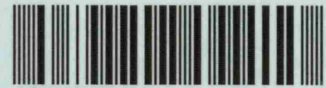
280907652X

REFERENCE ONLY

UNIVERSITY OF LONDON THESIS

Degree PhD Year 2007 Name of Author TECCHIA Franco

COPYRIGHT

This is a thesis accepted for a Higher Degree of the University of London. It is an unpublished typescript and the copyright is held by the author. All persons consulting the thesis must read and abide by the Copyright Declaration below.

COPYRIGHT DECLARATION

I recognise that the copyright of the above-described thesis rests with the author and that no quotation from it or information derived from it may be published without the prior written consent of the author.

LOAN

Theses may not be lent to individuals, but the University Library may lend a copy to approved libraries within the United Kingdom, for consultation solely on the premises of those libraries. Application should be made to: The Theses Section, University of London Library, Senate House, Malet Street, London WC1E 7HU.

REPRODUCTION

University of London theses may not be reproduced without explicit written permission from the University of London Library. Enquiries should be addressed to the Theses Section of the Library. Regulations concerning reproduction vary according to the date of acceptance of the thesis and are listed below as guidelines.

A. Before 1962. Permission granted only upon the prior written consent of the author. (The University Library will provide addresses where possible).

B. 1962 - 1974. In many cases the author has agreed to permit copying upon completion of a Copyright Declaration.

C. 1975 - 1988. Most theses may be copied upon completion of a Copyright Declaration.

D. 1989 onwards. Most theses may be copied.

This thesis comes within category D.

☐ This copy has been deposited in the Library of UCL

☐ This copy has been deposited in the University of London Library, Senate House, Malet Street, London WC1E 7HU.

Bound By
Blissett Bookbinders
020 8992 3965
www.blissetts.com

An image-based approach to the rendering of crowds in real-time

Franco Tecchia

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
of the
University of London.

Department of Computer Science
University College London
October 2006

UMI Number: U594430

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

UMI U594430
Published by ProQuest LLC 2013. Copyright in the Dissertation held by the Author.
Microform Edition © ProQuest LLC.
All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

DECLARATION

I, Franco Tecchia, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

ACKNOWLEDGMENTS

Firstly, I would like to thank Prof. Mel Slater, my supervisor, and Dr. Yiorgos Chrysanthou, my second supervisor, for all of their patience, help, and advice. This work would not have been possible without their continuous guidance and encouragement.

Secondly, I would like to thank Celine Loscos, that shared with me a large amount of time and effort on this exciting research.

Finally, I would like to thank all my very precious friends. I would not have been able to get through all this without their continuous support.

It has been an incredible and formative experience; this work is dedicated to all of you.

"*CROWD*"

"A large group of people who have gathered together"
Oxford English Dictionary

"A number of persons congregated or collected into a close body without order"
Webster's Revised Unabridged Dictionary

"A large number of things or people considered together"
WordNet 2.0 - Priceton University

"A group of people with something in common"
Encarta® World English Dictionary, North American Edition

"The lower orders of people; the populace; the vulgar; the rabble; the mob."
Webster's Revised Unabridged Dictionary

ABSTRACT

The wide use of computer graphics in games, entertainment, medical, architectural and cultural applications, has led it to becoming a prevalent area of research. Games and entertainment in general have become one of the driving forces of the real-time computer graphics industry, bringing reasonably realistic, complex and appealing virtual worlds to the mass-market. At the current stage of technology, an user can interactively navigate through complex, polygon-based scenes rendered with sophisticated lighting, at times interacting with AI-based synthetic characters or *virtual humans*. As the size and complexity of the environments continuously increase, there is a growing need to add common real-life phenomenons such as the presence of crowds. Rendering highly populated urban environments requires a good synthesis of two partially overlapping problems: the management of vast and detailed environments, and the visualisation of large-scale crowds. While in the past a large amount of research has gone into the investigation of optimisation strategies and speed-up methods dedicated to large and static polygonal models, real-time visualisation of animated crowds poses novel challenges due to the computational power needed to visualize a multitude of animated characters; scenarios where thousands of characters are on-screen simultaneously can easily lead to polygon budgets exceeding millions of triangles, that are hardly possible to render at interactive frame-rates even on the most powerful graphics technology available today. The present thesis extensively investigates this topic, and proposes the usage of an image-based data representation in order to speed-up rendering of animated characters so to achieve interactive frame-rates even when crowds composed by thousands of individuals are on-screen. We advance over the state of the art in this field introducing a novel form of impostor rendering techniques for animated crowds, and presenting methods that exploit this novel form of representation to handle also advanced rendering effects such as crowd lighting and shadowing; we also discuss the important aspects of compatibility of our new method with existing polygonal based scene-graphs architectures. The results are showing that real-time crowd rendering is an application scenario where the introduction of Image Based Rendering methods can truly be beneficial, making possible the rendering of crowds composed by thousands of individual in real time in any existing rendering software framework even on commodity hardware.

Table of Contents

Chapter 1 - Introduction **13**

1.1 Crowds and Virtual Environments 14

1.2 Computer Graphics and Human-Computer Interaction 16

1.3 The challenges of real-time rendering 16

1.4 Real-time crowd rendering 18

1.5 Motivation and scope of this work 18

1.6 Novel contributions 19

1.7 Organization of this thesis 21

Chapter 2 - Background and previous work **22**

2.1 Real-time rendering using polygons 23

2.2 Acceleration techniques for polygonal rendering 24

2.2.1 Visibility culling 25

2.2.2 Level of Detail management 27

2.3 Alternative approaches to polygonal rendering 29

2.4 Point based rendering 30

2.5 Image based rendering 32

2.5.1 Impostors rendering 33

2.6 Drawbacks of the impostors approach 36

2.7 Real time rendering of crowds 37

2.7.1 Modeling the human body 37

2.7.2 Rendering crowds using polygons 38

2.7.3 Rendering crowd using points 40

2.7.4 Rendering crowd using impostors 41

2.8 Summary 43

Chapter 3 - Crowd rendering using impostors **44**

3.1 Rendering human - like characters using impostors 45

3.2 Pros and cons of precomputed impostors 46

3.3 Building the impostors database 47

3.4 Impostors run - time rendering 49

3.5 Impostors and billboards 50

3.6 Visual artifacts caused by impostors 51

3.7 Optimal placement of the impostor plane 53
3.8 Colour modulation of the impostors 55
3.9 Interactive impostor lighting 58
3.10 Approximate dynamic lighting 59
3.11 Impostors per-pixel lighting 62
3.12 Per-pixel lighting equation 63
3.13 Summary 66
Chapter 4 - Placing crowds in Virtual Environments 68
4.1 Using animated impostors in a polygonal scenario 69
4.2 Illumination of a crowded scene 70
4.3 Ambient lighting for a crowded Virtual Environment 71
4.4 Shadowing 73
4.5 Shadows in populated Virtual Environment 75
4.6 Casting shadows on the surrounding environment 76
4.6.1 Using fake shadows 76
4.6.2 Using shadowmaps 77
4.7 Shadowing effects of the environment on the crowd 79
4.8 Visibility computation 81
4.9 Beyond rendering 83
4.9.1 Path finding and obstacles avoidance 83
4.9.2 Approximate and fast collision detection for crowd navigation 85
4.9.3 Height map generation 85
4.9.4 Collision detection and avoidance 86
4.10 Summary 88
Chapter 5 - The crowd rendering system 89
5.1 General guidelines 90
5.2 Creation of the impostor database 91
5.3 Memory management 93
5.3.1 OpenGL memory management 93
5.3.2 Using OpenGL texture compression 95
5.3.3 Tight packing of the impostor images 96
5.4 Organisation of the texture working set 97

5.5 Efficient run-time impostors management........ 99
5.6 Taking distance into consideration........ 101
5.7 Visibility computation........ 104
5.7.1 Occluder selection and building of the occlusion tree........ 106
5.8 Approximate shadowing of animated impostors........ 107
5.8.1 Shadow texture coordinate computation........ 109
5.9 Summary........ 110
Chapter 6 - Performance analysis........ 111
6.1 Thesis Objectives Revisited........ 112
6.2 Methods of Assessment........ 112
6.3 Scenarios and characters models........ 113
6.4 Test systems........ 115
6.5 Factors influencing impostor rendering performance........ 116
6.6 Test1: Speed and scalability of the basic approach........ 117
6.7 Test 2: The multi-pass colouring approach........ 119
6.8 Test 3: Hybrid polygonal - impostor rendering........ 120
6.9 Test 4: Speed comparison against LODs rendering........ 125
6.10 Test 5: Increasing the number of samples........ 128
6.11 Advanced impostors effects........ 130
6.11.1 Test 6: Approximate dynamic lighting........ 130
6.11.2 Test 7: Per-pixel impostor lighting........ 131
6.11.3 Test 8: Ambient lighting........ 133
6.11.4 Test 9: Shadowing effects........ 134
6.11.5 Test 10: Occlusion culling performed on crowd........ 136
6.12 Discussion........ 137
6.13 Summary........ 138
Chapter 7 - Conclusions........ 139
7.1 Summary........ 140
7.2 Review of aims........ 140
7.3 Summary of contributions........ 141
7.4 Implications of this work........ 142
7.5 Critical review........ 142

7.6 Future directions........143
7.6.1 Vertex and pixel shaders........143
7.6.2 Dynamical impostors generation........144
7.6.3 Impostors clustering........144
7.7 Conclusions........145
References........146

List of Figures

Figure 1.1 - Crowd are a common phenomenon in many real life situations. 14
Figure 1.2 - A CG generated army from 'The Fellowship of the Ring' (New Line Cinema – 2001). 15
Figure 1.3 - The presence of a crowd increases the realism of urban environments. 18
Figure 1.4 - Using crowds in virtual urban environments. 19
Figure 1.5 - High-performance crowd rendering using textured impostors. 20
Figure 2.1 - A scene from the movie 'Star Wars Episode II: attack of the clones' (Lucasfilms – 2002). 22
Figure 2.2 - Representing objects geometry using polygons. 23
Figure 2.3 - A simple rendering pipeline. 23
Figure 2.4 - The Visibility Culling process. 26
Figure 2.5 - Multiple LODs can be used to approximate the shape of distant objects. 27
Figure 2.6 - Progressive Meshes. 28
Figure 2.7 - Point based rendering at various resolutions of a generic 3D object. 31
Figure 2.8 - Warping of panoramic images to compensate perspective distortion. 32
Figure 2.9 - Embedding images in a polygonal scenario: the impostor. 34
Figure 2.10 - A complex object is replaced by a single, texturised, quadrilateral 34
Figure 2.11 - Impostors warping using image layers. 35
Figure 2.12 - Incorrect visibility due to impostor lack of depth information. 36
Figure 2.13 - Reproducing the human shape needs a large polygon budget. 37
Figure 2.14 - Simplification of polygonal characters. 38
Figure 2.15 - Polygonal crowd used to populate a 3D model. 39
Figure 2.16 – Rendering an army using polygons. 40
Figure 2.17 - Rendering animated characters using points 40
Figure 2.18 - Crowd rendering using points. 41
Figure 2.19 - Using impostors for crowd rendering 42
Figure 2.20 - Aubel's dynamic impostors performing a 'Mexican wave'. 42
Figure 3.1 - A scene from the movie 'Troy' (Warner Bros. - 2004). 44
Figure 3.2 - Impostors can be used to replace the complex geometry of a human character. 45
Figure 3.3 - A key-framed walking sequence can be captured using multiple impostors. 46
Figure 3.4 - Taking a discrete set of images of a polygonal character. 48
Figure 3.5 - Conceptualization of the impostors run-time rendering pipeline. 50
Figure 3.6 - Billboards (a) and impostors (b) are conceptually different. 51
Figure 3.7 - Projecting points on the impostor plane 52
Figure 3.8 - Different choices for the impostor projection plane 54
Figure 3.9 - Simulating variety using colour modulation. 56
Figure 3.10 - Fine-control of the impostor colours. 57
Figure 3.11 - Effect of the random colouring of different body parts. 58
Figure 3.12 - Global illumination effects can increase the realism of a scene. 59

Figure 3.13 - Simulating the effects of a (white) local light source. 61
Figure 3.14 - Simulating the effects of multiple coloured light sources. 61
Figure 3.15 - Storing per-pixel normal information in the impostors image samples. 63
Figure 3.16 - The standard OpenGL lighting equation. 64
Figure 3.17 - Impostor per-pixel lighting. 65
Figure 3.18 - Final effects of per-pixel lighting on an impostor-based crowds. 66
Figure 4.1 - An image from the movie 'Shrek' (Dreamwork – 2004). 68
Figure 4.2 - High-performance crowd rendering using textured impostors.. 69
Figure 4.3 - Global illumination effects on crowd. 71
Figure 4.4 - An urban scenario and the resulting light intensity map. 72
Figure 4.5 - The light intensity modulation at work on the impostors 73
Figure 4.6 - The fake shadows technique applied to impostor rendering. 77
Figure 4.7 - An early version of our rendering system: crowd casting (fake) shadows on the scenario. 79
Figure 4.8 - Simulating shadows on the impostors. 80
Figure 4.9 - Example of the visual produced by our shadowing algorithm. 81
Figure 4.10 - Similarly to the static geometry, buildings can be very effective occluders. 82
Figure 4.11 – Visibility Culling: our system detects occlusion caused by building on the crowd 82
Figure 4.12 - Crowd navigating in the web of streets of a city. 84
Figure 4.13 - A simple urban model and its corresponding height-map (below, in grey scale). 86
Figure 4.14 - A simple collision avoidance strategy using cells elevation. 87
Figure 5.1 - A scene from the movie 'The Fellowship of the Ring' (New Line Cinema – 2001) 89
Figure 5.2 – Two examples of urban environments rendered in real-time in our system. 90
Figure 5.3 – Using impostors can be subdivided in two macro phases: 91
Figure 5.4 - Storing impostors data. 93
Figure 5.5 - A simplification of the rendering pipeline. 94
Figure 5.6 - Packing impostor together. 97
Figure 5.7 - Some elevations of the camera needs a larger variety of samples to be rendered. 98
Figure 5.8 - The Lookup, Selection and Rendering phases of the crowd rendering pipeline. 100
Figure 5.9 - The three stages of memory involved in the rendering process. 101
Figure 5.10 - Using less samples for the distant characters. 102
Figure 5.11 - Subdividing textures to optimise memory management. 103
Figure 5.12 - The complete run-time crowd rendering pipeline. 104
Figure 5.13- Subdivision of the scenarios' occluders. 105
Figure 5.14 - Computing visibility. 107
Figure 5.15 - Height Map (a) and Shadow Volume Map (b) computed for the same scenario. 108
Figure 5.16 - Filtering the values of adjacent cells to produce smooth shadow transitions. 109
Figure 6.1 - A scene from the movie 'Troy' (Warner Bros. - 2004) 111
Figure 6.2 - Test Scenario 1: a simple urban-like scenario (3,216 triangles). 113
Figure 6.3 - Test Scenario 2: the Garibaldi Square model (43,866 triangles). 113

Figure 6.4 - Test Scenario 3: the town model (41,260 triangles). 114
Figure 6.5 - The character models used in the tests. 115
Figure 6.6 - Representing the impostor rendering pipeline as a sequence of 4 functional blocks. 116
Figure 6.7 - Rendering a crowd with the basic impostor algorithm 118
Figure 6.8 - Average time to renderer a frame against the number of impostors. 118
Figure 6.9 - Introducing variety using colour modulation. 119
Figure 6.10 - Average time to renderer a frame against the number of impostors. 120
Figure 6.11 - Mixing polygonal characters and impostors. 121
Figure 6.12 - Mixing polygonal characters and impostors (Test Scenario 1). 121
Figure 6.13 - The camera trajectory used for Test 3. 122
Figure 6.14 - Measuring how the introduction of impostors affects the rendering speed of large crowds. 123
Figure 6.15 - Mixing polygonal characters and impostors. 124
Figure 6.16 - Measuring the benefit of impostors when they replace simpler polygonal models. 125
Figure 6.17 - Using multiple LODs for crowd visualisation. 126
Figure 6.18 - Using static polygonal LODs to render a crowd. 126
Figure 6.19 - Populating Test Scenario 1 with a LODs - based polygonal crowd . 127
Figure 6.20 - Measuring rendering time in the LODs - based system. 128
Figure 6.21 - Measuring rendering speed for different amount of impostor data (Test System Low). 129
Figure 6.22 - Measuring rendering speed for different amount of impostor data (Test System High). 130
Figure 6.23 - Dynamic illumination of a crowd using a variety of local lights configurations. 131
Figure 6.24 - Testing dynamic lighting. 131
Figure 6.25 - Impostors per-pixel lighting. Testing speed and scalability of the method. 132
Figure 6.26 - Testing per-pixel lighting. 132
Figure 6.27 - Taking in account ambient light intensity in the Garibaldi Square scenario. 133
Figure 6.28 - Testing ambient lighting. 134
Figure 6.29 - Testing speed and scalability of our shadow methods in the Test Scenario 3. 135
Figure 6.30 - Testing shadowing effects. 135
Figure 6.31 - Performing real-time occlusion culling on the crowd (Test Scenario 3). 136
Figure 6.32 - Testing crowd occlusion culling. 137
Figure 7.1 - A scene from the movie 'Star Wars Episode II: attack of the clones' (Lucasfilms – 2002). 139

CHAPTER 1 - INTRODUCTION

Computer graphics is concerned with rendering a visual representation of a synthetic 3D environment on a display, a task that has attracted a great deal of scientific research in the past 50 years. A commonly accepted taxonomy differentiates techniques and algorithms for off-line content creation, where the time needed to create a single computer graphics (CG) image is relatively unimportant, and for on-line content creation, where a sequence of images needs to be generated quickly enough in order to guaranty a target frame-rate. The ability to render highly populated scenes using computer generated crowds can be desirable for both on-line and off-line content creation. In particular, Virtual Environments applications (and other real-time applications such as games) may use crowds to breathe life into otherwise static scenes, with the goal of enhancing the overall believability of the scenario. Due to the large amount of computational resources needed to perform this task, the simulation of crowded scenes is best accomplished through the use of special dedicated techniques, that only recently started to be investigated in the context of real-time computer graphics. A major goal of this thesis is to show that real-time crowd rendering is best supported through a particular approach to computer graphics known as image-based rendering. It will be shown that thousands of virtual characters can be made to populate and move through a virtual environment in real-time, and that the approach is flexible enough to support many facilities such as illumination and shadowing previously considered to be incompatible with real-time image based graphics.

1.1 Crowds and Virtual Environments

Crowds are a very common phenomenon in our everyday life. We can see crowds when using many forms of public transport, attending sporting events, going to the cinema or at a concert or just shopping around. We spend a large part of our life as members of some crowd, as this is an intrinsic characteristic of any society.

Figure 1.1 - Crowds are a common phenomenon in many real life situations.

With crowds being such a familiar element of everyday life, it comes with no surprise that the ability to simulate and visualise large crowds is interesting for both off-line and on-line Computer Graphics research. The current trend in Cinema industry acknowledges this importance, and more and more feature films present situations were crowds are important, to such an extent that often they become an essential element of a film plot: the clash between large armies in films such as Troy, or the reproduction of historical crowd gatherings as in the Gladiator's Colosseum, are some good examples of situations where their presence plays an important role. In some cases crowds can even become the distinguishing element of a movie: the battles between massive armies composed by thousands of animated characters in the Lord

of the Rings Trilogy are amongst the most awe-inspiring moments of the movie. With the costs of modern film production, scenes involving thousands of moving individuals have become prohibitively expensive without the development and use of computer-generated crowds. Also, dangerous situations, like for example the sinking scene of the movie Titanic, would have been impossible or too dangerous to film using a multitude of real actors.

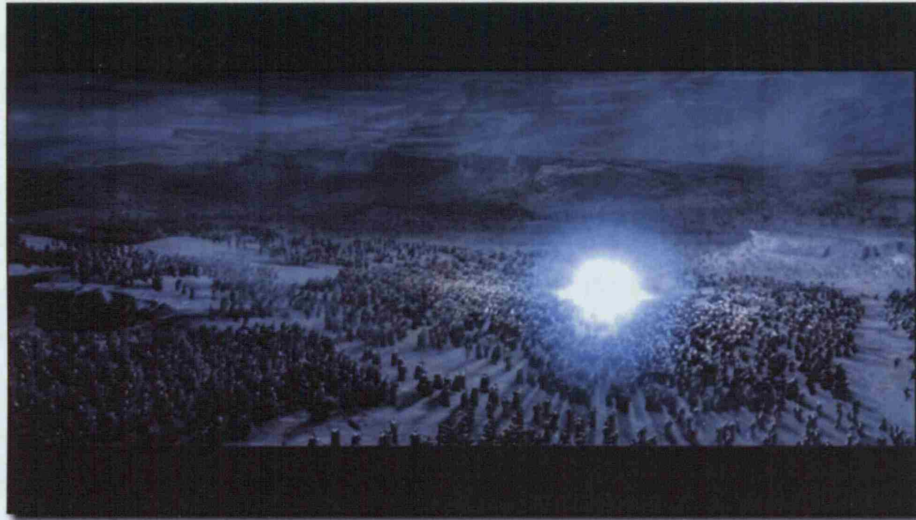

Figure 1.2 - A CG generated army from 'The Fellowship of the Ring' (New Line Cinema – 2001).

Even if increasingly frequently used in the Cinema industry, the importance of crowds in CG is not at all confined to the world of off-line content generation. Many Virtual Environment applications could benefit by their presence. As the size and complexity of the 3D scenarios presented in these applications continuously increases, there is a growing need to populate them with more than just a few interacting characters. For instance, when simulating complex urban scenarios, the believability of any Virtual Environment application is diminished by the general absence of crowds, and models of large cities cannot appear realistic if they are not populated by a large number of individuals. Unfortunately, visualising large crowds using computer graphics is not an easy task. Programmers of any VE application need to deal with having limited rendering resources available to compose each frame, and with many hundreds or thousands of potentially visible individuals forming a crowd, many traditional optimisation techniques cannot help achieving interactive frame-rates. As a consequence, new and dedicated approaches to the rendering, animation and behaviour control of the crowds are needed.

1.2 Computer Graphics and Human-Computer Interaction

Computer graphics was born from the use of CRT and pen devices early in the history of computers; many Computer Graphics techniques date from Sutherland's Sketchpad PhD. thesis (1963) that essentially marked the beginning of computer graphics as a discipline [Suth63]. Human-Computer Interaction, intended as the discipline concerned with the design, evaluation and implementation of interactive computing systems for human use, arose as a field with intertwined roots in operating systems, human factors, ergonomics, industrial engineering, cognitive psychology, and the systems part of computer science [Cox99]. Some of the research performed in Computer Graphics combined with the HCI natural interest in interactive graphics (e.g., how to manipulate solid models in a CAD/CAM system), generated the field of Virtual Environments, where concepts and procedures coming from both fields are mixed together. In 1965 Sutherland stated that one of the biggest challenges in Computer Graphics was to allow the user to be able to look at the display device as a window into a virtual world and to make such virtual world appear realistic. Thanks to the advances in computer algorithms, processing power, memory, and graphics display systems, this goal is getting closer every year. Modern Virtual Environments research focuses on accomplishing and expanding the initial Sutherland vision with the goal to fully immerse people into computer generated environments that they can interact with, and expose them to situations that could be impossible or very difficult to replicate in the real world.

1.3 The challenges of real-time rendering

While off-line CG is focused on the quality, the realism, and the detail of the generated images, on-line CG pursues mainly the goal of interactivity: images must be generated in a rapid succession to represent the ongoing state of a synthetic and evolving scenario that the user can interact with, reason why it is often called *interactive graphics*. Time is here the critical resource, and the accurate simulation of visual phenomena, albeit always desirable, becomes less important than rendering speed. Achieving a trade-off between visual fidelity and computation time becomes important and, as many algorithmic efforts are spent to achieve the maximum possible image quality respecting the tight time boundaries of two consecutive frames, the word "realism" assumes new connotations. The computer-based generation of images under this time constraints is called *Real-Time Computer Graphics*. Virtual Environments applications are an example of this kind of graphics, where the essential

requisite for the rendering activity is to take place at an "interactive frame-rate". A widely accepted measure of an application interactivity is a minimal image generation rate of about 20 Hz: in these conditions there is at most 50 milliseconds of time to fully generate a single frame of a rendering sequence. The modern rules of interactivity push these limits even further, and frame rates of 60 Hz (or more) are not uncommon in these days.

Due to its robustness and relatively simple hardware implementation, polygonal rendering has become the most widely adopted approach to Virtual Environments visualisation: to compose any element in the scene a multitude of basic graphics primitives such as point, lines and triangles are used. As the complexity and the realism of the presented scenarios continues to grow, the brute force approach of representing any visual object using a multitude of small triangles can become problematic. Noticeable computation delays appear between frames when the visual complexity reaches a critical threshold, decreasing the quality of the visualization and the ability of the user to interact with the application, and not even the most advanced hardware technology can offer enough raw speed to handle every possible scenario at interactive frame rates. This is why heuristics and methods to optimise the use of available rendering resources are a well-studied research topic, with literally hundreds of algorithms proposed to handle many different (and specific) situations. Traditional optimisation algorithms generally address the task of rendering large static models, as this is the most common example where the rendering activity can be rationalised; for example, visibility culling is generally an effective acceleration technique for the static part of urban scenes because of the intrinsic heavy occlusions that these environments present. Being frequently geared to large static scenarios, some optimisation approaches cannot be used for dynamic situations, making them unusable in some cases. Crowd rendering, a relatively recent topic in this context, is an example of a complex non-static rendering scenario where a multitude of animated entities navigate around the environment, a situation that VE developers may try to avoid due to the technical challenges that this presents. Their dynamic nature makes crowd rendering a very special activity, something that cannot be addressed in the same way that the rendering of large and static polygonal scenarios is carried out. Still crowds are very important: in spite of simulated virtual worlds appearing increasingly realistic, unless these complex synthetic worlds are populated in a similar way to what we commonly experience in our everyday life it is not really possible for the user to achieve the suspension of disbelief ideally required by many Virtual Environments applications.

1.4 Real-time crowd rendering

The introduction of crowds into virtual environments poses non-trivial technical challenges, as the restrictions imposed by the constraint of real-time rendering imposes severe limitations to the overall computation resources available for their visualisation. The main technical difficulty comes from the fact that the human body has an elaborate shape and a resulting complex polygonal mesh is usually needed to represent each individual. Situations where thousands of characters are on-screen simultaneously (Figures 1.3 and 1.4) can then easily lead to polygon budgets exceeding millions of triangles, making it difficult or impossible to render the scene in real-time without the use of some special, dedicated technique.

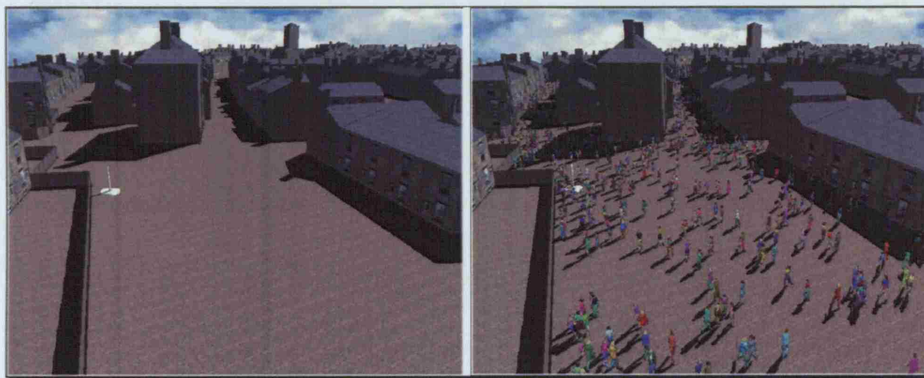

Figure 1.3 - The presence of a crowd increases the realism of urban environments.

We claim that from a general point of view, when dealing with the visualisation of large-scale crowds many traditional optimisation approaches are either not applicable or lead to non-optimal results. Radically different methods are better suited to provide the speed boost needed in order to achieve an interactive frame rate even with thousands of individuals in the scene.

1.5 Motivation and scope of this work

The research described in this thesis was motivated by the need to populate complex urban models used in several Virtual Environments applications with crowds composed of a very large number of animated characters, with the goal to enhance the overall realism of the resulting scene thanks to the simulation of a dynamic population of computer-managed individuals. The focus of this work is on the rendering aspects of the problem, where specific

techniques will be proposed to handle the otherwise overwhelming rendering needs of real-time crowd visualisation. In particular, the thesis proposes the use of an image-based representation of data in order to speed-up the rendering process, and it discusses the ability of such representation to handle additional graphical tasks such as lighting, shadowing and visibility computation, and to coexist with traditional polygonal rendering-based scene graphs.

Figure 1.4 - Using crowds in virtual urban environments.
The EU Project CREATE – A populated Garibaldi Square model (Nice-France).

The overall organisation of the work reported in this thesis was arranged to demonstrate that, using the right form of representation, crowd rendering is one of the situations where the introduction of IBR methods does not require a radical re-engineering of existing scene-graph frameworks, while at the same time can lead to substantial improvement in rendering speed compared to the use of polygonal rendering. Nevertheless a very similar image quality can be attained, and the same (or better) flexibility in terms of interactive lighting of the resulting scenario.

1.6 Novel contributions

The fundamental kernel of the present thesis is based on a specific type of Image Based Rendering technique, known under the name of *impostors rendering:* as described in detail in Chapter 2, an *impostor* is a 2D image embedded inside a 3D scenario to replace one or more geometrical objects (see Figure 1.5). Despite having many desirable qualities, the use of impostors for the specific task of crowd rendering was underdeveloped prior to the present work: impostors were normally used to replace non-animated geometry (very different with

the case of crowds). Also this technique was considered rather inflexible, as normally impostors cannot provide a fine control on the visual appearance of the object they represent. Finally, they were considered too memory consuming to represent a large variety of objects and, mostly for this reason, normally computed at run time (dynamic impostors) and only used in small numbers, as the generation process can be time consuming.

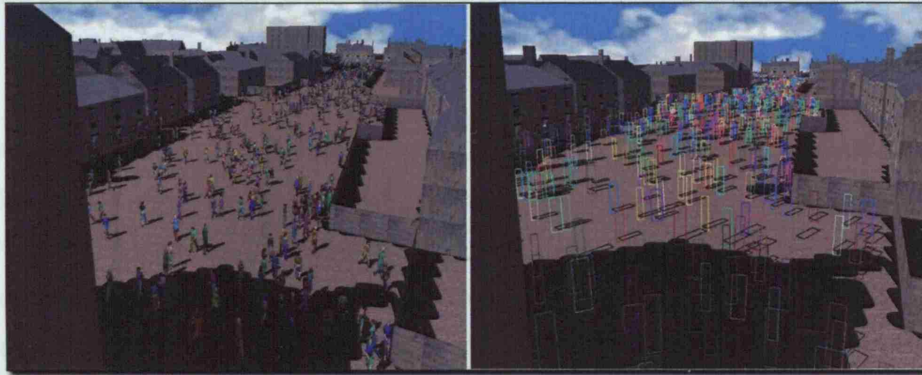

Figure 1.5 - High-performance crowd rendering using textured impostors.
Polygonal characters (left picture) can be replaced by image-based impostors (right picture).

Here a novel set of techniques that overcome or reduce these limitations are introduced, making it possible to render at interactive frame-rate crowds composed of thousands of individuals on commodity hardware using the technique called impostors rendering. The results show that impostor rendering can bring concrete advantages in many cases of real-time crowd visualisation. In particular this thesis reports on the following original contributions:

- A technique for rendering thousands of individuals in real time using *precomputed unstructured impostors;*
- A multi-pass algorithm that offers fine control on the colour of different parts of the impostor images;
- Approximate and per-pixel lighting algorithms suitable for crowd visualisation;
- Multiple approaches to crowd shadows generation and other global illumination effects;
- Visibility culling applied to crowd.

Also, some work on side aspects such as collision detection and navigation of crowds inside the Virtual Environments was performed through the work, with some material resulting in

scientific publications: where relevant, this material is also presented in the thesis chapters. It must be noted that, while we focused our work on the specific task of crowd rendering, the proposed techniques have a more general applicability, and could be used for a broader class of real-time visualisation problems.

1.7 Organization of this thesis

The thesis is composed by 7 chapters, organised as follows:

- Chapter 2 provides a detailed overview of the previous background and related work for real-time rendering methods, focusing in particular on the methods that are suitable for crowd rendering. For every approach (polygonal, point based, image-based) existing work on crowd visualisation is reported and discussed.
- Chapter 3 proposes the principle of using unstructured animated impostors for the visualisation of crowds in real-time, showing also how similar in flexibility to polygonal rendering impostors rendering can be in this case.
- Chapter 4 discusses how an impostor based representation of a crowd can be embedded in polygonal-based environments and how the two representations can interact flawlessly even for global effects such as shading and shadowing. Crowd visibility computation will also be discussed, and a simple but effective image-based collision detection algorithm will be proposed to deal with the problem of navigation of the crowd in the environment.
- Chapter 5 describes the representations and techniques that are needed to simulate large-scale crowds in a practical implementation. The successful integration of the method into an urban environment system is described, along with the main aspects for optimisation of performance.
- Chapter 6 analyses and formalises the performance of the crowd system, and shows the rendering performance that can be expected out of such a system.
- Chapter 7 summarizes the results and achievements of the present research and reports the final conclusions. Finally, future directions of the proposed approach are presented.

CHAPTER 2 - BACKGROUND AND PREVIOUS WORK

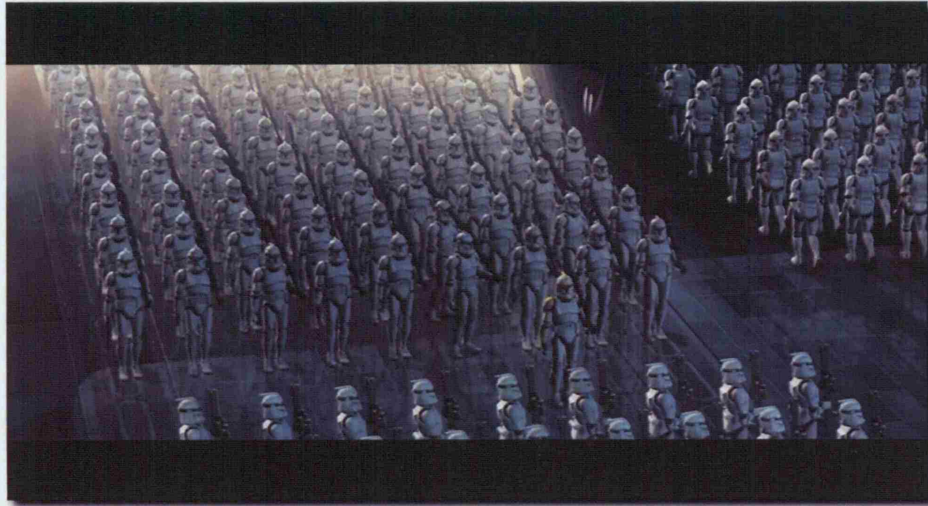

Figure 2.1 - A scene from the movie 'Star Wars Episode II: attack of the clones' (Lucasfilms – 2002).

The time-critical nature of Virtual Environments dominates the research field of Real-Time Computer Graphics. The available computing power is never sufficient for brute-force visualisation of complex scenarios, and there is a constant tension between trying to achieve a satisfactory frame-rate and visual realism of the generated image. Even with the most powerful graphics technology available today, very good care needs to be taken to handle the overwhelming complexity of Virtual Environment scenarios. The optimal management of the rendering process has been extensively analysed in the past 30 years, and numerous techniques to accelerate it have been proposed. In this context, the unique characteristics of highly populated urban environments add to the complexity of this general problem, requiring additional resources and dedicated approaches. The present chapter reviews the relevant research in the field, reporting, when possible, existing examples related to the challenging task of real-time crowd visualisation.

2.1 Real-time rendering using polygons

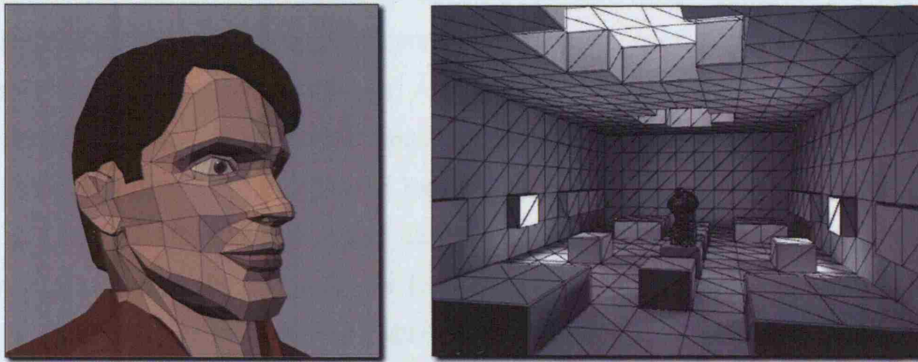

Figure 2.2 - Representing object geometry using polygons.

As real-time display of arbitrary geometry is mainly achieved with dedicated graphics hardware, model representation is generally limited to the graphics primitives supported by these architectures. To simplify on-chip implementation of the rasterization algorithms, graphics accelerators are usually optimized to handle simple graphical primitives such as points, lines and triangles (Figure 2.2). A multitude of arbitrarily small flat polygons is then by far the most common way to approximate any arbitrary surface. The standard method to achieve real-time rendering in Virtual Environments applications is to use a sequential rendering pipeline: in order to obtain a two-dimensional image starting from the initial three-dimensional representation, graphics primitives undergo a sequence of operations that can be summarized as in Figure 2.3.

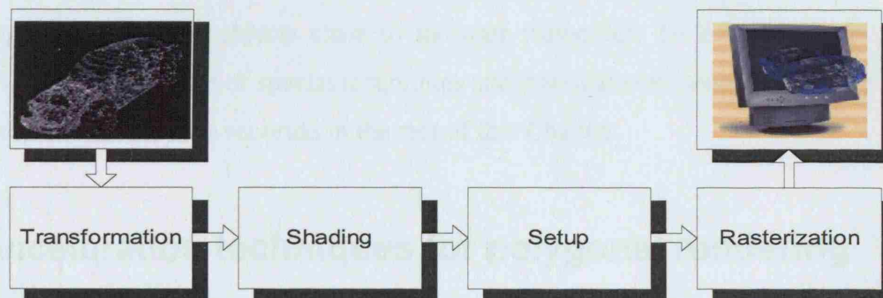

Figure 2.3 - A simple rendering pipeline.

Following this simple data-flow, vertices of the polygons are transformed into eye space and clipped to the extent of the viewing frustum, lighting effects are computed and finally the

mapping to the viewport is followed by the rasterization of the resulting primitives in order to display them as a collection of pixels on the raster device. These operations are carried out sequentially for every image to be generated, obtaining one of the frames in the smooth sequence of images displayed to the user. An in-depth review of a modern rendering pipeline can be found in any computer graphics book (e.g. [Watt99]).

Aside from requiring simpler dedicated hardware, there are other good reasons why planar facets, and in particular triangles, are so ubiquitous in computer graphics: creating polygonal objects is a straightforward process (at least for simple objects), and simple but visually effective algorithms exist to produce shaded versions of objects represented in this way. Gourad [Gouraud71] proposed a method for linearly interpolating a colour across a polygon's surface to achieve smooth lighting, giving a polygonal mesh a more smooth appearance. As a result of its visual quality and its modest computational demands (since lighting calculations are performed per-vertex and not per-pixel), it is still the predominant shading method used in 3-D graphics hardware, even thought more complex methods have also been proposed ([Phong75, Debevec02]) over time.

Advances in the throughput of rendering hardware are however continually challenged by the need to render more and more complex scenes and objects, and VEs consisting of tens or even hundreds of millions of polygons are becoming increasingly common. As the complexity of the scene increases, the brute-force polygonal approach to rendering reveals major limitations: standard tessellation does not take into consideration the size the final mesh will have on display, nor that different parts of a very large object could in theory need different resolutions depending to the distance from the user viewpoint, or that a large portion of primitives could result to be occluded by objects close to the user viewpoint. To be rendered in real-time, complex VEs need the use of special techniques and management strategies for their geometry data base, such as the ones reported in the rest of this Chapter.

2.2 Acceleration techniques for polygonal rendering

Sending all the triangles comprising a model down the rendering pipeline obviously produces a correct final image (presuming that transforming, clipping and depth-buffering is applied correctly). However, this is far from optimal considering the following characteristics of perspective projection:

- only a portion of the model lies within the current field of view
- objects are obscured by other objects closer to the point of view
- distant objects appear smaller exhibiting less visual detail

It is inefficient to transform objects lying outside the current field of view to image space because in the end they are clipped away. Similarly, rendering objects obscured by other objects consumes transformation, lighting and some rasterization resources and does not contribute to the final image because those objects will be discarded at the end of pipeline by the visibility test algorithm [Catmu74]. Furthermore, for the way rasterised images are generated on modern graphics workstations, most of the details present in distant object models will not be visible in the final image, while on the contrary this makes proper filtering of the final image a tough problem [Shoup73]. Cleverly designed algorithms can be used to discard early from the pipeline those primitives that will not contribute to the final image. Similarly, it is possible to select the appropriate level of detail of the geometric models which will be visible in each frame; these are all effective ways to improve rendering speed and produce little or no degradation in term of the images final quality.

