MODELING AND POPULATING VIRTUAL CITIES: AUTOMATIC PRODUCTION OF BUILDING MODELS AND EMERGENCY CROWD SIMULATION

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BİLKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Oğuzcan Oğuz

September, 2008

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Uğur Güdükbay (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Özgür Ulusoy

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Enis Çetin

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

MODELING AND POPULATING VIRTUAL CITIES: AUTOMATIC PRODUCTION OF BUILDING MODELS AND EMERGENCY CROWD SIMULATION

Oğuzcan Oğuz
M.S. in Computer Engineering
Supervisor: Assoc. Prof. Dr. Uğur Güdükbay
September, 2008

In this thesis, we present an automatic building generation method based on procedural modeling approach, and a crowd animation system that simulates a crowd of pedestrians inside a city. While modeling the buildings, to achieve complex and consistent geometries we use shape grammars. The derivation process incorporates randomness so the produced models have the desired variation. The end shapes of the building models could be defined in a certain extent by the derivation rules. The behavior of human crowds inside a city is affected by the simulation scenario. In this thesis, we specifically intend to simulate the virtual crowds in emergency situations caused by an incident, such as a fire, an explosion, or a terrorist attack. We prefer to 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. In order to speed up the animation and visualization of the resulting simulation, we employ an offline occlusion culling technique. During runtime, we animate and render a pedestrian model only if it is visible to the user. In the pre-processing stage, the navigable area of the scene is decomposed into a grid of cells and the from-region visibility of these cells is computed with the help of hardware occlusion queries.

Keywords: Procedural modeling, emergency, crowd simulation, crowd animation, occlusion culling, from-region visibility.

ÖZET

SANAL ŞEHİR MODELLEME VE NÜFUSLANDIRMA: OTOMATİK BİNA MODELİ ÜRETİMİ VE ACİL DURUMLAR İÇİN KALABALIK SİMULASYONU

Oğuzcan Oğuz
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Uğur Güdükbay
Eylül, 2008

Bu tezde, yordamsal modellemeye dayalı bir otomatik bina üretim sistemi ve sanal şehirler içinde kalabalık simülasyonu sağlayan bir simülasyon sistemi sunulacaktır. Binalar modellenirken, detaylı ve tutarlı geometrilerin üretilebilmesi için şekil gramerleri kullanılmıştır. Model üretme süreci belli noktalarda olasılıksal olarak işleyerek, gerekli çeşitlilikte bina modellerinin üretilmesine olanak sağlamaktadır. Bina modellerinin üretim süreci sonunda sahip olacağı geometri, türetme kuralları tarafından belirlenmektedir. İnsan kalabalıklarının şehir içi davranışları, simülasyon senaryosuna bağlıdır. Bu tezde, şehir içindeki insan kalabalıklarının, özellikle yangın, patlama ya da terörist saldırı sonucunda meydana gelebilecek acil durumlardaki davranışlarının simülasyonu amaçlanmaktadır. Kaçışan insan kalabalıklarının simülasyonunda kullanılan akışkan mekaniğine dayalı yaklaşım diğer yaklaşımlardan daha verimli olmaktadır. Simülasyon, sadece acil duruma sebebiyet veren olaya yakın bölgelerde yapılmaktadır. Elde edilen simülasyonların canlandırılmasını ve görüntülenmesini hızlandırmak amacıyla, kapatılan alanlar önceden hesaplanarak depolanmaktadır. Simülasyonun görüntülenmesi sırasında, kullanıcı tarafından gorülemeyecek insan modellerinin canladırılması ve görüntülenmesi engellenmektedir. Önişlem aşamasında, şehir modeli içinde dolaşılabilecek alanlar, birbirine eş hücrelere bölünüp, bu hücrelerin bölgeden görülme bilgileri grafik işlemci ünitesinin kapatılan bölge sorguları yardımıyla hesaplanmaktadır.

Anahtar sözcükler: Kural bazlı modelleme, acil durum, kalabalık simulasyonu, kalabalık animasyonu, kapatılan alanların ayıklanması, bölgeden görüş.

Acknowledgement

I wish to thank all those who helped me. Without them, I could not have completed this thesis.

I would like to acknowledge Assoc. Prof. Dr. Uğur Güdükbay who not only served as my supervisor but also encouraged and guided me throughout my academic program.

I would like to thank my jury members Prof. Dr. Özgür Ulusoy and Prof. Dr. Enis Çetin for their invaluable comments to improve this thesis.

I especially want to thank my family for their love, support and motivation to me. I want to thank my friends for their endless supports.

I would like to thank Ateş Akaydın for his help in the development of Crowd Simulator. Thanks to Dr. Türker Yılmaz for his help and valuable comments. Building model generation part of this research is a joint work with him.

This work is supported by the Scientific and Research Council of Turkey (TÜBİTAK) under Project Code 104E029. Vienna2000 Model is courtesy of Peter Wonka and Michael Wimmer. Pedestrian models are courtesy of CAL3D Character Animation Library.

Contents

List of Figures

Chapter 1

Introduction

Modeling and visualization of large and complex environments is a popular research area in computer graphics. Recent developments in processors and graphics cards, the amount of available memory, and the development of computer graphics modeling and rendering techniques facilitate to run high quality simulations. To this end, virtual cities should be modeled in order to be used in such simulations. One major component of a virtual city that affects the realism is the building models. In a virtual city, building models should have a high level of geometric detail and be consistent with the architectural style of the virtual environment. The other major component of a virtual city is the pedestrian and vehicle population. In a large urban environment like a city, there would be a large number of pedestrians that would really slow down an application unless the proper techniques are applied to simulate and animate the crowd.

In this thesis, we propose a continuum dynamics based emergency simulation system. Simulations take place in outdoor environments. We also propose a procedural modeling system that is used to generate building models to be placed in a city model.

1.1 Building Generation

Most countries have ground plans of the actual cities in the digital format. There has been more and more research in the last 10 years about generating 3D models using 2D ground plans and visualizing of these models. For instance, Google released a geographical visualization system, named Google Earth, which combines the satellite photos with the 3D models obtained using city plans to generate 3D city models [10]. Currently, generated city models consist of only a few cities, mostly in US. A similar application named CitySurf was developed in Turkey [32].

The visual modeling of large and complex systems has a long tradition in computer graphics. Improvement of machines and algorithms that are able to run high quality simulations, more virtual environments are needed. Today's machine capabilities facilitate the visualization of both large and complex 3D models. Applications cover a large spectrum, from military training and city planning to video games and tourism. Modeling exactly existent cities to be used in such applications can be tedious and is not intended in this work. Modeling each and every building in detail by hand is inefficient, even the use of aerial images or airborne laser scan data requires a great deal of manual work.

The original motivation for this work is to generate 3D city models using 2D city ground plans consistent with the real shapes of the buildings as much as possible and to visualize the city models in real time. City ground plans are used to produce city models by generating every building using its 2D ground plan. Generated models are then transferred to the visualization algorithm so that real time visualization is performed.

1.2 Crowd Simulation in Emergency Situations

Crowd animation is a crucial problem in computer graphics and animation since the crowds are a major component of a virtual scene such that without crowd animation or with an unsuccessful application of it, a virtual scene would not be

realistic at all whatever the qualities other components might have. Many applications of computer graphics such as computer games, virtual reality applications and animated films require high quality crowd animation. Moreover physically correct crowd simulations also have applications outside of computer graphics in 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 would sacrifice some of the characteristics of the crowd animation that affects realism whereas process of producing animated films have the required time and computational power to aim a more sophisticated simulation. For all of the applications, crowd animation need to be controllable. Virtual crowds should exhibit the intended behavior in computer games and animated films. In transportation research, the virtual crowd needs to be initialized by some statistical data to produce a realistic simulation.

A crowd animation should have the characteristics of real crowds. However, real crowds show very complex behaviors making the problem of crowd animation difficult. A surprising and difficult to model characteristic of real crowds is that in even very dense regions, people are able to move towards their destination smoothly and very few collisions occur. Additionally, people that move in the same direction tend to form lanes when they face other people moving in the opposite direction. The formation of lanes reduces the amortized area a single pedestrian seen by the people moving in opposite direction. Almost all the techniques to animate crowds were agent-based. In agent-based approaches, every single agent has its own computation of future behavior. Path planning and collision avoidance is performed for each agent in the scene. This approach is the most natural one since it is the way that real crowds work: each human makes his own motion decisions according to only the information 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 expertise 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 takes the goals, obstacles and other pedestrians into account when predicting the motion of a pedestrian.

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

In the outdoor environments of a city, an emergency situation causes the nearby crowd to show complex behaviors. Due to the incident, the people in the crowd would be in panic and would behave a lot 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 escape and hide in reaction to an incident. Some people may have tendency to show mass behavior. In contrast to the simulation of a normal crowd behavior, the simulations aiming emergency situations requires an approach that supports directly the simulation of homogeneous behavior. Continuum crowds approach perfectly fits this requirement.

1.3 Outline of the Thesis

The rest of the thesis is organized as follows: Chapter 2 reviews related work about reconstruction of city models, and crowd simulation. In Chapter 3, we

introduce our method to procedurally generate building models. Chapter 4 describes our crowd simulation system to simulate and visualize the behavior of a pedestrian crowd in emergency situations. We present the experimental results in Chapter 5. Chapter 6 gives conclusions.

Chapter 2

Background

2.1 Procedural Modeling of Buildings

One promising approach to the reconstruction of city models is the use of aerial imagery to extract the buildings and streets, using computer vision methods [16, 17, 34, 35]. Another promising approach is to use range scanning with the help of laser airborne scanning and other remote sensing methods [11, 18]. Both of these approaches aim to get the models of the real cities. There were obtained quite successful results, however, there are also some problems related with these methods. One of the problems is that these methods are not fully automated. They cannot identify all of the geometric structures in a city because of the high geometric variation of the buildings. Another problem is that city models with high level of geometric detail can only be constructed if, for every building in the city, specific data is acquired and processed. However, photographing or scanning every building in a city would be quite labor intensive.

Grammars, mainly L-systems, were applied to the modeling of streets [28]. The derivation of general detailed buildings using split grammars was demonstrated to be highly successful [38]. Split grammars are a composition of set grammars and shape grammars. Split grammars split or transform 3D shapes to sub shapes that are included in the volume of the parent shape. Derivation

ends when terminal shapes are derived which represents the building design. This derivation is steered by the attributes, so specific building designs and architecture trends could be achieved. During derivation, a parameter matching system is invoked that allows the user to specify multiple high-level design goals and controls randomness to guarantee a consistent output. An idea of control grammars was introduced that are simple context free grammars which handle the spatial distribution of design ideas not randomly, but in an orderly way that corresponds to architectural principles. CGA Shape Grammar is an improvement over split grammars [21]. CGA Shape presents context sensitive shape rules and can make use of complex mass models. Resulting buildings have underlying consistent mass models and high level of geometric detail. CGA Shape Grammar rules can be created from building images to generate a model of an existent building [22].

