LEABHARLANN CHOLÁISTE NA TRÍONÓIDE, BAILE ÁTHA CLIATH | TRINITY COLLEGE LIBRARY DUBLIN
Ollscoil Átha Cliath | The University of Dublin

Terms and Conditions of Use of Digitised Theses from Trinity College Library Dublin

Copyright statement

All material supplied by Trinity College Library is protected by copyright (under the Copyright and Related Rights Act, 2000 as amended) and other relevant Intellectual Property Rights. By accessing and using a Digitised Thesis from Trinity College Library you acknowledge that all Intellectual Property Rights in any Works supplied are the sole and exclusive property of the copyright and/or other IPR holder. Specific copyright holders may not be explicitly identified. Use of materials from other sources within a thesis should not be construed as a claim over them.

A non-exclusive, non-transferable licence is hereby granted to those using or reproducing, in whole or in part, the material for valid purposes, providing the copyright owners are acknowledged using the normal conventions. Where specific permission to use material is required, this is identified and such permission must be sought from the copyright holder or agency cited.

Liability statement

By using a Digitised Thesis, I accept that Trinity College Dublin bears no legal responsibility for the accuracy, legality or comprehensiveness of materials contained within the thesis, and that Trinity College Dublin accepts no liability for indirect, consequential, or incidental, damages or losses arising from use of the thesis for whatever reason. Information located in a thesis may be subject to specific use constraints, details of which may not be explicitly described. It is the responsibility of potential and actual users to be aware of such constraints and to abide by them. By making use of material from a digitised thesis, you accept these copyright and disclaimer provisions. Where it is brought to the attention of Trinity College Library that there may be a breach of copyright or other restraint, it is the policy to withdraw or take down access to a thesis while the issue is being resolved.

Access Agreement

By using a Digitised Thesis from Trinity College Library you are bound by the following Terms & Conditions. Please read them carefully.

I have read and I understand the following statement: All material supplied via a Digitised Thesis from Trinity College Library is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of a thesis is not permitted, except that material may be duplicated by you for your research use or for educational purposes in electronic or print form providing the copyright owners are acknowledged using the normal conventions. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone. This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

Hybrid Representations and Perceptual Metrics for Scalable Human Simulation

A Thesis

Submitted to the Office of Graduate Studies

of

University of Dublin, Trinity College

in Candidacy for the Degree of

Doctor of Philosophy

by Simon Dobbyn

April, 2006

TRINITY COLLEGE
12 MAY 2006
LIBRARY DUBLIN
THESIS
7867

Declaration

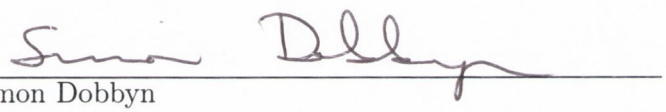
I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Simon Dobbyn

April 5, 2006

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.



Simon Dobbyn

April 5, 2006

Abstract

The simulation of large crowds of humans is important in many fields of computer graphics, including real-time applications such as games, as they can breathe life into otherwise static scenes and enhance believability. Such applications need to deal with having limited resources available at each frame. With many hundreds or thousands of potential virtual humans in a crowd, traditional techniques rapidly become overwhelmed and are not able to sustain an interactive frame-rate. Therefore, simpler approaches to the rendering, animation and behaviour control of the crowds are needed. Additionally, these new approaches must provide for variety, as environments inhabited by carbon-copy clones can be disconcerting and unrealistic.

The thesis describes the research and software development of a system that enables the rendering of large crowds of virtual humans at interactive frame rates. This system evolved from a previous system capable of simple virtual humans, which however was unable to cope with the strain of simulating large crowds, also only supported simple pre-scripted animations. We added more autonomous behaviour for the humans, including conversational, path-finding and object interaction. Furthermore, collaboration with AI researchers led to the integration of more intelligent behaviours. The crowd simulation method we developed provides for a hybrid combination of image-based (i.e., impostor) and detailed geometric rendering techniques for virtual humans. By switching between the two representations, based on a 1:1 *pixel to texel* ratio, our system allows visual quality and performance to be balanced. We improved on existing impostor rendering techniques and developed a programmable hardware based method for adjusting the lighting and colouring of the virtual humans skin and clothes. Additionally, the shading of each representation is performed such a way that the interchange of these representations is imperceptible to the viewer. Using extensive perceptual experiments, we have validated that the ratio at which we switch representations does indeed lead to imperceptible transitions from geometry to image, and vice versa. Our virtual crowds are embedded in an urban simulation system and interactive frame rates are maintained for crowds of over 70,000 individuals on commodity PCs.

Acknowledgments

Firstly, I would like to thank Carol OSullivan, my supervisor, for all of her help, advice and encouragement. This work would not have been possible without her.

Secondly, I would like to thank past and present members of the Image Synthesis Group for making the last four years a thoroughly enjoyable experience, and especially Chris and Keith for putting up with annoying questions.

Thanks to Brian - his collaboration was a great help in making the ALOHA system a success.

My parents, for their relentless encouragement and interest in everything I do.

Finally, I would like to thank Rachel. I would not have been able to get through the last year without her. Thank you so much for putting up with everything.

SIMON DOBBYN

University of Dublin, Trinity College
April, 2006

Contents

Abstract iv

Acknowledgments v

List of Tables ix

List of Figures xi

Chapter 1 Introduction 1

1.1 Scope . . . 2
1.2 Crowds in the Entertainment Industry . . . 3
1.3 Contributions . . . 4
1.4 Summary of Chapters . . . 6

Chapter 2 Background and Related Work 8

2.1 Character Visualisation . . . 9
2.1.1 Character Model . . . 9
2.1.2 Character Rendering . . . 10
2.1.3 Acceleration Techniques for Rendering Large-Scale Crowds . . . 14
2.2 Character Animation . . . 21
2.2.1 Layered Animation . . . 22
2.2.2 Animation of a Character's Skeleton . . . 22
2.2.3 Animation Level of Detail . . . 25
2.2.4 Simulation Level of Detail . . . 25
2.3 Behavioural Techniques for Characters . . . 26
2.3.1 Agent-Object Interaction . . . 26
2.3.2 Navigation Strategies . . . 27
2.3.3 Intelligent Virtual Agents . . . 29

2.3.4 Level of Detail Artificial Intelligence 30
2.4 Summary and Conclusion 31

Chapter 3 The ALOHA System **32**
3.1 The Original ALOHA System 33
3.1.1 Original Framework 33
3.1.2 Implementation of the Original ALOHA System 36
3.2 ALOHA System's Development Time-Line 36
3.3 ALOHA System's Models 39
3.3.1 Virtual Human Model 39
3.3.2 World Model 41
3.3.3 Animation and Rendering System 43
3.4 Virtual Human Innate Behaviour 45
3.4.1 Navigation Behaviour 45
3.4.2 Agent-Object Interaction Behaviour 47
3.4.3 Conversational Behaviour 51
3.5 LODAI for Autonomous Virtual Humans 55
3.5.1 The Proactive Persistent Agent Architecture 56
3.5.2 The PPA Believability Experiment 60
3.6 Summary and Conclusion 60

Chapter 4 Geopostors: Crowd Rendering System **62**
4.1 Levels of Detail Framework for Virtual Human Representation 63
4.1.1 High Geometric LOD: Deformable Mesh Model 64
4.1.2 Low Geometric LOD: Pre-generated Impostor Model 71
4.1.3 Switching Between Representations 80
4.2 Integration of Geopostors into an Urban Simulation Environment 83
4.2.1 Virtual Human Pedestrian Behaviour 83
4.2.2 Virtual Human LOD Shadows 89
4.2.3 Performance Optimisations 91
4.3 Summary and Conclusion 95

Chapter 5 Evaluation **96**
5.1 Computational Evaluation of Virtual Human Representations 97
5.1.1 Computational Evaluation Test Conditions 97
5.1.2 Set I Tests: Deformable Mesh Representation 98
5.1.3 Set II Tests: Pre-Generated Impostor Representation 103

5.1.4 Video Memory Consumption 109
5.1.5 Geopostor System's Bottlenecks 112
5.2 Performance of Crowd System in an Urban Simulation System 115
5.2.1 Minimizing Texture Thrashing Evaluation 120
5.3 Perceptual Evaluation of Virtual Human Representations 121
5.3.1 Psychophysics 121
5.3.2 Experiment 1: Impostor Vs. Mesh Detection Experiment 125
5.3.3 Experiment 2: Low Vs. High-Resolution Mesh Discrimination Experiment 130
5.3.4 Experiment 3: Impostor/Mesh Switching Discrimination Experiment . 134
5.4 Discussion of Evaluation Results 137

Chapter 6 Conclusions and Future Work **141**
6.1 Summary of Contributions 141
6.2 Future Work 143
6.2.1 Integration of the Geopostor System into the ALOHA System 143
6.2.2 Future Enhancements to the Mesh Representation 143
6.2.3 Future Enhancements to the Impostor Representation 144
6.2.4 Hybrid Representation 146
6.2.5 Perceptual Evaluation 146

Appendices **147**

Bibliography **153**

List of Tables

4.1 The different levels of detail for animation that can be used for a crowd's deformable mesh representation (Geometry$_{LOD}$ is HIGH). 71
4.2 The different levels of detail for variation that can be used for the crowd's deformable mesh representation (Geometry$_{LOD}$ is HIGH). 71
4.3 The different levels of detail for animation that can be used for a impostor representation (Geometry$_{LOD}$ is LOW). 79
4.4 The different levels of detail for variation that can be used crowd's impostor representation (Geometry$_{LOD}$ is LOW). 79

5.1 The vertex percentage, number of vertices and number of triangles for each mesh resolution tested. 99
5.2 The vertex percentage, number of vertices and number of triangles for each head mesh resolution used in the hybrid representation's tests. 108
5.3 The vertex percentage, number of vertices and number of triangles for each mesh resolution used in the hybrid representation's tests for comparison. . . . 108
5.4 The amount of video memory consumed by the impostor's detail map and normal map for a single frame of animation. 110
5.5 The amount of video memory consumed by the mesh's diffuse texture. 111
5.6 The amount of video memory consumed by a mesh's vertex buffer object storing a single pose. 111
5.7 The total amount of video memory consumed by a single human model's impostor and mesh representation. 112
5.8 Frame-rate of the geometric representations used in the Geopostor system measured using NVPerfKit. 113
5.9 Population of the city for each LOD representation so that the frame rate does not drop below a threshold of 35fps. 117

5.10 One-to-one pixel to texel ratio distances for the virtual human with a 45° Field of View at 800x600 Resolution 129
5.11 Low resolution mesh details 132
5.12 Mean PSE and JND for low resolution geometry 134
5.13 PSE comparisons for distance 134
5.14 The distance at which $\text{LOD}_{Geometry}$ models are perceptually equivalent and their associated rendering cost (where $\text{Variation}_{LOD} = 3$) using the results from Section 5.1. 140

1 ALOHA specific file formats. 152

List of Figures

1.1 Virtual crowds created with the Geopostor system in the Virtual Dublin system. 1
1.2 Computer generated crowds by Massive Software, as used in the motion picture *Lord of the Rings: Return of the King.* 3
1.3 Crowds in the games industry. 5

2.1 Simple texturing-mapping 10
2.2 Per-pixel lighting effects such as environment mapping and normal mapping . 11
2.3 Real-time hair and skin rendering techniques 12
2.4 Five discreet mesh models containing (a) 2,170 (b) 1,258 (c) 937 (d) 612 and (e) 298 triangles. 15
2.5 Rendering crowds using a discrete LOD approach 16
2.6 Geometric-based representations rendered with various per-pixel shading effects. 17
2.7 Image-based crowds 19
2.8 Point-based crowds 21
2.9 ViCrowd system. 30

3.1 High level view of the proposed framework for the original ALOHA system. . 34
3.2 High level view of the implemented original ALOHA system. 37
3.3 Virtual human's 2-layer model consisting of a skeleton and a hierarchy of rigid meshes (without texture-mapping). 40
3.4 A virtual human's skeleton animated with various animations and the subsequent mesh deformation. 42
3.5 Path-finding in the ALOHA system. 46
3.6 (a) A smart object model for a bar object showing its user slots and usage steps for both slots. (b) A *barman* and *customer* agent interacting with each other using the smart object. 50
3.7 Characters using the conversational behaviour in the ALOHA system. 54

3.8 A sample PPAFCM for a basic agent. 58
3.9 University campus simulation using the PPA architecture: (a) Students going to lectures. (b) Academic presenting the lecture. (c) Academic and students going to the pub (d) People socialising in the pub. 59

4.1 Virtual human models in the ALOHA system: (a) Hierarchy of rigid meshes without texture-mapping. (b) Single deformable textured-mapped mesh. . . . 64
4.2 (a) The original high resolution mesh (100%) and 4 generated low resolution meshes at vertex percentages of (b) 60%, (c) 45%, (d) 30%, and (e) 15%. . . 65
4.3 (a) High LOD mesh representation in 3D Studio MAX. (b) The grouping of triangles based on material used (shown by the different colours). (c) The grouping of triangles based on the body part it represents (shown by the different colours). 66
4.4 Precalculating and storing the deformation of a mesh performing a walk animation for 10 key-frames. 67
4.5 (a)Example of an artist in progress of generating an outfit for a model using the 'Outfit Editor' application. (b) Nine outfits for three template meshes. . . 68
4.6 Mesh shading and colouring sequence. 69
4.7 Generating a higher level of variety by layering a head animation on top of a default walk animation. 70
4.8 A MaxScript plug-in removes unused space from each viewpoint image and combines 17*8 viewpoint images into a single 1024x1024 image for a particular frame. 74
4.9 Impostor shading and colouring sequence. 77
4.10 Using programmable texture addressing to add variety to the impostor representation. 78
4.11 (a) Problem of an impostor's appearance being flat and pixellated when it is near to the viewer. (b) Problem solved by switching to a mesh representation at an appropriate distance. 81
4.12 Calculating impostor image pixel size. 82
4.13 Virtual Dublin screen shots. 84
4.14 (a) Walk map. (b) Potential field map. (c) Height map. 87
4.15 (a) Projected impostor shadow. (b) Projected mesh shadow. (c) Crowd and city without shadows. (d) Crowd and city with projected LOD shadows. . . . 90
4.16 Normal map split into smaller elevation images. 94

5.1 (a) Rendering cost and (b) number of humans displayed at 30 frames per second for a mesh at a resolution of 100%, 60%, 45%, 30% and 15%. 101
5.2 (a) The cost of using 1, 5 and 10 different template meshes. (b) The number of humans that can be rendered at 30 frames per second using 1, 5 or 10 different meshes. 102
5.3 (a) The cost of adding variation using the OpenGL fixed function pipeline and programmable graphics hardware. (b) The number of humans that can be coloured at 30 frames per second using either programmable graphics hardware or OpenGL's fixed function pipeline. 103
5.4 Rendering cost of using the impostor representation. 105
5.5 Number of humans displayed at 30 frames per second using the impostor representation. 106
5.6 Number of humans displayed at 30 frames per second using the impostor and mesh representation. 107
5.7 (a) The cost of using 1, 5 and 10 different template impostors. (b) The number of humans that can be rendered at 30 frames per second using 1, 5 or 10 different impostors. 107
5.8 (a) The cost of adding head behaviour. (b) The number of humans with head behaviour displayed at 30 frames per second. 109
5.9 (a) GPU timing performance of the Geopostor system. (b) GPU Geometry Performance of the Geopostor system. 114
5.10 (a) Walkthrough of Virtual Dublin containing no crowds. (b) Distribution of humans within Virtual Dublin populated with 10,000, 25,000, 50,0000, 75,0000 and 100,000 humans 115
5.11 Walkthrough of Virtual Dublin containing crowds using different mesh resolutions. 118
5.12 Walkthrough of Virtual Dublin containing crowds using the impostor model. 118
5.13 Walkthrough of Virtual Dublin using the Geopostor system 118
5.14 (a) Number of impostors drawn each frame. (b) Number of meshes drawn each frame. 119
5.15 Geopostor system with collision avoidance and projected ground-plane shadows. 119
5.16 Frame rates using 1024x1024 textures Vs. using elevation textures 120
5.17 An Ogive function fitted to a participant's data for a yes-no design. 122
5.18 Example of the stepping procedure for an up-down staircase terminated after four reversals. 124
5.19 Virtual human impostor Vs. mesh detection experiment. 127

5.20 Results of the impostor Vs. mesh detection experiments 130
5.21 Mean PSE and JND values for low resolution models. 133
5.22 Results of the popping detection experiments (showing the PSE and the JND in terms of pixel to texel ratio) for humans facing viewer (1) and spinning (2). 138
5.23 Distances at which different LOD representations are perceptually equivalent to the highest resolution mesh. 140

1 The PPAFCM used by the basic agent in the simulation of a university campus. 147
2 University campus simulation using the PPA architecture: (a) Students going to lectures. (b) Academic presenting the lecture. (c) Academic and students going to the pub (d) People socialising in the pub. 148
3 Basic agent with the following roles layered on top of it: (a) LECTURER Role. (b) STUDENT Role. (c) BAR CUSTOMER Role. (d) BAR KEEP Role. . . 150

Chapter 1

Introduction

Figure 1.1: Virtual crowds created with the Geopostor system in the Virtual Dublin system.

The simulation of large crowds of humans is important in many fields of computer graphics, including real-time applications such as games, as they can breathe life into otherwise static scenes and enhance believability. Although many new games are released each year, it is very unusual to find large-scale crowds populating the environments depicted. Such applications need to deal with having limited resources available at each frame. With many hundreds or thousands of potential virtual humans in a crowd, traditional techniques rapidly

become overwhelmed and are not able to sustain an interactive frame-rate. Therefore, simpler approaches to the rendering, animation and behaviour control of the crowds are needed. Additionally, these new approaches must provide for variety, as environments inhabited by carbon-copy clones can be disconcerting and unrealistic.

In this thesis, we describe the development of a software system that enables the rendering of large crowds of virtual humans at interactive frame rates. We also describe the further development of the ALOHA (Adaptive Level Of detail for Human Animation) system, upon which the human animation and rendering is based. ALOHA also provides for sophisticated behaviours to control the autonomous virtual humans in our simulations. Our contribution was in the implementation of the animation sub-system required to animate and visualize these behaviours. The crowd simulation method we developed provides for a hybrid combination of image-based (i.e., impostor) and detailed geometric rendering techniques for virtual humans. Impostor techniques have to date not been used extensively in games when realism is required, due to their poor appearance when close to the viewer. The unique advantage of our system is the combination of computationally efficient crowd rendering, on a scale not seen in real-time games before, coupled with the high realism afforded by our method due to the ability to imperceptibly switch to a high-detail human model for closer inspection. By switching between the two representations, based on a *pixel to texel* ratio, our system allows visual quality and performance to be balanced. We improved on existing impostor rendering techniques and developed a programmable hardware based method (targeting consumer grade GPUs from NVIDIA and ATI) for adjusting the lighting and colouring of the virtual humans' skin and clothes. Our virtual crowds are embedded in an urban simulation system (as shown in Figure 1.1) and interactive frame rates are maintained for crowds of over 70,000 individuals on a desk-top PC.

1.1 Scope

The main focus of this thesis is the representation, rendering, animation control, and perceptual evaluation of virtual humans for real-time graphical applications. While the ALOHA system can provide sophisticated behavioural attributes for the humans, this work was performed in collaboration with other researchers, and our main contribution was in the rendering and animation scheduling in order to realize these behaviours. Therefore, apart from providing an overview, we do not focus on the development of Artificial Intelligence algorithms or high-level animation techniques in this thesis. In terms of animation creation, all of the motions used by the virtual humans have to be either generated in a commercial modelling package, or acquired as motion capture clips, which are then exported and scheduled within

our system.

1.2 Crowds in the Entertainment Industry

When it comes to the entertainment industry, large-scale crowds consisting of virtual characters have become commonplace in films over the recent years. From a galaxy far, far away to lands ruled by one ring, virtual crowds have considerably added to the impact of movies (e.g., *AntZ* [PBG98], *Titanic* [LB98] and *The Lord of the Rings Trilogy* [MAS]). Without virtual crowds, memorable scenes such as the battle scenes in Lord Of the Ring: The Two Towers and in Star Wars: Attack of the Clones would have been impossible to simulate with real actors. The most notable software in this industry is MASSIVE (Multiple Agent Simulation System In Virtual Environment) which was used to generate crowds made up of about 50,000 individual characters in the *Battle of Helm's Deep* in *The Two Towers*. Massive characters function as complex beings subject to physical forces, with specific body attributes that range from biological to the behavioural. Up to 350 potential motions can be assigned to each character, and the character's brain determines how these actions are played out. The brain for each MASSIVE character is a tangled web of numerous behavioural logic nodes, which provide the rules on how to make decisions and act. The final result represents a milestone in computer-generated filmmaking (see Figure 1.2). While crowd systems such as MASSIVE provide a level of realism that would be impossible to achieve at interactive rates, it has has set the standard for crowd simulation and rendering in movie special effects.

Figure 1.2: Computer generated crowds by Massive Software, as used in the motion picture *Lord of the Rings: Return of the King.*

Due to the hardware limitation of current PCs and consoles, only a small number of

realistic on-screen characters are typically found in the majority of games. Renderware Crowd FX provides a significantly simpler approach to crowd rendering that facilitates only the simplest of animation (designed for sports game audience rendering). While these fixed view crowds are realistic from the distance, its application is limited as a result of the crowd's cyclical behaviour and the fact that they become unbelievable once viewed from other angles.

Action games such as Grand Theft Auto: San Andreas are crying out for large realistic virtual crowds. The game itself is set in a large fictional city where you roam about performing missions for different crime bosses. Even though the city is beautifully modelled, the game itself has a post-apocalyptic atmosphere as a result of a sparse number of non-player characters inhabiting the city's streets irrespective of time of day or weather. Detailed models are used for the characters and can only be displayed in small numbers due to processing and display limitations. Typically fewer than 50 humans are observed at any one time (see Figure 1.3 (b)). In addition to this lack of NPCs, if you follow any NPC you will notice that their behaviour is extremely basic and they do not give the impression that they are getting on with the running of their every day life. Without properly simulated crowds, the total experience of this game is reduced. In Rockstar Game's State of Emergency, they achieve larger real-time animated crowds. However the game hides the characters' blocky models and unrealistic behaviours by distracting the user's attention with the frantic style of game-play (see Figure 1.3 (c)).

The game most notable for simulating large-scale crowds is *Rome: Total War* (shown in Figure 1.3 (d)). This real-time strategy game involves controlling large armies of several thousand soldiers in famous historic battles. Each soldier uses low-detailed models and can be animated with one of a thousand different moves. While the detail and behaviour of the soldiers is quite simple when viewed at a close distance, the sheer size of these real-time battle scenes is impressive.

