Universitat Politecnica de Catalunya

Master in Innovation and Research in Informatics

Computer Graphics and Virtual Reality

Master Thesis

Improving Image-Base Rendering for Crowds and Perceptual Evaluation

Student: Maria Izquierdo Torrent

Directors: Nuria Pelechano Gomez & Carlos Andujar Gran

December 2014

Contents

List of Figures

Abstract

Human characters are usually represented as high-detailed textured polygonal meshes. Since rendering and animating these characters implies high computational cost, simulating large crowds of hundreds or thousands of agents in real-time has become a major limitation. Several impostor methods have been proposed in order to improve crowd simulation performance. We have focused on image-based impostors since they produce better visual results, more concretely on two per-joint image-based representations: *Relief Impostors* and *Flat Impostors*. First, to ease the processes of generation and rendering of impostors we have also developed a module of software, which is integrated in a prototyping framework for crowd simulation. The structure of this module is sufficiently generalized for including any type of impostor approach and we illustrate it implementing *One Texture Quad Impostors* as an example. After studying both per-joint techniques, we have centered on their limitations and proposed optimizations for some of their major restrictions without affecting significantly visual quality and performance. In order to analyze the visual quality of these techniques, we have carried out a perceptual experiment to study the optimal distances at which each one of the mentioned impostor techniques should be rendered without noticeable artifacts.

Chapter 1

Introduction

1.1 Introduction

During the last decade, crowd simulation has become an increasingly active area of research in computer graphics. Crowd simulation does not only simulate the movement of large number of characters or agents, it also includes individual behavior and interactions between these agents. It has to be as realistic as possible, in the sense that the appearance, the interaction and the animation of characters results believable and lifelike.

Taking advantage of the large number of improvements that have been made in the recent years, applications using crowds have expanded into other fields such as video-games and the film industry. Crowd simulation is appropriate in situations where realistic collective behavior is required, but the conditions can be unfeasible or unsafe for people in real life. Potential dangerous situations for human beings can be avoided by substituting these scenarios with virtual environments. Psychological and behavioral studies or simulations of building evacuations are some other examples of crowd simulation applications.

Besides the process of simulation, rendering and animation of each individual agent is remarkably important. The characters are represented as polygonal meshes and are textured to reproduce high-resolution details. Character's mesh is associated with a hierarchical skeleton-like structure so that each vertex of the mesh is related to one or more bones of the skeleton. Then, to animate the character, a three-dimensional transformation is applied to these bones, deforming the surrounding mesh and changing the current pose. Since there is a large amount of information to be computed per each agent per frame, rendering a crowd of several hundreds or thousand of agents often implies a loss of performance. Thus, rendering and simulating a realistic multitude of people in real time is still a challenging problem, especially for interactive applications, despite the significant advances of the graphic cards in recent years.

Several impostor methods have been presented in order to improve the rendering performance of crowd simulation. The general idea consists of simplifying the rendering method for those agents that are situated at a further distance from the camera position, but still rendering the nearest agents with their full geometry. By doing this, the visual quality of the crowd is not affected significantly and at the same time the performance is considerably improved. A common approach is to reduce the number of polygons of the character representation depending on the distance: the further away the agents, the fewer polygons used to render them [16]. However, reducing the number of vertices results in a noticeable loss of detail in the original mesh, which is clearly aggravated when the agent is animated.

A proper alternative are image-based object representations, because they are unaffected by the agent's complexity. In image-based precomputed impostors, the basic approach is rendering each agent as a single textured polygon [20]. During the pre-process, each character is rendered from predefined points of view and for a specific set of animation frames. While rendering a crowd, the selection of which image to use at a given time for each agent depends on its current viewpoint. One disadvantage of these methods is that the set of animations is predefined and fixed by the preprocess. Another inconvenient is the amount of memory used to store the generated textures.

As an improvement of these techniques, per-joint image-based impostors use a set of textures per each bone of the character [3] [1]. In this manner, the impostor is animated through bone transformations and precomputed textures remain independent from animations.

A prevalent limitation of impostor techniques is that they can only be applied for the furthest away parts of the scene, because they produce noticeable visual artifacts when they are applied to agents near the camera. Usually, perceptual studies have been conducted in order to analyze the most pronounced visual artifacts [10] [12]. Examples of these problems are popping artifacts due to the changing of views or, in the case of per-joint impostors, discontinuities in the joints resulting from the rigid bone animation.

In addition, some frameworks have been developed in order to let the user concentrate only on developing one part of the processes: simulation, animation or rendering of the agents. This kind of software allows the developer to focus on a specific area of research while using well known methods for the other areas. [2].

1.2 Objectives

Rendering realistic animated characters and providing high detail is still a limitation in large crowd simulation. Although image-based impostors are a good alternative to improve the high cost of rendering agents as polygonal meshes, there is an important problem to define a balance between performance and realism while rendering several characters in movement.

Hence, the main goal of this thesis is to improve crowd simulation rendering, in terms of performance and visual perception, so as to achieve a realistic rendering of a large multitude of agents through a real-time simulation. To achieve this objective, this project centers its attention on the major limitations of image-based impostor techniques. More specifically, we focus on techniques based on per-joint impostors, proposing some improvements to reduce visual artifacts without a loss of performance.

Another important issue that we cover in this thesis is the facilitation of the impostor generation pre-process. Therefore, we have designed and developed a new software module to generalize the impostor generation and rendering for a recently-presented crowd simulation framework.

Furthermore, since the visual evaluation of these techniques is merely subjective, we realize a rigorous study to analyze the users' perception of the common visual artifacts that appear when using rendering methods based on impostors to replace geometry. By doing this, our goal is to estimate the most appropriate distances from the camera at which a determined type of impostors can substitute geometry agents without affecting visual quality.

1.3 Contributions

In this work, after analyzing the state of the art, we present a module of software for the impostor generation and rendering processes. This software is not specific only for per-joint impostors or image-based impostors, it allows for any kind of impostor to be easily incorporated into the system. As an example, we implement the first approach presented in the literature of image-based impostors: *Real-Time Rendering of Densely Populated Urban Environments*, of *Tecchia et al.* [20]. Moreover, this application is integrated into an existing crowd simulation framework, easing the simulation of large number of agents including the added impostors.

Besides, we analyze two already presented image-based per-joint impostors' methods. Our contribution consists of detecting the most visible artifacts, determining the cause of these irregularities and then proposing a series of adjustments to correct them.

The first method studied is *A flexible approach for output-sensitive rendering of animated characters* [3] in which *Beacco et al.* introduced the concept of having precomputed per-joint information. The basic idea is to divide the character in a set of per-joint bounding boxes and store color, normal and depth values for each one of their faces. Then, the whole character is made up of the composition of each one of its joint boxes, rendered through *Relief Texture Mapping* [14].

The second method is *Efficient rendering of animated characters through optimized per-joint impostors* by *Beacco et al.* [1]. In this method precomputed information is also independent per each joint, but it is obtained by sampling each joint from a discrete set of viewpoints. The character is rendered using a single texture per joint and the selection of texture used depends on the joint orientation respect to the camera.

Finally, in order to validate our impostor improvements, we have run a user study to evaluate the perception of the visual artifacts of each method. This study is carried out by comparing full-geometry rendering to each one of the impostors' methods at different distances from the camera. The aim of this experiment is to determine, for each impostor type, the appropriate distance from the observer at which rendering impostors instead of rendering geometry agents does not affect the visual quality.

1.4 Organization

The rest of the work is organized as follows. In the next chapter 2 we discuss the state of the art on crowd rendering with impostors and perceptual studies. In chapter 3 we introduce our framework's module for the impostor's generation and rendering and we show an example. Then, in chapter 4 we present an accurate explanation of the Relief and Flat impostor methods, our improvements to each of them and the results obtained. The perceptual study design, its procedure and the results obtained are addressed in chapter 5. Finally, in chapter 6 we present conclusions and future work.

Chapter 2

State of the Art

The subject of rendering realistic crowds in real time has been studied in recent years. However, improving the rendering performance of a large number of agents without affecting the visual perception is still an intricate question. In order to understand the current state of the art on crowd rendering, in this chapter we explain the basic concepts about character representation and the way it is animated. Then, we discuss some crowd rendering acceleration techniques, focusing mostly on image-based impostors. Lastly, we analyze some work on perceptual studies applied to crowd simulation and rendering.

2.1 Character Representation and Animation

The most common approach for representing human characters are textured polygonal meshes. To animate them, a hierarchical structure called *skeleton* is placed inside the mesh or *skin*. The nodes of the skeleton represent joints and the edges represent the bones. Since each bone can be easily identified by its origin, we can use the term joint interchangeably. Then, the vertices of the mesh are assigned to the bones of the skeleton, through a *weight* factor. Consequently, the transformations applied to a bone are propagated to the linked vertices, deforming the mesh into a new pose. This process of association by weights is called *skinning*, and it is done with the mesh in the reference pose or *rest pose*.

The value of the weight factor is between 0 and 1, being exactly 1 the total sum of all weights per vertex. Normally, the vertices belonging to a joint and their neighboring ones are assigned to more than one bone, depending on how much the joint movement affects them. For instance, vertices closed to the knee would be linked to both the calf and the thigh, with weight less than 1. However, other vertices situated throughout the upper leg would only be assigned to the thigh and their weight would be exactly 1. This process of assigning weights and the skeleton fitting is called *rigging*, and it allows more flexible mesh deformations.

An animation is a smooth transition of movements in a fixed period of time. It is composed of a series of *keyframes*, each one defining a character pose for an instant of time. These body deformations are obtained by applying geometrical transformations to each bone of the skeleton. During the animation, the poses in between each frame are created by interpolating the two closest keyframes.

The transformations affecting joints in the hierarchy are assumed to be rotations. These rigid transformations are represented as matrices, so there is a matrix per each bone and keyframe. Since interpolating simple matrices can result in artifacts when deforming a mesh, the usual method for defining motions are *dual quaternions.* It avoids the creation of artifacts due to non-rigid transformation results from interpolation, such as the *candy-wrapper* effect.

2.2 Crowd Rendering Optimization

In order to simulate a realistic crowd in real-time, the major factors to consider are: the simulation process (including individual behavior and collective interactions), the animation of the agents and the rendering of each individual. Crowd rendering is usually the most costly in terms of performance, so to accelerate it the following techniques can be applied: culling, level-of-detail and image-based impostors.

To accelerate the rendering process of any crowded scene, culling techniques are a recurrent and effective option. They consist of discarding the objects of the scene that are not visible. *Frustum Culling* discards the objects that are not inside the camera frustum. Moreover, *Occlusion Culling* disposes of the objects that are occluded by other objects of the scene. Another possibility can be the usage of occlusion queries, which are part of a hardware functionality present in modern graphic cards. Nevertheless, these methods are usually applied to static geometry, so using them for crowd rendering is not a direct approach; it has to be taken into account that the agents are continuously in movement.

In crowd rendering, a widely used optimization technique is using different levels of detail *(LOD)* of the characters depending on their distance from the camera position [16]. An appropriate representation of the character is chosen according to their position in respect to the observer. Less complex geometry models are rendered as the agents move away from the viewer, and this considerably reduces the number of polygons to render. This multiple resolutions concept can be complemented with image-based rendering, using impostors for the furthest character's representations.

An option for improving the runtime performance are *static meshes* [15] as one of level-of-detail representation. Here, instead of saving only the

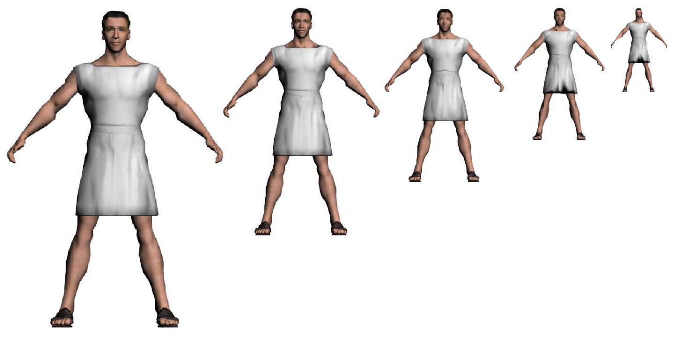


Figure 2.1: Levels of detail of geometry meshes

geometry of the reduced mesh and then animating it in real time, another possibility is storing a set of static or pre-deformed meshes. During a pre-process, a set of deformations of the low-resolution mesh are precomputed for each animation. By doing this, the rendering process is faster but it limits of the number of animations to use.

A different example of an optimization method for characters' rendering is *point-based impostors* [22]. This technique consists of dividing the view space into cells and then sampling the model from those cells. Fundamentally, the goal is to create a point based model based on the saved images, avoiding uncovered areas (holes) and aliasing from a specific view point. This technique does not need to maintain point-connectivity information. The best results are achieved when a triangle of the geometric model covers a pixel or less.

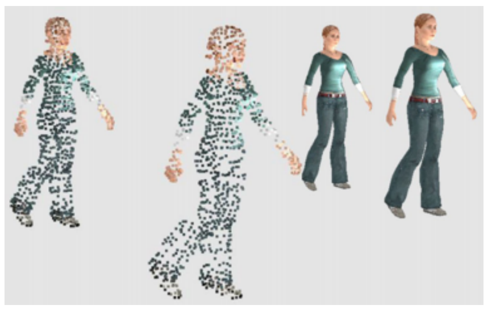
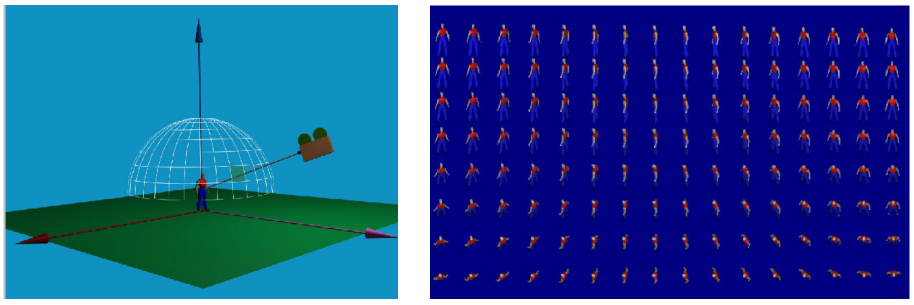


Figure 2.2: Examples of point-based animated representations

Nevertheless, in the following section we will focus on simple textured polygon impostors.

2.2.1 Image-based Impostors

In [20], *Tecchia et al.* proposed a technique for efficiently rendering the most furthest agents of a scene, without affecting the visual impact. Their approach consists of using only one textured quad per agent, approximating the agent appearance from that point of view. In order to animate the impostor, they capture a set of 32 views of the agent per each animation frame (covering it only from above). These views are saved as images and stored in a texture atlas. While rendering, the appropriate texture for each character is selected depending on the view point and the animation frame. In a posterior work [21], they optimized this method by mirroring the animation to take advantage of the body symmetries. In addition, to save storage memory, the textures are stored in a single compressed texture, and they are packed and the empty space between them is removed. Nevertheless, the method still suffers from the 'popping' effect when changing textures, due to the change of viewpoint.



(a) Discretization of view directions (b) Texture atlas of different character views

Figure 2.3: Generation pre-process of one textured quad impostor [20]

More recently, *Beacco et al.* [3] presented a per-joint impostor approach in which each character is represented by the set of oriented bounding boxes of its skeleton bones. For each one of the boxes' faces, the color, normal and depth information is saved in two textures as a pre-process. Then, each bone box is rendered through dual-depth relief mapping to simulate the original shape and is animated corresponding to the rigid transformation of the bone. In this manner, the pre-generated images do not depend on the animation used, so any motion can be applied to the character and at the same time the number of stored textures are notoriously reduced. Although the number of primitives to draw is smaller than rendering geometry, there is a higher per-fragment overhead. However, they found through a user study that using these *Relief Impostors* at certain distance is faster than rendering simplified geometry.