2.2.1 Visibility culling

Avoiding the processing of the geometry of the environment that will not contribute to the final image was the first optimization approach investigated in Real-Time Computer Graphics and is known as *visibility computation*. What makes visibility very important is that, for large scenes, the number of visible fragments is often much smaller than the total size of the input geometry (Figure 2.4). The traditional approach to visibility culling involves various kinds of geometrical operation performed on the scenario polygonal database and targeted to identify the polygons that are occluded by other obstacles or do not lie in the field of view. The aim is to avoid rendering of objects which are not visible and thus will not contribute to the final image. The most common form of visibility culling is *view-frustum culling*: in this technique. the viewing frustum is delimited by six planes in three-dimensional space. Single points can be classified as lying within or outside the viewing frustum by intersecting the half-spaces described by the six planes. Only points which lie in this intersection lie within the viewing frustum. Similarly points can be rejected if they lie outside any of the half-spaces. In order to classify more complex geometry with respect to the viewing frustum, bounding volumes

enclosing the geometry as axis aligned boxes or spheres can been used. (see [Tesch05] for details and classifications of more complex bounding volumes).

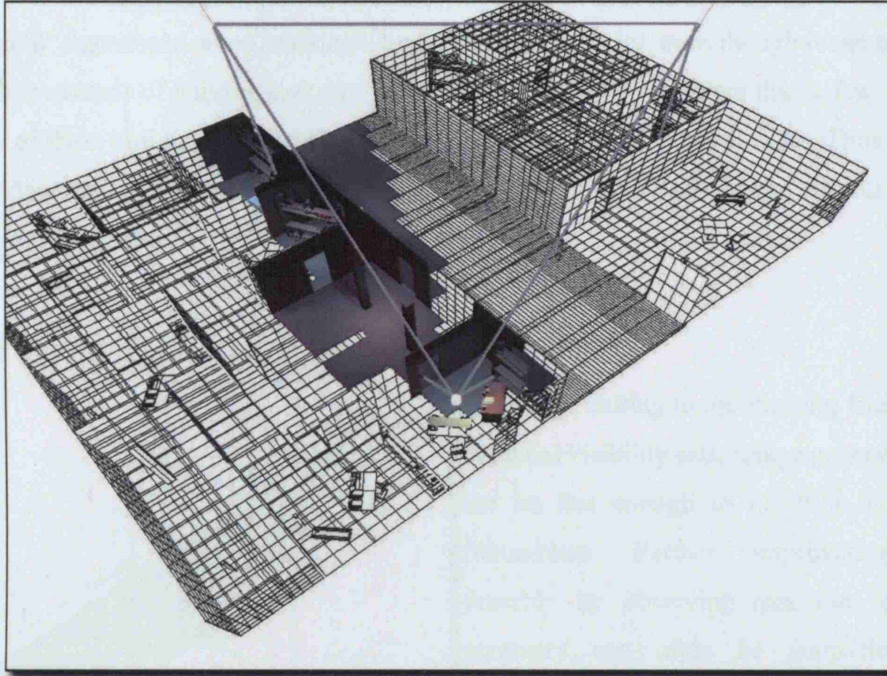

Figure 2.4 - The Visibility Culling process (image - J. Bittner).
The parts of the scene represented in wireframe are either not in the field of view of the user or occluded by objects in the foreground. Note that the occlusion test is conservative and that it considers the presence of doors inside the view-frustum.

When rendering large static scenarios, objects which are outside the view frustum can be quickly identified using a hierarchical view volume culling method [Clark76]. Sub-linear effort culling tests can be constructed with bounding volume hierarchies or hierarchical space partitioning data structures (k-d-trees, bsp-trees, octrees) [Cohe95][Naylor90][Samet90] . Even if view frustum culling rationalises and reduces the total amount of geometry that needs to be processed, it still does not avoid the rendering of those objects that are partially or totally occluded by other objects placed in front of the viewer even though they are inside the view-frustum. *Occlusion culling* algorithms aim at quickly discarding objects that are hidden from the view by other objects. Building interiors were the first application of such methods [Airey90], [Teller91], [Luebke95] since they have some well defined properties that can be exploited (high occlusion and well defined occluders mostly rectilinear). Other more general

methods appeared later [Greene93], [Zhang97], where the culling is done by hierarchically comparing the scene against the image area occupied by the rendering of the occluders. A special case of VEs are urban environments models, where it is normally possible to apply 'customized' algorithms; when walking along the streets of a city, even though it can be a huge city with thousands of buildings, at any given moment we never see more than a few. Also the majority of these buildings extend from the ground upwards with vertical walls. Thus they can be considered to be 2.5D objects (2D + height) which allows for much simpler and efficient algorithms [Wonka99].

2.2.2 Level of Detail management

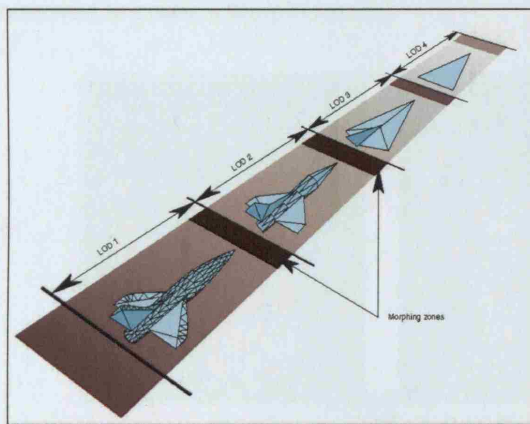

Figure 2.5 - Multiple LODs can be used to approximate the shape of distant objects (image SGI).

Even with culling to the viewing frustum and potential visibility sets, image generation may not be fast enough to result in interactive frame-rates. Further improvements are possible by observing that the remaining geometry can often be simplified; with perspective projection distant objects only project onto small regions in image space and, as a result, any fine detail in the models of these objects is lost because of the discrete sampling of a rasterized image. In an effort to optimise rendering speed one would like not to render any detail at all which does not contribute to this final image, as proposed by Clark [Clark76]. When, instead of one model per object at full resolution, several models of an object are available which approximate the object's geometry at several levels of detail (LODs), one of these LODs can be selected depending on how large the object will project onto the image (Figure 2.5). Other LOD selection factors that can be used are screen distance, priority, and perceptual factors. Even if the LOD principle is simple, a great deal of research has been carried out both on how to better approximate the shape and how to construct more flexible data structures to represent it. Since the work of Clark, the literature on geometric LOD has become quite extensive. It has been used since the early days of flight simulators, and has later been incorporated in walkthrough systems for complex environments ([Funkhouser92], [Maciel94]). In order to

have several LODs available per object, methods are needed to generate them. When generated manually, the addition of LODs multiplies the effort to model an object. Several automatic methods are available, some of which come very close to the quality of human-generated LODs at the cost of considerable processing complexity, but these automatic methods are not equally applicable to all kinds of models.

A more sophisticated form of the LODs technique are continuous LODs, also called *progressive meshes* [Hoppe97] (Figure 2.6). These continuous LOD algorithms have been presented to overcome the two major problems of discrete LOD algorithms: increased storage requirements and fidelity discontinuities both in time and space. These approaches build a hierarchy on top of the vertexes in the model which describes how vertexes can be slowly removed to create a simpler model.

Figure 2.6 - Progressive Meshes.
The algorithm generates a long sequence of decimated meshed, resulting in smooth transitions between LODs

In addition to "vertex removals" (or equivalently "edge collapses"), the changes to the triangle mesh containing the vertexes are recorded and used to update the triangle mesh to the required fidelity. As a result only one data structure is built without replication of data in several discrete LODs. Moreover the geometry can be approximated to varying degrees of fidelity in different parts of the geometric model. The historical drawback of progressive meshes is represented by their lack of compatibility with hardware representation. The rigid and highly optimized pipeline of graphics hardware architectures is better suited to deal with a static vertex buffer than with a variable amount of vertex data. The continuous flushing of the pipeline and geometric cache-misses caused by progressive meshes nature could result in inferior performance compared to the use of the static data in the original form. Later work has shown that it is possible to have continuous and hardware friendly meshes [Forsyth01].

What is still missing is an accepted standard for continuous meshes and the ability for the graphical hardware to directly compute the transition between different levels, a task still demanding for the relatively slow CPU.

2.3 Alternative approaches to polygonal rendering

If most of the initial work on real-time graphics for VEs has involved rationalising the data management process, nowdays even the choice of which graphics primitives is best suited to represent geometrical surfaces is under question. This has a very strong motivation: in the pursuit of photo-realism in conventional polygon-based computer graphics, models have become so complex that most of the polygons are smaller than one pixel in the final image. Formerly, when models were simple and the triangle primitives were large, the ability to specify large, connected regions with only three points offered a considerable efficiency in storage and computation, but now that models contain nearly as many primitives as pixels in the final image, we should rethink the use of geometric primitives to describe complex environments. Citing from [Watt98] '*In many ways modeling and representation is an unsolved problem in the computer image. The most popular way of representing an object - by approximating it with a set of planar facets - has many disadvantages when the object is complex and detailed. In mainstream computer graphics the number of polygons in an object representation can be anything from a few tens to hundreds of thousands. This has serious ramifications in rendering time and object creation cost and in the feasibility of using such objects in an animation or virtual reality environment.*' As complexity increases, coherence decreases, and beyond some threshold level, the time saved through the evaluation of coherence does not justify its expense, and when the number of polygons in the object exceeds the number of pixels on the screen, coherence becomes almost useless. This is why one of the most important single advances in the realism of computer imagery came with the development of texture mapping that associating a tabular two-dimensional array of intensity values with each geometric primitive, allows the visual complexity to far exceed the shape complexity.

Expanding these considerations, approaches emerged to deal with extremely complex scenarios using alternative forms of objects representation. One possibility is the adoption of a higher level of representation to specify an object shape using, for instance, surface functions and introducing the concept of dynamic tessellation: one of the benefits of using parametric

equations would then be that the number of triangles dedicated to approximate the various surface regions of an object could at run-time be adapted to the projected size of that region on the screen. Despite this and others potential advantages, the use of surfaces in real-time rendering is not widely supported by rendering hardware, due to difficulties inherent in this type of representation. Controlling the continuity of a surfaces proved to be particularly problematic, often leading to shading artifacts that are hard to be avoided. Also, higher order primitives are not suitable as a universal form of representation: irregular shapes are inherently difficult to be represented using smooth curves. Finally, while geometrically defined primitives, curves or surfaces are an efficient means for describing man-made environments, in nature there are many objects and phenomena that are not readily modeled using classical surface elements. Terrain, foliage, clouds and fire are examples in this class.
Radically different approaches to CG rendering have been proposed lately with the introduction of Point-Based Rendering and Image-Based Rendering, that are at present the most notable alternative to polygons, and those principles are resumed in the following paragraphs; what makes these techniques very interesting in our case is the fact that they can be successfully applied to the case of real-time crowd rendering.

2.4 Point based rendering

In certain situations even points can be used as an alternative graphics primitive to the triangles. For instance, it has been shown that points are good primitives for the modeling of intangible objects. Csuri [Csuri79] successfully used points to model smoke, while Blinn has used points to model the shading of clouds [Blinn82] and Reeves to model fire and trees [Reeves83]. While all of these efforts treated classes of objects that could not be modeled using classical geometries, this is not, however, a necessary restriction; points are highly versatile and a wide class of geometrically defined objects, including both flat and curved surfaces, can be converted into points. The general use of a point based representation to represent 3D objects was suggested for the first time in [Levoy85], where the authors showed that as the complexity of computer generated scenes increases, triangles as display primitives become less and less attractive compared to points. Later Max and Ohsaki used point samples obtained from orthographic views and stored with colour, depth and normal information to model and render trees [Max95]. The fundamental challenge of point sample rendering is the reconstruction of continuous surfaces. While in polygon rendering continuous surfaces are

represented closely with polygons and displayed to high accuracy using scan conversion, point sample rendering can only represent surfaces as a collection of points which are forward mapped into the destination image. It is entirely possible for some pixels to be 'missed', causing gaps to appear in surfaces (Figure 2.7). It is therefore necessary to somehow fix these surface holes in a manner that does not seriously impact rendering speed. Rendering primitives larger than a single point (called *splats*) is a typical approach to this problem.

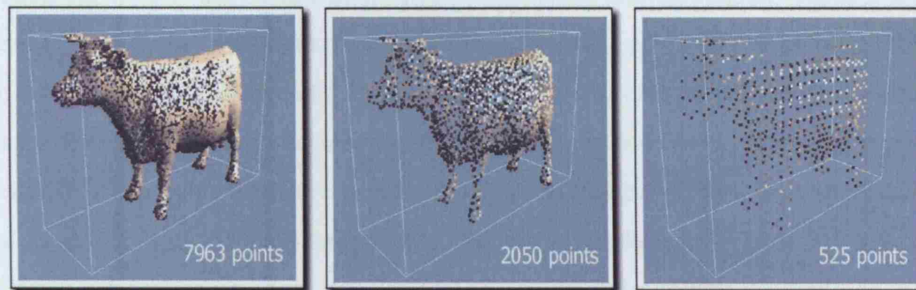

Figure 2.7 - Point based rendering at various resolutions of a generic 3D object (image M. Wand).

A secondary challenge of point based rendering is the automatic generation of efficient point sample representations for objects. The problem is normally to select an appropriate set of directions from which to sample the object, and to select a suitable set of point samples from each direction such that the entire set of samples forms a complete representation of the object with minimal redundancy. A common approach is to organize the object geometry in a hierarchical octree structure, as suggested in [Yemez99], containing enhanced samples of the underlying geometry, where points can have additional information such as surface normal and/or local curvatures. The depth of the octree is usually limited by a point count budget or curvature variation estimated using Principal Components Analysis (PCA) and a correctly oriented representative normal associated with each cell. During rendering, points in each cells are stochastically sampled and rendered. The number of samples is determined by multiple visual cues including image size, local surface features, silhouette containment and occlude-potential. Larger number of points samples are rendered in regions of high curvature and around the silhouette, while lesser number are rendered in flatter regions or when the occlude - potential is high. The representative normal is used to correctly orient the local normal. The local normal and curvature at the point sample are used to decide on the splat shape, size and illumination.

2.5 Image based rendering

At the root of Image Based Rendering (*IBR*) is the observation that in many cases it is much faster to render an image representing a complex object or scene than it is to render the scene itself starting from its original representation inside a scene graph. Even considering a scene as composed of several sub-objects, if we can render one image representing each complex object and properly compose the final image, this process will be orders of magnitude faster than rendering directly the objects themselves. Also, using photographs or videos it is possible to represent scenes that are not explicitly modeled. The first paper on Image Based Rendering was in 1976 entitled "Texture and Reflection in computer Generated Images" [Blinn76]. In this paper Blinn and Newell formalised the idea, which had been previously suggested in [Catmull75], of applying a texture to the surface of an object. Modern IBR approaches are a generalization of Blinn initial concept, and they try to represent complex 3D environments with sets of images that in some way include information describing the depth of each pixel along with the colour and other properties.

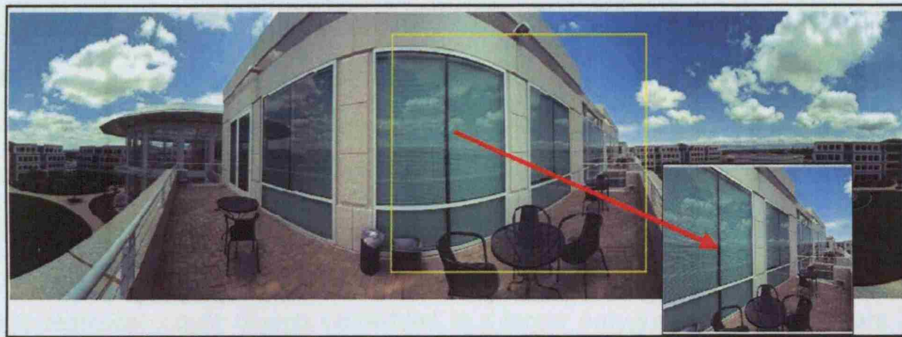

Figure 2.8 - Warping of panoramic images to compensate perspective distortion (image S. E. Chen).

Using this set of 'rich' images, various warping algorithms can be applied to produce new images from viewpoints that were not included in the original image set [Chen95]. Thus, using a finite set of source images, it is possible to produce new images from arbitrary viewpoints. In [Lipman80] part of a small town was modeled by recording panoramic images at 3 meters intervals along each of several streets and recording these images to videodisc. A user could then virtually drive through these streets and look in any direction while doing so. It must be noted that camera rotation was continuous, but camera translation was only discrete – there was no interpolation between neighboring viewpoints. Chen extended this idea to

provide continuous motion between viewpoints by morphing from one to the next [Chen93]. However, the virtual camera was constrained to lie on the lines joining adjacent viewpoints, and the linear interpolation of the optical flow vectors which was used for morphing provided approximate view reconstruction only. In [Laveau94] and [McMillan95] the optical flow information was combined with knowledge of the relative positions of the cameras used to acquire the images. This allowed the researchers to provide exact view reconstruction from arbitrary viewpoints. In [Maciel95], the hybrid approach of using texture mapped quadrilaterals, referred to as *planar impostors*, to represent objects in order to maintain an interactive frame rate for the visual navigation of large environments, was presented.

2.5.1 Impostors rendering

Rendering an environment using only images is a very powerful approach, but deviates quite radically from the standard polygon-oriented rendering pipeline. Dedicated hardware architectures have been proposed in the past to deal with Image-Based Rendering [Molnar92] [Torborg96][Eyles97], but these systems never achieved commercial success. Also, polygons can be useful for other properties not strictly related to graphics - for example, tasks such as collision detection and physical simulation need geometrical information about the objects in the scene, making the pure IBR approach in many cases too far-fetched and impractical for real-life use. The technique called *impostors rendering* was then proposed in an attempt to combine the standard, geometry-based rendering pipeline with IBR principles, generating an hybrid approach where IBR is mixed with the standard polygonal rendering. The idea of using images to represent single objects embedded in a larger polygonal scenario (Figure 2.9) was first mentioned in [Chen93] and realized by the 'object movies' described in [Chen95]. This system used image morphing to approximately reconstruct novel views of an object (a simpler approach to generating approximate novel views is to simulate 3D movement by applying affine transformations to existing views). In effect, this treats the object as a flat rectangle textured with its image. To construct a nearby view, the textured rectangle is simply rendered from the new viewpoint.

Figure 2.9 - Embedding images in a polygonal scenario: the impostor (image P. W. C. Maciel).

This method has the important advantage of being able to make use of standard texture mapping hardware, but it is obviously a gross approximation of the proper perspective transformation.

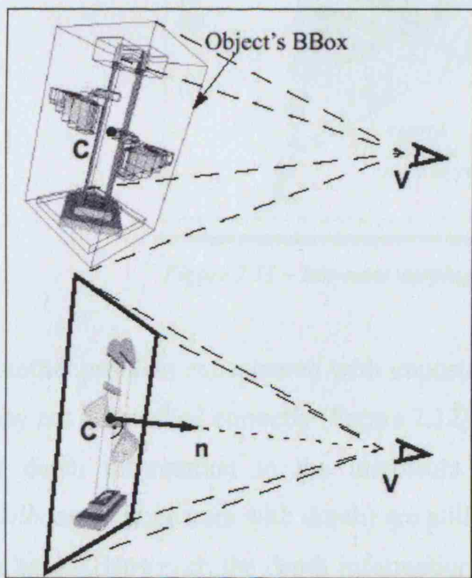

Figure 2.10 - A complex object is replaced by a single, texturised, quadrilateral (image G. Shaufler)

Single image-based entities that are placed inside the polygonal scenario are called *impostors (Figure 2.10)*. There are two main approaches to the generation of the impostor images: in [Maciel95] the impostors were precomputed and selected for use at render time (*static generation* – also referred to as pre-generated impostors), while Schaufler [Schaufler95] and Shade [Shade96] independently presented the concept of dynamically generated impostors (*dynamic generation*). In the latter case, object images are generated at run-time and re-used as long as the introduced error remains below a given threshold. This latter approach was also used in the Talisman rendering system ([Toborg96]) to provide higher refresh rates than would be possible using polygons alone. Numerous algorithms can be found in the literature that try to generate better approximations of an object view starting from the available samples. For example Chen et al. [Chen93]and McMillan et

al. [McMillan95] warp the images to adapt them to different viewpoints while Mark et al. [Mark97] and Darsa et al. [Darsa97] apply the images on triangular meshes to better approximate the shape of the object. Unfortunately, due to the complexity of the operations involved, many warping procedures have to take place on the CPU, producing a data traffic between the CPU and the GPU that can attenuate or even completely remove the speed advantages of using impostors; Schaufler [Schaufler98] proposed an interesting hardware-assisted approach to image warping using layered impostors (Figure 2.11) and Dally et al. [Dally96] used an algorithm that is very efficient in storing image data starting from a number of input images.

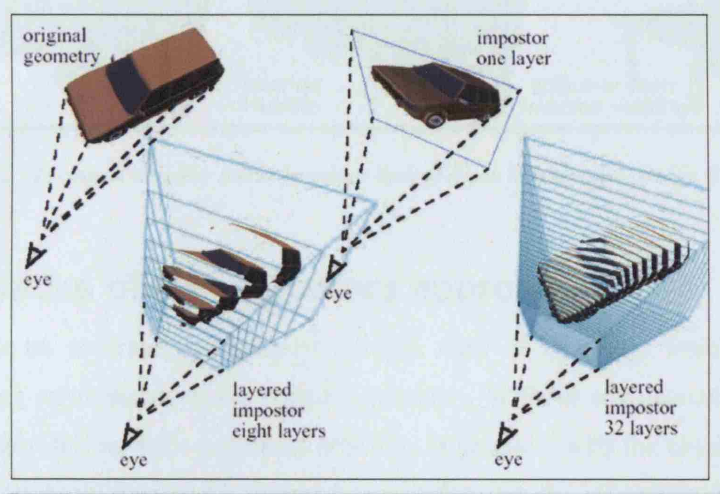

Figure 2.11 – Impostor warping using image layers (image G. Shaufler)

Another problem encountered with impostors is that because they are flat, object intersections may not be handled correctly (Figure 2.12). To address this problem, Schaufler added a 'layer' of depth information to the impostors [Schaufler97]. The resulting primitives, named *nailboards* (impostors with depth) are still rendered using the same 2D affine transformation as before. However, the depth information is used to compute approximate depths for pixels in the destination image; in this way proper object intersections can be handle by the standard Z-buffer mechanism. Obviously, the depth information can also be used to compute an exact 3D warp as in [Mark97]. Later, due to planar impostor providing a good visual approximation for complex objects at a fraction of the rendering cost, a large amount of research has refined the impostor concept, introducing further innovations in the main idea. Examples include

types of impostors such as layered impostors [Decor99], billboard clouds [Decor03], and texture depth images [Jesch02] for rendering acceleration of various applications. A survey of these different types, including their application and their advantages and disadvantages, can be found in [Jesch05].

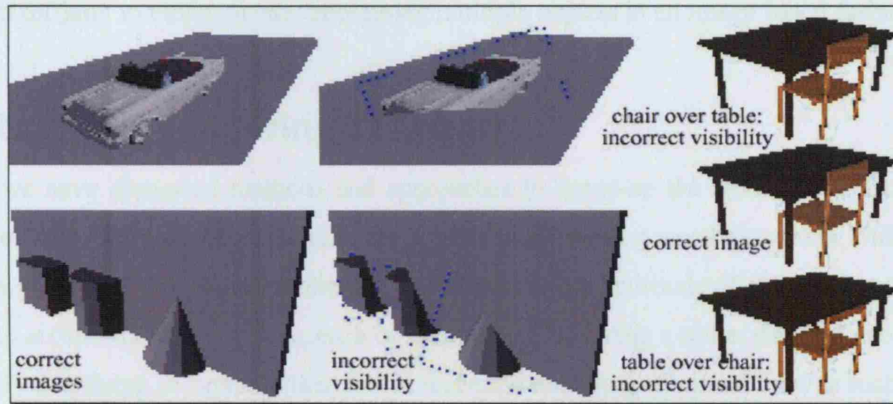

Figure 2.12 - Incorrect visibility due to impostor lack of depth information (image G. Shaufler).

2.6 Drawbacks of the impostors approach

It is impossible to accurately reconstruct a novel view of an object from existing views without knowing anything about the object's geometry. Without any geometric information, we will invariably end up with geometric artifacts. In practice, with the notable exception of impostors which treat objects as textured rectangles, nearly all image based rendering algorithms make use of some sort of geometric information. This information is either in the form of correspondences between images [Chen93], [Chen95], [Laveau94], [McMillan95], explicit per-pixel depth information with known transformations between views [Dally96], [Pulli97], or an approximation of the object's shape which is used to alleviate, but not eliminate, the geometric artifacts [Gortler96], and the extent to which an image based rendering algorithm suffers from geometric artifacts depends on the manner in which it makes use of geometry. Another side-effect of rendering without the complete geometric information available in polygon systems is the inability to use dynamic lighting. Depending on the situation, this is usually seen as an advantage rather than a drawback of image based rendering: the lighting computation, although static, comes for free since it is captured in the images of the object, and thus it requires no computation at render time. Furthermore, it will contain all of the complex and usually computationally intensive lighting effects that are

present in the original images such as shadows, specular highlights and anisotropic effects. Notwithstanding these advantages of static lighting, there are certain applications which demand dynamic lighting. Finally, since a large number of images are required to properly sample an object, image based techniques are usually fairly memory intensive. This usually makes it difficult to render scenes containing multiple objects in an image based fashion.

2.7 Real time rendering of crowds

So far we have discussed methods and approaches to speed-up the rendering of large static scenarios, but many of the principles are applicable to more general situations. Still, crowd rendering presents unique characteristics, as it deals with a multitude of animated entities that navigate around the environment, each of them ideally showing a rather detailed geometry. It is mainly this shape complexity that makes real-time rendering of virtual crowds such an hard task even using modern graphics hardware.

2.7.1 Modeling the human body

As with any other kind of models, the most common way to represent characters in 3-D computer graphics is the polygonal mesh. However, as the need for realism increases, more detailed models are necessary, requiring higher polygonal budgets (i.e., several thousand) to model the character. This detail comes at a great rendering cost, needing to find a balance between realism and interactivity, especially when the number of characters to render is large.

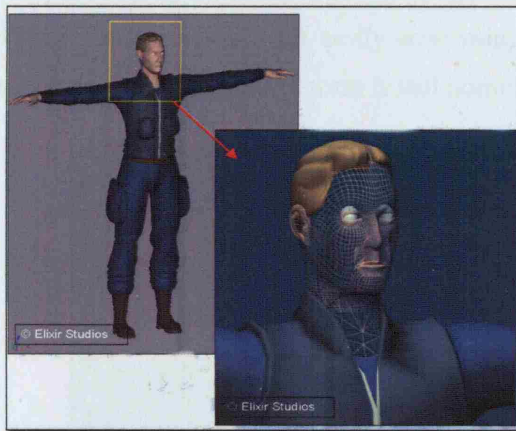

Figure 2.13 - Reproducing the human shape needs a large polygon budget.

Also, proper real-time lighting of these meshes is important in order to enhance the overall crowd realism. Typically, the lighting of the character's mesh in games is still implemented with basic Gouraud shading [Gouraud71] even if, more recently, more sophisticated forms of lighting have been used for interactive simulations such as diffuse lighting using ambient occlusion[Zhukov98] and image-based lighting[Debevec02]. Very frequently

texture mapping is used to attach a two-dimensional image onto the polygon's surface, in this way greatly improving the realism of the mesh. These textures are usually artist-drawn or scanned photographs and are typically used to capture the detail of areas such as human's hair, clothes and skin. Many other special techniques have been proposed to increase the realism of human characters rendering (realistic hair rendering [Kim02] and sub-surface scattering skin illumination effects [Jensen01] just to name a couple), but these are normally restricted to the rendering of a single, very detailed model. For real-life applications the severe time-constraints of interactive graphics still impose for crowd rendering the use of just the basic OpenGL lighting style.

2.7.2 Rendering crowds using polygons

As discussed earlier, when a plain polygonal representation is used for each mesh in a crowd, the overall scene complexity often results in a polygonal budget that is impossible to handle at interactive frame rates. A first way to speed-up crowd visualisation is the use of the LODs technique, similarly to what can be done with static scenarios. Depending on the scene layout, the number of triangles in each character's mesh (or any other element of the scene) can be reduced depending on the projected size of the character on the screen, to optimise the computational load and achieve a real-time frame rate. A discrete LOD framework can obviously be used to handle a geometric LODs hierarchy of virtual humans, and highly detailed mesh can be simplified using automatic tools to create multiple low resolution meshes varying in detail. Unfortunately, due to the complex topology of a human body, simplification artifacts can easily arise using automatic simplification procedures, so that a time-consuming manual process is still normally necessary (Figure 2.14).

Figure 2.14 - Simplification of polygonal characters.
People are very very acquainted to the shape of the human body and can easily spot the artefacts of automatic mesh simplification.

Several examples of polygonal-based crowd rendering exists: in order to solve the problem of rendering large numbers of humans, De Heras Ciechomski et al. [Ciechomski05] avoid computing the deformation of a character's mesh by storing precomputed deformed meshes for each key-frame of animation, and then carefully sorting these meshes to take cache coherency into account. Ulicny et al. [Ulicny04] improve on their performance by using 4 LOD meshes consisting of 1,038, 662, 151 and 76 triangles and disabling lighting for the lowest LOD, thereby achieving a frame rate several times higher.

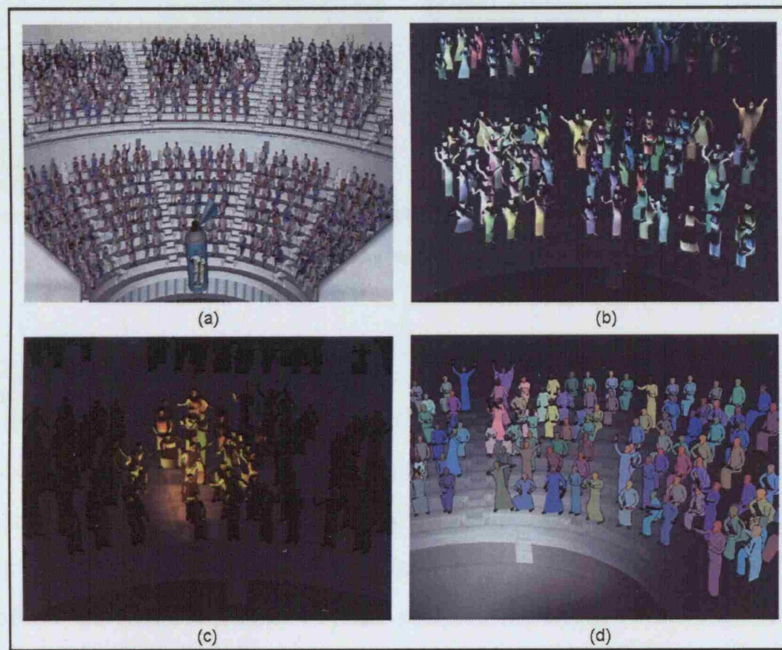

Figure 2.15 - Polygonal crowd used to populate a 3D model (image U.B. Ulicny)
(a) rendering with texture, (b) shading and no texture,(c) local lighting effects, (d) cartoon-like shading.

Visual variety is another critical aspect of crowd rendering. Ulicny et al. [Ulicny04] introduce crowd variety using several template meshes for the humans and then modifying some of the attributes at run-time, such as textures, colours, and scaling factors. In this way they succeed in simulating several hundred humans populating an ancient Roman theater (Figure 2.15) and a virtual city while maintaining interactive frame-rates. Gosselin presented an efficient technique for rendering large crowds while taking variety into account optimising the way geometric data is prepared for the graphic hardware[Gosselin05] (Figure 2.16) . His approach involves reducing the number of API calls needed to draw a character's geometry by rendering multiple characters per draw call, each with their own unique animation. This is achieved by

packing a number of instances of character vertex data into a single vertex buffer and implementing the skinning of these instances in a vertex shader. As vertex shading is generally the bottleneck of scenes containing a large number of deformable meshes, they minimise the number of vertex shader operations that need to be performed.

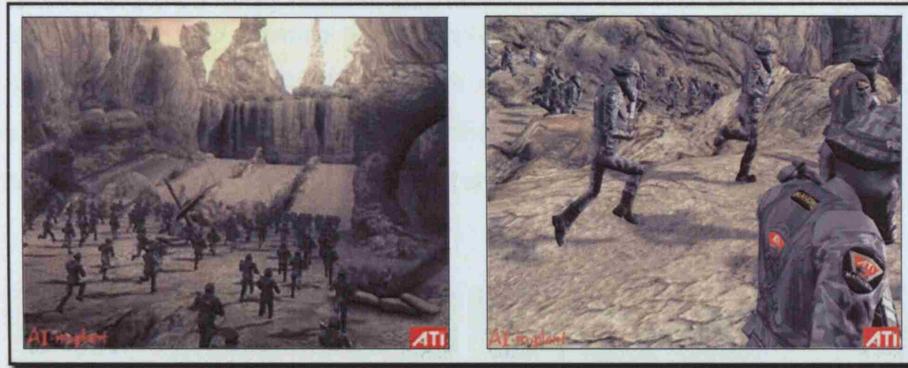

Figure 2.16 – Rendering an army using polygons (image D. Gosselin).

2.7.3 Rendering crowd using points

Even if point based rendering is a rather generic approach to real-time visualisation, this sampled-based approach has also been applied to the specific case of rendering virtual humans. As in the general case, point-based crowd rendering involves replacing a mesh with a cloud of points, approximately pixel-sized (Figure 2.17) . This involves converting key-frame animations of meshes into a hierarchy of point samples and triangles at different resolutions.

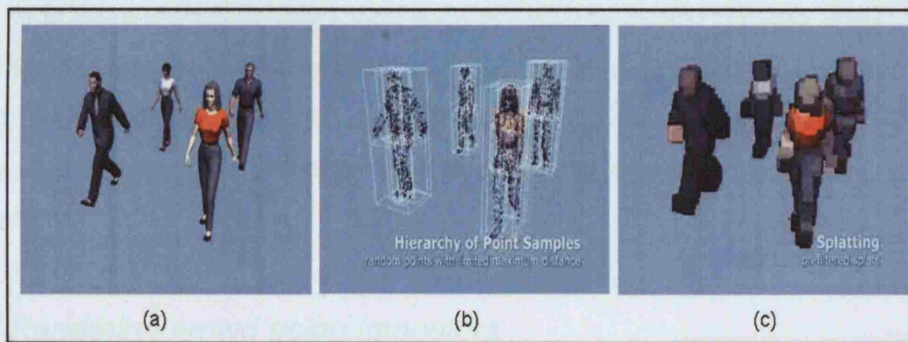

Figure 2.17 - Rendering animated characters using points (image M. Wand)
a) the original models, b) subdivision of the original mesh and point sampling, c) rendering with different splats sizes.

Wand et al. use a precomputed hierarchy of triangles and sample points to represent a scene [Wand02]. They partition the scene's triangles using an octree structure and choose sample points which are distributed uniformly on the surface area of the triangles in each node. Using this multi-resolution data structure, they are able to render large crowds of animated characters (Figure 2.18). For smaller crowds, consisting of several thousands of objects, each object is represented by a separate point sample and its behavior is individually simulated. Larger crowds are handled differently, with a hierarchical instantiation scheme, which involves constructing multi-resolution hierarchies (e.g., a crowd of objects) out of a set of multi-resolution sub-hierarchies (e.g., different animated models of single objects).

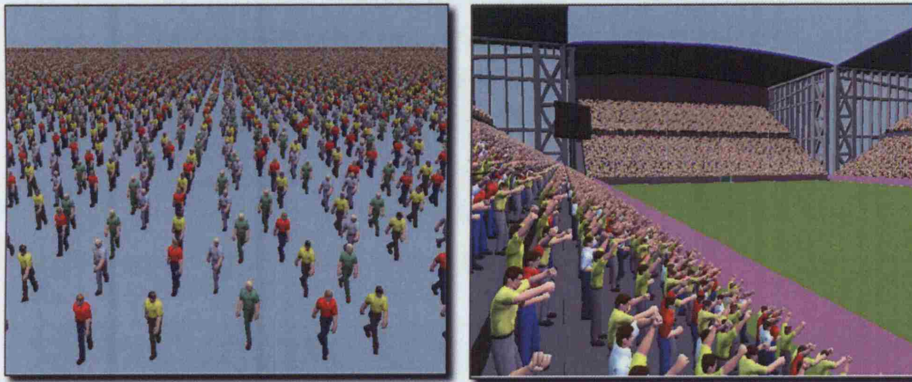

Figure 2.18 – Two examples of crowd rendering using points (images M. Wand)

While multi-resolution hierarchies allows them to render arbitrarily complex scenes, such as 90,000 humans walking on the spot and a football stadium with 16,000 spectators (Figure 2.18(b)), less flexibility is provided for the motion of the objects, since the hierarchies are precomputed and therefore cannot be used in simulating a large crowd moving within its environment.

2.7.4 Rendering crowd using impostors

An alternative to polygonal and point-based rendering was presented for the case of large crowds by Aubel et al. [Aubel98]; in particular, the use of impostors for the rendering of the virtual humans was presented. In [Aubel98] each human is replaced by a single dynamic impostor while in [Aubel99] the authors take a much more detailed approach where each body

part is replaced by a dynamic impostor, overall using 16 impostors for each human (Figure 2.19).

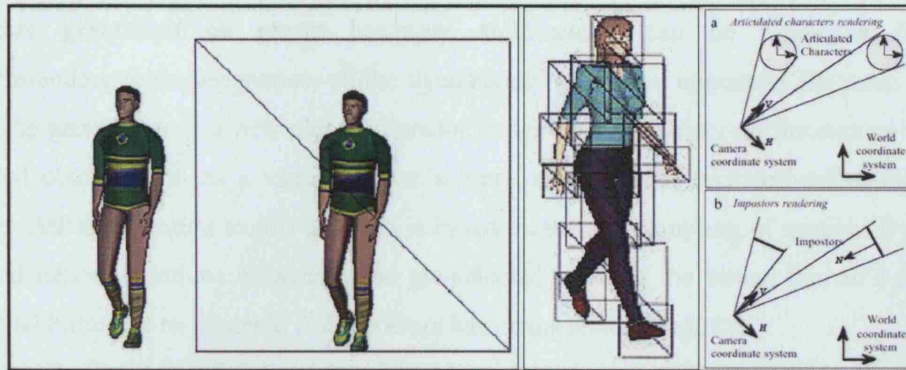

Figure 2.19 - Using impostors for crowd rendering (image A. Aubel)
In Aubel's original work, a human-like polygonal model can be replaced either by a single dynamic impostor (left), or by a collection of smaller impostors refreshed at a different frame rates (right).

In [Aubel00] an example with dynamically generated impostor used to render a crowd of 200 humans performing a 'Mexican wave' is reported (Figure 2.20).

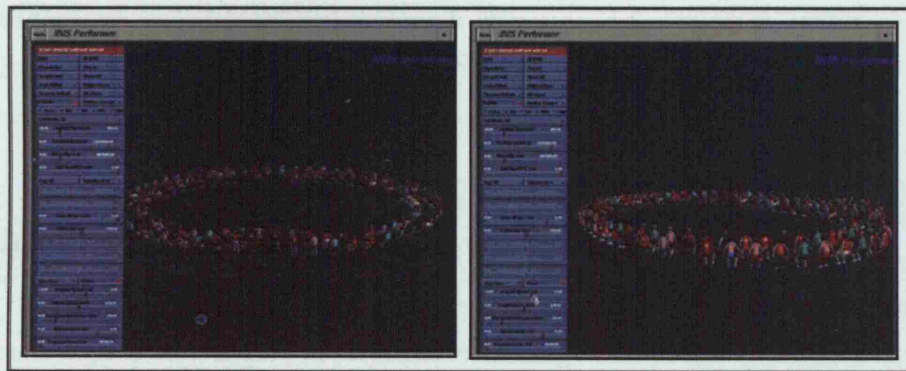

Figure 2.20 - Aubel's dynamic impostors performing a 'Mexican wave' (image A. Aubel).

When using dynamically generated impostors, the impostors images are updated at run-time by rendering the object's mesh model to an off-screen buffer and storing this data in the image. The image is displayed on a quadrilateral, which is dynamically orientated towards the viewpoint. This uses less memory, since no storage space is devoted to any impostor image that is not actively in use. On the other hand, dynamic generation of impostors is fairly computational intensive, and strongly influenced by the particular graphics hardware

architecture and software driver efficiency. Some of the operations involved in this process used to be extremely slow to process, and only recently these operations have been optimized in the rendering pipeline. An extensive analysis of what can be expected using dynamic impostors generation on recent hardware architectures can be found in [Day05]. Notwithstanding these progresses, unlike dynamically generated impostors for static objects, where the generation of a new object impostor image depends solely on the camera motion, animated objects such as a virtual human's mesh also have to take self-deformation into account. Aubel's solution to this problem is based on the sub-sampling of motion. By simply testing distance variations between some pre-selected joints in the virtual human's skeleton, the virtual human is re-rendered if the posture has changed significantly.

As for static scenes, the planar nature of the impostor can cause visibility problems as a result of the impostor inter-penetrating other objects in the environment. To solve this problem, Aubel et al. propose using a multi-plane impostor which involves splitting the virtual humans mesh into separate body parts, where each body part has its own impostor representation. However, this approach can cause problems similar to those discussed in Section 2.2, resulting in gaps appearing in the final image. Unfortunately, dynamically generated impostors rely heavily on reusing the current impostor image over several frames in order to be efficient, as animating and rendering the human's mesh off-screen is too costly to perform regularly. Therefore, this approach does not lend itself well to scenes containing large dynamic crowds, as this would require a coarse discretization of time, resulting in jerky motion.

2.8 Summary

The present Chapter has analysed the mainstream approaches to real-time rendering of complex Virtual Environments, in particular examining their specific application to the task of rendering large crowds in real time. Accelerated techniques for the rendering of large crowds has been discussed in some detail as this is the most computationally intensive part of crowd simulation, accounting for the biggest overall system bottleneck. Previous work of real-time rendering of crowds using polygons, points and dynamic impostors has been presented.