1.3 Contributions

We have developed a system capable of rendering large realistic crowds with the visual realism of a high-resolution geometry rendering system, but at a fraction of the rendering cost. Our system allows visual quality and performance to be balanced. Our results and user studies to date have demonstrated that our impostors are an excellent substitute for geometry in real-time crowd simulations, not only because of proven efficiency gains, but also in terms of visual fidelity. At a reasonable distance, it is virtually impossible to determine whether the high-resolution model or the impostor is being rendered in our simulations. We have also developed algorithms to **drive the behaviour of the individual characters** in our

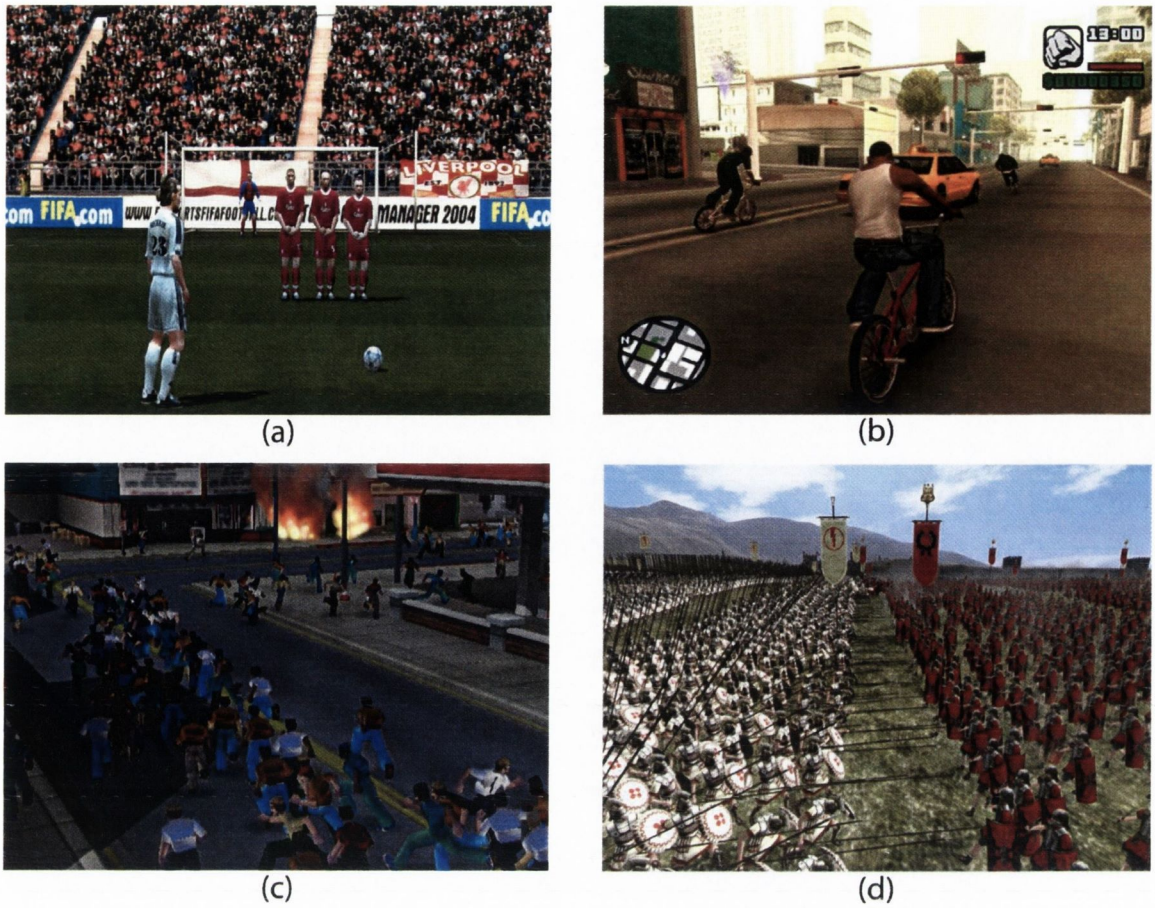


Figure 1.3: Crowds in the games industry: (a) Fifa 2005 (© Electronic Arts), (b) Grand Theft Auto: San Andreas (© Rockstar Games), (c) State of Emergency (© Rockstar Games) and (d) Rome Total War (© The Creative Assembly Limited).

crowds, thus allowing further realism, as the autonomous agents go about their business in a convincing manner.

Our main contribution is the ability to **imperceptibly switch** between high resolution geometric representations, which are computationally expensive but look good up close, to a low cost impostor representation, which is indistinguishable from the geometric model at a certain distance (or pixel to texel ratio). Furthermore, we have **improved upon previous pre-generated impostor rendering techniques** with respect to two factors: performance and realism. Firstly, the advances in graphics hardware allowed us to dynamically light and add colour variation to the impostors in a **single rendering pass** (previous methods needed at least 5 passes). By clever usage of 1-D "outfit" textures, **variation of the crowd** was substantially increased with negligible memory usage, and further realism was possible

by using realistic impostor shadows that blend with the world and with each other. The **perceptual metrics** used in our system have been **rigorously validated with extensive user trials.**

As a result of our research, games developers and other practitioners will now be able to add real-time crowds, at a scale and level of realism heretofore not possible, into their simulations. The software is self-contained and easy to integrate into existing applications, as evidenced by our use of the system to add crowds of pedestrians to our Virtual Dublin system, shown in Figure 1.1. In particular, with next generation game consoles targeting high definition TVs with resolutions significantly higher than those present in current generation games, the appearance of even distant non-player characters in the game will be very important. Our method will allow game developers to focus on the development of high resolution characters for the game without the need to create many different representations of the character at multiple distances. This could represent a significant saving in terms of art resources required to ensure high fidelity character rendering in the game. Although the most obvious application domain for our research is the gaming industry, it would also be very useful in other areas, such as security and emergency simulations.

1.4 Summary of Chapters

This rest of the thesis has been divided up into the following chapters:

- **Chapter 2** provides a detailed overview of the previous background and related work for the scalable simulation of virtual humans, including various techniques from the fields of computer graphics, computer animation, and artificial intelligence.

- **Chapter 3** introduces the original ALOHA system and the enhancements made during its development. These include the integration of basic behaviours and the implementation of the animation sub-system required to reflect these behaviours. An overview of collaborative work that involved the integration of artificial intelligence techniques is also presented.

- **Chapter 4** describes our Geopostor system, detailing the representations and techniques that are needed to simulate large-scale crowds. The successful integration of the system into an urban environment system is described, along with the optimization of its performance.

- **Chapter 5** presents the evaluation of the Geopostor system with regards to the computation cost of the models used and its performance within an urban environment

system. Experiments that perceptually evaluate the system's human representations are presented, and how the subsequent results are used to enhance the system's visual realism is discussed.

- **Chapter 6** provides a summary of our contributions, and a discussion of future work.

Chapter 2

Background and Related Work

Computer generated crowds have become increasingly popular in films. However, their presence in the real-time domain, such as computer games, is still quite rare. Even though there has been extensive research conducted on human modelling and rendering, the majority of it is concerned with realistic approximations using complex and expensive geometric representations. When dealing with the visualisation of large-scale crowds, these approaches are too computationally expensive, and different approaches are needed in order to achieve an interactive frame rate.

This chapter describes the main research related to the real-time visualisation, animation, and behaviour of virtual crowds in the following manner:

- We first introduce general **character visualisation** techniques using the fixed function graphics pipeline, and show how recent improvements in graphics hardware has greatly improving the realism of characters in computer games. Furthermore, we describe **acceleration techniques** for the rendering of large crowds which can be subdivided into three categories: **visibility culling** methods, **geometrical *level of detail* (LOD)** and sample-based rendering techniques such as using **image-based** and **point-based** representations.

- Next, we describe **character animation techniques**, including how a character's model is animated using the **layered approach**, and the various techniques for generating character animations such as **kinematics**, **physically-based animation** and **procedural animation**. We also describe how **animation and simulation level of detail** provides a computationally efficient solution for the simulation of crowds.

- Finally, we detail **behavioural techniques** used to endow virtual characters with artificial intelligence (AI), including **agent-object interaction techniques** and **nav-**

igation strategies to simulate characters interacting with objects and moving within their environment. We also provide an overview of the research on **intelligent virtual agents** and present how the technique of **level of detail AI** has been employed in computer games.

2.1 Character Visualisation

2.1.1 Character Model

The most common model used for representing characters in 3-D computer graphics is the mesh model. A mesh is defined as a collection of polygons, where each polygon's surface is made up of three or more connected vertices, and is typically used to represent an object's surface such as a character's skin. Since 3-D graphics hardware is optimised to handle triangles, meshes are typically made up of this type of polygon in 3-D applications. A simple model, consisting of a low number of triangles (i.e., several hundred), can be used to model a character's general shape. However, as the need for realism increases, more detailed models are necessary and require a high number of triangles (i.e., several thousand) to model the character's hands, eyes and other body-parts. This extra detail comes at a greater rendering cost and a balance between realism and interactivity is necessary, especially when rendering large crowds of characters. While current graphics cards can render over eight million unlit triangles per second (e.g. ATI's and NVIDIA's current cards), a static scene such as an urban environment populated with multiple characters could require rendering several hundred thousand triangles. Therefore, depending on the scene complexity, the number of triangles in the character's mesh or any other scene object is limited in order to maintain a real-time frame rate.

Real-time lighting of these meshes is necessary to provide depth cues and thus enhance the scene's realism. Otherwise, the triangles are rendered with a single colour creating a flat unrealistic look. Typically, the lighting of the character's mesh in games is implemented with basic Gouraud shading [Gou71]. Gouraud shading is a method for linearly interpolating a colour across a polygon's surface and is used to achieve smooth lighting, giving a mesh a more realistic appearance. As a result of its smooth visual quality and its modest computational demands, since lighting calculations are performed per-vertex and not per-pixel, it is by far the predominant shading method used in 3-D graphics hardware. aAdditionally, texture-mapping [Cat74], which allows the attaching of a two-dimensional image onto the polygon's surface, can greatly improve the realism of a humans's 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 (as shown in Figure 2.1). The image is loaded into

memory as a rectangular array of data where each piece of data is called a *texel* and each of the polygon's vertices are assigned texture coordinates to specify which texels are mapped to the surface.

Figure 2.1: Simple Texturing-Mapping: (a) Mesh without texture-mapping, (b) Texture Map (c) Texture-mapped mesh.

2.1.2 Character Rendering

Until a few years ago, the only option for hardware-accelerated graphics was to use the fixed function pipeline. This is where texture addressing, texture blending and final fragment colouring are fixed to perform in set ways. The introduction of the *mulitexture* extension [Ope04], allowed lighting effects involving several different types of texture maps to be performed in a single rendering pass. This extension provides the capability to specify multiple sets of texture coordinates that address multiple textures, which means that the previous and slower method of multi-pass rendering can be avoided. More recently, hardware vendors have exposed general programmable pipeline functionality, allowing for more versatile ways of performing these operations through programmable customisation of vertex and fragment operations [Ope04]. With the introduction of multi-texturing and programmable graphics hardware, coupled with the improvements in hardware capability such as the increase in triangle fill-rates, texture memory size and memory bandwidth, we are seeing an exciting era of realistic character rendering and animation techniques which were previously unfeasible to employ at interactive rates.

There has been extensive research on enhancing the realism of a character's mesh by applying various per-pixel lighting effects (see Figure 2.2). Environment mapping [BN76] can be used to simulate an object reflecting its environment. For characters such as soldiers wearing shiny armour, environment mapping can greatly improve their realism. Per-pixel bump mapping [Kil00] can be used to perturb the surface's normal vector in the lighting equation

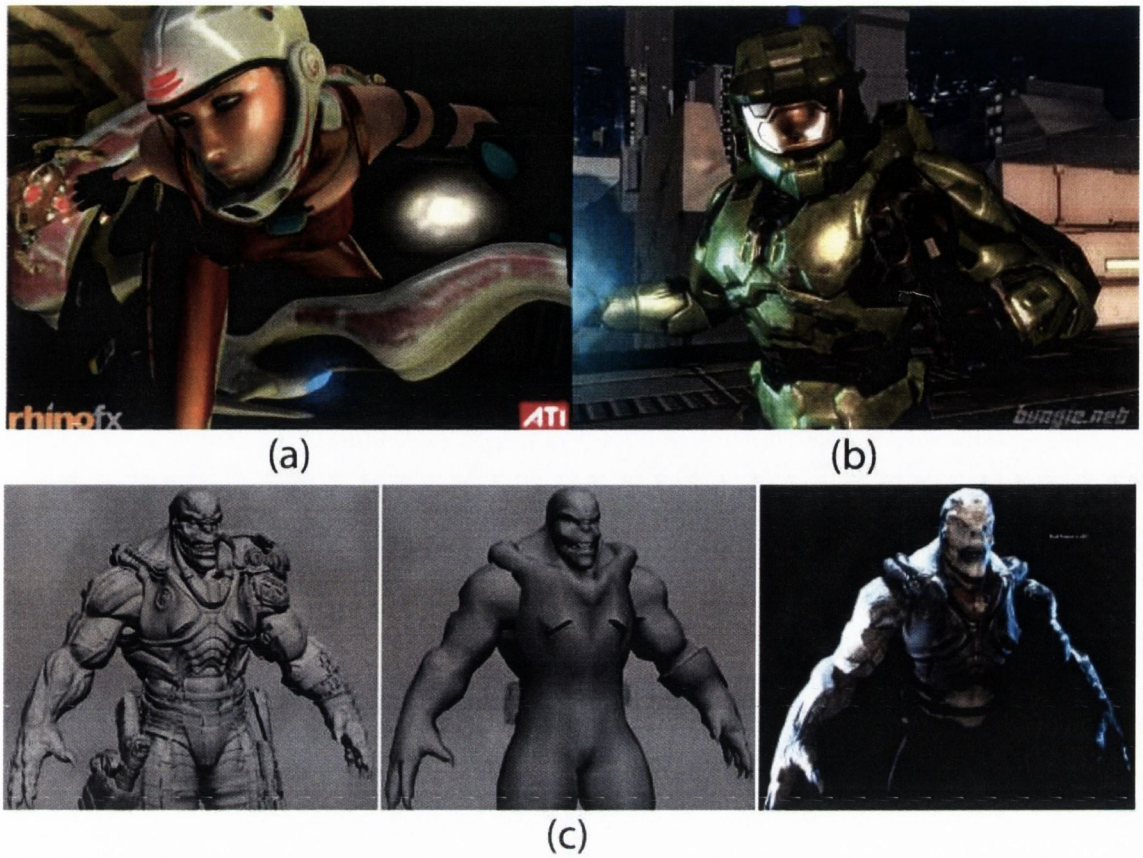


Figure 2.2: Per-pixel lighting effects such as environment mapping in (a) Ruby Demo ((© ATI Technologies) and (b) Halo 2 (© 2004 Microsoft Corporation), and (c) Normal mapping in Unreal Engine 2003 (© 2005 Epic Games Inc).

to simulate wrinkles or bumps. This is used to increase the visual detail of the character's clothing and appearance without increasing geometry. More recently, this approach has been extended by using a normal map image, generated from a highly detailed character's mesh, in conjunction with a low detailed mesh to improve its visual detail [COM98, Map]. Displacement mapping is another method which adds surface detail to a model by using a height map to translate vertices along their normals [Don05]. In order to speed up the lighting calculations for a static object, the lighting can be pre-computed and stored for each polygon in a texture called a light map [SKvW+92] and this method was made famous by iD Software's "Quake" games. In addition to the speed increase, this method allows complex and more realistic illumination models to be used in generating the map. With dynamic objects, the light map needs to be calculated on a per-frame basis, as otherwise shading artefacts will manifest. Sander et al. [SGM04] recalculate the light map using graphics hardware for each

frame in order to correctly shade the character's skin as it moves within its environment. However, generating real-time light maps for a large number of characters is unfeasible at interactive frame-rates.

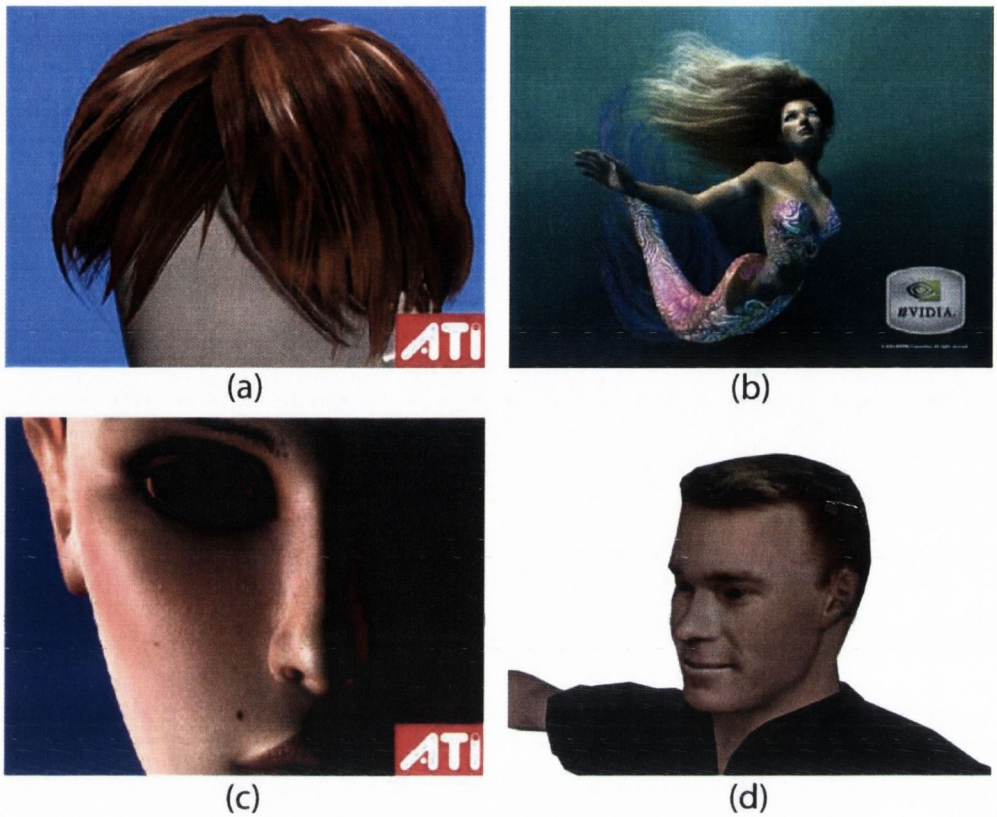


Figure 2.3: Real-time hair rendering based on the light scattering of human hair fibres [MJC+03] and a fur rendering model [KK89] using a (a) polygonal model [Sch04] (b) a particle system [Wlo04]. (c) Real-time skin rendering based on subsurface scattering [SGM04]. (d) Hair and skin rendered with simple texturing.

More recently, more realistic character effects borrowed from the film industry have been implemented in real-time. Based on the technique used to light the face of digital characters in the film *The Matrix Reloaded*, Sander et al. [SGM04] produced realistic looking skin in real-time. Scheuermann et al. [Sch04] improved the rendering of real-time hair using a polygonal model, where the hair shading is based on the work on light scattering of human hair fibers by Marschner et al. [MJC+03] and on Kayiya et al.'s fur rendering model [KK89]. While this technique has greatly improved the realism of real-time hair, in addition to using low geometric complexity, it assumes little or no hair animation and is not suitable for all hair styles. Wloka [Wlo04] uses a similiar rendering approach for underwater hair which

is animated by treating it as a particle system. Unfortunately, these techniques can only be used for a limited number of characters, since they are computationally intensive, and therefore simple texture-mapped triangles are typically used for an individual's skin and hair detail within large crowds (Figure 2.3).

Shadow Techniques

Shadows enhances the realism of a scene's lighting and provide important depth cues. The three main techniques to shadowing are planar shadows, shadow mapping and shadow volumes.

Planar shadows involve projecting an object's geometry onto a shadow plane and rendering the shadow plane on top of the ground plane's geometry. While this technique is very fast, it is limited to scenes containing flat floors, can suffer from z-buffer fighting, and does not take self-shadowing into account. However, the z-fighting problem has been solved by implementing the algorithm in hardware using the stencil buffer [Kil99].

Shadow mapping [Wil78] is an image-based shadowing technique involving two rendering passes. The first pass involves rendering the scene's depth information into an off-screen buffer from the viewpoint of the light source. The resulting discretized image is known as a shadow map and stores the distance of each object in the scene to the light source. The second pass consists of rendering the scene from the camera's viewpoint, whereby the distance of each pixel is projected into light space and is compared with depth value stored in the shadow map. If the distance is less than or equal to the corresponding value stored in the shadow map then the surface is lit by the light source. However, since this technique is image-based it can suffer from aliasing problems depending on the resolution of the shadow map and techniques have been developed to solve this [SCH03, SD02].

Shadow volumes is a geometry-based algorithm first described by Crow [Cro77] which avoids the aliasing problem associated with shadow mapping. The algorithm involves constructing a shadow volume for an occluder object to identify points that are shadowed by it. The shadow volume is constructed by extruding the model's polygon edges away from the light source to form a semi-infinite frustum. Shadow volumes can be implemented using graphics hardware via the stencil buffer in four rendering passes [Hei91]. However, since this technique is geometry-based, it can cause the application to be fill rate and polygon limited and produces incorrect shadowing with alpha blended polygons.

2.1.3 Acceleration Techniques for Rendering Large-Scale Crowds

The requirement in interactive systems for real-time frame rates means that only a limited number of polygons can be displayed by the graphics engine in each frame of a simulation. Visibility culling techniques provide the first step to avoid rendering off-screen characters, and therefore reducing the number of triangles displayed per frame. However, other rendering techniques are needed since a large portion of the crowd could potentially be on-screen.

Visibility Culling

Culling provides a mechanism to reduce the number of triangles rendered per frame by not drawing what the viewer cannot see. The basic idea behind culling is to discard as many triangles as possible that are not visible in the final rendered image. The two main types are visibility and occlusion culling.

Visibility culling discards any triangles that are not within the camera's view-frustum. In the case of a large scenes containing several thousand characters, it would be computationally expensive to view-frustum cull each character's triangles. However, it can be used to avoid rendering potentially off-screen characters by testing their *bounding-volumes* with respect to the view-frustum. For further details on various optimized view-frustum culling techniques utilizing bounding-volumes see [AM00].

The aim of occlusion culling is to quickly discard any objects that are hidden by other parts of the scene. Various research has been conducted on effective ways of establishing occluding objects utilizing software methods or 3-D graphics hardware. For a detailed survey of these techniques see [COCSD03]. For crowds populating a virtual city environment, occlusion culling is a method that can greatly improve the frame rate, since a large portion of the crowd will be occluded by buildings, especially when the viewpoint is at ground level.

Geometric-Based Rendering and Level of Detail

Level of detail (LOD) is an area of research that has grown out of the long-standing trade-off between complexity and performance. LOD stems from the work done by James Clark where the basic principles are defined [Cla76]. The fundamental idea behind LOD, is that when a scene is being simulated, it uses an approximate simulation model for small, distant, or important objects in the scene. The main area of LOD research has focussed on geometric LOD, which attempts to reduce the number of rendered polygons by using several representations of decreasing complexity of an object. For each frame, the appropriate model or resolution is selected, usually based on the object's distance to the camera. In addition to distance, other LOD selection factors that can be used are screen space size, priority, hysteresis, and

perceptual factors. Since the work done by Clark [Cla76], the literature on geometric LOD has become quite extensive. Geometric LOD has been used since the early days of flight simulators, and has more recently been incorporated in walkthrough systems for complex environments by Funkhouser et al. [FST92, FS93], and Maciel et al. [Mac93].