Lately, the same authors proposed *Flat Impostors* [1], another per-joint impostor method, based on rendering each bone as a single oriented texture. The whole character is made up by the composition of its joint textures for a given viewpoint. So per each bone, color and normal information are stored from a number of pre-defined view directions, which are taken from a cube

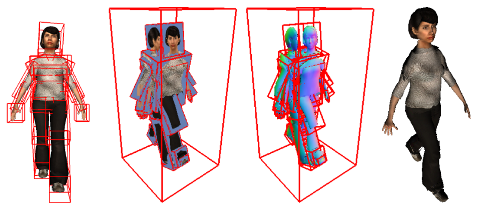


Figure 2.4: Relief Impostor example

map, in which a *Voronoi map* has been projected onto its faces. In order to avoid holes in the articulations when animating the character, in each texture is also codified through a mask parts of the neighboring joints. At runtime, the most appropriate texture per bone is chosen depending on the view, which is easily obtained by consulting the cube map textures. Then, animating the character basically consists of applying the rigid transformation of the representing bone. This image-based technique provides an efficient impostor rendering with animation-independence.

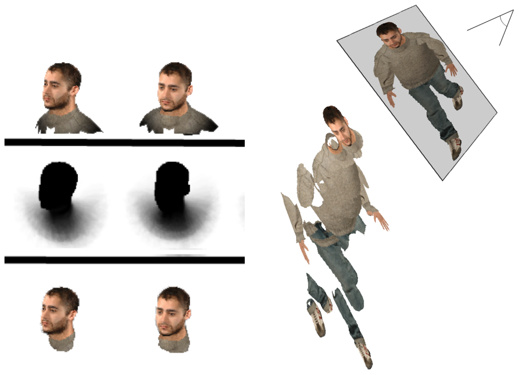


Figure 2.5: Flat Impostors example

2.3 Perceptual Studies for Crowds

Many physical phenomena (temperature, pressure, weight, etc.) can be objectively measured by instruments. However, there are some other concepts that are fundamentally subjective (such as appearance). The aim of perceptual studies is to analyze the response and sensations of the users to specific tasks or stimuli. In recent decades, these kinds of studies have been performed to investigate a wide range of topics, such as human behavior.

In computer graphics, perceptual studies are conducted to analyze the influence of the main factors that contribute to human-like simulation and rendering. Specifically, they produce a measure that helps to quantify the relationship between the parameters of the algorithms, the results they produce and the visual experience they create. For instance, in data compression, perceptual studies can help to determine whether an algorithm produces a visible loss of image quality. Or in photo-realistic rendering, where the results are expected to be lifelike, they can contribute determining if outcomes are plausible (in terms of shape, lighting, materials, etc.). Furthermore, in virtual reality, perceptual studies are most used to analyze and improve users' immersion in virtual environments. A scientific methodology to carry out these studies is *Psychophysics* [5], which proposes experimental proceedings to conduct an objective measurement of subjective experience.

Since in crowd simulation the goal is to represent a realistic group of human people, it should include individual behavior, character animation, collective interaction and believable appearance. In order to reduce the workload, perceptual studies can provide procedures of optimization and effectiveness improvement, to compute and generate only what is necessary. Moreover, these studies are a useful resource to analyze visual quality of the resultant crowd.

Focusing exclusively on animation studies, *Pražák at al.* [17] studied the motion cloning problem for crowds, where one animation is applied to several characters. For this study they used only one mannequin character composed of rigid segments and a set of human captured animations. They tested whether the number of crowd agents and the speed of the motion affect crowd variety. However, the main goal was to determine the minimum number of individual animations required to create a varied crowd of walking human. As a result, they found that with only 3 animations it is possible to create an animate varied crowd.

Alternatively, *Jarabo et al.* [8] studied the effect of lighting on crowd perception. Concretely, the goal was to determine whether approximations in global illumination for dynamic scenes are perceptually noticeable. For the experiments they used a scene with ambient illumination and a crowd of 28 non-textured agents, with some degrees of geometric, motion and illumination complexity. As a remarkable result, they found that motion affects the perception of lighting artifacts, at the same time that geometric complexity of agents masked them. Furthermore, artifacts were perceived easier with direct illumination.

The following sections center their attention in experimental studies that analyze the perception of characters variety and the most salience artifacts due to the use of impostor techniques.

2.3.1 Character Variety Perception

Rendering a large crowd using a different model per each agent and with no repetitions is unfeasible, due to the amount of resources needed. To achieve variety, some characters and animations should be replicated in the same crowd. The important point is finding the balance between resources' usage and character diversity.

The first ones to study how cloning appearance and motion affect crowd variety were *McDonell et al.* [11]. In their study, twenty capture motions and twenty human models were used, identifying manually each region of the models' textures to create diverse outfits for each model (varying hair, skin, clothing and shoes color). Firstly, through a series of tests where six static characters were shown and only two were cloned, they found that by color diversification it is possible to distinguish the appearance of repeated characters. Another study was conducted to determine the effect of cloned motions by means of identifying the only two repeated and in-step motions into a set of twelve identical models. The outcome was that duplicate animations were more difficult to be distinguished. Finally, a last experiment was conducted with a crowd of twenty agents in three scenarios: with several clones and no motion, with several motions and the same model for all of them, and with several clones and several motions. After running the three tests, they found that appearance clones can be detected quicker than repeated motions. Most of the false positives obtained were due to similar clothing between characters. Another finding was that appearance clones with the same motion but out-of-step was as effective as having many different motions.

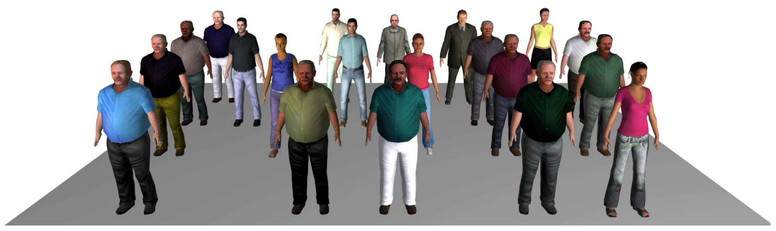


Figure 2.6: Scene example of appearance cloning tests

In the work of *McDonell et al.* [12], they studied (with the aid of eye-trackers) if there are parts of human characters within a crowd that are more looked at than others, and, if any, which are these parts. For that goal, they used ten human models and cloned five of them with color variation. The participants were asked to indicate whether a series of scenes with eight characters (with no occlusion between them and with different orientations) contain cloned characters. The study confirms that the head and the upper-torso were the parts most fixated on and also that varying the lower part of the body does not create visual diversity. Consequently, three variation techniques were proposed and analyzed: facial texture variation (including make-up and beards), facial geometry variation (altering face parts) and accessories addition. Here, ten characters were displayed and only one was cloned at each scene (using for all of them color variation and adding textures to the top clothes). As a result, they found that the three of these techniques were equally effective in order to differentiate clones.

2.3.2 Impostor Artifacts Perception

In crowd rendering, to obtain an accurate but inexpensive representation of the characters' models, the most used methods are pre-generated image-based impostors and low level geometric meshes. Therefore, many studies have been conducted to obtain an adequate balance between visual fidelity and low complexity.

In order to study the effectiveness of using impostors, *Hamill et al.* [6] experimented to find the point where the geometry of animated human models and buildings can be replaced by impostors without affecting visual perception. For that goal, they used pre-generated impostors for the human character but dynamic impostors for the four models of buildings (generating them when necessary). The first part of the investigation was focused on distinguishing the geometric representation from the impostor one at different distances (for humans and buildings separately), to detect the transition between them and to determine the acceptability of variations of the building dynamic impostor. In the second part, by comparing the animation of the human impostor representation to the geometric model, they found the impostor replicates accurately the motion. Lately, *McDonnell et al.* [13] has deepened this work with a study on the perception of motion and appearance of the geometrically simplified human models.

Most recently, *Larkin et al.* [10] studied the visual effects of geometric simplification of human characters using levels-of-detail (*LODs*). Specifically, they analyzed the artifacts' impact of reducing the complexity of textures, silhouettes, lighting or all three combined in both static and animated characters. Furthermore, there were only two different motions (idle and fast walking) and two ways of simplifying the models: fully automatic or aided, to take care of the areas which suffer from more artifacts while animation, as joints. By showing scenes with the original model in the center and a smaller one on each side (at different distances each time), the participants had to decide whether the model on the right or the left was the simplified one. The study showed that, despite the animation used, the most detectable artifact was the one due to silhouette simplification, especially at further distances. Finally, they experimented to find the optimal distances for using this simplification: distances extracted from the user perception and a metric obtained thought image comparison.

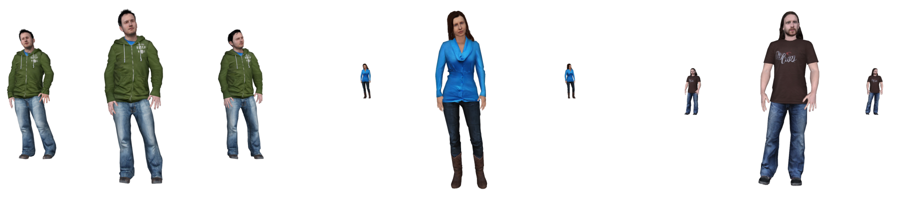


Figure 2.7: Scene example of simplifying artifacts test

2.4 Conclusions

The main problem using geometrical models to simulate large crowds is that rendering performance depends on the complexity of the characters. It means that the more number of detailed human models we have, the more primitives to render and, in consequence, the higher is the drop in performance.

In order to render a crowd of several hundred or thousand agents, some acceleration techniques have appeared in literature. Initial approaches have different levels of detail of simplified geometric meshes depending on the distance to the viewer [16]. Even though model simplification results in better performance, these meshes with less vertices introduce visual artifacts when they are animated. Static meshes [15] were thought to avoid these artifacts on simplified meshes. However, saving all the geometry information of the reduced mesh for a set of keyframes, produces an increment of the amount of memory needed and limits the number of animations. A totally different alternative are point-based methods [22], which consist of rendering surfaces using a huge number of points. This technique does not need connectivity information, but it gives the best results when the character is far away.

An alternative for simplifying geometry as a rendering acceleration are pre-generated image-based impostors. The first method consists of rendering an oriented quad and textured with different saved views of the character and for a set of animation frames [20]. It works properly for the furthest levels of the scene, but as inconvenient, a large amount of memory is needed and the animations are restricted to the stored ones. On the other hand, per-joint impostors render an impostor per body part instead of per character. A first approach [3] renders each joint as an oriented bounding box by relief mapping, using the previously stored information. The character is animated by applying the bone transformations to the joint boxes. A second method [1] represents each joint as an oriented quad, textured with a set of stored views from different directions. Similar to relief impostors, the bone transformations are propagated to the joint quads to animate the character.

Both techniques are untied to a predefined animation.

In order to perceptually analyze crowd simulation, several studies have been conducted. These studies examine separately each one of the factors that affect realistic simulation, such as character appearance, collective behavior, individual animation [17] or scene conditions like lighting [8]. Deepening in studies related to agents' appearance, there are two main groups. The first one examined the effects on diversity of cloning characters into a multitude [11] [12]. They also proposed some variation techniques to mask the use of repeated characters. The other group centered their attention on the perception of artifacts when using impostors [6] [10]. The main purpose of these studies were to find the optimal distance at which using impostors instead of geometry is not noticeable.

As mentioned in the previous chapter, our main objective is to improve realistic crowd rendering without affecting visual perception. The state of the art of image-based impostors encourage our work to improve some existing techniques, more concretely per-joint impostors, and to work on their major limitations. At the same time, the perceptual experiments revised in this section have evidenced the requirement of a rigorous user study to analyze and validate our proposal.

Chapter 3

Impostor Generation

This chapter first explains the *CAVAST* platform, a framework for simulating and rendering crowds. Secondly, the details of our impostor generation module are explained. Finally, we show a functional example of how an image-based impostor can be generated and rendered with our software.

3.1 CAVAST Platform

Crowd simulation consists in rendering human characters and animating them, at the same time that simulating collective behaviors. Despite these three areas are usually treated separately in research, they depend on each other during the simulation process. For instance, in the case of testing a rendering improvement some artifacts may have not appeared using static characters. Or a method is too expensive computationally and has to be optimized or can not be applied. Thus, these three elements: simulation, animation and visualization have to be considered as continuously interacting processes.

In order to provide a tool covering the three areas, *Beacco et al.* [2] presented a framework for researchers which allows to focus on one of these parts without losing sight of the other two. *CAVAST: The Crowd Animation, Visualization, and Simulation Testbed* is a new prototyping and development framework specialized in crowds. The objective of this work is offering essential tools to start working in this field and provide a basis on simulation, animation and rendering, avoiding the tedious work of starting a project from scratch.

As mentioned before, the structure of this frameworks is divided into three main areas: simulation, animation and rendering (see figure 3.1). As a link between all of them, there is the class *Agent* that provides a visual representation of the human characters to render. The explanation of the three parts of the framework and how they are related is explained in the following paragraphs, but with attention centered on the rendering part.

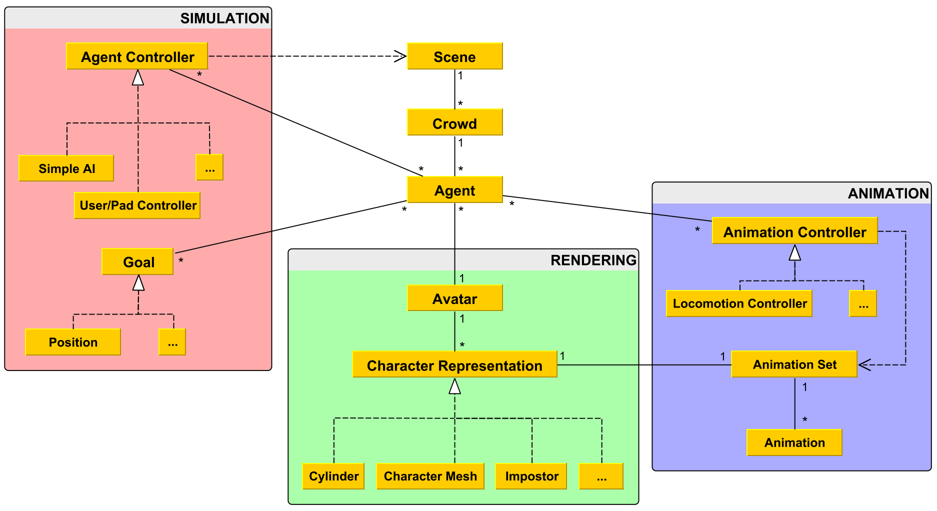


Figure 3.1: Diagram with an overview of the classes in CAVAST

The simulation module is responsible for moving the agent with the scene through the *Agent Controller* interface. This interface is the representation of a behavioral model, for example, walking through the scene (from one defined goal to another), and avoiding collision with obstacles and other agents. So basically, the *Agent Controller* interface is in charge of modifying the position, orientation and velocity of an *Agent* at each simulation execution, following a specified path. For simplicity, the scene is divided into nodes forming a regular grid. The assigned route of each *Agent* is computed by the *Pathfinder* interface, which uses a basic *A** algorithm to compute the optimal path from one node of the scene to another. The *Crowd* class is the one that iterates over all the agents during the simulation execution.

The *Animation Controller* is the interface in charge of animating each one of the *Agents*. In this case the approach used is skeletal animation. In fact, the animation is not applied directly to the *Agent*, but it is applied to the current used *Character Representation* of the *Agent* (as is explained in the rendering part3.1.1). For animating an *Agent*, the *Animation Controller* selects or synthesizes the best animation of an *Animation Set* in order to properly follow its current motion (depending on the velocity values of the agent's movement). Furthermore, if for example it uses an impostor approach as the current representation, the procedure of applying the animation may be different. In any case, animations have to be consistent between the different *Character Representations* used for the same agent.

Agents are sorted and grouped by their *Avatar* and *Character Representation* in order to send the transformation matrices to the GPU into vertex buffer objects (*VBOs*), and with only one render call with instancing. Since the objective is having individual agents performing different animations, the animation clips are loaded into the GPU to perform *matrix palette skinning*. By doing this, the computation of the blended pose is done in the vertex shader (with different animations instances and weights). It is done by blending within the two closest keyframes to the current one, for each motion clip and between the resulting poses of the different weighted animations.

The libraries used in this framework for simulation, animation and rendering are independent from each other and can be easily extended. However, this framework takes advantage of a library called *HALCA* [19] to animate and also render the characters. This hardware accelerated library, based on *Cal3D* format [4], performs the rendering and skinning parts on the GPU.

