Volume xx (200y), Number z, pp. 1–13

Visualizing Crowds in Real-Time

Franco Tecchia[1], Celine Loscos[2] and Yiorgos Chrysanthou[3]

[1] Laboratorio PERCRO, Scuola Superiore S.Anna, Pisa, Italy
[2] University College London, Computer Science Department, London
[3] University of Cyprus, Computer Science Department, Nicosia, Cyprus

Abstract

Real-time crowd visualization has recently attracted quite an interest from the graphics community and, as interactive applications become even more complex, there is a natural demand of new and unexplored application scenarios. However, the interactive simulation of complex environments populated by large numbers of virtual characters is a composite problem which poses serious difficulties even on modern computer hardware. In this paper we look at methods to deal with various aspects of crowd visualization, ranging from collision detection and behavior modeling to fast rendering with shadows and quality shading. These methods make extensive use of current graphics hardware capabilities with the aim of providing scalability without compromising run-time speed. Results from a system employing these techniques, seem to suggest that simulations of reasonable complex environments populated with thousands of animated characters are possible in real-time.

1. Introduction

The wide use of computer graphics in games, entertainment, medical, architectural and cultural applications, has led it to becoming a prevalent area of research. Games and entertainment in general have become one of the driving forces of the real-time computer graphics industry, bringing reasonably realistic, complex and appealing virtual words to the mass-market. At the current stage of technology, a user can interactively navigate through complex, polygon-based scenes rendered with sophisticated lighting effects, and in interactive applications like virtual environments or games, animated characters (often called agents or virtual humans) able to interact with the users are becoming more and more common. As the size and complexity of the environments increase, there is a growing need to populate them with more than just a few of well-defined characters, and this has brought to the attention of the developers community the problem of rendering crowds in real-time. However, due to the computational power needed to visualize complex animated characters, the simulation of crowded scenes with thousands of virtual humans is only now beginning to be addressed sufficiently for real-time use.

One of the main obstacle to interactive rendering of crowds lies in the computation needed to give them a credible behavior. It must be noted that behavior can be simulated at two different levels, global and local, that can be combined. Global behavior is simulated when taking into account mainly global parameters of the environment and it is more suitable to describe group behaviors. On the other hand, local behaviors are simulated using properties of the agents and local parameters: agents are seen as independent entities acting from their own properties, the simulation is

Figure 1: *Real-time rendering of a village populated with 10.000 agents*

done locally for each unit, and designers have a high level of control. The basic idea of global behavior is that from the use of simple local rules the emergent behavior should result human-like. Developing such behavior is a hard task as it can be difficult to understand how complex behaviors emerge from simpler rules with the results often being quite unexpected, but on the other hand, using simple general rules is the only viable solution in case of real-time crowds rendering, as the number of agents to simulate is too high for individual scrip or perception-driven behavior.

Agents behavior is not the only hard task to perform in crowds simulation, the graphical activities involved can prove to be equally challenging. In fact, the rendering of highly populated urban environments in real-time requires the synthesis of two separate problems: the interactive visualization of large-scale static environments, and the visualization of animated crowds and traffic. Both tasks are computationally expensive and, using the current technology, only models composed by a few hundred of thousands of polygons in total can be displayed and visualized at interactive frame rates. The main problem in rendering crowds comes from the fact that the human body has an elaborate shape, and so a complex polygonal mesh is usually needed to represent it. Also, a human body has a very familiar shape to the eyes of a user which would be very sensitive to even the smaller artifacts that can be introduced by any simplification process. Situations where thousands of characters are on-screen at once can easily need well over a million polygons, making it impossible to render the scene in real-time.

In the rest of the paper we discuss possible solutions to the above problems, mostly taken from our own research on the field. Although more research is surely needed, preliminary results seem to suggest that simulations of reasonable complex environments populated with thousands of animated characters are possible in real-time. An example of the current results can be seen in Figure 1.

The main focus of our research has mostly been the real-time graphical rendering aspect, but we ended up building a complete platform for crowd visualization, and feel that some of our technical decision could be of interest for the community. In the following sections, we analyze in more details three of the main tasks needed for an interactive simulation of crowds, taking them in the same order as they are performed during a generic simulation. Each time-step of the simulation normally starts with an initial collision detection test, performed for each individual of the population (or performed on the subset of the population that is active at a given time); such test is then used as an input for the following phase which gives the agents a behavior. Once an action is assigned to each of them, the graphical rendering task is performed to visualize the final situation of the time-step.

2. Collision Detection Task

The collision detection test is used to make each agent aware of the surrounding environment; it is essential for tasks such as path planning and obstacle avoidance. There are many techniques to detect interference between geometric objects [17]. Many of them use hierarchical data structures, for example, hierarchical bounding boxes [7, 40], spheres trees [15], BSP trees [27] and Octrees [31]. However, the majority of these approaches try to solve the harder problem of exact interference between complex objects. For this reason, they tend to be much more precise than what is needed to simulate crowd flows. Due to the large amount of moving objects and the inherent time constraints of this particular application, we need to look at other solutions, and trade off small errors in exchange of greater speed and scalability.

To reduce the computational load, the fastest approach is probably to perform collision detection through discretization; the most relevant work to our idea is that of Myskowski [26] and Rossignac [30]. Like in our case, they use graphics hardware to perform the rasterization necessary to find the interferences in their models, but they then focus this task on a small number of very complex 3D CAD objects. Instead, in the case of crowds moving around in an environment, we can exploit some special situation: even though the geometry is still in 3D, the movement of humans is usually restricted to a 2D surface in space (often called 2.5D), or possibly more than one if we consider elements such as bridges. Bearing in mind this and the fact that the environment itself is static, fast collision detection can be performed.

Solutions dealing with the 2.5D case also exist. Steed [39] used a planar graph based on the Winged Edge Data structures for navigation in virtual environments. In Robotics, the problem was studied extensively for navigating mobile robots. Lengyel [16], for example, exploited raster hardware to generate the cells of the configuration space used to find an obstacle-free path. Bandi and Thalmann [5] also employed discretization of space using hardware to allow human navigation in virtual environments. However, in their case a coarse subdivision is used on the horizontal plane and repeated on several discrete heights, while in our system we consider the height of the obstacles in a more continuous way: we want not only to detect an obstacle, but also to detect its size; this is because the overall idea of the algorithm [43] is to represent crowd individuals as particles, and controlling their navigation trough a discreet representation of the virtual environment that we call the *height map*.

