to draw the geometry of the urban scene. In order to reduce this effect, we use a city model with a simple geometry. Furthermore, the visualization of the geometry of the city incorporates a high level of occlusion culling for the building models.

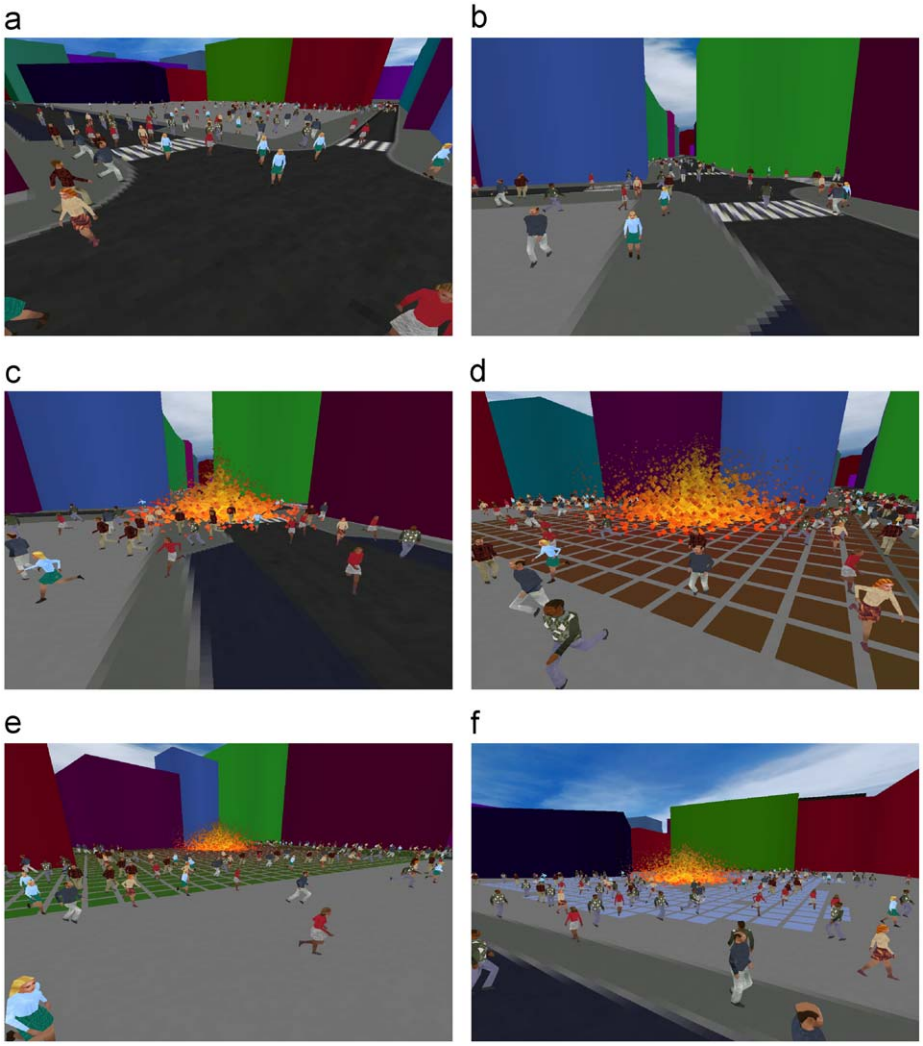

Fig. 4. Still frames from a crowd simulation in an urban environment: (a) and (b) show normal crowd behavior; (c), (d), (e), and (f) show emergency crowd behavior. The grid on the floor in part (f) shows the panic field spread out by the escaping agents.

We also performed simulation tests using simple boxes for the body parts of the agents. In these tests, models are rendered in wireframe without any texturing. Table 2 presents the average frame rates for these simulations. The results given in Table 2 are an indication of the rendering performance when simple geometric or impostor based models are used, which are less costly than the detailed models that include complex geometry for the body parts and texturing.

In the simulation tests of emergency behavior, we observed that the agents follow smooth and efficient paths while escaping from the incident. In some configurations of the nearby structures and the crowd distribution, complex behaviors emerge. For instance, escaping agents do favor but do not always take the paths that are composed of the points that are strictly growing away from the incident region. This behavior is more frequent at the regions that are far away from the incident since coming closer to the incident is harder in the close proximity of the incident due to higher discomfort values near the incidents.