3.1.1 Rendering

An *Agent* has an *Avatar*, which is a collection of one or more *Character Representations* and the scaling dimensions of it. By having more than one *Character Representation* per agent we get different levels of detail (LODs). For instance, we can render the full-geometry representation only in the nearest part to the camera and render agents that are further away as impostors. The class *Avatar* is the one that has the representation size (width and height) and allows to maintain the consistency between representations.

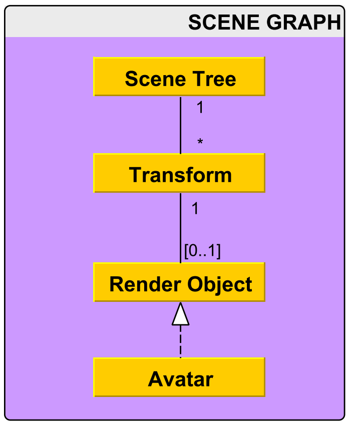


Figure 3.2: Diagram of the Scene Graph

For the scene render, this framework uses a *Scene Tree* as a scene graph (figure 3.2), and the nodes are *Transforms*. These *Transforms* are composed of one absolute transformation matrix and another one relative to its parent. They can also have a *Render Object*, which is an interface class for the scene graph to render the objects. Finally, an *Avatar* is a subclass of *Render Object*.

3.1.2 Impostor Generation and Rendering Module

The first part of our work consists of the software development to generate and render impostors. To do this, we have create and attached a new framework module for *CAVAST*, taking advantage of the already developed functionalities it has.

Our module is split into two different, but not independent parts: the generation and the rendering classes (see figure 3.3). Then, the most important classes are *Impostor Generation* and *Impostor Render* respectively. *Impostor Render* is a subclass of *Character Representation* and it is an interface class to create an impostor representation that can be added to the character *Avatar* and rendering. Additionally, *Impostor Generation* is a detached class used as a pre-process for the generation of the impostor images.

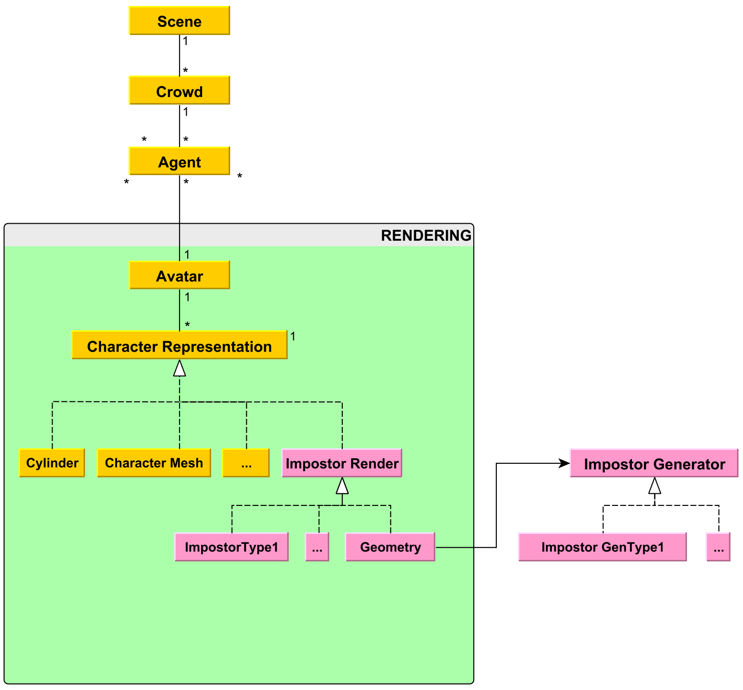


Figure 3.3: Rendering diagram with the Impostor Module (pink classes)

The key issue of module is the class *Impostor Geometry*, an *Impostor Render* subclass. This relevant class represents the full-geometry mesh of an avatar and it is used as input of the impostor generation process. Thus, it is altered to provide a wide range of tools, mostly for per-joint impostor creations. The most important features are:

- Rendering impostor joints separately, selecting the bone to render and without showing the rest of the mesh. Additionally, it is possible to change how many related vertices of the current bone are shown by the associated weight value. There is the option to show the parent and child bones from the skeleton of the current bone shown.

- Applying an animation to the character and also changing the current pose through the morph value (a position into the normalized animation time).

Generation

As mentioned before, the generation part uses an *Impostor Geometry* to store the information necessary for an impostor representation. Usually, this process is excessively complex to be performed every time before rendering the impostor (it may last a few minutes per agent). So generally, it is done as a pre-process, saving the information as images to disk. However, depending on the impostor type to create and if the generation process is sufficiently fast, it can be done on-the-fly each time before rendering.

The idea is to add a new subclass of *Impostor Generation* for each one of the impostor type the user wants to create. In some cases, the user interface should be modified in order to introduce required parameters for the generation.

Rendering

Once the pre-process is done and the information needed is saved, the next step is creating a class for rendering the impostor. This class has to inherit from *Impostor Render*, in order to render it as a character representation.

The usual process done in this new class is reading first the needed information from disk, and then creating the vertex buffer objects to render the representation.

Usually, during the generation process, the information related to the avatar (as vertices' topology, computed bounding boxes, etc.) is not saved to disk. However, we do save the reference to the original mesh from which we created the impostor representation. Since the geometric representation is usually loaded to be used for the characters' closest to the camera, when loading an impostor we can extract these parameters from the geometry.

3.1.3 Example: One Textured Quad Impostors

In order to offer a better understanding of these procedures, in this section we illustrate them with an example.

The example chosen is the approach of *Tecchia et al.* from the work *Visualizing Crowds in Real-Time* [20]. As explained in 2.2.1, their technique consists of replacing the furthest agents by one textured quad impostors. Basically, the process consists of sampling the character from determined view directions and for each frame of the animation. Then, the representation is rendered using the saved images, choosing them depending on the viewpoint and the animation point.

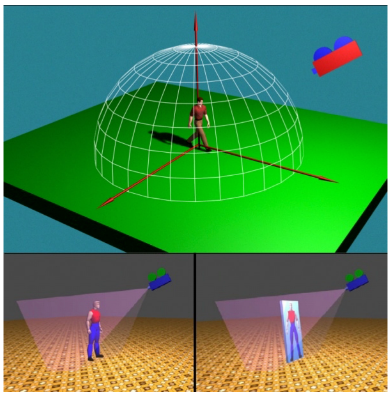


Figure 3.4: Sampling the geometry and replacing it with images [21]

Since this impostors were implemented for the purpose of perceptual evaluation, we have not included any of the optimizations that were carried out in later work to improve performance or memory footprint. Notice that non of these improvements affected the visual results.

Generation

First of all, we have to create an *OTQ Impostor Generation* class. This class has associated a geometry representation and the main functions to create it. Additionally, it has an animation clip defined since this specific type of impostor is tied to the motion. Thus, the number of sampled animation frames has to be defined in advance.

Once the character and the animation is selected, the procedure of the impostor generation function is the following: for each animation frame and for each predefined view direction, we set up an orthographic camera, and then render the whole geometry character. The camera is aligned to the current direction and is positioned in a way that the whole character is visible. Then, the color and normal information of the render are stored to disk as separated texture atlases.

The number of view directions is a parameter configurable by the user. For simplicity, we create an icosahedron to sample the character from a positive semi-sphere evolving it, and take its face normals as the view samples. Doing this, the user only has to indicate the number of face subdivisions as an easy way to define the number of views.

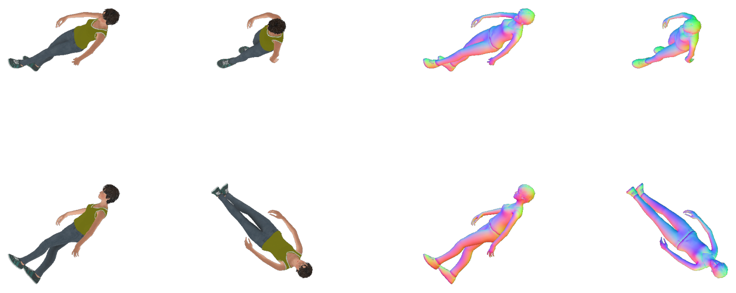

(a) Color textures example (b) Normal textures example

Figure 3.5: Texture atlases examples

Rendering

The character rendering is substantially tied to the generation process: we can only simulate the impostor characters that have been generated previously, and only with the motions they had during the creation.

The first step is reading from disk the images with color and normal information (this one is optional) and binding them as texture arrays. A cube map texture is also loaded, mapping the directions of the pre-computed view samples.

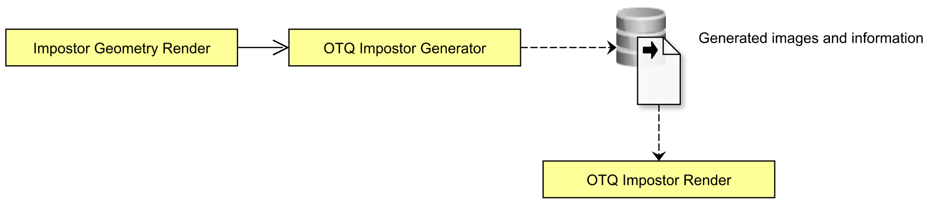
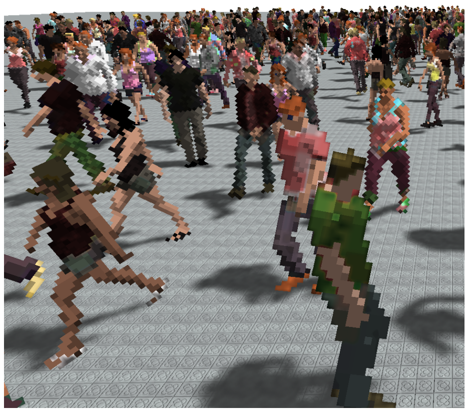


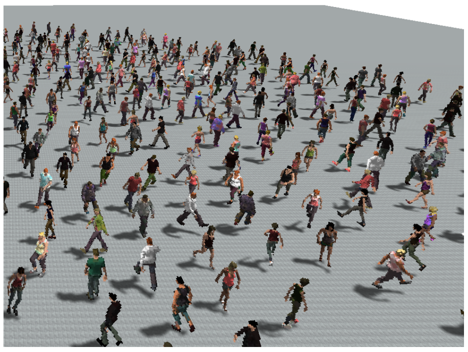
Figure 3.6: One Textured Quad Impostors' workflow

In addition, a single vertex buffer object (*VBO*) is created per agent type, storing the character's information. In this *VBO* is encoded: the center of the agent's bounding sphere and its radius (used for situating the quad impostor and scaling it), the number of textures per animation frame and the number of images per texture. This *VBO* is rendered as *GL_POINTS* primitives, converted later into two triangles forming the quad. In the vertex shader, the discrete view from the cube map texture is computed (depending on the impostor's orientation to the camera) and the appropriate texture is selected depending on the current animation frame. Then, the geometry shader creates two attached triangles forming a quad at the agent's position, oriented to the current view direction, and it calculates the texture coordinates. Finally, the fragment shader performs a texture mapping to compute the final color.

The following images show some scenes with agents represented as *One Textured Quad Impostors*.



(a) Street view



(b) Aerial view

Figure 3.7: Examples of two scenes completely rendered using *One Textured Quad Impostors*

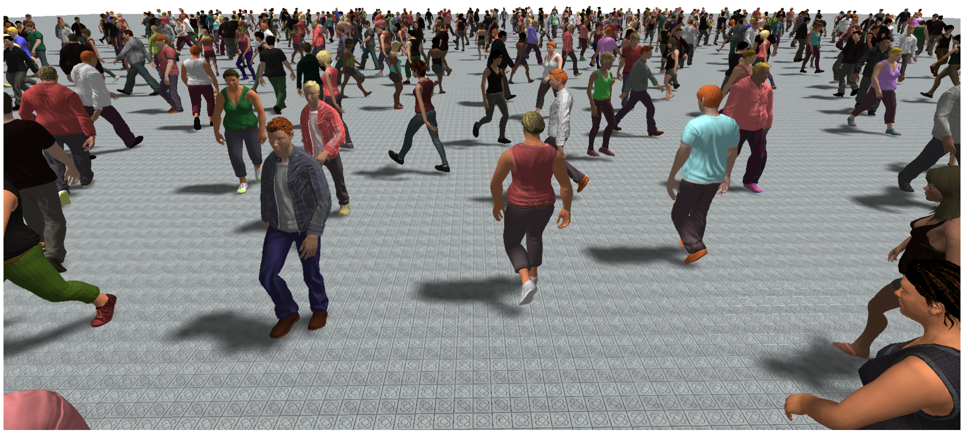


Figure 3.8: Scene where the closest agents are rendered as high detailed polygonal meshes and the *One Textured Quad Impostor* is used for the furthest away characters

Chapter 4

Impostor Improvements

In this chapter we present our improvements for the two well-known methods of per-joint image-based impostors. For each one of them, the first section presents a short overview of the approach. Next, the main problems of the techniques are analyzed and we explain the solutions we have proposed. Finally, we show the results we obtained with these new optimizations.

4.1 Relief Impostors

When we talk about *Relief Impostors* we refer to the impostor's technique presented in *A flexible approach for output-sensitive rendering of animated characters* by *Beacco et al.* [3].

The key issue of their project consists of rendering a character with a collection of oriented bounding boxes (*OBBs*), one for each bone of the skeleton. Each box is rendered through the *Relief Mapping* algorithm [14], projecting a texture onto its faces with the color and depth values. Then, this character representation is animated by applying the bone's transformations to each attached box.

The fact of having an impostor per joint instead of per character eliminates the necessity of being aware of the animation beforehand. Thus, only one set of textures per impostor is needed, independently of the viewing angles and the number of animations we want to simulate later. These features make evident the impostor versatility and also imply a considerable size reduction of the memory to store pre-computed information.

In order to simplify the animation stage, this approach takes advantage of *HALCA* [19] animation library.

4.1.1 Overview

The first step is choosing the number of divisions for the per-joint impostors' creation, in order to define which vertices are projected onto each bounding box. The easiest way for partitioning the character is throughout its joints, having a single box per each joint. However, it may lead to too many impostors per character and some of these boxes could fall inside a bigger one. A simple solution may be grouping more than one joint into the same bounding box, and applying to them the parent transformation for animating it. For instance, merging the hand joint with all the finger parts reduces considerably the number of boxes and the character motion is not visually affected. To avoid holes between boxes or occluded areas while capturing the images only the current joint is rendered at a time, with the neighboring vertices that are influenced by this joint. Different configurations of merged bones can be used as various levels of detail.

Secondly, a suitable character pose has to be selected for capturing the impostors' images. This pose has to represent an average pose to minimize the artifacts during the animation and it also has to guarantee a minimum total volume of the bounding boxes to save memory space. Once the character is in the selected pose, the oriented bounding boxes of the joints are computed from the initial axis-aligned ones by applying the required rigid transformations.

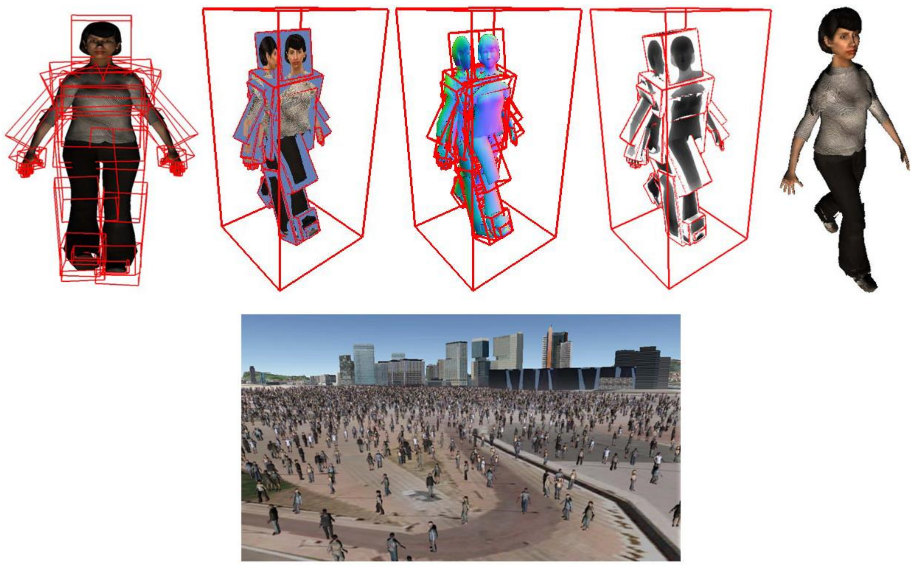


Figure 4.1: *Relief Impostors* overview: (*top from left to right*) character subdivision per joints, impostor boxes textured with color, normal and depth information respectively and resultant impostor representation. The image below shows a scene populated with this type of impostors [3]

Finally, the last step of generation is to render the polygon mesh in the selected animation pose, to capture each one of the six faces of the impostors' bounding box. For each face, an orthographic camera is situated with its viewing direction aligned with the face's normal vector. Then, only the part of the mesh related to the box is rendered, and the color, normal and depth information are captured and stored as texture images.