2.2 Crowd Simulation

Most of the work about crowd animation was agent-based in which each agent plans its motion individually. The agent-based approaches could get quite complex when one wants to consider cognitive aspects, such as knowledge, learning and emotional states [9]. The visibility and path planning was added to the Funge's work by Terzopoulos and Shao [33]. Later in 2006, production quality software, Massive Software, was proposed giving the animator full responsibility to define each agent's behavior [19]. However, Massive Software requires considerable effort to come up with sufficiently realistic large groups of people.

Hughes was the first one to view a crowd as a continuum and derive the set of equations to simulate large crowds [15]. 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 was later confirmed with real crowd data [14].

An inspiration from Hughes' model resulted in continuum crowds [36]. Continuum crowds approach makes the simulation of crowd flows possible by transforming Hughes' continuous crowd field into a particle representation. Treuille et al. made numerous improvements to Hughes' model to make the simulation able to exhibit a number of visually interesting and empirically proven behavior.

Chenney defines flow tiles for representing and designing velocity fields easily [6]. 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 can not assign goals to a single pedestrian or a group of pedestrians.

2.3 Emergency Simulation

Most of the research on pedestrian behavior in reaction to an emergency situation has been carried out by social psychologists [5, 8]. Panicking people have found to show maladaptive escaping behaviors different than normal socially-controlled behaviors [20]. In the case of bottlenecks such as doors and exits for a building, these panic behaviors cause jamming and overcrowding [8].

Most of the emergency simulation related computer models aim indoor evacuation scenarios [3, 27, 29, 30, 31]. Since most building evacuation situations include jamming and overcrowding, high congestion simulations are aimed to avoid the limitations proposed by grid resolution [12, 13, 30]. Helbing views and simulates the crowd as a self-driven many-particle system [13]. In this model, crowd dynamics of pedestrians are driven by a mixture of psychological and physical forces. Based on Helbing's model of social forces, Adriana et al. propose a model in which the virtual agents are endowed with different attributes and individualities [3]. Their model involves alarm and danger propagation which are important concepts for building evacuation simulations. 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 [30]. 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.

Chapter 3

Procedural Modeling of Buildings

In this chapter, we propose a shape grammar-based approach for procedurally-generating building models to construct city models [25, 26]. The work presented in this chapter is a joint work with Türker Yılmaz and also presented in his Ph.D. Thesis [39].

3.1 Building Model Generation

We use Data eXchange Format (DXF) of AutoCAD as the file format to store building geometry. DXF format is known to be common divisor of all 3D model formats since it is very simple, standardized, and is accepted by the community. The system outputs final building models in DXF file format. City plans are given in DXF format as input to the system and the system extracts individual building ground plans to derive individual buildings (see Figure 3.1).

The generated building models are composed of several facades, one for each edge of the ground plan. Each edge of the ground plan is handled at a time. A facade that corresponds to an edge of the ground plan is composed of a number of floors. A facade could be composed of multiple copies of the same type of floor or different type of floors.

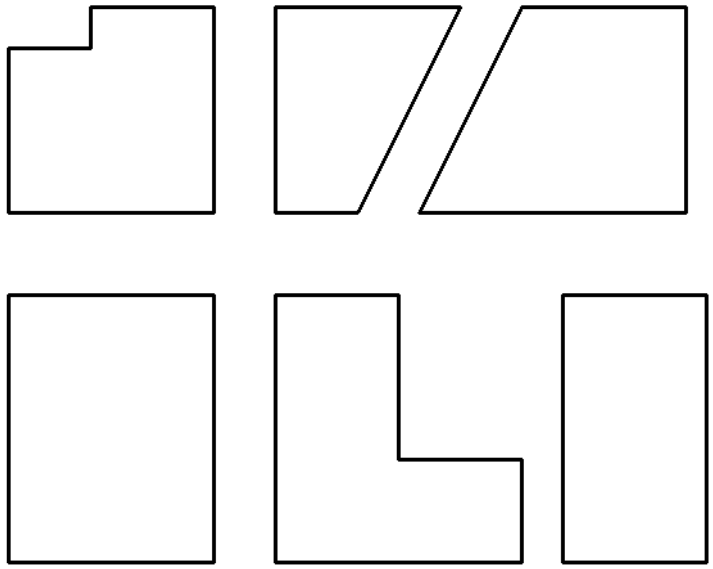


Figure 3.1: A simple portion of a city ground plan. Since the system generates each facade of a building by handling each edge of the building ground plan at a time, ground plans could be any kind of polygon.

A facade is split into floors. Floors are actually 2D floor face objects. Floor objects are then split by the pre-defined split rules in order to generate new 2D temporary objects (see Figure 3.2). This process goes on until all the paths end to a terminal object. All the terminal objects, such as windows, walls or balconies, are predefined in DXF format and they are transformed by translation, rotation and scaling. Finally, they are output to the model file in DXF format at the end of the facade generation process. When all edges of the ground plan are transformed to a corresponding facade, a building model is generated.

3.2 Shapes

There are two types of objects defined in the system:

Terminal Shapes: These are the shapes that are defined in a DXF file and stand for the basic shapes that are designed by a 3D design program best. Terminal shapes are composed of any number of planar surfaces, defined by

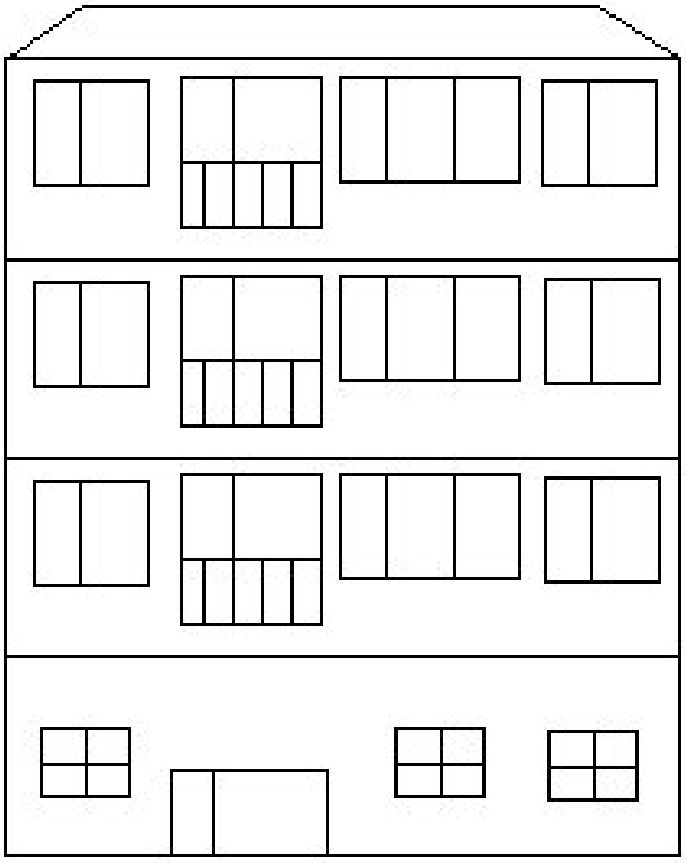


Figure 3.2: A building facade that is composed of the same type of floors. It should be noted that this may not always be the case.

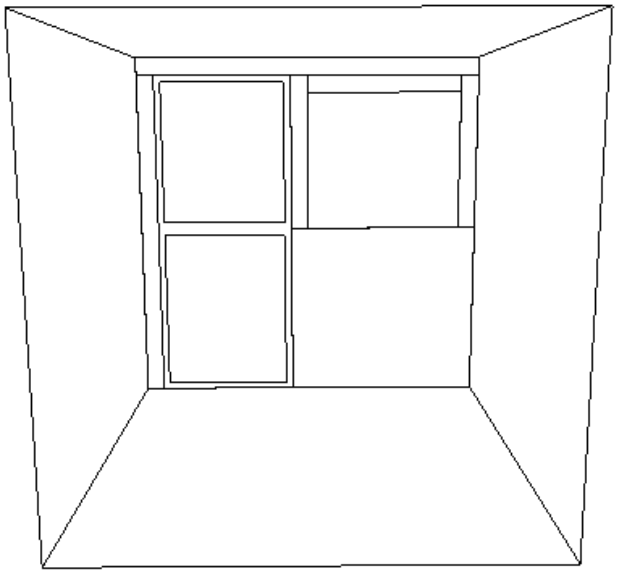


Figure 3.3: A very basic terminal shape that could stand for a balcony.

3DFACE entities in DXF format, and has unit dimensions to be able to be scaled (see Figure 3.3).

Temporary Shapes: These are the shapes that are split into other temporary shapes or terminal shapes by the rules that are defined in the configuration file in the XML format. The derivation process is initiated by a floor object. All the temporary shapes are attached an arbitrary number of attributes. These attributes are used to steer the splitting process by excluding some of the rules that do not apply to the temporary object. Temporary shapes are simple rectangular surfaces in 3D.

3.3 Rules

Until all the objects are terminal, all temporary objects are split by the rules. The rules guide the derivation process to construct a building model. All the temporary shapes have a set of rules that apply only to it. All the rules are attributed; i.e., an arbitrary number of attributes are attached to each rule. These

attributes play an important role in the split rule selection for a temporary object. Within the set of rules that apply to a temporary object, only the rules that have attributes with appropriate values could be applied to the object and one of them is selected randomly. Furthermore, all the rules could be given a probability if some rules to be selected more frequently. There are two types of split rules: *random split* and *fixed split.*

Random Split: Given a temporary shape, a random split rule splits the object into a 2D grid that is composed of randomly-placed rows and columns. The minimum width of a column and the minimum height of a row are given by the rule. All the rows are of $[minHeight, 2 * minHeight]$, and all the columns are of $[minWidth, 2 * minWidth]$ at the end. In a random split, all the newly-created objects are of the same type of shape, which could be a terminal shape or a temporary shape. In Figure 3.4, an instance of random split is shown. In Figure 3.5, an application of a random split rule is demonstrated.

Fixed Split: Fixed split rules split the given object into a certain number of rows and columns whose proportions of lengths are defined in the rule. When a temporary object is split by a fixed split rule, the sizes of the rows and columns that formed are directly proportional to the width and height of the temporary shape. When a fixed split rule is selected to be applied to a temporary shape, the number of newly created shapes is fixed, and the type and attributes of every newly created shape are defined in the rule. When the fixed split rules are applied to a temporary object, the newly-created shapes obtained could be terminal shapes or temporary shapes. An example fixed split rule is shown in Figure 3.6. In Figure 3.7, a series of applications of various fixed split rules is demonstrated.

```
<Floor>
   <Split Balcony="0 0" Window="+">
      <Random minWidth="2" minHeight="3">
         <BaseFace Balcony="0 0" Window="1 1"></BaseFace>
      </Random>
   </Split>
   . . .
</Floor>
```

Figure 3.4: A random split rule called `Split`. It is defined to split the temporary object called `Floor`. `Face` is the constructed object when the rule is applied. There could be other rules defined within the `Floor` block. These rules apply to object named `Floor`.