6. Conclusion

We propose a framework to simulate and visualize pedestrian crowds in emergency situations. The proposed crowd animation system simulates the agents with a continuum dynamics-based approach applied to the crowd model of Hughes [14,16]. During outdoors emergency situations, people are coarsely distributed and show homogeneous behaviors. The taken continuum-approach is able to simulate a number of agent groups up the resolution of the simulation grid, and the cost per agent is amortized for each group of agents. Thus, the continuum-based approach is more suitable for outdoors emergency simulations, as opposed to the computationally-demanding agent-based approaches. We represent the pedestrian crowds in the emergency situations by a number of behavior groups. At each step, the potential function is computed for each behavior group, and the positions of the agents are updated accordingly. For the city model in which the simulation takes place, we first extract the navigable space and render all the cells occupied by the buildings as non-navigable for the agents. In a large city model, we only simulate the near proximity of the viewer by defining and maintaining an active grid around the view point. The active grid provides a good way to simulate the crowd in the local proximity of the viewpoint. One disadvantage of using such a local grid is that, in some cases, the potential function and the paths of agents may show steep changes. This is because the goals of the behavior groups are defined on the navigable borders of the active grid and the navigability of the borders could show steep changes when

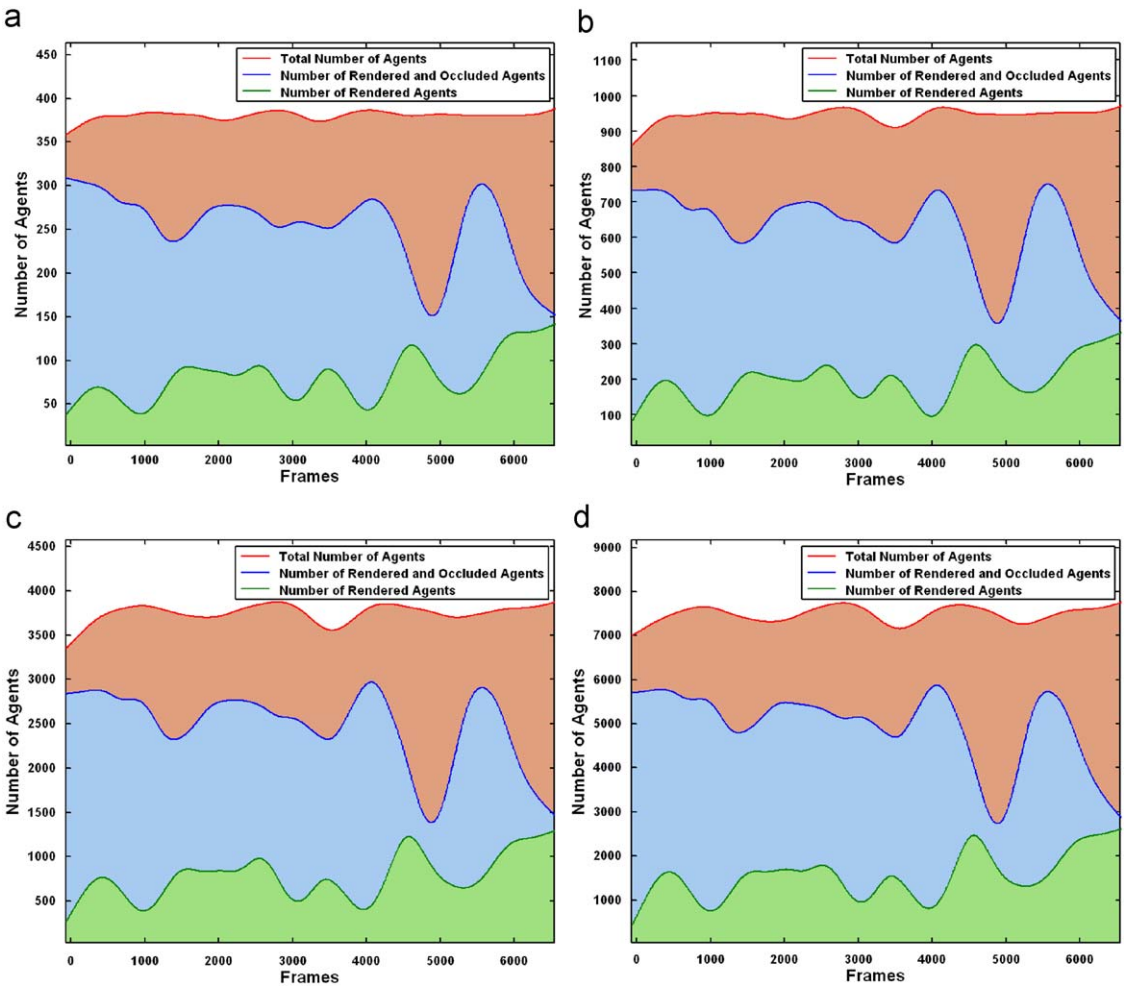

Fig. 5. The effects of occlusion culling and view frustum culling on crowd rendering. The stacked graphs depict the total number of agents, the number of rendered and culled agents, and the number of rendered agents for crowd populations of (a) 400, (b) 1000, (c) 4000, and (d) 8000 agents. Blue region indicates the number of culled agents due to occlusion whereas red region indicates the culled agents due to view frustum. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
The average frame rates (frames per second) for crowd simulations that are run with different number of agents are shown.

Number of agents	Average fps without occlusion culling	Average fps with occlusion culling	Speed-up
400	38.64	45.76	1.18
1000	21.99	31.32	1.42
2000	14.11	18.98	1.35
4000	8.20	12.27	1.50
8000	4.49	7.09	1.58

Simulations are run with and without occlusion culling. The performance gain due to occlusion culling can be seen on the speed-up column.

Table 2
The average frame rates (frames per second) for crowd simulations with simple geometric models for agents.

Number of agents	Average fps with simple geometric models
400	57.67
1000	48.20
2000	29.98
4000	23.79
8000	13.95

During the simulations, occlusion culling is employed.