Once the set of impostors per character have been generated, the rendering process starts loading the generated images and binding them to the GPU as separate texture arrays (one for color and another one for normals, encoding depth values into both alpha channels). All the information relative to the joint bounding boxes is also bound as a *VBO*.

During the simulation execution, the *HALCA* library sends the rigid transformation matrices of each joint corresponding to the current motion pose to the GPU. With this information, the vertex shader computes the transformations of the joint bounding boxes following the skeletal animation.

The fragment shader is responsible for rendering each face of the overall bounding boxes, using a dual-depth version of the Relief Mapping algorithm. This method consists of finding the intersecting point of the fragment's viewing ray with the height field (encoded by the depth values into the textures). It is done by a linear search, sampling first the ray at regular intervals to find a ray sample inside the object, and this is followed by a binary search to find the intersection point. This allows recovering the diffuse color of the fragment and its normal component.

Having in mind how this approach works, in the following section we analyze the limitations of this approach while also presenting our improvements.

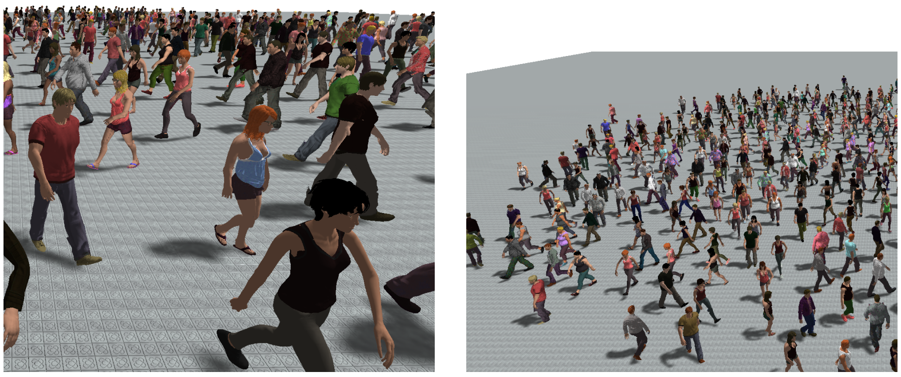

(a) *Relief Impostor* street view scene (b) *Relief Impostor* aerial scene

Figure 4.2: Scene completely populated with *Relief Impostors*

4.1.2 Our Improvements

In order to have a better understanding of the *Relief Impostors* technique we have re-implemented it from scratch, exploiting the resources offered by the impostor module embodied into the *CAVAST* framework. The initial approach of this method has been obtained by following step by step the details of their algorithm, creating a generation interface separated from all of the impostor rendering functionalities.

For the *Relief Impostors* we propose the following optimizations, which are described in detail below:

- Different levels of detail into the same character
- Customizable parameters through a user interface
- Variable number of linear and binary steps of Relief Mapping algorithm
- Optimizations on Relief Mapping algorithm

The following subsections describe each one of the above improvements and the limitations they cover.

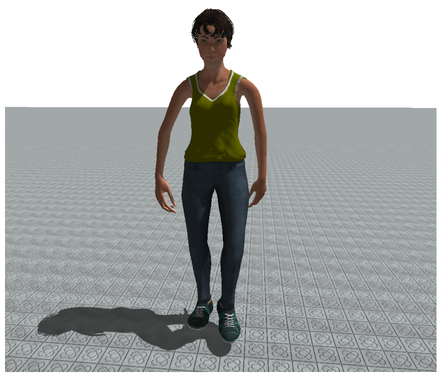
Different levels of detail into the same character

On the basis of the work of *McDonnell et al.* [12], the head and the upper-torso are the parts most fixated on when rendering a crowd. However, when using a single texture atlas for all the impostors of a character, the same image resolution has to be used for the overall body parts, regardless of their size or importance. For this reason, we propose rendering the character as separate parts with different parameters. More specifically in two main parts: the head and the rest of the body.

The process of rendering the character is similar to previous work, but handling now two texture atlases and two different *VBOs* (one encoding the head's information and another for the rest of the character's bounding boxes). Thus, two render calls per character are performed, allowing to vary some parameters between them.

It allows for better resolution on the head than on the body. Thus, the memory used for storing and transferring the images to the GPU can be decreased without an influence on the visual quality. However, rendering bigger textures implies increasing fragment processing. Additionally, having more than one texture atlas results in binding and unbinding textures at each frame, but this cost can be considered negligible compared to the overall rendering cost.

Taking advantage of splitting the character rendering into two parts, another approach consists of mixing both the polygonal mesh and the impostors in the same representation. Since the head is one of the most relevant parts of the character, it can be rendered as a high detailed mesh while the rest of the body is formed by impostors. Obviously, this approach will have a higher performance cost in respect to the fully impostor representation. However, it can be a desirable alternative for some parts of the scene.


(a) Geometrical representation

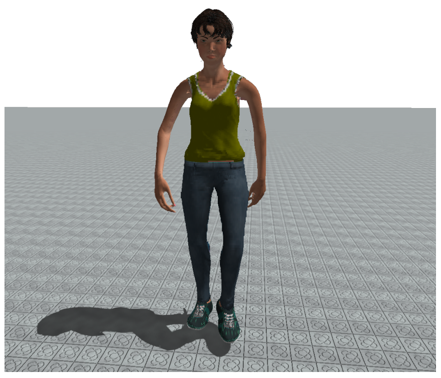

(b) *Relief Impostor*

Figure 4.3: *Relief Impostor* rendered using higher resolution textures for the head

Customizable parameters through a user interface

The desirable parameters used to create or render the impostors can vary depending on the scene or the system requirements. In order to ease both processes, we allow a modification of some parameters throughout the user interface.

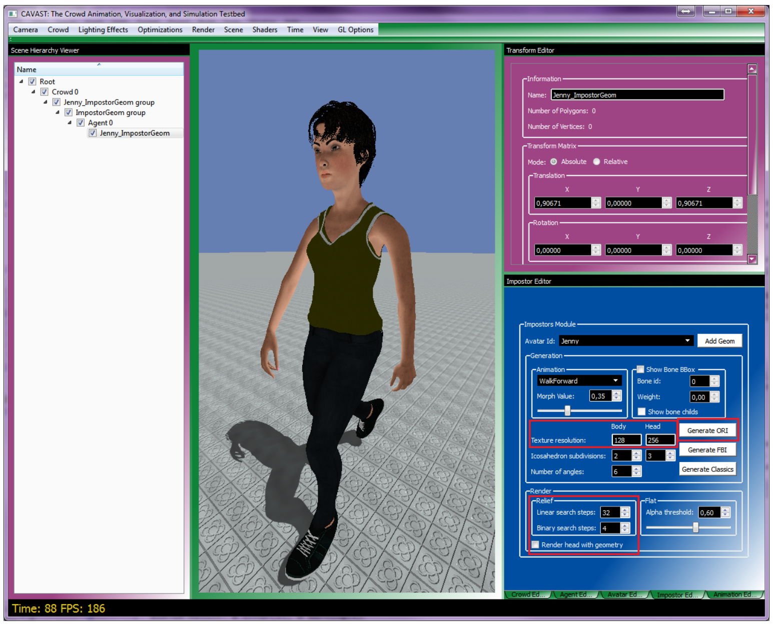
For the generation process, the texture resolution for the head and for the other parts of the body can be selected by the user. While rendering the *Relief Impostor* representation, the number of linear and binary steps of the *Relief Mapping* algorithm can be changed on-the-fly.

These aparently small improvements are extremely useful while searching the balance between visual quality of the representation and performance.

Variable number of linear and binary steps of Relief Mapping algorithm

The algorithm of *Relief Mapping* [14] consists of recovering the original geometry of the model by finding the intersection on a height field. To find this intersection point, an approached point is obtained through a linear search by sampling the viewing ray until the current sample falls inside the geometry or the maximum number of steps is reached. Then, a binary search is used to refine this first approximation, with another maximum number of steps. Since these two searches are executed per each fragment, they have a significant impact on the overall performance.

We propose a variable number of steps for each search depending on the agent distance to the camera. We have fixed a range for the number of steps of each search: from 16 to 128 for the linear search and from 4 to 10 for the


CAVAST: The Crowd Animation, Visualization, and Simulation Testbed
Camera Crowd Lighting Effects Optimizations Render Scene Shaders Time View GL Options
Scene Hierarchy Viewer
Transform Editor
Impostor Editor
Add Geom
Generate ORI
Generate FBI
Generate Classics
Crowd Ed... Agent Ed... Avatar Ed... Impostor Ed... Animation Ed...

Name
Root
Crowd 0
Jenny_ImpostorGeom group
ImpostorGeom group
Agent 0
Jenny_ImpostorGeom

Information
Name: Jenny_ImpostorGeom
Number of Polygons: 0
Number of Vertices: 0

Transform Matrix
Mode: ● Absolute ○ Relative

Translation
X 0,90671 Y 0,00000 Z 0,90671

Rotation
X 0,00000 Y 0,00000 Z 0,00000

Impostors Module
Avatar Id: Jenny

Generation
Animation: WalkForward
Morph Value: 0,35
Show Bone BBox
Bone id: 0
Weight: 0,00
Show bone childs

	Body	Head
Texture resolution:	128	256
Icosahedron subdivisions:	2	3
Number of angles:	6	

Render
Relief
Linear search steps: 32
Binary search steps: 4
Render head with geometry
Flat
Alpha threshold: 0,60

Time: 88 FPS: 186

Figure 4.4: Screenshot of impostors user interface

binary search. Then, further away agents will use the minimum number of steps for each search and as they get closer to the camera, this value will be increased..

Optimized binary search of the *Relief Mapping* algorithm by the *Secant Method*

As mentioned before, *Relief Mapping* recovers the original geometry by a finding the intersection on a height field in the fragment shader. Thus, it is an output-sensitive algorithm, which means the rendering time is proportional to the screen projection of the geometry instead of the polygonal complexity of the model.

To find the intersection of the viewing ray against the depth map, a linear search is performed along the ray in order to obtain a first point inside the surface. Then, a binary search provides a more accurate intersection point.

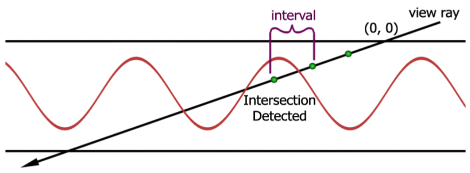


Figure 4.5: Linear search of *Relief Mapping* [18]

Since this technique has a high impact on per-fragment performance, we propose to optimize the binary search part of the algorithm by the *Secant Method* of *Risser et al.* [18]. Unlike *Relief Mapping* where a midpoint is assigned for the upper and lower bound, at each iteration of the *Secant Method* the intersection point is taken as one of the bounds. Then, with only a few iterations (usually less than four) the intersection of the surface is found. The figure below 4.6 shows an example.

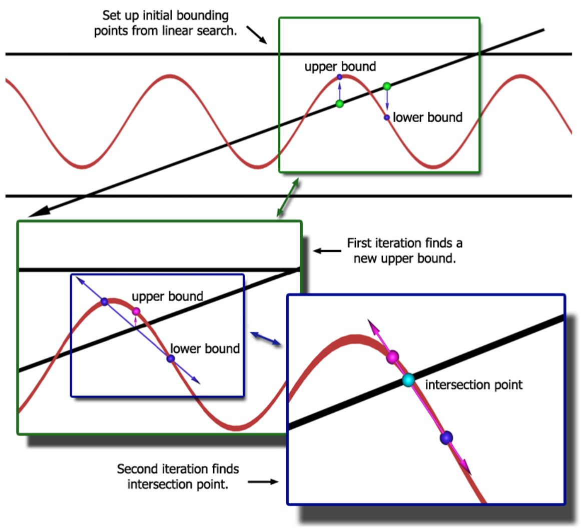

Set up initial bounding points from linear search.
upper bound
lower bound
First iteration finds a new upper bound.
upper bound
lower bound
intersection point
Second iteration finds intersection point.

Figure 4.6: Binary search using the *Secant Method* [18]

Even though this variant of binary search for *Relief mapping* converges in less steps, it is hardly noticeable since *Relief Impostors* are used in areas of the scene far away from the camera, and the impact on performance of the number of fragments is low.

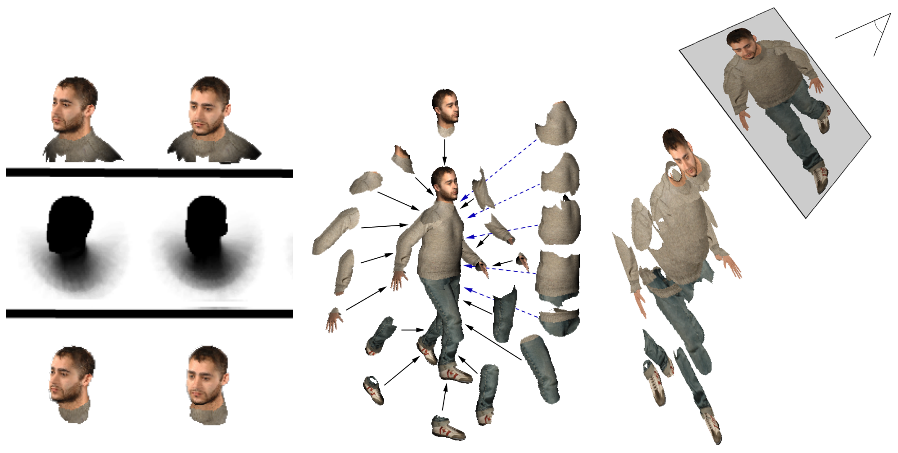
4.2 Flat Impostors

The name of *Flat Impostors* references another per-joint impostors' approach presented by *Beacco et al.*: *Efficient rendering of animated characters through optimized per-joint impostors* [1].

The main contribution of this work is representing each bone of the skeleton with a single oriented textured quad. In the pre-process each bone is sampled from a discrete number of views and saved as textures. During rendering, these textures are projected into the quads, depending on the joint orientation in respect to the camera. Character animation is done by applying each bone transformation matrix to the corresponding textured quad.

This work is based on the strategy of *Tecchia et al.* [20], which used a single textured quad for representing the whole character. The fact of having a collection of pre-computed views of the character increases the performance of the simulation, since the cost of rendering a textured polygon is low. Besides, *Flat Impostors* exploit the benefit of having a separate impostor for each part of the body, which unties them from having to know the animation cycle in advance.

As *Relief Impostors* does, this approach uses *HALCA* [19] as animation library.



(a) Mask generation (b) Texture combination to render the whole character (c) Texture orientation to the view direction

Figure 4.7: *Flat Impostors* generation and rendering

4.2.1 Overview

The first stage of the impostor creation is to decide which vertices of the polygonal mesh will be codified in each impostor to compose the whole character. Dividing the character exactly by its skeleton bones may result in an excess of impostors, some of them unnecessary. Thus, it is reasonable to merge groups of bones in a single impostor, such as each hand with its fingers or each foot with its toes. The resulting representation can be easily animated using the parent bone transformation for each part. For instance, the impostor containing one hand and its fingers will be animated using only the hand's transformations and ignoring the ones of the fingers.

Next, each joint is sampled from a discrete number of view directions using a *Voronoi* map. For simplicity, this map is replaced by a once-subdivided icosahedron projected into a cube map, each face of the icosahedron representing a *Voronoi* region, or in this case, a different view direction. Thus, this cube map encodes for each texel a color identifying its nearest discrete sample. The same cube map is used for the overall joints, thus, each joint is uniformly sampled as many times as the others. For each joint, the color and normal information of each view is saved to disk as a group of texture atlases, using an orthographic camera aligned to the view direction.