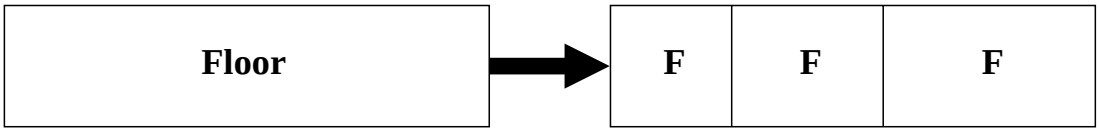


Figure 3.5: The application of a random split rule to a temporary object, called `Floor`, resulting in the creation of three terminal shapes, called `F`.

```
<BaseFace>
   <Window Window="+">
      <Fixed>
         <xProportions x1="1" x2="4" x3="1"></xProportions>
         <yProportions y1="3" y2="4" y3="2"></yProportions>
         <Elements>
            <Wall></Wall>
            <Wall></Wall>
            <Wall></Wall>
            <Wall></Wall>
            <Window></Window>
            <Wall></Wall>
            <Wall></Wall>
            <Wall></Wall>
            <Wall></Wall>
         </Elements>
      </Fixed>
   </Window>
   <BaseDoor>
      <Fixed>
         <xProportions x1="1" x2="4" x3="1"></xProportions>
         <yProportions y1="8" y2="2"></yProportions>
         <Elements>
            <Wall></Wall>
            <Wall></Wall>
            <BaseDoor></BaseDoor>
            <Wall></Wall>
            <Wall></Wall>
            <Wall></Wall>
         </Elements>
      </Fixed>
   </BaseDoor>
</BaseFace>
```

Figure 3.6: A simple fixed split rule, called `Window`, defined for the shape `Face`. The proportions of the size of the rows and columns to be formed are defined as attributes. The children of the <`Element`> tag are the shapes, which could be terminal shapes or temporary shapes, constructed when the split rule is applied.

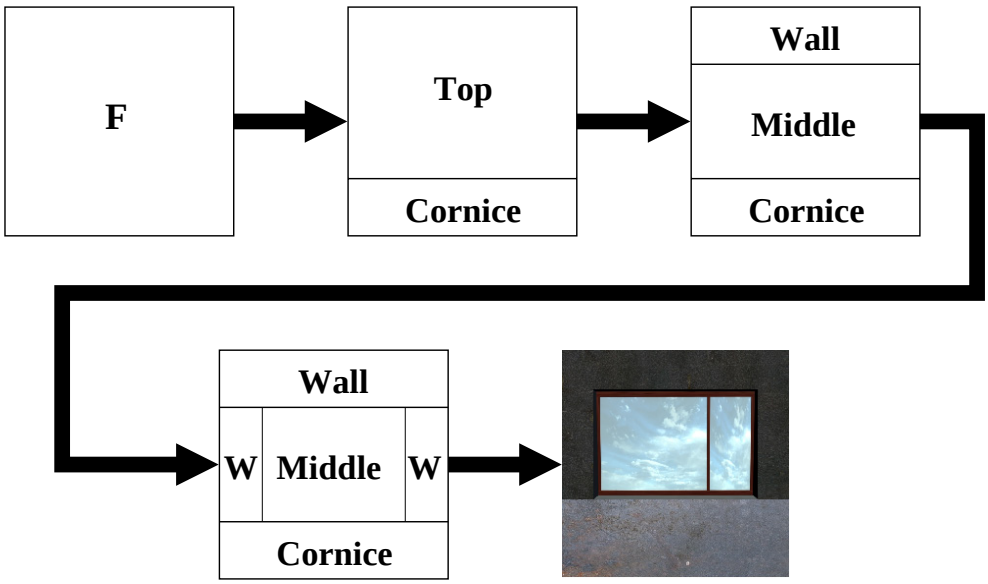


Figure 3.7: A series of applications of various fixed split rules. The series is initiated with a temporary shape called `F`. The derivation process ends when all the temporary shapes are transformed to terminal shapes.

Chapter 4

Emergency Simulation in Urban Areas

In this chapter, we give a description of our system for simulating and rendering human crowds in emergency situations. The intended animation is to capture the behavior of a crowd in an outdoor environment of a city. First, we mention some reactions of a crowd to an emergency situation to capture and point out the differences between outdoor and indoor emergency behaviors of crowds. Then, we describe our approach of simulating the crowd. Finally, the techniques that provide efficient animation and rendering of the simulated pedestrians are discussed.

4.1 Human Crowd Behavior during Emergency Situations

In this section, we explain how people react to an incident that occurs in a city. We are only interested in the movement behaviors of people and we exclude the gestures. At any instant, the movement of an agent could be defined by a direction vector and a speed scalar. We define the movement behavior of an agent

as changes in its movement under the effect of continuously changing conditions in the nearby environment. The affecting conditions include type and position of incidents, the nearby agents and their movement, and the structure of the city.

By emergency situations, we mean catastrophic events that may harm nearby pedestrians, such as a fire, an explosion, or a terrorist attack. The incidents need to be located in a well-defined area of the city and would only harm the people that are sufficiently close to an incident so that the reactions of the human crowd to escape and/or hide would be meaningful.

Most of the research on the simulation of emergency behavior of human crowds are focused on indoors [3, 27, 29, 30, 31]. They try to capture the movement of people in escape panic during a building evacuation. The resulting simulations help to foresee and examine any possible problems that may occur during emergency evacuations. The problems include any design deficiencies in the escape paths of buildings, possible decoration choices that may cause jamming, and the coordination issues of groups. Helbing summarizes characteristic features of escape panics as follows [13]:

1. People try to move considerably faster than normal, sometimes unnecessarily.
2. Individuals start pushing each other, and interactions among people become physical in nature.
3. Moving and, in particular, passing of a bottleneck becomes uncoordinated.
4. At exits, arching and clogging are observed.
5. Jams build up.
6. The physical interactions in the jammed crowd add up and cause dangerous pressures up to $4,450 Nm^{-1}$, which can bend steel barriers or push down brick walls.
7. Escape is further slowed by fallen or injured people acting as 'obstacles'.

8. People show a tendency towards mass behavior; i.e., they do what other people do.

9. Alternative exits are often overlooked or not efficiently used in escape situations.

The features listed above apply to most indoor escape panics. Also, a specific type of outdoor escape panics shows all the listed characteristics: the escape panics occurring in the places like indoors, which have well defined exits, gateways or bridges. These include overcrowdings occurred at stadiums and at Mina during the Stoning of the Devil ritual. However, since the outdoor space of a city would have much larger area per each agent compared to the indoors of a building, some of the listed features would not apply to our case. In a typical city environment, there would be streets and open spaces but no bottlenecks like doors or gateways. So, no matter how crowded the region is, there would not be any jams or clogging during an emergency situation in a city. Still, the items 1, 2, 7 and 8 are in common for the two cases. Additionally, in an open space, some people may tend to choose the escape paths farther away from the emergency region or point. During an emergency, the choices made are dependent on the agents' personalities and so would vary from an agent to another. So, some people may prefer to show mass behavior rather than running farther, since they feel safer that way and some people may try to hide in a nearby building. A common feature that is independent of an agent's personality, would be increasing tolerance to overcrowding with respect to normal situations. Additionally, people would not only walk on the sidewalks, they would be running on the roads.

During an emergency situation, in addition to escaping people, there would be a group of people whose duty is intervening the incident. These may be firefighters, policemen, etc. To be able to intervene the incident, these people try to get close enough to the emergency region or point. Some may try to get to the injured people or to the people that need help to get away.

These features of escape panics occurring outdoors of a city are merely observations. They are not based on any related psychological or sociological research or experiments. Surely, there are numerous distinct movement behaviors shown

by people due to an incident. To capture and simulate all those behaviors can only be achieved by modeling each and every agent in the scene distinctly. This approach would result in a very complex development process. Besides, it would be hard to make the whole system consistent and stable. For these reasons, we consider only the most generalizable distinct features of an escaping crowd such that we can divide the whole crowd into a small number of behavior groups. Given a small number of behavior groups we simulate each group within a constant time and get a amortized cost for each agent in that group.

4.2 Crowd Simulation in Emergency Situations

In crowd simulations based on the agent-based approach, each agent has its own computation of future actions. The agent-based approach is an approximation to the real life crowds and so it is a more natural way of simulating a crowd than particle motion 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 interacts with the other agents. However, in an emergency situation, people generally show more homogeneous behaviors: they all try to escape some way. Additionally, 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 [36]. 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.

Indoors emergency situations would involve jams and clogging caused by

bottlenecks like doors. People would push each other in panic. So, to effectively simulate indoor emergency situations, self driven many particle systems are used [13]. Particles are driven by a force model which assume a mixture of socio-psychological and physical forces that influence the behavior in the crowd. However, during an emergency situation at the outdoors of city, the crush of people is not crucial and would not affect the way the crowd move. So, our system does not involve any interaction forces. Additionally, we do not perform any collision detection since the continuum approach takes care of collision of agents as far as the resolution of the gird permits: a cell of the grid could include only one agent at any time. In our case, this limitation is not crucial since there would not be any jams requiring high congestion simulation. Helbing's model does not involve any global path planning, which is essential for an outdoors simulation. In contrast, the continuum approach involves global path planning to make agents reach their goals and local path planning to avoid congestion at the same time. Now, we briefly describe how the continuum crowds work.

4.2.1 Continuum Crowds

The continuum crowds approach is useful and efficient when large homogeneous groups of people are moving in order to reach specific goals. At a time step, motion planning is computed for each group people consisting of the agents that have the same destination. This characteristic meets our requirements since we can define the agents that are all trying to reach same specific goals in the same way as a group. The amortized cost per agent is substantially reduced if groups include lots of agents. We need a mathematical model, which is derived from the hypothesis about the virtual crowd, to simulate crowd dynamics [36].

The first hypothesis is that each person is trying to reach a geographical goal in the scene. When an agent tries to leave the scene, we assign a portion of the scene boundary to it as the geographical goal. The second hypothesis is that the people move at maximum possible speed f at its location for a certain direction θ. The velocity for a person at location x moving in direction θ is expressed as:

$$\dot{x} = f(x, \theta)\, n_\theta, \tag{4.1}$$

where n_θ is the unit vector in direction θ.

The next hypothesis is that agents try to avoid going through the locations that have higher discomfort value. A discomfort field g exists such that, all things being equal, people would prefer to be at the point where the discomfort is minimal.

At the end, people choose paths that minimize a weighted linear combination of the following terms:

- the length of the path,
- the amount of time to the destination, and
- the discomfort felt, per unit time, along the path.