the viewpoint is moved. However, if the active grid is chosen to be large enough, the changes in the environment while the active grid is moving will not affect the potential fields and the agents flow significantly. The emergency behavior in reaction to the incidents inside the active grid is achieved by placing the goals for the escaping crowd at the navigable border cells of the active grid. In order to achieve a crowd flow radiating outwards from the incident points, for each incident, we define a Gaussian-like discomfort field centered at the incident region. Thus, when the simulation is running with the proper discomfort values set at the cells surrounding the incidents, the crowd would move in the direction of the gradients of the contours that are radiating from the incident points. In our simulation tests, we observed that the agents escape from the introduced incidents in a sensible way. The agents find escape paths through the streets and the spaces in the city avoiding the incident regions.

We also propose an extension to the previous work, a from-region occlusion culling method to avoid the animation and rendering costs of the simulated pedestrian models that are occluded by the buildings. First, we decompose the space where the view point can be located into a uniform grid of view cells. Then, another uniform grid of target cells in which the agents could reside is formed. The cell-to-cell visibility between the view cells grid and the target cells grid is computed with the help of the hardware-based occlusion queries. We query the visibility of each

target cell for each view cell. By shrinking the occluders in the scene by the maximum distance traveled in a view cell, and sampling the visibility at the center of the view cells, conservative occlusion culling is achieved. During the simulation, the pose of the pedestrian model is computed and the resulting mesh is rendered only if the target cell that contains the agent is visible to the view point. Used together with view frustum culling, the proposed occlusion culling method enables animation of the simulated crowd at high frame rates, with detailed pedestrian models. The technique is effective while rendering large number of agents in a city like environment; most of the agents in the scene would be occluded due to the nearby occluders. The effectiveness of the technique would be higher if there are a large number of pedestrians in the scene and the pedestrian models are detailed. This technique is not suitable if the simulations are run in an open environment or there are a small number of pedestrians with simple geometric models.

In the future, we plan to behaviorally diversify the simulated crowd by adding new behavior groups. The new behavioral groups would compose of the agents that try to hide in buildings or try to intervene the incident during emergency situations. To make the active grid adaptive to the nearby structures would represent better the interest area of the viewer. In addition, using parallel algorithms for level set formulations [29], a substantial speedup might be achieved if the simulation process can be extended to parallel domains, such as GPU or multi-core domains. In the current implementation, the view cells' grid is placed on the ground level. The culling approach can be extended to fly-through scenarios by defining a view-cells grid for each height level in which the view position can be located.

Appendix. Supplementary data

Supplementary data associated with this article can be found in the online version at doi:10.1016/j.cag.2009.12.004.

Acknowledgements

The work described in this paper is supported by the Scientific and Research Council of Turkey (TÜBİTAK) under Project Code EEE-AG 104E029. The Vienna2000 Model is courtesy of Peter Wonka and Michael Wimmer.