The height map represents simply the height of each cell of the subdivision, and it gets stored in main memory. In Figure 5, you can see an example of a height map and its associated 3D model. For every frame of the simulation, before moving a particle to its new position, we check its current elevation against the elevation stored in the height-map for the target position. If these values are too different, we assume that the step necessary to climb either up or down the

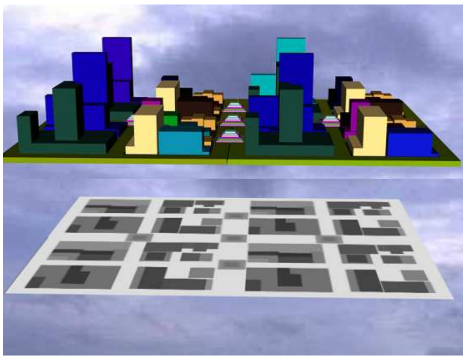


Figure 2: *Using a 2D grid to sample the environment. The top image is an example of a 3D model. The bottom image is the corresponding discretized heightmap used to perform collision.*

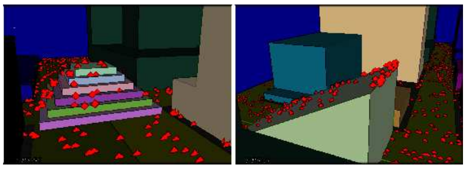


Figure 3: *Agents, represented here as red particles, correctly detect and interact with gradual slopes, avoid falling off the edge or going through objects taller than a threshold.*

cell is too big and cannot be taken, otherwise we move the particle and update its height according to the value stored in the height-map.

We also perform a second test, trying to influence the collision detection task with what lies ahead of the current particle position. Instead of simply checking whether our next step is possible from the current position, we also check whether the i^{th} step is possible from the predicted $(i-1)^{th}$ position. If this is not the case, then we will still allow the particle to move but start already changing the direction in anticipation of the collision. This results in a smoother animation. On the other hand, we now need two accesses to the height map, and this makes our test slower than in the previous case. The aim of these simple tests is to find a free path avoiding querying directly the geometrical database for valid directions, as this is essential in order to keep the cost of the collision test low.

In Figure 3 we can see an example of the emergent behaviour from these rules. Using the height map the particles correctly detect the different dimension of obstacles, climbing on them if the steps are small enough and updating their elevation without accessing the geometrical database of the model. This simple algorithm seems to be sufficient for basic collision detection tests, and it has several advantages upon polygonal approaches: using the graphics hardware to produce a rasterization of space is very fast and the data structure generated can be queried in minimal time. As a result, we can now perform collision tests for a population of thousands of individual in milliseconds.

3. Behavior

Researchers from different disciplines ranging from psychology to architecture and geography have been making observations of the micro scale behavior of pedestrians for over thirty years. For example, Goffman [12] discusses the techniques that pedestrians use to avoid bumping into each other. He discusses not only inter pedestrian avoidance, but also makes observations of differential flow, the role of attractors (shop windows), and how pedestrians negotiate junctions. Early work was also done at University College London, were researchers began to systematically develop techniques for observing and analyzing patterns of pedestrian flows, and correlating these to spatial properties of the environments navigated environment. Examples of these techniques are documented in [14, 13]. Observations had been made purely by hand, with the sole research aim of being able to better understand how people moved through space, both at macroscopic (e.g [14, 13]) and microscopic (e.g. [12]) level. A second important goal was to be able to predict real world movement; but ideas of using such observations as the basis of rule sets to simulate pedestrian movement or to populate virtual worlds with realistic humans were hampered by computer processing power. To address these difficulties, researchers have recently begun an attempt to devise simple rule sets to drive navigating agents.

Many techniques have been borrowed from (or adapted from) parallel research on real-world navigating robots, such as Prescott et al. [29] as researchers working on navigating robot problems have occasionally used software simulations to test their ideas and areas of crossover are present between the two fields.

The majority of work undertaken on simulating pedestrian movement has involved simulating densely populated crowd scenes such as in [24, 37]. Much of the work done tend to focus upon problem scenarios such as emergency situation evacuations and in [25] a definition of a crowd is proposed as being “a large group of individuals in the same physical environment sharing a common goal”. Although serving as useful precedents, this work is less useful for games programming, where the aim is frequently to populate environments with autonomous individuals that not necessarily share all the same goal.

Work done on natural movement includes early work by Penn et al. [28] in which rules were applied to agents, with distinct groups of agents using different heuristics for navigating from origins to destinations assigned randomly. The resulting paths taken were compared to spatial analyses of the environment and observed movement in the corresponding real environment (a district of London). Sophisticated variations on natural movement modeling include work done on the weighting and use of interest attractors by Smith et al. [36]; attractors in this environment include shop doorways, recreational areas, and street entertainers. Other refinements of standard natural movement models include Mottram et al. [23], in which agents behavior is modified through foveal and peripheral visual cues, and Thomas et al. [44] in which the micro scale behaviors required to navigate convincingly around road junctions and crossing roads are included in the agent's rule sets.

As the definition of the rule set for emergent behavior is a very complex task on its own, during our research on crowd simulation we felt necessary to develop a dedicated tool that could make the process of testing and debugging rules easier. With this intention, a platform that allows a user to develop and visualize the behavior of large numbers of agents was developed [42]. The use of space discretization employed earlier for the collision detection (Section 2) is carried here as well: a 2-dimensional grid containing various types of information is over imposed on the environment and agents navigate using the data contained in it. This 2D representation of the scenario is composed of four different layers. By combining the effect of each layer, an individual agent reacts depending on its position and the relative position of the other agents. The layers are ordered from the more basic (detection of possible collisions) to the more complex behaviors. Each cell of the grid corresponds to an entry to each layer. When an agent reaches a cell, it checks from the first to the fourth layer to decide what is going to be its next action. During each time-step of the simulation, an agent can check one or more cells for each layer. The original implementation uses the same cell size for each layer, but this is not strictly necessary. In the following we name and describe these four layers, in the same order an agent accesses them during a simulation.

Collision detection layer This layer is used to perform environment collision detection and defines the accessibility of areas. An image is used as an input to the platform, encoding in grayscale the elevation of the cell, or the information is created from a 3D model as described in Section 2. By examining this map, an agent can decide if it can pass by, climb up or descend in order to continue its journey. If the difference in elevation is above a given threshold, the agent must search for a new direction.

Intercollision detection layer This layer is used for agent-to-agent collision detection. Before moving to a new cell, an agent checks it to be sure that the target cell is not already occupied. The user can specify how much ahead to check.