So, an agent would choose the path P minimizing

$$\underbrace{\alpha \int_P 1 ds}_{Path\ Length} + \underbrace{\beta \int_P 1 dt}_{Time} + \underbrace{\gamma \int_P g dt}_{Discomfort} \tag{4.2}$$

Here α, β and γ are the weights for individual terms. ds and dt are related by $ds = f dt$, where f is the speed. Then, we may rewrite the equation as

$$\alpha \int_P 1 ds + \beta \int_P \frac{1}{f} ds + \gamma \int_P \frac{g}{f} ds, \tag{4.3}$$

which can be simplified to

$$\int_P C ds, \quad \text{where } C \equiv \frac{\alpha f + \beta + \theta g}{f} \tag{4.4}$$

is the *unit cost field.*

We describe how the agents find the optimum paths according to the above metric. A potential function is defined over the scene such that the potential function at a location is equal to the cost of the optimal path to the destination. For an agent, the strategy is to move in the negative direction of the gradient of the potential function. Intuitively, the potential function is defined to be zero at the goal locations and the norm of the gradient of the potential function is equal to the cost in the direction of the gradient everywhere else:

$$\|\nabla\phi(x)\| = C \tag{4.5}$$

So, the velocity of an agent at point x is defined as the velocity in the opposite direction of the gradient of the potential field scaled with the maximum permissible speed:

$$\dot{x} = -f(x,\theta)\frac{\nabla\phi(x)}{\|\nabla\phi(x)\|} \tag{4.6}$$

Once the potential function is computed, the locations of all of the agents related with that potential can be updated according to equations above. At every step, it is sufficient to compute a potential field for each group since all agents in the group tries to reach the same geographical goals and have the same environment variables such as discomfort field and weights for the terms in the unit cost definition. The requirement of equality of environment variables inside a group may seem a restriction but inside a portion of the city it is perfectly acceptable that some of the pedestrians have the same limitations.

Maximum permissible speed is a density-dependent term. At low densities, the speed is dominated by the terrain whereas at high densities, the speed is dominated by the movement of the nearby people, preventing an agent from trying to move in the opposite direction of the movement in a very dense region. At medium densities, the speed is computed by interpolation. This definition of permissible speed supports lane formation between the agents moving in the

```
1. Convert the crowd to a density field.
2. For each group:
   2.1 Construct the unit cost field C.
   2.2 Construct the potential field and its gradient.
   2.3 Update the people's locations.
   2.4 Enforce the minimum distance between people.
```

Figure 4.1: The simulation algorithm executed for each time step.

same direction.

To simulate the model, the model should be discretized. At each time step, the algorithm in Figure 4.1 is executed. The discretization is done by defining the scene area as a grid and store the scalar and vector values properly. The density value ρ for each grid cell is computed such that density for every agent is splat to the four grid cells surrounding the agent. The grid cell that the agent lays gets the highest contribution. To estimate the permissible speed in highly dense regions, an average velocity v is computed for each grid cell by summing and scaling the velocities of the agents that are inside or near to that cell.

Once the permissible speed and the unit cost is computed at every cell in each of the four directions for every group, potential field need to be computed again for every group. This step of the algorithm is the most time consuming part of the system. To overcome this burden, fast marching method is used. In order to prevent agents going through the buildings in the scene we do not take the building regions into account when computing potential field by defining them as boundaries. After computing the potential fields, gradient of potential function is computed and scaled with the permissible speed to get velocity values for agents. Then, all of the agents are updated for a timestep.

4.2.2 Navigable Space Extraction

In order to employ continuum approach on a city environment, we need a distinction between the navigable grid cells and non-navigable grid cells. Non-navigable cells can be the cells occupied by a building or a road. Agents cannot penetrate

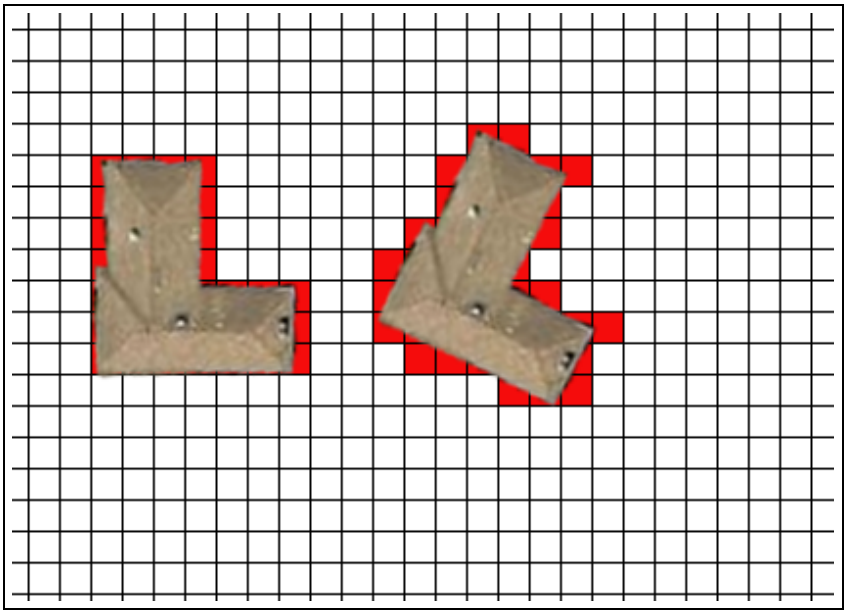


Figure 4.2: Non-navigable cells occupied by two differently aligned instances of the same building. The red cells are rendered non-navigable by the buildings.

to non-navigable cells so we do not need potential functions to be computed at non-navigable cells. By excluding non-navigable cell from potential functions computation, we prevent continuum flow from going through the boundary cells. Essentially, we prevent the agents from passing through the non-navigable cells which saves the need for any collision detection.

Navigable space needs to be extracted as accurate as possible. However, the maximum accuracy of extraction is determined by the resolution of the grid used since we must define a cell either as navigable or not. In order to get the ground level cells that intersects with a building, we test each cell against all the primitives of the buildings. Here, the only assumption about the scene is that the buildings need to be composed of triangle primitives. Tested cells are called seed boxes and seed testing is based on the approach in [40]. Redundant tests are avoided by testing a cell only against the buildings whose bounding boxes intersect the tested cell. After the seed testing is done, only the cells that intersects the primitives are extracted. However, the cells that reside in the buildings are also non-navigable. So, we set the cells that have no connection with the cells that are certainly navigable, as non-navigable.

Non-aligned building faces with the x and y axes of the navigation grid causes a higher degree of information loss than the aligned buildings would cause during navigable area extraction (see Figure 4.2). This kind of information loss is more crucial and apparent at narrow streets since a higher proportion of the navigable area is lost which could make some sufficiently narrow streets non-passable. The best we can do is to choose an alignment for the grid that the buildings would agree most. Increasing the resolution of the simulation grid would surely help but that would cause an increase in the computational cost of crowd simulation also.

4.2.3 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 any number of agents, wherever they are positioned on the grid, can be 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 of all the groups across the grid. Besides, in a city, 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 that are out of the interest area. To avoid high computational costs and redundancy to simulate crowds in a large city, 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 grid the which we call the active grid.

Due to aligned streets in a city, the grid 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 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

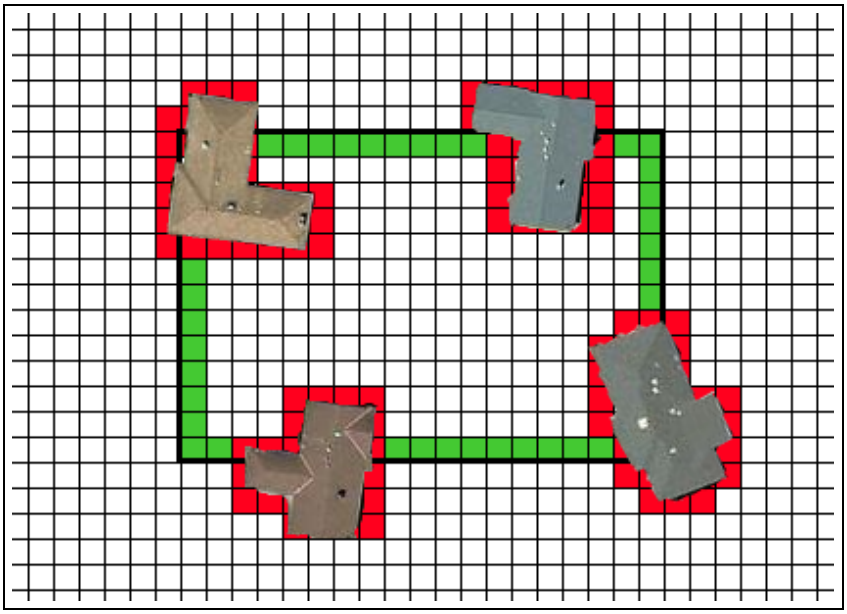


Figure 4.3: A configuration of the active grid. The green cells are the navigable border cells of the active grid. The agents are inserted at and directed to the navigable border cells of the active grid.

user cannot follow and view an agent if the agent leaves the active grid. However, since we aim to simulate and demonstrate the behavior of the crowd during emergency situations, 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. However, 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.

4.2.4 Normal Crowd Behavior Before the Incident

In order 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. It is necessary to add agents at navigable cells and direct them to sensible goals so that they contribute to the crowd flow. 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. Furthermore, the cells to which the agents are added and directed, need to be navigable cells. We define entrance region for the agent groups as some group of cells located in one or two side of the active grid. The goal region for a group is accordingly defined as the sides that are opposite to the entrance regions (see Figure 4.3). More complex flows can be defined as we define more groups of agents but that will also increase the computational cost.

4.2.5 Emergency Behavior

In our system, there are two major reactions to an incident. One is getting away from the incident and the other is getting close to it. As the incident occurs, the agents inside the active grid try to get away from it. After some time passed, police, firemen, or some interested people may try to get close 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 that try to get away would run to either ends of the street. To adapt all of the structure conditions, we place the goals for the escaping groups only at the navigable border cells of the active grid (see Figure 4.3). 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

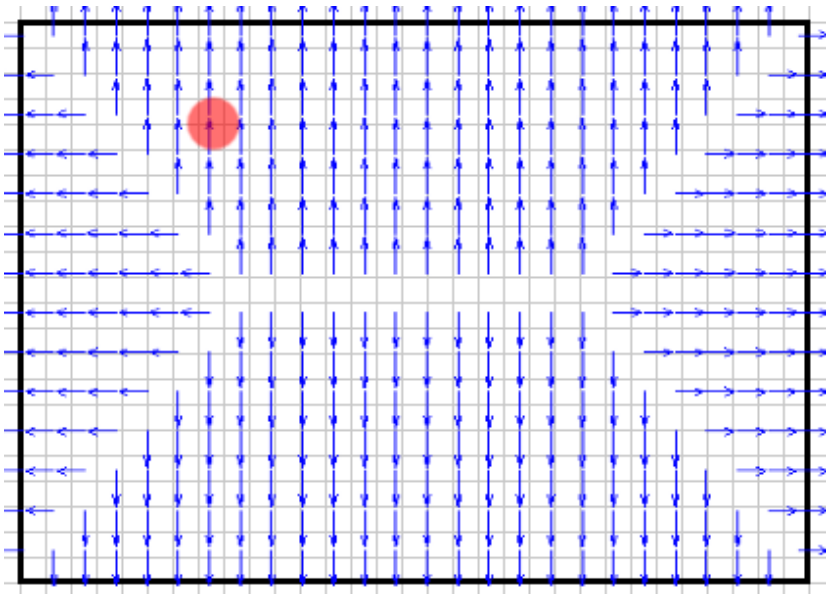


Figure 4.4: The vector field that the agents would follow if all the border cells are defined to be goals with equal priorities. Some agents would go through the incident region (filled circle) in order to take the shortest path to a border cell.