CHAPTER 3 - CROWD RENDERING USING IMPOSTORS

Figure 3.1 - A scene from the movie 'Troy' (Warner Bros. - 2004).

The extreme geometrical simplicity of impostors makes them one of the fastest alternatives to the visualisation of objects having a complex geometrical shape. Despite this remarkable quality, prior to the present work impostors were used rarely for real time crowd rendering and always in their dynamic variant. A closer examination of this task and the availability of graphics hardware with larger amounts of dedicated video memory suggest a reexamination of this technique. A novel way of using static impostors is proposed in this Chapter: we first introduce the basic principles of unstructured impostor rendering applied to crowds, and then discuss how such impostors could be built, stored and managed. Additional techniques for impostors colour control and lighting are also presented, showing that in many situations impostors can be effectively used for real-time crowd visualisation tasks.

3.1 Rendering human - like characters using impostors

The basic idea of the impostors is to use a single image to replace some complex geometry. Even if originally used to replace parts of static scenarios, impostors can obviously be used also to replace polygonal objects representing human-like figures. In the case of unstructured impostors, a single image captures the complete visual apparency of the (complex) polygonal object. In its simpler description, our approach to crowd rendering uses a set of precomputed textures to store a collection of sampled images, each one corresponding to a frame of animation of a human performing some actions (a walking sequence, for example). At run time, depending on the view position with respect to each individual, the most appropriate image is chosen and displayed on an impostor. The principle is depicted in Figure 3.2, where the polygonal mesh of a character is replaced by a single image, properly positioned in 3D space.

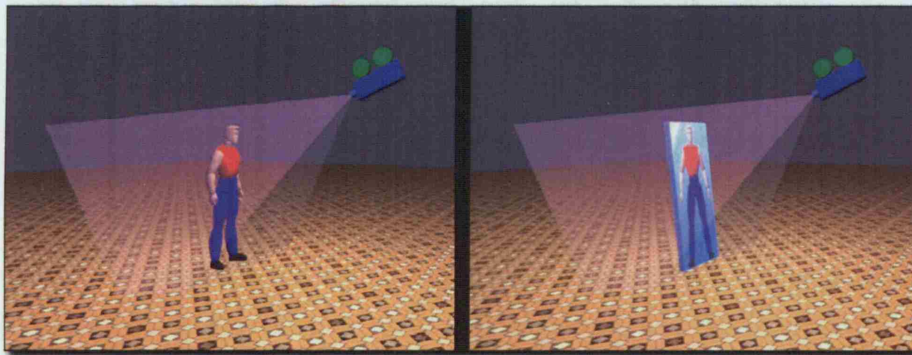

Figure 3.2 - Impostors can be used to replace the complex geometry of a human character.

Clearly, an impostor has the same visual aspect of the replaced object only as long as the camera is placed exactly in the sampling position; moving the viewpoint around would need to replace both the impostor image and its projection plane in space. As reported in Chapter 2, the set of sampled images needed to represent an object as seen from different view directions can either be created in a preprocessing phase (*static* impostors) or at run-time (*dynamic* impostors). We propose for the task of crowd rendering the idea of a *static, unstructured impostors representation:* our approach achieves the maximum possible rendering speed using only one polygon per human, and a precomputed set of images sampled from a given collection of view-directions; this makes our approach substantially different from the use of dynamic impostors proposed by Aubel [Aubel98].

3.2 Pros and cons of precomputed impostors

Due to their short life-span and high memory requirements, impostors are frequently created dynamically at run-time. As their computation can be very time consuming, dynamic impostors are commonly used in limited numbers and only to replace static objects, to avoid the need of frequent updates. Conversely, static impostors creation takes place at preprocessing, without the need of run-time updates for the images database, thus producing much faster rendering. While static impostors can, in principle, be used to replace animated objects (several independent impostors can be used to reproduce the key-frames of an animation sequence), storing a multitude of image samples might require a very large amount of memory, something that prevents their intensive use on systems equipped with a small amount of video storage space.

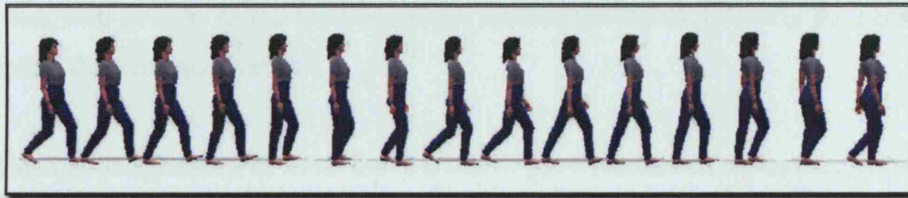

Figure 3.3 - A key-framed walking sequence can be captured using multiple impostors.

In the task of crowd visualisation, we have to deal not only with animated geometry, but also with the problem of visual variety: the individuals composing a crowd should not look all the same. This may raise even more the memory requirements of the impostor technique, as more variety usually means that more memory is needed to store more samples. Another drawback of static impostors is the inability to change the illumination stored in the image samples, that apparently make impossible the simulation of dynamic lighting or shadowing effects.

On the other hand, computational and memory resources constantly increase on modern graphics hardware. The apparent large consumption of texture memory produced by the use of static impostors is becoming less and less critical, while the rendering speed advantage of this form of representation is always extremely attractive. Moreover, when applied to the specific case of crowd rendering there are some peculiar aspects that can be exploited to save storage space, allowing us to reduce the total number of necessary image samples:

1. Characters in a crowd can have individual orientations, but each orientation is usually

defined simply by the amount of rotation around the vertical axis. This is a key consideration for the applicability of static impostors to crowd rendering: it is not necessary to sample the character mesh for all its possible orientation. Using simple geometrical transformations, impostors can be rotated at run-time and have an arbitrary orientation around the vertical axis.

2. In scenes that show a large number of humans, the vast majority of the characters are far from the viewpoint; impostors can then be used to replace the far individuals, maintaining a geometrical representation for the closer characters. With this mixed technique it is possible to use a limited image resolution for the object samples, tuning accordingly the memory requirements.
3. In a typical crowd simulation the viewpoint is normally placed above the crowd, as very rarely we have situations where a crowd is seen from below; for this reason we can limit our sampling process to the top hemisphere around the object, avoiding the necessity of sampling the remaining views.

The following paragraphs will show that the use of novel, dedicated techniques of impostors management can overcome the problem of visual variety as well as the limitations for dynamic lighting of impostors-based crowds. In practice, carefully handling of impostor data with the set of dedicated techniques presented in this thesis can effectively reduce the overall memory requirements and increase the flexibility of the basic method, leading to the successful application of the static impostors technique on a wide range of graphics hardware. The basic principles of our approach are reported in the following paragraphs, while a more in-depth description of the implementation details discussed in Chapter 5, where the general software architecture of our rendering system is also presented.

3.3 Building the impostors database

The approach described in this thesis uses only a single texturised polygon per human. Since no interpolation is used between different views (the reason for this is explained later in the present Chapter), each image sample represents only the closest match to the correct image, and it is chosen depending on the view direction discretization and on the frame of animation. The first phase of the algorithm consists in the generation of a sufficiently large set of sample views for the object. A representation of this process is depicted in Figure 3.4.

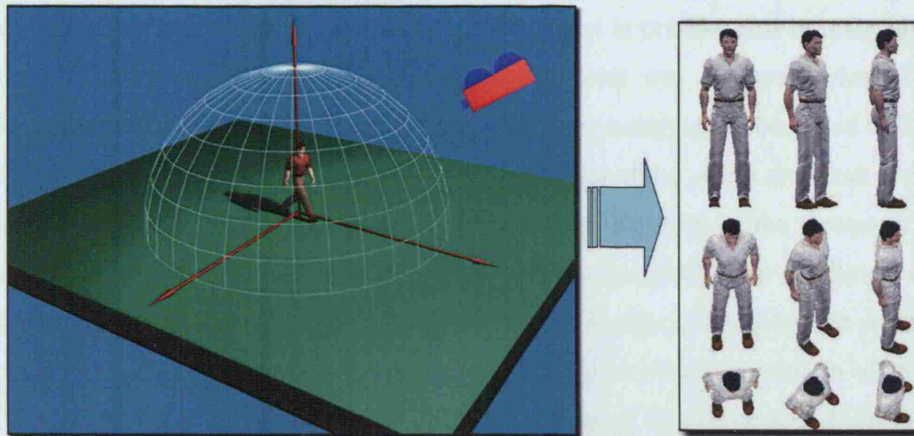

Figure 3.4 - Taking a discrete set of images of a polygonal character.

A sampled hemisphere is used to capture the character as seen from several positions around and at various elevations. At run time, depending on the view position with respect to each individual, the closest match will be selected and displayed on a properly placed image projection plane. The sampling process does not involve just image capturing: some geometrical data needs to be associated to each impostor, to store the vertices position of the impostor on the projection plane, and the normal direction to the impostor plane expressed in local coordinate, that we can use for impostor lighting computation as Section 3.9 will show. Also, when impostor images are not placed in texture space on a regular grid, an appropriate (u,v) coordinate set need to be stored for each sample; this is the case in our implementation, as the image samples are tightly grouped together in texture memory to save space.

To summarise, a generic sample is represented by the following set of information:

- an RGBA image, sampling the object from a given view direction;
- 4 vertices, to properly place the impostor in space;
- 4 sets of (u,v) coordinates, used to apply texture mapping to the impostor image;
- a three-dimensional vector representing the normal to the impostor plane.

It must be noted that the number of object samples needed to represent an objects with an acceptable precision is largely dependent on the particular application scenario, and can be adjusted to accommodate specific needs. Clearly, larger amounts of image samples correspond

to larger memory requirements, and a trade-off between quality and memory should be defined. Our experiments with crowd visualisation suggest in practice that in many situations an acceptable sampling density for a polygonal character was achieved when the view directions around the model are discretised using a regular subdivision composed of 32 views around replicated for 8 different elevations. Sample images of the object are taken from these directions, each row of images corresponding to a different elevation of the viewpoint. As the impostor representation is used only to replace distant geometry, the resolution of each impostor image can be limited to the size that the object would occupy when the replacement takes place. In many experiments we used for this reason a maximum resolution of 256 x 256 pixels for each object sample. To each image correspond an *activation distance* of the impostor representation, that is the distance at which the geometrical object is replaced by the impostor in our rendering system. As a way to reduce memory requirement, we also made use of mirroring of some of these samples at run time, taking advantage of the pseudo-symmetry of the human body in certain postures.

At the end of the sampling procedure, all the images are stored in video memory as RGB textures, so standard texture mapping can be used to project the impostor image on the impostor projection plane. Chapter 5 reports in detail on how the resulting database is managed and how the images are grouped together to minimise texture memory requirements and maximise rendering speed.

3.4 Impostors run - time rendering

Once the database grouping all the samples is created, the impostors can be used at run-time to replace the geometry of the distant individuals. The visualisation of a single character is obtained by projecting, using texture mapping, a sample image contained in the impostor database over a single polygon having the right space location, dimensions and orientation. This process is rather computationally intensive, as in a crowded scene it involves computing these values for a multitude of entities and on a frame-by-frame basis: provided that each individual can have an arbitrary rotation around the y-axes (vertical), the viewpoint-to-impostor direction is first transformed in the coordinate system of the impostor. The resulting view direction is then discretised following the same rules and algorithm used in the sampling phase, and two indices are computed reporting the closest sampling direction and elevation. These indices are used to retrieve from the sample database the impostor image representing

the closest match for the object, together with information needed for its proper placement in 3D space, such as orientation, (u,v) texture-space coordinates and projection plane displacement distance (the meaning of the last term will be exposed in the following paragraphs). Even if these operations are performed at every frame and for a very large number of individuals, in practice the overall process (summarised in Figure 3.5) accounts only for a small fraction of the rendering time for each frame (as the performance tests of Chapter 6 will show), making feasible the display of crowds exceeding ten thousands individuals on commodity hardware.

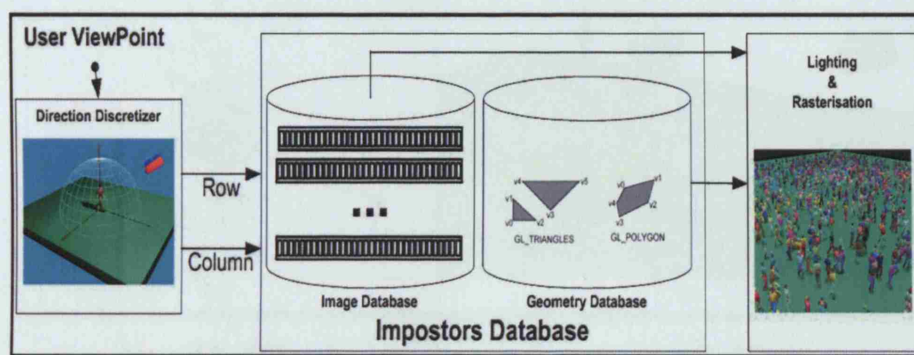

Figure 3.5 - Conceptualisation of the impostors run-time rendering pipeline.

3.5 Impostors and billboards

Billboards and impostors share the same idea of replacing a complex geometry with an image, but there are fundamental differences between these two entities. Billboards are frequently used in real-time applications as a simple form of image-based representation aimed at replacing with a single textured polygon objects that have cylindrical symmetry. A billboard is in practice a textured quadrilateral presenting the image of the object that it replaces, hinging around a vertical axis through its centre in order to be always facing the camera. This is different from what a proper impostor presents. More precisely, moving the view point around does not produce any rotation of the impostor, that always maintains the orientation (in the world space coordinate system) computed during the sampling procedure; it is only when the camera moves too far away from the original sampling direction, that the impostor is eventually replaced by another one, that having a different orientation better represents the object from the current point of view. So, in the case of the impostor a generic 3D model is

not represented by a single hinging image, but rather from a collection of static quadrilaterals, with only one active at any given time, depending on which sample is the best approximation given the current view direction. Finally, in the case of impostors the image plane can present a displacement from the centre of its local frame of reference, something that does not happen with normal billboards. Such displacement is exaggerated for illustration purposes in Figure 3.6 (b).

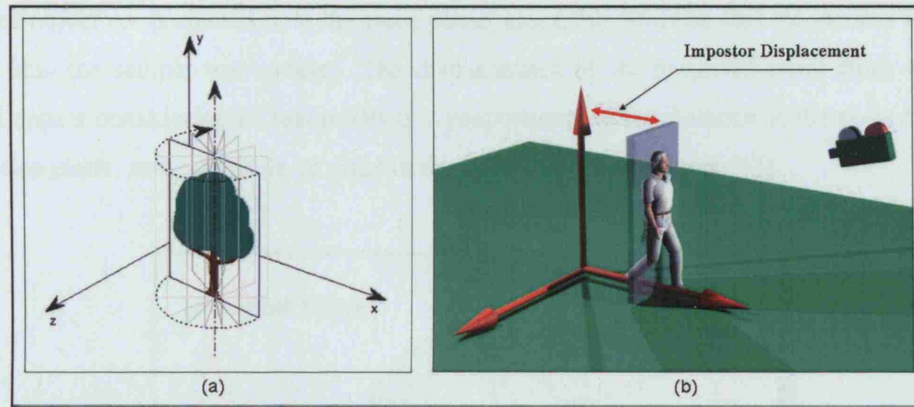

Figure 3.6 - Billboards (a) and impostors (b) are conceptually different. Billboards hinge around a central vertical axis and are used as a replacement for objects with cylindrical symmetry. Impostors do not rotate, they approximate the object only for a small number of view directions, and the image plane can be displaced from the origin of the local frame of reference.

As Section 3.7 will show, the fine tuning of this displacement can be used to attenuate some of the visual artifacts introduced by the use of impostor, and has important implications both in the way impostors need to be organized and managed, than in the kind of final image they can produce.

3.6 Visual artifacts caused by impostors

Lighting and shading aspects aside, an impostor can be considered an exact visual replica of the original object only when the position of the user view-point exactly matches the one used in the capturing process; in that case an impostor is indistinguishable from the replaced geometry. As discussed in Chapter 2, the more the view-point moves away from the position used in the sampling phase, the greater visual artifacts may arise in comparison with what would be generated by a real 3D representation. More generally, when IBR techniques are

used to render a complex object, two common forms of artifacts may arise: missing data due to incomplete handling of inter-occlusion may cause black regions to appear, and 'popping' effects may occur at the switch of different images or when the image samples are warped and/or blended to obtain the replacing image.

In our impostor approach we try to maximize the rendering speed using minimal geometric complexity for each entity, and we use a single planar polygon as the plane on which to project a sample. The popping artifact arises then because all the points on the surface of the sampled object are projected onto the same plane, along the direction that the camera is facing at the time the sample was created. The displacement of the projected point from where it should appear considering its real position is proportional to the distance of the point from the projection plane, and constitute an error in the rendered image (Figure 3.7).

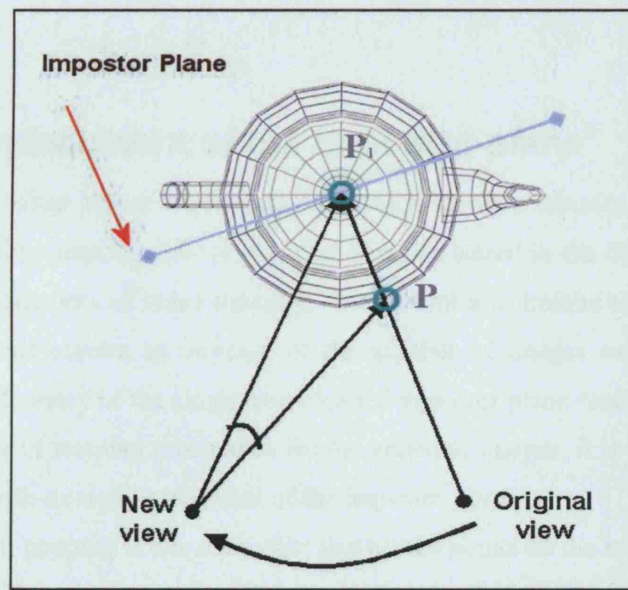

Figure 3.7 - Projecting points on the impostor plane.
When the viewpoint moves there is a mismatch between the position expected for the point P and the position of its old projection P_1.

A technique at times used to alleviate this effect is to simulate a 'fake' morphing operation blending two 'adjacent' samples together [Meyer01]. Unfortunately, the alpha-test operation used in the multi-pass rendering strategy of our system to improve the crowd variety (introduced in Section 3.8) prevents us from using blending.

Due to the absence of warping procedures, when only a limited number of samples are stored in memory, the image produced by an impostor may at times turn out to be slightly imprecise. These artifacts are hardly noticeable during the lifespan of a single impostor, but can be detected when there is a switch between different samples, as the switch can cause quite a few of the features to suddenly change position. When impostors are used for crowds, focusing the attention on a single virtual human when the camera moves, the rapid succession of different samples can sometimes be detected - obviously depending on the sampling rate used. It must however be noticed that, when the scene is populated with crowds composed of thousands of animated individuals, each of them performing a key-framed animation, such popping effect becomes almost unnoticeable. In the following paragraph we show that even if this type of artifact cannot be completely removed, given the absence of sophisticate warping procedures, it is possible to attenuate them with an optimal placement of the impostor projection plane.

3.7 Optimal placement of the impostor plane

Since the popping effect occurs when there is a switch between samples, it can be obviously reduced increasing the total number of impostor samples stored in the database, but this may require very large amounts of video memory. We present a technique to reduce this kind of artifact that does not require an increase of the number of images sampled and does not compromise the efficiency of the single-non blended impostor plane rendering. We claim that even in the absence of warping procedures for the impostor images, it is possible to attenuate popping artifacts with a careful placement of the impostor plane.

As discussed above, popping is due to the fact that all the points on the surface of the sampled object are projected onto a single plane. This plane is normally orthogonal to the direction that the camera is pointing when the sample is created. Obviously, as the camera position changes, the projection of such points on the impostor plane might change, so that the current impostor is no longer an exact replica of the object appearance. The amount of error generated for a generic point on the object surface is proportional to the distance of the point from the projection plane. Different placements of the impostors plane with respect to the cloud of visible samples can then lead to different amount of error. A common choice for the impostor projection plane is the one passing from the centre of the replaced object and perpendicular to the view direction from which the sample image was taken. The placement of this plane does

not take into account the shape of the object nor any kind of self-occlusion that could be present in the image. A different approach has been used here: given an object and the camera position from where the sample image is created, we search for the projection plane passing through the object that reduces the sum of the absolute distances of the sampled points and the projection plane. Figure 3.8 shows some choices for the projection plane that have been used.

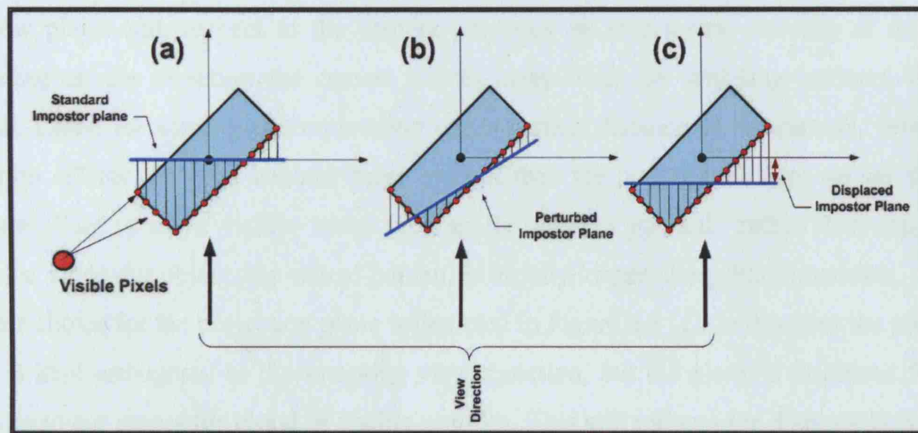

Figure 3.8 - Different choices for the impostor projection plane.
(a) The projection plane is at the origin of the object reference frame and perpendicular to the view direction, (b) the projection plane cuts the visible pixels across to minimise the distance from each visible sample, (c) the projection plane is perpendicular to the view direction but displaced from the origin of the object reference frame

In particular, Figure 3.8 (b) reports the choice for the plane that generates the minimum amount of displacement error, referred here as a *perturbed impostor plane*. To compute the equation of this plane, we project back in 3D space all the pixels in the impostor image, computing in this way the 3D coordinates of the object's visible samples only. This can be done in several ways. Since OpenGL is used in the sampling phase of the impostor representation, we can compute the 3D coordinate of each pixel composing the image using the glUnproject API command, and taking advantage of the availability in the depth buffer of a z-value for each visible pixel. Once the cloud of 3D points forming the visible samples with respect to a particular direction and distance of the camera is available, we search for the projection plane that, passing through the cloud could minimise the projection displacement effect. A way to compute this is to apply a Principal Component Analysis (PCA) to the cloud of 3D points, identifying the two principal eigenvectors, and using them as the principal

directions describing the projection plane. Using such a plane as the projection plane for the visible samples potentially results in a significantly better approximation of the position of the pixels in respect to their real positions in 3D as points, and we tested the use of this plane in our development system. However, experiments revealed that this does not translate in practice to a higher quality visualisation of the resulting impostors: when using the perturbed plane to project all the pixels, other visual artifacts arise, as the non-orthogonal orientation of this new plane with respect to the camera produces an asymmetric warping of the image depending on the direction the camera moves away from the sampling position. In some extreme cases, for some plane orientation and a certain distance of the camera, perspective distortion effects can even become more evident than the popping artifacts we are trying to alleviate. This is more visible when moving the camera upwards rather than around the impostor, since our object (the virtual human) is usually longer along that dimension.

A better choice for the projection plane is depicted in Figure 3.8 (c). In this case the projection plane is kept orthogonal to the sampling view direction, but the plane is displaced from the origin so to cut across the cloud of visible samples. This still reduces the distance between the visible samples and their projection, and does not produce the asymmetric warping effect of the perturbed plane. We found this to be the best choice in term of overall visual results: this plane is termed the *displaced impostor plane* in this thesis, and we have used it extensively in our development system.

3.8 Colour modulation of the impostors

Visual variety is very important when rendering a crowd composed of thousands of individuals, since it would be unrealistic if they all looked the same. This is a common problem in crowd rendering, independently of the form of representation used for the characters: designing hundreds of geometrically and visually different meshes is a daunting task, and the memory requirements that would arise from storing them in memory would be considerable, even in the case of polygonal meshes. A way to improve the perceived variety without having to change the geometrical shapes of the individuals in the crowd is by controlling and varying the colours of clothes and different body parts of each individual [Ciechomski04]. Figure 3.9 shows how much the visual variety of three polygonal models can be improved simply using colour modulation.

Figure 3.9 - Simulating variety using colour modulation (image S. Noverraz).

While modulating the material colours of individual parts of an object having a polygonal representation is relatively straightforward, things are more difficult in the case of impostors, as all we have in that case is a precomputed image of the object. The intrinsic inability to modify the content of an impostor sampled image and the resulting inability to use colour modulation as a way to contribute to visual variety, has traditionally represented a very big drawback of the impostor representation. A novel technique is presented here that, storing additional information in the Alpha channel of the impostors samples, allows the fine-tune colour modulation of isolated regions in the samples images. This is achieved by introducing a multi-pass strategy in the impostor rendering procedure, and exploiting an alpha test operation to control the differences between consecutive rendering passes.

The technique works as follows: each impostor sample is stored in texture memory as a 4 components RGBA image. The Red, Blue and Green colour channels are used to store the colour of the pixels, while the Alpha channel value is used to assign a region ID for each pixel on image. Assuming 8-bits resolution for the alpha channel we can identify up to 256 different sub-regions in the image. Pixels having the same alpha value are assumed to belong to the same area (or sub-object image), see Figure 3.10. Having defined an alpha value for each pixel, we can use the standard OpenGL alpha-test feature: the alpha test discards a fragment

conditional on the outcome of a comparison between the incoming fragment's alpha value and a constant value. The comparison is enabled or disabled with the generic glEnable and glDisable commands using the symbolic constant ALPHA_TEST.

We combine alpha-test with multi-pass rendering: by changing the alpha test value pass by pass (to select the region of interest inside a single impostor) a final, composed image gets rendered in several rounds, overlapping the various regions one at the time. Since for each rendering pass we can change the base colour of the impostor polygon, this technique results in the ability to control independently the colour generated by each rendering pass, effectively modulating the final colour of all the pixels lying in the same region.

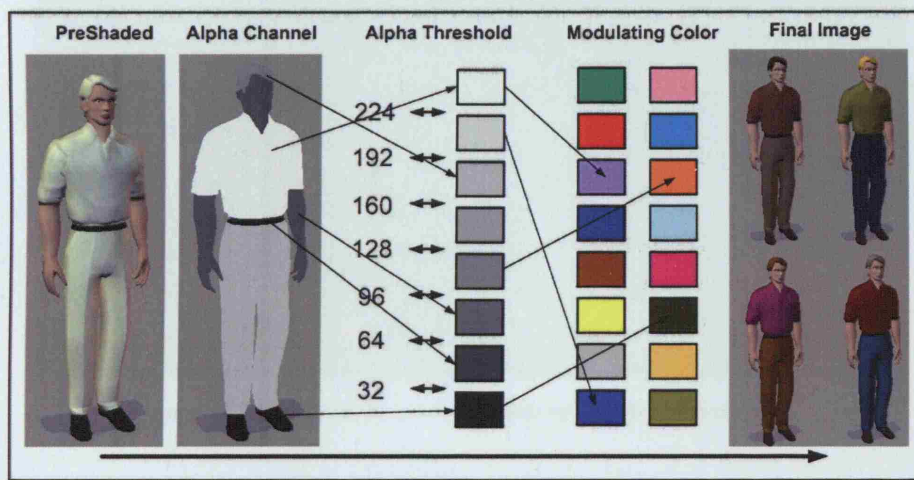

Figure 3.10 - Fine-control of the impostor colours.
Alpha-testing and multi pass rendering can be used to select different regions of the impostors images and modulate their colour.

It must be noted that modulating the colour of each region is not mandatory; in our implementation we have both regions where we store the final desired colour of the impostor and regions where we just store a grey-scale image of the impostor. We can avoid modulating the colour of the former, and concentrate on the colour changes only in the grey-scale regions of the image. Figure 3.11 shows the result obtained for impostors having 4 regions of interest: skin, hair, shirt and trousers, resulting in 4 rendering passes for each individual. More passes can be carried out since, as we said, we can identify up to 256 regions. However, impostors multi-pass rendering can be fill-rate intensive; to avoid a decrease in rendering speed we can limit the number of independent regions required to be controlled. Also, for impostors that are

far for the view-point we can avoid rendering the smaller image regions, as they give a negligible contribute to the final image. This is similar to the concept of Levels of Detail for polygonal models.

Figure 3.11 - Effect of the random colouring of different body parts.

3.9 Interactive impostor lighting

As the tests in Chapter 6 will show, the basic techniques illustrated in the previous paragraphs allow for a very efficient crowd rendering, where populations exceeding 10,000 individuals can be rendered in real-time on commodity PCs.

So far we have illustrated the tasks of rendering and colour modulation, but nothing has been said yet about how illumination (both static and dynamic) can be taken into account. There are situations where the precomputed lighting stored in the impostor images imposes severe restrictions to the realism of a simulation, and the ability to introduce interactive dynamic lighting would be desirable. For example, in a complex urban environment with a mix of short and tall buildings, regions with different illumination characteristics are present, and a realistic simulation should take this into account. In the following paragraphs, novel methods are proposed to handle dynamic illumination using impostors, showing that the use of an

image-based representation makes it possible to achieve advanced lighting effects, with a further discussion on global illumination and shadows postponed to Chapter 4. We begin the discussion introducing a basic but effective dynamic lighting model that proposes uniform light intensity across each impostor, and continue presenting a more sophisticated per-pixel lighting techniques. The discussion will show that, far from being a restrictive technique, impostor shading can, in certain situations, be very flexible.

3.10 Approximate dynamic lighting

When a 3D object is replaced by an impostor, it normally has a fixed lighting, that is the lighting active when the image sampling process took place. This lighting information, being pre-encoded in the samples, cannot normally be changed at run time; some work on this topic has been proposed in [Meyer01], where a hierarchy of bidirectional textures and cube maps are used to render image-based trees covering a landscape with consistent shading and shadows. Conversely, when populating virtual cities with crowds, it is desirable that dynamic lighting conditions are taken into consideration, as when the individuals navigate in the environment appropriate variations of their illumination would enhance the overall scene realism. Even if the fine-detail illumination of an impostor is usually pre-stored into the image samples, in many situations a general attenuation of the overall colour intensity of the impostor can be sufficient to simulate an attenuation of the ambient light intensity due, for example, to the passage of an individual in a narrow and dark alley (see the example in Figure 3.12).

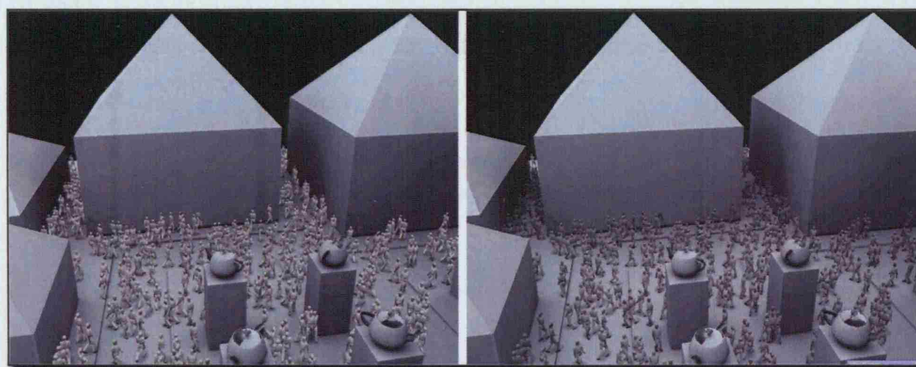

Figure 3.12 - Global illumination effects can increase the realism of a scene. Adjusting the colour intensity of each impostor to the ambient light intensity (right) harmonize the overall visual appearance of the image.

If this approximation is sufficient, we can very easily replicate on per-impostor basis the mathematics of OpenGL lighting. In the OpenGL lighting process, the possibility of one object blocking light from another is not taken in account (so shadows are not automatically created). Also, it is assumed that illuminated objects do not radiate light onto other objects. As a result of this assumption, an OpenGL-compliant rendering pipeline computes the colour produced by lighting a vertex as the sum of several terms: the final vertex colour is the sum of the material emission at that vertex, the global ambient light scaled by the material's ambient property at that vertex and the ambient, diffuse, and specular contributions from all the light sources, properly attenuated. After lighting calculations are performed, the colour values are clamped (in RGBA mode) to the range [0,1]. We can perform similar computations for the impostors assuming that the light intensity that we compute will modulate the intensity of the whole image. If we assume that impostors do not present light emission or specular effects, we can compute the final colour of each impostor pixels as:

$$FinalColor = ImpostorImageColor * ImpostorLight\ Intensity$$

If we have *n* light sources in the scene, we can sum their effects in the following way:

$$ImpostorLightIntensity = GlobalAmbientIntensity + \sum_{i=0}^{n} Att_i * (AmbientTerm_i + DiffuseTerm_i)$$

and where ***Att*** is the light intensity attenuation factor due to the distance of the impostor from the the various light sources. OpenGL attenuation is referred to the distance from a vertex to the light source and its function has constant, linear and quadratic components

$$Att = \frac{1}{(K_c + K_l * d + K_q * d^2)}$$

where d is the distance between a vertex and the light source. Attenuation can then be computed on the fly for each impostor, and the resulting *ImpostorLightIntensity* can be used to modulate the overall impostor colours, to produce local lighting effects such as those depicted in Figure 3.13, and 3.14.

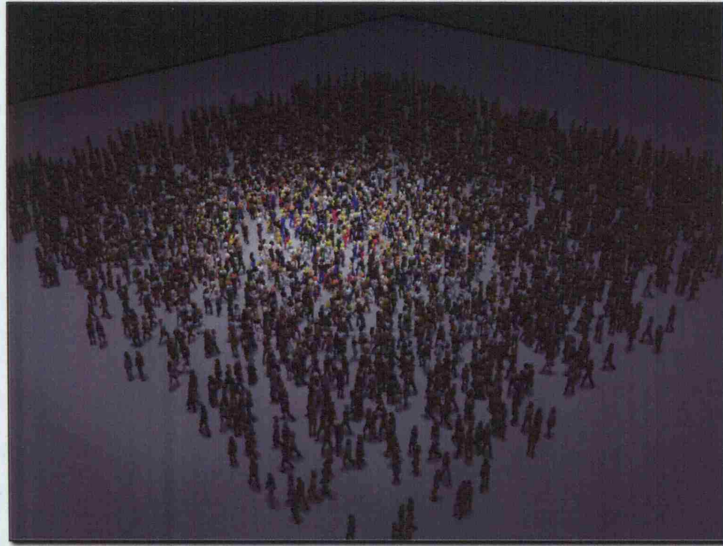

Figure 3.13 – Simulating the effects of a (white) local light source.

If more than one light source is present in the scenario, the final intensity can simply be computed as the sum of the single contributions and using this method, coloured light sources are possible as well. Also, dynamic effects such as moving light sources and spot light simulation can be achieved. An example is shown in Figure 3.14, where 3 coloured lights and one white light of the same intensity were used, placed in different locations.

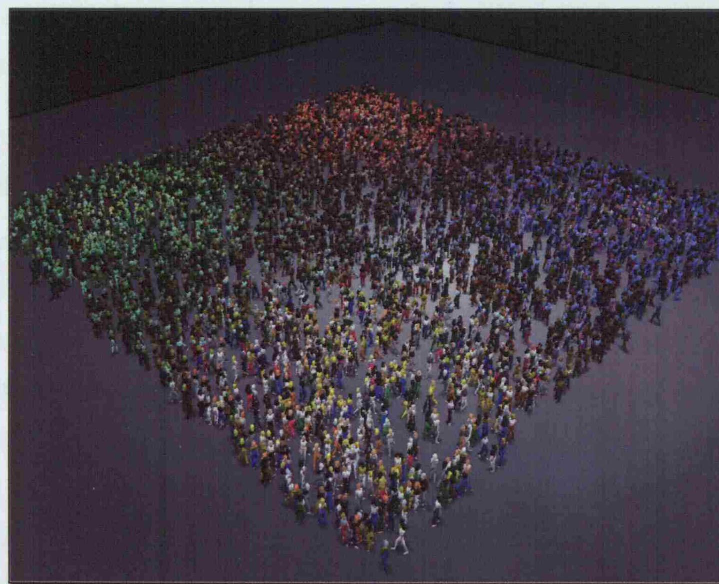

Figure 3.14 - Simulating the effects of multiple coloured light sources .

Even if approximate lighting using colour intensity modulation can increase the realism of a scene, it constitutes only an approximate solution to the problem of impostor dynamic lighting. While this may be perfectly acceptable in many applications, there are certain situations where a more accurate lighting computation is preferable. In particular, when impostors are very close to a local light source, the approximate method introduced in the previous paragraph can result in image artifacts at the switch from the polygonal representation to the image-based counterpart or vice-versa. As discussed previously, such operations can introduce 'popping' artifacts in the rendered image, that we can classify in two categories: the first has a geometric nature, and is due to the misalignment in the final image of the pixel location computed with the proper geometric transformations and the location of the pixels generated by the use of a single-layer impostor; the second form of artifacts has a lighting nature, and it is due to the possible clash of illumination variations between the polygonal mesh and the impostor image, as the latter has the detailed illumination pre-encoded in the sampled images. On the other hand, the polygonal representation can reflect any lighting condition through the standard lighting model of OpenGL, as lighting is in this case computed dynamically for each polygon or vertex. The switch between images and polygons then produces a shimmering of the object, and for some lighting conditions this can be more distressing than the popping due to geometric misalignment. The difference between these two lighting effects can be avoided using precomputed anisotropic illumination for both the impostors and the polygonal characters. However, the simulation of any dynamic effect becomes in this case impossible.

3.11 Impostors per-pixel lighting

To allow the use of more complex and accurate lighting effects in applications where this is required, we present a novel illumination method for impostors that takes advantage of the OpenGL 1.3 support for per-pixel dot product to achieve fully dynamic impostor lighting [Segal01]. With an appropriate use of per-pixel dot product computation, the intensity of each pixel in the impostor image can be controlled individually, and it is possible to relate the intensity of the light on each impostor pixel to the dot product between the pixel normal and an incoming light direction. In this way, it is even possible to reproduce on an impostor image a per-pixel computation of the standard OpenGL lighting equations. In order to take advantage of OpenGL per pixel dot product, we need first to change the type of information stored in the

RBG channels of the impostors image database: instead of storing a grey-scale image holding a fixed lighting information as in the case of static lighting, we store now the normal associated at each visible pixels (Figure 3.15). We compute such a normal in the preprocessing phase, using the geometric data associated to the object. According to the OpenGL 1.3 specification, the spatial components *x*, *y*, and *z* of the normal of each pixel can be encoded and stored in the texture RGB space using the following convention: x is mapped in the Red channel, y and z are mapped to the Green and Blue channel. Since OpenGL1.3 only has the ability to store up to 8 bits for each colour channel of an RGB texture, normals stored as RGB values undergo a quantization process. Also, it must be noticed that the normals stored as RGB textures can also take advantage of OpenGL1.3 texture compression.

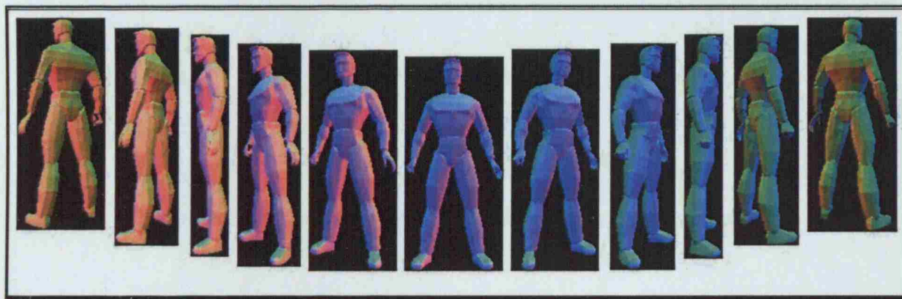

Figure 3.15 - Storing per-pixel normal information in the impostors image samples. Each pixel in the image stores the (x,y,z) normal components in the 8-bits RGB channels.

Once the colour channels are filled with the normals' information, we can still add to the image an alpha channel, and store there the same information about pixels regions reported in Section 3.8, as the multi-pass technique to control impostors colours is compatible with the per-pixel lighting approach. This information is then used at run time with the same multi-pass rendering strategy discussed before.