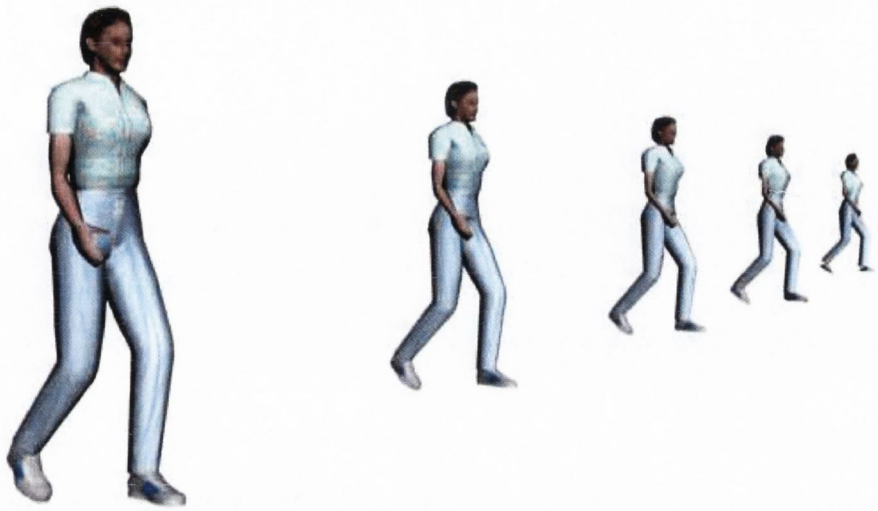


Figure 2.4: Five discreet mesh models containing (a) 2,170 (b) 1,258 (c) 937 (d) 612 and (e) 298 triangles.

One approach for managing the geometric LOD of virtual humans is using a discrete LOD framework. A discrete LOD framework involves creating multiple versions of an object's mesh, each at a different LOD, during an offline process (see Figure 2.4). Typically, a highly detailed (also known as a high resolution) mesh, is simplified by hand or using automatic tools to create multiple low resolution meshes varying in detail. At run-time, depending on the LOD selection criteria, the appropriate resolution mesh is chosen in order to maintain an interactive frame rate.

Another good solution for altering the geometric detail of a character in games is through the use of subdivision surfaces [Lee02]. In the beginning, one of the main problems with geometric LOD was the generation of the different levels of detail for each object, which was a time-consuming process as it was all done by hand. Since then, several LOD algorithms have been published in order to automatically generate the different levels of detail for an object [EDD+95, Hop96]. Subdivision surfaces is one method, based on a continuous LOD framework, where a desired level of detail is extracted at run-time by performing a series of edge collapsing/vertex splitting on the model. Starting with a low-resolution mesh, a subdivision scheme can be used to produce a more detailed version of the surface by using masks to define a set of vertices and corresponding weights, which are in turn used to create

new vertices or modify existing ones. By applying these masks to the mesh's vertices, a new mesh can be generated. An advantage of using masks is that different type of masks can be used in order to deal with boundary vertices and crease generation. In [OCV+02], O'Sullivan et al. describe a framework that uses subdivision surfaces as a means to increase or decrease the appearance of a human's mesh within groups and crowds depending on their importance to the viewer.

In order to solve the problem of rendering large numbers of humans, De Heras Ciechomski et al. [dHCUCT04] avoid computing the deformation of a character's mesh by storing pre-computed deformed meshes for each key-frame of animation, and then carefully sorting these meshes to take cache coherency into account. Ulicny et al. [UdHCT04] improve on their performance by using 4 LOD meshes consisting of 1038, 662, 151 and 76 triangles and disabling lighting for the lowest LOD, thereby achieving a frame rate several times higher. To introduce crowd variety, they use several template meshes for the humans, and clone and modify these meshes at run-time by applying different textures, colors, and scaling factors to create the illusion of variety. They succeed in simulating several hundred humans populating an ancient Roman theatre and a virtual city at interactive frame-rates.

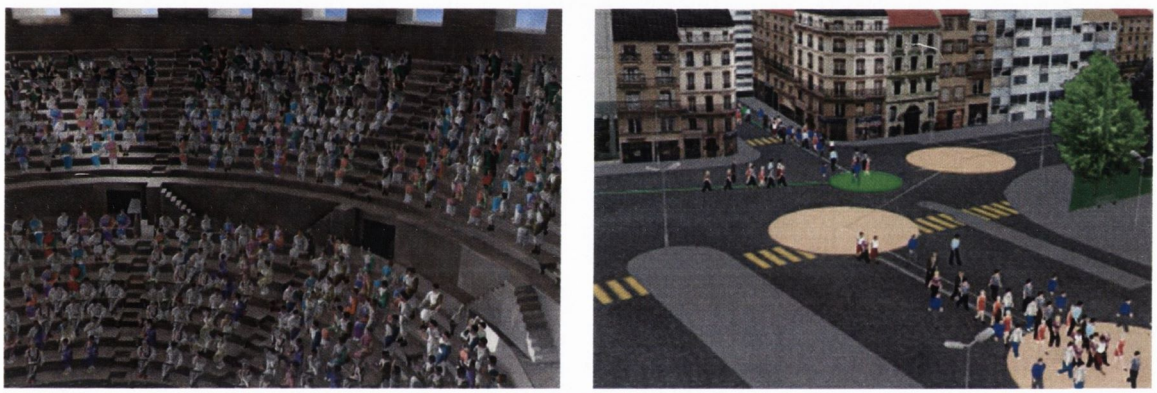


Figure 2.5: Rendering crowds using a discrete LOD approach [dHCUCT04].

Gosselin et al. [GSM05] present an efficient technique for rendering large crowds while taking variety into account. Their approach involves reducing the number of API calls need 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 such scenes containing a large number of deformable meshes, they minimize the number of vertex shader operations that need to be performed.

In their simulation, they use one directional light to simulate the sun, and three local diffuse lights. The shading of each character's mesh is performed by per-pixel shading and a normal map generated from a high resolution model is used. Specular lighting is calculated for the sun and is attenuated using a gloss map to allow for parts of the character to have differing shininess. Realism is further increased by using an ambient occlusion map generated from the high resolution model. This map approximates the amount of light that could reach the model from the external lighting environment and provides a realistic soft look to the character's illumination. Finally, using a ground occlusion texture which represents the amount of light a character should receive from the sun based on their position in the world, the illusion that the terrain is shading the characters as they move within the environment is created. So that the characters are not a carbon copy of each other, they use a colour lookup texture, which specifies 16 pairs of colours that can be used to modulate the character, with a mask texture to specify which portions should be modulated. In addition to this, decal textures to add other various details to the character's model, such as badges, are applied (see Figure 2.6).

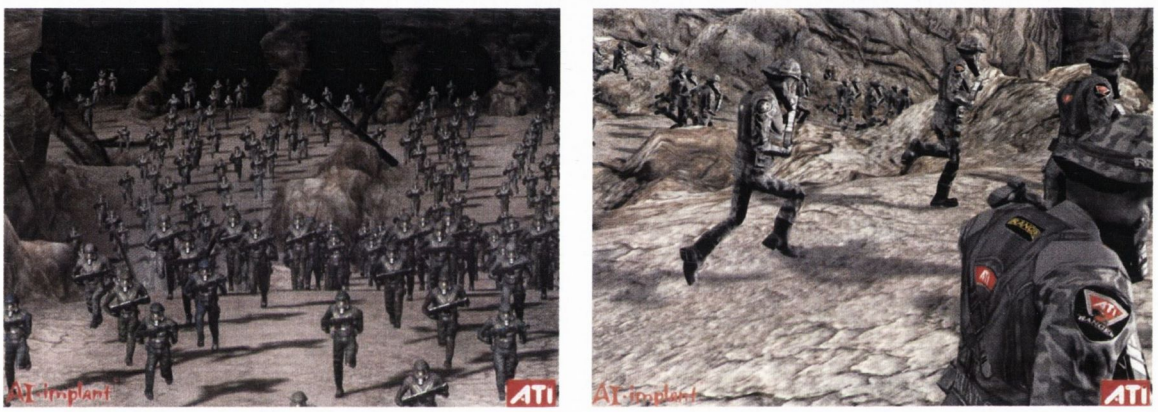


Figure 2.6: Geometric-based representations rendered with various per-pixel shading effects [GSM05] (© 2005 ATI Technologies 2004).

Image-Based Crowd Rendering

Torborg et al. [TK96] propose a new hardware architecture called *Talisman* in an attempt to solve the problems of bandwidth limitations and low frame rates which hindered real-time 3D graphics applications at that time. The architecture exploits both spatial and temporal coherence in real-time applications via the use of image layers. Each layer (termed *sprite*) is an image of an individual object's geometry and is reused by applying an affine transformation to simulate the motion of the object. Lengyel et al. [LS97] warp sprites using the Talisman architecture by a best fit affine transformation based on a set of sample points in

the underlying 3D model. Due to improvements in hardware such as increases in CPU and RAM speeds and the introduction of the AGP bus, the development of the Talisman project became obsolete and was not continued.

Image-based rendering (IBR), stems from the research by Maciel et al. [MS95] on 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. Schaufler [Sch97] introduces the idea of *nailboards* which extends the idea of planar impostor to store per-pixel depth information in the impostor's texture. Visibility artefacts caused by interpenetrating impostors are solved using these depth values in a two-pass rendering algorithm. Consequently, due to planar impostors providing a good visual approximation to complex objects at a fraction of the rendering cost, a large amount of research has introduced different types of impostors such as *layered impostors*, *billboard clouds*, and *texture depth images* for rendering acceleration of various applications.

Layered impostors use a number of pre-generated depth-augmented images at different distances from the viewer [DSSD99, Sch98b, Sch98a]. Image warping is used on these layers to allow the object to be viewed from a larger set of viewpoints. In comparison to planar impostors, layered impostors consume a larger amount of texture memory since it is composed of several images. However, layered impostors can be viewed from a larger range of viewpoints, whereas planar impostors can be viewed from viewpoint used to generate the image. Billboard clouds [DDSD03] represent a model by projecting and rendering the model's polygons to suitable alpha-textured planes. The selection of these planes is optimized so that they cover the largest possible area of the model's geometry. However, the generation of these alpha-textured planes involve long pre-processing times. Textured depth meshes [DSV97, SDB97, JW02] simplify an object by triangulating a rendered image of the object based on the image's depth values. Textured depth meshes are expensive to generate and can also suffer from disocclusion artefacts resulting in image gaps. A survey of these different types of impostors, including their application and their advantages and disadvantages, can be found in [JWP05].

To represent a virtual human, Tecchia et al. [TC00a] and Aubel et al. [ABT00] both use planar impostors. However, they differ in how the impostor image is generated. The two main approaches to the generation of the impostor images are: dynamic generation and static generation (also referred to as pre-generated impostors).

Aubel et al. use a dynamically generated impostor approach to render a crowd of 200 humans performing a 'Mexican wave' [ABT00]. With dynamically generated impostors, the impostor image is updated at run-time by rendering the object's mesh model to an off-screen buffer and storing this data in the image. This image is displayed on a quadrilateral, which is

Figure 2.7: Image-based crowds: (a) Dynamically generated image-based crowds [ABT00] (b) Pre-generated image-based crowds [TC00a].

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. 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 et al.'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 significantly changed.

The planar nature of the impostor can cause visibility problems as a result of it interpenetrating other objects in the environment. To solve this problem, Aubel et al. propose using a multi-plane impostor which involves splitting the virtual human's mesh into separate body parts, where each body part has its own impostor representation. However, this approach can cause problems similar to those mentioned in Section 2.2, resulting in gaps appearing. 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.

Tecchia et al. [TC00a] use pre-generated impostors for rendering several thousand virtual humans walking around a virtual city at an interactive frame rate. Pre-generated impostors involve the pre-rendering of an impostor image of an object for a collection of viewpoints (called reference viewpoints) around the object. Unfortunately, since virtual humans are an-

imated objects, they present a trickier problem in comparison to static objects. As well as rendering the virtual human from multiple viewpoints, multiple key-frames of animation for each viewpoint need to be rendered, which greatly increases the amount of texture memory used. In order to reduce the amount of texture memory consumed, Tecchia et al. reduce the number of reference viewpoints needed for each frame by using a symmetrical mesh representation animated with a symmetrical walk animation, so that already generated reference viewpoints can be mirrored to generate new viewpoints. At run-time, depending on the viewpoint with respect to the human, the most appropriate reference viewpoint is selected and displayed on a quadrilateral, which is dynamically orientated towards the viewer. To allow for the dynamic lighting of the impostor representation, Techia et al. [TLC02b] pre-generate normal map images for each viewpoint by encoding the surface normals of the human's mesh as a RGB colour value. By using a per-pixel dot product between the light vector and a normal map image, they compute the final value of a pixel through multi-pass rendering and require a minimum of five rendering passes.

The main advantage of this approach is that it is possible to deal with the geometric complexity of an object in a pre-processing step. However, with pre-generated impostors, since the object's representation is fixed, 'popping' artefacts are introduced as a result of being forced to approximate the representation for the current viewpoint with the reference viewpoint. To avoid these artefacts, the number of viewpoints around the object for the pre-generation of the impostor images can be increased. However this can later cause problems with the consumption of texture memory. Image warping is another technique of reducing the popping effect, but this method can also introduce its own artefacts. Since a pre-generated approach requires a large number of reference viewpoints for several frames of animation, this makes it unsuitable for scenes containing a variety of human models that each needs to perform a range of different motions.

Point Sample Rendering

Another sampled-based approach for the visualisation of virtual humans is point sample rendering, which involves replacing a mesh with a cloud of points, approximately pixel-sized [LW85]. Wand et al. [WS02] use a pre-computed hierarchy of triangles and sample points to represent a scene. This involves converting key-frame animations of meshes into a hierarchy of point samples and triangles at different resolutions. 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.

For smaller crowds, consisting of several thousands of objects, each object is represented

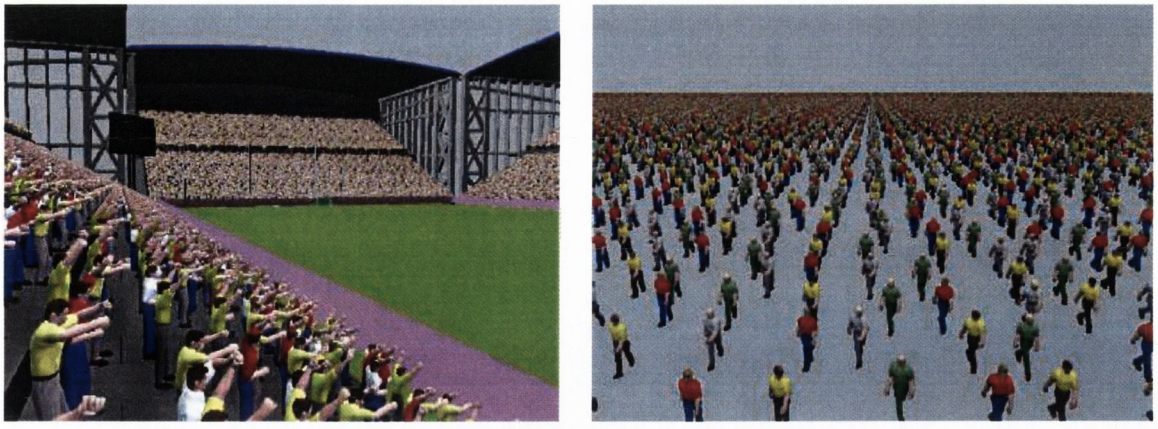


Figure 2.8: Point-based crowds: (a) Stadium populated with animated 16,000 fans and (b) Crowd of 90,000 humans walking on the spot.

by a separate point sample and its behaviour 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). While this allows them to render arbitrarily complex scenes, such as 90,000 humans walking on the spot and a football stadium inhabited by 16,000 fans (see Figure 2.8), less flexibility is provided for the motion of the objects, since the hierarchies are pre-computed and therefore cannot be used in simulating a large crowd moving within its environment.

2.2 Character Animation

The problem with using a mesh to represent a dynamic object, such as human character, is that a way of animating the mesh is needed to reflect the motion of the character. In older generation games, the character consisted of a hierarchy of meshes, where each mesh represented a particular body part and was animated in some way (e.g., Lara Croft in Tomb Raider). However, the main problem with this approach is that holes can appear where two or more meshes meet. These gaps can be hidden either by clever modelling using clothing or armour, at the cost of requiring extra polygonal detail, or by constraining the movement of the bones. However, depending on the type of character being modelled, this is not always possible. Nowadays, a character's mesh is typically animated by using a layered animation approach.

2.2.1 Layered Animation

The layered animation approach works by layering a character's mesh on top of a skeleton structure and deforming the mesh based on the animation of the underlying skeletal layer. The skeleton consists of a hierarchy of joints interconnected by bones, where each joint defines where a bone begins and is used as its pivot point. Except for the bone at the root of the hierarchy (know as the *root bone*), each bone is linked to a parent bone and has either one, multiple, or no child bones. To easily transform a bone from one coordinate space to another, each bone's position and rotation is stored in a transformation matrix. The global transformation matrix of each bone is dependent on the matrices of all of its parents, and can be calculated as a function of both its local and parent's global transformation matrices.

In order to deform the mesh, the mesh and the skeleton first need to be setup in a reference pose, typically using DaVinci's Vitruvian man pose, to facilitate their respective alignment. Each vertex in the mesh is assigned either one or more influencing bones with a corresponding weight to specify the amount of influence each bone has on it. Linear blend skinning (LBS) is used for deforming the mesh [Lan98, Lan99], where the deformation of each vertex's position (V') and normal (N') is calculated as a function of the vertex's original position relative to each deforming bone (V_i), its normal (N), each deforming bone's global transformation matrix (TM_i) and its influencing weight (w_i) (Equation 2.1). When calculating the deformation of the normals, only the rotational component is used by getting the inverse transpose of the global transformation matrix ($(TM_i{}^{-1})^T$).

$$\begin{aligned} V' &= \sum w_i \times TM_i \times V_i \\ N' &= \sum w_i \times (TM_i^{-1})^T \times N \end{aligned} \tag{2.1}$$

Linear blend skinning can be implemented through programmable graphics hardware by using a vertex program and this greatly improves its performance [Dom, GSM05]. This technique is fast to compute and therefore has become widespread in recent games. While problems can arise for large bone rotations, causing the mesh to collapse to a single point, this can be solved by adding extra bones [Web00], or using spherical blend skinning [KZ05].

2.2.2 Animation of a Character's Skeleton

Traditionally, an articulated structure, such as a skeleton, is animated using computer animation data stored as key-frames. A key-frame allows the transformation of a bone (i.e., its position and rotation) to be specified as a function of time. This allows complicated an-

imations to be simply stored as a set of key-frames for each bone. While the most simple method of generating key-frame animations for articulated structures is through kinematics, extensive research on providing other ways of generating animation data has been carried out, focusing on physical simulation and procedural animation.

Kinematics

A common method for animating an articulated structure in real-time is with kinematics, which is based on properties of motion such as position and velocity over time. A character's key-frame animation is typically generated from data that has been created manually through kinematics by an animation artist using a key-frame editor.

Forward kinematics specifies joint rotations as a function of time and is useful in pre-generating character animations in modeling/animation packages, such as 3D Studio Max. Once the animation has been created, it can be subsequently exported as key-frame data to be used within an application. Motion capture systems allow the movements of a real actor to be captured or stored as animation data by using different types of capture hardware and this was the predominant method for animating characters in *The Lord of the Rings Trilogy* [Sco03]. While the quality and realism of manually created animations depends on the skill of the artist, motion captured animations are extremely realistic as a result of using a real human actor. With regards to animating crowds, the main limitation of forward kinematics is that a large database of pre-generated or pre-captured motions is necessary in order to achieve some type of variation amongst the crowd. Otherwise, a crowd consisting of individuals performing the same animation can significantly reduce realism.

Inverse kinematics can resolve the skeleton's joint angles and the corresponding key-frame data so that an end-effector (e.g., the hand bone) is animated towards a target position. The main advantage of this is that it can be used for the real-time generation of various character animations (e.g., pointing in a particular direction, looking at an object and opening a door). Several algorithms exist to resolve the joint angles with varying computational accuracy of the results, the majority of which can be used with groups of characters in real-time. The main limitation of this technique is that, even though it generates a correct solution, it might not be a high-fidelity human motion.

Physically-Based Animation

Physically-based animation provides a good approach to generating unique and context-sensitive motion and in theory can produce an unlimited number of motion types. However, the problem with using the approach is that it is can involve computationally intensive algo-

rithms and the generated animation is somewhat dependent on various character properties. Therefore, this type of animation is not easily reusable and thus not well-suited for the real-time animation of a large number of characters of various shapes and sizes. Dynamic simulations use Newtonian force-based methods to generate animations utilizing forces that occur in articulated structures (e.g., velocities, mass, collision), in addition to kinematic properties. Physically-based animations have been used for animating virtual athletes in realistic sport simulations [HWBO95], generating physically correct swimming motion for fish [TT94], and characters walking on an uneven terrain [SM01].

Procedural Animation

Procedural algorithms reuse animation data from a library of motions to generate new animations. The two main approaches are combining, and altering animation data. Combining animations involves reusing animations with various techniques such as fading functions, overlapping and blending techniques. Various research has been conducted on providing smooth transitions between motions, such as the simple use of fade-in and fade-out functions [PG96, RCB98] and the more complex weighting and summing techniques [SBMTT99]. Perlin et al. [PG96] reuse and overlap animations by considering human motions as a "combination of temporarily overlapping gestures and stances". In general, combining animation data provides a good and fast approach for animating characters in real-time applications. However, to allow for some variation, it is important that there is a large library of pre-generated motions that can produce plausible combinations. Motion graphs can be compiled, which are directed graphs that describe how motion may be recombined, to automatically generate transitions to connect motions. The motion graph is generated from the library by identifying similar frames between each pair of motions and using these to form the nodes of the graph. These nodes provide plausible transitions between motions and allow the character to perform more complicated performances [KGP02].

The second approach to procedural animation involves altering the style of animation data based on various techniques such as noise functions [PG96], and emotional transforms based on character-based properties [ABC96]. Even though more realistic and less repetitious animations are produced by altering the data, these techniques can be computationally intensive and should only be considered for the real-time animation of a limited number of characters.

2.2.3 Animation Level of Detail

LOD research has recently extended from the area of geometry into areas such as motion and simulation, thus providing a computationally efficient solution for the simulation of crowds. In [GMPO00], Giang et al. propose a LOD framework for animating and rendering virtual humans in real-time. In order to achieve a scalable system, they use a LOD resolver that controls the switching between levels of detail and specifies parameters for controlling the geometric and motion controller. Through these parameters, the LOD resolver has the ability to request different animation levels of detail. The different levels of detail used relate to how the motion is simulated (e.g., pre-defined forward kinematics, inverse kinematics, or dynamics), and its update frequency. This results in smooth realistic animations being applied to virtual humans rated with high importance, while lower level animation techniques are applied to virtual humans in the background, taking minimal perceptual degradation into account.