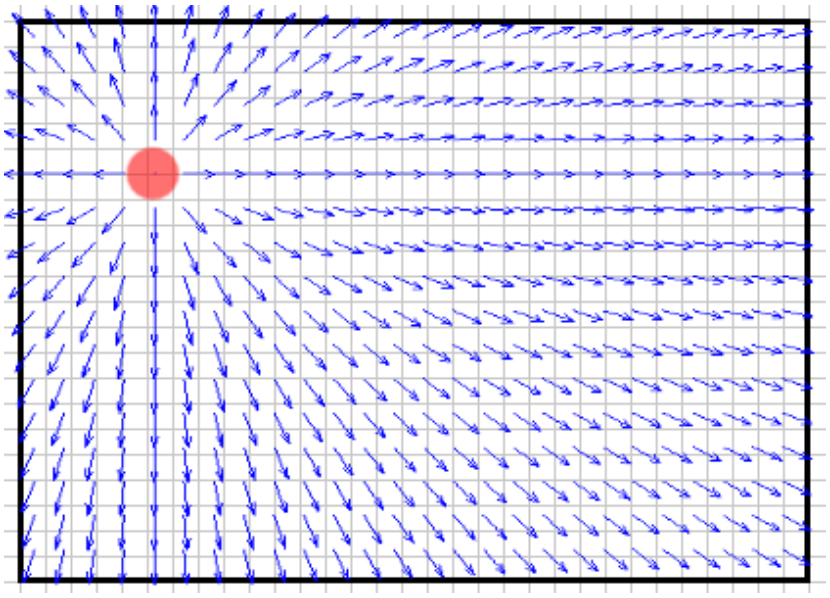


Figure 4.5: The ideal vector field that the agents should follow when an incident occurs at the filled circular region.

the incident region (see Figure 4.4). 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 favor the distant goals. In Figure 4.5, we want the agents to follow the vectors; moving outwards from the incident. What we do actually is defining an incident point as a local maximum inside the active grid.

The original continuum crowds approach sets the potential values of all the goals to zero initially for every time step, and then, to compute potential function across the grid, computes the level sets radiating from the goal cells using the fast marching method to compute the potential function across the grid [36, 37]. After the computation is done, an agent moves in the direction of the gradient of the potential function, which minimizes the cost for achieving one of the goals. So, if an agent has two goal cells at equal cost distance, the agent may choose to go to either goal cells depending on the iteration order. This is because the potential of the goal cells are also equal (they are both set to zero). In the light of this fact, we set the potential values of a goal cell in proportion to its Euclidean distance to the incident origin. If we assume a continuous domain and care only the Euclidean distance as a cost, this configuration will provide us with a vector field with vectors going outwards from the incident region.

At the center of the incident region, we want all the paths, every one of which is a shortest path to a goal cell, to minimize the potential value for the cell at incident center. The cost of a shortest path to a goal cell from the incident cell is computed by summing the unit cost through the path (cf. Equation 4.4). Let the potential value of the incident cell be zero. Then, if we consider only the distance as a cost, a goal cell that have d Euclidean distance to the incident cell would have its potential value set as $-\alpha d$, where α is the weight for the path length in Equation 4.2.

However, the unit cost also have the speed and discomfort components that depend on the environment state at the time of computation. Initially, they cannot be known for sure. In order to get a more close approximation for the potential values of the goal cells, we may assume an average value for the maximum

permissible speed through the path. Thus, the potential value of a goal cell that have d Euclidean distance to the incident is:

$$\phi = -d(\alpha + \frac{\beta}{\Delta f}), \tag{4.7}$$

where Δf is an average value for *maximum permissible speed* f. The average discomfort component is not included in the equation since we do not want the agents to favor the paths that introduce higher discomfort cost.

On the other hand, for the cases with more than one incident, we can not take the shortcut of defining a goal value in proportion to the distance to the incident. If the goal values are not set properly, an agent may run into an incident region while escaping from another incident. In order to set the potential values at the border cells, we run the fast marching algorithm from the emergency regions towards the border regions this time. We set a potential value for each emergency region and compute the potential values at the border cells using the constant unit cost value in Equation 4.8. Some incidents in the scene can be favored by setting their potential values lower than the potential values of the other incidents.

$$C_p = \alpha + \frac{\beta}{\Delta f} \tag{4.8}$$

Since the unit cost definition of continuum crowds approach should include varying components, such as time span and felt discomfort, agents would take paths different than the ideal outwards paths under some conditions, like in the case of high congestion. Even though the agents should avoid high congestion and to take alternative paths to the goals, they should not go through the incident region in any case. Additionally, we increase the unit costs of moving from one cell to another if the destination cell is closer to the incident. The closer an agent is to the incident, the more challenging would it become to move closer to the incident. Thus, the unit costs of getting closer to the incident are scaled in proportion to the distance of the start and destination cells to the incident. The unit costs of getting away from the incident stay the same. The scale factor for

the cost of moving from the cell A to the cell B is

$$S_{A\to B} = \begin{cases} c\frac{d_{B,I}}{d_{A,I}} & \text{if } d_{B,I} < d_{A,I} \\ 1 & \text{otherwise} \end{cases} \tag{4.9}$$

where $d_{A,I}$ and $d_{B,I}$ are the Euclidean distances of the cells A and B to the incident region, respectively, and c is the weight.

Given that the largest active grid has the dimensions $maxx$ and $maxy$, we precompute the possible unit cost coefficients and the goal values to be set in the case of an incident. Since, during the simulation, the incident can be positioned anywhere on the active grid we compute the unit cost coefficients and the goal values for a $(2 * maxx - 1) * (2 * maxy - 1)$ grid that has the incident as its center cell.

We define two behavioral groups that make the escaping crowd. The above unit cost scaling is done for both groups as they try to avoid getting closer to the incident. One group's goals are defined to be the navigable border cells, as explained above. The agents in it try to get away from the incident as quickly as possible. The other escaping group's goals are the building entrances. The agents try to hide in a building. The goals at the building entrances are also scaled by the distance of the goal cell to the incident cell.

The crowd trying to get closer to the incident is composed of two different groups: the people that got interested in the incident and the people that are on duty, like police and firemen. Unit costs for these groups are not scaled and goals are set to be the incident regions.

When the 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 (cf. Equation 4.4). Furthermore, agents are permitted to move into the roads. The simulator algorithm is given in Figure 4.6.

```
For each timestep:
  1. If the view point's distance to the active
     grid's 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 Set new goal values according to the new position
               of the incident with respect to the active grid.
  2. Add new agents to the crowd.
  3. Convert the crowd to a density field.
  4. For each group:
     4.1 Construct the unit cost field C.
     4.2 Construct the potential field and its gradient.
     4.3 Update the people's locations.
```

Figure 4.6: The emergency simulation algorithm.

4.3 Crowd Rendering

Once we simulated the crowd, we need to visualize the agents. In our system, we use a skeletal animation library [4] 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.

4.3.1 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. For this reason, 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

from-region visibility approach, 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 outdoors of a city 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. However, we aim to compute visibility of the agents which are dynamic objects and their position cannot be known for sure. Therefore, we define a uniform grid of target cells which are essentially 3D boxes with the height of an agent and placed at ground level. Occlusion information of each target cell for each 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 [2, 23]. The approach we take is based on the notion of occluder shrinking [7]. By shrinking the occluders present in the scene and sampling the visibility at discrete locations in a view cell, conservative occlusion culling can be achieved. If the occluders are shrunk by the maximum distance traveled in a view cell, sampling at the center of the view cell would retain conservativeness. Shrinking prevents any object that is occluded at the point of sampling, from becoming visible as the view point moves to any point in the same view cell.

Once the shrunk version of every occluder in the scene is computed, we need to check the visibility of every 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. However, hardware occlusion queries are dependent on the limited viewport resolution and prone to sampling and precision errors. A visibility information calculated by using hardware occlusion queries is certainly conservative

for the configuration in which hardware occlusion query is issued. However, 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.

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. Firstly, a region quadtree that partition the entire scene area is constructed. A node of the quadtree can be tested for visibility by testing a 3D box whose height is equal to an agent's height, which covers the area of the node. Then, we calculate the visibility information of the scene up to resolution of the quadtree. In this way, the queries for target cells that are completely inside of a non-visible node of the quadtree are avoided.

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 $\Theta\left(v^2 * t^2\right)$, 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 inside in the visible leaves of the quadtree. In this way, the required memory space is remarkably reduced. Storing a quadtree for the view cells would propose an overhead due to querying the quadtree to get visibility information for a view cell. However, since

the depth of the stored quadtree is fixed and is not very high, the query overhead would not be crucial.

4.3.2 View Frustum Culling and the Levels of Detail

In addition to occlusion culling, we perform view-frustum culling. If an agent is visible, then its bounding box is tested against the view frustum. The agent's pose is computed and the resulting mesh is rendered, only if its bounding box intersects the view frustum.

For every agent, three levels of detail of mesh geometry are pre-computed and stored [4]. During the simulation, switching between the three discrete LODs are performed based on the Euclidean distance of the agent to the view point.

Chapter 5

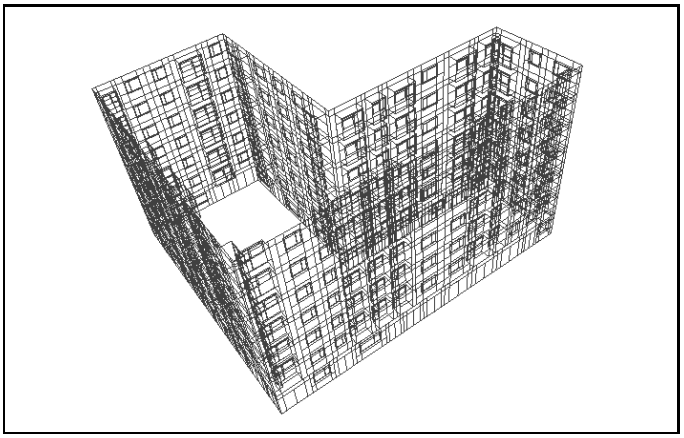
Results

The proposed algorithms were implemented using C++ Programming Language. The simulated crowd and the city environment are visualized using OpenGL libraries. Hardware-based occlusion culling is performed with the help of GL_NV_occlusion_query extension of OpenGL, provided by NVIDIA Corporation [24]. 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 [1]. All the preprocessing and simulation tests are run in a 2GHz Centrino Duo with an NVidia GeForce Go 7400 graphics card.