Behavior layer This third layer corresponds to more complex behaviors encoded for each local region of the grid. A color map is used as an input file, so that with 8 bits per component in a RGBA space, up to 232 distinct behaviors can be encoded. The user then associates a color to the corresponding behavior. When an agent reaches a cell, it checks the encoded color to decide which behavior to adopt. It may be a simple behavior like 'waiting' or 'turning left' or more complex like 'compute a new direction depending on the surrounding environment. For example, we can use a visibility map (Figure 4b) to encode more probable paths, or an attractor map (Figure 4c), which may reflect how agents are attracted by some points of interest such as a bus stop or a shop window.

Callback layer Using this layer callbacks can be associated to some cells of the grid in order to simulate agent-environment behaviors. Such callbacks can allow the environment to react to the presence of agents; for instance callbacks can be used to call elevators or, in a simulation of city traffic, to make buses detect the presence of agents waiting at a bus stop.

In our experience, the combination of the described four layers permits the creation of complex crowd behaviors that can appear realistic and still suitable for interactive applications; as an example, the four layers are sufficient to control the actions of an agent walking along a pavement to reach a bus stop. Whilst walking the agent can avoid obstacles such as rubbish bins, telephone kiosks and other agents in front of him. On reaching the cell that corresponds to the bus stop (for which the associated behavior is to wait), the agent can pause and wait. When the bus arrives, a callback gets activated, causing the agent to climb into the bus. The flexibility of the callbacks mechanism is that even if they are triggered by the arrival of an agent, they can define local rules and actions of the environment on the agent (not necessarily the one that triggered the event). Since each rule is applied only locally, the callback, which is a more computational expensive procedure, is executed only when needed so that the whole series of behaviors can still be computed in real-time even if the environment contains many thousands of agents. Even the simple application scenario reported makes use of all the four layers described above.

4. Graphical Rendering

Rendering realistic virtual environments populated by thousands of individuals may need much more geometric power than what is available on current hardware. Techniques to efficiently handle large static polygonal models are a well-studied topic in computer graphics literature, but most of them are unable to handle complex dynamic entities such as crowds. Generally speaking, the acceleration techniques for the rendering of large environments can be subdivided in three main categories: visibility culling methods, level-of-detail (LOD) methods and image-based rendering (IBR) techniques. Although both culling and LOD can be very effective under the right circumstances, in cases where hundreds of detailed objects are visible simultaneously (see Figure 1) they can proven insufficient. This led us to choose IBR as the basic acceleration which lies at the core of our whole

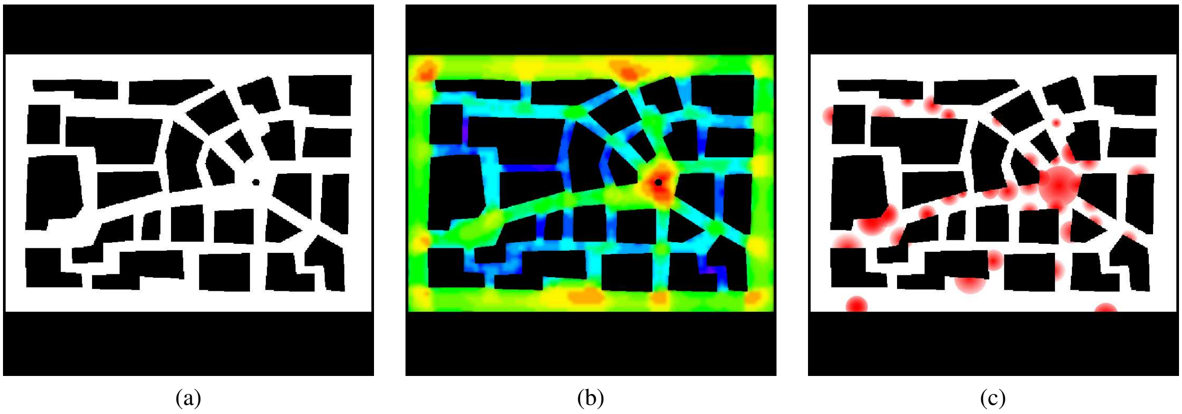


Figure 4: *(a) An example of a collision map. The regions where agents can move are encoded in white and inaccessible regions in black. (b) and (c) Examples of behavior maps. (b) Visibility map. (c) Attraction map.*

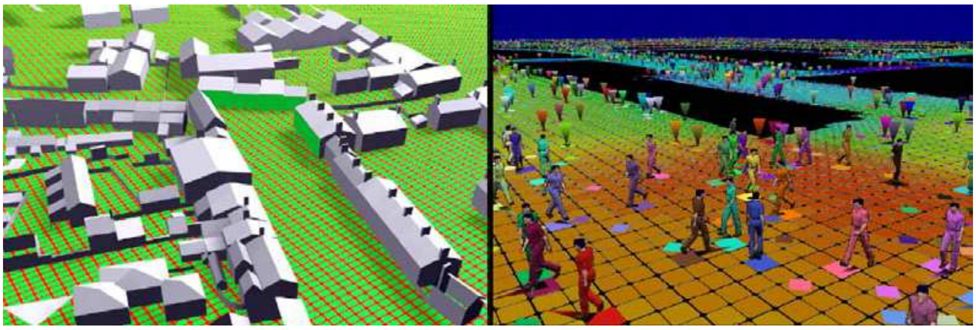


Figure 5: *The underlying grid used for the behavior (left) and a snapshot of the development system (right)*

rendering system. IBR allows us to reduce the amount of rendered geometry drastically, in addition we can build on it algorithms for efficiently providing other visual effect such as shadows and real-time shading.

The area of IBR has received a lot of attention recently resulting in a great body of research results [10]. The basic principle of Image-Based Rendering is to replace parts of the polygonal content of the scene with images. These images can be either computed dynamically [33, 35, 32] or a priori [19, 8, 2] and can be used as long as the objects are far enough from the viewpoint or as long as the introduced error remains below a given threshold.

These image substitutes are of course approximations which degrade as the viewpoint moves away from the reference position from which they were created. Image warping [21, 11] or the use of triangular meshes instead of single impostor planes [9, 20] can be used to reduce the artifacts but they come at increased rendering cost.

An IBR method which is close to our approach, also applied to the rendering of humans, is that of Aubel et al. [3, 4]. There, however, the impostors are computed dynamically and used only for a few frames before being discarded. Since the availability of large texture memory buffers is rapidly growing, in our work [1, 41] we decided to try to maximize rendering speed through the use of fully pre-computed impostors.

4.1. Precomputed Impostors

In a preparation phase, a set of textures is created representing a virtual character, with different textures corresponding to different frames of animation. Each texture is composed of a set of images of the character taken from different positions: a sampled hemisphere is used to capture the images, from 32 positions and 8 elevations around the character. At run time, depending on the view position with respect to each individual, the most appropriate image is chosen and displayed on an impostor. No interpolation is used between different views, as this is normally too CPU-intensive on current hardware. The appropriate texture to map is chosen depending on the viewpoint and the frame of animation.