3.12 Per-pixel lighting equation

We now examine how we can use the pixel normals stored in the compressed RGB channels of the textures to achieve per-pixel lighting at run time, and how we can make the resulting equation very similar to the standard local lighting computation taking place in the OpenGL pipeline. If we consider the local reflection model used by OpenGL (leaving temporarily aside

the issue of colour and distance attenuation), we can write down the light intensity equation in the usual form:

$$I = A + K_d \cdot \vec{L} \cdot \vec{N} + K_s \cdot (\vec{R} \cdot \vec{V})^n$$

where n is used to model the specularity of a surface. Being the mirror direction $\vec{R}$ computation computationally expensive, the equation is normally considered in the following simplified form:

$$I = A + K_d \cdot \vec{L} \cdot \vec{N} + K_s \cdot (\vec{H} \cdot \vec{N})^n$$

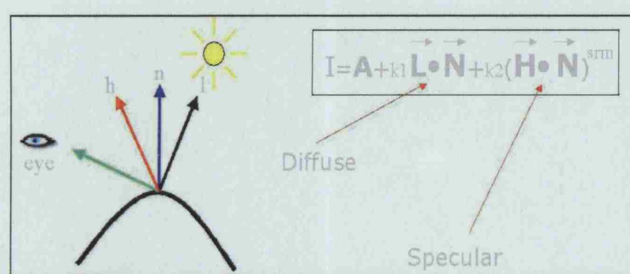

Figure 3.16 - The standard OpenGL lighting equation.

where $\vec{H}$ is the halfway direction between the light direction L and the viewing direction V. We can now use OpenGL DOT3_RGB_ARB texture parameter to perform the equation's dot products on a per-pixel basis and perform multiple rendering passes, accumulating on the frame buffer the partial results of the intensity equation. Also, we can use multi-pass rendering to sum-up all the components and compute the final value of each pixel intensity as well as to raise the specular component to a certain power. To accomplish this, and in accordance with the OpenGL specifications, the RGB codification of vectors L and H are used as the fragment colour of the polygon. Our method uses a very important assumption: as the per-pixel computation of L and H is costly, we consider them constant over each impostor. With this simplification, it is necessary to compute L and H on a per-impostor basis only, depending on the current impostor position and orientation with respect to the considered light source, but it does not reproduce exactly the effect of a local light source. Still, it's very difficult, if not impossible, to notice the difference when hundreds of individuals are going around on screen. To accumulate in the frame buffer all the lighting component we use at present 5 passes per impostor. The first 3 passes are used to compute the specular component and to rise it to the

power of 3 (greater values are possible, but at the cost of slowing down the rendering process); the next pass is used to compute and add the effects of the ambient component, and a last one to compute and add the effect of the diffuse component. Using intensity modulation of the of the polygon colour, we also introduce in the equation the attenuation factors K_d and K_s , so that the effects of local light sources can be simulated.

At this point, the frame buffer contains the grey scale image of the impostors representing the correct illumination with respect to the actual light position and surface propriety as reported in Figure 3.17.

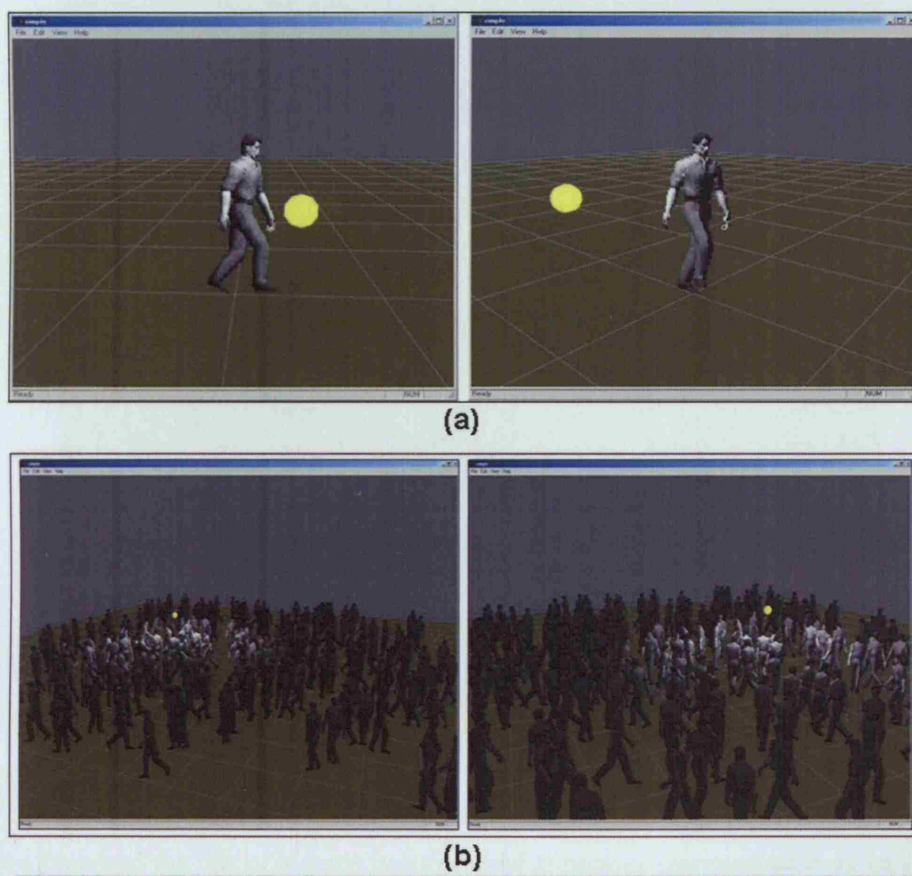

Figure 3.17 - Impostor per-pixel lighting.
(a) per-pixel lighting applied to a single impostor (b) per pixel lighting on a crowd.

The process can also be repeated to accumulate the effects of multiple light sources; in this case the limited numerical precision of the frame buffer should be considered, as the standard

8 bits per colour channel could present some numeric precision issues. Using a uniformly coloured texture in the second texture unit, we can modulate the colour of the resulting intensity computation, making it possible to simulate even coloured lights. Once the illumination is in the frame buffer, we can still use several additional passes (4 in our case) to modulate different regions with different colours, using the alpha test technique described in the previous paragraphs. Figure 3.18 shows the results of the described process, starting from the light intensity calculation to the final colour modulation using `the alpha-test`.

Figure 3.18 - Final effects of per-pixel lighting on an impostor-based crowds.

3.13 Summary

We have proposed the use of a *static, unstructured impostors representation* as an effective mean to render densely populated large scale environments with crowds in real-time. Impostors have a flexibility that is often underestimated, and we proposed several management techniques that allow for the display of a large number of different animated people, using fully precomputed animated impostors. The fundamental advantage of impostors is that the rendering time of each character is independent of the complexity of its

polygonal model; this makes it possible to render thousand of agents at interactive frame rate. The method improves pre-existing methods in several aspects. The choice of the impostor plane is adapted to the object to render, thus reducing popping effects when changing view. We tested several options: the plane orthogonal to the view direction and passing from the geometrical centre of the object, the plane orthogonal to the view direction and passing from the centre of the visible samples (we found this to give the best results), and the plane minimising the sum of the squared distance of each visible sample (computed using PCA). We use a multi-pass algorithm that allows visual variety using colour modulation, exploiting OpenGL Alpha Test and the alpha channel content to control the colour of different regions of the body, and a technique to approximate dynamic lighting on an impostor-by-impostor basis. Finally, we introduce a novel way to store and represent per-pixel impostor data that, through the use of hardware accelerated dot product operation, removes the fixed lighting limitation previously associated to impostors, making it possible to achieve per-pixel dynamic lighting effects. Our implementation has only minimal OpenGL1.1 requirements (OpenGL1.3 for per-pixel lighting), so it can be ported to basically every available graphic system. In the following chapters we will show how global lighting effects as well as shadow effects can be achieved, and how this form of IBR can coexist with standard polygonal rendering inside the same scene-graph management system.

Chapter 4 - Placing crowds in Virtual Environments

Figure 4.1 - An image from the movie 'Shrek' (Dreamwork – 2004).

This Chapter will illustrate how impostor based crowds can be used inside polygonal scenarios, obtaining what in practice is an hybrid approach for complex Virtual Environments rendering. Issues such as impostors casting shadows on a polygonal model or receiving shadows generated by other scenarios elements will be discussed, as well as visibility computation for crowds and basic collision detection tests that characters perform in order to navigate in the polygonal model. The following will show that, far from being a restrictive technique, using impostors to populate conventional polygonal models can successfully lead to a high degree of visual realism whilst retaining an high frame rate, something that would not have been possible using plain polygonal rendering alone.

4.1 Using animated impostors in a polygonal scenario

Animated impostors are an effective means of rendering thousands of characters in real-time, but the differences between this approach and that of standard polygonal rendering must always be carefully taken in consideration when using both approaches inside the same application. Impostors are fundamentally 2D entities with no retained information about the volume of the original object they represent. When used to populate large urban-like models (Figure 4.2), their different nature has several important implications for the way that they can coexist with polygons, both for rendering-related tasks (such as illumination and shadowing effects) and for more general tasks (such as collision detection, behaviour and navigation).

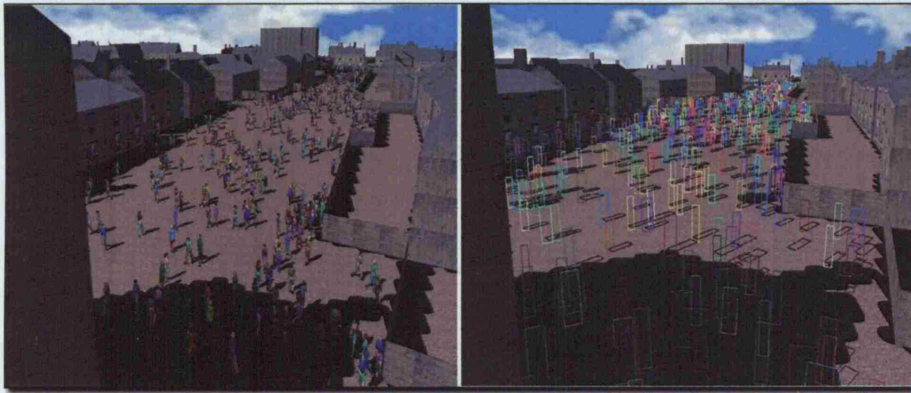

Figure 4.2 - High-performance crowd rendering using textured impostors..

The structure of a city is a perfect example of an environment where many complex interactions between these different entities can potentially take place. The ability to perform collision detection between the impostors and the scenario, or between the impostors themselves, is a fundamental requirement in order to simulate the crowd dynamic behaviour. Also, shadowing and illumination interactions exist between the crowd and the environment that should be considered, and even the ability to perform visibility computation, i.e. taking into account the occlusion effects of the environment on the impostors, may be crucial, especially when the goal is the visualisation of virtual cities where the population is counted in hundreds of thousands. Some of these forms of interaction have been studied in previous research in the context of large scale visualisation, but the hybrid nature of rendering introduced by the use of impostors imposes a careful rethinking of existing techniques. The following short list summarises what has to be kept in consideration when planning to use

animated impostors inside a polygonal scenario:

- As impostors do not retain much of the geometrical information about the objects they replace, there are implications for the collision detection task between impostors and the surrounding scenario.
- Casting shadows of an object onto another object normally requires some sort of volume information that our impostors do not have. As the presence of shadows cast by the crowd onto the environment is important for realism, a specific approach for impostors-to-polygons shadow casting is needed.
- Shadows cast by the environment on the population are equally important. Again, as our impostors do not retain true volume information, many existing approaches are not directly applicable.
- When dealing with large crowds only a small fraction of the overall population may be visible; occlusion effects of the environment on the population need to be taken in account, as well as the fact that crowds have a dynamic nature, and the individuals are continuously moving around. More importantly, given the extreme 'rendering lightness' of the impostors, the run-time test to decide the visibility of a single individual needs to be extremely efficient, or any speed advantage would be lost.

The following paragraphs present how the work address these points.

4.2 Illumination of a crowded scene

In Chapter 3 we proposed a technique to compute the effects of positional light sources on the crowd, taking into account per-pixel lighting where possible. As in the standard OpenGL lighting computation, our approach assumes that no occlusions are present between the light sources and the crowd individuals. This is frequently not the case in Virtual Environments: once a crowd is used to populate an urban scenario such as the 3D model of a city, where a mixture of short and tall buildings are present, the illumination effects become more complex, and variations in the intensity of ambient light in different part of the scenario may arise (for instance narrow alleys tend to be darker than the large streets – Figure 4.3). Also, when simulating direct sunshine the tall buildings should cast shadows on the streets. It is desirable that these different lighting conditions are taken into consideration in crowd simulation, as

ambient light intensity variation and shadows of the environment on the crowd (and of the crowd on the environment) greatly contribute to the overall perceived realism of the scene.

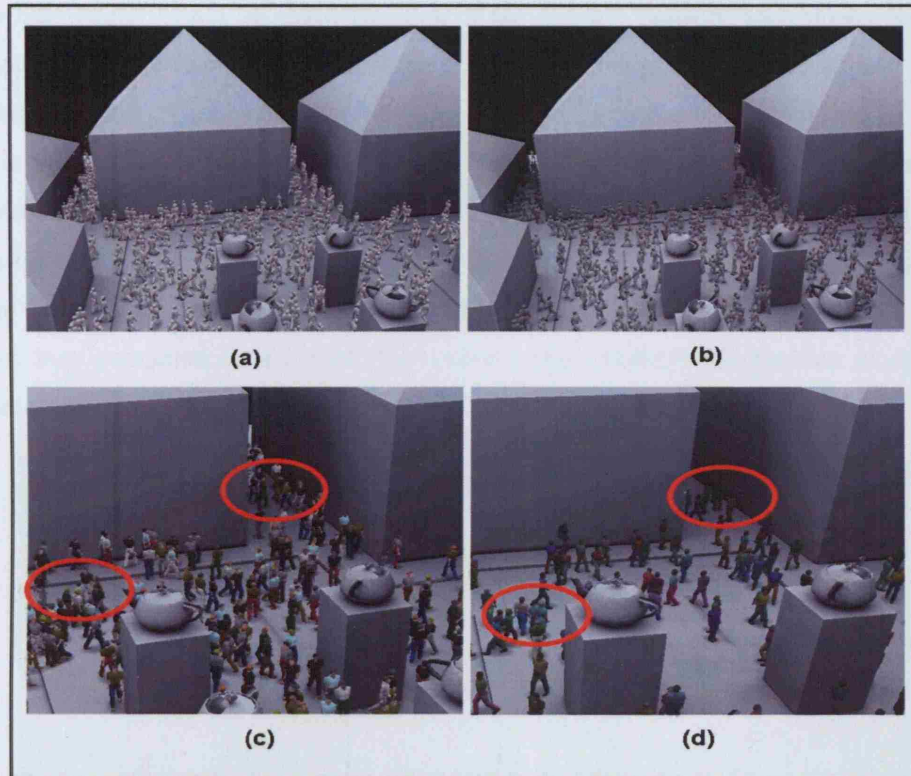

Figure 4.3 - Global illumination effects on crowds.
Our impostors-based crowd in a scenario without (a,c) and with (b,d) ambient lighting.

The following 3 paragraphs will expose several techniques to add global illumination effects to the rendered scenes, trying at the same time to meet the always demanding requirement of interactive rendering of a large crowds. It must be noted that the applicability of some of the techniques here presented is not restricted to the impostor approach only, and could also be used with other types of representation.

4.3 Ambient lighting for a crowded Virtual Environment

As discussed in Chapter 3, the intensity of an impostor image can be modulated on a per-character basis, to approximate the illumination effects of local light sources; this simplification is possible because, when a dense crowd is scattered around an urban

environment, the artifacts caused by this approximation are hardly noticeable. A similar approximation can be used to simulate the variation of light intensity that the animated agents composing the crowd undergo while navigating the environment. While in Chapter 3 we used for the light attenuation a quadratic function of the distance between each impostor and the light sources, in this case the attenuation is a function of the position of the impostor within the urban scenario. A technique that takes advantage of the mostly 2.5 D nature of a urban scenario is used to compute a 2D discretization of the light intensity over the scenario, in a similar fashion to the technique of precomputed lightmaps; the resulting data is stored in a 2D structure, named *ambient intensity map,* that for each cell of the discretisation reports the average intensity of the incoming ambient light. Figure 4.4 shows an example of such an *intensity map* computed for a model of a square using a skylight illumination model inside 3DStudio Max.

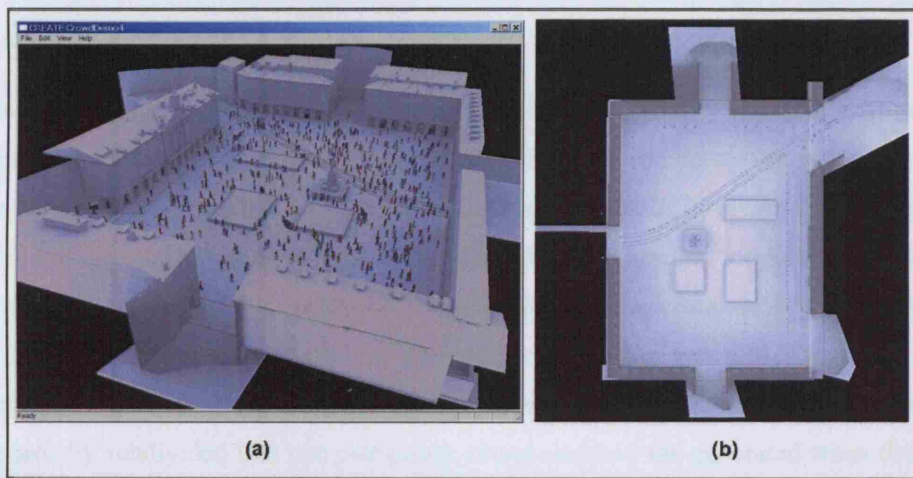

Figure 4.4 - An urban scenario (a) and the resulting light intensity map (b).

Once this map is available, the contained information can be used to modulate the colour intensity of each impostor on the basis of its position in the scene (Figure 4.5). Accessing intensity information using a regular 2D cell subdivision is extremely fast and efficient, so this technique does not degrade much the rendering performance even in situations with thousand of individuals (measured in Chapter 6). Since large scenarios might need very large intensity maps, a hierarchical memory structure and simple image compression (for regions having constant or near-constant intensity values) could be used instead of a flat 2D organization, at the price of an increased number of memory accesses required to obtain the final value, although this feature has not been implemented in the test platform.

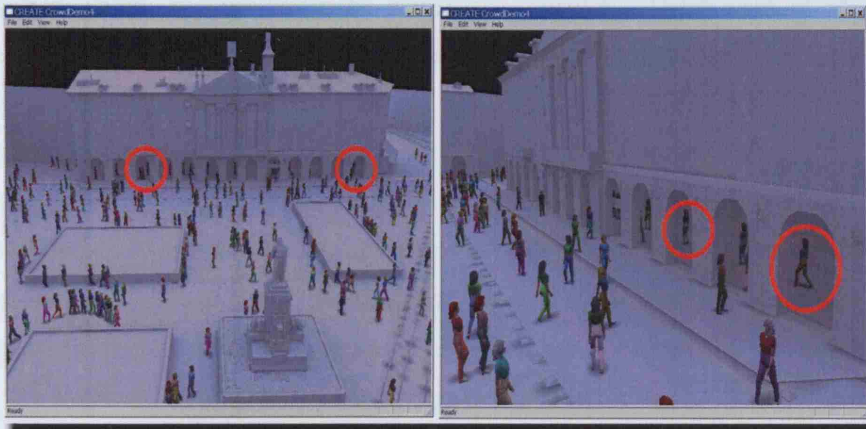

Figure 4.5 - The light intensity modulation at work on the impostors. Individuals walking in the darker regions are assumed to receive less light.

4.4 Shadowing

Shadows are useful in VEs for a variety of reasons: they help understand relative object placement in a 3D scene by providing visual cues, and they can dramatically improve image realism allowing the reproduction of complex lighting effects. A large number of algorithms exist in the literature for generic shadow generation; depending on the application the emphasis is placed either on a guaranteed frame rate or on the visual quality of the shadows. There is always a trade-off between realism and speed, and no general technique exists that can render physically correct shadows in real-time for every dynamic scene. These approaches can be broadly subdivided into two categories: *sharp shadows* are generated when the light is assumed to be at infinity or a point light source, while *soft shadows* are produced by taking the light source to be a finite area or volume. The latter case is more accurate and realistic, but requires more complex visibility computations, therefore it is not commonly used for real-time rendering of complex dynamic scenes. Scenes populated by a crowd of thousands of individuals poses novel challenges due to the high number of moving (and complex) objects casting shadows.

The existing general approaches to shadow computation were considered in this thesis for their suitability; one of the more effective approaches to real-time shadow mapping is still the fake shadows technique proposed by Blinn [Blinn88]. After rendering the model from a given view point, a special matrix is added on the stack that projects every vertex onto a plane (usually the floor). The scene is then rendered again painting the projected polygons as 'grey'

and semi transparent, to give them the aspect of a shadow. These kind of shadows are computed at each frame, and an additional rendering pass for the scene is needed for each receiving surface. Other methods, such as the Shadow Volume BSP (SVBSP) tree [Chin89] precomputes the shadows so that they just need to be rendered for each new viewpoint. The computation of a full solution on all surfaces is possible but is time consuming. Although incremental update is possible for a small number of dynamic objects [Chrysanthou97], this would be too costly for a highly dynamic setting such as a crowd navigating in a urban environment. In many cases, the number of extra shadow polygons that are needed for the display of shadows can greatly affect rendering performance. Techniques such as discontinuity meshing [Lischinski92] are the most accurate for shadow computation and rendering; they find not only the shadow boundaries but also the edges where the direct illumination on the surfaces varies discontinuously. However, they are slow to compute and render because of the resulting complex mesh. More approximate techniques that use texture mapping instead of partitioning the input polygons into a mesh [Soler98] are faster to render. However, they still cannot be computed in real time for any sizable scene and they require separate textures for different surfaces, so that for real time applications the rendering of shadows has often been restricted to sharp ones. Shadow volume methods [Crow77] were used to delimit a spatial region in shadow between the object surface and a receiver, and using the stencil buffer [Heidmann91], regions in shadow can be automatically detected by the hardware. An interesting alternative method for computing shadow planes was suggested by McCool [McCool00]. The scene is drawn first from the light position and the z-buffer is read; the shadow volumes are then reconstructed based on the edges of the discretized shadow map. One of the drawbacks of this method is that the shadow planes tend to be very large and they have a detrimental effect on the rendering time. Even if in practice this effect can be limited by using the method not for a complete solution but rather for shadows only from the 'important' objects, the discretization can still introduce artifacts.

Recently the use of graphics hardware to compute shadows at per-pixel level has been introduced, a technique that avoids some of the aforementioned shadows computation and storage problem. These methods are based on the concept of shadow maps [Williams78] and are pixel-based, mixing depth information provided both from the light source and the user's points of view. Shadow areas are determined by comparing the depth of the points from both the light source viewpoint and the user viewpoint. Even if this approach benefits from the

availability of hardware allowing fast shadow computation for complex geometrical environments, shadow maps need to be carefully used, as they are prone to exhibit artifacts due to the finite image resolution: images can be aliased if the resolution does not permit to accurately decide on depth (and unfortunately this is often the case for large scenes for which objects might be represented by few pixels). Also, because these methods are pixel-based, the resulting frame rate might be influenced by the size of the computed shadow map and by the fill-rate available in the rasterizer.

4.5 Shadows in populated Virtual Environment

When deciding how to approach the problem of shadowing in the context of a virtual city populated by a large number of animated humans, four typical cases of sharp shadow computations can be distinguished:

1. Shadows between the static geometry (e.g. tall buildings casting shadows onto the static scenario);
2. shadows from the dynamic onto the static geometry (e.g. from the humans onto the environment);
3. shadows from the static onto the dynamic geometry (e.g. from the buildings onto the avatars);
4. shadows between dynamic objects (e.g. shadows of avatars onto other avatars).

Case (1) is a classical problem in real-time computer graphics. To display shadows from the buildings on the ground the test system can use either the well-established method of precomputed lightmaps (stored as texture maps) or, when the urban model has a single flat ground, the technique of polygonal fake shadows that has the advantage of producing sharper shadow edges. There is a broad literature on the computation and usage of light maps whose discussion is not in the scope of this work. In the context of this thesis we addressed case (2) and case (3). In our thesis we did not address case (4); it has proved to be very hard to solve impostor shadowing and self-shadowing problems without using programmable graphics hardware. Very interesting work has recently emerged in this direction, for example in [Ryder06].

4.6 Casting shadows on the surrounding environment

This requires efficient computation of the shadow cast on the surrounding environment by every individual in the crowd. Out of all the possible approaches to shadow computation, fake shadows proved in many cases to be a very efficient way to add a good amount of visual realism to the scene – obviously with some limitation. Hardware-assisted shadow mapping has also proved to be compatible with the image based representation of the impostors. Both of these techniques presents unique aspects when they are combined with the use of unstructured impostors for the crowd rendering task, discussed in the following paragraphs.

4.6.1 Using fake shadows

The first method used to compute shadows of individuals in the crowd onto urban scenarios was through of the use of fake shadowing: in particular ground-aligned impostors were used to display shadows cast by the humans. This is inspired by the way the texture for the impostor is computed - instead of using the user viewpoint as a reference to compute the needed impostor sample, the light source position is used instead (assuming a single directional light in the environment). By replacing the viewpoint by a point light source, the silhouette of the projected shadow would be given by the position of the human viewed by the light source. This shadow can therefore be represented as the image viewed from the light source mapped onto a projected polygon (Figure 4.6). Precomputing and storing a set of projected polygons for each possible light direction, a process in many ways similar to what is anyway carried out for the basic impostor technique, allows shadows to be displayed very efficiently, even in the case of moving light sources. The position of the resulting projected polygon is relative to the position of the human so that the feet of the human always touch the feet of the shadow. At run time, the appropriate texture image is chosen corresponding to the light position and the frame of animation of the human.

The texture is then mapped on to the projected polygon, and the RGB values (0,0,0) are used as the modulating colours to darken the texture. In this way no new texture needs to be generated just for the shadow and as the texture is already loaded to render the impostor, the only additional rendering cost is for this new polygon. The advantage of this approach is that the same sample images that are already inside the impostor database can be used again for the shadows. This enables high-quality shadows representing the exact shape of the walking human with extremely low additional cost, both in terms of memory requirements and

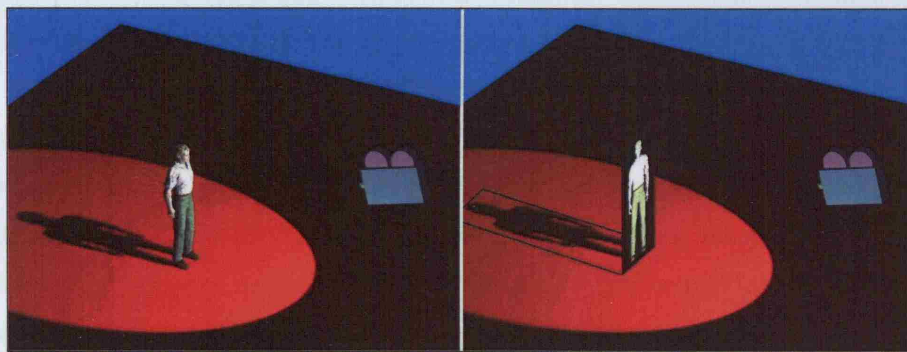

Figure 4.6 – Casting a fake shadow on the ground.
This technique can be used both for polygonal models (left) and for impostors (right).

rendering speed. There are problems too - the most noticeable is that the shadow is cast only onto an assumed horizontal floor, and this can generate bad intersections with the surrounding buildings and people. However, using a discretization of the environment such as the one reported in Chapter 5 it is possible to detect the presence of non-flat elements in the region occupied by the shadow and revert to polygonal rendering if a more accurate shadow computation is needed. Figure 4.7 shows an example of this method final resuts.

4.6.2 Using shadowmaps

Using fake shadows is a simple and extremely effective technique to improve the realism of a scene, and they are computed almost for free when using for crowds an impostors-based representation. However, it is not the only approach to shadows suitable for impostor based rendering. When available, hardware accelerated shadow maps may be used to render shadows effects. Typically, shadow maps use a 2-pass rendering approach: in the first pass the scene is transformed so that the eye point is at the light source and the objects casting shadows in the scene are rendered. The resulting depth buffer is read back, and stored into a texture map. On a second rendering pass this depth texture is mapped onto the primitives of the original scene (this time viewed from the eye point) using the texture transformation matrix and eye space texture coordinate generation. The value of each texel value, the texture's 'intensity', is then compared against the texture coordinate's r value at each pixel. This comparison is used to determine whether the pixel is shadowed from the light source. This procedure works because during the first pass the depth buffer records the distances from the light to every object in the scene, creating a shadow map.

An impostor-based representation for the individuals in a crowd is compatible with the principle of shadow mapping, at least for the computation of the shadows that the crowd cast on the environment. Adaptation of the shadow map method to the case of impostor-based crowd rendering is straightforward: during the first pass impostors are rendered as seen from the light source; this is perfectly feasible even if the impostors' geometrical information is minimal, since the shape of the silhouette of the object casting the shadow is inherently preserved by the impostor representation. The resulting depth texture is used for the second pass, when the shadow maps projects dark areas on the surrounding environment. As in the standard case of purely polygonal rendering, the main advantage of using shadow maps over fake shadows is the correct handling of projections over non flat surfaces; using shadow maps makes trivial projecting people shadows on the walls of the buildings, as well as projecting shadows on even non-flat ground, with no need to actively inspect the geometry that lies below. Consequently, when they are available, hardware accelerated shadow maps can simplify the development of realistic shadowing effects.

The drawbacks to using shadow maps in the case of crowd rendering are the same normally associated with this technique: since the shadow map is point sampled and then mapped onto objects from an entirely different point of view, aliasing artifacts might become a problem. When the texture is mapped, the shape of the original shadow texel does not necessarily map cleanly to the pixel in the framebuffer and aliased shadow edges can appear in the resulting image. Using shadowmaps of very large resolution attenuates the problem, but introduces memory problems and reduces rendering performance. The problem can be attenuated using the technique of projective shadow mapping [Stamminger02]. Recent work also suggested the use of multiple 'tiled' shadowmaps for when the scene is lit by more than one light source [Day05]. The situation where shadow maps really cannot be used is when one computes the light effects of the building casting shadows on the crowd: in this case, the lack of proper geometrical information of the impostors results in severe and view dependent artifacts. To handle this important situation we have developed a dedicated approach, discussed in the following section.

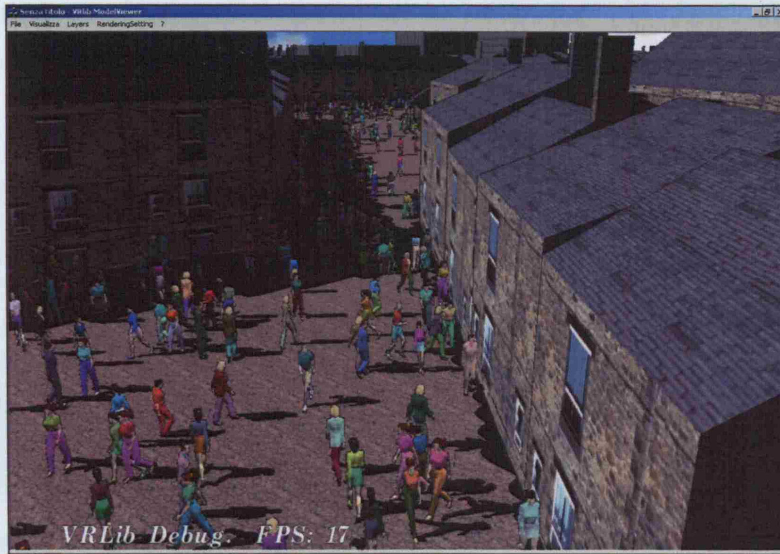

Figure 4.7 - An early version of our rendering system: crowd casting (fake) shadows on the scenario.

4.7 Shadowing effects of the environment on the crowd

In a complex urban environment, tall buildings can cast shadows on the crowd of individuals that walk around the streets of the city. Real-time rendering of this kind of shadowing effects is not simple, as it means that thousands of dynamic objects (and their shadows) need to be updated at interactive frame rates. Unfortunately, when the crowd is represented using impostors, techniques such as shadow maps or shadow volumes cannot be directly applied, as the impostors do not retain the necessary geometry information. This problem is extremely complex if considered in a general case. However, we have seen in our case that the static scene where the individuals navigate can be assumed to be 2.5D and therefore the volume covered by the shadows can be approximated by a 2.5D map. The idea, in many ways is similar to what was discussed in Section 4.3, is to discretise the shadow volumes and to store them as a 2D height map, termed the *shadow height map* (Figure 4.8a). Using the information stored in this map it is simple to compare the height of the people with the height of the shadows and accordingly to compute the degree of coverage of a human. To simulate the effect of the shadow over the impostor, we use an additional texture mapped on top of the impostor to darken the part that results in shadow; this is simply a regular 2D texture divided in two uniform black and white regions (Figure 4.8b). Varying the (u,v) coordinates at the vertexes of the impostors is used to move the border up and down along the impostor. The

way these texture coordinates are computed and the texture applied is described in Chapter 5, that also reports further details on the computation of the shadow height map.

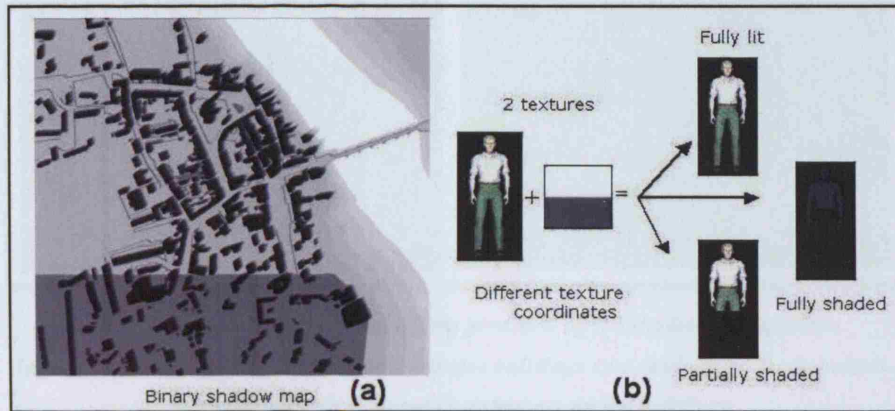

Figure 4.8 - Simulating shadows on the impostors.
(a) shadow height map for a simple urban scenario, (b) using a secondary texture to approximate the shadow casted by buildings over an impostor.

While improving overall realism of the scene, Figure 4.9 shows clearly that our current implementation of the algorithm provides only a rough approximation of the shadowing effects (note the shadows cutting across impostors horizontally). Also, our algorithm can only generate approximate results because of the space discretisation we use and for our assumption that when a shadow is cast it covers everything under it, assuming a 2.5D configuration of the scene. This excludes for example the case of roofs overhanging the edge of buildings, bridges, and other kind of 'non 2.5D' objects.

On the other hand, while for our purposes we compute shadows in the case of moving objects represented by impostors, this approach is quite generic and it could be used regardless of the type of representation used: and should the objects have a polygonal representation, the information stored in the shadow height map could also be used to quickly compute shadows onto the polygons.

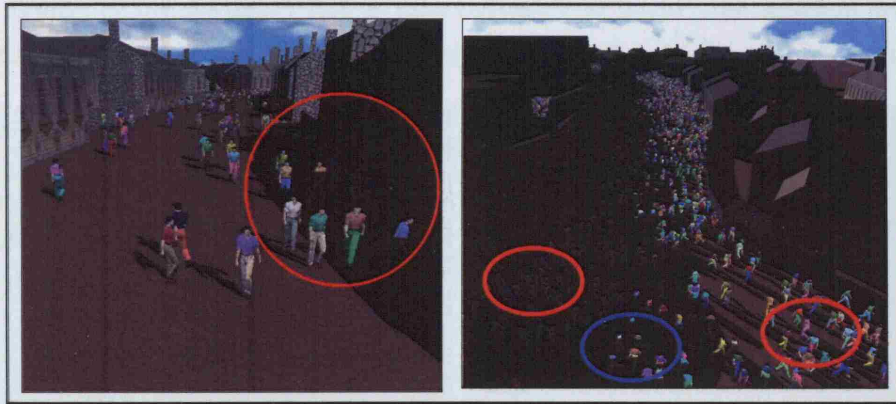

Figure 4.9 - Example of the visual effects produced by the shadowing algorithm. Impostors cast shadows on the ground, and also buildings cast shadows on the impostors. However, impostors do not cast shadows on the buildings.

4.8 Visibility computation

Visibility computation can be a very effective technique to optimize the management of large geometrical databases. In particular, when dealing with urban scenarios occlusion culling can lead to large performance gains. The presence of large and tall buildings implies a lot of potential to discard from the rendering pipeline large portions of the scenario that would not contribute to the final image . While occlusion culling is normally performed on the static part of the scene (the urban model), when a large crowd is added it might require a large portion of the rendering power; the complexity of the crowd can in extreme cases exceed the complexity of the scenario, and there are some situations where crowds are too large to be visualised in real-time even using the technique of impostors rendering. For this reason it makes sense to apply visibility culling to the crowd as well, because buildings can be very effective occluders in this case too (see Figure 4.10 and 4.11), but this requires a dedicated approach, as classic occlusion culling algorithms are normally applied to large, static scenarios and can not handle a large number of dynamic entities. From an occlusion culling perspective, the differences between animated impostors and normal polygonal models are significant. Impostors usually replace complex objects with a very light geometric primitive (the impostor quad) and, as such, they are much more fill-rate intensive than geometry-rate intensive. Culling away a single impostor from the rendering pipeline frees very little graphics resources. On the other hand, the number of impostors in a crowd is very large, and the cumulative benefit of impostor occlusion culling can lead to significant frame rate increases (see Chapter 6). A very

careful design of the occlusion test is then necessary to avoid the situation where the computation overhead of the test does not exceed the little gain of the impostor removal.

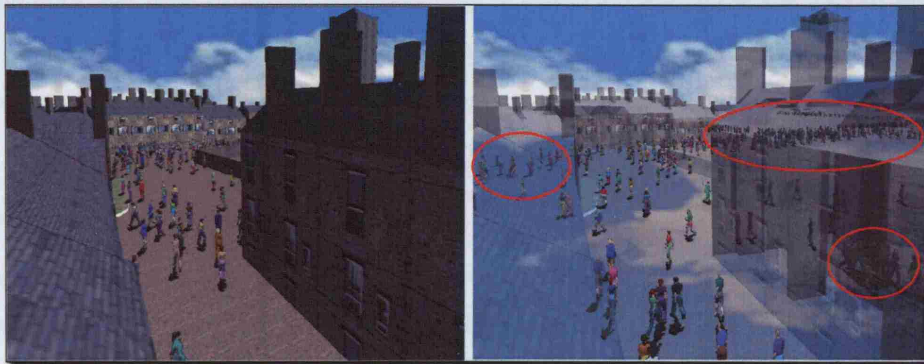

Figure 4.10 - Similarly to the static geometry, buildings can be very effective occluders. (Left) - normal rendering; (right) rendering with transparent buildings reveals that a large part of the crowd is occluded.

Despite the plethora of available occlusion culling methods, some of them reported in Chapter 2, there seems to be nothing that is dedicated to the problem of a large number of dynamic objects such as the case of crowds moving in the environment. The most similar work in this area was done by Sudarsky and Gotsman [Sudarsky96], where they used temporal bounding volumes to enclose the space-time extent of any moving object so as to avoid considering it when not visible for several frames. Even this kind of method however would not scale enough to work efficiently for thousands of dynamic objects.

An original method was designed that is specifically targeted to complex urban scenarios populated by crowds: the starting assumption is that when the viewpoint is close to the ground, the buildings become very effective occluders hiding most of the static geometry and avatars. This means that, in an urban-like environment, we can consider the geometry of the major occluders (i.e. mostly the buildings) to be of a 2.5D nature, meaning it can be defined by the building footprint + the height of each. The culling algorithm devised makes use of this property to simplify and extend a

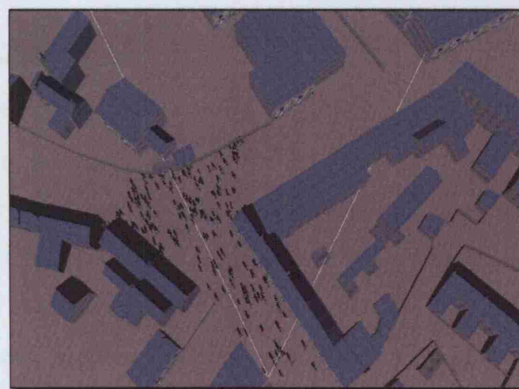

Figure 4.11 – Visibility Culling: our system avoids rendering of the individuals occluded by buildings.

classic BSP tree algorithm which performs intersection of binary trees using tree merging. The overall idea is to be able to quickly cull away the parts of the discretization used for the rest of the rendering, not only to avoid the rendering of the static geometry but also to avoid the rendering of the crowd individuals, and possibly the computation involved in animating them. Further details of this occlusion culling algorithm are presented in Chapter 5.

4.9 Beyond rendering

The more general problem of simulating the flow of people inside an urban environment is an interesting research topic that has been one of the motivations of the present work. Even if the present thesis is mainly concerned with the rendering aspects of crowds, it is important to consider some more general concepts and approaches related to animated crowds, to have an wider understanding of what are the challenges implied in the dynamic nature of crowds.
Inside a complex urban scenario, a crowd is a highly dynamic entity: individuals walk around streets and squares and their position is always changing. One of the building blocks of any crowd simulation is the ability to study the interaction between individuals and the surrounding environment. In some research, such as Space Syntax [Hillier76], an agent-centred approach is proposed for solving the task of navigation: the individuals in a crowd, perceiving the environment around them, take relatively simple individual decisions, causing a correlated *emergent behaviour* at a global crowd level. From a practical point of view, perceiving the environment implies the ability to perform some sort of collision detection which is considered in the next section.

4.9.1 Path finding and obstacles avoidance

Throughout a simulation, many of the animations that an individual performs will be based on interactions with the outside world. In allowing an single unit to conduct interactions with objects in the world, a number of general approaches may be taken. Path finding is a perfect example of this kind of interaction, since it is necessary for humans to navigate the environment they inhabit in a successful and realistic manner. In order for the crowd to perform path finding inside a virtual environment, the environment itself needs to store path finding information upon which a search can be performed. Typically, this is achieved by adding an invisible layer of nodes for the environment's terrain, where each node stores all

accessible neighbouring nodes. Using this information, a virtual human can perform a search across these nodes for the shortest walkable path between its current position and goal position. Navigation strategies need collision detection to be performed between the crowd and the surrounding environment. In some cases, a 2D discretisation of the environment is used [Hillier76, Ross92, Tecch01a] (Figure 4.12).