In order to avoid cracks or overlapping between joints while rendering the whole impostor representation, it is important to define which part of the geometry influenced by each joint is captured. For the contrary, including all the vertices which have any influence in that joint may lead in salient artifacts around joint boundaries. Thus, they compute an opacity mask defining a swept area for each joint boundaries, which covers all possible projections of the neighboring geometry influenced by the joint. For each joint, this mask is obtained by rendering a subset of realizations by uniformly taking different rotation angles from fixed angle limits. By doing this, the mask obtained covers a similar range of the movements of each joint and minimizes the overlapping artifacts between joints.

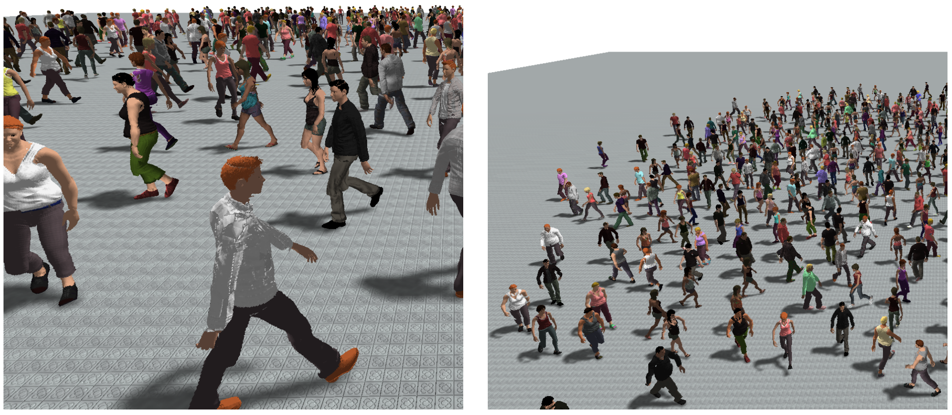

(a) *Flat Impostor* street view scene (b) *Flat Impostor* aerial scene

Figure 4.8: Scene completely populated with *Flat Impostors*

Once the impostors of the character have been generated, the created images are loaded and bound to the GPU as texture arrays (one texture array for color images and one for normals) and the cube map encoding the precomputed view directions is also sent to the GPU.

For rendering the character, a *VBO* is also bound to the GPU with the character information: a vertex position per joint (which represents the center of the impostor quad), the bone identification, the normal information of the joint bounding box and the layer of the texture array where the images of the joint are stored.

As in *Relief Impostors*, *HALCA* library is in charge of sending at each frame the rigid transformation matrices per joint for animating the character. Then, the vertex shader transforms the vertex position and the joint normals according to the current pose and it computes the view that best matches the joint orientation looking at the cube map texture.

The geometry shader creates two attached triangles defining the impostor quad and the texture coordinates for accessing the texture array. Finally, the fragment shader applies the texture to the quad by the usual texture mapping, computing the final color of the fragment. By using the opacity mask (codified in the alpha channel of the color texture), the amount of joint boundary that is shown can be adjusted.

In the next section, we explore the limitations of *Flat Impostors* while presenting our improvements for this method.

4.2.2 Our Improvements

After re-implementing *Flat Impostors* technique from scratch, including it in the impostor's module of the *CAVAST* framework. First, we developed an approach following exactly the proposed algorithm, but we have modified some parts in order to optimized the method and to obtain better results.

For *Flat Impostors*, we propose the following improvements, which are described in detail below:

- Different levels of detail into the same character
- Customizable parameters through a user interface
- Different angle ranges for joint blending

The following subsections detail each one of the above optimizations and the limitations they cover.

Different levels of detail into the same character

Similarly to *Relief Impostors*, using the same texture atlas for the overall impostors of the character limits to method having the same resolution on all the parts of the body. Furthermore, in the case of *Flat Impostors* a single cube map texture is in charge of encoding the different viewpoints for all the impostor joints. This implies that all joint impostors will be sampled from the same number of view directions. However, as mentioned earlier, the head is one of the body parts most looked at within a crowd [12], and providing more resolution to the head will bring on better visual results. For this reason, we propose rendering the character as two different main sections: the impostor head on one side, and the rest of the body on the other side.

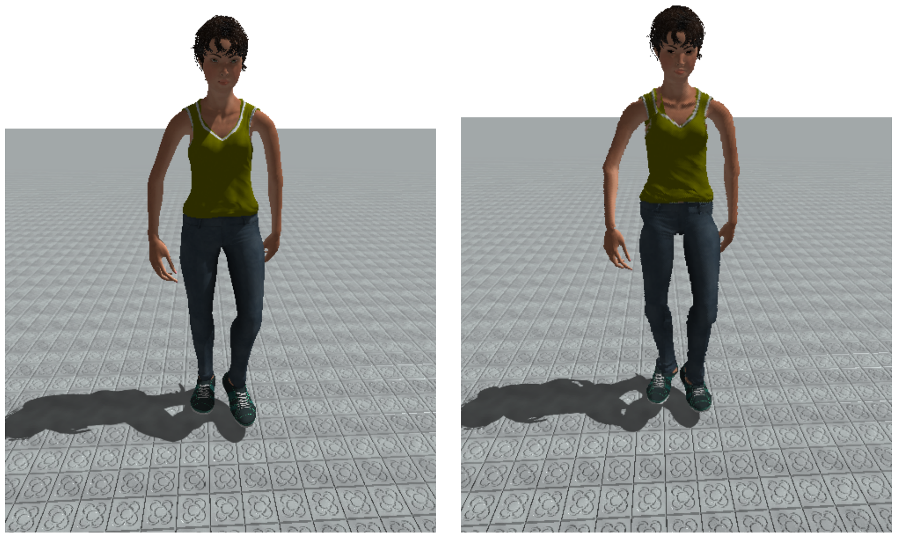

(a) Geometrical representation (b) *Flat Impostor*

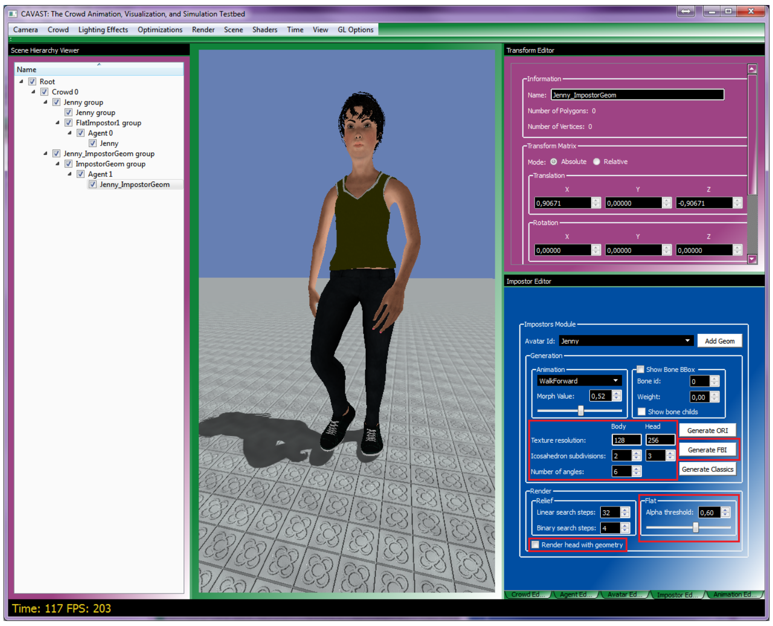
Figure 4.9: *Flat Impostor* rendered using higher resolution textures for the head

For the sake of giving prominence to the impostor head, we have created a new texture atlas and another *VBO* for this part. By this, the head's information is processed and rendered separately from the data of the rest of the body impostors. Thus, an independent cube map can be used only for codifying the view directions for the head, which allows having more samples from the head and with more resolution.

Even though creating images of higher resolution improves substantially the visual quality, the fact of having a different sample rate for the head has a bigger impact on the final rendering. Impostors with higher number of views increase the number of switches between views when the character changes its orientation. Thus, the 'popping' effect is less noticeable since changes between views are not so abrupt (images of adjacent views are more similar between them).

By doing this, the visual quality of the character representation is improved and the artifacts of relevant parts such as the head becomes less perceptible. It is achieved with a low impact on the current memory size, which makes it an important point since the high memory requirement is the most restraining imposition of this method.

In this case is also possible to create a hybrid representation: rendering the head as a polygonal mesh and having the rest of the body formed by *Flat Impostors*. Evidently, this will have a high repercussion on rendering performance but it removes completely the distracting artifact of 'popping' of the head and at the same time it provides high resolution detail.



Name
- [✓] Root
  - [✓] Crowd 0
    - [✓] Jenny group
      - [✓] Jenny group
      - [✓] FlatImpostor1 group
        - [✓] Agent 0
          - [✓] Jenny
    - [✓] Jenny_ImpostorGeom group
      - [✓] ImpostorGeom group
        - [✓] Agent 1
          - [✓] Jenny_ImpostorGeom

Information
Name: Jenny_ImpostorGeom
Number of Polygons: 0
Number of Vertices: 0

Transform Matrix
Mode: [✓ Absolute] Relative

Translation

	X	Y	Z
Translation	0,90671	0,00000	-0,90671
Rotation	0,00000	0,00000	0,00000

Impostors Module
Avatar Id: Jenny

Generation
Animation: WalkForward
Morph Value: 0,52
Show Bone BBox [ ]
Bone id: 0
Weight: 0,00
Show bone childs [ ]

	Body	Head
Texture resolution:	128	256
Icosahedron subdivisions:	2	3
Number of angles:	6	

Render
Relief
Linear search steps: 32
Binary search steps: 4
Flat
Alpha threshold: 0,60
Render head with geometry [ ]

Time: 117 FPS: 203

Figure 4.10: Screenshot of impostors user interface

Customizable parameters through a user interface

The parameters used to create or generate the impostor representation may be different depending on the situation. We have added to the interface a set of widgets to allow the user modifying these parameters.

In this case, the user can choose the image resolution for the head and for the rest of the body before generating the impostor images. At the same time, it can select a different number of views for sampling the head for the other body parts. For simplicity, since we approximate a *Voronoi* map with a subdivided icosahedron to compute the view directions, the user has to choose the number of icosahedron subdivisions instead of specifying the exact number of desired views.

To compute the opacity mask of the joints, another parameter that is tunable by the user is the number of angles sampled in the defined range. The higher the number of angles chosen, the more precise the mask will be but the slower this process becomes.

While rendering the *Flat Impostor*, the user can also adjust the amount of joint boundary that is shown by modifying the alpha mask value.

Different angle ranges for joint blending

To avoid overlapping joints or holes between impostors, an opacity mask is calculated for each joint boundary. To compute it, all possible projections of the geometry that is influenced by the joint have to be taken into account. For simplicity, they defined a range of rotation angles for each one of the three degrees of freedom (*DoF*) of a joint, and they rendered a discrete subset of realizations per each joint by sampling values within these angle limits.

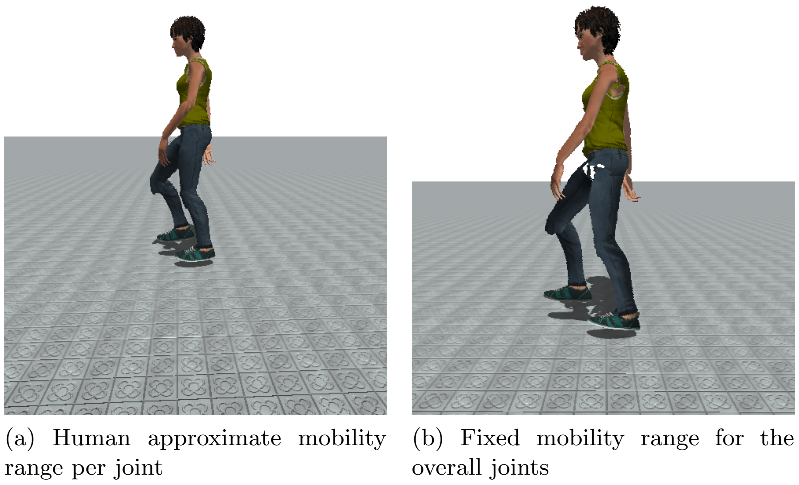

(a) Human approximate mobility range per joint
(b) Fixed mobility range for the overall joints

Figure 4.11: Differences

The main restriction of this, is that they used the same range of angles for all the *DoF* of all the joints, unlike human joints which have different values for the different joints. For example, the neck has rotations values very different from the knee, or articulations such as the elbow only has one degree of freedom for its rotation. For this reason, we propose to adapt each one of these angle limits to the real human values. Some of these values have been extracted from the chapter *Joint-Articulating Surface Motion* of the book *Biomechanics* [9], but most of them were taken from the book *Physics of the human body* [7]. The following table shows the range of mobility for the most relevant joints 4.1.

By limiting the rotation angles of each joint we can obtain opacity masks more realistic and accurate, fitting better for generalized animations.

opposing movements	mean	SD
shoulder flexion/extension	188/61	12/14
shoulder adbuction/adduction	134/48	17/9
shoulder medial/lateral rotation	97/34	22/13
elbow flexion	142	10
forearm supination/pronation	113/77	22/24
wrist flexion/extension	90/99	12/13
wrist adbuction/adduction	27/47	9/7
hip flexion	113	13
hip adbuction/adduction	53/31	12/12
hip medial/lateral rotation (prone)	39/34	10/10
hip medial/lateral rotation (sitting)	31/30	9/9
knee flexion (prone) - voluntary, arm assist	125, 144	10, 9
knee flexion - voluntary (standing), forced (kneeling)	113, 159	13,9
knee medial/lateral rotation (sitting)	35/43	12/12
ankle flexion/extension	35/38	7/12
foot inversion/extension	24/23	9/7

Table 4.1: Mobility range for the most relevant joints [7]

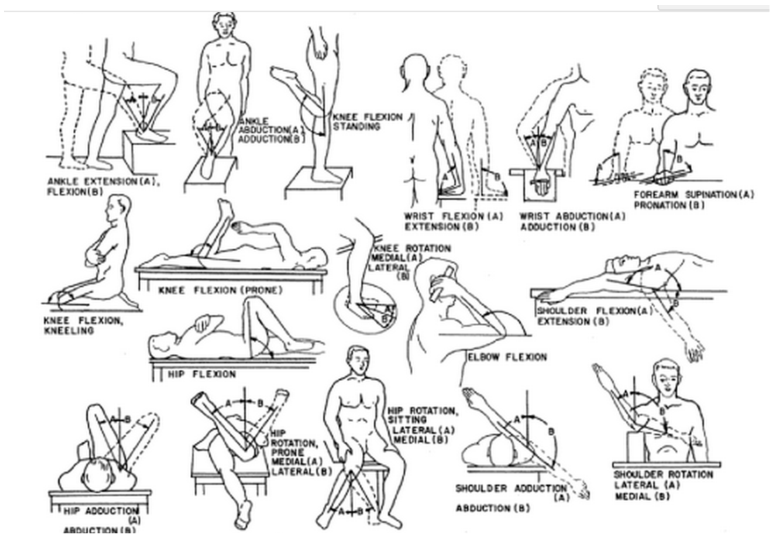

ANKLE EXTENSION (A), FLEXION (B)
ANKLE ABDUCTION (A) ADDUCTION (B)
KNEE FLEXION STANDING
WRIST FLEXION (A) EXTENSION (B)
WRIST ABDUCTION (A) ADDUCTION (B)
FOREARM SUPINATION (A) PRONATION (B)
KNEE FLEXION, KNEELING
KNEE FLEXION (PRONE)
KNEE ROTATION MEDIAL (A) LATERAL (B)
SHOULDER FLEXION (A) EXTENSION (B)
HIP FLEXION
ELBOW FLEXION
HIP ADDUCTION (A) ABDUCTION (B)
HIP ROTATION, PRONE MEDIAL (A) LATERAL (B)
HIP ROTATION, SITTING LATERAL (A) MEDIAL (B)
SHOULDER ADDUCTION (A) ABDUCTION (B)
SHOULDER ROTATION LATERAL (A) MEDIAL (B)

Figure 4.12: Postures used for capturing the angle ranges [7]

4.3 General Improvements

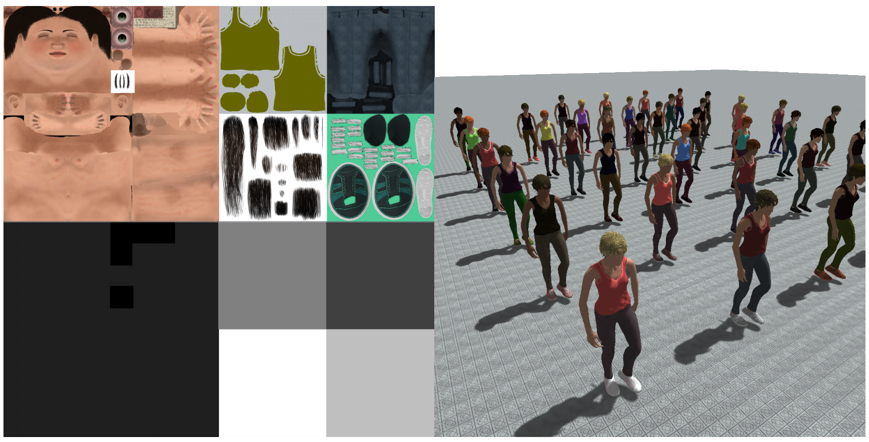
Aside from the specific improvements for both techniques *Relief* and *Flat Impostors*, we propose the following series of optimizations that are independent from the method used, which are explained in detail below.