In [dHCUCT04], the deformation of a character's mesh was pre-computed and stored to avoid these computations at run-time. However, these characters were limited to the number of animations they could perform due to the size limit of memory. To improve on their previous system, in [dHCSMT05] they propose rendering crowds animated using the layered animation approach (see Section 2.1.3) to reduce the consumption of memory and accelerate the animation of the skeleton and the subsequent mesh deformation using a level of detail caching scheme for animations and geometry. They update a character's animation at a specific frequency dependent on its level of detail instead of on a per-frame basis. For example, characters are updated at a minimum of 4Hz at the lowest LOD and at a maximum of 50Hz at the highest LOD, where the LOD selection criteria is based on the character's distance from the camera. The animation of the skeleton and the subsequent mesh deformation are done in software so that they can be reused in a caching scheme.

2.2.4 Simulation Level of Detail

In [CH97], Carlson and Hodgins use less accurate animation models for selected one-legged creatures in order to reduce the computational cost of simulating groups of these creatures. Three simulation LODs are used for the motion of these creatures: rigid-body dynamics, point mass simulation with kinematic joints and point mass simulation with no kinematic motion of the leg. Their selection of an individual's simulation LOD is based on a individual's importance to the viewer or action in the virtual world.

Ulicny et al. [UT02b] discuss the challenges of real-time crowd simulations, focussing on the need to efficiently manage variety, and propose the idea of *levels of variety*. They define

a system's variety based on the following levels: level of variety zero (*LV0*) if a task uses a single solution, level of variety one (*LV1*) if it has a choice from a finite number of solutions, and level of variety two (*LV2*) if it is able to use an infinite number of possible solutions. For example, a crowd composed of a single human model would be *LV0*, several pre-defined model types would be *LV1*, and finally an infinite number of automatically generated model types would be *LV2*. Using this concept, they define a modular behavioural architecture based on rules and finite state machines, to provide simple yet sufficiently variable behaviours for individuals in a crowd.

2.3 Behavioural Techniques for Characters

Endowing characters with behaviours to reflect their motivations and internal states, provides a way of controlling their low-level motion, resulting in them being autonomous. This section will describe techniques implemented to simulate basic character behaviour, such as object interaction and navigation, and will present the research on existing intelligent agent systems for the control of virtual characters.

2.3.1 Agent-Object Interaction

Throughout a simulation, many of the animations that an agent conducts will be based on interactions with the outside world. In allowing the agent to conduct interactions with objects in the world, a number of general approaches may be taken. One option is to provide the agent with low level rules and a learning model, and allow him to learn how to use objects. Unfortunately, this approach is not suitable where ready-made worlds with competent actors are required. Also, endowing individual agents with different mental models for every object in a large world would not be efficient in terms of storage. The other option is a system where there is a shared concept of how objects work. All agents in the system can have access to the same knowledge about how an object can be manipulated. Although this approach is not as realistic as the first approach, it decreases the complexity of the task enormously.

Smart objects extend the idea of object specific reasoning, whereby objects contain more information than just intrinsic object properties [Lev96]. A smart object is an object that is modelled with its interaction features, which are all parts, movements and descriptions of an object that have some important role when interacting with an agent. Smart objects provide the necessary parameters for motion generation. Features are identified in such a way as to provide important information to the motion generator. Smart object applications provide a number of advantages over more commonplace approaches: they decentralise animation control, separate high level planning from low level object reasoning and allow the same object

to be used in multiple applications. They also allow behaviours to be easily connected with high-level planners, and provide for Object Oriented Design since each object encapsulates data. Extensive research has been conducted by Kallman and Thalmann [KT98, KT99a, KT99b] on agent-object interactions using smart objects.

2.3.2 Navigation Strategies

Path Finding

Path finding is necessary for humans to navigate the environment they inhabit in a successful and realistic manner. In order to do this, the environment needs to store pathfinding information across 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. While various search algorithms exists, such as simple breadth or depth first searches, A* has become the standard in modern game development as it provides good, predictable performance without compromising optimality. To improve the realism of the path chosen, other important information can be stored in the nodes so that the search takes into account other heuristics in addition to distance, such as the danger element, or the terrain difficulty of a path. For a detailed discussion of the path-finding problem and the A* algorithm, see [Sto96, HS02].

Obstacle Avoidance

To prevent the human colliding with the environment and other agents, Tecchia et al. utilize a space discretization approach [TC00a]. Using a height-map to store the height of the environment at each point, the human performs collision avoidance if the difference in height between its current and next position is above a certain threshold. Otherwise, the human moves to its new position, updating his height above the ground based on the height-map. To avoid the humans getting too close to each other, a collision-map is used to store which positions in the map are occupied. The human's direction is adjusted depending on whether there are other humans in a 3×3 neighbourhood.

In [TLC01], Tecchia et al. extended their previous research on collision avoidance to use a platform that segments the virtual world into a 2-D grid in order to accelerate the development of agent behaviours. The 2-D grid is composed of 4 layers, where the grid cells in each layer contain specific data to govern the behaviour of individuals. The four types of layers are: inter-collision detection layer, collision detection layer, behaviour layer, and callback layer. The first two layers are used to compute collision detection between an agent

and its environment or with other agents, while the other two layers provide more complex and individual behaviours. The behaviour layer encodes specific behaviour in each grid cell as a colour. Depending on the cell that the agent is inhabiting, the colour-encoded behaviour instructs the agent to perform simple actions such as wait, or turn left. The callback layer provides for more complicated agent-environment behaviour using an event-driven approach. This allows an agent to perform actions, such as waiting at a bus stop and getting onto the bus, by activating the callback when the bus arrives. The main advantage of this layer is that, even though the associated behaviours are quite complicated, they are only executed when needed.

Steering Behaviours

Reynolds [Rey87] created an artificial life technique for simulating the flocking behaviours exhibited in nature by schools of fish, flocks of birds, and herds of animals, based on a particle system approach. Particle systems are a large collection of entities, each having its own behaviour or rules that alter its properties such as position and velocity. By simulating generic simulated flocking creatures, termed *boids*, as particles, Reynolds defined three simple steering behaviours from which the boids' flocking behaviour emerges. In addition to these steering behaviours, Reynolds improved the boid's navigational system by allowing them to perceive their dynamic environment in order to perform obstacle avoidance, and by simulating the laws of physics ruling the boids' motion e.g., gravity, thrust, and lift in the case of a bird. These three steering behaviours are:

- **Separation**:
  steers a boid to avoid other local boids.
- **Alignment**:
  steers a boid towards the average direction of heading of local boids.
- **Cohesion**:
  steers a boid towards the average position of local boids.

Since the pioneering work of Reynolds [Rey87], methodologies from many fields have been employed to address the problem of motion control for virtual humans [Vin97, Rey99, Pot99]. Although AI techniques [HP88, BY95, FTT99] have shown very promising results, such methods do not scale well with the number of virtual humans or obstacles and therefore are not suited for real-time applications. Alternatively, learning, perception, and dynamics-based techniques are easily adaptable to dynamically changing environments [TT94, NRTT95, HP97, BMH98b]. The idea of using force fields around virtual humans in order to guide them

originated from work on path planning in robotics [GLM98, GLM99]. Egbert and Winkler proposed a force-field technique which used a vector field around objects to prevent collisions between them [EW96].

2.3.3 Intelligent Virtual Agents

In spite of simulated virtual worlds appearing increasingly realistic, unless these worlds are populated with wholly believable virtual humans it is not possible for users to achieve the suspension of disbelief required for truly immersive simulations. One of the problems with current techniques for the control of virtual humans is that characters appear to have no existence outside of their interactions with human users [MC01]. By giving characters the appearance of being involved in their own lives, even when they are not involved with a human player, this would add an extra degree of believability that is lacking in current real-time applications.

Aylett et al. [AL00] present the *Spectrum of Agents* in an attempt to capture the differences between the numerous approaches to simulating virtual humans. Within this spectrum, physical agents inhabit one end while cognitive agents inhabit the other. Physical agents are mainly concerned with realistic physical behaviours and a significant example is the Virtual Stuntman project [FvdPT01], which gives virtual actors the capability of life-like motion. At the other end, cognitive agents are mainly concerned with reasoning, decision-making, planning and learning. A definitive example is Funge's cognitive modelling approach [Fun99]. However, the most effective systems sit between the two extremes of the spectrum. Amongst these are c4 [BID+01], used to simulate a virtual sheep dog with the ability to learn new behaviours, and the planning base *Intelligent Virtual Agent* system [CT00]. For video games, commercial solutions are also available including AI-Implant [AI.] which enables rule-based control of game characters and Renderware A.I. [Ren] which focuses on tactical behaviours.

The need for further realism is motivated by the notion of virtual fidelity, as described by Badler et al. [BBB+99]. This refers to the fact that the application should determine which capabilities a virtual human should display. MacNamee et al. [MDCO03] focus on controlling Non-Player Characters (NPC) in character-centric simulations i.e., simulations which focus on interactions between characters rather than action. This positions an agent of this system towards the cognitive end of the spectrum and leads to a specific set of fidelity requirements. These agents are required to behave believably in a wide range of situations, possess sophisticated social ability, behave in real-time, use few resources and ease authoring for game designers. It can be argued that none of the aforementioned systems satisfy this particular flavour of virtual fidelity.

The ViCrowd system [MT01] was created to generate and model crowds with various

degrees of autonomy. The crowd is modelled as a hierarchy of groups and individuals. Depending on the complexity of the simulation, a range of behaviours, from simple to complex rule-driven, are used to control the crowd motion with different degrees of autonomy (see Figure 2.9. The crowd behaviour is controlled in three different ways:

1. Using innate and scripted behaviours.

2. Defining behavioural rules, using events and reactions.

3. Providing an external control to guide crowd behaviours in real time.

In order to achieve the groups' and individuals' low-level behaviour, three categories of information are used: knowledge which is used to represent the virtual environment's information; beliefs which are used to describe the internal status of groups and individuals; and intentions which represent the goal of a crowd or group. The ViCrowd system has been used in various research projects, such as the simulation of virtual humans in networked virtual environments [PBC+01] and the simulation of crowd behaviours in panic and emergency situations [BMdOB03].

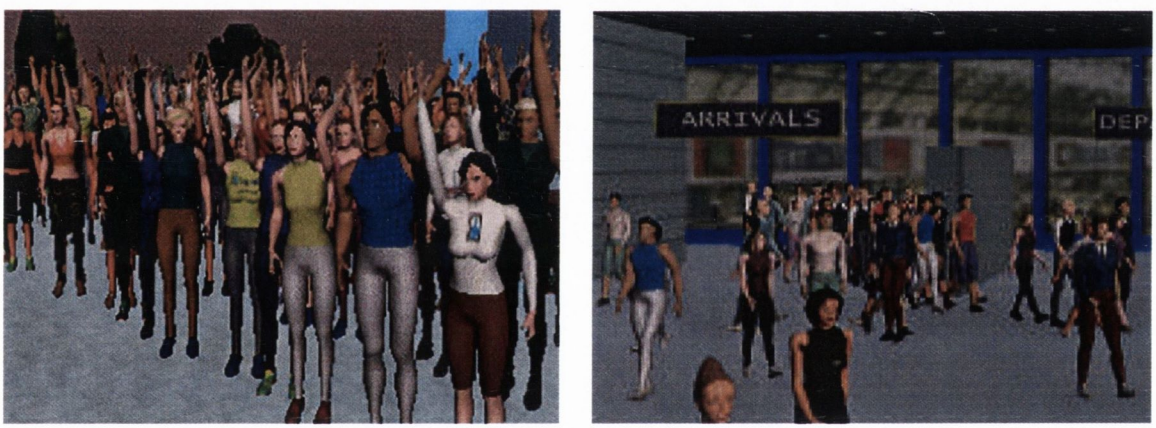


Figure 2.9: ViCrowd system.

2.3.4 Level of Detail Artificial Intelligence

Modelling a scene populated with large crowds is a challenging process. However, the simulation does not need to process every agent in order to present a believable experience. For example, agents that are not in the view-frustum should not worry about detecting future collisions with other agents. Hence, a virtual world needs to present a believable world, but this does not mean that it has to be an accurate model of a real world. Level of detail AI

(LODAI) reduces high CPU demands by approximating the behaviour of agents who are rated with a low-level of importance (e.g., if the agent is not in the view-frustum or is at a great distance from the viewpoint) with minimal perceptual degradation.

In BioWare's *Neverwinter Nights*, each character's level of AI depends on whether the character is a player character (PC), a non-player character (NPC) fighting or interacting with a PC, a NPC within fifty metres of a PC, a NPC in the same large-scale area of a PC, and finally a NPC in areas without a PC. Thus the classification of an agent's LODAI determines whether or not to exploit features such as processing frequency, the level of pathfinding detail, pathfinding cheating, and collision avoidance.

2.4 Summary and Conclusion

We have presented a detailed review of the techniques from the field of computer graphics, animation, and artificial intelligence that are required for the scalable simulation of believable virtual characters. 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. With regards to this thesis, various LOD techniques for the animation, visualisation and simulation of virtual humans are presented (see Chapter 4), since these techniques provide a viable means for the scalable simulation of characters.

We also introduced the layered model, which is typically used in animating characters in computer games, and forms the foundation of the virtual humans used in our system (see Section 3.3.1). General techniques for generating animation data for a character's skeleton were reviewed, and the suitability of these approaches for the animation of large crowds was discussed. An introduction to previous research conducted on controlling a character's motions through basic low-level behaviours was presented, and the implementation of these behaviours are described in more detail in Section 3.4. This is followed by a review of intelligent agent systems, and how LODAI techniques can be employed to deal with the computational demands of simulating large groups of believable characters. The integration of artificial techniques into our system is presented in Section 3.5.

Chapter 3

The ALOHA System

The ALOHA (Adaptive Level of detail for Human Animation) system is a framework, first proposed in 2000, with the aim of animating and rendering virtual humans in real-time [GMPO00]. Since then, various members of the Image Synthesis Group (ISG), in Trinity College Dublin's Computer Science Department, have been involved in its development and it has been utilized in various research projects.

This chapter describes the resulting system, which allows the real-time simulation of groups of autonomous characters in dynamic virtual environments. A brief overview of the system's architecture will first be presented, including a detailed description of the level of detail framework proposed by Giang et al., specifying what was actually implemented in the initial system. We will then detail the development history of the system, focusing on the author's contributions. A low-level view of the current virtual human model, including how it is animated and displayed within the system, will also be presented. The next section will present the low-level motions that were implemented in order to endow the virtual humans with some basic behaviours. Finally, we will describe how artificial intelligence techniques were integrated to produce a LOD framework for autonomous virtual humans that behave believably in a diverse range of situations.

3.1 The Original ALOHA System

The ALOHA system is a C++ application running on the Win32 API, utilizing the OpenGL rendering library as the core component of its rendering subsystem. OpenGL allows the development of graphics applications by providing an application programming interface (API) for hardware-accelerated rendering features. The OpenGL standard is maintained by the OpenGL Architecture Review Board (ARB) and the standard's specifications are revised regularly, allowing constant improvement. In addition, new hardware advances are developed and provided by video-card manufacturers as OpenGL *extensions*, and are included in the next OpenGL standard if approved by the OpenGL ARB. These extensions allow for new hardware innovations, including various hardware accelerated rendering techniques, that facilitate the real-time performance of the ALOHA system. The original ALOHA system was based on the framework proposed by Giang et al. [GMPO00] to solve the challenging problem of animating and rendering numerous virtual humans in real-time, as the computational cost of such simulations can quickly mount.

3.1.1 Original Framework

The framework was based on a LOD approach whereby it attempted to take advantage of the viewer's perception to compute less accurate models when loss of accuracy would be unnoticed. It amalgamated animation (Animation_{LOD}) and geometric (Geometric_{LOD}) LOD with the aim of providing a totally scalable system for virtual human simulation. In this section we will describe the proposed components of the framework, not all of which were actually implemented. The ALOHA framework consisted of the following four high-level modules which controlled the virtual human's animation and visualisation (see Figure 3.1):

- The ***Knowledge Base*** controlled the level of detail for a virtual human using simple rules.
- The ***LOD Resolver*** controlled the switching between different levels of detail and provided the parameters that guide both the motion and the geometric controller.
- The ***Motion Controller*** managed the animation of a virtual human at the current level of detail specified by the LOD resolver.
- The ***Geometric Controller*** managed the geometric detail of a virtual human's model at the level of detail specified by the LOD resolver.

A four layered model was designed for the virtual human consisting of skeleton, muscles, fat, and skin, in order to create perceptually realistic character deformations during motion.

Each layer exploited the functionality of the layer below, and by scripting the innermost layer (i.e., skeleton) for animation, they achieved consistent deformation. The muscle layer was implemented by preserving the volume of the muscles, modelled using ellipsoids. The fat layer was modelled using stiff linear springs to preserve the distance between the muscles and the skin. Finally, the skin layer was simply a mesh of points constructed from the outer spring ends used in the fat layer.

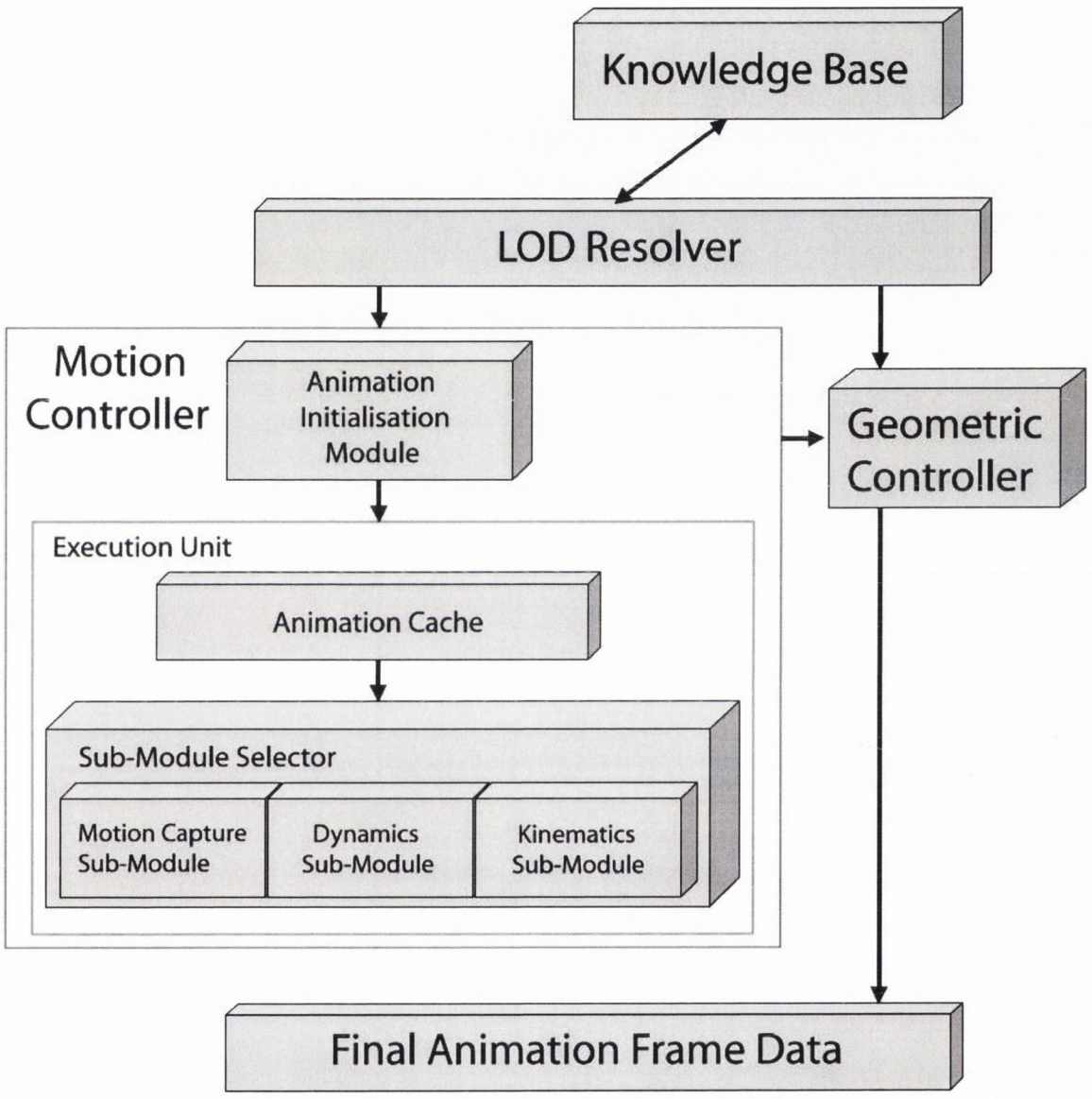


Figure 3.1: High level view of the proposed framework for the original ALOHA system.

Knowledge Base Module

The knowledge base module contained a set of rules, pre-defined by the user to account for hardware capabilities of the PC's processor and graphics card, that controlled the level of detail for a virtual human. Separate entries were defined controlling the virtual human's geometric and animation level of detail and were based on simple level of detail selection criteria (e.g., distance with respect to the scene's camera, velocity and position in the viewing plane).

LOD Resolver Module

The LOD resolver module controlled the switching between levels of detail, which was only allowed when the viewer would not perceive them. The resolver passed a virtual human's current animation and geometric LOD, determined by the knowledge base's rules, to the respective controllers on a master/slave basis. In the case of virtual humans that exhibited no motion, the LOD resolver did not call the motion controller to avoid needless computations that would be too expensive for scenes containing a large number of static humans.

Motion Controller Module

The motion controller handled any animations required by the virtual human at the specified Animation_{LOD}. The motion controller supported an initialisation module that initialised all necessary information needed for the animation method chosen by the LOD resolver. It also included a motion execution unit, containing an animation cache, which stored the skeleton's pose for re-use in subsequent frames, and several sub modules, which controlled the animation's production method (e.g., kinematics and dynamic).

To achieve a scalable framework, the LOD resolver had the ability to request an animation at different levels of detail. At the lowest level of detail, virtual humans were animated with pre-generated forward kinematics (either manually created or motion captured) to obtain high-speed, low-detail animation. Inverse kinematics was used for dynamically generating motions for situations where characters interacted with dynamically changing objects. These animations were assigned a number of iterations, depending on their importance, with which the inverse kinematics algorithm could solve the joint angles. Finally, for characters requiring a high level of realism, dynamic techniques were used to simulate the motion in question.

Geometric Controller Module

The geometric controller managed the virtual human's appearance based on the Geometric_{LOD} specified by the LOD resolver. Since the outermost layer (i.e., skin) needed to reflect the skele-

ton's current pose, peer to peer communication occurred between both controllers, which allowed the geometric controller to deform the layers based on the motion execution unit's state vectors. The tessellation of the skin was indirectly controlled by the detail of ellipsoids (specified by Geometric_{LOD}), since the number of springs in the fat layer was directly proportional to the number of vertices in the ellipsoids and the skin's vertices are made up of the outer end of the springs.