Figure 4.12 - Crowd navigating in the web of streets of a city. Individuals need to perform collision detection for obstacle avoidance. Our approach uses a discretisation of the environment as small 2D cells.

Given a city model and a moving crowd represented as a set of particles it is important to be able to detect any possible collisions between the particles and the surrounding environment. There are many techniques to detect interference between geometric objects [Lin98]. Many of them use hierarchical data structures, for example, hierarchical bounding boxes [Cohe95, Gott96], spheres trees [Hubb93], BSP trees [Naylor90] and Octrees [Samet90]. However, the majority of them try to solve the harder problem of interference between complex objects. They tend to be much more precise and involved than what what is required for the crowd application. As an example, in the busy streets of London during the day, there can easily be hundreds of thousands of people moving around. Trying to perform exact collision detection using standard methods for every moving entity is probably not necessary, especially if the goal is to have a general overview on all the moving entities at the same time, with the viewpoint located quite far away. Due to the large number of moving objects and the inherent time constrains of the application, other approaches can be considered which can trade off small errors in exchange for greater speed and scalability. To this extent, collision detection through discretisation of space has been used before. The most relevant work is that of

Myskowski [Mysk95] and Rossignac [Ross92]. They use graphics hardware to perform the rasterization necessary in order to compute the geometrical interference between polygonal meshes, but they focus on performing this task on a small number of very complex 3D CAD objects.

More specific to urban environments, even though the geometry is still in 3D, the movement of humans is usually restricted to follow a 2D surface. Bearing this in mind and the fact that the environment itself is static, simpler solutions can be developed. In Robotics, the problem has been studied extensively for navigating mobile robots. Lengyel [Leng90], for example, used raster hardware to generate the cells of the configuration space used to find an obstacle-free path. Bandi and Thalmann [Band98] also employed discretization of space using hardware to allow human navigation in virtual environments. However they use the information for automatically computing a motion path for a human in an environment with obstacles using a coarse subdivision on the horizontal plane and repeating that on several discreet heights, while in our case we want to consider the height of the obstacles in a more continuous way.

4.9.2 Approximate and fast collision detection for crowd navigation

An approximate method based on space discretization has been developed, with the moving humans represented as a particles system. The overall idea of the algorithm is to create a discrete representation of the static model (termed the *height map*) and use this to detect collisions of the moving particles with the environment. This map stores the height at each point in the environment and it is maintained in memory. For every frame of the simulation, before moving a particle to its new position we check its current elevation against that stored in the height-map for the target-position. If these values are too different, it means that the step necessary to climb either up or down to get to the new position is too big and cannot be taken, otherwise we allow the particle to move and update its height according to the value stored in the height-map. The algorithm is organized in two phases: the generation of the height map and the run-time collision test.

4.9.3 Height map generation

The height map is generated using standard OpenGL functions. This is done at the start of the simulation by positioning the camera over the centre of the model looking down at it with the

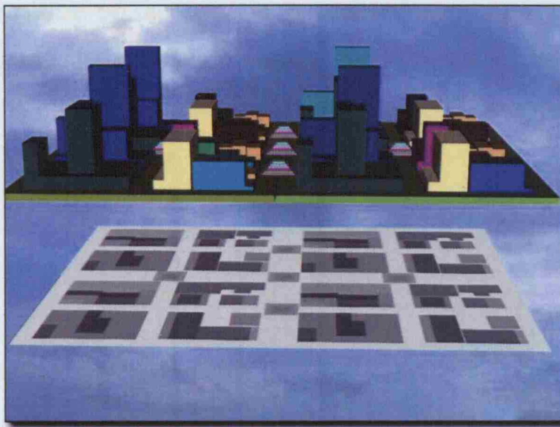

Figure 4.13 - A simple urban model and its corresponding height-map (below, in grey scale).

view frustum adjusted to match the model boundaries. The model is then rendered using an orthogonal projection and the resulting contents of the z-buffer, that represent a discreet map of the heights of the model, are copied into the main memory where they can be accessed in a faster way (Figure 4.13).

Using OpenGL to generate the height-map allows the algorithm to be simple and very fast thanks to the use of dedicated hardware. Obviously, during the generation of the height-map, the complexity and the scale of the model must be taken in consideration; in order to permit collision detection tests with the right order of precision, the height map has to be of sufficient resolution. However, it is important to notice that although higher resolution maps are more expensive in terms of memory requirements, the speed of the height map test for each particle doesn't seem to be noticeably affected by the size of the map.

4.9.4 Collision detection and avoidance

Depending on the situations, two different approaches for detecting and avoiding collision of the moving particles were used. The principle is the same in both cases: as each particle moves in the assigned direction the presence of obstacles in front of it using the information stored in the height-map is checked (Figure 4.14). In the first case, the position that the particle is going to occupy after the current movement is checked and this position is computed and mapped onto the height map. If the height at this point is found to be close enough to the current height of the particle the movement is considered valid and the particle is allowed to move there. If the difference in heights is too large a new itinerary needs to be found. This is done by gradually rotating the particle's direction in small angle steps until an obstacle-free direction is found. In the second case the collision detection task ahead of the current particle position is shifted. Instead of checking whether the next step is possible from the current position, whether the i-th step is possible from the predicted (i-1)-th position is checked. If not, the direction is again rotated by a small angle as in the previous case, but the

position of the particle is updated anyway so that the particle starts changing direction gradually before colliding against an obstacle, producing as a result a smoother animation. On the other hand, two accesses to the height map are needed, making this method slower than the previous one.

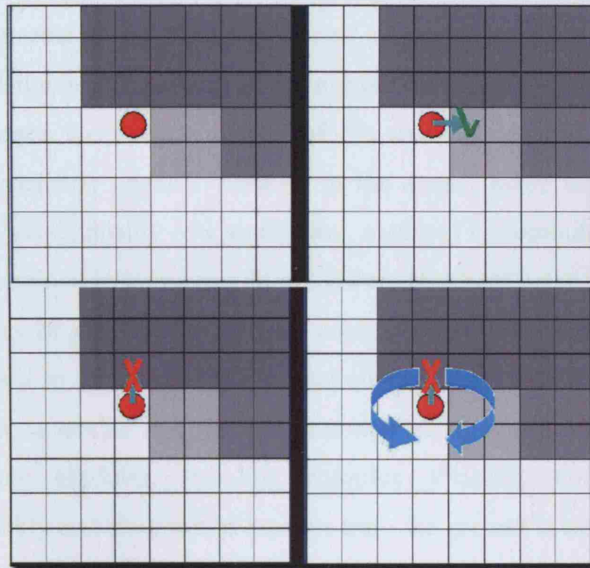

Figure 4.14 - A simple collision avoidance strategy using cells elevation. The particle search for obstacles comparing the elevation of the current cell against the elevation of the destination.

The aim of this simple trial and error strategy to find a free path is to avoid querying directly the geometrical database for valid directions, in order to keep the cost of the collision test low. Using the height map, the particles correctly detect the different dimension of obstacles, climbing on them if the steps are small enough and updating their elevation without accessing the geometrical data representing the model.

A similar technique can be used for inter-collision detection between particles: an additional map (termed *intercollision map*), can be used to detect possible collisions between individuals: before moving to a new cell, a particle checks the destination cells to determine if free or occupied by the ID of another particle. Similarly to the particle-ambient collision detection, it is possible to specify how far ahead this check should occur, so to start changing direction gradually before collision occurs.

Being based on a crude discretization of the environment, our algorithm is not suited for fine-detail collision detection, and as such should be used only for the crowd navigation task, reverting to proper polygonal representation for the individuals in the foreground of the scene.

4.10 Summary

Techniques have been presented to further extend the possibility of animated impostors rendering, enhancing the realism of the rendered scene. An appealing aspect of this method for real-time crowd visualisation is the mix between traditional and image-based rendering strategies; both can coexist in the same framework, making it possible to visualise scenarios that would be too complex using a polygonal approach alone. In particular, we presented a new method to enhance the overall realism of the scene modulating the intensity of the impostors images depending on their position in the scene. Since the display of shadows greatly improves the visual quality of a simulation, methods to compute and update shadows for thousands of dynamic objects moving in a 2.5D environment were developed, focused on improving consistency of positioning between objects rather than accurate shadow casting. This method is adapted to the context of a virtual city simulation with an animated crowd of humans, and in order to enable fast shadow detection of buildings on the crowd, we use a 2.5D map to locate shadows, avoiding complex visibility computations. Impostor representation to quickly cast shadows of humans onto the ground is also used. We have also presented a new approach to visibility computation that takes into consideration the peculiarities of impostors as a rendering primitive, stressing the importance of occlusion culling in real time rendering of densely populated virtual environments. Finally, we presented a new and fast approach to collision detection (both particle-environment and particle-particle) using a 2D discretisation of the environment, showing how it can be applied to the problem of crowd navigation in complex urban environments. Chapter 6 will show that the computational cost of these techniques is relatively low, and that the system still allows an interactive walk through in large and crowded urban environments.

CHAPTER 5 - THE CROWD RENDERING SYSTEM

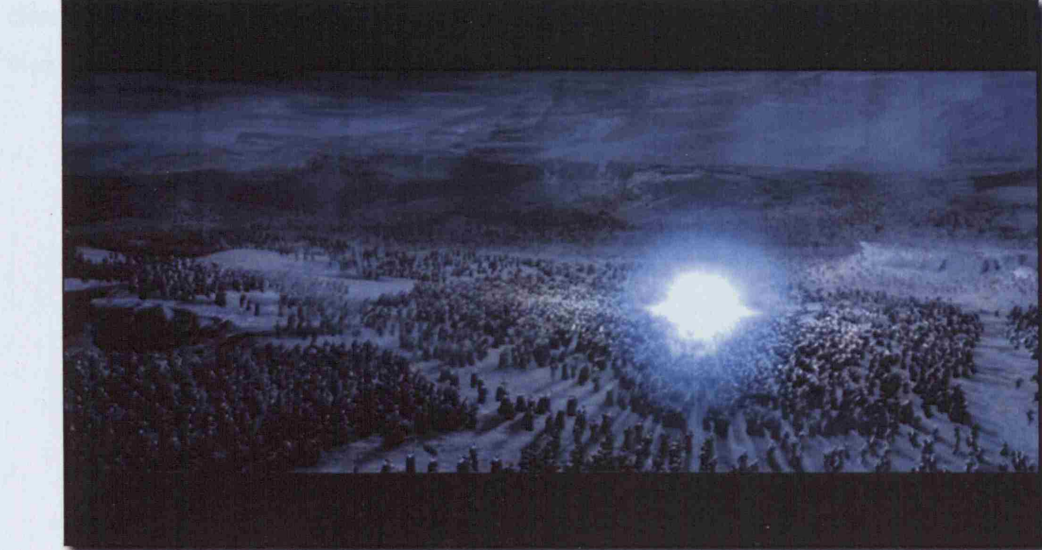

Figure 5.1 - A scene from the movie 'The Fellowship of the Ring' (New Line Cinema – 2001)

Usually a novel real-time rendering technique becomes widely adopted if it can be realised using existing graphics hardware and needs only marginal changes to pre-existing software architectures. Image-based techniques are particularly critical with respect to this, as they often require substantial modification to both the rendering pipeline and the underlying scene-graph in order to achieve optimal performance. An advantage of using impostors for crowd rendering is their compatibility with traditional polygonal-based rendering frameworks. In order to demonstrate this, a prototype rendering system and a set of flexible libraries and tools were developed. This shows that unstructured animated impostors can be used inside a traditional scene graph performing crowd rendering on OpenGL1.2 or OpenGL1.3 compliant graphics hardware. The present chapter gives an overview of the system's general architecture, showing the design guidelines that were used. Some extra details are exposed on the most relevant aspects of our implementation.

5.1 General guidelines

A complete rendering system was developed able to load and display the large urban scenarios that were used for real-time crowd rendering testing (Figure 5.2). The system uses many of the classic optimisation techniques for large models rendering, such as OpenGL state sorting, view frustum culling, occlusion culling and polygonal LODs management.

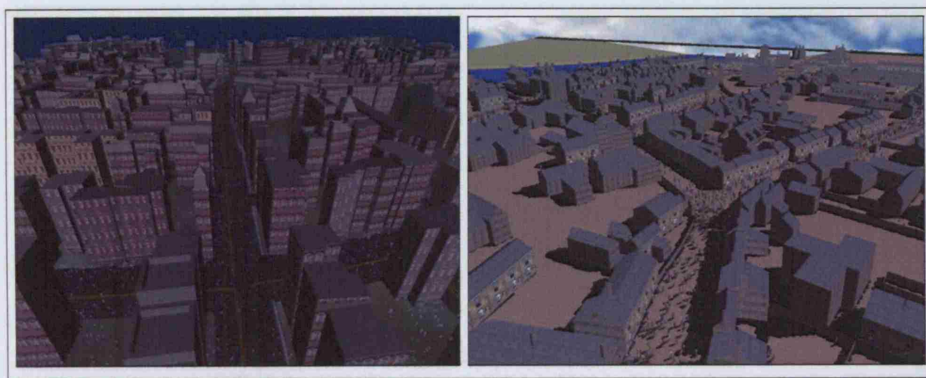

Figure 5.2 – Two examples of urban environments rendered in real-time in our system.

Concerning the rendering modules specifically dedicated to impostor based rendering, a set of tools was generated that can be broadly divided in two groups:

- various tools for generation and optimisation of animated impostors from polygonal models used in a preprocessing phase;
- a dedicated software library (named *CrowdLib*) based on OpenGL that takes care of the majority of rendering aspects linked to run-time crowd rendering, with a sufficient interface abstraction so that it can easily be plugged also in generic scene-graph infrastructure.

The two general design criteria at the basis of CrowdLib development were performance and portability. The cross-platform portability of the algorithms is assured by the use of standard OpenGL1.3 for the run-time rendering, a choice that allows the use of the library on almost any platform supporting basic hardware accelerated graphics. CrowdLib has been compiled and used under Windows, Linux, Apple OSX, and SGI Irix. On the Win32 platform a binary version is available using the DLL (Dynamic Link Library) mechanism, making it particularly

simple to use the library inside larger projects. Today, there are applications using CrowdLib on a variety of platforms, from desktop PCs to CAVE-like installation such as UCL's CAVE-like system (Trimension ReaCTor). The library was successfully embedded in a variety of scene-graphs such as OpenSG [Aróstegui05], XVR [Carrozzino05] and the UE CREATE project scene-graph [Loscos03, Drettakis04].

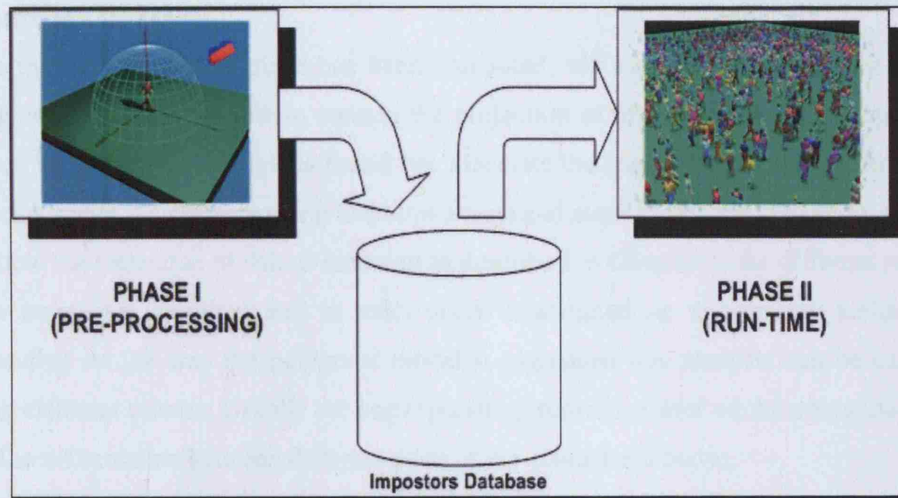

Figure 5.3 - Using impostors can be subdivided in two macro phases:
Creation of the database occurs at pre-processing, while samples rendering takes place at run-time.

5.2 Creation of the impostor database

The first step in the process of creating an impostor-based crowd is the construction of the impostors database, taking place in a pre-processing phase. The polygonal models used for the animated characters were imported from standard modelling and animation tools (3DStudioMax [Disc3dMax06], Maya [AutMaya06], Poser [Clans06]).
Impostors are created starting from the 3D polygonal models following a sequence of steps:

- each model is rendered from a fixed set of different view-positions as exposed in Chapter 3. The number of samples and the camera distance from the object during this sampling process are tuned on the basis of the total texture memory budget allocated to the impostors storage. The pixels that get rendered on the framebuffer are considered 'visible samples' of the objects;

- the frame buffer content is grabbed and analysed, and the RGB colour components of each pixel composing a single image are stored in a temporary memory buffer. Each visible sample is re-projected back into 3D space coordinate using their XYZ windows coordinate (Z is the normalised z-buffer content). Once re-projected in the 3D space, they form the cloud of visible 3D samples of the object from that particular direction and distance. The cloud is then analysed and the best projection plane is computed following the guidelines reported in Chapter 3;
- once the best projection plane has been computed, we search for the smallest rectangle lying on this plane and able to contain the projection of all the visible 3D samples of the object. Once such a rectangle is found, we associate the spacial coordinate of the resulting 4 vertices with the corresponding impostor image and store it into the impostors database;
- to allow the technique of colour modulation described in Chapter 3, the different regions of each image are analysed and to each pixel is assigned an appropriate alpha value - depending on the way the polygonal model is organised this analysis can be carried out using different criteria. Usually the corresponding material colour on the polygonal mesh is used to differentiate between different parts of the character's body;
- the last per-pixel computation step involves computing a normal for each pixel to be used to achieve per-pixel lighting: the normal (in object space) of each visible 3D sample is computed. The resulting XYZ components are quantised using 8 bits for component and stored as RGB values of the corresponding pixel of the impostor image. 8-bit quantisation of the values is usually performed due to the texture format restriction of OpenGL1.3;
- when all the samples are computed, the impostors images are packed together, storing them as conventional RGBA texture-maps; more details on this step are given in Sections 5.3 and 5.4.

At the end of the capturing phase impostors are fully stored in the database in the form of a collection of texture-maps, a set of geometrical vertices defining the impostor planes and a set of texture coordinates that encode the mapping from the image-space to the geometry (Figure 5.4).

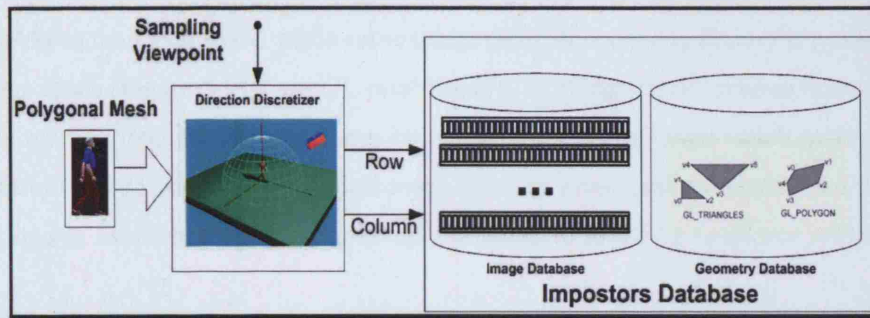

Figure 5.4 - Storing impostors data.
The impostors are organised as a set of RGBA textures and the geometric information (vertexes, (u,v) coordinates, normals) needed to place the images inside the scenario.

5.3 Memory management

When there is a large number of geometrically different characters, each animated with a long sequences of key frames, the described steps can generate a potentially large amount of impostor data. Therefore, a high memory requirement is the main drawback of the impostor techniques, and although the amount of texture memory available on modern graphics hardware has greatly increased in recent times (graphics accelerators equipped with 512 Mb of on-board memory are now available even on the consumer market), still efforts should be made to optimise the memory storage strategy and manage the total memory amount needed by the impostors.

5.3.1 OpenGL memory management

Some knowledge as to how OpenGL manages texture memory helps in the development of optimisation strategies. In a typical rendering pipeline there is a certain amount of video memory that serves multiple purposes. Some of this memory is allocated to store frame buffer data, some is used for the depth-buffer and some (optionally) for the stencil buffer. With the exception of small amounts dedicated to special features such as display list and vertex arrays, the majority of the remaining memory is available as texture-memory. To improve the overall rendering capacity of the systems, OpenGL allows the total amount of texture memory to be larger than the physical video memory of the graphics accelerator, and implements a virtual memory mechanism that uses the system RAM as an additional space to place unused or less frequently used textures when the video-ram is full. When a texture residing in main memory

is needed for the rasterization of a fragment, it is pulled through the bus that connects the system RAM to the video RAM while some temporarily unnecessary texture is pushed back to make some space (Figure 5.5). OpenGL establishes a 'working set' of textures that are resident in texture memory and these textures can be bound to a texture target much more efficiently than textures that are not resident. To influence how the virtualisation mechanism will handle the working set, for each texture it is optionally possible to specify a *residence priority*.

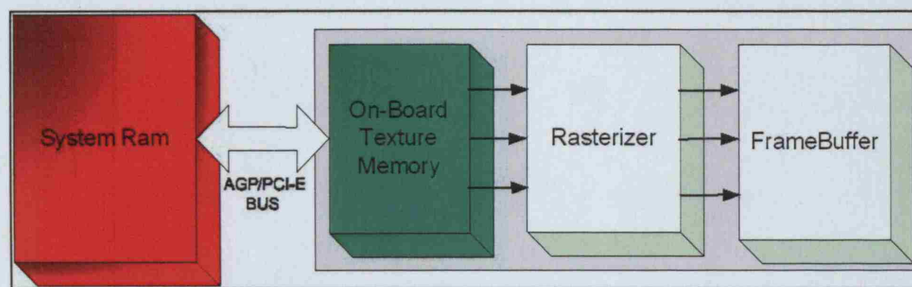

Figure 5.5- A simplification of the rendering pipeline.
OpenGL virtualises the amount of memory dedicated to textures using (slow) communication between the on-board memory and the the system RAM.

Since the bus connecting the system and memory RAM is characterised by relatively high latencies and a low throughput, this mechanism works with acceptable performance only as long as the data traffic is limited. Scenarios using a large amount of texture to render a single frame may cause an overload of the bus leading to serious performance degradation, a phenomenon commonly termed *texture trashing*. While the use of impostors is very effective in reducing the total geometrical load that needs to be processed for each frame, the use of a large number of impostor data favours the insurgence of texture trashing, also because large city models often require a large amount of texture memory themselves. Reducing the occurrence of texture trashing can be achieved in three ways: using OpenGL1.3 texture compression features, reducing the amount of texture memory necessary to store the impostor images, or organising the impostor data using a subdivision of the resulting database into smaller memory chunks, as this helps the virtualisation mechanism to handle bus data at the maximum efficiency should texture swapping become necessary. The following two sections will present how these strategies are achieved inside the CrowdLib module.

5.3.2 Using OpenGL texture compression

When the graphics hardware offers support for texture compression (unfortunately this excludes the SGI Reality Monster that still runs several CAVE-like installations), CrowdLib can use it to reduce the amount of video memory needed to store the impostors images. Initially developed by the company S3 Graphics Co. Ltd, texture compression [Iourcha99] was introduced as a standard feature in OpenGL starting with version 1.3. Using compressed texture is extremely efficient and brings advantages not only in terms of storage space but also in terms of rendering speed, as the reduced memory transfer traffic taking place in the hardware normally provides a 10-15 % fill-rate speed boost, while image quality does not decrease noticeably. Due to the large savings of video-memory, the performance bonus and the ease of use, texture compression has been quickly adopted by the entire OpenGL developer community. Using texture compression is simple: OpenGL 1.3 (or later versions) introduces a new series of internal compressed texture formats, all starting with the prefix S3TC, that be specified when storing an image. S3TC compression ratios are fixed on a per-format basis [Segal01]. A list of compression ratios is given in Table 1.

S3TC Compression Format	Compression (bits/texel)	Compression ratio (8bits/channel)
GL_COMPRESSED_RGB_S3TC_DXT1_EXT	4	6:1 / 8:1
GL_COMPRESSED_RGB_S3TC_DXT3_EXT	8	4:1
GL_COMPRESSED_RGB_S3TC_DXT5_EXT	8	4:1

Table 5.1 – The S3 OpenGL Compressed Texture formats.

The S3TC compression formula is:

$$ImageSize = BlockSize\,x\,\frac{Width}{4}\,x\,\frac{Height}{4}$$

where BlockSize is 8 bytes for DXT1 and 16 bytes for DXT3/5. [1]

However, image compression is lossy, so it is achieved at the expense of a slight change in the way RGBA channels are managed, and need to be carefully handled to avoid unexpected image artefacts. In fact, even if they share the same compression ratio, there are important

1 Because 24 -bits textures are actually stored as 32-bit textures, the format GL_COMPRESSED_RGB_S3TC_DXT1_EXT can be seen as having an effective 8:1 compression ratio.

differences between the compression algorithm used in the DXT3 and DXT5 formats: DXT3 applies for the RGB channel the same compression scheme of DXT1, and then sub-samples the original 8 bits of the alpha channel using only 4 bits. For this reason, only 16 different values are really possible for the ALPHA component once it has been stored, and this leads to a limitation that there is a maximum of 16 different regions in our multi-pass algorithm. On the other hand, the sub sampling mechanism of alpha channel is simple and controllable, as it is basically just a shift of the 4 most significant bits of the channel, allowing easy management of the alpha regions following the strategy described in Chapter 2.

5.3.3 Tight packing of the impostor images

While the use of texture compression already reduces the effective amount of memory needed by the impostors approach, designing an optimised packing strategy of the sampled images is crucial for the success of the method, as well as having a strong impact on the overall rendering speed. A first optimisation strategy investigated was to pack multiple impostor samples on a single texture in order to reduce the texture binding operation performed at run time. The starting point for this was a simple strategy based on regular grid (as reported in [Tecch00b]): each image was placed inside a regular table, tuning at run-time the texture-coordinate in order to appropriately associate a given image to the impostor geometry. In that phase, each image sample was a pre-rendered ray-traced image of the character (256 by 256 pixels resolution), generated off-line using a commercial ray-tracer (3DStudioMax) and its scripting code. A set of the original images were then scaled down and packed into larger textures in order to minimise the amount of context switching occurring at run-time. Apart from being simple to implement, the use of a regular grid has some advantages over more elaborate solutions: since the amount of texture space allocated to each image is the same for every impostor regardless of the view direction, UV coordinate lookup can be done using simple integer arithmetic on the bases of the elevation and orientation indexes. On the other hand, this packing strategy, albeit simple and computationally efficient, may lead to a waste of texels as the regular subdivision generally means that there are large gaps between one image and the others (refer to Figure 5.6). A greatly improved occupation of texture memory can be achieved by relaxing the regular subdivision constraint. In [Tecch02b] image samples are placed very close to each other in order to reduce the unused space that would otherwise get wasted. Finding the best disposition involves computing for each sample the smallest

rectangle containing all the pixel, and can be done at pre-processing time. Then, all the samples can be packed in a single image, always leaving a row or a column of empty pixels between adjacent samples to avoid or attenuate the artefacts that might be caused by OpenGL texture-filtering. After tight packing, the aggregate image is much smaller than when using a regular grid with resulting large savings in the overall texture memory occupancy.

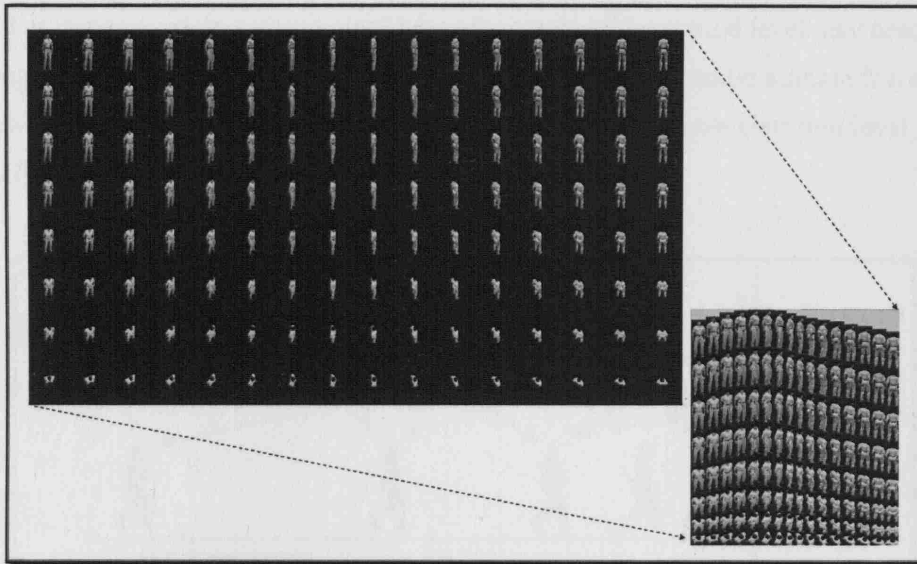

Figure 5.6 - Packing impostors together.
Tight packing of the impostor samples reduces the amount of unused texture regions.

While the use of a tight packing strategy improves the overall texture memory consumption, it also introduces some complications in the rendering process, due to the fact that the samples are no longer disposed using a regular grid: individual texture coordinates need either to be computed at run time, or to be precomputed and stored for each sample. The latter solution was chosen for our system, computing the UV coordinates at preprocessing and storing them in the impostor database. Then, at run time we use this information to compute on-the-fly the right impostor's size and orientation, in order to avoid distortions of the sample image.

5.4 Organisation of the texture working set

In principle, the best way to store the impostor samples would be to use a single, very large texture for the whole impostor database, as this effectively minimises the number of context

switches required during the run-time rendering phase. Apart from not being feasible due to general hardware restrictions of the graphics hardware addressing system (a single texture-map cannot generally be larger than 2,048 x 2,049 or 4,096 x 4,096 pixels), such a solution, requiring a fixed amount of texture memory, would not exploit the fact that not all the impostor images are necessary at all times. In fact, given a generic view direction and elevation, a large proportion of the impostors images may not be used at all in a graphics frame. For instance, while a view-point high with respect to the ground level may need all the sampling levels of the impostors to be present in video memory to render a single frame, when the view-point is close to the ground only the samples related to one elevation level may be necessary, as depicted in Figure 5.7

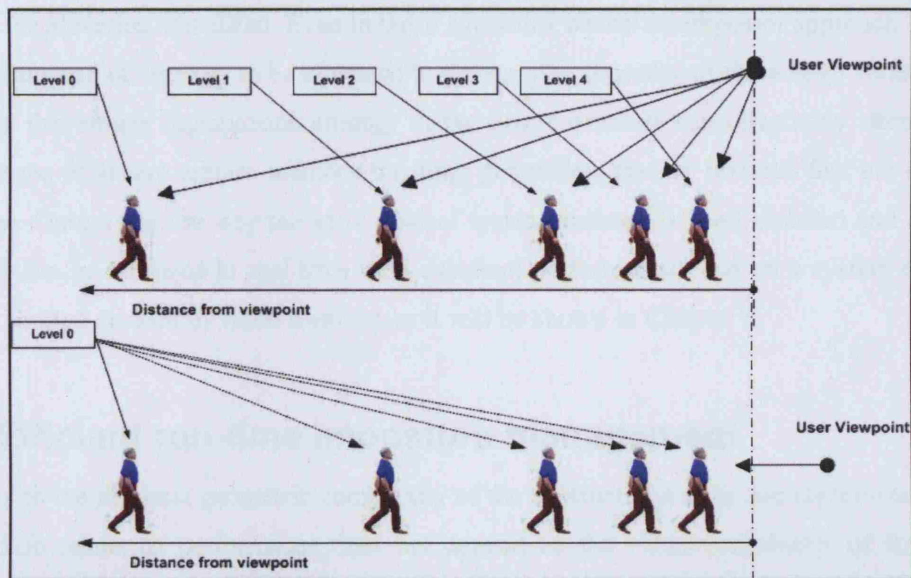

Figure 5.7 - Some elevations of the camera needs a larger variety of samples to be rendered. When the user viewpoint is close to the ground only one set of samples is needed (level 0).

Aside from tightly packing together the impostor images to avoid wasting texture memory with unnecessary gaps between images, additional subdivisions of the database can thus be used to optimise the texture allocated to impostors on a frame-by-frame basis. The main idea is to use the large system RAM as an additional pool of memory, trying to keep in video memory only the minimum amount of textures necessary for the impostors in each frame. In this way, it becomes possible to exploit the image coherence between two consecutive frames,

swapping gradually over the slow bus only the data that really is useful for the next frame. This form of fine organisation of the way impostors are subdivided, can effectively optimise the way data transfers on the video bus, leading to a more effective virtualisation of video memory and at the same time freeing more memory resources for normal polygonal rendering. There are similar considerations in relation to the animation sequences: while the impostor database could contain hundreds of key frames of animated characters (walking, running, sitting or chatting) a generic visualisation frame may need just a subset of them. It was realised that an effective strategy for impostors organisation is to subdivide the image samples packing together all those related to the same object and to the same frame of animation (it should be remembered that the same impostor database can contain images of different objects, both animated or not), with a further subdivision on the basis of the sample from which the elevation was taken. Even in those situations where the impostor approach requires a large amount of memory to be allocated to the samples (because of the a large variety of the crowd), this simple organisation strategy of the image database can effectively attenuate the insurgence of severe texture memory trashing, generating smaller textures that are easier to manage. Optimising the way the extra pool of system memory is used, detailed and complex crowds can be rendered in real time with excellent performance even on a system equipped with a limited amount of video memory, as it will be shown in Chapter 6.

5.5 Efficient run-time impostors management

Thanks to the minimal geometric complexity of the unstructured impostor representation, the CrowdLib rendering performance does not depend on the visual complexity of the human model represented, but rather on the combined computation taking place in what could be called the crowd rendering pipeline (Figure 5.8). At the end of the capturing phase impostors are fully stored in the database in the form of a collection of texture-maps, a set of geometrical vertices defining the impostor planes and a set of texture coordinates that encode the mapping from the image-space to the geometry. At run time, the virtual camera position goes through the same discretization process used in the sampling phase, generating row and column indexes that are used to look-up the corresponding impostor data. Once the vertices, texture images and texture coordinates are retrieved, the impostor is rendered with the multi-pass algorithm described in Chapter 3, with the lightness of this process making impostor rendering so effective.

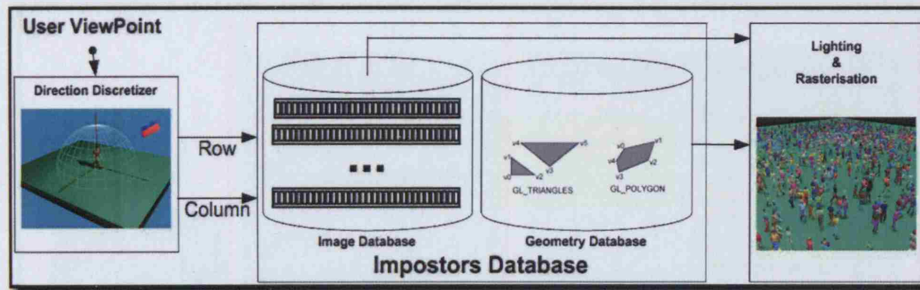

Figure 5.8 - The Lookup, Selection and Rendering phases of the crowd rendering pipeline.

The efficiency of the discretization algorithm is an important consideration for the performance, as this operation needs to be performed by the CPU every frame and for every impostor, but the management strategy of the OpenGL pipeline can also have a large effect on rendering speed. The detrimental impact of frequent state changes is a well known characteristic of rendering hardware, that in an attempt to exploit the intrinsic parallelism of graphics operations, implements deep operation pipelines and a large number of registers. Also, small high-performance cache memories are used inside the graphics hardware to speed-up transformation and rasterisation of triangles (Figure 5.9). Cache-miss events result in memory transfers and bus traffic, as there is a need to upload in the high-speed caches new content, copying it from the local video memory. Even if high performance buses are used, the plethora of little glitches associated to cache refilling inevitably contribute to sub-optimal performance. Such a complex architecture can only work at full speed when there is not frequent command and registers flushing.

From a software point of view, OpenGL is a simple state machine with two operations: setting a state, and rendering utilizing that state. Reducing the number of times a state needs to be set reduces the amount of work the graphics card and it's software driver have to do. This technique is generally referred to as *state sorting* and attempts to organize rendering requests based around the types of state that will need to be updated. Generally, the goal is to sort the render requests and state settings based upon the cost of setting that particular part of the OpenGL state. With regards to our crowd visualisation code, rendering is optimized by sorting the virtual humans in the following order based on the most to least expensive state changes: binding a texture, setting the modulating colour and changing the alpha test threshold. CrowdLib sorts at run time the impostors by template model, then by the current key-frame of animation, then by elevation of the virtual camera position and finally by impostor LOD based

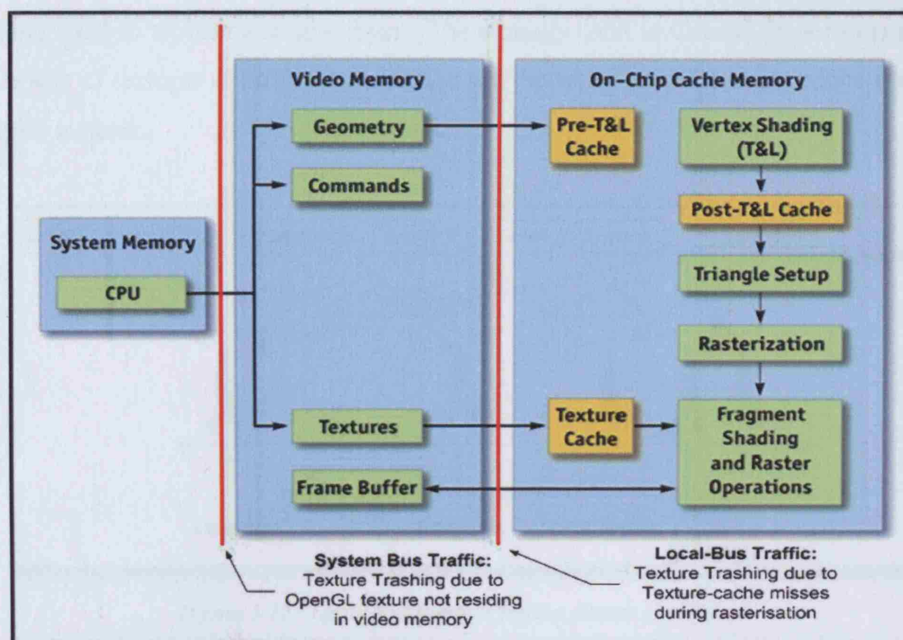

Figure 5.9 - The three stages of memory involved in the rendering process.

on the individuals distance, organising the rendering of the crowd accordingly. This reduces the number of times that states have to be changed, and this includes the setting of lighting parameters, alpha test enabling and disabling and texture loads and binds.

5.6 Taking distance into consideration

As pointed out in the previous paragraph, the worst case in terms of memory occupation when using our impostors-based crowd visualisation technique is when the viewpoint is high with respect to the ground, since this requires samples from all the different elevations to be present in memory at the same time. For this situation an additional database optimisation technique has been devised that can reduce the overall amount of texture memory needed at run-time. A technique similar in principle to the concept of multiple LODs for polygonal rendering is employed: the idea is to use less data for the far objects, exploiting the fact that they need much less visual detail for rendering. There are conceptual differences from the case of LODs for polygonal models, since the geometry of an impostor does not depend on the object complexity, and being already minimal (an impostor has just 4 vertices) cannot be further simplified. Still, two parameters can be varied as a function of the distance: sample

image size, that can be reduced as the distance of the object to replace increases, and number of samples used to replace a single object. The strategy used in CrowdLib is to avoid having multiple sets of textures at different resolution for the same object, and to reduce the number of samples instead.

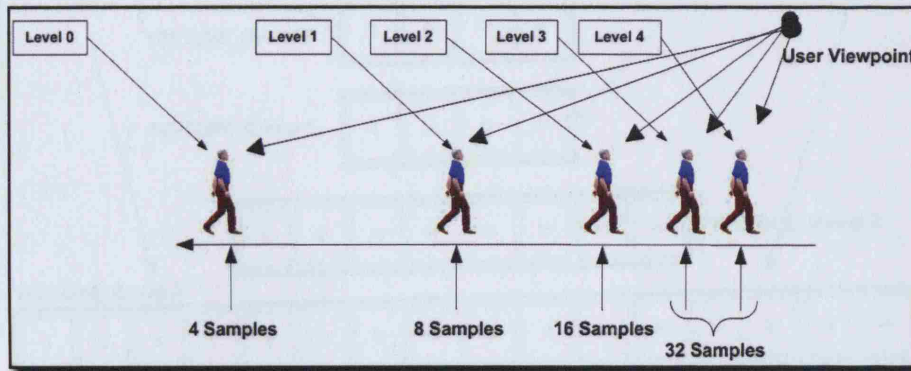

Figure 5.10 - Using less samples for the distant characters.
The number of samples needed to approximate a character can be changed depending on the distance from the viewpoint, as more distant characters can use less samples.