- Generalization for different skeletal configurations
- Color diversity
- Pre-computed lighting

The following subsections detail each one of the above optimizations.

Color variety

To provide variety to a large crowd it is not feasible to have hundreds of different models, because it will greatly affect the performance. The usual number of different models used for a crowd is between 3 and 10 [11]. However, the simplest way to add variety is 'per body part color modulation'. This hardware accelerated technique consists of modifying manually the alpha channel of the model's texture, given different values to the different parts: skin, hair, top and bottom cloths and shoes. We capture this channel during impostor creation and added it into the impostors' textures.



(a) Diffuse color (*top*), color masks (*bottom*)

(b) Scene with the same model with different colors

Figure 4.13: Color variation example

Then, in the fragment shader the *HSV* color of the pixel can be modified depending on this value. For the hair, we have two types of variation (depending if we decide the agent will have blonde or brown hair). Skin color is taken from a set of predefined colors, in order to be as realistic as possible.

Pre-computed lighting

Adding color variety through color modulation occupies another channel of the impostors' generated image. In the case of *Relief Impostors* it was a problem since we saved for each impostor: 3 channels for color, 3 channels for normals and 2 channels for depths, in two *RGBA* images.

Since we prioritize character variety more than illumination, a luminance factor is computed rather than saving the 3 normal components. Then, during the generation process, we fixed a light on the scene and computed the resultant diffuse color of the model. With this color, we calculate a luminance factor for each fragment.

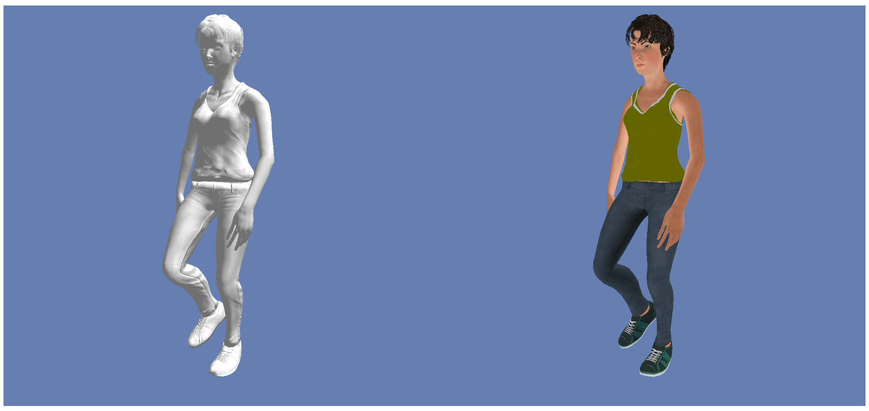

(a) Luminance factor (b) Diffuse color

Figure 4.14: Luminance factor and diffuse color of the model

During the rendering, this luminance factor is used for computing the shading of the fragment instead of using the normal components.

Chapter 5

Perceptual Study

This chapter explains the experimental evaluation realized to analyze the perception of the common visual artifacts due to image-based impostors rendering methods. Concretely, we compare the visual perception of rendering impostors against pure geometry to obtain the preferable distance at which using each impostor method is not visually appreciable. Then, after explaining the study setup and the procedure used, we analyze the results obtained and discuss some final aspects.

It has been observed that every impostor technique produces visual artifacts when it is used for rendering the closest characters of the scene. Consequently, the further away from the camera the impostors are situated, the less noticeable their artifacts become. For this reason, the nearby characters are usually rendered as a high-detailed mesh and switching to their impostor representation at a determined distance. The key issue of each impostor method relies on finding the optimal distance to achieve a trade-off between performance and visual quality.

In order to find the most favorable distance to render each impostor type, we have carried out an experiment to evaluate the perception of the most common visual artifacts in crowds, due to image-based impostors rendering methods. Concretely, the aim of this study is to determine the optimal distance at which it is feasible to switch from rendering polygonal meshes to each one of the impostors without losing visual quality. More specifically, the three types of impostor techniques that we have analyzed separately are *Relief Impostors* [3], *Flat Impostors* [1] and *One Textured Quad Impostors* [20].

5.1 Experiment Design

To analyze visual quality of impostors in crowds, we set up a collection of populated scenes.

In real-time crowd applications, typically between 3 and 10 templates are used. Thus, for our study, we used a set of 6 characters (3 males and 3 females) representing typical pedestrian types: middle-aged and wearing casual attire. Each character had its single texture map which incorporated all of the textures for hair, skin and clothes. As explained in 4.3, we manually created an alpha map encoding color variety.

McDonnell et al. found in their work [11] that having a single animation played out-of-step has the same impact on motion variety as having many different animations. For this reason, we only use a unique normal-walking motion cycle for all the characters played off-phase.

The scene consisted of a textured quad for the ground plane (representing the typical street ground tiles) and a plain white background. For the simulation we use a simple controller included in the *CAVAST* 3.1 framework. For each agent, this controller randomly assigns a new goal position once the agent has reached its current one by following a pathfinding algorithm. We did not force it to produce a deterministic simulation for all the different scenarios, to avoid users searching similarities between simulations or being influenced by resultant patterns.

For the study, we have fixed some parameters such as: texture resolution of impostors, number of views and number of merged joints.

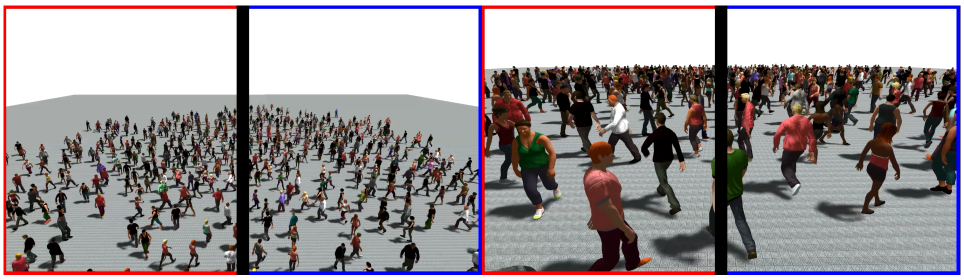

(a) Camera 1 (b) Camera 3

Figure 5.1: Two snapshots of the videos showed to the users

For the experiment, we had recorded a set of videos showing an animated crowd of 500 agents, with a duration of 30 seconds each. There were a total of 12 videos: one for each type of impostor method, from two different camera's viewpoints and with two different crowd-density values. In both points of view the camera is situated above pointing to the multitude.

In each video the viewport was vertically divided into two parts: agents positioned on one half were rendered completely as high-detailed polygonal meshes and on the other side were represented as impostors. As an initial configuration of the impostors' side, only the furthest agents were rendered using an impostor representation while rendering the rest as polygonal meshes. During the simulation, the frontier for switching between representations was decreasing (moving towards the camera), so at each time the area of the scene with impostors starts closer to the camera. It means that during the video the number of impostors in the scene increases gradually, until the overall agents of the impostor's side of the viewport were represented as impostors. The decision of which side of the video showed the impostor representations was done randomly.

During the simulation, this frontier between the geometric and impostor representation moves towards the user with a non-uniform velocity. To avoid fast representation switches when an agent is moving around the frontier, agents have a minimum time for being rendered with the same representation. They also have different distance offsets to the switching distance, preventing a visible wavefront when changing representation.

All geometric and impostor representations have pre-computed illumination (as explained in 4.3), and project shadows to the ground.

5.2 Experiment Procedure

Twenty two participants (14 males and 6 females, aged between 22 and 66) participated in this experiment. They were from different educational backgrounds and only 8 of them had any experience with the field of computer graphics (the rest were naïve). All the participants had normal or corrected to normal vision.

The experiment was displayed on a wide-screen 23 inch LCD monitor with a resolution of 1920*1080 pixels. The distance from the user to the screen was approximately 60 cm and they were asked to respect that distance. Lighting conditions of the room were the same for all the cases. The user forms are attached in the annex 6.2.

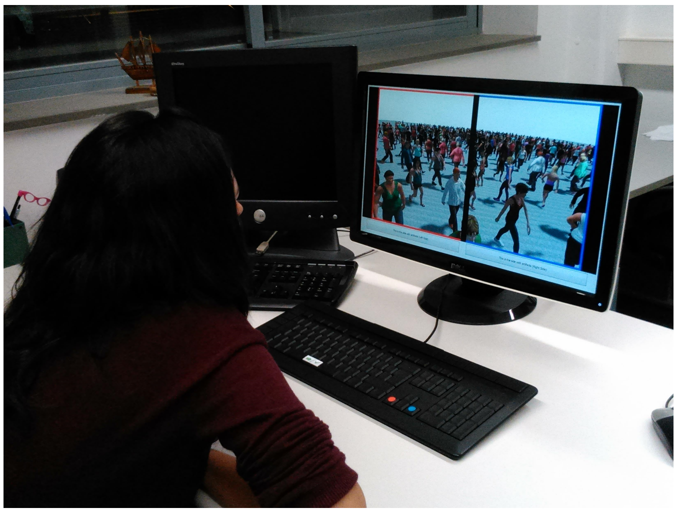


Figure 5.2: User test setup

During the experiment, the participants watched all the videos 3 times, in a random order to avoid ordering effects. So, each user watched 36 trials (3 impostor types * 2 densities * 2 viewpoints * 3 repetitions), with a withe screen in between videos to alleviate eye strain.

The users' task was to determine which side of the video is the one with visual artifacts. Thus, users were asked to select one side as quickly as possible when they had noticed any visual difference during the video reproduction. If the video finished, the last frame image was shown frozen and the user had to make a choice. Participants indicated their decision by pressing a button on a modified keyboard. We recorded both the accuracy of their responses and their reaction times.

At the beginning of the experiment, the participants watched a video with a simulated crowd formed only by polygonal characters (using the same character as in the test videos). It was done to familiarize them with both the characters and the simulation used, showing them an example of what we considered an optimal rendering. We remarked the fact that other effects related to the simulation (such as foot sliding or errant character paths) or scene rendering (such as lighting or shadows) were not the kind of errors we were looking for. However, we did not specify what the expected artifacts looked like in order to obtain neutral results.

At the end of the test, the participants had to indicate what artifacts they found throughout the videos.

5.3 Results

After running the experiment, we collected the overall data to analyze the results.

We had encoded into a file the correspondence between frame second and distance of switching to the impostor representation for each video, so we obtained that distance from the time response of the users.

Impostor type	**OneTexturedQuad**	**Flat**	**Relief**
Mean time (s)	10,5	21,1	23,9
Mean distance	15,5	10,7	8,9
Miss percentage	6%	9%	15%

Table 5.1: Mean results per impostor type

The data was processed in the following way: we computed a mean of the three trials per video for the distance and the time response, but ignoring the cases when the user did not select the correct side of the video (the one with artifacts). We also counted the number of user hits per video. At this point, we had one sample per user and per video (22 users per 12 videos).

First of all, we computed general mean times of response and mean impostor distances for each one of the impostor types for the overall users. Additionally, the percentage of hits per technique is shown 5.1. From this table we can clearly deduce that *One Textured Quad Impostors* are the most easily detectable, since the users answered quicker and with only 6% of wrong answers. On the other side are *Relief Impostors* with the highest failure rate (15%) and the longest response time. However, *Flat Impostors* seems to be in the middle point of the two extremes.

Looking at this results, we could assume that, for achieving good visual results, the rendering representation order should be: *Relief Impostors* for closest areas to the camera, then *Flat Impostors* and finally *Classic Impostors* to the furthest part. However, a deeper analysis has to be performed to have a better understanding of the factors that influence the results obtained.

There was a video for each type of impostor, for 2 different crowd densities and from 2 points of view, so the three factors to analyze are: *impostor type*, *density* and *camera* or view point. Since we were measuring the distance of the impostor representation area to the camera, the goal is to find out if the obtained distance is influenced by these factors.

Considering that the two views used were decidedly different (an aerial camera and a street view), we can assume beforehand that the camera position is a decisive factor and has high influence on visual perception.

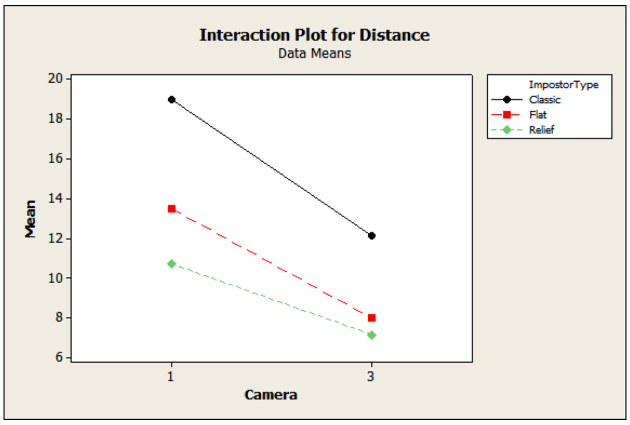


Figure 5.3: Distance vs. Camera interaction plot

An interaction plot shows how the relationship between one factor and a continuous response depends on the value of another factor. Then, looking at the plot, the more non-parallel the lines are, the greater the strength of the interaction. In the interaction plot 5.3, we can observe that the view point used has an influence on the distance obtained for the different impostor types. This is a reasonable result because a closer view to the crowd will lead in more occlusions depending on the density and, at the same time, some artifacts become more noticeable because of the characters proximity to the camera.

Since the camera position through the scene is obviously a decisive factor, we have divided the rest of the analysis in two parts (one for each view point) to study the influence of the other factors.

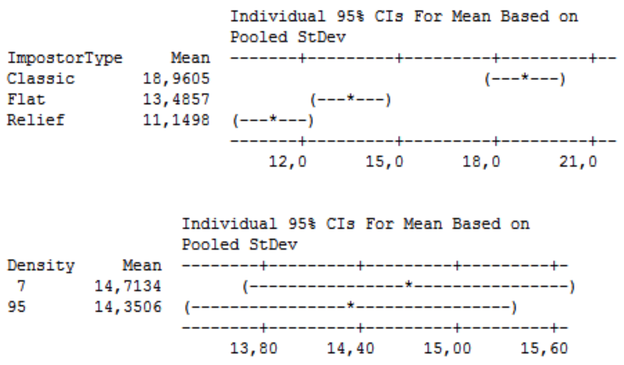
5.3.1 Two-way ANOVA

In order to study the relation between the distance and the two factors (impostor type and density), we have applied a *Two-way ANOVA* analysis (Two-way Analysis Of VAriance) with 95% of confidence for each one of the view points.