References

[1] Yılmaz T, Güdükbay U. Conservative occlusion culling for urban visualization using a slice-wise data structure. Graphical Models 2007;69(3–4):191–210.
[2] Funge J, Tu X, Terzopoulos D. Cognitive modeling: knowledge, reasoning and planning for intelligent characters. In: ACM Computer Graphics (Proceedings of SIGGRAPH '99), 1999. p. 29–38.
[3] Shendarkar A, Vasudevan K, Lee S, Son Y-J. Crowd simulation for emergency response using BDI agent based on virtual reality. In: Proceedings of the winter simulation conference, 2006. p. 545–53.
[4] Nguyen Q.-A.H, McKenzie FD, Petty MD. Crowd behavior cognitive model architecture design. In: Proceedings of the conference on behavior representation in modeling and simulation (BRIMS), 2005. p. 55–64.
[5] Shao W, Terzopoulos D. Autonomous pedestrians. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation, 2005. p. 19–28.
[6] Massive Software - Artificial Life Solutions. 〈http://www.massivesoftware.com〉, Accessed at June 2009.
[7] Legion Sofware. 〈http://www.legion.com〉, Accessed at June 2009.
[8] Simulex Software. 〈http://www.crowddynamics.com/egress/simulex.html〉, Accessed at June 2009.
[9] Luo L, Zhou S, Cai W, Yoke Hean Low M, Tian F, Wang Y, et al. Agent-based human behavior modeling for crowd simulation. Computer Animation and Virtual Worlds 2008;19(3–4):271–81.
[10] Pettré J, de Heras Ciechomski P, Maïm J, Yersin B, Laumond J-P, Thalmann D. Real-time navigating crowds: scalable simulation and rendering. Computer Animation and Virtual Worlds 2006;17(3–4):445–55.
[11] Pelechano N, Allbeck JM, Badler NI. Controlling individual agents in high-density crowd simulation. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation (SCA '07), 2007. p. 99–108.
[12] Helbing D, Farkas I, Vicsek T. Simulating dynamical features of escape panic. Nature 2000;407:487–90.
[13] Chenney S. Flow tiles. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation, 2004. p. 233–42.
[14] Hughes RL. A continuum theory for the flow of pedestrians. Transportation Research, Part B: Methodological July 2002;36(6):507–35.
[15] Hongwan L, Wai FK, Chor CH. A study of pedestrian flow using fluid dynamics. Technical Report, 2003.
[16] Treuille A, Cooper S, Popović Z. Continuum crowds. ACM Transactions on Graphics (Proceedings of SIGGRAPH '06) 2006;25(3):1160–8.
[17] Shopf J, Barczak J, Oat C, Tatarchuk N. March of the froblins: simulation and rendering massive crowds of intelligent and detailed creatures on GPU. In: ACM SIGGRAPH '08 classes; 2008. p. 52–101.
[18] Tecchia F, Loscos C, Chrysanthou Y. Visualizing crowds in real-time. Computer Graphics Forum 2003;21(4):753–65.
[19] Canter D. Fires and human behaviour, 2nd ed. David Fulton Publishers, Ltd; May 1990.
[20] Elliott D, Smith D. Football stadia disasters in the United Kingdom: learning from tragedy?. Organization Environment 1993;7(3):205–29.
[21] Miller DL. Introduction to collective behavior and collective action, 2nd ed. Waveland Press; February 2000.
[22] Yılmaz T, Güdükbay U. Extraction of 3D navigation space in virtual urban environments. In: Proceedings of 13th European signal processing conference (EUSIPCO '05), 2005.
[23] CAL3D Character Animation Library. 〈http://home.gna.org/cal3d/〉, Accessed at June 2009.
[24] Bittner J, Prikryl J, Slavik P. Exact regional visibility using line space partitioning. Computers & Graphics 2003;27(4):569–80.
[25] Nirenstein S, Blake EH, Gain JE. Exact from-region visibility culling. In: Proceedings of the 13th Eurographics workshop on rendering (EGRW '02), 2002. p. 191–202.
[26] Décoret X, Debunne G, Sillion F. Erosion based visibility preprocessing. In: Proceedings of the Eurographics workshop on rendering, 2003. p. 281–8.
[27] NVIDIA, NV_occlusion_query. 〈http://www.opengl.org/registry/specs/nv/occlusion_query.txt〉, Accessed at June 2009.
[28] Algorithmic Solutions Software GmbH, LEDA. 〈http://www.algorithmic-solutions.com/index.htm〉, Accessed at June 2009.
[29] Weber O, Devir YS, Bronstein AM, Bronstein MM, Kimmel R. Parallel algorithms for approximation of distance maps on parametric surfaces. ACM Transactions on Graphics, 27(4):Article no. 104, 2008.16 pp.