If we take the worst-case scenario (viewpoint hight with respect to the ground so that all the samples related to all the elevations need to stay at the same time in texture memory) and refer to Figure 5.10, it can be seen that some elevation levels (such as Level 0) only appear for the most distant characters. When an impostor is so far away we can assume that reducing its visual detail should not introduce notable errors in the final image. Instead of using smaller object images, the technique used reduces the number of samples around the object that it is necessary to keep in memory. This is done by subdividing the original single set of images (in this example 32, but this number can vary depending on the precision we want to obtain from our impostors), in four smaller chunks, (TEXTURE_Group0,1,2,3) – see Figure 5.11. Note that in this way TEXTURE_Group0 contains orientations 0,8,16, and 24, that are the four main character orientations (East, South, West and North). At run time just these orientations are used for the distant characters (those using Elevation Level_0), avoiding loading the other texture groups and leaving in this way more video memory space available for closer levels. For Elevation Level_1, more samples are added, introducing also TEXTURE_Group1, containing orientation 4, 12, 20, 28 (South-East, South-West, North-West, North-East). Now there is a total of 8 possible orientations to choose from, a rather better approximation.

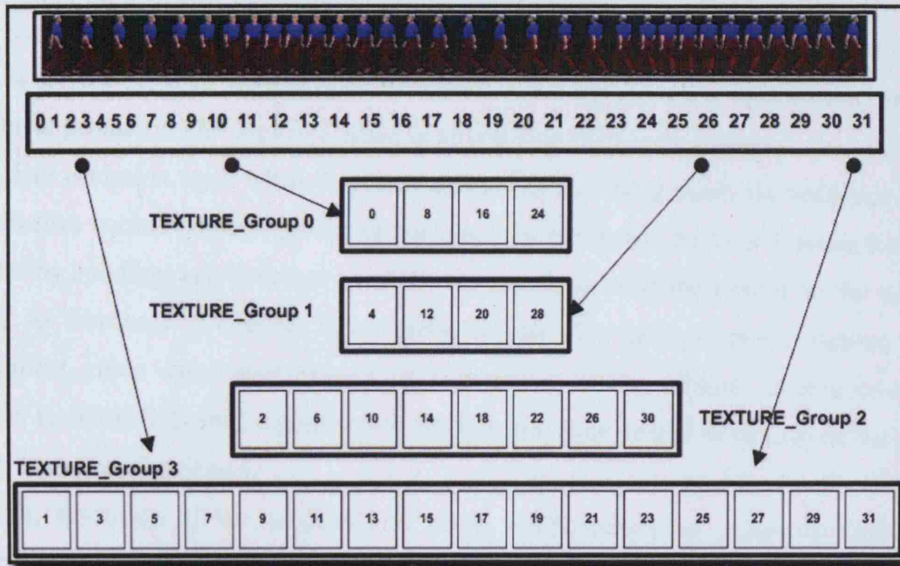

Figure 5.11 - Subdividing textures to optimise memory management.
A single texture set can be subdivided in four smaller ones to alleviate the effects of texture trashing in highly populated scenarios.

Similarly, adding TEXTURE_Group2 brings the total number of samples to 16, and those can be used for intermediate elevation levels (2 and 3). Finally, adding TEXTURE_Group3 brings back all the original samples. This is what is needed for the impostors closer to the camera. This further segmentation of the samples introduces some small overhead in terms of texture switching, but it can greatly reduce the total amount of video memory in those situations when it's more needed, that is when the viewpoint is high from the ground. It offers obviously no advantages when the viewpoint is close to the ground and a single elevation level is present for all the impostors, independently from the distance. In Figure 5.12 this additional stage is added to the crowd rendering pipeline.

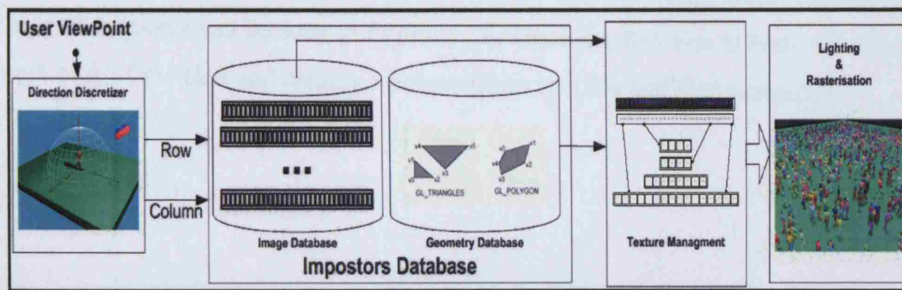

Figure 5.12 - The complete run-time crowd rendering pipeline.

5.7 Visibility computation

View frustum culling is another classic optimisation technique that can be applied to crowd visualisation to improve rendering performance avoiding rendering individuals hidden by foreground geometry. The nature of urban environments helps with respect to the construction of efficient occlusion tests: when the viewpoint is close to the ground, the buildings become very effective occluders: large groups of humans may fall inside the view frustum but still be occluded by buildings and therefore would be unnecessarily rendered were it not for occlusion culling. As discussed in Chapter 4, occlusion culling of a dynamic crowd against a static environment poses some new aspects in comparison to the classic scenario-to-scenario situation, therefore dedicated algorithms to perform real-time occlusion culling on the moving crowds have been developed.

By taking advantage of the properties of urban environments an occlusion method was developed based on BSP tree merging [Naylor92]. The general strategy was to perform occlusion computation in every frame and on every individual, inside a procedure called just before the rendering phase of each frame. (Note: in the following the terminology from [Naylor92] is used: a hyperplane is used to denote an infinite plane in 3D or line in 2D, while a sub-hyperplane is the part of the hyperplane that falls within the domain of the sub-tree in question)

Phase I – preprocessing: The first step of the approach involves building a KD-tree [Samet90] on the scene geometry. This is a 2D tree using the x-y of the bounding boxes of the objects but each node (leaf and internal) holds also the height of the geometry within it. The partitioning planes of the KD-tree are restricted not only to be axis aligned but also to be coincident with a tile edge of the collision detection grid. This ensures that any tile of the collision grid falls neatly within a single tree leaf (of course a tree leaf can contain many tiles). An example of the results of this process can be seen in Figure 5.13. Once the KD-tree is built, the algorithm runs through each of the tiles and creates a pointer from it to the leaf that contains it.

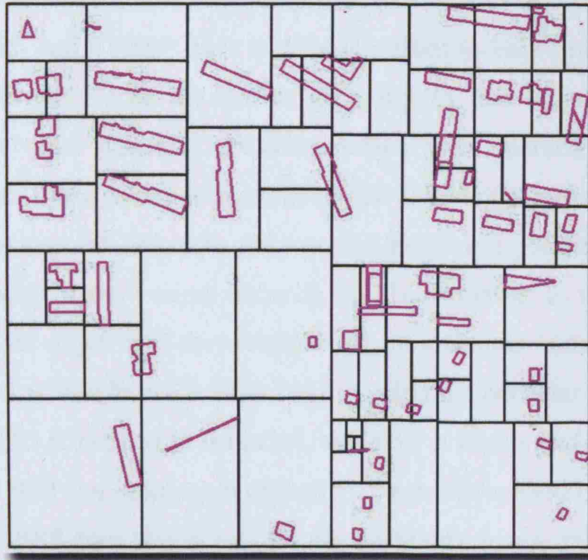

Figure 5.13 - Subdivision of the scenario's occluders.

Phase II – Run Time: The second phase is performed once per frame, and takes place just before the rendering of the crowd: we perform an occlusion test on the KD-tree, marking the leaves that are visible with the ID of the current frame. When the rendering of the static scenario is complete we start rendering the crowd; when we run through each individual, before processing it any further, we check if the tile in which it resides holds a pointer to a leaf that is visible in this frame or not and, in the latter case, we discard the impostor.

The algorithm to perform the occlusion test at each frame has two steps:

- First we build a BSP tree from the viewpoint using occluders selected in the previous frame. This tree is similar to the SVBSP tree of [Chin89] but in 2D plus height.
- Then we merge the SVBSP and KD trees by inserting the former into the latter. This means that the KD tree is not actually modified during the merging but rather when parts of it are found to be in homogeneous regions of the SVBSP tree they are just rendered if the region is marked visible or ignored if the region is marked occluded. The SVBSP tree on the other hand is fragmented in the process but that is of little consequence since it changes in every frame as the viewpoint moves and we have to rebuild it anew.

5.7.1 Occluder selection and building of the occlusion tree

The merging of the kd- and SVBSP trees is done is a front-to-back order with respect to the current viewpoint. As we render the visible geometry in this order a list of the visible buildings is kept, or other potential occluders which were marked as such in the pre-processing. In the next frame, since it is assumed that the viewpoint moves smoothly rather than jumps from point to point, these are still some of the closest good occluders. The SVBSP tree starts as a tree made of the 4 edges defining the view volume. In this initial tree we add one by one the occluder edges until they reach a leaf, in a manner similar to [Chin89]. If the leaf reached is marked as visible, we replace the leaf using the occluder and its shadow edges. Otherwise, if the leaf ID is marked as occluded, we leave it unchanged. The trees are merged along the lines of the BSP tree merging described by Naylor [Naylor92]. It is possible to think of the KD-tree as a BSP tree which has all the partitions being axis-aligned. We do an intersection operation and we set the operand for intersecting a cell of the SVBSP tree with a part of the KD-tree to display all the objects if the cell is marked as visible (Figure 5.14). If the cell is marked as occluded, the height of the KD-tree node is compared against the height of the occluded cell. If it is lower, everything in the KD-sub tree is occluded. If it is higher, we test each of the children of the KD-tree recursively. If a leaf is still higher, we render the objects in it.

A general merging algorithm, although very elegant, can sometimes be slow because of the need to explicitly compute and maintain the sub-hyperplanes of the two trees being merged. However in our case the problem is simpler than the general case and we can use that to our advantage. Since we are inserting the SVBSP tree into the KD-tree, it is the sub-hyperplanes of the latter that need to be computed and inserted into the SVBSP tree for the partitioning.

It should be noted that the hyperplanes of the KD-tree are trivially computed at almost no cost since they are axis aligned edges and they remain unchanged throughout. The SVBSP tree though is not axis aligned and it changes every frame. At first sight it seems difficult to find a fast way of computing these, but the tree is essentially a 1D structure with height. Basically all the planes start from the viewpoint and end at the far clip plane. The planes can be parametrised with just one value, the angle around the viewpoint. Except for those of the occluders themselves which are always bounded by the shadow planes and therefore cannot extend out and intersect any other object (their sub-hyperplanes are the same as the edges themselves), so there is no need to compute them explicitly. Sometimes during the

partitioning of a tree the face on a node that was separating two sub trees falls entirely on one side of the partitioner and thus we need to merge together the parts of the two sub-trees which fall on the other side of the partitioner.

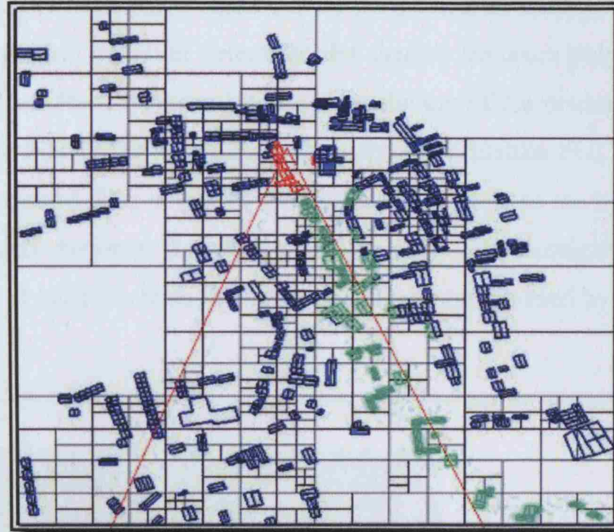

Figure 5.14 - Computing visibility.
The occluders are reported in red, while the green buildings are the only geometry that is visible.

In such a case we will normally need to find the sub-hyperplane of one of these sub-trees to use for the merging. Due to the 1D nature of the tree we can just do point insertion of one of the sub-trees (using any point on that sub-tree) into the other, resulting in a very fast algorithm.

5.8 Approximate shadowing of animated impostors

As introduced in Chapter 4, a 2D shadow height map can be used to store information about the shadow volumes projected in a given urban scenario. A height map is used to represent the area covered by the shadows of the static scene, and to compute, for a given light source, the shadow planes enclosing the shadow volumes. Using this information, it is then possible to compare the height of the people with the height of the shadows and to compute the degree of coverage of a human accordingly, and a two-regions texture can be used to simulate the effect of the shadow over the impostor. The shadow planes can be computed in various ways; the

strategy adopted was to place a plane below the whole scene, searching for the intersection between this plane and the rays going from the light source to each vertex in the urban model. The original vertices together with the projected ones define, for each edge, a shadow plane. The discretisation of the shadow volumes can be performed in a similar way to what we use for the height map used in collision detection; the shadow volumes polygons are rendered in an off-screen buffer and from a viewpoint placed on the top of the model, using an orthogonal projection. From the z-buffer of this image, the depth information that we store at the usual grid resolution is extracted. The height of the shadows relatively to the height of the objects is given by the difference between the shadow height map and the original height map of the geometry (Figure 5.15 (a) ,(b)). Note that this is 0 in area non-covered by shadows.

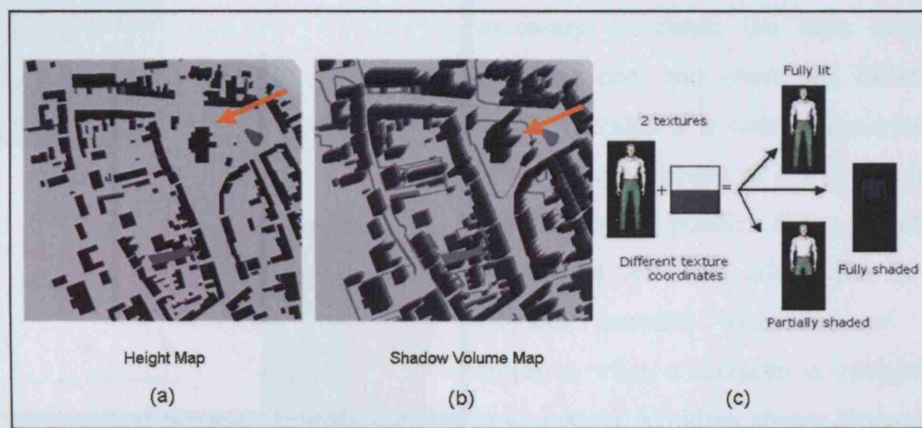

Figure 5.15 – Height Map (a) and Shadow Volume Map (b) computed for the same scenario. (c) a secondary 2-colours texture is overlapped to the impostor for additional shadowing effects.

Once this information is available, impostor shadowing is performed as it follows: while characters move, their position is located on a 2D grid (usually the same used for collision detection and ambient lighting). For each cell occupied by an agent, we check whether the current height of the individual is higher than the height of the shadow stored in the shadow volume map. The difference in height gives the percentage of coverage of the shadow on the impostor. Since the discretised shadow volumes have a convex configuration, the number of cases is limited to uncovered, partially covered and fully covered. When detecting in which case the shading of the impostor corresponds, we set up the tag for the display. When fully covered and uncovered, the impostor support polygon could be rendered either in white or

grey. When partially covered, we use a two regions (black and white) texture mapped onto the impostor to reflect the shadow boundary (Figure 5.15(c)). When possible, this secondary texture is applied using the multi-texturing feature of the graphic cards. When a second texture unit is not available two rendering passes can be used instead.

5.8.1 Shadow texture coordinate computation

To correctly map the shadow texture, the appropriate texture coordinates need to be computed for each impostor. As shadows rarely describe horizontal boundaries in a city lit by the sun, the elevation value of the shadow normally varies between adjacent cells. To detect the inclination of the shadow, it is therefore necessary to check the cells around the occupied one, and check the difference in depth. The idea is to compute an average out of the shadow height of the neighbour cells, weighted by the position of the particle inside the current cell. This information is used to simulate smooth transitions of shadow elevation when a character is navigating the environment, avoiding abrupt changes when going from one cell to the other. Also, since the particle can occupy any position inside a single cell of the discretisation, proper shadow computation needs to consider the effective position of the impostor within the cell. For faster computation, we separate the neighbourhood into four quadrants, described by the connection of the middle of each cell (Figure 5.16). The weighted average is done only considering the cells of the quadrant the particle is in. Finally, as a shadow boundary can be computed for each side of the impostor, it is also possible to display the inclination of the shadow. This has to be done at rendering time, because the borders of the impostor are view dependent.

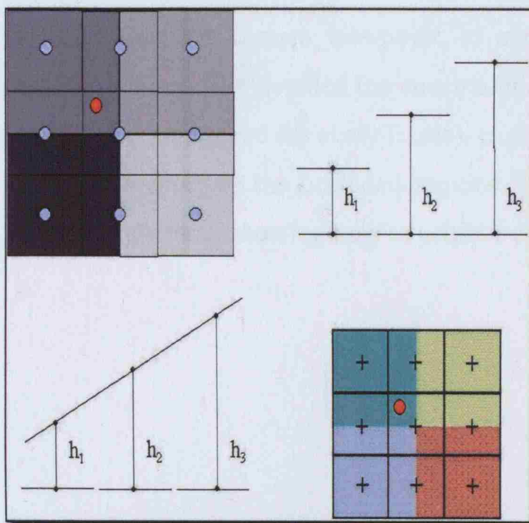

Figure 5.16 - Filtering the values of adjacent cells to produce smooth shadow transitions.

5.9 Summary

In this Chapter we have reported on some important details of the crowd rendering system that bear on real-time performance. We have discussed the main factors influencing the overall speed of the crowd rendering pipeline and presented how in our system impostors are created, stored and retrieved. We detailed the data that is stored to represent the crowd as well as how this data is compressed, segmented and organised (both at pre-processing and at run time). We have also presented some implementation details on our occlusion culling algorithm for crowds: a conservative technique based on from-region visibility that uses a combination of data structures (a kd-tree and a SVBSP tree) to classify which parts of the scenario are occluded from the camera viewpoint, to avoid the rendering of impostors that are inside occluded regions. We detailed the construction of the kd-tree (done at preprocessing) and the SVBSP tree (built once for every frame), explaining how the two trees are merged together at run time to compute the occluded regions. Finally, we have provided some insight on our shadowing method, showing how to achieve approximate building-to-impostors shadowing.

Chapter 6 - Performance analysis

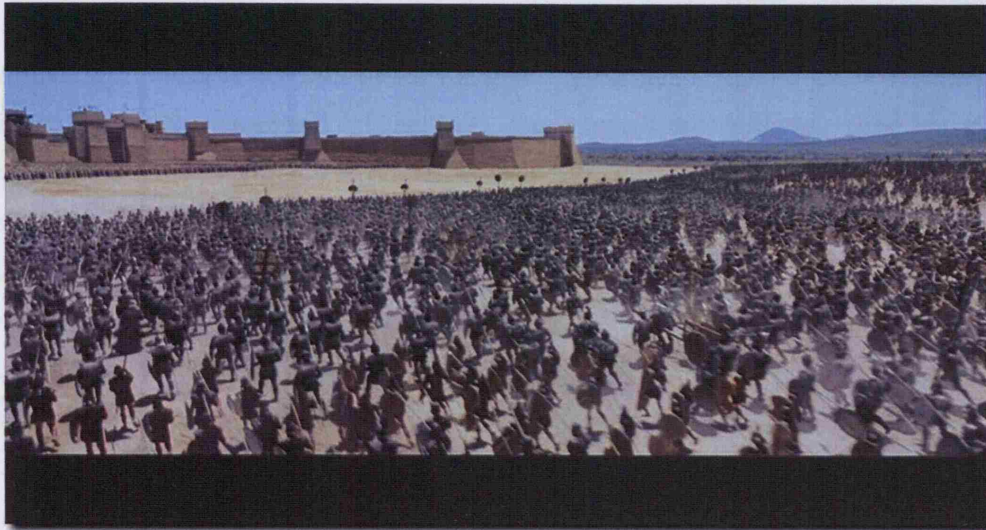

Figure 6.1 - A scene from the movie 'Troy' (Warner Bros. - 2004).

This chapter presents the analysis of the performance of the Impostor based crowd rendering system introduced in Chapter 5. We show the results of tests of speed and scalability performance on a number of virtual scenarios that are populated with crowds. There are three types of test employed: first we measured the raw speed of the unstructured impostors approach after populating the scenarios with large crowds, showing that real-time visualisations of complex environments populated with crowds exceeding thousands of individual are in fact made possible by this method, followed by a detailed analysis of the factors that affect the performance of the technique. We then compared the speed of this approach to plain polygonal rendering to give a measure of the extreme performance boost that can be achieved using an impostor based representation in a practical case. Finally, some experiments were conducted to measure the impact on the impostors rendering performance of the various advanced techniques presented in Chapters 3 and 4: approximate dynamic lighting, per-pixel shading, ambient lighting effects, shadows casting/receiving, and visibility culling.

6.1 Thesis Objectives Revisited

The motivation of this research was the need to populate complex virtual environments with large crowds as a large population is an essential part for the perceived realism of a computer-generated urban scenario. To achieve the necessary speed and be able to render thousands of animated human-like characters in real-time, the image-based rendering approach of the unstructured impostors was used. The goal was to be able to render more than 10,000 animated individuals in real time with sufficient variety and visual detail. An important requirement was for the resulting rendering system to have the ability to be plugged into existing scene graph systems, implying the capability to coexist with the traditional polygonal-based rendering of complex scenarios. OpenGL1.1 was used for the main parts of the system in order to maintain compatibility with the greatest number of systems, including the Cave system at UCL, which was driven by a SGI Onyx reality engine, but we have also tested the system on platform equipped with OpenGL1.3, as we wanted to investigate, when available, the advanced effects made possible by the the additional features available in this API such as per-pixel dot product and texture compression, with the purpose to prove the flexibility of this approach and exploit its full potential.

6.2 Methods of Assessment

All the algorithms proposed in the previous chapters were implemented in C++, and integrated inside existing scene-graph managers, therefore providing the opportunity to conduct analytical tests and to assest the performance and characteristics of the method. We selected some urban scenarios of increasing complexity and suitable for population with virtual crowds. Three sets of tests were then performed: first the raw speed of the unstructured impostors approach, populating the scenarios with large crowds and measuring the rendering speed achieved when several parameters of the rendering system were varied (total population number, use of multi-pass approach, number of frames of animation, use of per-pixel lighting an such). Then we compared the speed of the approach to plain polygonal rendering to measure the performance boost of the impostor based representation over the polygonal one in a practical case. Finally, we measured the speed of impostors rendering when advanced techniques such as impostors approximate dynamic lighting, per-pixel shading, ambient lighting effects, shadows casting/receiving, and visibility culling are used. Relevant tests were performed on two different target systems both equipped with OpenGL1.3 compliant graphics

hardware but with different performance level and a different amount of on-board dedicated video memory, to study the dependency of the method from the PCs levels of performance.

6.3 Scenarios and characters models

We selected for our test three urban-like scenarios, each having its own peculiar characteristics. The simpler scenario (Test Scenario1 - Figure 6.2) represents a block of a simplified urban model, with a variety of smaller and taller building, narrow alleys and larger streets and is composed by 3,216 triangles. The simplicity of this scenario allows the use of most of the graphics resources to the rendering of the crowd. The model was created using the commercial modeler 3DStudioMax.

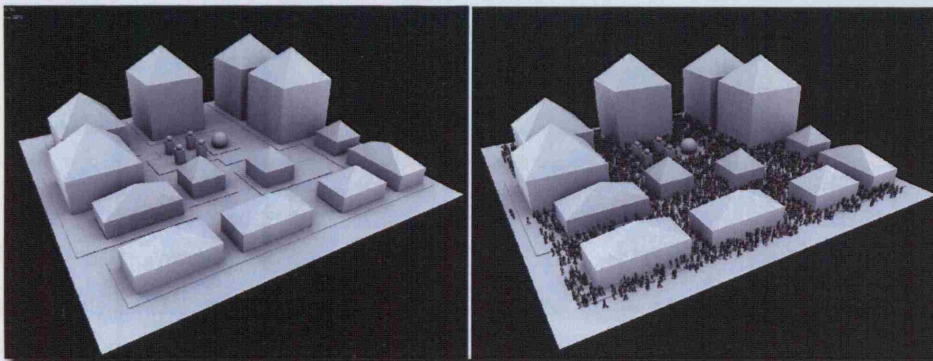

Figure 6.2 - Test Scenario 1: a simple urban-like scenario (3,216 triangles).

A second scenario used in the tests is the Garibaldi Square model, a 3D replica of real city square located in the city of Nizza, France (Test Scenario2 - Figure 6.3).

Figure 6.3 - Test Scenario 2: the Garibaldi Square model (43,866 triangles).

The Garibaldi Square model was employed in the European-funded project CREATE [Loscos03], and it is composed of 43,866 triangles. A large collection of real photographs and the commercial software *Image Modeler* from the company RealVIZ [RVImgmdl06] was used for its creation. This model has more complex global illumination properties, especially in the area of the porches around the square perimeter, and was used for the tests involving the illumination algorithms presented in Chapter 3 and 4.

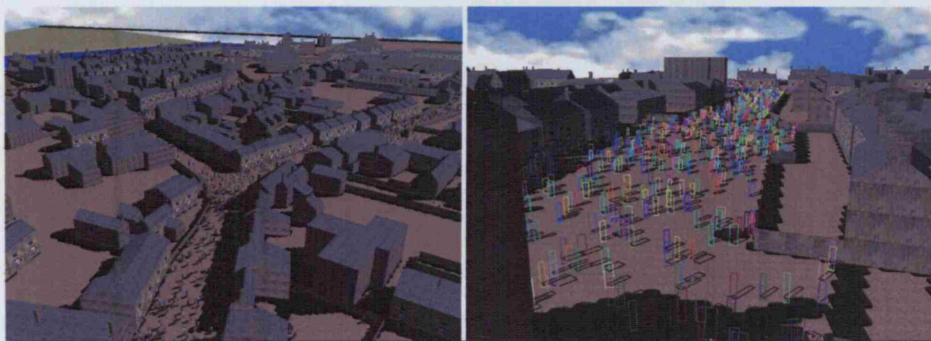

Figure 6.4 - Test Scenario 3: the town model (41,260 triangles).

The third scenario was a simple city model consisting of 41,260 triangles, representing a simplified urban environment (Figure 6.4). It was used to perform measures on the effectiveness of the occlusion culling algorithm, and for some performance tests on shadow-related algorithms. To create the initial impostor database, as well as to measure the speed benefits of impostor rendering against plain polygonal rendering, three sets of models of human characters were used (Figure 6.5). Two of the triangular meshes are highly detailed (the men and the woman characters in Figure 6.5-a and 6.5-b), and were created using the commercial animation program Poser. The simpler mesh in Figure 6.5-c is a freely available VRML 2.0 model. Models (a) and (b) possess a variety of animation key frames. Each key frame is stored statically as an independent polygonal mesh, and no skeletal animation/deformation is performed: every key frame has its own set of vertices data, while they all share the same connectivity information and material properties structure. This arrangement allows for very efficient rendering, but it leads to higher memory requirements compared to skeletal animation, as quite an amount of memory is needed for each key frame to store all the vertices data. The geometry is passed to the graphics hardware in the form of OpenGL display lists, to reduce the CPU-GPU data traffic and maximize rendering speed.

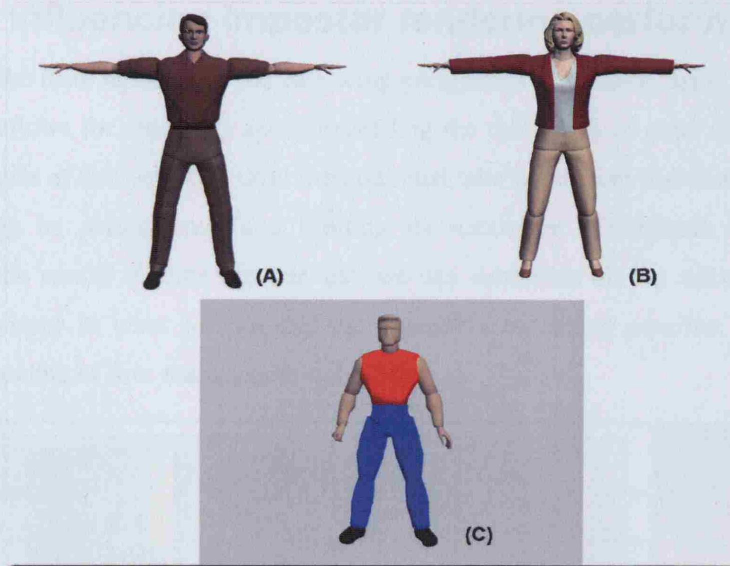

Figure 6.5 - The character models used in the tests.
(a),(b): Two detailed polygonal meshes (21,414 triangles, 27,376 triangles)
(c) Low detail polygonal model (2,006 triangles).

6.4 Test systems

Measures on the implemented test system were conducted on two different computer configurations, belonging to different performance class. In the following, *Tests System Low* is the conventional name used for the less powerful machine, while *Test System High* is the faster one. Both machines use Microsoft Windows XP, but they differ in processor speed, maximum theoretical rendering speed, amount of system and video memory, and in the technology used for the data bus connecting the CPU to the GPU, all factors that have an influence on the performance of our impostor methods.

Test System Low:

Pentium 4 – M
1.7 Ghz – 512 Mb RAM
ATI Radeon 7500 M – 32 Mb RAM
AGP Bus

Test System High:

Pentium 4 - M
2.0 Ghz – 1024 Mb RAM
NVIDIA GeforceFX 6800 Go – 256 Mb RAM
PCI Express Bus

6.5 Factors influencing impostor rendering performance

As the results of the tests reported in the following paragraphs will show, using impostors to visualise crowds allows for rendering speed exceeding the thousands of units in real-time. A performance analysis of this very powerful method must take in account that there are several factors influencing its performance and limiting its maximum throughput. To help the interpretation of the results obtained in our test, we can subdivide all the activities that are performed every frame in what we can call the *impostors rendering pipeline,* reported in Figure 6.6 as a pipeline of four main functional blocks:

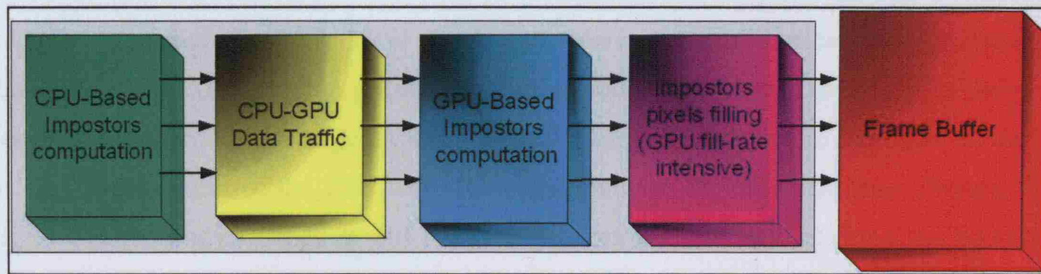

Figure 6.6 - Representing the impostor rendering pipeline as a sequence of 4 functional blocks.

The first block (*CPU-Based Impostor Computation*) is intended to group all the activities performed at the CPU level. This includes several operations: selection of the best fitting impostors image, computation of the impostor geometrical parameters on the basis of the current view-point position (this activity is performed on a per-impostor basis), impostor orientation and frame of animation, geometrical transformations needed for per-pixel lighting and additional computation related to shadow effects (when they are active). The second block (*CPU-GPU Data Traffic*) represent the operations involved in passing data from the CPU to the GPU such as vertex positions, texture coordinate, connectivity information, texture IDs, colour and alpha related attributes and lighting parameters. As the CPU and the GPU are physically separated, this traffic occurs on dedicate communication buses, such as the AGP bus on older INTEL architectures(slower) and the PCI-Express bus (faster) on the newer systems. The third block (*GPU-Based Impostor Computation*) groups all the operations performed on the GPU to transform the impostor vertex position and to manage the OpenGL state: current colour, texture ID, alpha threshold, per-pixel dot product information, etc.. The last functional block of the pipeline (*Impostors Pixels-filling*) represents operations involved in the rasterization process performed by the GPU to render the impostors.

It is not possible to state which, out of these four functional blocks, is the main bottleneck of the method, as different architectures exhibit different behaviours: on older systems (such as System Test Low) the second and fourth block often have greater impact on performance. The limited bandwidth of the AGP bus (with respect to more modern solution) means that it can frequently get 'jammed' when large amounts of data are passed between the CPU and GPU and the system and video memory. Also, while the low geometrical complexity of the impostor makes relatively light the geometry transformation phase, their rasterization needs millions of pixels to be rendered on screen, placing a lot of stress on the fill-rate capabilities of the hardware; techniques such as multi-pass rendering and per-pixel lighting introduce an additional computational load that is mainly absorbed by the rasterization stage. On the other hand, the evolution of modern graphics hardware toward supporting very complex per-pixel operations, has produced a large increase in the fill-rate capability of the rasterizer, as well as a large improvement of the CPU-GPU bus speed, and on the newer systems (System Test High) it is frequently the first stage (CPU-based Impostors Computation) that is the slowest of the pipeline.

6.6 Test1: Speed and scalability of the basic approach

The first set of tests were used to measure the performance and scalability of the basic rendering technique. An increasing number of walking characters going around in pseudo-random directions are rendered as depicted in Figure 6.7 using impostors only. There are 3,840 image samples in the impostor database, each of them having a maximum resolution of 128 x 128 pixels. The same size was used for all the tests reported in this Chapter. Figure 6.8 shows the average time to render a frame (computed over a sequence of 1,000 frames) against the number of impostors composing the crowd. During the test the camera is kept orbiting around the crowd. At this preliminary stage colour modulation is not yet performed, as the test is used to evaluate the speed and scalability of the basic technique of impostor-based crowd rendering. In particular, the test provides some insight into the speed of the view direction sampling algorithm (black dotted line of Figure 6.8), as well as on the impact that the frame buffer resolution has on the overall performance of the impostors method. The results show that the frame rendering time linearly increases with the number of impostors on both the tests systems (so the method scales up nicely), with the fastest machine showing some non linearity close to the origin of the graph. These deviations from linearity are not due

to the impostor rendering algorithm, but to some peculiarities of our scene-graph implementation, that also imposes a limit of 100 frame per second on the simulation rendering speed.

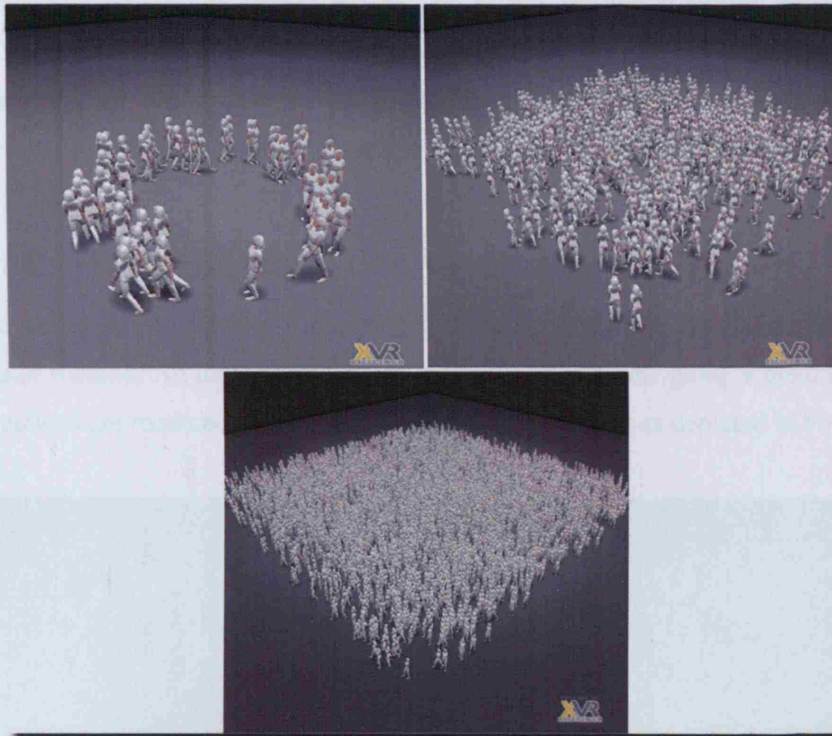

Figure 6.7 - Rendering a crowd with the basic impostor algorithm (colour modulation is not performed and no polygonal characters are rendered).

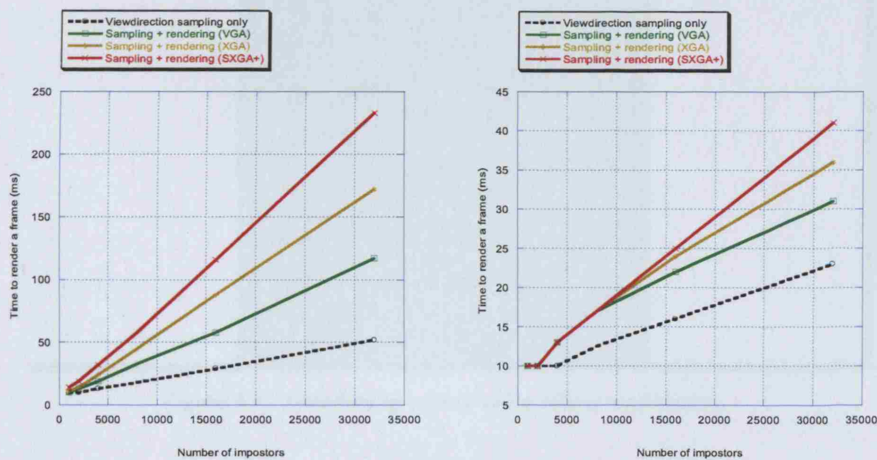

Figure 6.8 - Average time to renderer a frame against the number of impostors. The graphs reports measures obtained with Test System Low (left) and Test System High (right).

The graph shows clearly that an increase of the framebuffer resolution results in a slower overall rendering speed of the impostors, due to the higher fill-rate requirements (a larger number of pixels needs to be rendered for each impostors). Interestingly, the rendering speed of Test System High appears unaffected by the screen resolution unless the population exceeds 8,000 units. In all probability, this is due to the hight fill-rate performance of this machine. In this case, the operation that limits the total impostor throughput is the time needed by the CPU to sample the view direction and to compute the correct impostor image.

6.7 Test 2: The multi-pass colouring approach

This second test is used to measure the impact of the technique of colour modulation over the basic impostor method. An increasing number of walking characters going around in pseudo-random directions are rendered using impostors colour modulation as depicted in Figure 6.9.

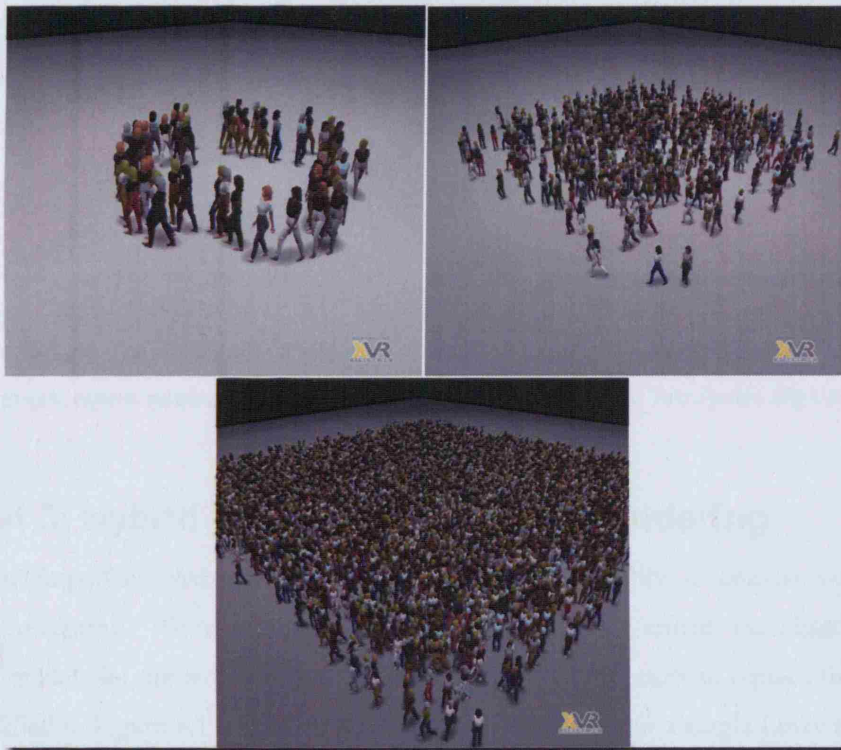

Figure 6.9 - Introducing variety using colour modulation.

During rendering the camera is kept orbiting around the crowd. Figure 6.10 shows the average frame rendering time (computed over a sequence of 1,000 frames) against the number

impostors composing the crowd. The framebuffer resolution is kept at XGA resolution (1,024 x 768 pixels). The results still show a linear increase of the frame rendering time for a growing number of impostors on both the tests systems. As expected, the total rendering time is influenced by the number of passes used for the impostors, as this increases the fill-rate requirements of the technique; more rendering passes provide the ability to control the colour of more regions on the impostor image, but also slow down the rendering. From the graph can be seen how in the case of 4 rendering passes the System Test High machine exceed 20 Fps for a population of 15,000 individuals. The same framerate is achieved by System Test Low for a population of 5,000 individuals.