Source	DF	SS	MS	F	P
ImpostorType	2	1414,40	707,199	40,87	0,000
Density	1	4,34	4,345	0,25	0,617
Interaction	2	1186,43	593,217	34,28	0,000
Error	126	2180,18	17,303		
Total	131	4785,36			

S = 4,160 R-Sq = 54,44% R-Sq(adj) = 52,63%

Table 5.2: Two-way ANOVA of camera 1


```
                       Individual 95% CIs For Mean Based on
                       Pooled StDev
ImpostorType     Mean  -------+---------+---------+---------+--
Classic       18,9605                              (---*---)
Flat          13,4857          (---*---)
Relief        11,1498  (---*---)
                       -------+---------+---------+---------+--
                           12,0      15,0      18,0      21,0


                  Individual 95% CIs For Mean Based on
                  Pooled StDev
Density     Mean  --------+---------+---------+---------+-
 7       14,7134        (----------------*----------------)
95       14,3506  (----------------*----------------)
                  --------+---------+---------+---------+-
                       13,80     14,40     15,00     15,60
```

Figure 5.4: Intervals of confidence Camera 1

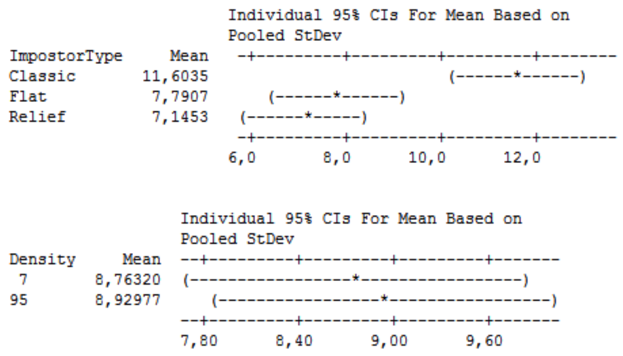
For the *camera 1* (from above), the obtained values are shown in the table below 5.2. From these values we can infer that there was a main effect of changing the impostor type ($p - value = 0.00 < 0.05$), so there were significant differences between the three groups. This is reasonably since each group represents a completely different technique. On the other hand, we found no effect of density ($p - value = 0,617$), which implies that the distance at which participants detected artifacts (using this camera) is not affected by crowd density. We can also concluded that exists an interaction between the two factors (p-value of the interaction is less than 0.05).

Source	DF	SS	MS	F	P
ImpostorType	2	510,82	255,412	12,74	0,000
Density	1	0,92	0,916	0,05	0,831
Interaction	2	183,04	91,520	4,56	0,012
Error	126	2526,12	20,049		
Total	131	3220,90			

S = 4,478 R-Sq = 21,57% R-Sq(adj) = 18,46%

Table 5.3: Two-way ANOVA of camera 3

For the *camera 3* (from eye height), the table below shows the resultant values 5.3. From these values we can deduce that there was also a main effect of changing the impostor type ($p - value = 0.00 < 0.05$), so it is reasoned since we are testing three different techniques. As with the other view, we found no effect of density ($p - value = 0,831 > 0.05$), which implies that the distance is not affected by crowd density. In this case does also exist an interaction between the the impostor type and the density ($p - value = 0.012$).



```
                       Individual 95% CIs For Mean Based on
                       Pooled StDev
ImpostorType      Mean  -+---------+---------+---------+--------
Classic        11,6035                         (------*------)
Flat            7,7907      (------*------)
Relief          7,1453   (------*-----)
                        -+---------+---------+---------+--------
                       6,0       8,0       10,0      12,0

                  Individual 95% CIs For Mean Based on
                  Pooled StDev
Density      Mean --+---------+---------+---------+-------
 7        8,76320 (-----------------*-----------------)
95        8,92977    (-----------------*-----------------)
                  --+---------+---------+---------+-------
                  7,80      8,40      9,00      9,60
```

Figure 5.5: Confidence intervals of camera 3

The main conclusion of these analysis is that the type of impostor influences the mean distance, unlike crowd density which does not affect it. However, the two conditions (type of impostor and density) interact by any means.

5.3.2 Cumulative probability of distance

The following graphs illustrate for each configuration *how frequently was detected that one side of the video had worse render quality than the other.* So, the vertical axis measure the probability that at given distance users will detect artifacts in one side (taking only into account the correct answers). The resultant curves are known as a *psychometric function* (or frequency of seeing curve) [5], and they show that the probability of detecting visual artifacts increases with the distance.

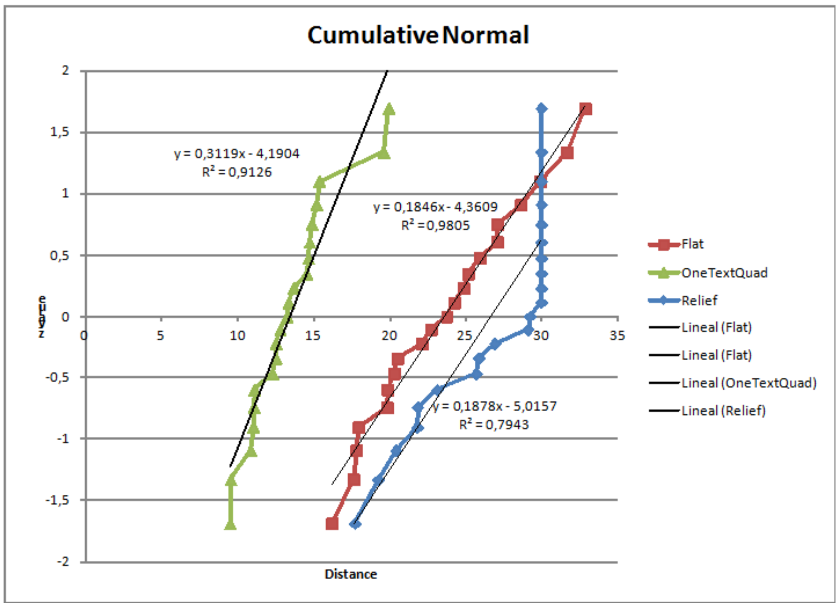


Figure 5.6: Cumulative probability of distance: Camera 1, density 7

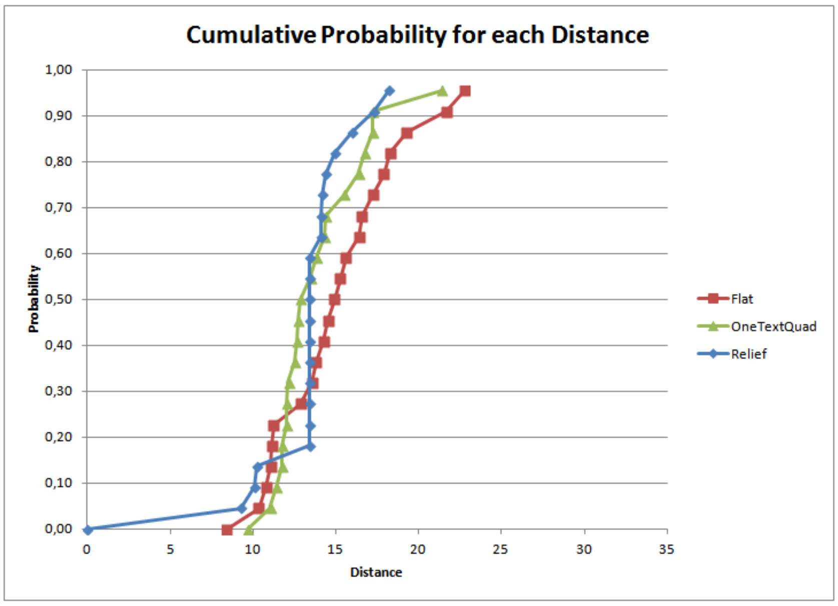


Figure 5.7: Cumulative probability of distance: Camera 1, density 95

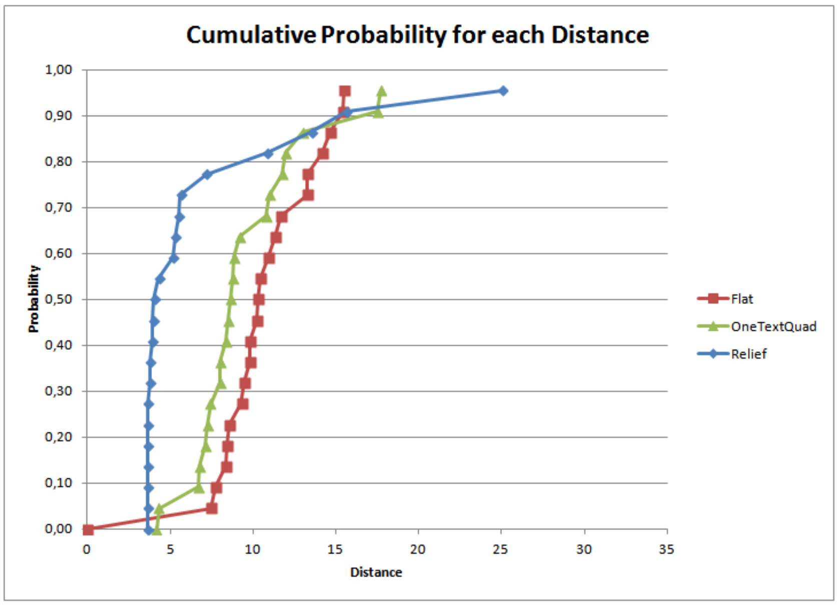


Figure 5.8: Cumulative probability of distance: Camera 3, density 7

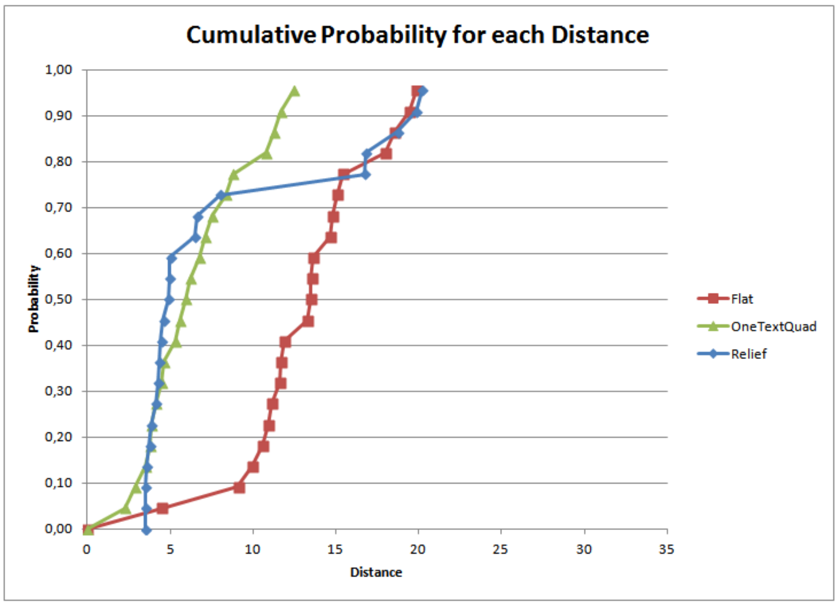


Figure 5.9: Cumulative probability of distance: Camera 3, density 95

These graphs are obtained by fitting the data with a model that describes the response behavior (*cumulative normal* or *probit analysis*). The psychometric function of the cumulative model is:

$$p^i = \int_{-\infty}^{z_i} e^{z_i^2/2} dz$$

$$z = (i - \mu_i)/\theta_i$$

when p is the integral of an exponential function and z is a parameter that is related to both the mean and the variance of the responses collected in the experiment. We can compute these z values that correspond to the probabilities in the cumulative normal function and plot then as a function of the distances. Then, if the cumulative normal model is a good fit to the data, it can be approximated by a straight line.

Each one of the following graphs, show the cumulative normal model and the approximated regression line. Each line is described by its equation and its R^2 parameter which quantifies the goodness of this approximation. For instance, a curves with a line approximation with $R^2 = 0,9126$ means that this curve can be approximated by the given line with a 91,26% of confidence (example *OneTexturedQuad* figure 5.10).

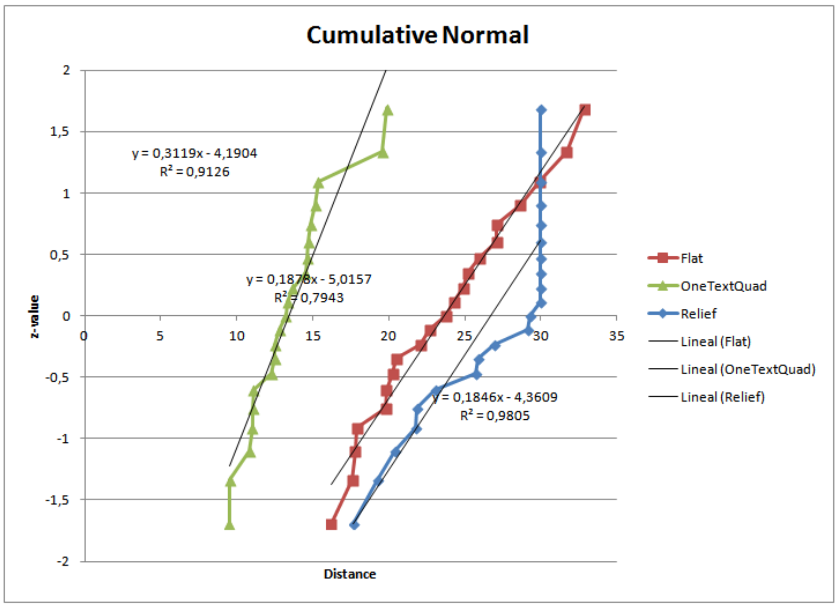


Figure 5.10: Cumulative normal: Camera 1, density 7

From this line approximations, we can use their slope and intercept to calculate the mean and standard deviation of the cumulative normal function that fits the data. With this information, we can extract the predicted distance at which users will start detecting artifacts for each one of the

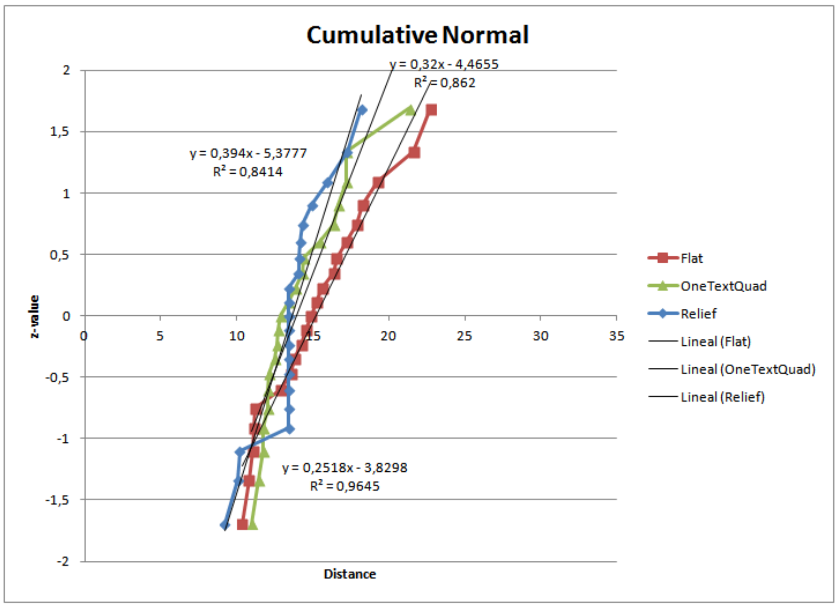


Figure 5.11: Cumulative normal - Camera 1, density 95

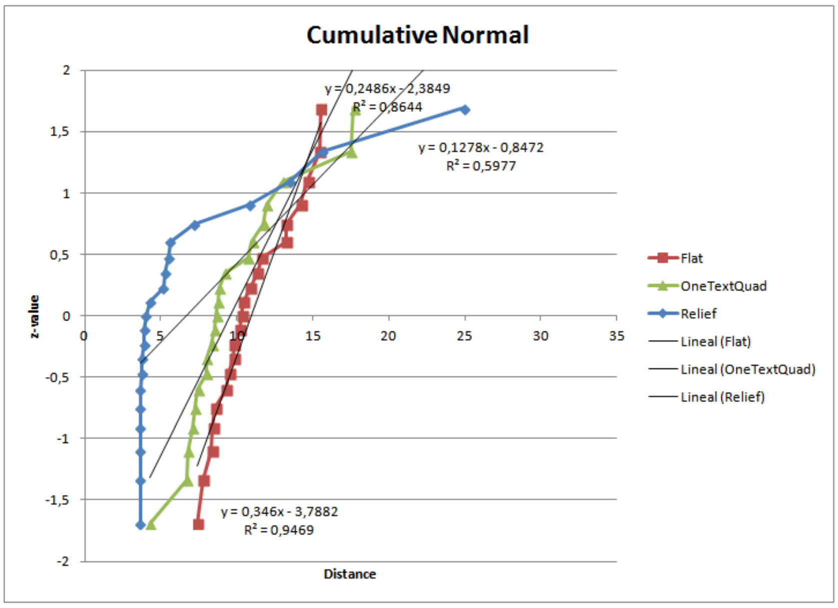


Figure 5.12: Cumulative normal: Camera 3, density 7

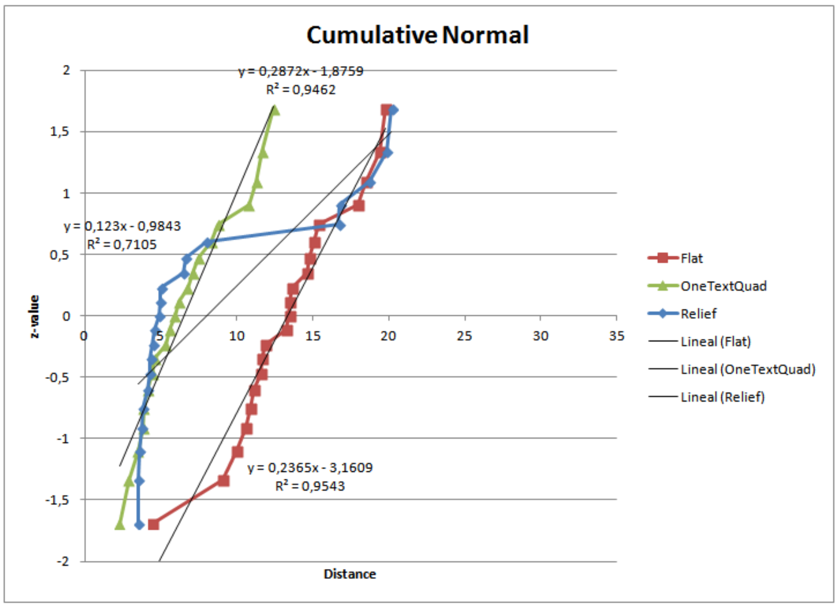


Figure 5.13: Cumulative normal: Camera 3, density 95

configurations. The predicted distances for each configuration are shown in the tables below 5.4 5.5 5.6 5.7.