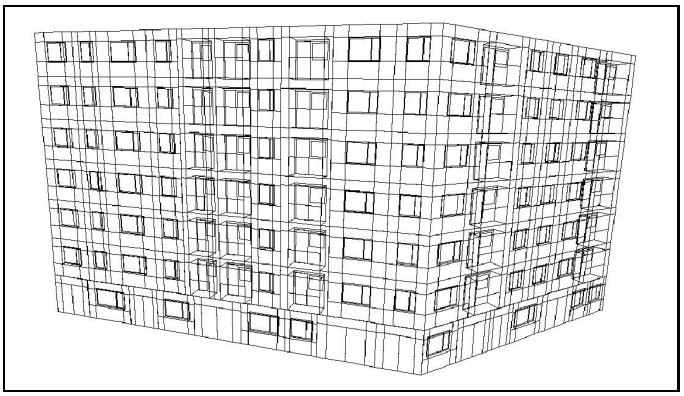
5.1 Building Generation

Generated building models could be seen in Figure 5.1. In Figure 5.2, a building model covered with textures is shown. Generated models could then be visualized by the visualization algorithm that performs three graphic acceleration techniques (occlusion culling, view frustum culling, hidden surface removal).

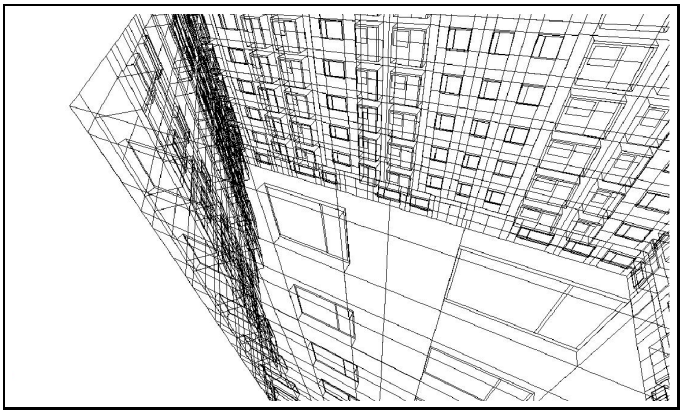
Building models that are produced by using only height information and floor plans neither have enough details nor reflect the architectural style of the actual



(a)



(b)



(c)

Figure 5.1: Generated building models.

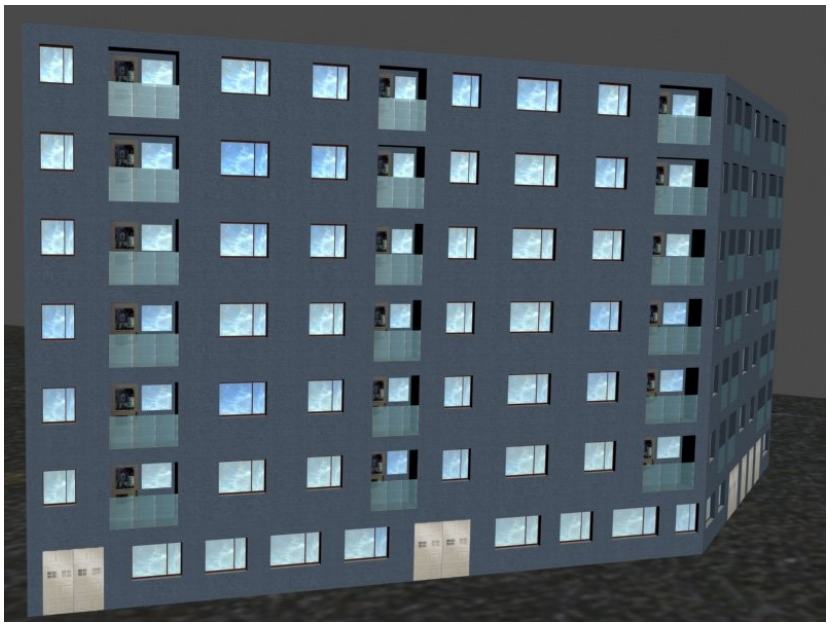


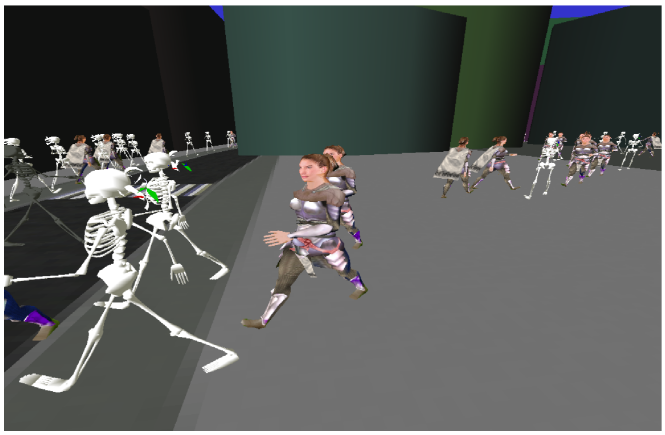
Figure 5.2: A building model covered with textures.

building. The proposed method allows fast generation of building models that are of intended architectural style.

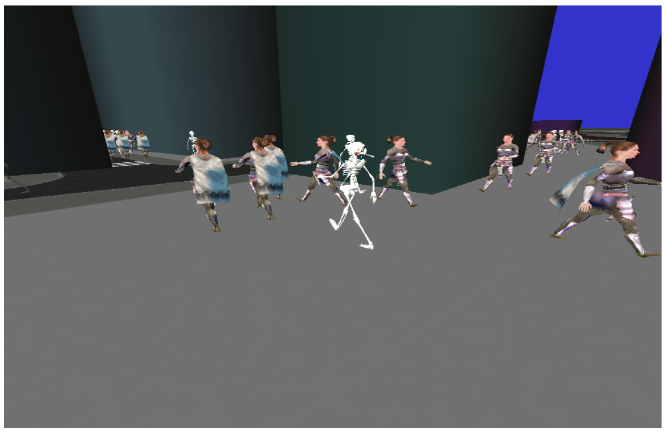
5.2 Emergency Simulation

The city model used is the Vienna2000 Model, which is model of the city of Vienna. The city model is composed of 805 buildings and a total of 23K triangles. The extracted navigable space in ground level is defined on a 1000x817 grid and shown in Figure 5.3. The extraction of the navigable space take less than 15 minutes. The navigable space grid is used as the simulation grid used to simulate the human crowd. If the cell size of the simulation grid need to be different than of the navigable space grid, we query the navigable space grid to see if a cell in the simulation grid is navigable. However, this process would result in some information lost. The best is to extract and have the navigable space information for several grids with differing resolutions, and to use the navigable space grid that have the same resolution with the simulation grid.

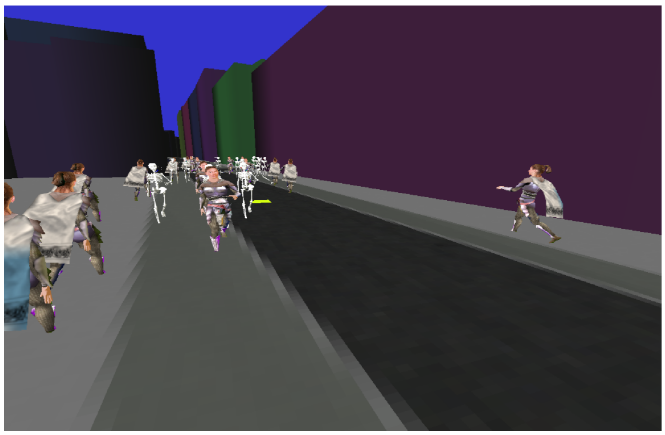
Figure 5.3: The extracted navigable space for Vienna2000 city model.



(a)

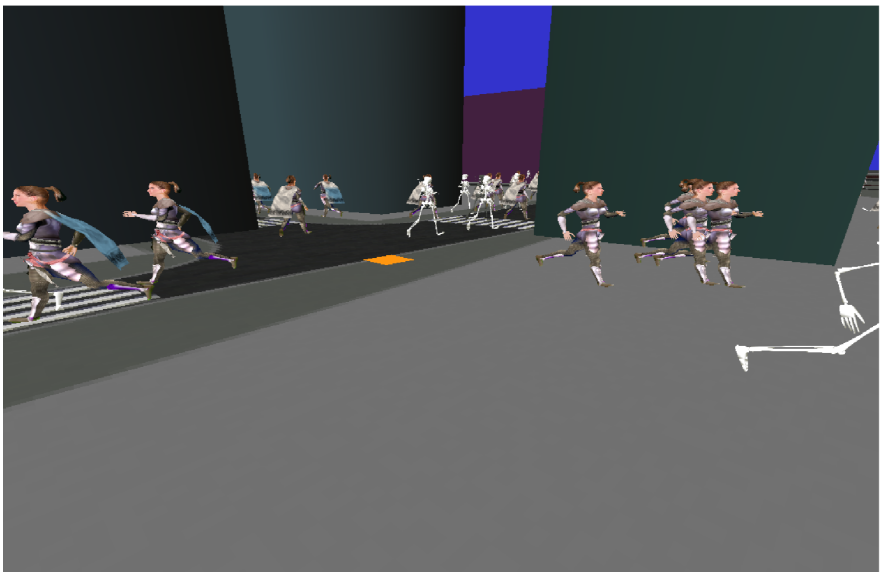


(b)

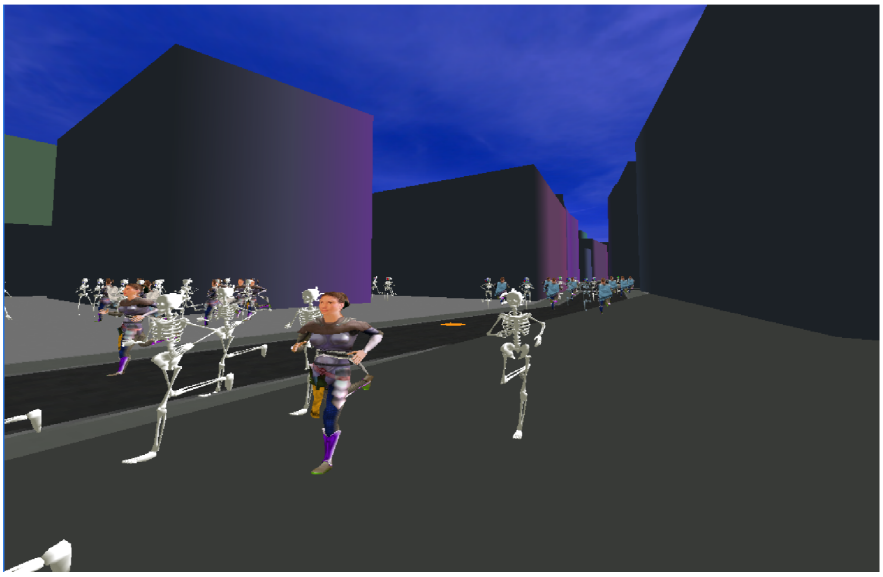


(c)

Figure 5.4: Still frames from a normal crowd behavior simulations.