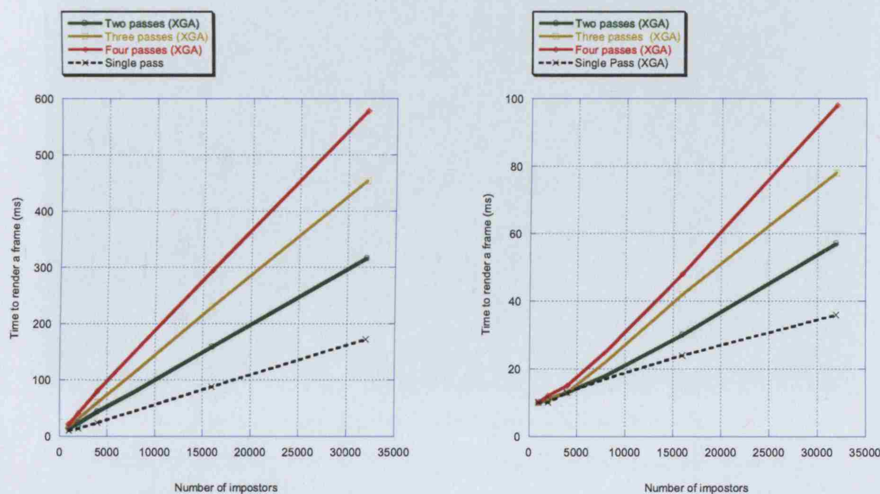

Figure 6.10 - Average time to renderer a frame against the number of impostors.
The graphs reports measures obtained with Test System Low (left) and Test System High (right)

6.8 Test 3: Hybrid polygonal - impostor rendering

One of the important characteristic of impostors is their ability to coexist with normal polygonal rendering. We conducted several tests on hybrid crowd visualisation, using polygonal models for the individuals in the foreground and impostors to replace the far ones, as exemplified in Figure 6.11. In the context of the present section a single Level of Detail is used for the polygonal mesh, while section 6.9 analyses what happens when impostors are introduced in a LODs-based rendering system.

Test Scenario1 was populated with a crowd of 5,000 individual (Figure 6.12). One of the high-complexity triangle meshes reported in Figure 6.5-b is used for each individual in case of

polygonal rendering (the detailed female model), while we use for the impostors the same database of 3,840 image samples as used for Test1. Modulating the colour of the different body part of each individual is used both on the impostors and on the polygonal model to enhance the crowd visual variety. Colours are synchronized between the polygonal model and the impostors. In this case, the polygonal models are pre-lighted with a generic skylight effect (using texture mapping), so switches between the two representations are hardly noticeable.

Figure 6.11 - Mixing polygonal characters and impostors.
Polygonal meshes are used for the characters in the foreground, while the background individuals are rendered using impostors. Impostors colour modulation is kept off for illustration purposes only.

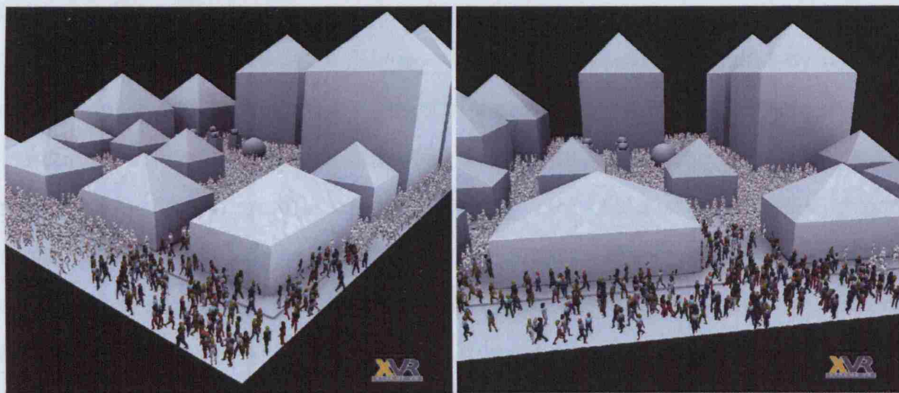

Figure 6.12 - Mixing polygonal characters and impostors (Test Scenario 1).

The scene is rendered with the virtual camera moving along a precomputed trajectory consisting of 1,000 discrete steps. For each step, corresponding to a frame of our simulation, we measured the frame rendering time, including the time to render the crowd as well as the time to do the rendering the model of the urban environment. The camera transits along a descending oval and is always pointed to a fixed target, placed near the centre of the environment (Figure 6.13).

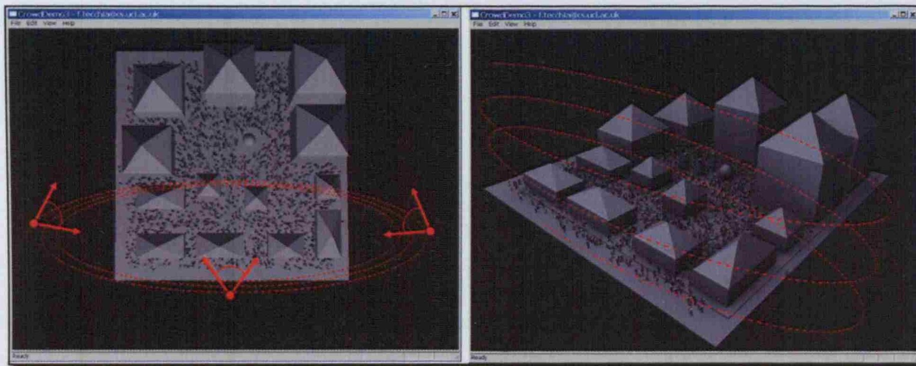

Figure 6.13 - The camera trajectory used for Test 3.

This trajectory has a strong impact on the rendering system in several ways: when the camera is high over the environment, perspective projection makes the individuals appear very small on screen. As they occupy now less pixels, both polygonal rendering and impostor rendering makes it less fill-rate intensive. This has only a limited performance impact in the polygonal case, as rendering speed is mostly limited by the geometrical transformation of the vertex data (represented by the GPU-based impostor computation functional block). On the contrary, impostors benefit from having a smaller on-screen size (as demonstrated by Test1) as their rendering speed is basically fill-rate limited. On the other hand, when the camera is high over the environment our impostors algorithm tries to keep in texture memory a larger amount of image samples, due to the amount of different impostor inclinations present at once in the scene, as pointed out in Chapter 5, thus placing more stress on the memory management procedure. As the camera advances over its precomputed path, it's elevation with respect to the ground level becomes smaller, resulting in the need to maintain in video memory a smaller variety of impostors image samples, but at the same time requiring to render larger impostor images, thus placing more load on the triangle rasterizer. The camera path also offer a good

mix of visibility conditions, varying from full visibility of the scene in proximity of the more external points of the curve, to the restricted visibility of the central parts of the trajectory.

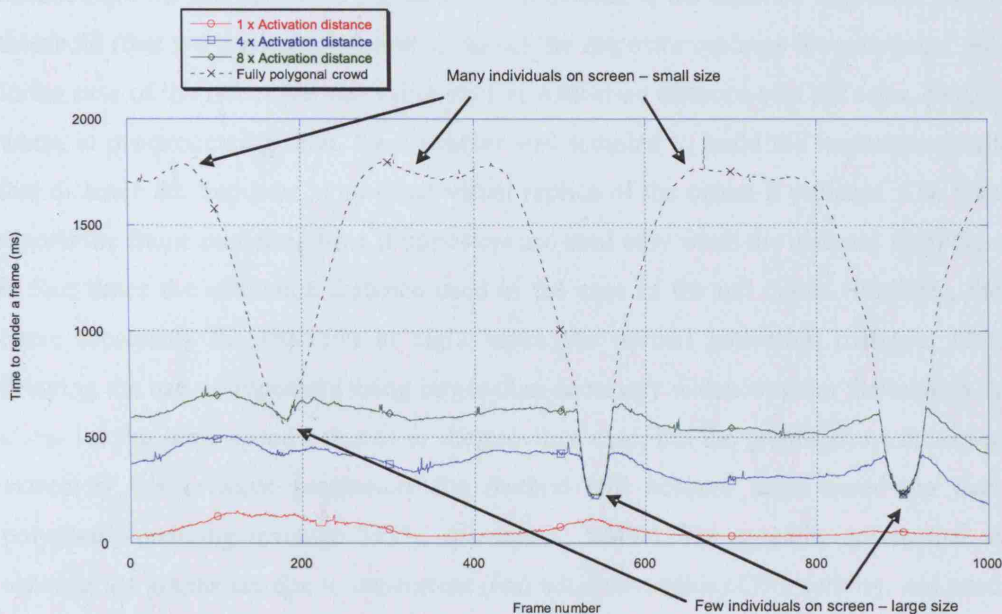

Figure 6.14 - Measuring how the introduction of impostors affects the rendering speed of large crowds. The black curve refers to plain polygonal rendering, while the red, blue and green curves show the rendering speed (for different activation distances) when impostors are used to replace the characters in the background. Each impostor replaces a polygonal mesh composed of 21,414 triangles.

Figure 6.14 shows the results obtained during the simulation (Test System High was used in this case). Four different measures are reported: the slowest simulation (in black) refers to plain polygonal rendering - no impostors were used in this case and even the more distant individuals are rendered with their full geometry. It can be seen that the frame rendering time has a cyclical profile. This is due to the different visibility resulting from the camera trajectory and the consequent view frustum culling. Frames with less individuals in the field of view are the fastest to be rendered. It must be noted also that no visibility culling (neither view-frustum culling nor occlusion culling) was explicitly performed by our framework; it is likely that the higher speed in regions with a lower visibility may be due the view frustum culling performed directly by the OpenGL graphics driver. The fastest rendering, reported by the red curve in the graph, was obtained when using the polygonal models for the rendering of the foreground individuals and impostors for the characters in the background. It is easy to see from the

graph how large a speed gain can be obtained by the introduction of our impostors technique in the rendering of large crowds. The distance of each character from the camera is used to choose between the polygonal model or the impostor: if the distance is greater than a given threshold (that we named *activation distance*) the impostor replaces the polygonal geometry. In the case of the red curve, the value used as activation distance was the same distance used when, at pre-processing time, the character was sampled to build the impostors database: at that distance the impostor is an exact visual replica of the object it replaces. The blue curve reports the frame rendering time if impostors are used only when the distance from the camera is four times the activation distance used in the case of the red curve. Similarly, the green curve represents the situation at eight times the normal activation distance. Obviously, delaying the use of impostors using larger-than-necessary distances from the camera does not allow for the same speed gains as in the previous case, but the graphs show that even using extremely conservative parameters the method still achieve large speed-ups over plain polygonal rendering (average 353%, maximum 504%). The sporadic spikes that at times occur in the graphs are due to concurrent (and not controllable) CPU activity, and need not to be considered caused by any simulation or rendering activity.

The test was then repeated using a simpler polygonal model to represent each individual in the crowd (we used the mesh depicted in Figure 6.5-c, composed by 2,006 triangles).

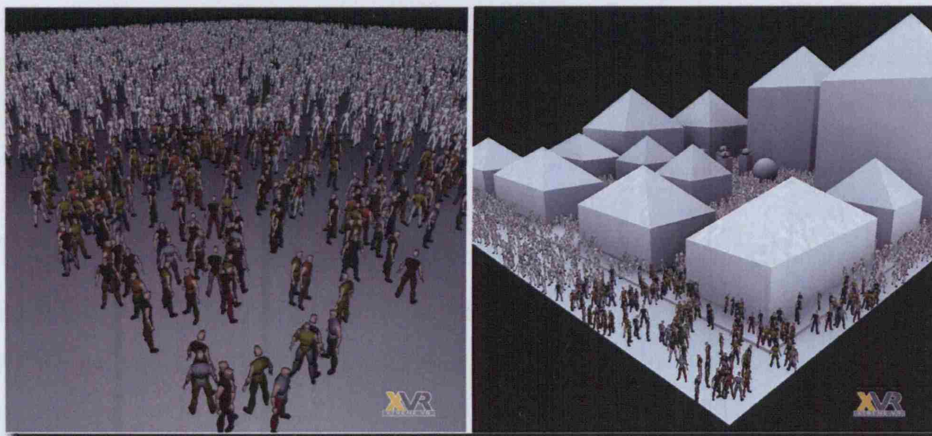

Figure 6.15- Mixing polygonal characters and impostors.

Simple polygonal meshes are used for the characters in the foreground, while the background individuals are rendered using impostors.

In this case impostors replace much simpler polygonal objects, so that the benefits introduced by impostors could be expected to be inferior than to the previous case. The graph in Figure 6.16 shows the measures that we get from the test, repeating the criteria used before (a total

population of 5,000 individuals, 3 different choices for the activation distance). The average rendering speed is in this case higher than before, but the same general consideration are still valid: replacing distant individual with impostors allows for large speed gains over the use of plain polygonal rendering, even when using extremely conservative parameters.

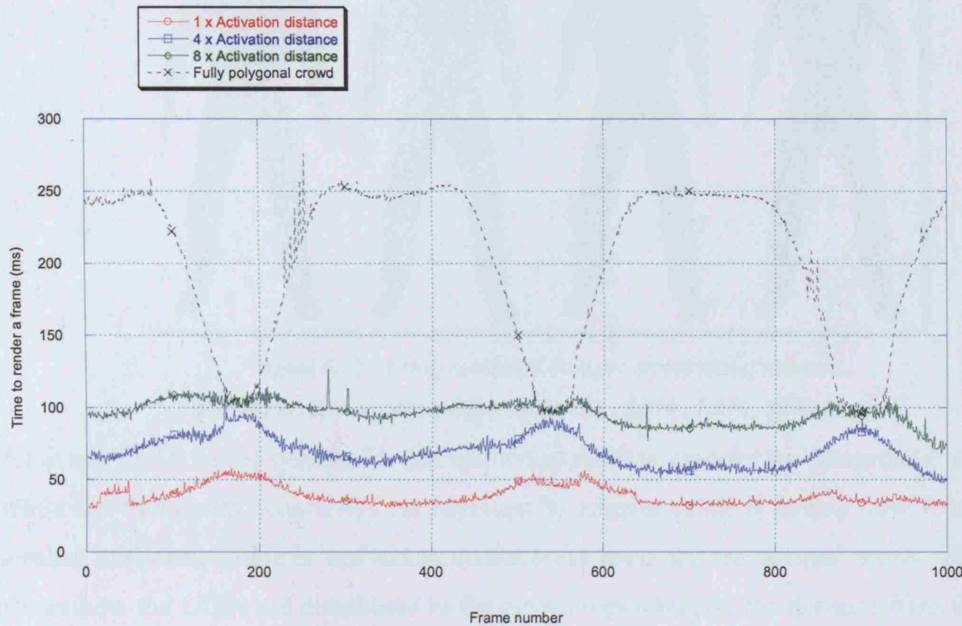

Figure 6.16 - Measuring the benefit of impostors when they replace simpler polygonal models. The black curve refers to plain polygonal rendering, while the red, blue and green curve measure the rendering speed (for different activation distances) when impostors are used to replace the characters in the background. Each impostor replaces a polygonal mesh composed of 2,006 triangles (curves appear noisy due to the limited granularity (1ms) of the timer).

6.9 Test 4: Speed comparison against LODs rendering

Additional tests were conducted in order to measure the impact on performance of the use of impostors when they are added to a rendering system capable of Level Of Detail management. We repeated the rendering conditions of Test 3 using for the animated crowd 4 polygonal LODs consisting of 21,414, 5,070, 1,876, 920 triangles. The four meshes (Figure 6.17), were generated using automatic simplification inside 3DStudioMax. It must be noted that any automatic simplification process is prone to introduce visual artefacts in the resulting meshes; we purposely ignored this effects, as our interest was the speed comparison between a purely polygonal approach against the combination of polygons and impostors. A manual (time-

consuming) simplification of the polygonal models it is likely to produce better looking model, but this has no effect on the final frame-rate of the simulation.

Figure 6.17 - Using multiple LODs for crowd visualisation.
From left to right, model complexity:21,414 , 5,070 , 1,876 , 920 triangles.

As in any LODs based system, the full resolution mesh is used for the foreground individuals, while the background characters are replaced by simpler models; having now four different meshes available, multiple activation distance are used for the various levels. Figure 6.18 shows how the LODs are distributed in the crowd depending on the distance from the camera (fake colouring of the various LODs is used for illustration purposes).

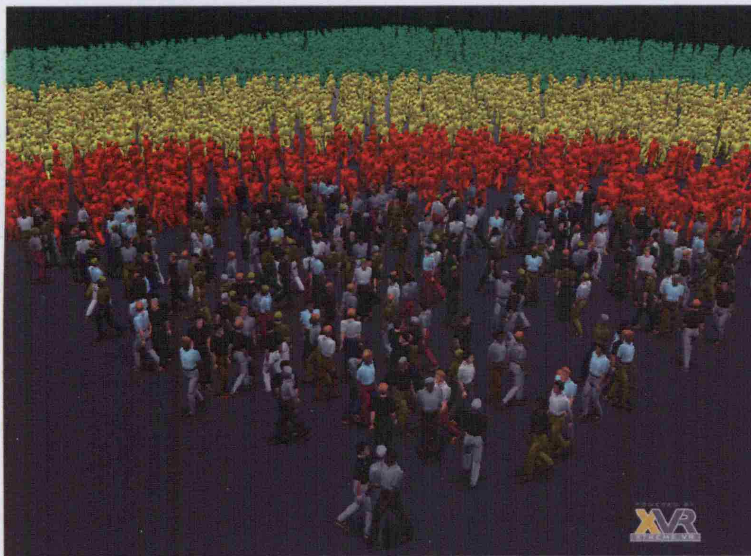

Figure 6.18 - Using static polygonal LODs to render a crowd.
Detailed meshes are used for the characters in the foreground, and simpler static LODs are used as the distance from the view point increases (fake colouring for illustration purposes).

Similarly to Test 4, a crowd of 5,000 individual is used to populate the Test Scenario 1, as depicted in Figure 6.19. The impostor database remains as in the case of Test2, and is created from the high-resolution mesh. When the simulation runs, the virtual camera moves inside the environment following the same precomputed animation path of Test2. For each step, the total rendering time was recorded, including both the rendering of the crowd and the rendering of the urban environment model. The OpenGL framebuffer is kept at XGA resolution (1024 x 768 pixels).

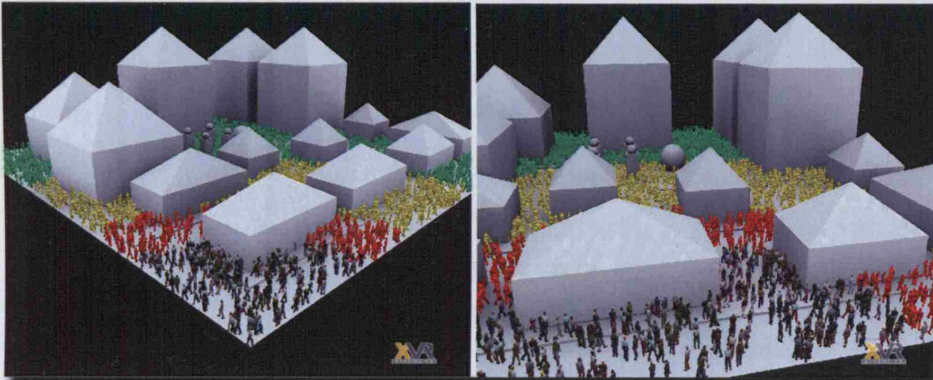

Figure 6.19 - Populating Test Scenario 1 with a LODs - based polygonal crowd .

The test was executed under several different conditions, gradually introducing the use of LODs and finally impostors: the black curve in the graph of Figure 6.20 reports the timing we get when a single mesh (the detailed one) is used for each individuals and impostors are not used. The red curve shows what happens when we start using a single additional LOD in the simulation: the second most detailed mesh (5,070 triangles) replaces in this case all the individuals in the background. This produces a speed-up of the rendering, but the beneficial effects are still limited. With the introduction of a second LOD (blue curve) the simulation is much faster. In this case 21,414 triangles are used for the foreground characters (the detailed mesh), 5,070 triangles for the middle ones, and finally 1,876 for those in background. In terms of speed, the best results obtained with polygonal only rendering are shown by the yellow curve: all 4 of the LODs, including the 920 triangles model, are in use, and the system shows a very large speed gain (average 315%, maximum 430%) over the single detailed mesh case. Still, the curve reported in green shows that the speed obtained using just a single LOD plus the impostors for the distant characters is much higher then in all the other cases.

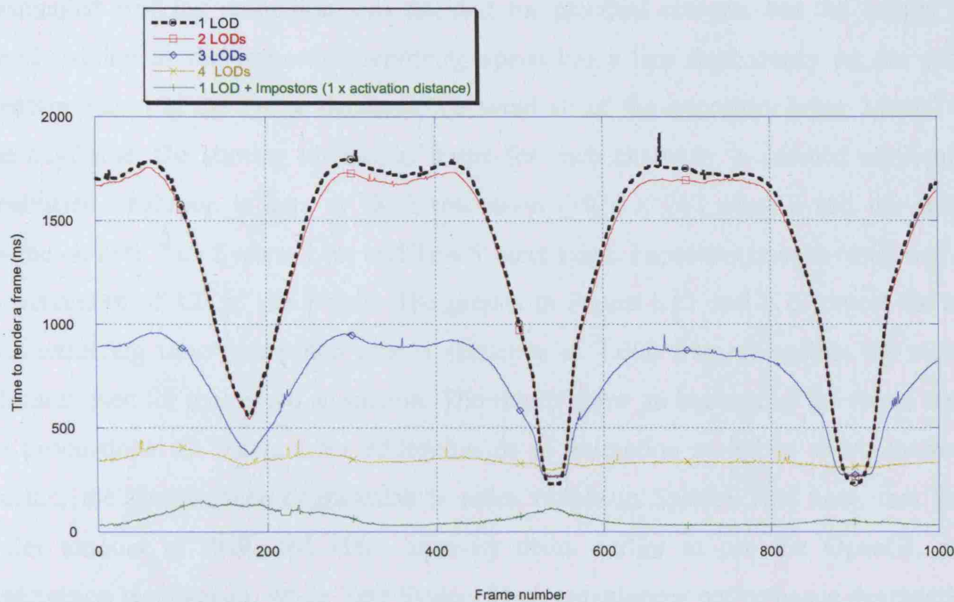

Figure 6.20 - Measuring rendering time in the LODs - based system.
The black curve is referred to plain polygonal rendering, while the red, blue and green curve measure the rendering speed (for different activation distances) when impostors are used to replace the characters in the background. Each impostor replace a polygonal mesh composed by 2,006 triangles.

6.10 Test 5: Increasing the number of samples

An important aspect of impostor-based crowd rendering is related to the amount of texture memory needed to store the impostors data. As discussed in Chapter 3, sampling key-framed animated geometry implies that many images have to be independently stored in memory; for crowds that are composed of a large variety of animated models this process may generate a very large amount of data. As seen in Chapter 5, when the amount of textures exceed the storage space available on the high-performance video memory of the graphics hardware, a memory virtualisation mechanism of OpenGL transforms the main system RAM in an additional pool of resources. In this case, frequent data transfer operations between the system RAM and the video RAM have a detrimental effect on overall rendering speed.

The following test examined how the performance of the two test systems are affected by the number of samples stored in the impostors database. Similarly to Test 1 and 2, a crowd of walking characters going around in pseudo-random directions is rendered, but in this case the number of keyframes stored in the database is varied, while the total population is always kept at 5,000 individual. During rendering the camera is kept orbiting around the crowd. A single,

oversampled walking animation was decided for practical reasons, but the results have a general validity as the impostors rendering speed has a low dependency on the particular animation stored in the image database. To avoid all of the impostors being 'synced' on the same keyframe, the starting animation frame for each character is decided randomly. The framebuffer resolution is kept at XGA resolution (1024 x 768 pixels), and the tests were repeated on both Test System Low and Test System High. Impostors images resolution is kept at a maximum of 128 x 128 pixels. The graphs in Figure 6.21 and 6.22 report the average frame rendering time (computed over a sequence of 1,000 frames) against the number of keyframes used for the crowd animation. The results show an increase of the frame rendering time proportional to the number of keyframes of animation stored in video memory. As expected, the performance degradation is more visible in System Test Low, that having a smaller amount of dedicated video memory starts earlier to use the OpenGL memory virtualisation mechanism, while Test System High experiences performance degradation at a later stage. The graph also show that even with 600 keyframes stored in the impostors database the System Test High machine exceed 20 Fps for a population of 15000 individuals.

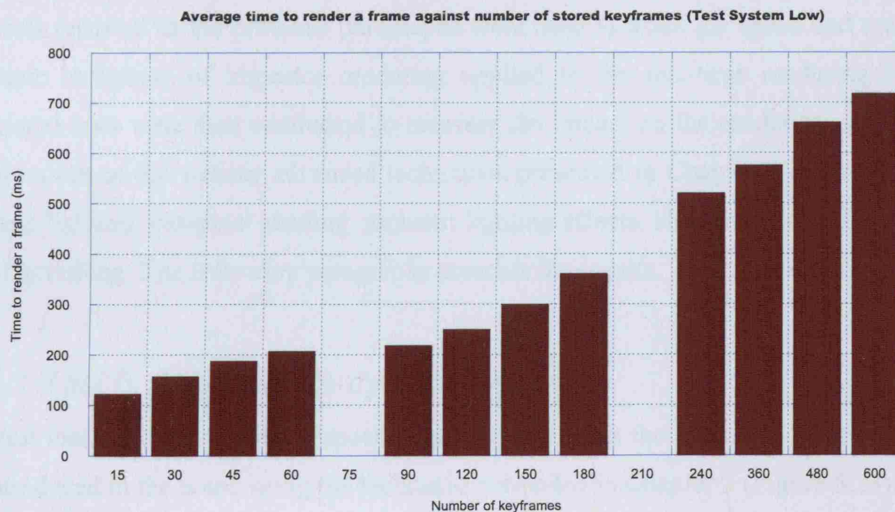

Figure 6.21 - Measuring rendering speed for different amount of impostor data (Test System Low). The graph reports the average time to render a frame when more animation keyframes are introduced.

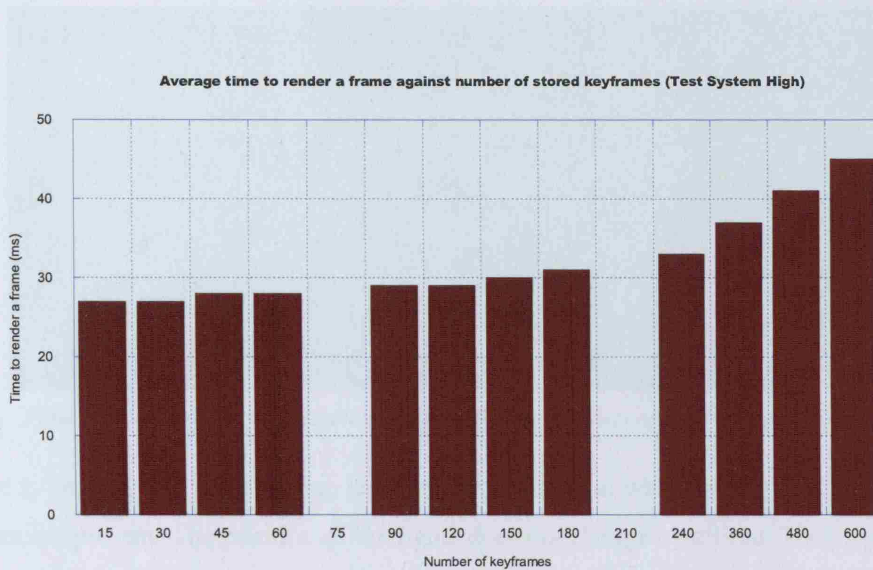

Figure 6.22 - Measuring rendering speed for different amount of impostor data (Test System High). The graph reports the average time to render a frame when more animation keyframes are introduced.

6.11 Advanced impostors effects

The tests reported in the previous paragraphs were used to asses the speed and scalability of the basic technique of impostor rendering applied to the real-time rendering of crowds. Additional tests were then conducted to measure the impact on the rendering performance of this approach of the various advanced techniques presented in Chapter 3 and 4: approximate dynamic lighting, per-pixel shading, ambient lighting effects, shadows casting/receiving, and visibility culling. The following paragraphs presents the results.

6.11.1 Test 6: Approximate dynamic lighting

This test measured the rendering speed of the system when the effects of local lights sources are introduced in the scene using the technique presented in Chapter 3 (Figure 6.23). Similarly to Test 1 and 2, an increasing number of walking characters going around in pseudo-random directions are rendered with the camera orbiting around the crowd; the graph in Figure 6.24 shows the average frame rendering time (computed over a sequence of 1,000 frames) against the total number of impostors rendered and for an increasing amount of light sources.

The black column shows the rendering time when no lights are present (i.e. standard impostor rendering is performed), the red column reports the rendering time when a single local light

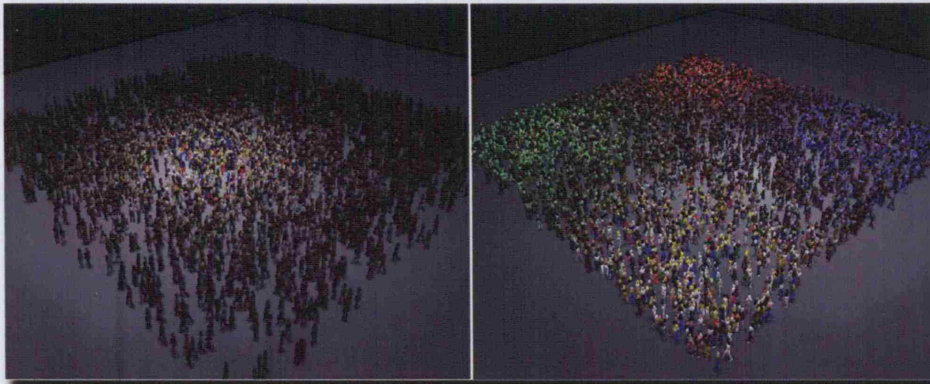

Figure 6.23 - Dynamic illumination of a crowd using a variety of local lights configurations.

source is present, and the blue line is referred to a scenario where 10 randomly coloured light sources are present. The position of the lights does not change over time. The display window was set to XGA resolution (1,024 x 768 pixels). The results show a negligible impact (less that 1%) of this technique on the speed of impostor rendering for both the test systems.

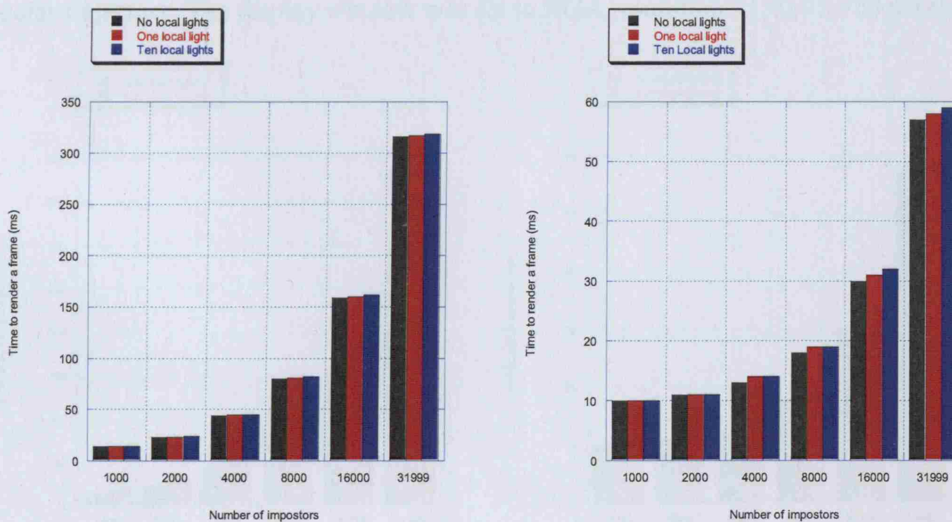

Figure 6.24 - Testing dynamic lighting.

Average time to renderer a frame against the number of impostors for an increasing number of local lights. The graphs reports measures obtained with Test System Low (left) and Test System High (right).

6.11.2 Test 7: Per-pixel impostor lighting

This test measured the rendering speed of the system when the per-pixel lighting (presented in Chapter 3) is activated, and compares the results to standard impostor rendering, where a

generic (and fixed) illumination is hard-coded in the image samples. Similarly to Test 1 and 2, an increasing number of walking characters going around in pseudo-random directions are rendered, while the virtual camera is orbiting around the crowd (Figure 6.25).

Figure 6.25 - Impostors per-pixel lighting. Testing speed and scalability of the method.

The position of the light source changes during the test, moving over the crowd along a circular trajectory. The display window was set to XGA resolution (1,024 x 768 pixels).

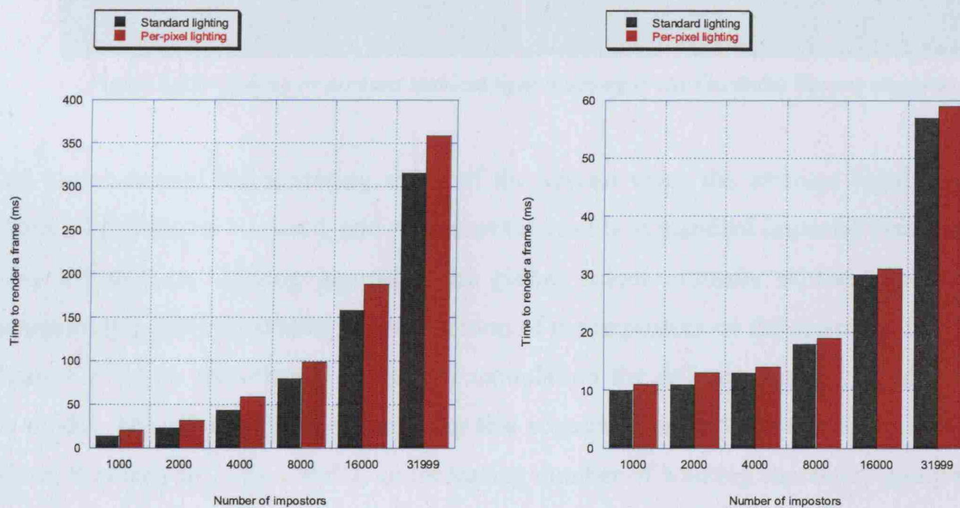

Figure 6.26 - Testing per-pixel lighting.

Average time to renderer a frame against the number of impostors using standard or per-pixel lighting. The graphs reports measures obtained with Test System Low (left) and Test System High (right).

The graph in Figure 6.26 shows the average frame rendering time (computed over a sequence of 1,000 frames) against the number impostors composing the crowd and for an increasing

amount of light sources: the black column shows the rendering time in the case of standard lighting and the red column reports the rendering time when per-pixel lighting computation is activated (a single local light source is present). The graphs show a relatively small impact of this technique (20% slower in the worst case) over precomputed (static) lighting. This is even more evident for Test System High (reported on the right), that apperas to be more efficient in handling per-pixel operations.

6.11.3 Test 8: Ambient lighting

Figure 6.27 - Taking in account ambient light intensity in the Garibaldi Square scenario.

This test measured the rendering speed of the system when the ambient lighting technique presented in Chapter 4 is used, and compares the results to standard impostor rendering. When using the ambient lighting algorithm the global colour intensity of the impostor images becomes modulated depending on the position of the impostors on the scenario. The images in Figure 6.27 show the effects of intensity modulation for different lighting conditions across the model. The 2D light intensity map for this scenario is stored at a resolution of 512 x 512 values. Similarly to Tests 1 and 2, an increasing number of walking characters going around in pseudo-random directions are rendered. During the tests the camera moves along a precomputed trajectory composed by 1,000 steps (that is also the total number of rendered frames), The framebuffer was set at XGA resolution (1,024 x 768 pixels). The graphs in Figure 6.28 report the average frame rendering time against the number of impostors that compose the crowd with and without using ambient lighting modulation. The black column reports the rendering time for standard lighting, the red column reports the rendering time when ambient lighting modulation is activated. The graphs suggest a relatively small impact

(less than 3%) of this technique over using a constant ambient light intensity.

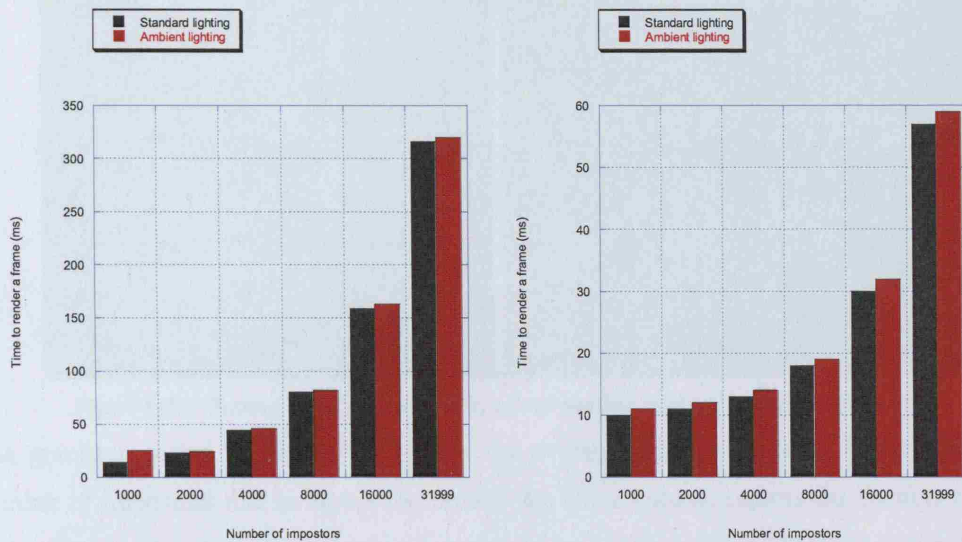

Figure 6.28 - Testing ambient lighting.

Average time to renderer a frame against the number of impostors with or without the ambient lighting algorithm. The graphs reports measures obtained with Test System Low (left) and Test System High (right).

6.11.4 Test 9: Shadowing effects

This test measured the rendering speed of the system when the shadow methods presented in Chapter 4 are used, and compares the results to standard impostor rendering. We populated Test Scenario 3 (the town model) with an increasing number of individuals walking around, performing also collision detection and a simple form of obstacles avoidance. The images in Figure 6.29 show different lighting conditions and different views of the model populated by 10,000 animated people. We tested both impostors-to-environment shadow casting (using the fake shadow technique) and the environment-to-impostors technique, using the information stored in the shadow height map whose creation takes less than 2 second for such a model, including shadow volume vertices computation (which takes only 0.1 second). This map is then stored at a resolution of 4096 x 4096 values, representing for each cell the elevation of the shadow (quantized on 24 bits). During the tests the camera moves along a precomputed trajectory composed by 1,000 steps (that is also the total number of rendered frames). The framebuffer was set to XGA resolution (1,024 x 768 pixels).

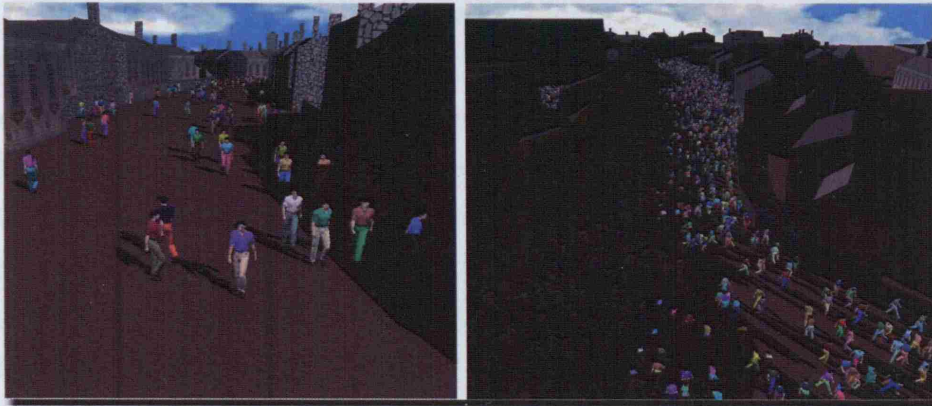

Figure 6.29 - Testing speed and scalability of our shadow methods in the Test Scenario 3.

The graphs reported in Figure 6.30 show the average time to render a frame against the number of impostors that compose the crowd: the black column reports the timings obtained when the populated scene is rendered without any shadow effect. The red column shows timings obtained after turning on shadow projection of the humans on the floor (fake shadows method). The blue column shows the results when everything is simulated, including the shadows projected by the buildings on the humans. The measures reported on the graph show that the additional computational cost of shadow computation and display is fairly low (less than 25%) even when updating more than 30,000 moving characters.

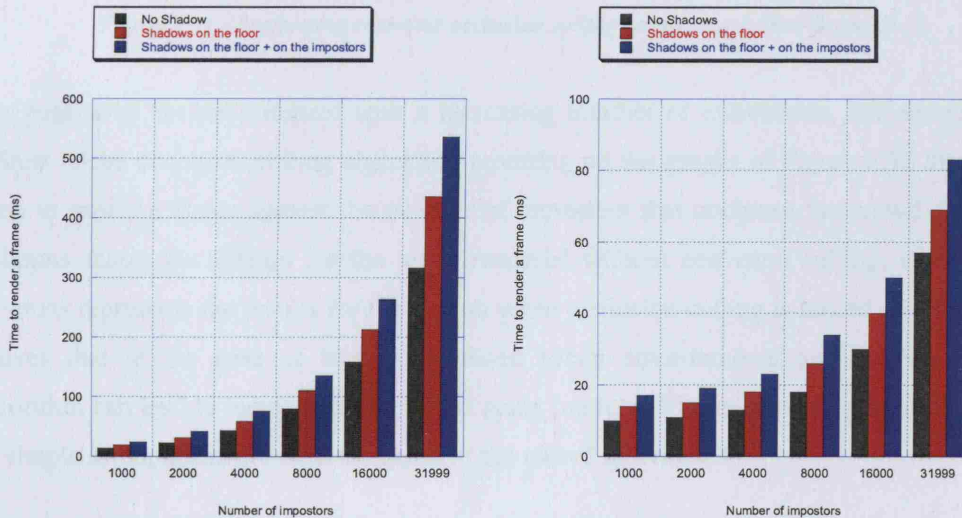

Figure 6.30 - Testing shadowing effects.