Percentage	z	OneTextQuad	Flat	Relief
50%	0	23,62	13,44	26,71
60%	0,253	25,00	14,25	26,06
70%	0,524	26,46	15,12	29,50
80%	0,842	28,18	16,13	31,19
90%	1,282	30,57	17,54	33,53
99%	2,326	36,23	20,89	39,10

Table 5.4: Predicted distance for camera 1 density 7

Percentage	z	OneTextQuad	Flat	Relief
50%	0	10,95	9,59	6,63
60%	0,253	11,68	10,61	8,61
70%	0,524	12,46	11,70	10,73
80%	0,842	13,38	12,98	13,21
90%	1,282	14,65	14,75	16,66
99%	2,326	17,67	18,95	24,83

Table 5.5: Predicted distance for camera 3 density 7

Percentage	z	OneTextQuad	Flat	Relief
50%	0	15,21	13,95	13,65
60%	0,253	16,22	14,75	14,29
70%	0,524	17,29	15,59	14,98
80%	0,842	18,55	16,58	15,79
90%	1,282	20,30	17,96	16,90
99%	2,326	24,45	21,22	19,55

Table 5.6: Predicted distance for camera 3 density 95

Percentage	z	OneTextQuad	Flat	Relief
50%	0	13,37	6,53	8,00
60%	0,253	14,44	7,41	10,06
70%	0,524	15,58	8,36	12,27
80%	0,842	16,92	9,46	14,84
90%	1,282	18,78	10,99	18,42
99%	2,326	23,20	14,63	26,92

Table 5.7: Predicted distance for camera 1 density 95

5.4 Discussion

From the statistical analysis we can conclude that *One Textured Quad Impostors* are the most detectable (artifacts due this technique are quickly detected, which means that are identified at lager distances). This is reasonable since the texture resolution is crucial in this technique, thus, from a certain distance image pixels become too evident (as users commented after the experiment). On the other side, *Relief Impostors* are the most difficult render technique to detect among the tested, with the closest distances to the camera and the highest miss rate of response. In the middle point are *Flat Impostors*, whose artifacts are less noticeable than *One Textured Quad* but more evident than *Relief*.

These results lead us to the conclusion that the representation with best results is *Relief Impostors*, so it can be used for the widest range of distances. However, as we have mentioned in previous chapter, this technique implies large cost per fragment. So using *Relief Impostors* for the overall distance ranges will drop considerably the performance. On the other hand, *Flat Impostors* can be used as a more efficient method in further away distances without a loss of image quality. Finally, since *One Textured Quad Impostors* are the cheapest ones in terms of rendering performance, this representation can be used in the furthest part of the scene.

To summarize, we have observed that each one of the distance thresholds will depend mainly on the type of view used. Additionally, even though the density of agents used to simulate the crowd interacts with the impostor type, it does not affect the mean distance of each representation.

Chapter 6

Conclusions and Future Work

We will now present the conclusions reached during our work. Finally, to close this chapter we propose some future work.

6.1 Conclusions

The main problem of realistic crowd simulation relies on achieving real time performance while using hundreds or thousands of agents. Usually, increasing the detail on geometry models used leads to a drop in the number of agents the system is able to render. Thus, image-based impostors are a good alternative since they substitute high-detailed geometry by a few textured polygons.

After studying the state of the art of crowd rendering and knowing its limitations, we have notice that per-joint imaged based impostors can improve considerably rendering performance while simulating large crowds, without affecting significantly the visual quality. Specially, we have focused on two recent techniques that introduced the concept of per-joint impostor techniques: *Relief* [3] and *Flat Impostors* [1]. Furthermore, we have studied the state of the art on the perceptual evaluation of the different factors in crowd simulation. That led us to the conclusion that a perceptual experiment is essential in order to achieve the best results both in visual quality and performance, while using impostor representations for crowd rendering.

We have first developed a module of software for impostor generation and rendering included in a prototyping and development crowd simulation framework *CAVAST*. This module provides basic tools for easing the impostor creation, and at the same time integrates the impostor rendering with an already developed crowd simulation functionalities. We have demonstrate the simplicity of adding a new impostor type to the system by implementing an example: *One Textured Quad Impostors* [20].

After studying the main issues of both per-joint image-based impostor techniques, we have presented a collection of improvements in order to palliate their major limitations. The most prominent optimization for the two methods is the introduction of levels of detail within the same representation. In this way, most relevant parts of the character such as the head, can be enhanced without affecting substantially the general performance and the memory consumption. In the case of *Flat Impostors*, we have also adapted the method of computing the opacity masks for joint boundaries to use approximate actual human values for joints bending. Moreover, other optimization techniques such as character variety by color modulation or pre-computing the lighting have been included into the system, while improving the overall performance and contributing in visual variance.

In order to deeply analyze the two studied impostor techniques (*Relief* and *Flat Impostors*), we have carried out a rigorous perceptual experiment. We have also included to the comparison the *One Textured Quad Impostor* technique exposed as an example. Throughout this experiment, we have determined the optimum distance for rendering each one of the three representations without a loss of image quality in a large crowd simulation. We have also obtained which are the most noticeable artifacts of each representation, to continue working in them.

6.2 Future Work

As discussed in the previous section, there are some aspects and limitations we would like to work on in our future research.

We would like to add the implementation of other kind of impostors in our framework, in order to provide a complete and varied basis to start working with, while simulating large crowds with impostor techniques. At the same time, this could be used to carried out a general experiment comparing other impostor representation techniques.

The user response measurement in the study would be more accurate if the distance at which the geometry representation switches to the impostor would vary in screen space units, rather than world space.

Some impostor artifacts such as cracks or joint overlapping can be masked by choosing the correct amount of neighboring geometry influenced by the joint. These artifacts are more noticeable near articulations with high level of bending. Thus, this can be improved by selecting a different amount of neighboring vertices per joint while generating impostors.

Finally, we would like to improve crowd simulation by adding realistic agent behaviors. Furthermore, our simulation only uses one walking animation, thus we would like to extend the simulation using several motion cycles: walking at different speeds, turning to the left and right, idle motion, etc.

Bibliography

[1] A. Beacco, C. Andújar, N. Pelechano, and B. Spanlang. Efficient rendering of animated characters through optimized per-joint impostors. *Journal of Computer Animation and Virtual Worlds*, 23(2):33–47, 2012.

[2] A. Beacco and N. Pelechano. Cavast the crowd animation, visualization, and simulation testbed. In Pere-Pau Vázquez and Adolfo Muñoz, editors, *CEIG - Spanish Computer Graphics Conference*, 2014.

[3] A. Beacco, B. Spanlang, C. Andújar, and N. Pelechano. A flexible approach for output-sensitive rendering of animated characters. *Computer Graphics Forum*, 30, 2011.

[4] Cal3d. 3d character animation library.

[5] James A. Ferwerda. Psychophysics 101: How to run perception experiments in computer graphics. In *ACM SIGGRAPH 2008 Classes*, SIGGRAPH '08, pages 87:1–87:60, New York, NY, USA, 2008. ACM.

[6] J. Hamill, R. McDonnell, S. Dobbyn, and C. O'Sullivan. Perceptual evaluation of impostor representations for virtual humans and buildings. *Computer Graphics Forum*, 24(3):623–633, 2005.

[7] Irving P Herman. *Physics of the Human Body.* Biological and Medical Physics, Biomedical Engineering. Springer, Berlin, Heidelberg, 2007.

[8] Adrian Jarabo, Tom Van Eyck, Veronica Sundstedt, Kavita Bala, Diego Gutierrez, and Carol O'Sullivan. Crowd light: Evaluating the perceived fidelity of illuminated dynamic scenes. In *Computer Graphics Forum*, volume 31, pages 565–574. Wiley Online Library, 2012.

[9] Kenton R . Kaufman and Kai-Nan An. *Biomechanics: Principles and Applications*, volume Chapter 3: Joint-Articulating Surface Motion. CRC Press 2002, 2002.

[10] Michéal Larkin and Carol O'Sullivan. Perception of simplification artifacts for animated characters. In Rachel McDonnell, Simon J. Thorpe, Stephen N. Spencer, Diego Gutierrez, and Martin Giese, editors, *APGV*, pages 93–100. ACM, 2011.

[11] R. McDonnell, M. Larkin, S. Dobbyn, S. Collins, and C. O'Sullivan. Clone attack! perception of crowd variety. *ACM Trans. Graph.*, 27(3):26:1–26:8, August 2008.

[12] R. McDonnell, M. Larkin, B. Hernández, I. Rudomín, and C. O'Sullivan. Eye-catching crowds: saliency based selective variation. *ACM Trans. Graph.*, pages –1–1, 2009.

[13] Rachel McDonnell, Simon Dobbyn, and Carol O'Sullivan. Lod human representations: A comparative study. In Switzerland Lausanne, editor, *International Workshop on Crowd Simulation (V-CROWDS)*, pages 101–115, 2005.

[14] M. Oliveira, G. Bishop, and D. McAllister. Relief texture mapping. In *SIGGRAPH '00: Proc. of the 27th annual conference on Computer graphics and interactive techniques*, pages 359–368, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[15] J. Pettré, P. De Heras Ciechomski, J. Maïm, B. Yersin, J. Laumond, and D. Thalmann. Real-time navigating crowds: scalable simulation and rendering: Research articles. *Comput. Animat. Virtual Worlds*, 17(3-4):445–455, 2006.

[16] D. Pratt, S. Pratt, P. Barham, R. Barker, M. Waldrop, J. Ehlert, and C. Chrislip. Humans in large-scale, networked virtual environments. *Presence*, 6(5):547–564, 1997.

[17] Martin Pražák and Carol O'Sullivan. Perceiving human motion variety. In *Proceedings of the ACM SIGGRAPH Symposium on Applied Perception in Graphics and Visualization*, APGV '11, pages 87–92, New York, NY, USA, 2011. ACM.

[18] Eric Risser, Musawir Shah, and Sumanta N. Pattanaik. Faster relief mapping using the secant method. *J. Graphics Tools*, 12(3):17–24, 2007.

[19] B. Spanlang. Halca hardware acclerated library for character animation. Technical report, EVENT Lab. Universitat de barcelona, 2009.

[20] F. Tecchia and Y. Chrysanthou. Real-time rendering of densely populated urban environments. In *Proc. of the Eurographics Workshop on Rendering Techniques 2000*, pages 83–88, London, UK, 2000. Springer-Verlag.

[21] Franco Tecchia, Celine Loscos, and Yiorgos Chrysanthou. Visualizing crowds in real-time, 2002.

[22] Michael Wimmer, Peter Wonka, and François Sillion. Point-based impostors for real-time visualization. In Steven J. Gortler and Karol

Myszkowski, editors, *Rendering Techniques 2001 (Proceedings Eurographics Workshop on Rendering)*, pages 163–176. Eurographics, Springer-Verlag, June 2001.

Annex

Perceptual evaluation forms for Image-Base Rendering for Crowds

November 2014

Study Information

The purpose of this study is to evaluate the user perception of the visual artifacts resulting from some render methods applied to a crowd of agents.

This user study is part of the Master Thesis *Improving Image-Base Rendering for Crowds and Perceptual Evaluation*. Contact person:

- Maria Izquierdo, e-mail: *maria.izqrdo@gmail.com*

This experiment has no risk for people. The participants have the right to withdraw from the study at any time without prejudice and without providing a reason. All data collected from the users is confidential and it will be processed anonymously. There will be no economical compensation.

Previous information

Age:

Gender: ☐ Male ☐ Female

Is your vision impaired?

☐ Yes ☐ No

If yes, are you currently wearing corrective glasses or lenses?

☐ Yes ☐ No

With what frequency do you use the computer?

Less than an hour per day	☐	☐	☐	☐	☐	More than 6 hours per day

Do you have any experience in working on computer graphics? Or playing 3D videogames?

I do not know what it means	☐	☐	☐	☐	☐	More than 3 hours per day

User satisfaction

Did you understand perfectly the tasks to perform and the procedure?

I did not know what to do	☐	☐	☐	☐	☐	I perfectly understood

Did you find the tasks easy to do?

Very hard	☐	☐	☐	☐	☐	Very easy

Do you have any comments or suggestions?

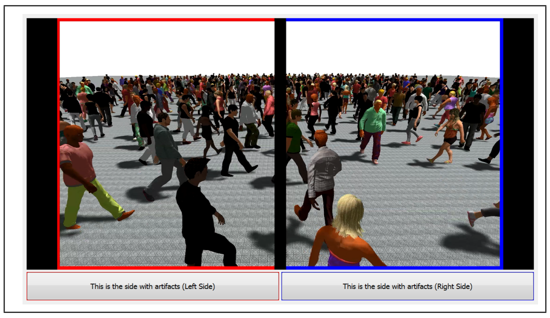
Instructions

A set of short videos are presented to the user. All of them show a crowd of agents, animated and simulated.

Each video is vertically divided through the middle point by a black line, splitting the crowd in two parts. The agents situated on one side are rendered using a method that does not produce artifacts. However, the method used on the other side usually produces more visual artifacts. Which method is used in each part is randomly chosen.

The user's purpose is to determine **which side is the one with the agents showing visual artifacts.**

Under the video player, there are two buttons corresponding to the left and to the right side of the crowd. During a video reproduction, when the user notices any difference in the rendering of the agents of a determined area, he has to indicate which side has visual artifacts pressing the correspondent button (by pressing the colored key or through using the mouse). At the end of the study, the user has to introduce what kind of artifacts he has detected throughout the whole study (what were the differences between the two sides of the videos that have led him to such a conclusion).


This is the side with artifacts (Left Side)
This is the side with artifacts (Right Side)
(a) Application snapshot

(b) Keyboard

At the beginning of the experiment, the first video shows a crowd rendered without artifacts on both sides, just to become familiar with the environment. During the test, a plain white image is showed in between videos, to ease eye strain.

Rendering artifacts on one of the two sides will become more apparent as the video goes forward. So do not worry if during the first few seconds of each video you notice no difference at all in rendering quality. Be patient and once you are sure which side suffers from the artifacts, please press the corresponding button as fast as you can. If the end of the video is reached, you have to choose which side you think is the one with visual artifacts.

It is important to keep in mind that the goal of the study is to find out artifacts of the agents' rendering method. Artifacts related to any other kind of errors, such as those caused by bad visualization (rendering, lighting, shadows...) or from the animation and the simulation (as foot sliding), are not part of the objective of this study and have to be ignored.

The approximated duration of this test is about 20 minutes.