3.1.2 Implementation of the Original ALOHA System

The existing system was far from being complete, and was limited to displaying scripted characters. The LOD techniques and the four high-level modules that were discussed, were actually not implemented. Apart from view-frustum culling, the system did not take advantage of any other accelerated rendering techniques. Instead of the proposed 4 layered model, the first set of human models used in the system were animated with a 2 layered model consisting of the skeleton and the skin. The system contained a scene database that was capable of storing basic scene parameters (i.e., lights, cameras, and virtual humans) and data associated with the virtual humans (i.e, meshes, skeletons, and animations).

The character's skeleton could be animated solely with the lowest level of detail, utilizing pre-generated forward kinematics. Higher level of detail animations, generated by inverse kinematics and dynamics, were not actualized. The problem with just using forward kinematics for animating characters, is that this method cannot be used to dynamically generate animations, so a large range of pre-generated animations need to be created (which will be described in Section 3.3.1. The system did support the animation cache, which allowed animations to be updated at various frequencies. However, no rules were defined to select the frequency of these updates based on the character's importance. Furthermore, no geometric LOD techniques were employed to change the tessellation of the skin's mesh.

The low-level implementation of the scene's database, and the animation and display of the virtual humans in the original ALOHA system is shown in Figure 3.2. This will be discussed in more detail in Section 3.3, including how the system has been improved throughout its development.

3.2 ALOHA System's Development Time-Line

Since the system's framework was first proposed by Giang et al. [GMPO00] in 2000, it has expanded from a simple system, limited to the real-time visualisation of pre-scripted characters, to a more complex system, capable of simulating numerous believable characters in a wide range of situations. The system has gone through various development stages, requir-

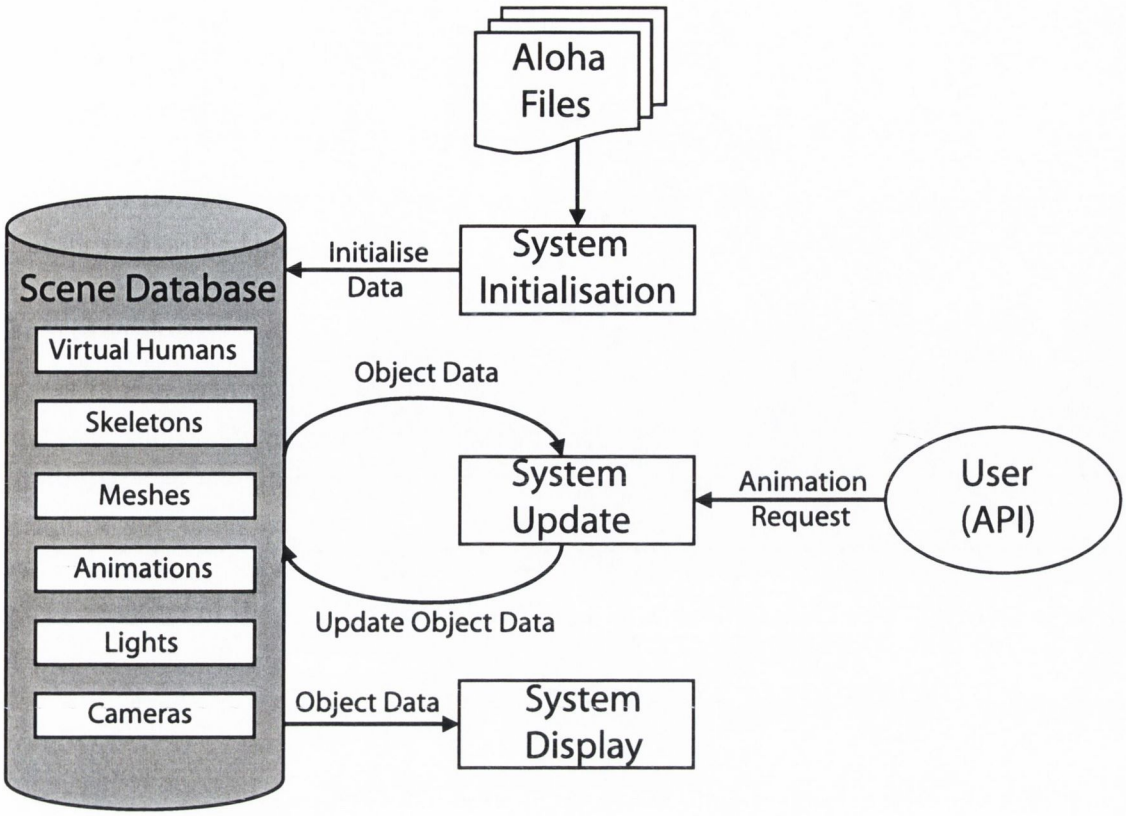


Figure 3.2: High level view of the implemented original ALOHA system.

ing an understanding of various techniques from the field of computer animation, graphics, artificial intelligence, and other areas related to the real-time simulation of virtual humans. Through this development process, this allowed us to gain insight on how the more challenging problem of simulating large crowd could be tackled, and assisted in the creation of the Geopostors System - a real-time crowd simulator. Starting with the ALOHA system's proposed framework, and finishing with the creation of the Geopostors system, the major stages involved in this development process are listed below:

- **2000:** Giang et al. [GMPO00] proposed the ALOHA system's initial framework, which was described in Section 3.1. The original implementation of the system was very basic, allowing the visualisation of virtual humans performing pre-scripted animations, and the majority of the techniques described in the framework were not implemented. The virtual humans utilized a two layered model consisting of a skeleton and skin (made up of a collection of rigid-body meshes), and could only be animated with pre-generated forward kinematics.

- **2001:** Since all of the models used in the system were modelled and animated in 3D Studio MAX, the first stage of the system's further development began with creating tools, which allowed the editing and exporting of these models to ALOHA-specific file formats (see Appendix B for a list of these files). The majority of this work, which was carried out as part of the author's undergraduate final year project [Dob01], involved becoming familiar with 3D Studio MAX's modelling and animation tools, and *MAXScript*, its scripting language which allows the building of custom import/export tools.

- **2002:** The research described in this thesis began with simple navigation and conversational behaviours being implemented by the author to provide a way of controlling the virtual humans' low-level motions, and thereby allowing them to walk around their environment without colliding with static obstacles (Section 3.4.1), and converse with each other (Section 3.4.3). Collaborative work resulted in the integration of artificial intelligence (AI) techniques to control these basic behaviours, resulting in the simulation of virtual humans with believable behaviour [MDCO02]. The system's framework was also extended for the simulation of groups and crowds, where O'Sullivan et al. [OCV+02] proposed levels of details for geometry, motion, and conversation. The use of subdivision surfaces for the control of the geometric LOD was investigated. However, on assessment it was found that they were not a viable solution for rendering crowds, and this work was abandoned (discussed in Section 4.1). Furthermore, collaborative work focussed on the generation of sophisticated conversational behaviours (Section 3.4.3). Higher-level motion generation was implemented by Peters and O'Sullivan [PO02b], which allowed the virtual humans to reach for objects using a vision-based approach with inverse kinematics. This virtual human model was further extended to include synthetic vision and a memory model based on "stage theory" [PO02a].

- **2003:** The control of the virtual humans' low-level motions was further extended to include the interaction with objects within their environment (Section 3.4.2). This involved the author integrating a smart object model into the system and allowing the object to control the virtual human's animation system. These objects also provided a means of driving the virtual human's gaze behaviour [PDMO03]. Peters et al. integrated the automatic generation of bottom-up visual attention behaviours [PO03b], and gaze and blink controllers for the execution of appropriate gaze motion in an expressive manner [PO03a]. Further collaborative work, involving the integration of level of detail AI techniques, resulted in a system capable of simulating believable virtual humans in diverse situations [MDCO03] (Section 3.5). This involved the author inte-

grating AI techniques developed by MacNamee into the system to control the virtual human's low-level motions.

At this stage of development, the display of the virtual humans accounted for the biggest overall system bottleneck due to the the number of triangles in their meshes. New meshes were acquired for the virtual humans, each consisting of a single deformable mesh, and linear blend skinning was implemented in the system. These models greatly improved the visual realism of the virtual humans and also provided some performance increase.

- **2004:** Utilizing various basic sub-components of the ALOHA system, the Geopostor system was created [DHOO05], with the aim of controlling the animation and rendering of large-scale crowds at various levels of detail (described in Chapter 4). Due to their geometrical simplicity, pre-generated impostor rendering techniques were integrated into the system, forming the foundation upon which the Geopostors system is built. This Geopostor system was integrated into an urban simulation system, and its performance was substantially increased by implementing hardware-accelerated occlusion culling, and taking state sorting and texture cache coherency into account (see Section 4.2). Simpler navigation behaviours were implemented to handle the computational strain of animating a crowd.

- **2005:** The crowds' low-level behaviour was improved by including random gaze behaviours, and allowing individuals to form groups. The system was made more complete by including another LOD representation - low resolution meshes. Perceptual experiments were carried out to find appropriate thresholds for selection of the virtual human's LOD representations [HDMO05], thus improving the crowd's visual realism (detailed in Section 5.3).

3.3 ALOHA System's Models

3.3.1 Virtual Human Model

A wide range of human models have been used in the ALOHA system. These models were bought from various 3D character artists and required editing in 3D Studio MAX before they could be used in the system. The virtual humans in the ALOHA System use two layer models, consisting of a skin animated by the underlying skeletal layer, which were created in 3D Studio MAX. Plug-ins were written in Maxscript by the author, to export these layers' data from 3D Studio MAX to simple text-based files that are needed to initialise, animate and render the model in the system.

The skeleton is an articulated structure composed of a hierarchy of linked bones, which was modelled in 3D Studio Max using the *Biped* plug-in. *Biped* allows the creation of a skeleton which can be easily animated in 3D Studio Max. In general, a skeleton consisting of 22 bones was fitted to each character's mesh (see Figure 3.4). However, it should be noted that a more detailed skeleton containing more bones is necessary for characters that need to perform more complex animations such as facial gestures and grasping.

The first set of human models in the original ALOHA system were based on those used in older generation games, whereby a separate mesh was linked to each bone in the skeleton. These models were made up of 22 separate meshes, consisting of 15,000 triangles in total, and were without any texture detail. Without texture-mapping, extra triangles were needed in particular areas of the model such as the head, which resulted in approximately 2,000 triangles (see Figure 3.3). Unfortunately, gaps frequently appear where two or more such meshes meet (i.e., at a joint) but this can be solved by covering these holes with specially designed meshes. However, this results in even more triangles being rendered at run-time and does not always produce aesthetically pleasing results.

Figure 3.3: Virtual human's 2-layer model consisting of a skeleton and a hierarchy of rigid meshes (without texture-mapping).

Previously, the virtual human's model was exported as a *3DS* 3D model format and loaded in using *lib3ds*, which is library for loading and saving 3DS files. Since the library

was quite new, it was poorly documented and quite buggy. Additionally, a lot of extra data was saved to the 3DS file, which was not needed for the initialisation of the virtual human's model. As part of the author's final year project [Dob01], a plug-in was written in MaxScript to export, from 3D Studio MAX to simple text-based files, only the data necessary for the virtual human's initialisation.

In the case of the skeleton, the plug-in allows each bone to be automatically assigned a unique ID, and exports the following data to an *ALOHA Skeleton Format* (ASF) file: the number of bones in the skeleton, each bone's ID, its parent bone's ID, and its local position with respect to its parent bone. To facilitate the setup of the model's hierarchy in the ALOHA system, the plug-in sets the name of each mesh to be the name of its parent bone. These meshes are exported to an *ALOHA Mesh Format* (AMF) file which stores each mesh's materials, vertices, vertex normals, texture coordinates, and triangles. These files are parsed into the ALOHA system and are used to initialise the virtual human's layered model.

In the ALOHA system, virtual humans are animated using pre-generated key-frame animations at the low end of the level of detail scale. Each animation's data is stored in a simple text based *ALOHA Keyframe Format* (AKF) file, which stores the following information: the number of objects animated; the duration of the animation (in seconds); the type of animation, which indicates whether it affects the entire skeleton, or specific parts e.g., the arms, the head, or the legs; and each object's key-frame data, which includes the time the keyframe occurs at and the object's position and orientation in local space.

At the initial stages of the system's development, there were no tools to export a character's animation data to this file format, and these files had to be manually created, which resulted in far from convincing results. To pre-generate and store plausible motions for the virtual humans, a plug-in was written in MaxScript by the author, which exported the animation data of a biped skeleton, animated in 3D Studio MAX, to an AKF file. The plug-in facilitated the altering of the animation's duration at run-time, by normalising the time at which each key-frame occurs before exporting the data into the AKF.

A large library of human animations was created in 3D Studio MAX using Character Studio's *freeform* or *footsteps* methods, or automatically through motion capture data, for a range of virtual human models. These animations were exported to AKF files, which allowed the virtual humans to perform a wide range of actions in the ALOHA system such as walking, sitting, standing, and various head and arm gestures (see Figure 3.4).

3.3.2 World Model

The aim of our research is to simulate large crowds of autonomous virtual humans within a 3D dynamic environment. In order to do this, the system's world model must store and

Figure 3.4: A virtual human's skeleton animated with various animations and the subsequent mesh deformation.

efficiently manage the resources and objects needed to simulate the scene. At the core of the world's model is the database, which stores the following types of data (described in various sections of this Chapter), for a particular simulation scene:

- Static Objects - store the properties of any static parts of the scene (i.e., buildings, roads, and props).
- Dynamic Objects - lights, cameras, interactive objects, and virtual humans.
- Skeletons - used to animate the virtual human models.
- Meshes - used for the visualisation of the static objects, interactive objects, and the virtual human models.
- Animations - pre-generated animations needed to animate the virtual human models and the interactive objects.

- Octree - used to accelerate the rendering of the static objects in the scene.
- Scene Path-Finding Data - used by the virtual humans to navigate within their environment.

A *Scene-Editor* plug-in was written in MaxScript by the author, which allows a modeller to assign simulation-specific information to a scene in 3D Studio MAX, and export this information to a simple text-based file. Through the tools provided by 3D Studio MAX, the plug-in facilitates the editing of arbitrary scenes by allowing the modeler to define specific attributes (e.g., static and dynamic objects within the scene, the meshes used for their visualisation, and the motions necessary to animate them) and this information is exported to a *Scene Descriptor Format* (SDF) file. At run-time, the ALOHA system provides a command-line interface that allows the user to specify the scene file to be loaded in, resulting in its information being stored the scene's database. Each virtual human is capable of accessing the database, since it stores the needed parameters and resources for the behaviour, animation, and visualisation of a virtual human.

3.3.3 Animation and Rendering System

Animation System

The virtual human's animation system controls how it is animated in its virtual environment. When a virtual human requests to perform an animation, a *task* is created to store the following information needed for the character's animation: its AKF file's keyframe data; a list of the bones' names that the animation affects; its start-time, which indicates the time at which the virtual human should perform the animation; its duration, which stores the length of the animation in seconds; its priority, which stores the animation's LOD with respect to importance; and its weight, which is used for motion blending with other tasks. Tasks are implemented within a class hierarchy, with a generalised animation task at the top, which is extended to implement more specific animations such as: animating a skeleton in local space, animating a virtual human in world space, and animating a virtual human in world space along a path.

The animation system manages the character's requested tasks and animates the character based on these requests. The animation system stores a list of the character's bones (that it can animate) and a list of tasks (that the character wishes to perform). Once a character requests to perform a task, it is passed to the animation system and stored. Originally, the animation system was updated every frame, which can be computationally expensive as the number of characters increase. However, in the current system, the update frequency is

dependent on the importance of the character (specified by an Update_{LOD} variable), where low-rated humans are only updated at 5Hz, and high-rated humans are updated at 30Hz. The state of each task in the animation system's list is updated using the current time of the world and, depending on it start-time and duration, it can have one of the following states: *waiting*, *running*, or *finished*.

Once each task's state is updated, the animation system cycles through the list of bones and makes a separate list of running tasks that affect the current bone. Whilst making this list, it stores the priority of the task with the highest priority that is not greater than Animation_{LOD}. After creating this list, it cycles through and removes any tasks whose priority is less than the highest priority. Sorting this list based on the Animation_{LOD} is essential as this discards low-rated tasks when the virtual human's Animation_{LOD} is high, and vice-versa. For example, when a character performing a lowly-rated walk animation requests to play a highly-rated arm animation, this animation is ignored when the Animation_{LOD} is low and the arm bones are updated using the walk animation.

If only one task affects the current bone, the bone's position and orientation is interpolated using the task's key-frame data and the current world time. If several tasks with the same priority affect the current bone, the bone's position and orientation is interpolated in the same way, except that the amount of influence each task has on the bone is calculated using the task's weight, which allows the blending of tasks. Spherical interpolation is used for the bone's orientation, while linear interpolation is used for the bone's position. After the animation system has updated the skeleton's bones, the bones' global transformation matrices are stored in an animation cache.

Rendering System

The system was enhanced with the inclusion of an *octree* structure to accelerated the rendering of the scene's static objects. The octree is used to spatially partition the triangles of each static object's associated mesh and store them in its sub-nodes. Large unseen portions of the scene are not rendered by discarding the octree's higher-level nodes, and therefore their sub-nodes, whose bounding volumes are not within the camera's view-frustum. Since the majority of scenes used in the earlier simulations were indoor environments (e.g., a lecture room, and a pub) or a small subset of an outdoor environment (e.g., a street with several buildings), occlusion culling was not deemed important as these scenes contained a small number of occluding objects. However, a hardware accelerated occlusion culling method is described in Section 4.2.3 to determine the virtual humans occluded in a large-scale city environment.

In the case of a dynamic object, such as a virtual human, the associated mesh is not

rendered if the object's bounding volume is not within the camera's view-frustum. To render a virtual human, each mesh's vertices, normals and texture coordinates are all stored in separate OpenGL *vertex arrays* and the triangles are rendered using Gouraud shading implemented in OpenGL's fixed function pipeline. The mesh is transformed by the global transformation of its parent bone, which is stored in the animation cache. The main advantage of using a collection of meshes to represent the human's skin layer, is that it is computationally less demanding to animate as the position of each vertex can be stored once and subsequently transformed.

3.4 Virtual Human Innate Behaviour

The original ALOHA system allowed the users to view virtual humans and script them to perform a specific animation. While this is useful for testing various animation and rendering methods, the application of the initial system to the simulation of virtual humans was extremely limited. For example, a scene populated with several virtual humans would involve predefining which animations each of them would perform, and resulted in a uninteresting, cyclical simulation. By endowing the virtual humans with basic or innate behaviours, this allowed them to behave in a more autonomous manner, thus resulting in a more interesting simulation. The following behaviours were implemented to provide low-level motion control: *navigation behaviour*, *agent-object interaction behaviour*, and *conversational behaviour*.

3.4.1 Navigation Behaviour

In order to reduce the virtual humans' user-controlled constraints, a navigation behaviour was implemented to allow the humans plan their course as they walked within a virtual environment. Previously, in the original system, the virtual humans could be scripted with a cyclical key-framed walk animation and they walked either in the direction they were facing (usually through other objects) or along a pre-scripted path defined in 3D Studio MAX (and exported to an AKF file). To allow the virtual humans to realistically navigate themselves towards a destination, without colliding with static obstacles inhabiting the scene, path-finding is necessary. It should be noted that, in the ALOHA system, virtual humans currently do not perform collision avoidance with dynamic objects such as other virtual humans inhabiting the scene.

Path-finding is accomplished in the ALOHA system by splitting the scene into a grid of accessible or unobstructed nodes, and performing an A* search to return the shortest unobstructed path between two nodes. When a virtual human requires to walk to a particular destination, the nodes corresponding to its current position and destination are passed to its

navigation behaviour function. The behaviour module performs the A* search on these nodes and returns the shortest path (as a list of collision-free nodes), used to initialise a *walk* task, which constrains the motion of a pre-generated walk animation along the path.

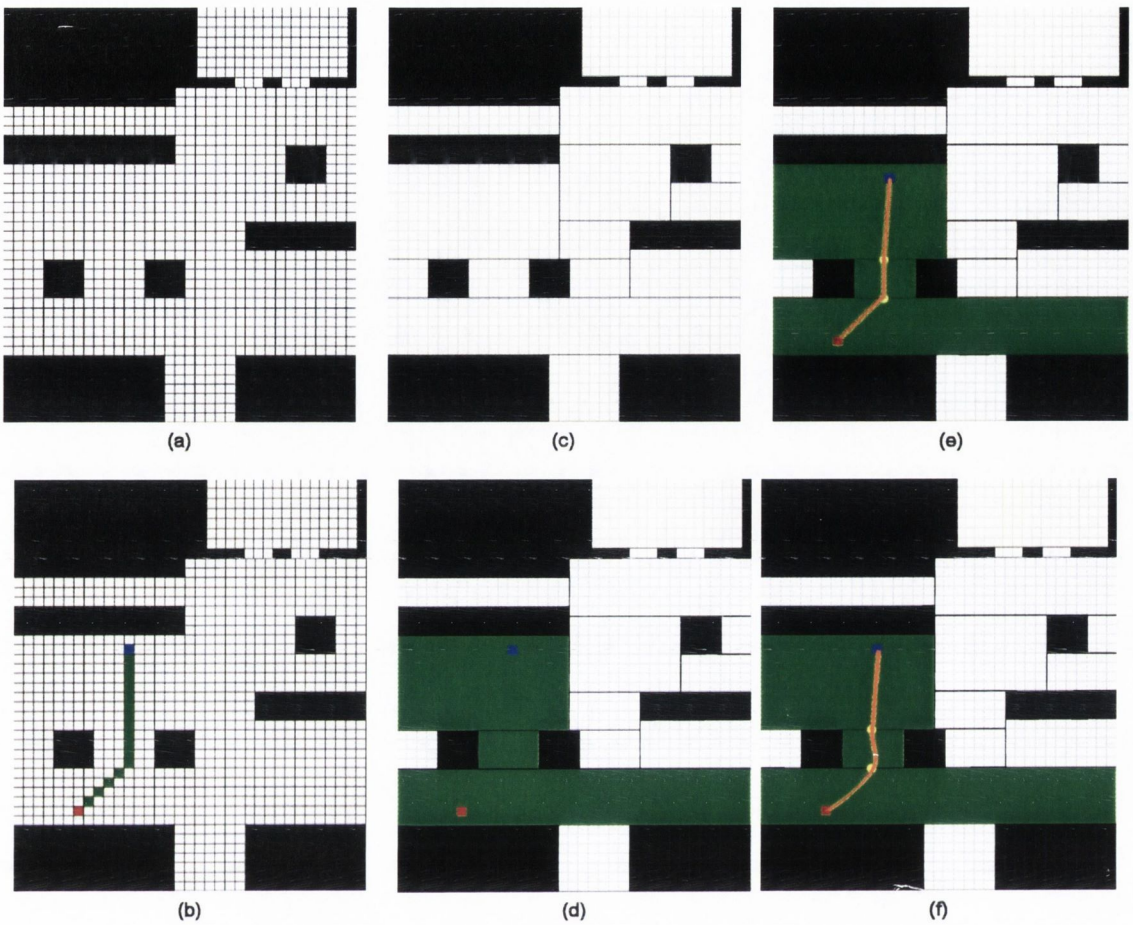


Figure 3.5: (a) Grid consisting of fixed-size nodes, (b) Path returned by A* search for fixed-sized nodes, (c) Grid consisting of variable-sized nodes, (d) Path returned by A* search for variable-sized nodes, (e) Straight Line path (Low LOD) and (f) B-spline path (High LOD).