Given the symmetric aspect of the human body performing a walking animation, we can reduce the number of samples and therefore the texture memory required for each frame. By mirroring the animation, we can cut in half the

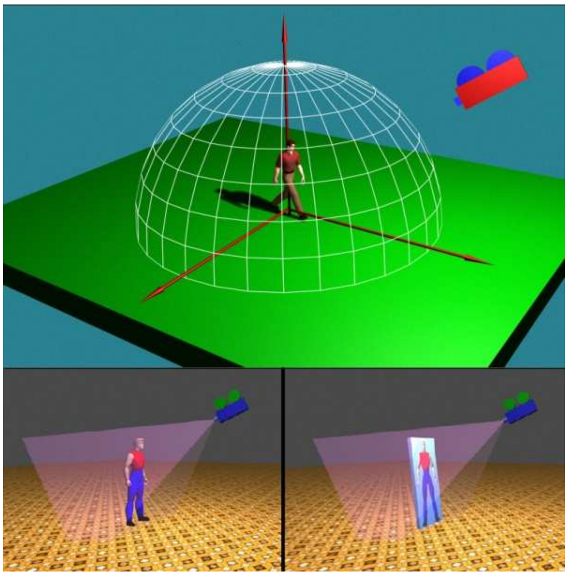


Figure 6: *Sampling the geometry and replacing it with images*

memory needed. For instance, we can reduce the 32 samples to 16 and get the others 16 by mirroring. The images for each human, per frame of animation, are collected together and stored in one big texture. Since each individual sample contains also a lot of wasted space (background), when we put them in the texture we can pack them closer together. This results in savings of up to 75% with the only disadvantage that the handling of the impostors is now a bit more complex since the images are not arranged in a regular grid in the texture [41]. In addition, texture compression can be used to store the image database and the use of OpenGL compressed format S3TC_DXT3 [34] gives a further memory compression ratio of 1:4. The particular compression format reserves 4 bits to the alpha channel values, that is extremely important for our multipass rendering algorithm, described in Section 4.3.

4.2. Choosing the Best Impostor Plane

Using impostors for representing complex objects such as virtual humans may lead to two common forms of artifacts. First, there might be missing data due to inter-occlusion and black regions may appear. Second popping effects may occur when the image samples are warped and/or blended to obtain the final image.

In our system, any artifacts are mainly due to the popping caused when switching the samples as the viewpoint changes. An intuitive approach to reduce the popping effect is, of course, to increase the number of samples. To keep the memory consumption down, we chose instead to improve the choice of the impostor to reduce the visual error between two different views. The amount of error for a generic point on the object surface is proportional to the distance of the point from the projection plane. Instead of computing the impostor plane as the one perpendicular to the view direction from which the sample image was taken, we decided to try a different approach. We choose the impostor plane as the one passing through an object that minimizes the sum of the distances of the sampled points and the projection plane given a camera position from where the sample image is created. In the case of samples of human polygonal models, using this plane leads to a significantly better approximation of the position of the visible pixels in respect to the actual point positions in 3D [41].

4.3. Improving Variety With Multipass Rendering

Our approach of using precomputed impostors can be demanding in terms of texture-memory. Even with the all compression techniques mentioned in Section 4.1, if we want to provide a high variety of humans forming the crowd, it is impossible to provide one individual representation of impostor per virtual human of the crowd without exploding the memory requirements and cutting down the rendering time. Instead, we have chosen to use a reduced number of virtual humans and at rendering time, the impostors are modified on the fly in the attempt to give to different agents a different aspect. As it would be more difficult to procedurally change the shape and the general silhouette of each human, we focused on re-coloring significant parts of their body, like cloth, hairs, and skin color. As we need at run time to efficiently identify these areas on the images, we pre-select the different regions and store them in an alpha channel image with a different alpha value for each part to modify (see Figure 7). If no texture compression is used we can store up to 256 different regions in the texture, or up to 16 if texture compression is used since only 4 bits of precisions are available in the latter case. At runtime the alpha channel is then used together with multi-pass rendering: for each pass, the alpha test value is modified allowing the rendering of one region at a time, while a different color is assigned to the impostor polygon per virtual human and the texture is applied using the flag GL_MODULATE. In our tests, we pre-selected 3 different regions but more regions could be selected. However the number of selected regions corresponds to the number of passes needed when rendering and the heavy use of multi-pass rending might slow down the overall rendering rate. One should decide on a tradeoff between the variety and the rendering time.

4.4. Real-time Shading of the Impostors

There are strong motivations in the attempt to introduce interactive lighting in the technique of animated impostors: apart from flexibility and aesthetic considerations, relying on

Figure 7: *Modulating colors using the alpha channel*

the simple, pre-computed lighting often associated to the impostors forces severe restrictions to the simulation, in particular when switching from the polygonal mesh to the image-based representations. Such operation can introduce disturbing popping artefacts in the rendered image, that we can classify in two categories: the first has a geometric nature, and is due to the misalignment in the final image of the pixels location computed with the proper geometric transformations and the location of the pixels generated by the use of the single-layer impostor, where all the geometry is projected on the impostor plane, see Section 4.2. The second form of artefacts has a lighting nature, and it is due to the clashing illumination condition between the polygonal mesh and the impostor image. The latter is generally pre-encoded in the sampled images, and as such it can't normally be efficiently changed; some work on the topic has been proposed recently [22]. On the other hand, this rigidity penalizes also the polygonal representation, that could reflect any lighting condition using the standard lighting model of OpenGL, including dynamically changing light conditions (local/moving light sources, colored lights and so on).

Given the current image sample rate (i.e. the number of samples taken around each object) and the current memory limitations, it can be said that the popping artefacts due to the lighting differences are by far the more distressing ones. The difference between these two lighting information imposes to use in the environment a fixed number of directional and static light sources and thus make it impossible the simulation of any reasonably flexible specular effect; while this may not be too important when rendering a crowd, it can be quite limiting when impostors are used for objects with a prominent specular nature, as cars or in general any object with glossy surfaces.

In this section we show how it is possible, using standard OpenGL1.3 per-pixel dot product, to achieve on animated impostors a dynamic lighting equivalent to the one available for polygonal models. OpenGL 1.3 per-pixel dot product is available trough the token DOT3_RGB_ARB of the texture environment parameters [34].