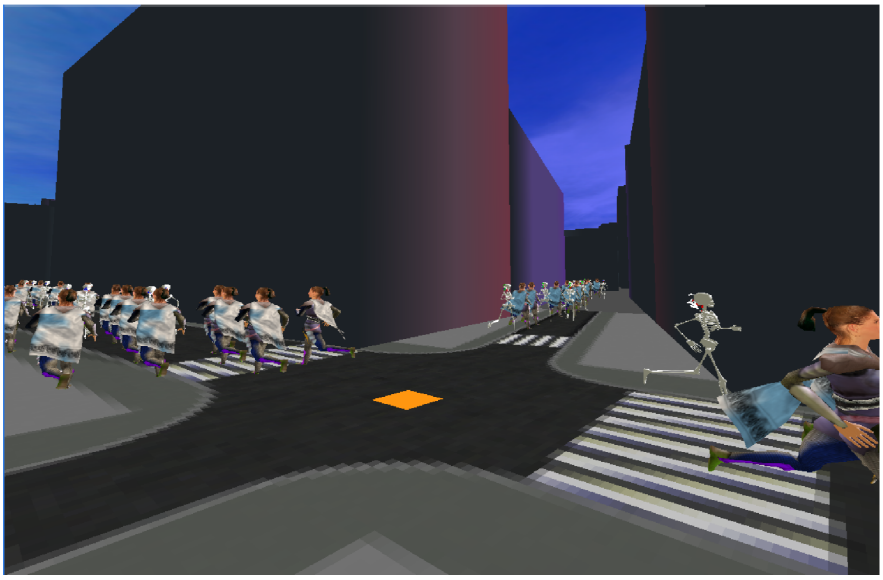


(a)

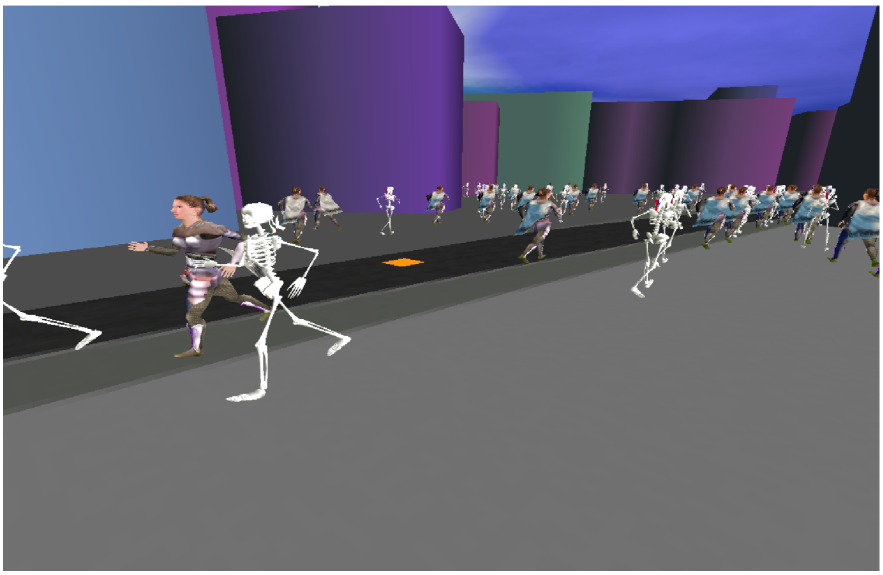


(b)

Figure 5.5: Still frames from an emergency crowd behavior simulations. The agents in the scene try to get away from the orange square.



(a)

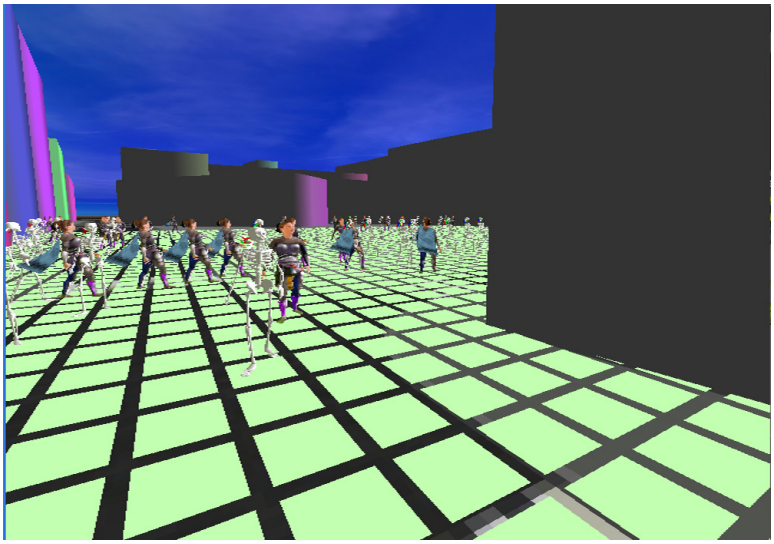


(b)

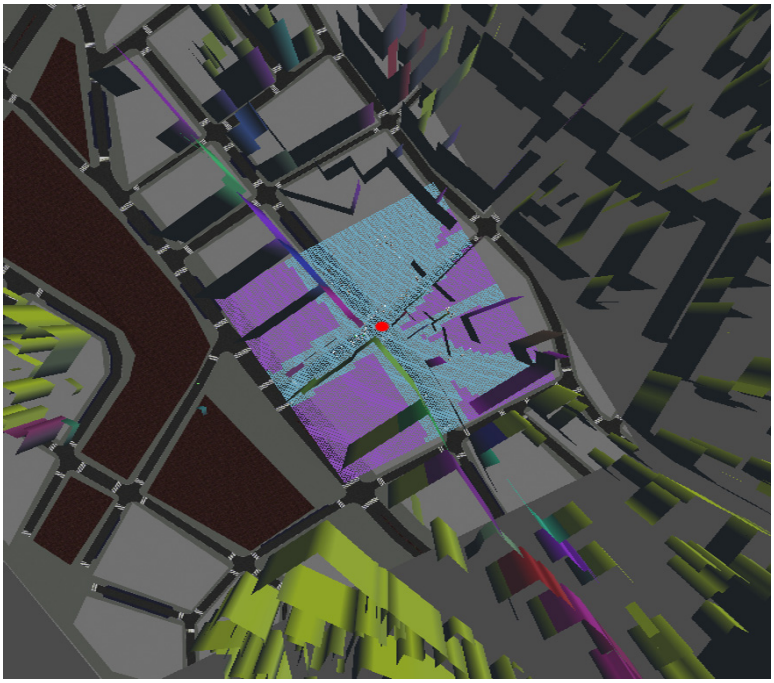
Figure 5.6: Still frames from an emergency crowd behavior simulations. The agents in the scene try to get away from the orange square.

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 preprocess 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. So, 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 preprocess time and larger memory space requirement. Also, making the target cells too small would waste the benefits of spatial coherence of visibility. In our tests, the view cells grid and the target cells grid are both set to have 170x130 resolution. Cell-to-cell visibility computation for this configuration take less than 20 minutes. The memory space required to store the visibility information is 48 MB. The use of calculated visibility information is shown in Figure 5.7.

We ran a series of simulations with varying active grid size, of total 5600 frames. Simulations are initiated without any emergency situations. Until an emergency situation is introduced, normal crowd behavior is simulated. When an emergency situation happens, the configurations of the agent groups are set to emergency mode, and all the agents behave in reaction to the emergency situation. Since we simulate four behavioral groups for emergency situations to capture the uniform emergency behaviors of all the agents, an agent can easily switch to another behavior by being transferred to another group. In Figures 5.4, 5.5, and 5.6, agents can be seen before and after the occurrence of the incident. The simulator runs at 60 frames per second on the average. 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 position of the agents between the two adjacent simulation steps. The crowd simulator



(a)



(b)

Figure 5.7: (a) A crowd scene captured at the ground level. Occluded agents are not drawn. (b) 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.

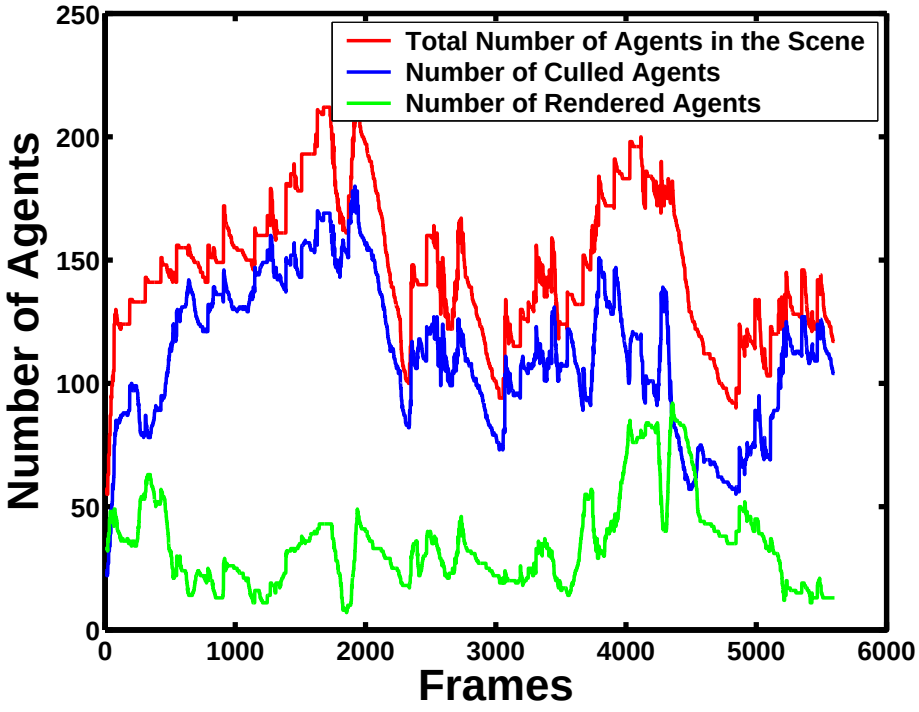


Figure 5.8: The total number of agents, the number of culled agents, and the number of rendered agents are plotted. The culled agents are either occluded by the buildings or out of the frustum.

is made to run at 5 frames per second, which makes the simulation step size 0.2 seconds. Please note 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. In Figure 5.8, the total number of simulated agents, the number of culled agents, and the number of agents that are rendered during a simulation are depicted in a graph. The culled agents are either occluded by the buildings or out of the frustum.