The *Scene-Editor* plug-in, described in Section 3.3.2, allows the splitting of the scene into a regular grid of user-defined nodes, where each node stores whether it is walkable or blocked (see Figure 3.5 (a)). The plug-in exports the grid information to a simple text-based *MAP* file. When loading this file into the ALOHA system, each node's accessible neighbouring nodes are computed and stored. With a fixed-sized grid, each node needs to store only the node accessible in the North, South, East and West direction. The time taken for the A* search to find the shortest path was reduced by constructing a grid of larger rectangular

nodes of either walkable or blocked values (see Figure 3.5 (c)). Since the nodes are of variable size, several nodes can be accessible from a node in a particular direction. To take this into account, each node stores four separate lists of nodes, where each list contains the nodes accessible in one of the directions. For example, Figure 3.5 (b) and (d), shows the nodes returned by the A* search (shown in green), between its current position (shown in red) and its destination (shown in blue).

For virtual humans far from the viewer, the path stored in the *walk* task can be constructed as a series of straight lines connecting the position of each node in the path. Since the nodes are of different sizes, a node's position with respect to a neighbouring node is the mid-point of their connecting edge (shown in yellow in Figure 3.5 (e) and (f)). When the *walk* task is updated in the animation system, the virtual human is orientated towards the position of the next node, and the human is translated along this direction. Once the virtual human reaches a node in the path, its direction is oriented towards the next node and this continues until it reaches its final destination node. The problem with using a straight-line path is that it can result in the virtual human performing sudden large and unrealistic changes in the direction it is walking (see Figure 3.5 (e)). Therefore, it should only be used when these artefacts are imperceptible to the viewer. For a higher level of detail, a B-spline curve is plotted, using the path's nodes as control points, to provide smoother directional changes (see Figure 3.5 (f)). This requires additional computations to translate the virtual human along the curve, but results in a more realistic behaviour and therefore should be employed for important characters.

3.4.2 Agent-Object Interaction Behaviour

Since real humans use objects in their everyday life, it is essential for virtual humans to be capable of interacting with objects in order to provide a more believable simulation. Our approach is based on research conducted on agent-object interactions using smart objects [KT98] [KT99a] [KT99b]. A smart object is an object that is modelled with its interaction features, which in turn provides the parameters needed to generate the agent's motion in order to simulate this behaviour. Our smart object model is based on that from [Kal98], but has has a number of compelling differences. The most important difference is that our smart object model is constructed in such a way as to promote the objects as being central to interactions between characters. The following sections will describe the key features of our smart object model.

User Slots

The first component involved in an agent's use of a smart object is a *user slot.* Each smart object can have any number of user slots associated with it. These can be considered dummy objects indicating, firstly, where an agent should stand when they begin using the object and in which direction they should face (see Figure 3.6 (a)). User slots are also labelled to indicate their type and this implies what kind of interactions can be performed at that slot. For example, Figure 3.6(a) shows an illustration of a *bar object* implemented as a smart object. The bar object contains two user slots: the barman slot and the general slot. Only a barman agent can use the barman slot and he must face in the direction of the bar, while the customer (who uses the general slot) should stand at the other side of the bar, facing the barman slot. Before any agent can begin to use a smart object, they must first obtain a free user slot of an appropriate type. Once a slot is obtained, the virtual human walks to the slot's position using the navigation behaviour (see Section 3.4.1), taking the direction that the agent should be facing at the slot into account by calculating extra control points for its path.

The use of user-slots is a departure from the smart object model used in [KT98]. The main advantage of user slots is that they avoid all of the concurrency problems that arise through the use of rules alone. This is particularly important if we are to have agent interactions centred around smart objects. For example, the bar object allows bar patrons to order a drink from the barman, the barman pours these drinks and gives them to the patrons, and the patrons pay for their drinks and drink them. All of this could lead to serious concurrency problems, making the user-slot notion particularly appealing. The biggest disadvantage of using user slots is that they can lead to slightly repetitive behaviours, as agents will always stand in the same position, and follow the same series of steps. However, this can be overcome by providing a range of user slots for each object, and pre-generating a range of animations that the agent can perform.

Usage Steps

Each user slot contains a number of *usage steps* which describe, in a step by step manner, how an agent should use an object. There are a number of key pieces of information at each usage step:

- the information required to animate the agent and the object at this step,
- the conditions which must be met in order for the agent to move onto the next step,

- details of any changes which are to be made to either the agent's attributes or the object's attributes on completion of this usage step,
- details of any information which should be passed to users of this object on completion of this step,
- whether or not the agent is free to socially interact with other agents while at this step,
- points of interest on the object upon which the agent should focus while at this usage step.

Our smart object model supports the use of animation LOD by allowing the usage steps to provide animation information at various levels of detail, which could be selected based on the agent's Animation_{LOD}. At the lowest level, the character performs a pre-generated key-frame animation, while highly rated characters could use inverse kinematics to solve for any target position specified at the usage step. When the agent-object interaction results in the agent moving or animating the object in same way, it uses the pre-generated key-frame animation specified for the object at that usage step. However, inverse kinematics or a physically based model could be used to generate the object's animation at a higher LOD.

After the animation information, the most important aspect of a usage step is the condition which allows the character at that step to move on to the next usage step. Conditions can take one of two forms. The first indicates that the agent must only wait for the animations required by the current step to be complete in order to move onto the next step. The second, and more interesting, form that a condition may take is that an agent at a particular slot must wait for an agent at another slot to reach a particular usage step in order to move onto the next step. It is in this way that we allow character interactions to be centred around a smart object.

Figure 3.6 (a) shows an illustration of the usage steps involved in using a bar object at both the barman and general user slots. The arrows show the conditions involved in moving from one usage step to the next. Usage steps with arrows going directly from that step to the next (for example going from step 0 (ASK) to step 1 (LISTEN) of the barman user slot) only require that the animations required by the first step have been performed in order to move on to the next step. If an arrow starts at a point along another arrow moving from one step to the next, and leads to the start of a step of another user slot, then the character at the current usage step must wait until the character at the other slot reaches the usage step indicated by this arrow. For example, if an agent at the general user slot is at step 1 (LISTEN), then they must wait until the agent at the barman slot has reached step 1 (LISTEN), before they can move on to step 2 (ORDER). Once a usage step is complete, changes can be made to

the attributes of both the users of an object and the object itself, and these are listed in each usage step.

Characters' decisions to use particular objects are based on internal motivations crossing thresholds (which will be described in Section 3.5) and these motivations are then adjusted based on the user attribute changes listed in a particular usage step. Along with attributes changing, important pieces of information must also often be passed between characters using the same object. These are also listed in the usage step. At some usage steps (for example the WAIT steps for the bar object) characters are free to engage in social interactions with other nearby characters (utilizing the conversational behaviour which will be described in Section 3.4.3). Whether or not such interactions are allowed is indicated at each step.

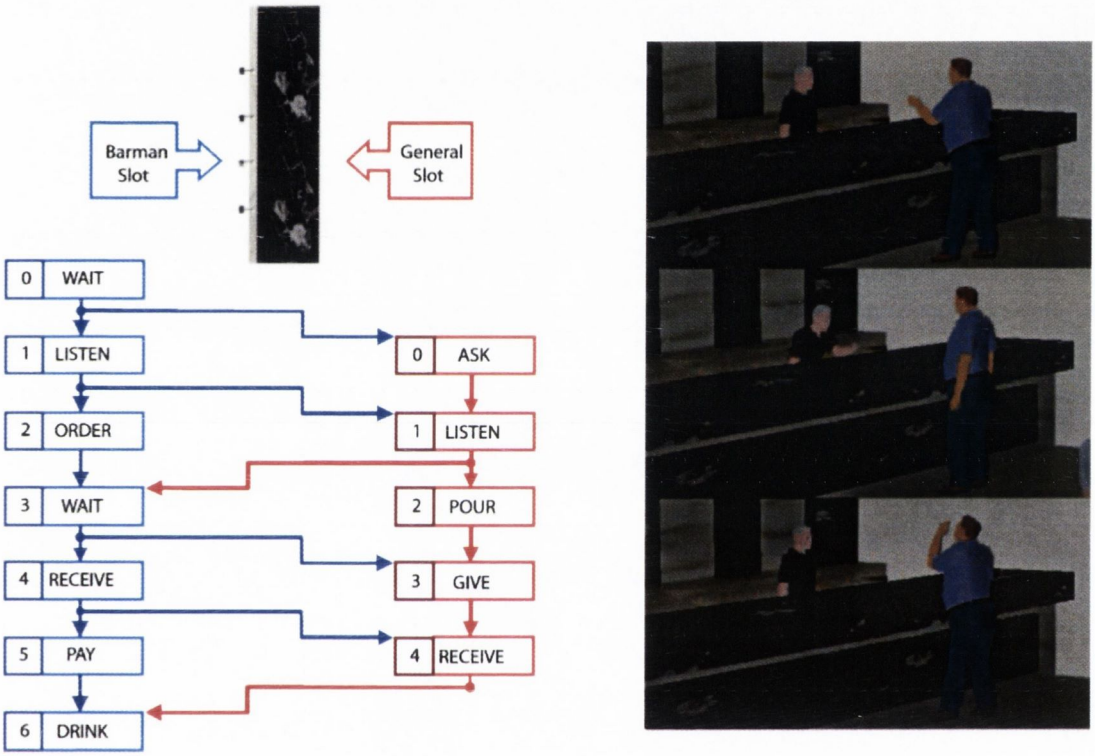


Figure 3.6: (a) A smart object model for a bar object showing its user slots and usage steps for both slots. (b) A *barman* and *customer* agent interacting with each other using the smart object.

Smart Objects for Attentive Agents

Our smart object model was extended to include information for directing the attention of agents when interacting with objects. Such information is useful for driving gaze behaviours,

for example when grasping objects. Our system supports both bottom-up (attention capture) and top-down (task driven) simulation of behavioural animation on a per-object basis [PO02a, PO03b, PO03a]. More details on our smart object model for attentive agents can be found in [PDMO03].

Smart Object Exporter

In order to allow world designers to define smart objects, an object editor plug-in was written in MaxScript by the author. The object editor provides an interface that allows the assignment of the interaction attributes to an object in 3D Studio Max and the exporting of these attributes to an XML file. The creation of a smart object begins in 3D Studio MAX with the selection of the mesh used to represent the smart object in the ALOHA system. The plug-in then allows a user to create and transform a user-defined number of user slots for the object. Once the user slots are in place, the plug-in allows various attributes to be defined, facilitating easy creation of smart objects. The smart object's data is exported to an XML file, which uses a proprietary DTD (Document Type Definition) developed by MacNamee [PDMO03]. This XML file is loaded into the ALOHA system and provides the agent-object interaction behaviour with the information on how the virtual humans can interact with and attend to the object.

3.4.3 Conversational Behaviour

When people come together to socialize and engage in conversations, they become players in an elaborate game governed by many complex rules, both explicit and implicit. These rules are needed to effectively establish and maintain channels of communication among multiple participants. The conversational conduct is carefully coordinated so that, for example, speakers get heard, listener feedback gets seen and topic changes can be negotiated. A large portion of the coordination is carried out by nonverbal behaviors. For example, those who wish to get the floor often bring their hands up from resting position to indicate their intent, and in turn receive a gaze cue from the current speaker as confirmation that the floor will indeed be turned over to them. While speaking, the speaker only needs to look at a listener and quickly raise their eyebrows to elicit immediate listener feedback such as a head nod [Gof63] [Goo81].

Computer games usually contain several *cut-scenes*, where characters tell each other important information related to the game's storyline. This involves the characters playing though a script, usually consisting of a pre-recorded conversation sound file, and pre-generated animations for the characters' skeleton and facial bones. The resulting end-product is ex-

tremely realistic, since the conversation is entirely pre-scripted, thus allowing the conversational conduct to be carefully coordinated. However, as participants in this conversational game, we quickly spot that the *in-game* behaviours used for the scene extras conversing in the background do not simulate the natural conversational flow. These less detailed behaviours involve the character playing back some pre-recorded sentence, rarely with any meaningful gestures, and typically break the rules of conversational conduct. Additionally, the majority of the time we are unsure of who is actually speaking, since the character says something without giving us any animation cues. Even though these scene extras are less important and therefore their behaviour should not be as noticeable, these anomalies can have a jarring effect on those that are observing the event.

A framework was developed by the author to allow agents to have the same properties as humans in face-to-face conversation, including the ability to produce and to respond to both verbal and nonverbal behavior [CVC+99]. Such characters, termed Embodied Conversational Agents (ECAs), have to deal with both multimodal generation and interpretation of conversational content (e.g., what is being said) and coordination (e.g., taking and giving the floor). The framework uses a LOD approach, whereby an agent's conversation is controlled with various levels of detail, termed $\text{Conversation}_{LOD}$. At the core of this framework lies the conversational behaviour, which controls the ECA's LOD based on visual and functional properties. When an agent decides to interact with other agents, its conversational behaviour is initialised, which involves keeping track of the other agents involved in the conversation and assigning the agent either a *talk* or a *listen* behaviour (depending on whether there is anyone else in the group currently talking). Additionally, the setup of the conversation behaviour depends on the following Converation_{LOD}:

- **$\text{Conversation}_{LOD} = 0$**
  This LOD is selected when the agents involved in the conversation can be seen but cannot be heard and the agents are not rated as important. The nonverbal communication is randomly generated and no conversational rules are maintained.

- **$\text{Conversation}_{LOD} = 1$**
  This LOD is selected when the conversing agents become the focus of the user's attention, but the conversation still cannot be heard. The nonverbal communication is randomly generated without repetition, whilst still maintaining the basic rules of the conversation.

- **$\text{Conversation}_{LOD} = 2$**
  This LOD is selected when the agents involved in the conversation can be seen and heard, and the conversation's dialogue is scripted. While it would be impractical to

store a large database of pre-recorded sentences, each agent should have at least several sentences to avoid some repetition. To avoid the agents gesturing the same way each time they say their line, several gestures associated with these sentences should be pre-generated , taking the actual linguistic and contextual analysis of the sentence into account. These gestures could be dynamically generated in order to achieve a higher LOD.

At the lowest level of detail, since the conversation dialogue cannot be heard, pre-generated gestures associated with the agent's current behaviour are randomly selected. Based on the research conducted by Cassell et al. [CVB01], nonverbal behavioural gestures for the head and arms were created in 3D Studio MAX and exported to separate AKF files. The arm animations were generated based on the following gesture types: beat, contrast, iconic, and point. The head gestures consisted of animations simulating various degrees of nodding in approval and disapproval. When the agents are far from the viewer, these animations are chosen at random and changed at different intervals, in order to give an impression of varied activity to the crowd.

As the conversation's LOD increases, the first step in improving its realism is by establishing and maintaining the channels of communication between the agents, which can be achieved through gaze cues. Both the talk and listen behaviours control the agents' head movements with respect to whom they should look at during the conversation. The listener behaviour selects the agent currently talking, while the talker randomly selects another agent in the group. The behaviour dynamically generates a simple key-frame animation to rotate the agent's head in the direction of the agent chosen as the focus of attention, which is passed to the animation system as a task.

The next step is to randomly generate nonverbal communication between the agents so that it reflects their role in the conversation. The talk behaviour creates a task initialised with one of the arm gesture animations selected at random, and passes it to the animation system. A higher degree of variety is achieved by reusing and altering these gestures' animation data to generate new animations. The approach used involves mirroring the original animation so that either or both of the agent's arms can perform the gesture. Each time an agent finishes performing an arm gesture, the talk behaviour requests new gestures until the agent's conversational behaviour decides that it should stop talking. When an agent decides that it has talked for long enough, its conversational behaviour first replaces the talk behaviour with a listen behaviour, and informs someone else involved in the conversation (randomly selected) that it is his turn to talk. Since the nodding of the head is enough to reflect the listener's feedback, the listener behaviour randomly selects a nodding gesture, which is initialised as a task and passed to the animation system. The animation system uses the animation cache

to access the current orientation of the head so that the nodding gesture can be performed in the direction that the head is facing. Figure 3.7 shows various conversations simulated at the lowest $\text{Conversation}_{LOD}$. It should be noted that a higher conversational LOD can be achieved using facial gestures and lip synching.

While facial animation greatly improves the realism of a character, we wished to focus on how less detailed character models could be animated to produce plausible conversational behaviour. The inclusion of more detailed models with facial and hand animations was carried out as part of two undergraduates' final year projects and implemented with some success. At a higher LOD, more complicated behaviours and rules are needed to simulate a conversation in a realistic manner. BEAT (Behaviour Expression Animation Toolkit) [CVB01] could be used to simulate more complex nonverbal behaviours, since it includes rules for turn-taking, speaker explicit addressing, feedback elicitation and corresponding listener feedback. Outputs of BEAT were actually used to create the pre-generated gestures used in our system, but in a static way [OCV+02].

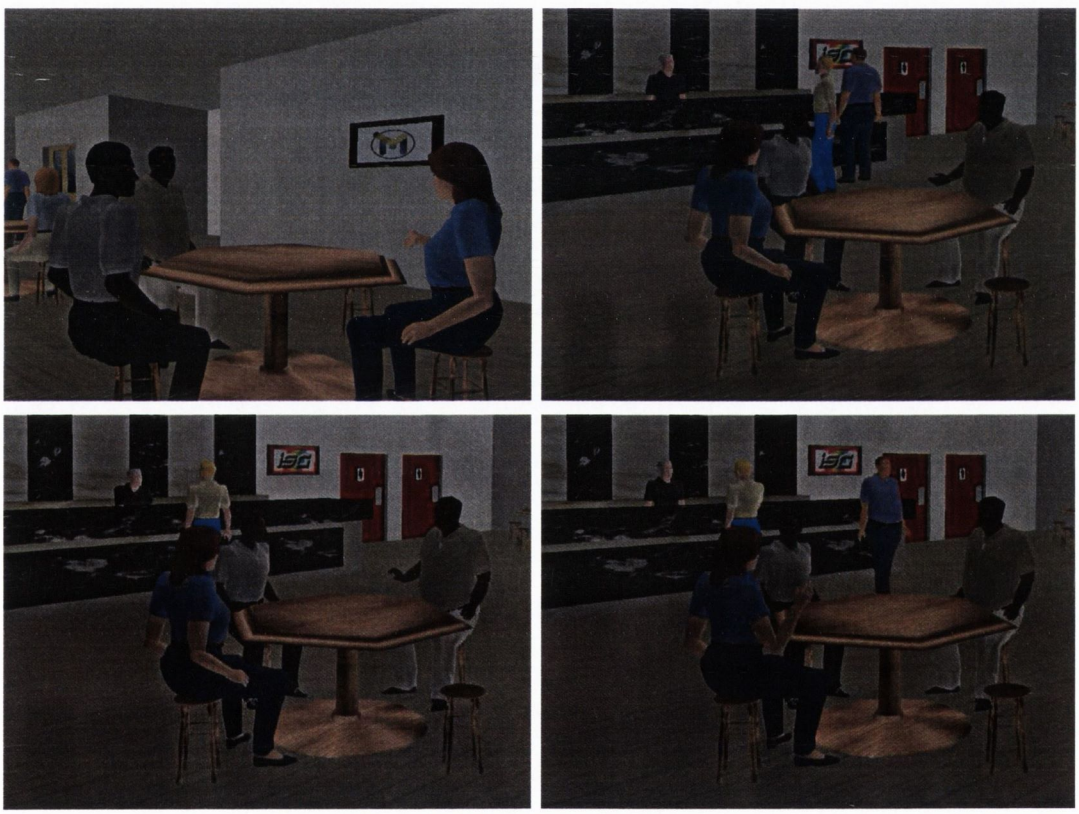


Figure 3.7: Characters using the conversational behaviour in the ALOHA system.

3.5 LODAI for Autonomous Virtual Humans

Collaboration between research groups focusing on different aspects of the overall problem of simulating large groups of virtual humans is one means of allowing groups to focus on their own particular research goals, and still pursue the goal of creating a complete simulation system. At the initial stages of the ALOHA system's development, scenes were strictly based on pre-defined scripts and a means for automatically driving the behaviour of virtual humans was needed. In this section, we describe how research based on Artificial Intelligence (AI) for computer games, was integrated into the ALOHA system by the author to create dynamic virtual environments inhabited by intelligent virtual humans.

The Artificial Intelligence Group (AIG), in Trinity College's Computer Science Department, is involved in applying sophisticated AI techniques to the realm of computer games [FFMC01]. The work most related to the ALOHA system is on using intelligent-agent technologies to add depth to computer controlled Non-Player Characters (NPCs) in character-centric video games i.e., games which focus on character interaction, rather than action. These agents are required to behave believably in a wide range of situations, run in real-time, use few resources and ease authoring for game designers.

In adventure and role-playing games, there is a trend for computer controlled NPCs to be very simplistic in their behaviour. Usually, no modelling of NPCs is performed until the player reaches the location in which an NPC is based. When the player arrives at this location, NPCs typically wait to be involved in some interaction, or play through a pre-defined script. This leads to very predictable, and often jarring behaviour. For example, a player might enter a room and meet an NPC who would perform a set of actions based on some script. However, if the player were to leave that room and re-enter, the NPC would play through the same script again. In order to overcome these limitations, MacNamee et al. [MC01] developed the Proactive Persistent Agent (PPA) architecture and its technique of role-passing which allows intelligent agents to take on different roles depending on the situation in which they are found.

By integrating this architecture developed by MacNamee into the ALOHA system, the synergy between these two projects brings significant benefits to both. Firstly, by using characters implemented using the PPA architecture, dynamic scenes can be created within the ALOHA system, whereby the behaviour of virtual humans is driven by their roles, rather than being scripted. Secondly, for the game AI project, the ALOHA system offers a graphically sophisticated test-bed for the newly-developed agent technologies.

3.5.1 The Proactive Persistent Agent Architecture

Agents based on this architecture are proactive in the sense that they can take the initiative and follow their own goals, irrespective of the actions of the player. Persistence refers to the fact that, at all times, all NPCs in a virtual world are modelled at least to some extent, regardless of their location relative to that of the player. This complements the use of level of detail AI (LODAI), whereby the characters can be controlled at higher or lower levels of sophistication based on their position, with respect to the player, in a virtual world.