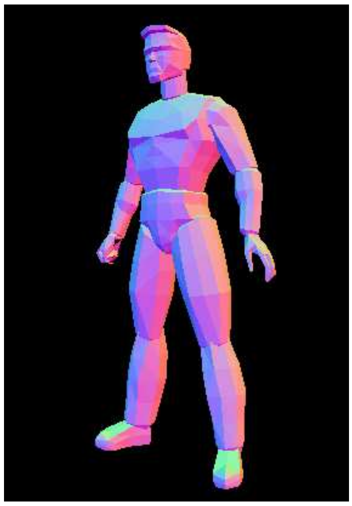


Figure 8: *Storing the normals information in the RGB space.*

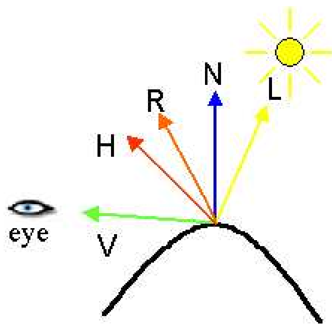


Figure 9: *Local illumination parameters. V is the viewing direction, L is the light direction, N is the normal of the surface, R is the mirror direction of L relatively to N, H is the bisector of the L and V.*

The first step of our approach requires to change the type of information stored in the impostors' image database: instead of storing a gray-scale image holding a fixed lighting information as suggested in [41], we need to store the normal associated at each pixels (see Figure 8 for a reference). According to the OpenGL 1.3 specification, the spatial components x, y, z, of the normal of each pixel are encoded in the texture RGB space using the following convention: $x \rightarrow r, y \rightarrow g, z \rightarrow b$. Once the color channels are filled with the normals' information, we store in the alpha channel the same information as suggested in [41]; we use such data to have a finer control on the impostor colors. Let's now consider the local reflection model used by OpenGL: leaving aside the issue of color, we can write down the intensity equation in the usual form:

$$I = A + K_d \overrightarrow{L}.\overrightarrow{N} + K_s(\overrightarrow{R}.\overrightarrow{V})^n \quad (1)$$

where n is used to simulate the degree of imperfection of a surface; when $n = \infty$ the surface is a perfect mirror (see Figure 9 for details in notations) . For other values of n an imperfect specular reflector is simulated. The mirror direction $\overrightarrow{R}$ being expensive to calculate, the equation is normally considered in the following form:

$$I = A + K_d \overrightarrow{L}.\overrightarrow{N} + K_s(\overrightarrow{H}.\overrightarrow{N})^n \quad (2)$$

where H is simply the halfway direction between the light direction L and the viewing direction V. We can now use the DOT3_RGB_ARB texture parameter to perform the equation's dot products on a per-pixel basis, accumulating on the frame buffer the partial results of the intensity equation. Using multipass rendering we can sum all the components, and compute the final value of each pixel intensity. To accomplish this, and in accordance with the OpenGL specifications, the RGB codification of vectors L and H are used as the fragment color of the polygon. To simplify the otherwise overwhelming computation of L and H for each pixel, we consider them constant over each impostor. In this way it is necessary to compute L and H on a per-impostor basis only, depending on the current impostor position and orientation with respect to the considered light source.

To accumulate in the frame buffer all the lighting component of equation (2), we use at present 5 passes per impostor. The first n passes are used for the specular component (in our tests we used an average of 3 passes; greater values are possible, but at the cost of slowing the rendering process); the next pass is used to render the effects of the ambient component, and a last one to add the effect of the directional component. Playing with the modulus of the polygon color, it is possible to introduce in the equation the factors k_d and k_s, and effects of local light sources complete with attenuation can be simulated. At this point, the frame buffer contains the grayscale image of the impostors representing the correct illumination with respect to the actual light position and surface propriety (see Figure 10). The process could be repeated to accumulate the effects of multiple light sources; in this case the limited numerical precision of the frame buffer should be considered, as the standard 8 bits per color channel could present some numeric precision issues. Using a uniformly colored texture in the second texture unit, we can modulate the color of the resulting intensity computation, making it possible to simulate even colored lights. Once the illumination is in the frame buffer, we use several additional passes (in our case 3) to modulate different regions with different colors, using the alpha test technique as described in [41]. Figure 10 shows the described process, starting from the light intensity calculation to the final color modulation using the alpha-test.

4.5. Adding Shadows

Shadows not only add greatly to the realism of the rendered images but they can also provide additional visual cues and help "anchor" objects to the ground. They can be however an expensive process. Given that the use of the impostors can greatly accelerate several aspects of the rendering, we decided to investigate if the same representation could be used to accelerate shadowing too. In the context of a virtual city with animated humans, we can differentiate 4 cases of shadow computations:

1. Shadows between the static geometry, e.g. buildings casting shadows onto the ground;
2. Shadows from the static onto the dynamic geometry, e.g. from the buildings onto the avatars;
3. Shadows from the dynamic onto the static geometry, e.g. from the humans onto the ground;
4. Shadows between dynamic objects, e.g. shadows of avatars onto other avatars.

In our current work [18] we address the cases 1 (partially), 2 and 3. We use fake shadows [6] to display shadows from the buildings on the ground and simple OpenGL lighting to shade buildings. The standard approach of using shadow maps was not used because it is problematic for very large scenes such as ours. The resolution of the shadow buffer is limited and thus the shadows end up appearing very blocky. Some recent work in improving this can be found here [38].

Addressing case 2 is not obvious. Having a multitude of virtual humans walking in a city model means having thousands of dynamic objects (and their shadow) to update in real time. This problem is extremely complex when considering it in a general case. However, our case can be assumed to be 2.5 D and therefore a 2.5D map can approximate the volume covered by the shadows. We call this map *shadow height map*. The idea is to discretize the shadow volumes and to store them in a 2D array similar to the height map of Section 2. In this way we can approximate the height of the shadows relatively to the height of the objects computing the difference between the value stored in the shadow height map and the original depth of the geometry (Figure 11). At run time is possible to compare the position of each agent against the height of the shadow volumes, and to compute the degree of coverage of a virtual human by a shadow. It should be noted that this approach works for any kind of animated object. If the objects are polygonal, the information stored in the shadow height map can be used to quickly compute shadows onto the polygons. In our case, we compute shadows for moving objects represented by the impostors. We then use a shadow texture mapped on each impostor to darken the part in shadow.