Average time to renderer a frame against the number of impostors with or without shadows effects . The graphs reports measures obtained with Test System Low (left) and Test System High (right).

6.11.5 Test 10: Occlusion culling performed on crowd

This test measured the rendering speed of the system with or without the occlusion culling algorithm presented in Chapter 4. We populated the Test Scenario 3 (the town model) with an increasing number of individuals walking around and performing collision detection and a simple form of obstacles avoidance. The left picture in Figure 6.31 shows how the buildings in an urban scenario can be very effective occluders. During the tests the camera moves along a precomputed trajectory composed by 1,000 steps (that is also the total number of rendered frames). The framebuffer was set to XGA resolution (1,024x768 pixels).

Figure 6.31 - Performing real-time occlusion culling on the crowd (Test Scenario 3).

We populated the environment with a increasing number of individuals, and measured the effects of the occlusion culling algorithm, reporting on the graphs of Figure 6.32 the average time to render a frame against the number of impostors that compose the crowd. The black columns shows the timings for the scene rendered without occlusion culling, while the red columns represents the results for the system when occlusion culling is turned on. The figures proves that in the case of highly populated urban environments our occlusion culling algorithm can lead to large rendering speed gains (up to 300%) even when graphics primitive as simple as impostors are used to represent the crowd individuals.

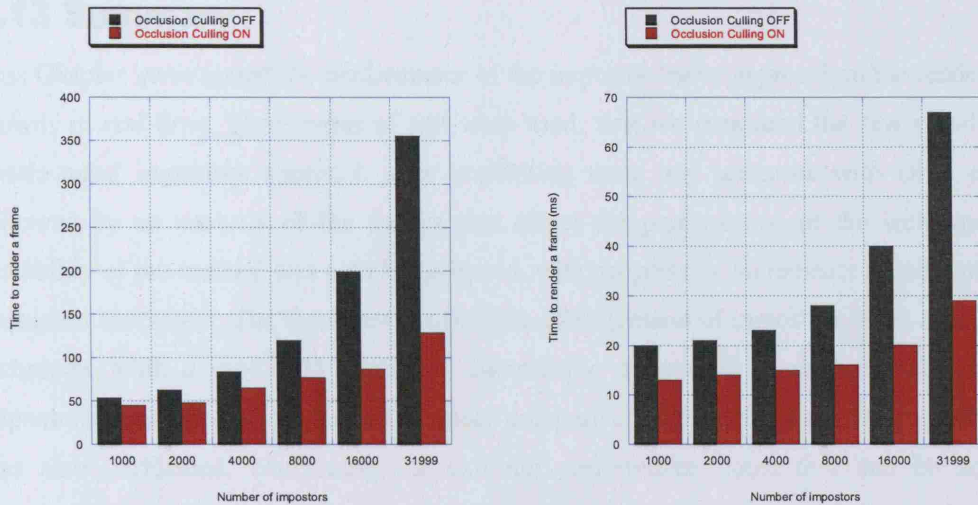

Figure 6.32 - Testing crowd occlusion culling.

Average time to renderer a frame against the number of impostors with or without occlusion culling . The graphs reports measures obtained with Test System Low (left) and Test System High (right).

6.12 Discussion

The tests carried out and discussed in this section demonstrate that image-based impostors can be used to visualise large crowds in real-time. When used to visualise a large number of animated characters, unstructured impostors offer a very good performance/quality ratio, and their high effectiveness as a light yet powerful rendering primitive leads to rendering speeds that are clearly superior to the case of normal polygonal models. With a geometrical complexity independent of the objects they represent, their rendering speed is linear in the number of primitives, as shown in test results reported in Sections 6.6, to 6.9. Even advanced effects, such as per-pixel lighting or shadowing do not introduce significant modifications to the overall rendering speed, as can be seen by tests reported in Section 6.11. With respect to memory usage, the tests reported in Section 6.10 shows that on recent hardware significant variations within the crowd can be achieved before incurring a severe performance degradation; what this means in practice is that crowd variety is more likely to be limited by the amount of modelling resources than by the use of a large amount of texture memory. Finally, the use of impostors makes it possible to visualise very large crowds on common graphics hardware, reaching rendering speeds that would be impossible to reach using plain polygonal rendering.

6.13 Summary

This Chapter investigated the performance of the impostor-based approach to the rendering of crowds in real-time. Three types of test were used: first we measured the raw speed of the unstructured impostors approach after populating three test scenarios with large crowds, followed by an analysis of the factors that affect the performance of the technique. The scalability of the method was also investigated, with progressive increments of the number of simulated characters. The measures confirm the effectiveness of impostors as an acceleration technique. With a very light geometric complexity, the overall rendering performance of impostors is mainly fill-rate limited. A speed comparison with traditional polygonal rendering was also performed, confirming the extreme performance boost that can be achieved introducing impostor based rendering in a generic scene graph implementation, and showing how impostors can improve system performance even when they coexist with polygonal character rendering. Results confirm also that impostors are compatible (and beneficial) when integrated into a polygonal LOD system. Finally, experiments were conducted to measure the impact on the impostors rendering performance of the various advanced techniques presented in Chapters 3 and 4: approximate dynamic lighting, per-pixel shading, ambient lighting effects, shadow casting/receiving, and visibility culling. All tests were carried out on systems having different levels of performance, to better assess the dependence of the methods from the overall hardware architecture.

Chapter 7 - Conclusions

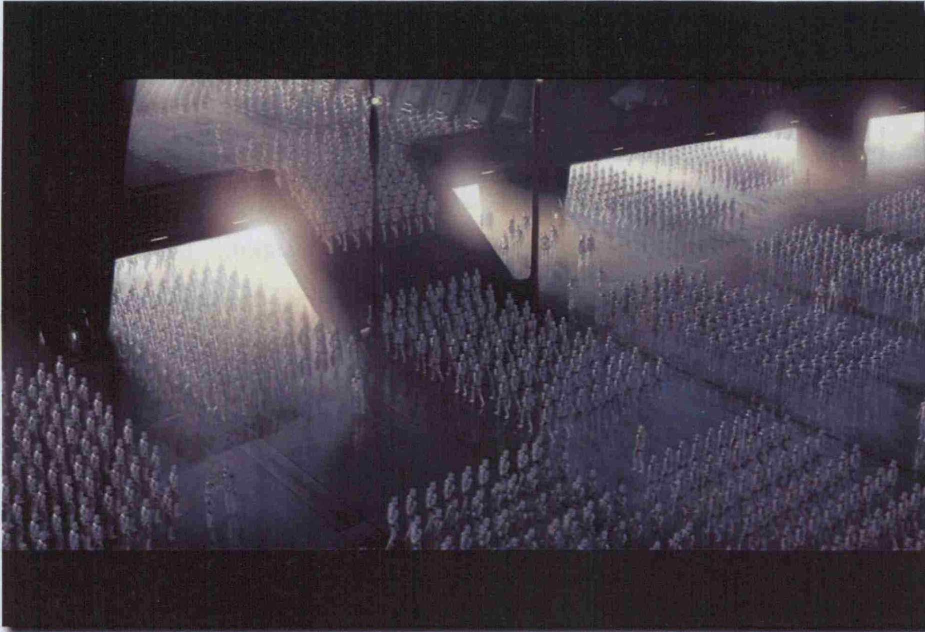

Figure 7.1 - A scene from the movie 'Star Wars Episode II: attack of the clones' (Lucasfilms – 2002).

The present chapter will conclude this thesis with a summary of research objectives and achievements. A review of research aims is presented in order to review what were the motivations behind the choice of unstructured impostors as the fundamental rendering primitive for crowds. A summary of the contributions will illustrate the novel features of this work for the field of Computer Graphics. A review of the implications will follow, discussing the impact the work has on present and future projects. Possible extensions of what has been achieved will be discussed and some final conclusions will be presented.

7.1 Summary

This thesis is focused on an image-based rendering approach to the task of real-time crowd rendering. The thesis has proposed the use of animated unstructured impostors as a method to achieve interactive frame rates for the visualisation of tens of thousands of moving virtual characters. Chapter 1 is an introduction to the task of crowd rendering that provides the motivation and explains the complexity of this task. Chapter 2 reviews existing literature on the visualisation of complex environments focusing, in particular, on those techniques that are used to visualize crowds. Polygonal-based, point-based and image-based rendering techniques are discussed, with an analysis of the weak/strong points of each approach. Chapter 3 proposes the use of impostors for the rendering of crowds. Also, techniques are presented to allow impostors to the reproduction of fine-scale illumination effects on basic OpenGL1.3 compliant hardware. The principles of the new methods are discussed, and arguments are proposed as to why this approach has the potential to outperform alternative approaches in many situations. Chapter 4 discusses the compatibility of unstructured impostors with polygonal-based scene graph architectures, and the possibility to mix different rendering approaches. Also, topics as global lighting, shadow casting and visibility computation are discussed. The research for this thesis required the realization of a rendering system and a fully functional set of tools and software libraries to populate complex environments with animated crowds, and these were described in Chapter 5. The results of substantial tests with this framework was reported in Chapter 6.

7.2 Review of aims

The research focused on a new solution to the problem of efficient real-time rendering of animated crowds, motivated by the need to populate complex Virtual Environments, since this is an essential part for the perceived realism of a computer-generated urban scenario. The goal that we pursued was to be able to render more than 10,000 animated avatars in real time with sufficient variety and visual detail. The analysis has shown that by compensating or attenuating the weak aspects of static, unstructured impostors, a new flexibility can be achieved that in some scenarios such as crowd rendering, makes them an effective and practical alternative to classic polygonal-only rendering.

7.3 Summary of contributions

The main contributions of this thesis were in the area of real-time rendering of complex virtual environments. An approach was presented that has made it possible to render large animated crowds at interactive frame-rates. To accomplish this, achieving at the same time a very high rendering speed and being able to render thousands of animated human-like characters in real-time, an innovative image-based rendering approach was proposed that made use of the hybrid approach provided by unstructured impostors. The results have shown that impostors are excellent for crowd rendering. The visual realism of the impostor representations is one of the clear advantages this method has over other approaches, such as using low detail polygonal models, since at certain viewing distances, it is almost impossible to determine whether the high-resolution model or the impostor is being rendered.

It should be noted that a qualitative analysis of the output produced by replacing far geometry with impostor is outside of the scope of this thesis. Building on the fundamental unstructured static impostor technique, an in-depth analysis of this matter, as well as the perceptual aspects of impostors rendering can be found in [Dobbyn06]. Dobbyn presents a system employing various geometrical LOD representations in order to visualise crowds at a real-time frame rate. The visual performance of the rendering system is in this case tuned by perceptually evaluating the effectiveness of impostors and low resolution meshes at replicating a highly detailed model's visual appearance. Dobbyn describes several experiments on visual perception: each experiment participant is simultaneously presented with two virtual humans using a high resolution mesh, a low-resolution mesh and an impostor at a particular distances, testing their ability to discriminate whether the two models are identical or not. The participant's responses for each case are recorded, and a psychometric function is applied to this data for each distance at which the virtual humans were displayed. The goal is to provide a guide to when a low-resolution mesh or an impostor could be used in place of the high resolution mesh without a user detecting the reduction of detail in the character's appearance.

In order to demonstrate the flexibility of impostors based rendering, we also proposed several techniques to augment the basic approach with advanced features such as per-pixel impostors lighting, global illumination and visibility computation for animated crowds. Most of the proposed algorithms only need the standard features of OpenGL1.1, insuring the applicability of the code on almost every available platform. Even per-pixel lighting and shadow mapping, both of which require the use of OpenGL1.3 or other dedicated extension, do not need

programmable hardware, and are easily replicable on any simple fixed-pipeline system.
A final important result is that the new rendering technique can coexist with traditional polygonal-based rendering of complex scenarios and is integrable with preexisting scene-graphs software.

7.4 Implications of this work

The results of the research reported in this thesis have been presented in a series of publications: the initial approach of unstructured animated impostors for the visualisation of large crowds was presented in [Tecch00b]; the use of multi pass rendering to allow a more precise control over the impostors colour was presented in [Tecch01b]; the topic of impostor shadows and shading was further developed in [Losc01]; the effectiveness of impostors packing strategy as a way to further preserve video-memory was presented in [Tecc02b]; per pixel lighting and related effects was presented in [Tecc02a][Tecch02c].
Moreover, to demonstrate the viability and flexibility of the approach, a large rendering system was developed. In the context of this system, several additional topics regarding crowd visualisation were also examined and addressed, generating additional innovative material on crowd-oriented collision detection (published in [Tecc00a]), a test platform for crowd behavior tests [Tecc01a], and crowd visibility computation [Tecc01b, Tecc02b]. Up to the present day, more than 50 publications have cited our work, including State Of The Art Reports [Jesch05, Ryder05]. In some cases, our work is the core upon which other researchers have developed their work [Coic05, Dobbyn05]. Finally, the EU project CREATE [Losc03] used the crowd rendering module developed during this thesis. Other projects that we were not involved in reimplemented the same basic technique.

7.5 Critical review

In this thesis, we explored the use of unstructured static impostors as a viable approach to the rendering of several thousand virtual humans walking around a virtual city at an interactive frame rate. Static impostors involves the pre-rendering of an impostor image of an object for a collection of reference viewpoints around. Sharing many concepts with the scalable technique of point-based rendering, impostor rendering is focused on the image side of the sampling process, and used to remove geometry complexity from where, in many situations, there is not

a great requirement of geometrical information. The main advantage of unstructured impostors is their minimal geometrical complexity, that is totally independent from the complexity of the objects that they replace, leading to an enormous improvement in rendering speed. The ratio between computation needs and preserved visual details of an impostor simply can't be matched by any other rendering primitive.
There remain, of course, some problems with the use of impostors: they require large amounts of dedicated memory, they do not allow many geometrical effects to be applied (but this is logical, since they contain only limited geometrical information), and they can introduce visual artifacts in the final generated image. Still, as the graphics capability of the rendering hardware continues to evolve, resource limitations lose importance, and approaches to mitigate or even to remove the remaining problems can potentially be developed. In many ways, we could state that the use of impostors for real-time rendering is future-proof.

7.6 Future directions

The basic technique of crowd rendering using impostors has a great potential for future developments. In particular, the following sections discusses three topics that the authors consider especially relevant.

7.6.1 Vertex and pixel shaders

An important initial design requirement for the research was that, in order for the new method to have a real impact on existing scene-graph architectures, the resulting algorithms needed to have a practical implementation on a vast range of hardware platforms. As a result, and as reported above, most of the techniques described here have an OpenGL1.1 compliant implementation, with only the most advanced effects requiring OpenGL1.3. Therefore, the method is nowadays usable on practically every single existing platform. Also, continuous advances in graphics card fill rate performance, a speed limiting factor in the case of impostors, are making this approach more and more attractive. Also, to the best of our knowledge, the set of algorithms here reported and their implementation represent the fastest possible approach, amongst all those presented, for real-time crowd rendering on a standard OpenGL pipeline architecture.
Relaxing this strict portability constraint would greatly increase the extensibility of the

method. In particular, the possibility to exploit the powerful per-pixel, per-fragment and per-vertex capabilities of the new generations of graphics hardware using high-level shading languages opens a whole new world of possibilities for impostor based rendering, as programmable graphics hardware gives a much finer control on vertices and pixels computation. Complex rendering algorithms that were before only possible on the CPU can nowdays be executed on the GPU, avoiding the bottleneck of CPU-GPU data transfers. Per-pixel displacement mapping, dynamic ambient occlusion, GPU-based computation of shadow volumes using the impostor's silhouette are just examples of interesting research directions, and improvements of our technique that use advanced hardware capabilities are emerging already.

7.6.2 Dynamical impostors generation

While the use of static impostors produce faster rendering than dynamic impostors, the latter have the advantage to require less video-memory. A practical implementation of dynamically generated impostors involves the capability to perform rendering tasks into off-screen pixel buffers (often called *pbuffer)*, a technique usually associated with computationally expensive context switch and frame buffer/texture memory operations. With the recent introduction from the OpenGL ARB community of the *EXT_framebuffer_object* extension into the official OpenGL specification [Segal04], the opportunity now arises for a much more efficient handling of the process. Beside its efficiency in terms of data transfers between frame memory and texture memory, the main advantage of the framebuffer object is that it only requires a single GL context, resulting in no switching between pbuffers, a much faster approach to the dynamic recreation of a texture.

Although this extension would improve the performance of generating an impostor image for a character's mesh at run-time, careful testing should be performed to asses the real potential of dynamic impostors, as the effectiveness of this technique relies heavily on re-using the current impostor image for a large number of frames for it be efficient, since the character's full geometry needs be rendered each time the image is updated.

7.6.3 Impostors clustering

Another extension of what we proposed would be to render several pre-generated impostors to the frame buffer object, and dynamically generate a single impostor containing these

characters, something similarly to what Wand et al. are doing for their Point Based Rendering technique [Wand02]. The advantage of this is that crowds in the background could be rendered as a single large dynamically generated impostor, thus reducing the number of draw calls per frame. While this could be done using a low-resolution mesh, it would still involve rendering several thousand polygons. In the case of applications that are populated with a large-scale regimented army, a single row of characters utilizing the pre-generated impostor could be dynamically generated as a single impostor image, and reused several times in the regiment. For example, a row of a thousand soldiers walking in step could be dynamically generated as a single impostor each time a new key-frame is needed, and subsequently reused a thousand times over the subsequent frames until the next key-frame, thus allowing the real-time visualisation of an army of a million soldiers. This would reduce the number of draw calls from a million to either two thousand (a thousand individuals and a thousand rows) when the current frame requires the image to be dynamically updated to reflect the soldiers' new key-frame, or a thousand (1,000 rows) when the current frame is in between key-frames.

7.7 Conclusions

This thesis has demonstrated that Image Based Rendering, and in particular the hybrid technique of unstructured animated impostors represents a very effective approach to the real-time rendering of large animated crowds. Beyond this, with the constant growth of available memory resources and the progressive improvement of frame-buffer access performance, that up until recently have been amongst the bigger obstacles to the use of impostors, it can be seen that there will be an increasing importance of these type of primitives, not only for the visualisation of crowds, but, more in general, for the rendering of a large class of visual phenomena. As the visual complexity of virtual environments increases, there is a growing need for advanced graphics primitives that can decouple the rendering of small surface details from their geometrical representation and placement. The availability on modern graphics hardware of feature such as texture compression, hi-speed data transfers, and per-pixel computing capabilities, demonstrates a new and flexible way forward, making it possible to address with IBR techniques a whole new series of application scenarios. Without requiring any dramatic change in existing scene-graph management, the hybrid technique of impostor rendering fits perfectly in an evolutionary, and not revolutionary, scheme of things. The author expects to see more and more of this powerful technique in use in the future.

REFERENCES

[Airey90] J. Airey, *"Increasing Update Rates in the Building Walkthrough System with AutomaticModel-Space Subdivision and Potentially Visible Set Calculations"*, PhD thesis, Department of Computer Science, University of North Carolina at Chapel Hill, TR#90-027, 1990

[Aróstegui05] M. Aróstegui, J.M. Miguel Fernandez, T. Gutierrez and J.I. Barbero. *"Virtual fire safety environment"*, Virtual Concept 2005, November 2005

[Aubel98] A. Aubel, R. Boulic and D. Thalmann. *"Animated impostors for real-time display of numerous virtual humans"*, Proceedings of the 1st International Conference on Virtual Worlds (VW-98), volume 1434 of LNAI, pages 14–28, Berlin, July1998

[Aubel99] A. Aubel, R. Boulic, and D. Thalmann, *"Lowering the cost of virtual human rendering with structured animated impostors"*, Proceedings of WSCG 99, Plzen, Czech Republic, 1999

[Aubel00] A. Aubel, R. Boulic, and D. Thalmann, *"Real-time display of virtual humans: Levels of details and impostors"*, IEEE Transactions on Circuits and Systems for Video Technology, 10(2):207–217, 2000

[AutMaya06] Autodesk *Maya* modelling package, http://usa.autodesk.com/adsk/servlet/index?siteID=123112

[Band98] S. Bandi and D. Thalmann, *"Space Discretization for Efficient Human Navigation"*, Proceedings of Eurographics 98, Computer Graphics Forum, Vol. 17(3), pp.195-206, 1998.

[Blinn76] J.F. Blinn and M.E. Newell, *"Texture and Reflection"*, Computer Generated Images, CACM, Volume 19, Number 10, pp. 542-547, 1976

[Blinn82] J. Blinn and F. James, *"Light Reflection Functions for Simulation of Clouds and DustySurfaces"*", Computer Graphics Forum, Volume 16, Number 3, pp. 21-29, July 1982

[Blinn88] James F. Blinn, *"Me and My (Fake) Shadow"*, IEEE ComputerGraphics and Applications, Volume 8, Number 1, pp. 82-86, January 1988

[Carrozzino05] M. Carrozzino, F. Tecchia, S. Bacinelli and M.Bergamasco *"Lowering the Development Time of Multimodal Interactive Application: The Real-life Experience of the XVR Project"*, Proceedings of ACE05, Valencia, Spain,2005

[Catmu74] E. Catmull, *"A Subdivision Algorithm forComputer Display of Curved Surfaces"*, PhDThesis, Dept. of Computer Science, Universityof Utah, Salt Lake City, Utah, U.S.A., 1974

[Catmull75] E.A. Catmull, "*Computer Display of Curved Surfaces*", Proceedings of the Conference on Computer Graphics, Pattern Recognition and Data Structure, pp. 11-17, May 1975

[Ciechomski05] P. de Heras Ciechomski, S. Schertenleib, J. Maim, and D. Thalmann. "*Reviving the roman odeon of aphrodisias: Dynamic animation and variety control of crowds in virtual heritage*", Proceedings of VSMM05, 2005

[Clans06] E-frontier's *Curios Labs Poser* modelling package http://www.e-frontier.com/go/poser6/whatsnew

[Chin89] N. Chin and S. Feiner, "*Near real-timeshadow generation using bsp trees*", Computer Graphics Forum (SIGGRAPH 89 Proceedings), pages 99–106, July 1989

[Chen93] S. E. Chen and L.Williams, "*View Interpolation for Image Synthesis*", Proc. SIGGRAPH '93. In Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, pp. 279-285, 1993

[Chen95] S. E.Chen, "*QuickTime® - An Image-Based Approach to Virtual Environment Navigation*", Proceedings of SIGGRAPH '95, ACM SIGGRAPH, pp. 29-37, 1995

[Chin89] N. Chin, and S. Feiner, *"Near real-time shadow generation using BSP trees"*, ACM Computer Graphics, 23(3):99–106, 1989

[Ciechomski04] PdH. Ciechomski, B. Ulicny and R. Cetre, "*A case study of a virtual audience in a reconstruction of an ancient roman odeon in aphrodisias*", Proceedings of VAST04 conference, 2004

[Ciechomski05] PdH. Ciechomski, S. Schertenleib, J. Maïm, D. Maupu and D. Thalmann, "*Real-time Shader Rendering for Crowds in Virtual Heritage*", Proceeding of the VAST 05 conference, 2005

[Clark76] J. H. Clark, "*Hierarchical geometric models for visible surface algorithms*", Communications of the ACM, 19(10):547–554, October 1976

[Cohe95] J. Cohen, M. Lin, D. Manocha, and M. Ponamgi. "*I-collide: An interactive and exact collision detection system for large-scale environments*", Proceedings of ACM Interactive 3D Graphics Conference, pp. 189-196, 1995.

[Coic05] J.M. Coic, C. Loscos, and A. Meyer, " *ThreeLOD for the Realistic and Real-Time Rendering ofCrowds with Dynamic Lighting*", Research Report RR-2005-008, Laboratoire d'InfoRmatique en Images et Sys-tèmes d'information, Université Claude Bernard, France, 2005.

[Cox99] M. Cox, L. Oestreicher, C. Quinn, M. Rauterberg, M. Stolze, "*HCI - Theory or practice in education*", Human-Computer Interaction--INTERACT-99 (Vol. 2, p. 134), Edinburgh Press, 1999

[Crow77] F. Crow, *"Shadow Algorithms for Computer Graphics"*, Computer Graphics (SIGGRAPH '77), Volume 11(2), pp.242-248,1977.

[Csuri79] C. Csuri, R. Hackathorn, R. Parent, W.Carlson and M. AND Howard, *"Towards an Interactive High Visual Complexity Animation System"*, 1979

[Dally96] W.J. Dally, L. McMillan, G. Bishop and H. Fuchs, *"The Delta Tree: An Object-Centered Approach to Image-Based Rendering"*, Artificial Intelligence Memo 1604, Massachusetts Institute of Technology, May 1996

[Day05] A.M. Day, and J.M. Willmott , "Compound Textures for Dynamic Impostor Rendering", Computers and Graphics, vol. 29, no. 1, pp. 109-124, 2005

[Darsa97] L. Darsa, B. Costa, and A. Varshney, *"Navigating StaticEnvironments Using Image-Space Simplification and Morphing"*, ACMSymposium on Interactive 3D Graphics, pp. 25-34, 1997

[Debevec02] P. Debevec, *"Image-Based Lighting"*, Computer Graphics and Applications, March/April 2002, Pages 26-34

[Decor99] X. Decoret, G. Schaufler, F. Sillion, and J. Dorsey, *"Multi-layered impostors for accelerated rendering"*, Computer Graphics Forum, 18(3):61–73, September 1999 (Proc. of Eurographics '99).

[Decor03] X. Decoret, F. Durand, F. Sillion, and J. Dorsey, *"Billboard clouds for extreme model simplification"*, ACM Transaction on Graphics, 22(3):689–696, 2003

[Drettakis04] G. Drettakis, M. Roussou, N. Tsingos, A. Reche and E. Gallo, *"Image-based Techniques for the Creation and Display of Photorealistic Interactive Virtual Environments"*, Proceedings of the Eurographics Symposium on Virtual Environments, June 2004

[Disc3dMax06] Autodesk *3DStudio Max* modelling package, http://usa.autodesk.com

[Dobbyn05] S. Dobbyn, J. Hamill, K. O'Conor and C. O'Sullivan, *"Geopostors: A Real-Time Geometry/Impostor Crowd Rendering System"*, ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games, pp. 95 – 102, 2005.

[Dobbyn06] S.Dobbyn, "Hibrid representations and perceptual metrics for scalable human simulation", P.h.D. Thesis, Trinity College, Dublin, 2006

[Eyles97] J. Eyles, S. Molnar, J. Poulton, T. Greer, A. Lastra, N. England, and L. Westover, "*PixelFlow: The Realization*", Proceedings of the ACMSIGGRAPH / EUROGRAPHICS workshop on Graphics Hardware, New York, The ACM Press, p. 57-68, 1997

[Forsyth01] T. Forsyth, "*Comparison of VIPM methods*", Game Programming Gems 2, M. DeLoura, Editor, Charles River Media, pp. 363–376, 2001

[Funkhouser92] T.A. Funkhouser, C.H. Séquin, S.J. Teller, "*Management of large amounts of data in interactive building walkthroughs*", Proceedings of the 1992 symposium on Interactive 3D graphics, p.11-20, Cambridge, Massachusetts, United States, June 1992

[Greene93] N. Greene, M. Kass, and G. Miller, "*Hierarchical z-buffer visibility*", Proceedings of SIGGRAPH 93, pages 231- 240, 1993

[Gortler96] S.J. Gortler, R. Grzeszczuk, R. Szeliski andM. F. Cohen, "*The Lumigraph*", Proceedings of SIGGRAPH 96, in Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, pp. 43-54, 1996

[Gott96] S. Gottschalk, M. Lin and D. Manocha, "*OBB-Tree: A Hierarchical Structure for Rapid Interference Detection*", SIGGRAPH 96 Conference Proceedings, Annual Conference Series, pp. 171-180, Addison Wesley, August 1996.

[Gosselin05] D. Gosselin, P. Sander, and J. Mitchell, "*Drawing a crowd*", ShaderX3, pages 505–517, 2005

[Gouraud71] H. Gouraud, "*Continuous shading of curved surfaces*", IEEE Transactions on Computers, 20(6): 623–628, 1971

[Heidmann91] T. Heidmann, "*Real Shadows, Real Time*", Iris Universe, 18:28-31, Silicon Graphics, Inc., 1991

[Hillier76] B. Hillier, A. Leaman, P. Stansall, and M. Bedford, (1976) "*Space syntax*", Environment and Planning B: Planning and Design,, Volume 3 (2), pp. 147-185, 1976

[Hoppe97] H. Hoppe, H. "*Progressive meshes*", In Proceedings of the 23rd Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '96, ACM Press, New York, NY, 99-108, 1996

[Hubb93] P. M. Hubbard, "*Interactive collision detection*", Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality, October 1993

[Iourcha99] K. Iourcha, K. Nayak, Z. Hong, "*System and Method for Fixed-Rate Block-based Image Compression with Inferred Pixels Values*", US Patent 5,956,431, 1999

[Lin98] M. Lin and S. Gottschalk, "*Collision Detection between Geometric Models: A Survey*", Proceedings of IMA Conference on Mathematics of Surfaces, 1998

[Jensen01] H. W. Jensen, S. R. Marschner, M. Levoy, and P. Hanrahan, "*A Practical Model for Subsurface Light Transport*", Computer Graphics Proceedings, Annual Conference Series, 2001, August 2001

[Jesch02] S. Jeschke and M. Wimmer, "*Textured depth meshes for real time rendering of arbitrary scenes*", Proceedings on Eurographics Workshop on Rendering, 2002

[Jesch05] S. Jeschke, M. Wimmer, and W. Purgathofer, "*Image-based representations for accelerated rendering of complex scenes*", In Eurographics 2005 STAR Report 1, 2005

[Kim02] T.Y. Kim, "Modeling, Rendering and Animating Human Hair", PhD thesis, University of Southern California, 2002

[Laveau94] S. Laveau, O.D. Faugeras, "*3-D Scene Representation as a Collection of Imagesand Fundamental Matrices*", INRIA Technical Report No. 2205, February 1994

[LengG90] J. Lengyel and M. Reichert, B.R. Donald and D. P. Greenberg, "*Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware*", Computer Graphics, Volume 24(4), pp. 327-335, August 1990

[Levoy85] M. Levoy and T. Whitted, "*The Use of Points as a Display Primitive*", Technical Report 85-022, Computer Science Department, University of North Carolina at Chapel Hill, January, 1985

[Lischinski92] D. Lischinski, F. Tampieri, and D. P. Green-berg, "*Discontinuity meshing for accurate radiosity*", IEEE Computer Graphics and Applications, Volume 12(6), 25-39, November 1992

[Lipman80] A. Lippman, "*Movie-Maps: An Application of the Optical Video disk to Computer Graphics*", Proceedings of SIGGRAPH 80, 1980

[Losc01] C. Loscos, F. Tecchia, Y. Chrysanthou, "*Real Time Shadows for Animated Crowds in Virtual Cities*" – ACM Symposium on Virtual Reality Software & Technology 2001, Banff, Alberta, Canada, November 2001

[Losc03] C. Loscos, H.R.Widenfeld, M.Roussou, A.Meyer, F.Tecchia, G.Drettakis, E.Gallo, A.R.Martinez, N.Tsingos, Y.Chrysanthou, L.Robert, M.Bergamasco, A.Dettori, S.Soubra, "*The CREATE project: mixed reality for design, education, and cultural heritage with a constructivist approach*", Proceedings of The Second IEEE

and ACM International Symposium on Mixed and Augmented Reality, 2003

[Luebke95] D. Luebke, C. Georges. "*Portals and mirrors: Simple, fast evaluation of potentially visible sets*", 1995 Symposium on Interactive 3D Graphics, pages 105–106, ACMSIGGRAPH, April 1995

[Maciel94] P.W.C. Maciel , "*Interactive Rendering of Complex 3D Models in Pipelined Graphics Architectures*", Technical Report TR403, Department of Computer Science, Indiana University, 1994

[Maciel95] P.W.C. Maciel and P. Shirley, "*Visual navigation of large environments using textured cluster*", Symposium on Interactive 3D Graphics, pages 95–102, April 1995, ACM SIGGRAPH

[Max95] N. Max, K. Ohsaki, "*Rendering Trees from Precomputed Z-Buffer Views*", 6th Eurographics Workshop on Rendering, pp. 45-54, June 1995

[Mark97] W.R. Mark, G. Bishop: "*Memory AccessPatterns of Occlusion-Compatible 3D ImageWarping*", Proceedings of Siggraph/Eurographics Workshop on Graphics Hardware, 1997, pp. 35-44, 1997

[MCCool00] MCCool, "*Shadow volume reconstruction from depth maps*", ACM Transactions on Graphics, Volume 19(1), pp. 1-26.

[McMillan95] L. McMillan, G. Bishop, "*Plenoptic Modeling: an image-based rendering system*", In Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, pp. 39-46, 1995

[Meyer01] A. Meyer, F.Neyret, and P. Poulin. "*Interactive rendering of trees with shading and shadows*", Proceedings of Eurographics Workshop on Rendering, July 2001

[Mysz95] K. Myszkowski, O.G. Okunev and T.L. Kunii, "*Fast collision detection between complex solids using rasterizing graphics hardware*", The Visual Computer, Volume 11(9), pp. 497-512, Springer-Verlag, 1995

[Molnar92] Molnar, S., J. Eyles and J. Poulton. "*PixelFlow: High-Speed Rendering Using Image Composition*" Computer Graphics: Proceedings of SIGGRAPH '92, Chicago, USA, pp. 231-240 July 1992

[Musse97] S. R. Musse and D. Thalmann. "*A model of human crowd behavior: Group interrelationship and collision detectionan alysis*", Eurographics '97 Workshop on ComputerAnimation and Simulation, pages 39–52, Budapest, Hungary, 1997

[Naylor90] B. Naylor, J. Amanatides and W. Thibault, "*Merging BSP Tress Yields Polyhedral Set Operations*", ACM Computer Graphics, Volume 24(4), pp. 115-124, August 1990

[Naylor92] B. F. Naylor ,"*Interactive solid geometry via partitioning trees*", Proceedings of Graphics Interface

'92, pp. 11–18, 1992

[Wand02] M. Wand and W. Straßer, "*Multi-resolution rendering of complex animated scenes*", Computer Graphics. Forum, 21(3), 2002

[Phong75] B. T. Phong, "*Illumination for Computer Generated Images*", Comm. ACM, Volume 18(6), pp. 311-317, June 1975

[Pulli97] K. Pulli, M. Cohen, T. Duchamp, H. Hoppe, L. Shapiro, W. Stuetzle, "*View-based Rendering: Visualizing Real Objects from Scanned Range and Color Data*", Rendering Techniques 97, Proceedings of the Eurographics Workshop on Rendering, pp. 23-34

[Reeves83] W. Reeves, "*Particle Systems - A Technique for Modeling a Class of Fuzzy Objects*", Computer Graphics, Volume 17(3), pp. 359-376, July 1983

[Ryder05] G. Ryder, and A. M. Day, "*Survey of Real-Time Rendering Techniques for Crowds*", Computer Graphics Forum, Volume 2(2), PP. 203 - , June 2005

[Ryder06] G. Ryder, and A. M. Day, "*High Quality Shadows for Real-Time Crowds*", Proceedings of Eurographics 2006 (short paper), Vienna, 2006

[Ross92] J. Rossignac, A. Megahed and B.0. Schneider, "*Interactive inspection of solids: Cross-sections and interferences*", Computer Graphics, Volume 26(2), pp. 353-360, July 1992

[RVImgmdl06] RealViz *Image Modeler* modelling package, http://imagemodeler.realviz.com/

[Samet90] Hanan Samet, "*The Design and Analysis of Spatial Data Structures*", Series in Computer Science. Addison-Wesley, Reading, Massachusetts,U.S.A., reprinted with corrections edition, April 1990

[Schaufler95] G. Schaufler, "*Dynamically generated impostors*", GI Workshop on Modeling, Virtual Worlds, Distributed Graphics, Bonn, Germany,1995

[Schaufler97] G. Schaufler, "*Nailboards: A rendering primitive for imagecaching in dynamic scenes*", Proceedings of Eurographics Rendering Workshop 1997, pp. 151–162, New York City, NY, 1997, Springer Wien

[Schaufler98] G. Schaufler, "*Per-Object Image Warping with Layered Impostors*", Proceedings of the 9th Eurographics Workshop on Rendering'98, pp.145-156, Vienna, Austria, 1998

[Segal01] M. Segal, and K. Akeley, K, "*The OpenGL graphics system: a specification (version 1.3)*", http://www.opengl.org. 2001

[Segal04] M. Segal, and K. Akeley, K, *"The OpenGL graphics system: a specification (version 2.0)"*, http://www.opengl.org. 2004

[Shade96] J. Shade, D. Lischinski, D. Salesin, T. DeRose, and J. Synder. *"Hierarchical image caching for accelerated walkthroughs of complex environments"*, SIGGRAPH 96 Conference Proceedings, pp. 75–82, August 1996, ACM SIGGRAPH

[Shoup73] R. Shoup, *"Some quantization effects in digitally-generated pictures"*, Proceedings of Society for Information Display, 1973

[Soler98] C. Soler, F. X. Sillion, *"Fast Calculation of Soft Shadow Textures Using Convolution"*, Proceedings of SIGGRAPH 98, pp. 321-332, 1998

[Stamminger02] M. Stamminger and G. Drettakis, *"Perspective shadow maps"*. ACM Transactions on Graphics (SIGGRAPH 2002), Volume 21(3), pp. 557–562, 2002

[Sudarsky96] O. Sudarsky and C. Gotsman, *"Output-Sensitive Visibility Al-gorithms for Dynamic Scenes with Applications to Virtual Reality"*, Computer Graphics Forum, Volume 15(3), Proceedings of Eurographics '96, pp. 249-258, 1996

[Suth63] I. Sutherland, *"SKETCHPAD: A Man Machine Graphical Communication System"*, Proceedings of the AFIPS Spring Joint Computer Conference , pages 329-346, April 1963

[Tecch00a] F. Tecchia and Y. Chrysanthou, *"Real time visualisation of densely populated urban environments: a simple and fast algorithm for collision detection. "*, EurographicsUK 2000, Swansee, UK, April 2000

[Tecch00b] F. Tecchia and Y. Chrysanthou, *"Real-Time Rendering of Densely Populated Urban Environments"*, Eurographics Workshop on Rendering 2000, Brno, Czeck Republic, July 2000

[Tecch01a] F. Tecchia, C. Loscos, R. Conroy and Y. Chrysanthou, *"Agent Behaviour Simulation (ABS): A Platform for Urban Behaviour Development"*, Proceedings of GTEC'2001, Hong Kong, January 2001

[Tecch01b] F. Tecchia, C. Loscos and Y.Chrysantou, *"Real-Time Rendering of Populated Urban Environment"*, Siggraph Sketches & Applications, Los Angeles, August 2001

[Tecch02a] F. Tecchia,C. Loscos and Y. Chrysantou, *"Real-Time rendering of virtual crowds"* ,Proceedings of Imagina 2002, Monte Carlo, February 2002

[Tecch02b] F. Tecchia, C. Loscos and Y. Chrysanthou, *"Image Based Crowd rendering"*, IEEE Computer

Graphics & Applications, Volume 22(2), March-April 2002

[Tecch02c] F. Tecchia, C. Loscos and Y. Chrysanthou, *"Visualizing Crowds in Real-Time"* Computer Graphics Forum, Volume 21(4), pp. 753-765, December 2002

[Teller91] SJ. Teller and C.H. Sequin. *"Visibility preprocessing for interactive walkthroughs"*, Computer Graphics (SIGGRAPH '91 Proceedings), Volume 25, pp. 61–69, July 1991

[Tesch05] M. Teschnerl, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser and P. Volino, *"Collision Detection for Deformable Objects "*, Computer Graphics Forum. Volume 24(1), 2005

[Torborg96] J. Torborg, and J.T. Kajiya, *"Talisman: Commodity Realtime 3D Graphics for the PC"*, Proceedings of SIGGRAPH '96, pp. 353-363, in Computer Graphics,Annual Conference Series, 1996

[Ulicny04] U. B. Ulicny, P. de Heras Ciechomski, and D. Thalmann. *"Crowdbrush, Interactive authoring of real-time crowd scenes"*, Proceedings of ACM SIGGRAPH Symposium on Computer Animation, 2004

[Wand02] M. Wand, W. Strasser, "Multi-Resolution Rendering of Complex Animated Scenes", In Computer Graphics Forum, Volume 21(3), pp. 483-491, 2002

[Watt98] A. Watt, F. Policarpo, *"The Computer Image"*, Addison-Wesley ACM SIGGRAPH Series

[Watt99] A. Watt, *"3D Computer Graphics (3rd Edition)"*, Addison Wesley, 3rd edition (December 6, 1999)

[Williams78] L. Williams, *"Casting Curved Shadows on Curved Surfaces"*, Computer Graphics (SIGGRAPH '78 Proceedings), Volume 12(3), pp. 270-274.

[Wonka99] P. Wonka and D. Schmalstieg, *"Occluder Shadows for Fast Walkthroughs of Urban Environments"*, Computer Graphics Forum, Volume 18(3), pp. 51-60, 1999

[Yemez99] Y. Yemez and E. Schmitt, *"Progressive Multilevel Meshes from Octree Particles"*, Proceedings of 3D Digital Imaging and Modeling, 1999

[Chrysanthou97] Y. Chrysanthou and M. Slater, *"Incremental Updates to Scenes Illuminated by Area Light Sources"*, Rendering Techniques 97, pp.103 - 114, 1997, Springer Computer Science.

[Zhang97] H. Zhang, D. Manocha, T. Hudson, and K. Hoff, *"Visibility culling using hierarchical occlusion maps"*, Computer Graphics Proceedings of SIGGRAPH 97, pages 77– 88, 1997

[Zhukov98] S. Zhukov , IA. Jones and G. Kronin, "*An ambient light illumination model*", Proceedings of the 9th EG Workshop on Rendering, pp. 45 – 56, 1998