The architecture has three key components - the Schedule Unit, the μ-SIC system, and the Role-Passing Unit. With regards to the integration of the PPA architecture into the ALOHA system, these three component's are stored in the virtual human's brain module. These components are connected to the scene's database and to percepts which come from the simulation itself, informing a virtual human about nearby objects, events and other characters. Based on these percepts and the contents of the scene's database, these units control an agent's behaviour and therefore the animation of the virtual human model.

The Schedule Unit

In order to enable PPAs to give the impression of having a life beyond their association with simulation users, it should be possible to observe a single character as, for example, she begins her day at home, goes to work, visits a bar and, finally, returns home again. The character should behave competently in each of these situations, and in this way the illusion will be created that the character has a life beyond her involvement with simulation users. To achieve this, each agent is equipped with a schedule (written in XML) indicating where they should be, and what they should be doing (i.e., which role they should adopt) over the course of a day of a simulation.

The μ-SIC System

The μ-SIC system is used when, as dictated by the role passing unit, a character is free to socially interact with other characters, or human users. The μ-SIC system uses a number of sophisticated psychological models to simulate a character's personality, moods and relationships. In order to use these models to drive the character's behaviour, an Artificial Neural Network (ANN) takes the values of the models as inputs and outputs which interaction a character should perform (e.g, chat, flirt, joke or insult). Currently, μ-SIC is only used in the ALOHA system for developing social relationships between agents based on their personality (e.g., characters only talk with other characters that share common interests). However, future work will involve applying emotional transforms to a character's animation to reflect

mood. For further details on how the μ-SIC system is implemented, see [MC03].

Role-Passing Unit

The technique of role-passing allows agents to assume different roles over the course of a simulation, whereby a role controls an agent's behaviour. The goal of this unit is to allow the NPCs to display believable behaviour, and behave believably in a wide range of situations within the same simulation. Role-passing works by layering different roles, which are dictated by the schedule unit, upon a basic agent. These roles endow the agent with basic behaviours that are driven by a small number of basic motivations and lead to a particular action. These basic motivations can change both between simulations and between agents in the same simulation.

The behaviour of the PPAs are controlled using *Fuzzy cognitive maps* (FCMs). FCMs are directed graphs in which nodes represent fuzzy concepts and arcs represent fuzzy rules, or the causal flow between concepts. At each concept node in an FCM, the activation levels of its input arcs are summed to give the node's activation, and this is passed on to a linked node through an output arc. For the purposes of the PPA architecture, an agent's FCM (termed PPAFCM) uses the following range of different fuzzy node types: concepts, motivations, actions, event occurrences, rules, and persistent flags. For more information on what each of these nodes represent, see [Mac04].

For example, Figure 3.8 illustrates a sample PPAFCM used to control a basic agent. The concepts for this basic agent are "Fear", "Fatigue", "Bladder", "Hunger" and "Socialise", all of which are driven by motivations bearing the same name. The motivation nodes represent an agent's internal motivations and maintains response curves which describe how the agent's motivations increase or decrease. Each concept node has an action node attached to it, which results in the agent performing an action suitable to that concept. The action nodes for the "Fear", "Fatigue", "Bladder", "Hunger" and "Socialise" motivations are, respectively, "Flee", "Sleep", "Pee", "Eat", and "Mingle". These nodes define how the agent should act.

The action node is a key component that allows the simulation of autonomous virtual humans in the ALOHA system. Once activated, this node specifies which of the basic behaviours the virtual human's brain should assume, which in turn controls its low-level motion. For example, the "Flee", "Pee", and "Mingle" action nodes would result in the virtual human's brain respectively assuming: the navigation behaviour, initialised with a a risk-free destination point; the agent-object interaction behaviour, initialised with a *toilet* object that can be used; and the conversational behaviour, initialised with the other agents involved in the conversation.

Role-passing operates by using the schedule unit to layer appropriate roles on top of a

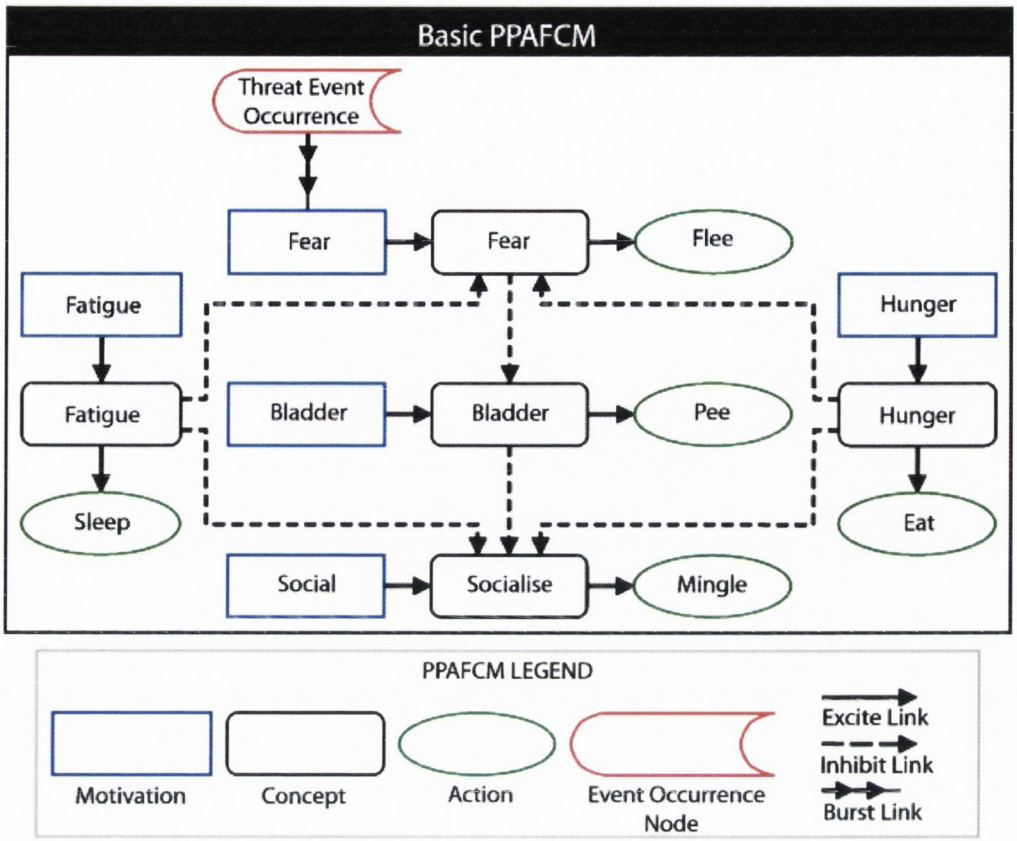


Figure 3.8: A sample PPAFCM for a basic agent.

very basic agent, at appropriate times within a simulation. A role is essentially a collection of nodes, which are added to the agent's basic PPAFCM. To create a simple hierarchy of needs in which the basic needs of agents are more important than those related to a particular role, inhibition links are automatically created from all of the concept nodes in the basic map, to all of the concept nodes in the role's map. Additionally, a role can impose thresholds upon the agent's basic PPAFCM, making sure that these links do not pass on activation through the map until their own activation reaches a sufficient level. This allows a PPA to repeatedly perform the action with the highest activation in their role and ignore basic motivations with low activations as the simulation proceeds. Performing an action will result in reducing the activation of that action node, and so allows other actions to take place.

The main advantage of role passing is that it offers a simple and efficient technique for the control of agents which move competently, and believably, between situations. Not only does this allow believable agent behaviour, it promotes the idea of situational intelligence, whereby issues unrelated to an agent's current situation are not taken into account, thus

reducing its processing load. Role-passing also makes populating a virtual world with agents a straightforward process. Placing agents within novel situations involves simply defining new roles, easing some of the complications involved in attempting to design very general agents. Furthermore designing a role involves simply drawing an FCM, and it is possible that this is more amenable to game designers than writing script or plans.

For demonstration purposes, a simulation of a university campus has been created. As the simulation progresses, a population of virtual humans moves between the different parts of the campus assuming and discarding a range of different roles. Implemented roles include students attending lectures, academics presenting these lectures, people relaxing in the college bar, and a barman serving customers (see Figure 3.9). For more details on the implementation of these roles' FCMs, see Appendix A.

Figure 3.9: University campus simulation using the PPA architecture: (a) Students going to lectures. (b) Academic presenting the lecture. (c) Academic and students going to the pub (d) People socialising in the pub.

3.5.2 The PPA Believability Experiment

The *PPA believability* experiment was carried out by MacNamee [Mac04] to evaluate whether an agent controlled by the PPA architecture continuously behaved in a consistent and believable manner. The experiment was implemented using the ALOHA system and consisted of a bar scene simulation that was populated with a barman character and a number of customer characters. In the experiment, participants were simultaneously shown two versions of this simulation that ran on two separate PCs. In the first version of the simulation, the customer characters were implemented using the PPA architecture, whereby they adopted a *Bar Customer* role which controlled their behaviour. The second simulation was populated with customer characters whose behaviours were randomly chosen from a small set suitable to the bar scene. In both simulations, the behaviour of the barman was controlled by a *Bar Man* role, allowing him to serve customers when they needed a drink.

A total of 13 participants took part in the experiment, and they were allowed to view both simulations simultaneously for as long as they wanted. After the experiment, the participants were asked to fill out a questionnaire asking a range of questions with regards to the believability of the characters' behaviours. When asked "in which they thought agents' behaviour was most believable", 77% of the participants chose the simulation utilizing the PPA architecture. Furthermore, these participants indicated that this was a result of the agents displaying a high degree of consistency (e.g., going to the toilet after they had a couple of drinks). Another factor that added to the believability of the PPA agents was the emergence of groups, whereby agents chose which table to sit at based on their relationship with the other characters at the table. For more details on this believability evaluation, see [Mac04].

No computational evaluation of the ALOHA system has been performed, since the virtual human models are too detailed (i.e., 15,000 triangles) and this limits the number of characters rendered per frame to small groups. For the rendering of large crowds, less detailed models are needed and this is described in the next chapter.

3.6 Summary and Conclusion

The ALOHA system adds an extra degree of believability to simulations by populating them with virtual humans capable of intelligent behaviour. This creates the illusion that they are carrying on with their lives, irrespective of what the user of the simulation is doing. The author's main contribution was in the development of various export tools written in Maxscript, endowing the virtual humans with innate behaviours, and the integration of MacNamee's Proactive Persistent Agent (PPA) architecture. The system's performance suffers once it attempts to simulate large crowds, since the human models are too detailed for this

challenging task. In the next chapter, we will describe the Geopostor system that allows for the rendering of large-scale crowds.

Chapter 4

Geopostors: Crowd Rendering System

This chapter will describe the Geopostor system, a real-time geometry/impostor crowd rendering system, which has been developed to solve the challenging problem of large-scale crowds. The Geopostor system solves this problem by simulating virtual humans as scene extras, equivalent to those found in films. Since these agents are in the background, they are not the focus of the user's attention and therefore simpler animation, rendering and behavioural techniques can be applied to them in order to reduce the computational load of crowded scenes. The Geopostor system is a separate system and utilizes various components of the ALOHA system. A geometric LOD framework is implemented and the PPA architecture is replaced by simpler behavioural techniques to allow for the simulation of large crowds of scene extras.

We first describe the system's LOD framework, including the different LOD geometric models used for our virtual humans and how the most appropriate model is selected. Following this, we describe the integration of our crowd into an urban simulation system. This includes various techniques to improve the realism and performance of the crowd.

4.1 Levels of Detail Framework for Virtual Human Representation

We present our Geopostor system, a real-time geometry/impostor crowd rendering system. Using various techniques, our system is capable of rendering thousands of virtual humans at interactive frame-rates. Our main contribution is that it provides for a hybrid combination of image-based (i.e., impostor) and detailed geometric rendering techniques for virtual humans. By switching between the two representations, based on a *pixel to texel* ratio, our system allows visual quality and performance to be balanced. We improve on existing impostor rendering techniques and present a programmable hardware based method for the lighting of impostors. Furthermore, we improve the realism of the crowd by adding variation to an individual's motion and appearance.

While a deformable mesh was the obvious choice for the virtual human's highest LOD, there are a number of reasons why we chose an impostor approach for the lowest LOD over a continuous and a discrete LOD framework. Firstly, impostors involve replacing a 3D object with an image of the object mapped onto a quadrilateral. This is advantageous mainly because it avoids the cost associated with rendering the object's full geometry. Secondly, automatic tools used to pre-generate low-resolution meshes required for a discrete LOD framework sometimes do not give the required results, thus necessitating a lot of time-consuming editing by hand. Finally, switching between two meshes of different resolutions can be quite noticeable as a result of the silhouettes not matching. A continuous LOD framework utilizing subdivision surfaces offers a good solution to this problem, since the detail of a character can be increased and reduced at run-time, as demonstrated recently by Leeson [Lee02]. While subdivision surfaces provide a means of improving the appearance of virtual humans [OCV+02], they are not suitable for a crowd's lowest geometric LOD representation, since the surface's original polygonal model, used as its starting point, consists of several hundred polygons.

With regards to our impostor model, we decided on a pre-generated approach, since dynamically generating impostors would involve reusing the current dynamically generated image over several frames in order to be efficient. For dynamically generated impostors, the generation of a new impostor image for a virtual human depends on both camera motion and the amount the virtual human's posture has changed. This methods works well with small groups of humans but as the number of virtual humans dramatically increases, numerous new impostor images need to be generated and this produces a bottleneck. Therefore, this method is not well suited for rendering large crowds of dynamic humans.

4.1.1 High Geometric LOD: Deformable Mesh Model

Setup of Deformable Mesh Model

For our virtual human's highest geometric LOD representation (Geometry_{LOD}), we acquired new models as the models previously described in Section 3.3.1 contained too many polygons, produced animation artefacts at the joints, and overall lacked realism. To enhance the realism of our virtual human's appearance, the collection of separate meshes was replaced by a single texture-mapped skinned mesh [Lan98, Lan99]. Each model uses a single detail image of 256*256 pixels and this reduced the number of triangles in the mesh to approximately 2,000. As shown in Figure 4.1, the new texture-mapped models only require 500 triangles to model the head and look more realistic in appearance in comparison to the first set of models. While a deformable mesh is more expensive to animate, the realism of the character's mesh is greatly improved as it is does not suffer from the artefacts caused by using a collection of rigid meshes. To allow for the deformation of the mesh within the ALOHA system, each vertex's influencing bones and their amount of influence (or weight) is assigned in 3D Studio MAX using the *Physique* modifier.

Figure 4.1: Virtual human models in the ALOHA system: (a) Hierarchy of rigid meshes without texture-mapping. (b) Single deformable textured-mapped mesh.

While a continuous LOD framework utilizing sub-division surfaces could be used, we decided on using a discrete LOD framework consisting of a number of different mesh resolutions for the crowd's next LOD representation. A discrete framework involves some additional work in generating several meshes varying in resolutions. However, unlike a continuous framework which generates the specified detail at run-time, the discrete framework only needs to select the appropriate mesh resolution at run-time and does not require any extra computations. Low resolution meshes were generated by hand in 3D Studio MAX using the *Multires* mod-

ifier (see Figure 4.2). Perceptual experiments are described in Section 5.3.3 to establish the resolution at which to generate these meshes so that a user of the system does not detect the reduction of detail in the character's appearance.

Figure 4.2: (a) The original high resolution mesh (100%) and 4 generated low resolution meshes at vertex percentages of (b) 60%, (c) 45%, (d) 30%, and (e) 15%.

To facilitate the introduction of colour and animation variation and to ensure that the pre-generated impostor matches its mesh counterpart, these models required additional setup steps to be implemented in 3D Studio MAX. The mesh's triangles were organised into groups where each group represented a particular body part (as shown in Figure 4.3(b) and (c)) and was assigned a specific pre-defined material. The organising of triangles into groups improves rendering performance since it minimizes OpenGL state changes, as well as facilitating the addition of variation to both the model's appearance and motion which will be discussed later. It should be noted that the diffuse colour of each material is set to white (as shown in Figure 4.3(a)) to allow colour modulation of the pre-generated impostors, which will be discussed later. The meshes in our system use a single image for the detail of the character and this was grey-scaled in 3D Studio MAX to allow colour modulation without losing detail.

Once these additional steps were carried out, the mesh was skinned and a walk animation was created for the underlying skeleton. This key-framed animation was created using Character Studio's footstep creation tool and consisted of a one second, cyclical animation with a key-frame occurring every 100 milliseconds (10Hz). While animations are typical sampled at a minimum of 30Hz, 10Hz was used in the system to reduce the virtual human's memory footprint (see Section 5.1.4). Using the plug-ins previously described in Section 3.3.1,

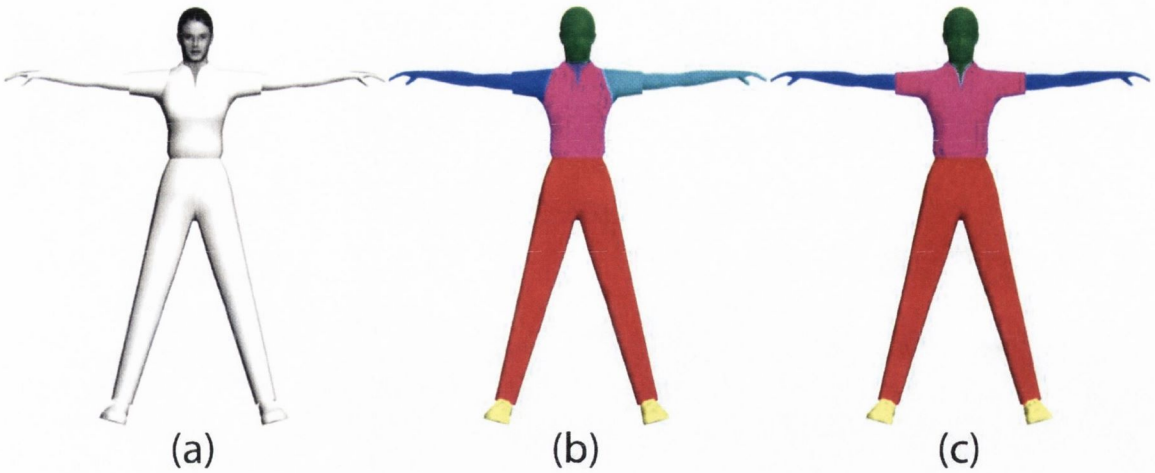


Figure 4.3: (a) High LOD mesh representation in 3D Studio MAX. (b) The grouping of triangles based on material used (shown by the different colours). (c) The grouping of triangles based on the body part it represents (shown by the different colours).

the skeletal, and animation data were exported from 3D Studio Max into their respective ALOHA-specific file formats. A new plug-in was created to export the mesh's skinning data to an *ALOHA Skinned Mesh Format* (ASMF) file which stores the mesh's materials, vertices, vertex normals, texture coordinates, triangles, and each vertex's list of influencing bones and their associated weight.

Rendering the Deformable Mesh Model

As described previously in Section 3.3.1, when rendering a deformable mesh for a virtual human performing a particular task, the skeleton's pose is updated first using the current task's data, and this pose is used to deform the mesh. In our system we optimize our rendering performance by taking advantage of the fact that lowly rated virtual humans perform a default walk. By pre-calculating and storing the deformation of the mesh for a default animation's key-frames, we replace the complex, articulated skeletal model with a simple rigid mesh or "pose" (as shown in Figure 4.4 for the walk animation). This avoids the cost of deforming the mesh, and allows the entire model to be stored relative to a single bone or matrix. To improve the rendering speed of a pose, we encapsulate the pose's data for each of the ten key-frames of the walk animation in a separate vertex buffer object (VBO). Vertex buffer objects allow data to be stored in high performance memory on the server side and therefore increase the rate of data transfer. These VBOs are stored in a geometry cache for reuse in subsequent frames. At run-time, depending on a virtual human's current frame of animation, the corresponding VBO pose is selected and rendered.

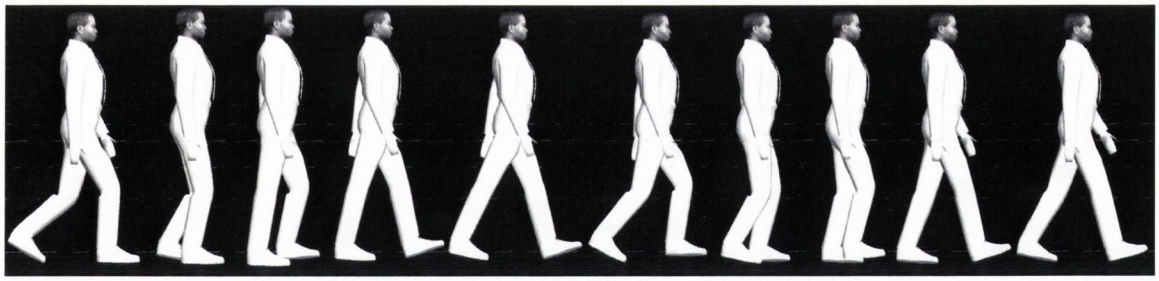


Figure 4.4: Precalculating and storing the deformation of a mesh performing a walk animation for 10 key-frames.

Variation LOD: Adding Variety to the Deformable Mesh's Appearance

At the lowest level of variety (Variation_{LOD}), individuals in a crowd use the same model and are a carbon copy of each other. While this level (or lack) of variety reduces the load on the limited computational resources per frame, this is only suitable for a specific type of crowd without having a disconcerting effect on the viewer e.g., the army of droids in Star Wars: Attack of the Clones. To increase a model's level of variety regarding its appearance, changing the colours of a virtual human's clothing and skin is a method that is simple and yet has high visual impact when viewed in a crowd. In order to do this, we use a set of different template human meshes and change their appearance by using different "outfits". Outfits define a set of colours for the virtual human's skin and clothes, where each colour is associated with a specific body part material. The production of these outfits is controlled entirely by artist-drawn textures produced in an 'Outfit Editor' application, allowing a quick and easy method of producing many different colour maps that are realistic and suitable to the model being rendered. The outfit editor is an OpenGL application that allows the artist to select particular colors for each body material from a colour palette (see Figure 4.5).

In order to add colour variation using the fixed function pipeline, each material used by the mesh is tagged with a specific ID in 3D Studio MAX to define whether it represents skin or a particular type of clothing (e.g., trousers, shirt, or jacket). When the mesh is rendered in the system, the diffuse colour of a material is changed depending on the material's ID and the colour associated with that ID, which is defined by the outfit being used. To allow colour modulation of the texture-mapped triangles, the texture is grey-scaled in 3D Studio MAX. Since the triangles are organised and rendered in groups based on their material properties, this reduces the number of times that the diffuse colour has to be changed to the number of materials used by the triangles. Figure 4.5(b) illustrates three template models with nine different outfits applied to them.

This type of variation can also be implemented using programmable graphics hardware.