Case 3 can be treated with a different approach from the previous cases. As it is impossible to compute accurately the shadow of each virtual human on the environment, we decided to use the impostor structure for displaying the shadows as well. The idea is to use the light source position to se-

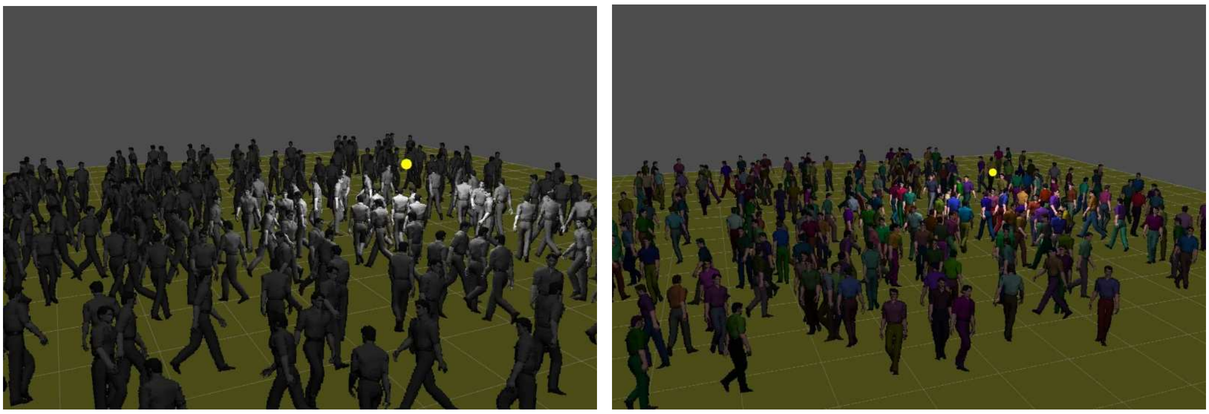
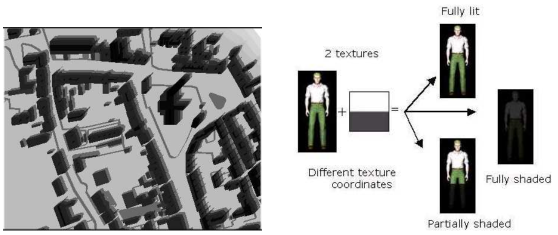


Figure 10: *Adding lighting and color information to the animated impostors.*



(a) (b)

Figure 11: *Computing shadows on the impostors using a shadow volumes-map. An example of a shadow height map is given in (a). In (b) the overall rendering of the shadows is illustrating, taking into account for the coverage of the shadow onto the virtual humans.*

lect the appropriate impostor image instead of the user view position. This image is then mapped on a black polygon projected on the environment (the ground in our case). Although this method might look simple, it is extremely powerful and allows highly realistic shadows, which are animated accordingly to the impostor. The texture is loaded once both for the virtual human impostor and for its shadow, which is important since it's an expensive operation. It doesn't take more memory consumption and it is as fast to modify the light position as it is to modify the view point. These shadows enhance greatly the realism with a negligible cost. One drawback though is the computation of the projected polygon. At the moment, we restricted it to be at ground floor, avoiding shadows on the building. We believe we could use a multi-level approach, computing the projection on the full environment only for the close objects. In [18], a detailed description of the results for the 3 cases can be found.

5. Implementation and results

Our test system was developed on a PC Pentium III - 800Mhz equipped with an NVIDIA 64 Mb GeForce GTS2 video card. This type of hardware is nowadays common and it even offers full support for OPENGL1.3 per-pixel lighting. We organized the rendering system in 2 separate modules: the first one is used to import the polygonal models created with a modeling software and to generate, optimize and assemble all the images resulting from the object sampling procedure. These data are stored in a single RGBA image, and saved on disk. At run time, our second module loads the images database, storing it in texture memory using a compressed RGBA format (GL_COMPRESSED_RGBA_S3TC_DXT3); these images are used to generate the impostors in real-time. As the base for our population, we used 6 different polygonal meshes (generated with CuriousLabs Poser), three for the male characters and three for the female characters. The limited number of different meshes used was due only to the lack of ready-available models, and a conspicuous part of the texture memory available was still unused.

5.1. Testing the Full Simulation, no Shading

At run-time, we rendered the impostors in 3 passes to draw different colors (chosen randomly for simplicity). We render up to 10,000 different instances of the base models, each with its own individual colours. These humans move in a village modeled with 41,260 polygons. The display is updated between 12 and 20 frames per second mostly depending on the polygonal complexity of the displayed geometry. It's important to note that, due to the nature of the impostors, there is no trade off between the character details definition and the speed of the rendering, at least as long as the user don't get too close to the avatars. Putting aside the popping artifacts mentioned before (often unnoticeable), the visual quality in most of the situation is reported by the majority of the

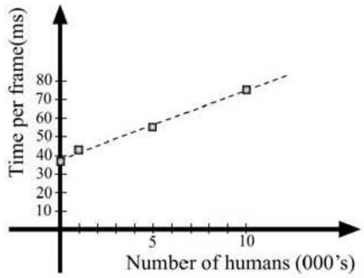


Figure 12: *Number of agents against time per frame.*

users to be the same as when using normal polygonal models.

To evaluate the scalability of our simulation, we also tried to run different simulations for 1,000, 5,000 and 10,000 people with a chosen camera path identical for each of them. By making the comparison of the rendering time, it has been noticed the lightweight representation of the crowd makes the rendering of the village model one of the slower tasks. For this reason, we believe that an occlusion-culling algorithm performed on the static model could further accelerate the overall rendering. The plot in Figure 12 represents the frame time vs. the number of agents in the simulation, and it shows clearly that the relation is almost linear, fact that makes the approach very scalable. It is to be noticed that these timings include the real-time collision detection, the basic behavior computation performed for each of the virtual humans simulated, and the shadows of the buildings and of the virtual humans.

5.2. Testing the Real-Time Shading

For the shading experiments we used a male character performing a cyclical walking animation. The polygonal count for the model was 8,440 triangles. The model was rendered from 32*8 different camera position, in two successive phases: the first is used to compute the pixels normal and the second for the regions that are controlled using the alpha-test.

The operations performed in our preliminary system were far from being optimized, but it nevertheless provides a basic platform sufficient to test the functionality of our algorithm, as it lets to the user the possibility to move a local light source around and to change parameters such as color, attenuation, and the intensity of the ambient, diffuse and specular component. We did not use the OpenGL higher precision accumulation buffer because rendering to the accumulation buffer was not hardware accelerated on our platform. It must be noticed that everything here was done with standard OpenGL functionalities; using different approaches, like using NVIDIA registers combiners functionalities, it should be possible to compact together some of the rendering passes, further speeding-up the lighting process.