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

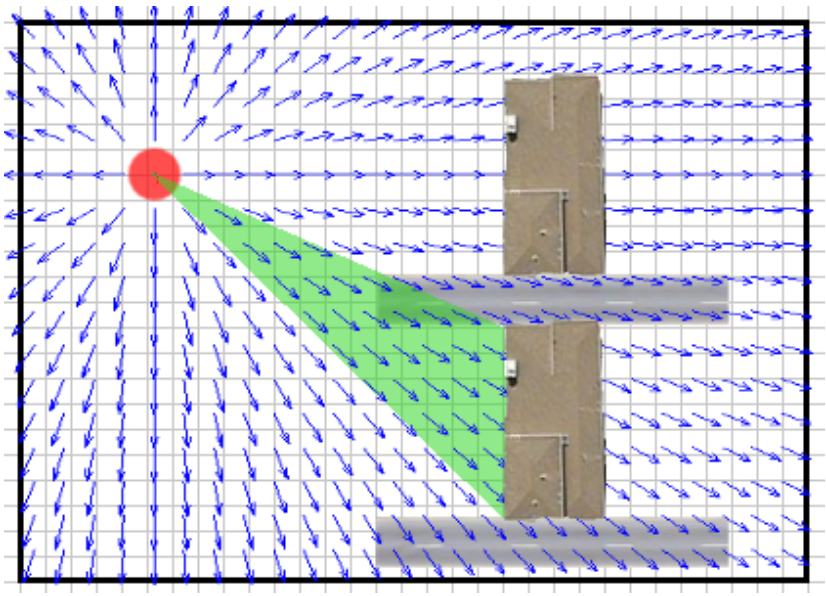


Figure 5.9: Depending on the structure of the environment and the nearby crowd distribution, escaping agents in the green colored area may take a path through the street that is closer to the incident than the other street.

region. In Figure 5.9, some of the agents that are in the green colored region may take a path through the street that is closer to the incident, rather than a path through the distant street. 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 unit cost scaling (cf. Equation 4.9).

Chapter 6

Conclusion

In this thesis, we propose a method capable of procedurally generating building models to be placed in virtual city models and a system to simulate pedestrian crowds in emergency situations.

The proposed building generation method makes it possible to generate building models with a high level of geometric detail. The desired number of building models to be placed in a city model can be generated within minutes. The building models are generated stochastically by means of the building footprints and shape grammars. The building generation takes the polygonal footprints of the buildings that are to be generated, and generates building models that sit on the given footprints. The building footprints can be found in the ground plans of the actual cities or, for a syntactic city area, footprints can be generated too. The other inputs of the system are the end shape designs and the building derivation rules. The system procedurally generates the buildings in the guidance of the derivation rules. The system initially creates a set of objects that represent the aimed building models, and then, the objects are separated into subparts according to the rules defined. Finally, the building model is generated using the end shapes. The end shapes stand for the basic building blocks of a model, such as windows and doors, that incorporates a high level of geometric detail and are hard to be procedurally generated. Since the model derivation process is steered by the derivation rules, consistent styles for the generated building models can

be achieved. The desired geometric variation in the generated models is achieved by incorporating randomness in the derivation process. The originality of a generated building model depends on the given end shapes and the rule set that will be used in the derivation process. With the help of the proposed method, a city of thousands of buildings can be generated without any manual processing. In the future, we plan to produce class libraries that are capable of modeling different styles of architectural constructions. Then, it will be possible to model cities in a more realistic way. At this point, it is necessary to emphasize that our aim is not to model the actual cities in a way remote sensing techniques do; our approach enables the buildings to appear in a more realistic way with higher level of geometric detail.

The proposed crowd animation system simulates the agents with a continuum dynamics-based approach applied to the crowd model of Hughes [15, 36]. In this approach, pedestrians are represented as a continuous density field. The system is driven by an evolving potential function defined so as to guide the density field optimally towards its goal. Instead of an agent-based approach, this continuum dynamics-based approach is more appropriate for the outdoors crowd simulations since the crowd flow is computed for the entire grid and the cost per agent is amortized. We represent the most generalizable behaviors of the pedestrian crowds in the emergency situations by the behavior groups. At each step, the potential function is computed for every 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-passable 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 potential field is computed only for the active grid. 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, we set the goal values at the navigable border cells of the active grid by running the fast marching algorithm with the initial potential values at the incident points. The unit cost for this potential function is mostly

based on the distance metric (cf. Equation 4.8). Thus, when the simulation is running with the proper goal values set at the border cells of the active grid, the crowd would move in the direction of the gradients of the contours that are radiating from the incident points. Still, depending on the configurations of the surrounding environment and the other pedestrians, an agent may choose to move through the incident regions. To prevent this type of behavior, additionally, we scale the unit costs used in the potential value calculation of the simulation. The unit costs of getting closer to an incident are scaled up to make it harder to move into a grid cell that is closer to an incident. 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 passing through the incident regions.

We also propose 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 that 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 information 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.

Bibliography

[1] Algorithmic Solutions Software GmbH, LEDA. Available at `http://www.algorithmic-solutions.com/index.htm`, Accessed at August 2008.

[2] J. Bittner, J. Prikryl, and P. Slavik. Exact regional visibility using line space partitioning. *Computers & Graphics*, 27(4):569–580, Aug. 2003.

[3] A. Braun, B. E. J. Bodmann, and S. R. Musse. Simulating virtual crowds in emergency situations. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 244–252, 2005.

[4] CAL3D Character Animation Library. Available at `http://home.gna.org/cal3d/`, Accessed at August 2008.

[5] D. Canter. *Fires and Human Behaviour*. David Fulton Publishers, Ltd, 2nd edition, May 1990.

[6] S. Chenney. Flow tiles. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 233–242, 2004.

[7] X. Décoret, G. Debunne, and F. Sillion. Erosion based visibility preprocessing. In *Proceedings of the Eurographics Workshop on Rendering*, pages 281–288, 2003.

[8] D. Elliott and D. Smith. Football stadia disasters in the United Kingdom: learning from tragedy? *Organization Environment*, 7(3):205–229, 1993.

[9] J. Funge, X. Tu, and D. Terzopoulos. Cognitive modeling: knowledge, reasoning and planning for intelligent characters. In *ACM Computer Graphics (Proceedings of SIGGRAPH '99)*, pages 29–38, 1999.

[10] Google Incorporated, Google Earth. Available at `http://earth.google.com/`, Accessed at August 2008.

[11] N. Haala and C. Brenner. Extraction of buildings and trees in urban environments. *ISPRS Journal of Photogrammetry & Remote Sensing*, 54(2-3):130–137, July 1999.

[12] D. Helbing, I. Farkas, P. Molnar, and T. Vicsek. Simulation of pedestrian crowds in normal and evacuation situations. In *Pedestrian and Evacuation Dynamics*, pages 21–58, 2002.

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

[14] L. Hongwan, F. Wai, and C. Chor. A study of pedestrian flow using fluid dynamics. Technical report, 2003.

[15] R. Hughes. A continuum theory for the flow of pedestrians. *Transportation Research, Part B: Methodological*, 36(6):507–535, July 2002.

[16] F. Jung, B. Jedynak, and D. Geman. Recognizing buildings in aerial images. In *Proceedings of the Workshop on Automatic Extraction of Man-Made Objects from Aerial and Space Images (Ascona'97)*, pages 173–182, 1997.

[17] Y. Liow and T. Pavlidis. Use of shadows for extracting buildings in aerial images. *Computer Vision, Graphics, and Image Processing (CVGIP)*, 49(2):242–277, Feb. 1990.

[18] H.-G. Maas and G. Vosselman. Two algorithms for extracting building models from raw laser altimetry data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 54(2/3):153–163, July 1999.

[19] Massive Software - Artificial Life Solutions. Available at `http://www.massivesoftware.com`, Accessed at August 2008.

[20] D. L. Miller. *Introduction to Collective Behavior and Collective Action.* Waveland Press, 2nd edition, Feb. 2000.

[21] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. V. Gool. Procedural modeling of buildings. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '06)*, 25(3):614–623, 2006.

[22] P. Müller, G. Zeng, P. Wonka, and L. Van Gool. Image-based procedural modeling of facades. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '07)*, 26(3), Article no. 85, 2007.

[23] S. Nirenstein, E. Blake, and J. Gain. Exact from-region visibility culling. In *Proceedings of the 13th Eurographics Workshop on Rendering (EGRW '02)*, pages 191–202, 2002.

[24] NVIDIA Corporation, NV_occlusion_query. Available at `http://www.opengl.org/registry/specs/nv/occlusion_query.txt`, Accessed at August 2008.

[25] O. Oğuz, M. E. Aran, T. Yılmaz, and U. Güdükbay. Automatic production and visualization of urban models from building allocation plans. In *Proceedings of the Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'06)*, 2006.

[26] O. Oğuz, M. E. Aran, T. Yılmaz, and U. Güdükbay. Bina tahsis planlarından 3-boyutlu şehir modellerinin üretilmesi ve görüntülenmesi (in Turkish). In *IEEE Sinyal İşleme ve Uygulamaları Kurultayı (SIU'06)*, 2006.

[27] M. Owen, E. R. Galea, and P. J. Lawrence. The Exodus Evacuation Model Applied to Building Evacuation Scenarios. *Journal of Fire Protection Engineering*, 8(2):65–84, 1996.

[28] Y. I. Parish and P. Müller. Procedural modeling of cities. In *ACM Computer Graphics (Proceedings of SIGGRAPH '01)*, pages 301–308, 2001.

[29] N. Pelechano and N. I. Badler. Modeling crowd and trained leader behavior during building evacuation. *IEEE Computer Graphics and Applications*, 26(6):80–86, 2006.

[30] N. Pelechano and A. Malkawi. Comparison of crowd simulation for building evacuation and an alternative approach. In *the 10th International Building Performance Simulation Association Conference and Exhibition*, Beijing, China, Sept. 2007.

[31] N. Pelechano and A. Malkawi. Evacuation simulation models: Challenges in modeling high rise building evacuation with cellular automata approaches. *Automation in Construction*, 17(4):377–385, May 2008.

[32] Piri Reis Data Processing Tech. Eng. Software Education Trade. Ltd., CitySurf. Available at `http://www.citysurf.com.tr/english/indexe.htm`, Accessed at August 2008.

[33] W. Shao and D. Terzopoulos. Autonomous pedestrians. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 19–28, 2005.

[34] L. Spreeuwers, K. Schutte, and Z. Houkes. A model driven approach to extract buildings from multi-view aerial imagery. In *Proceedings of the Workshop on Automatic Extraction of Man-Made Objects from Aerial and Space Images (Ascona'97)*, pages 109–118, 1997.

[35] V. Steinhage. On the integration of object modeling and image modeling in automated building extraction for aerial images. In *Proceedings of the Workshop on Automatic Extraction of Man-Made Objects from Aerial and Space Images (Ascona'97)*, pages 139–148, 1997.

[36] A. Treuille, S. Cooper, and Z. Popović. Continuum crowds. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '06)*, 25(3):1160–1168, 2006.

[37] J. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40(9):1528–1538, Sep 1995.

[38] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky. Instant architecture. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '03)*, 22(3):669–677, 2003.

[39] T. Yılmaz. *Visualization of Urban Environments*. PhD thesis, Department of Computer Engineering, Bilkent University, 2007.

[40] T. Yılmaz and U. Güdükbay. Extraction of 3D navigation space in virtual urban environments. In *Proceedings of 13th European Signal Processing Conference (EUSIPCO '05)*, 2005.