Population	250	500	1,000	2,000
Avg. frame time(ms)	6.3	9.2	14.7	23.2

Table 1: *Rendering time for increasing number of shaded avatars*

Screen resolution (pixels)	500x400	640x480	800x600	1024x768
Avg. frame time(ms)	7.5	8.1	11.1	14.7

Table 2: *Rendering time as a function of the image resolution*

To test the scalability of our approach, we rendered some scene populated with different number of animated characters, and in particular we measured the average frame rendering time for populations of 250, 500, 1,000 and 2,000 individuals performing a walk-in-place animation. Each character has an independent orientation in space, to avoid any coherence in the pattern of texture memory reuse. Table 5.2 summarizes the results. As it can be seen the relation between the number of humans and the rendering time is almost linear, fact that proves the good scalability of the approach. We also decided to measure and study the relation between frame rendering time and screen resolution, due to the fact that our approach minimizes the geometry complexity of the scene but has very high fill-rate requirements. In this case we kept constant the number of rendered characters (1,000) and varied the screen resolution. As it can be seen in Table 5.2, performances decrease more or less proportionally to the number of pixels on the screen; this is a hint to the fact that this approach is mainly fill-rate limited.

It should be noted that we didn't use any particular strategy or order in the rendering process: on modern hardware architectures some scene graph sorting could probably increase significantly the performances, due to the increasingly common implementation of early occlusion tests, like hierarchical z-buffer or tile-rendering strategies. To sum up, the results prove that the lightweight representation of the animated impostors makes the rendering of crowds very efficient, even with the support of dynamic lighting. The current algorithms could certainly be used to render different kinds of objects, in particular vehicles, so that a full simulation of a complex urban environment should be possible. Clearly the method is not limited to the simple random walking animation used in our tests, and more elaborate animations are possible as long as there is enough texture memory available.

6. Conclusion and future work

In this article we presented some of the results we obtained developing a system for real-time rendering of densely populated large scaled environments. We have described a method for fast collision detection in complex city models that uses graphics hardware to produce a rasterization of space, which can be queried in minimal time. As a result we have shown that it is possible to achieve collision tests for a population of thousands of individual in real time. The algorithm presented proved to be easy to implement and adaptable to various models with different complexity. We have presented a system that facilitates the development and the visualization of behaviors for moving independent agents. The representation combines a 2D grid implemented in four layers to encode different levels of behavior. We believe that these four layers can be used to encode complex behaviors. The rendering method used allows real-time rendering of crowds using fully pre-computed animated impostors; for this reason, the rendering time of each avatar is independent from the complexity of its polygonal model and it's possible to render thousands of agents at interactive frame-rates. The amount of texture memory is minimized using texture compression, and we also use a multi-pass algorithm to fine-tune the color of different regions of the impostors. Finally, we add efficient shading and shadowing techniques that enhance the overall perceived realism.

Crowd visualization is a vast research topic and our currently research tries to improve the results of the existing system on several fronts. We are investigating the use of an efficient data-compression strategy to reduce the storage requirements for both the height-map data and the data stored in the other layers of the behavior simulation; this could allow the efficient storage of even large scenarios and the refinement and precision of the data. From the rendering point of view, we are currently working on the improvement of the per-pixel lighting and use of shadow buffers to further improve the realism of the simulated illumination.

It is our opinion that with the continuous increase of texture memory available on commodity hardware, the use of IBR approaches will be more and more feasible for real-time crowd visualization. We also believe that there is great scope for further improvement and developments of the technique. It will not take long before hardware supporting displacement-maps will appear on the market, making it possible to perform image-warping of the impostor, leading to the complete removal of the current visual artifacts. Moreover in our implementation we made a number of assumptions that could be re-examined. As we used the impostor images for the generation of the avatar shadows we implicitly made the assumption that the position of the light source is at infinity. In our examples this was not a limitation, as we were assuming the only light source to be the sun for which this approximation is acceptable. However, should we perform a simulations with different light conditions (for instance night time with streetlights), then we would have to properly warp the shadow textures before using them. Quicker ways to compute the shadow volume information could allow interactive updates for moving light sources. Full development of occlusion culling working on both the moving agents and the static environment could speed up the rendering substantially. Finally, an extension that could greatly improve the realism of our system would be the use of real photographic images of humans instead of synthetic models. To avoid the complexity of the data acquisition process (we need the depth-buffer for each image sample), the availability of high quality, reality-scanned human models to generate the image would probably be enough to bring it close to photo-realism.

References

1. F. T. a and Y.Chrysanthou. *Real-Time Rendering of Densely Populated Urban Environments*, pages 83–88. Springer Computer Science, 2000. Rendering Techniques 2000. 5
2. D. G. Aliaga, J. Cohen, A. Wilson, E. Baker, H. Zhang, C. Erikson, K. E. H. III, T. Hudson, W. Sturzlinger, R. Bastos, M. C. Whitton, F. P. B. Jr., and D. Manocha. MMR: an interactive massive model rendering system using geometric and image-based acceleration. In *Symposium on Interactive 3D Graphics*, pages 199–206, 1999. 5
3. A. Aubel, R. Boulic, and D. Thalmann. Animated impostors for real-time display of numerous virtual humans. In J.-C. Heudin, editor, *Proceedings of the 1st International Conference on Virtual Worlds (VW-98)*, volume 1434 of *LNAI*, pages 14–28, Berlin, July 1–3 1998. Springer. 5
4. A. Aubel, R. Boulic, and D. Thalmann. Lowering the cost of virtual human rendering with structured animated impostors. In *Proceedings of WSCG 99*, Plzen, Czech Republic, 1999. 5
5. S. Bandi and D. Thalmann. The use of space discretization for autonomous virtual humans. In K. P. Sycara and M. Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (AGENTS-98)*, pages 336–337, New York, May 9–13 1998. ACM Press. 2
6. J. F. Blinn. Jim Blinn's Corner: Me and my (fake) shadow. *IEEE Computer Graphics & Applications*, 8(1):82–86, Jan. 1988. 8
7. J. Cohen, M. Lin, D. Manocha, and M. Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments. In *Proceedings of*

Figure 13: *A scenario rendered in real-time*

ACM Interactive 3D Graphics Conference, pages 189–196, 1995. 2

8. W. J. Dally, L. McMillan, G. Bishop, and H. Fuchs. The delta tree: An object-centered approach to image-based rendering. Technical Memo AIM-1604, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, May 1996. 5
9. L. Darsa, B. C. Silva, and A. Varshney. Navigating static environments using image-space simplification and morphing. In M. Cohen and D. Zeltzer, editors, *1997 Symposium on Interactive 3D Graphics*, pages 25–34. ACM SIGGRAPH, Apr. 1997. ISBN 0-89791-884-3. 5
10. P. Debevec, C. Bregler, M. F. Cohen, R. Szeliski, L. McMillan, and F. X. Sillion. Image-based modeling, rendering, and lighting, July 1999. 5
11. P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *ACM SIGGRAPH 96 Conference Proceedings*, pages 11–20, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996. 5
12. E. Goffman. *The Individual as a Unit. Relations in Public: Microstudies of the Public Order*. Allen Lane The Penguin Press, London, 1972. 3
13. B. Hillier, M. Major, J. D. Syllas, K. Karimi, B. Campos, and T. Stonor. Tate gallery, millbank: a study of the existing layout and new masterplan proposal. Technical report, London, Bartlett School of Graduate Studies, University College London, 1996. 3
14. B. Hillier, A. Penn, J. Hanson, T. Grajewski, and J. Xu. Natural movement: or, configuration and attraction in urban pedestrian movement. In *Environment and Planning B: Planning and Design 19*, 1992. 3
15. P. M. Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, Sept. 1995. ISSN 1077-2626. 2
16. J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenberg. Real-time robot motion planning using rasterizing computer graphics hardware. *Computer Graphics*, 24(4):327–335, Aug. 1990. 2
17. M. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In *Proceedings of IMA Conference on Mathematics of Surfaces*, 1998. 2
18. C. Loscos, F. Tecchia, and Y. Chrysanthou. Real time shadows for animated crowds in virtual cities. In *ACM Symposium on Virtual Reality Software and Technology*, pages 85–92, Nov. 2001. 8, 9
19. P. W. C. Maciel and P. Shirley. Visual navigation of large environments using textured clusters. In P. Hanrahan and J. Winget, editors, *ACM Computer Graphics (Symp. on Interactive 3D Graphics)*, pages 95–102. ACM SIGGRAPH, Apr. 1995. ISBN 0-89791-736-7. 5
20. W. R. Mark, L. McMillan, and G. Bishop. Post-rendering 3D warping. In M. Cohen and D. Zeltzer, editors, *1997 Symposium on Interactive 3D Graphics*, pages 7–16. ACM SIGGRAPH, Apr. 1997. ISBN 0-89791-884-3. 5
21. L. McMillan and G. Bishop. Plenoptic modeling: An

image-based rendering system. *Computer Graphics*, 29(Annual Conference Series):39–46, Nov. 1995. 5

22. A. Meyer, F. Neyret, and P. Poulin. Interactive rendering of trees with shading and shadows. In *Eurographics Workshop on Rendering*, July 2001. 7

23. C. Mottram, R. Conroy, A. Turner, and A. Penn. Virtual beings: Emergence of population level movement and stopping behaviour from individual rulesets. In *Space Syntax - II International Symposium*, Brasilia, Brazil, 1999. 4

24. S. Musse, C. Babski, T. Capin, and D. Thalmann. Crowd modelling in collaborative virtual environments. In *Proceedings of VRST'98*, Taiwan, Nov. 1998. 3

25. S. R. Musse and D. Thalmann. A model of human crowd behavior: Group inter-relationship and collision detection analysis. In *Workshop of Computer Animation and Simulation of Eurographics '97*, pages 39–52, Budapest, Hungary, 1997. 3

26. K. Myszkowski, O. G. Okunev, and T. L. Kunii. Fast collision detection between complex solids using rasterizing graphics hardware. *The Visual Computer*, 11(9):497–512, 1995. ISSN 0178-2789. 2

27. B. F. Naylor. Binary space partitioning trees as an alternative representation of polytopes. *Computer-Aided Design*, 22(4):138–148, May 1990. 2

28. A. Penn and N. Dalton. The architecture of society: Stochastic simulation of urban movement. *Simulating Societies*, pages 85–126, 1994. 3

29. T. J. Prescott and J. E. W. Mayhew. Adaptive local navigation. In *Blake, A. and Yuille, A. Active Vision*, Cambridge, MA, 1992. MIT Press. 3

30. J. Rossignac, A. Megahed, and B.-O. Schneider. Interactive inspection of solids: Cross-sections and interferences. *Computer Graphics*, 26(2):353–360, July 1992. 2

31. H. Samet. *The Design and Analysis of Spatial Data Structures*. Series in Computer Science. Addison-Wesley, Reading, Massachusetts, U.S.A., reprinted with corrections edition, Apr. 1990. 2

32. G. Schaufler. Per-object image warping with layered impostors. In *9th Eurographics Workshop on Rendering '98*, pages 145–156, Vienna, Austria, Apr. 1998. EUROGRAPHICS. ISBN 0-89791-884-3. 5

33. G. Schaufler and W. Sturzlinger. A three-dimensional image cache for virtual reality. *Computer Graphics Forum*, 15(3):C227–C235, C471–C472, Sept. 1996. 5

34. SGI. Opengl texture compression. http://oss.sgi.com/projects/oglsample/registry/EXT/texture compression s3tc.txt. 6, 7

35. J. Shade, D. Lischinski, D. Salesin, T. DeRose, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 75–82. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996. 5

36. D. Smith, S. Pettifer, and A. West. Crowd control: Lightweight actors for populating virtual cityscapes. In *Eurographics UK 2000*, pages 65–71, Swansea, UK, 2000. 4

37. D. S.R. Musse, F. Garat. Guiding and interacting with virtual crowds in real-time. In *Proceedings of Eurographics Workshop on Animation and Simulation*, pages 23–34, Milan, Italy, 1999. 3

38. M. Stamminger and G. Drettakis. Perspective shadow maps. In *ACM SIGGRAPH*. 8

39. A. Steed. Efficient navigation around complex virtual environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST-97)*, pages 173–180, New York, Sept. 15–17 1997. ACM Press. 2

40. M. L. Stefan Gottschalk and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In *SIGGRAPH 96 Conference Proceedings*, pages 171–180, August 1996. 2

41. F. Tecchia, C. Loscos, and Y. Chrysanthou. Image-based crowd rendering. *IEEE Computer Graphics and Applications*, 22(2):36–43, March-April 2002. 5, 6, 7, 8

42. F. Tecchia, C. Loscos, R.Conroy, and Y.Chrysanthou. Agent behaviour simulator (abs): A platform for urban behaviour development. In *GTEC'2001*, Jan. 2001. 4

43. F. Tecchia and Y.Chrysanthou. Real-time visualisation of densely populated urban environments: a simple and fast algorithm for collision detection. In *Eurographics UK*, Apr. 2000. 2

44. G. Thomas and S. Donikian. Modelling virtual cities dedicated to behavioural animation. In *Eurographics 2000*. Blackwell Publishers, 2000. 4