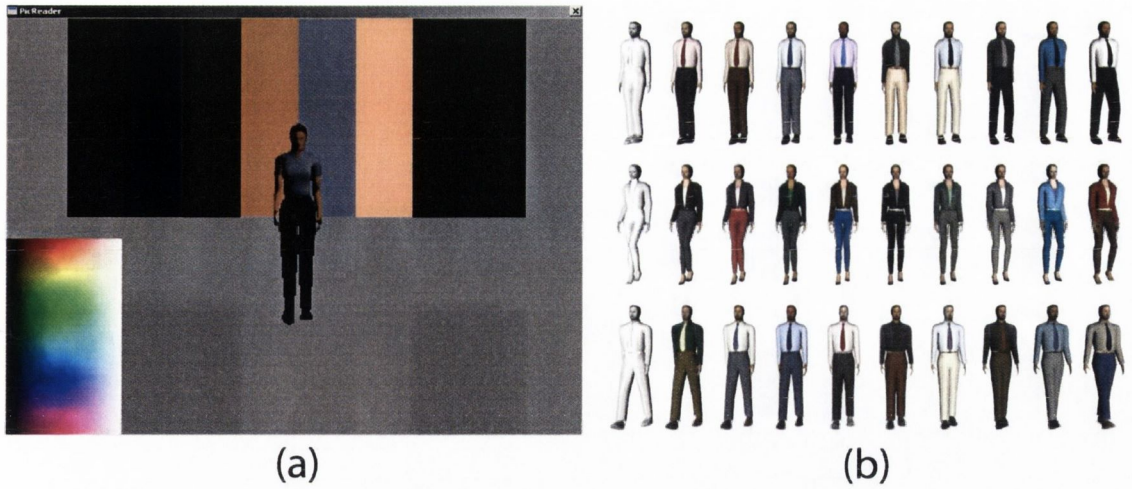


Figure 4.5: (a)Example of an artist in progress of generating an outfit for a model using the 'Outfit Editor' application. (b) Nine outfits for three template meshes.

We achieve this by storing distinct material IDs in the alpha channel of the detail texture, and use these IDs to address a changeable colour map at run-time. While this would not be possible with the fixed function pipeline, programmable texture addressing allows texture indirection to be employed, meaning the output of one texture lookup can be used as the texture coordinates of a subsequent lookup. The main advantage of adding variation through hardware, instead of the fixed function pipeline, is that the diffuse colour does not need to be changed for each material. Although this approach requires an additional texture for the colour map, these textures are one-dimensional and therefore many different outfits can be stored with negligible memory usage.

This has been implemented in hardware using vertex and fragment programs [Gra02b, Gra02a]. In our system, we use a single directional light with materials that have no specular, emission or shininess properties, resulting in a simplification of the OpenGL lighting Equation (4.1). This equation is implemented in hardware (see Equation 4.2), wherein the per-pixel dot products of the light vector and the mesh's normals are multiplied with the per-pixel *coloured regions* to produce the *shaded coloured regions* mesh. The per-pixel coloured regions is a result of the decal texture's alpha-encoded regions looking up the colour map. The shaded coloured regions is added to an ambient term, and multiplied with the detail texture to yield the final lit, coloured pixels. The overall shading and colouring sequence is illustrated in Figure 4.6.

$$
\begin{aligned}
Pixel_{Colour} &= DetailTexture_{RGB} * \\
&\quad (Ambient_{LightModel} * Ambient_{Material} + \\
&\quad (MAX(Vector_{Light} \cdot Normal_{Vertex}), 0) * \\
&\quad (Diffuse_{Light} * Diffuse_{Material}))
\end{aligned}
\tag{4.1}
$$

$$
\begin{aligned}
Pixel_{Colour} &= DetailTexture_{RGB} * \\
&\quad (Ambient_{LightModel} * Ambient_{Material} + \\
&\quad (MAX(Vector_{Light} \cdot Normal_{Vertex}), 0) * \\
&\quad (Colour_{Map}\ [DetailTexture\ _{\alpha}]) * Diffuse_{Light}))
\end{aligned}
\tag{4.2}
$$

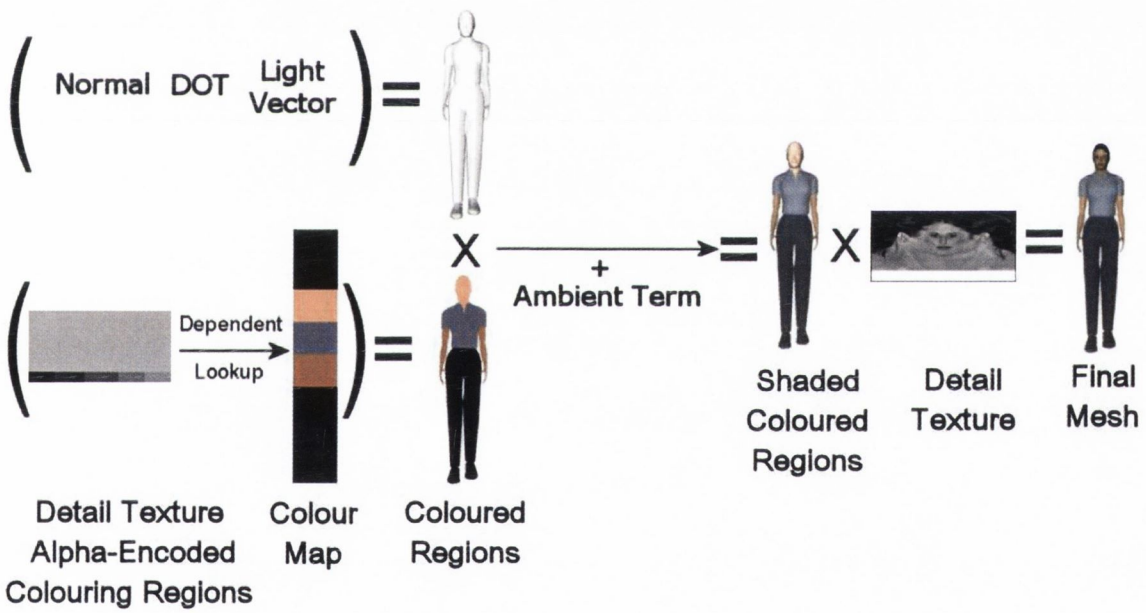


Figure 4.6: Mesh shading and colouring sequence.

Animation LOD: Adding Variety to the Deformable Mesh's Animation

To increase the realism of the simulation, the crowd should not look too uniform in motion. At the lowest Animation_{LOD}, this is achieved by creating default walk animations reflecting the age and gender of each template model. While this achieves variation amongst the different template models, variation amongst virtual humans using the same template model is necessary to avoid them moving in step. The easiest way to increase a template model's animation LOD, is to offset each virtual human's animation by a number of key-frames to achieve a more varied crowd motion. However, this Animation_{LOD} increases the computational burden on resources if each template model can perform several default walk animations. Since each key-framed pose is stored in a separate VBO, this would require several VBO poses per frame.

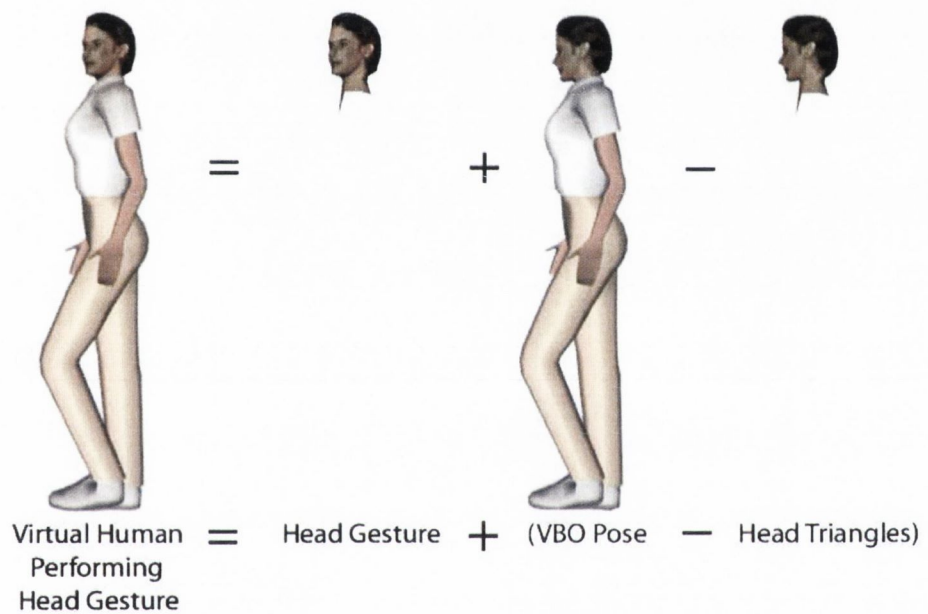


Figure 4.7: Generating a higher level of variety by layering a head animation on top of a default walk animation.

To increase a virtual human's sense of individualism, a higher level of variety for animation is needed. Our approach is to layer particular head and arm gestures on top of the default animations. For example, Figure 4.7 illustrates the sequence for generating a higher level of variety by layering a head animation on top of a default walk animation. In order to animate and render a body-part performing a gesture, the global transformation matrix of its parent bone is pre-calculated and stored for each keyframe of the default animation in the animation cache. Using the parent bone's global transformation matrix for the virtual

human's current frame of animation, coupled with the gesture's current key-frame's data, the global transformation matrix of the bones affecting the body part can be calculated. Once these gesturing bones have been updated, the deformation of the vertices and normals are calculated and the body part is rendered with the current mesh pose. Since the triangles of a virtual human's mesh are also batched based on the body part it represents, the pose can easily be rendered without a particular body part. The main advantage of this approach is that it avoids the cost of deforming the entire virtual human mesh, since the pre-calculated pose of the default animation is re-used and only the position of the vertices affected by the gesturing bones need to be deformed.

Table 4.1 and Table 4.2 respectively summarizes the Animation_{LOD} and Variation_{LOD} that can be employed for a crowd's deformable mesh representation.

$\text{LOD}_{Animation} = 0$	$\text{LOD}_{Animation} = 1$	$\text{LOD}_{Animation} = 2$
Precalculated VBO poses using a default animation	Precalculated VBO poses using a default animation with layered gestures	Deform mesh based on a specific skeletal animation

Table 4.1: The different levels of detail for animation that can be used for a crowd's deformable mesh representation (Geometry_{LOD} is HIGH).

$\text{LOD}_{Variation} = 0$	$\text{LOD}_{Variation} = 1$	$\text{LOD}_{Variation} = 2$	$\text{LOD}_{Variation} = 3$
1 template mesh 1 outfit Uniform crowd motion	1 template mesh N outfits Uniform crowd motion	1 template mesh N outfits Varied crowd motion	M template meshes N outfits per model Varied crowd motion

Table 4.2: The different levels of detail for variation that can be used for the crowd's deformable mesh representation (Geometry_{LOD} is HIGH).

4.1.2 Low Geometric LOD: Pre-generated Impostor Model

Generation of the Impostor Images

For our virtual human's lowest LOD representation, we use pre-generated impostors based on the work of Tecchia et al. [TC00a]. The same template mesh models previously described in Section 4.1.1 were used in the pre-generation of the necessary impostor images in 3D Studio MAX. With regards to the default walk animation, it is important that both the

mesh model and the motion are symmetrical in order to minimize the amount of texture memory the impostor images consume. This halves the number of viewpoints from which the model needs to be rendered, since a viewpoint image for a particular key-frame can be mirrored to obtain the opposite viewpoint for the corresponding symmetrical key-frame. Figure 4.4 illustrates a walk animation, where there is a difference of five key-frames between each pair of symmetrical key-frames. In the case of asymmetric animation, such as a side-step left or right motion, impostor images need to be generated around both sides of the model, doubling the amount of memory consumed. However, the impostor images only need to be generated for a side-step left motion since it can be mirrored to obtain a side-step right motion. Additionally, a side-step motion is typically short in duration (e.g., 0.5 seconds) and therefore less key-frames are needed.

A MaxScript plug-in was written to render the images needed by the impostor representation in 3D Studio Max. The process used is illustrated in Figure 4.8. The plug-in positions the virtual human mesh model at the center of a sphere consisting of 32 segments and a radius equal to the distance from which we wish to render the impostor images. For 10 frames of animation, a detail map image and a normal map image are rendered from 17 viewpoints around one side of the model and from 8 elevations:

- **Impostor detail map**
  This image is used to store the detail of the mesh's decal texture for each viewpoint. It is generated by rendering the mesh, with shading and anti-aliasing disabled, into an image of 256×256 pixels. To allow for variation, each pixel in the image's alpha channel needs to be encoded with a specific alpha value associated with the material at that particular pixel. In order to do this, the plug-in utilizes 3D Studio Max's Graphics buffer or *G-buffer* which allows data such as object ID, material ID, and UV coordinates to be stored in a number of separate channels. The plug-in stores the material ID at each pixel in the G-buffer and these values are used to lookup and store the associated alpha value in the alpha-channel. Background pixels are assigned an alpha value of 255 to distinguish which pixels need to be transparent when displaying the impostor at run-time.

- **Impostor normal map**
  This image is used to store the mesh's surface normals in eye-space for each pixel in the detail map. We assign barycentric texture coordinates to each triangle's vertices to provide an easy way to interpolate the normal at a specific point on a triangle. For each pixel, the triangle's ID (T_{id}) and its interpolated texture coordinate (u_i,v_i) at that pixel are stored in the *G-buffer*. Using the triangle's vertex normals ($\overrightarrow{NV_1}$, $\overrightarrow{NV_2}$ and $\overrightarrow{NV_3}$

which are accessed using T_{id}), the interpolated normal $\overrightarrow{N}$ at that pixel is calculated using Equation 4.3 and converted into a RGB colour ($Pixel_{RGB}$), using Equation 4.4.

$$\overrightarrow{N} = (1 - u_i - v_i)\overrightarrow{NV_1} + u_i\overrightarrow{NV_2} + v_i\overrightarrow{NV_2} \tag{4.3}$$

$$\begin{aligned} Pixel_R &= ((0.5 * N_x) + 0.5) * 255 \\ Pixel_G &= ((0.5 * N_y) + 0.5) * 255 \\ Pixel_B &= ((0.5 * N_z) + 0.5) * 255 \end{aligned} \tag{4.4}$$

Once these images have been generated, the plug-in removes any unused space and combines them into a single detail and normal map image of 1024*1024 pixels for a particular frame of animation. For each frame of animation impostor image, the data needed to render each viewpoint at run-time is stored in a text-based Impostor Data File (IDF). This file includes each viewpoint's row and column ID, position, width, height, and position of the parent bone of the model's skeleton within the image.

Rendering of the Impostor Model

The main problem with using a pre-generated impostor approach is the consumption of texture memory. In order to render a dynamically lit impostor, an impostor detail image and a normal map image are required for each frame of animation. The RGBA impostor detail image contains four channels (1024*1024*4 bytes) and the RGB normal map image contains three channels (1024*1024*3 bytes), resulting in 7MB of texture memory being required for a single frame of animation. By using DXT3 texture compression, the memory requirements are reduced by a factor of four for RGBA images and by a factor of six for RGB images, resulting in only 1.5MB (1024*1024*4*1/4 + 1024*1024*3*1/6 bytes) of texture memory for each frame. Unfortunately, DXT3 texture compression is not particularly effective at compressing normal maps, as it results in noticeable block artefacts. These artefacts can be avoided by using 3Dc, which is ATI's new compression technology, and provides 4:1 compression of normal maps with image quality that is virtually indistinguishable from the uncompressed version [Pap].

Our impostor representation is capable of using *mipmapping* techniques [Wil83]. Mipmapping avoids visual artefacts that occur when textures are mapped onto smaller dynamic objects, causing them to shimmer. OpenGL allows the generation of a series of pre-filtered

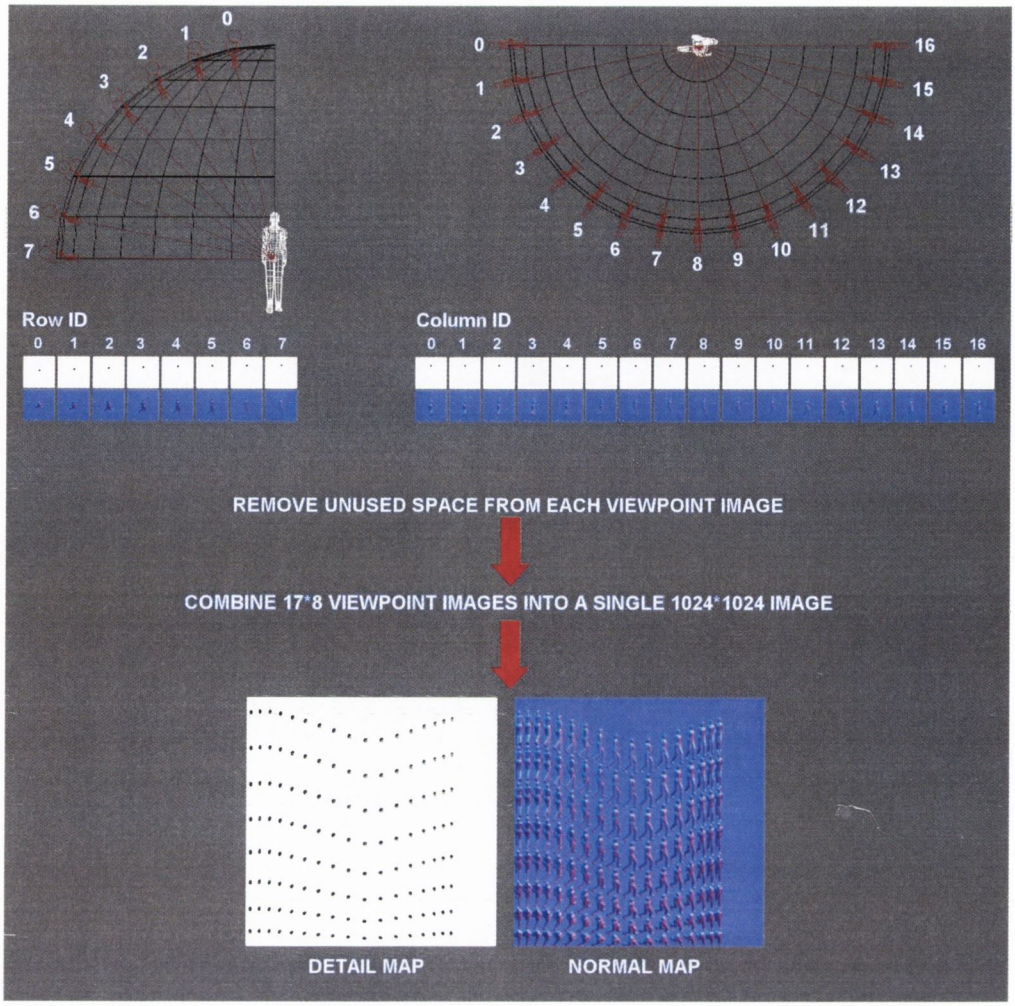


Figure 4.8: A MaxScript plug-in removes unused space from each viewpoint image and combines 17*8 viewpoint images into a single 1024x1024 image for a particular frame.

texture maps of decreasing resolutions, called *mipmaps*, which are selected based on the size (in pixels) of the object being mapped. Although mipmapping requires some extra computation and texture storage (which is increased by a third), this is necessary to maintain the impostor's realism when displayed at a distance. However, care has to be taken not to generate mipmaps at too low a resolution, as this causes other artefacts due to the averaging of several viewpoint images within the mipmap.

Given the amount of texture memory required by the system, we need a method of improving the variety and visual interest of large crowds of impostors, while keeping memory usage to a minimum and ensuring that rendering speed is uncompromised. Our contribution in this area is that we improve upon existing impostor techniques for adding variety by taking

advantage of recent improvements in programmable graphics hardware in order to perform an arbitrary number of colour changes in one pass. Since the colouring regions are encoded in the alpha channel (as described in Section 4.1.2), this number is limited only by that channel's precision. Our further contribution is the real-time shading of the impostors implemented in programmable hardware.

To render the impostors, we need to calculate which viewpoint image needs to be displayed and rotate its quadrilateral so that it always faces the viewer. Using the position of the virtual human's root bone $\overrightarrow{H}$ and the camera's position $\overrightarrow{C}$, the quadrilateral's normal vector $\overrightarrow{N}$ can be calculated using Equation 4.5. The vector from the camera to the human projected onto the ground plane $\overrightarrow{CH}$ can be calculated (Equation 4.6) using $\overrightarrow{N}$. It should be noted that in Equation 4.5, it is assumed that the ground is the XZ plane and that the camera's position cannot be lower than the ground. Therefore, it is not necessary to pre-generate any viewpoint images from these elevations.

$$\overrightarrow{N} = \frac{\overrightarrow{H} - \overrightarrow{C}}{|\overrightarrow{H} - \overrightarrow{C}|} \tag{4.5}$$

$$\overrightarrow{CH} = \frac{(N_x, 0, N_z)}{|(N_x, 0, N_z)|} \tag{4.6}$$

The amount by which to rotate the quadrilateral around the x-axis θ_x and y-axis θ_y is calculated using Equation 4.7. The viewpoint's row and column ID (V_{Row} and V_{Column}) can be used to lookup which viewpoint to render using Equation 4.8, where N_x and N_y are the number of viewpoint images pre-generated around the x- and y-axis.

$$\begin{aligned} \theta_x &= \cos^{-1}(N_y) \\ \theta_y &= \cos^{-1}(CH_z) \end{aligned} \tag{4.7}$$

$$\begin{aligned} V_{Row} &= \theta_x \times \frac{N_x}{90} \\ V_{Column} &= \theta_y \times \frac{N_y}{180} \end{aligned} \tag{4.8}$$

For improving realism, interactive lighting of impostors is highly desirable. Additionally, since we are presenting a hybrid system that switches between two representations, it is crucial that there is no difference in the shading of each representation for the interchange to be imperceptible to the viewer. By using a per-pixel dot product between the light vector and a normal map image, Tecchia et al. [TLC02b] computed the final shaded value of a pixel through multi-pass rendering, which required a minimum of five rendering passes. However, multi-pass rendering can have a detrimental effect on rendering time, which limits the number of impostors that can be shaded in real-time.

We improve upon this technique by taking advantage of programmable graphics hardware and shade the impostors in a single pass. The impostors are rendered with the same lighting and material properties as the mesh representation in Section 4.1.1, and thus the shading of the impostor is based on Equation 4.1. Similar to the mesh representation, the lighting of the impostor representation has been implemented in hardware using both texture shaders and register combiners [Gra99a], and vertex and fragment programs [Gra02b, Gra02a]. This involves implementing Equation (4.9) in hardware, whereby the per-pixel dot products of the light vector and the pre-generated normal map is multiplied with each pixel in the coloured region map (which will be discussed in the next section) to produce a shaded coloured region map. This result is added to an ambient term, and multiplied with the detail map to yield the final lit, coloured pixels. The overall shading and colouring sequence is illustrated in Figure 4.9.

$$\begin{aligned} Pixel_{Colour} = \; & DetailMap_{RGB} * \\ & (Ambient_{LightModel} * Ambient_{Material} + \\ & (MAX((Vector_{Light} \cdot NormalMap_{RGB}, 0) * \\ & (ColourMap\ [DetailMap\ _{\alpha}]) * Diffuse_{Light})) \end{aligned} \tag{4.9}$$

Similar to the mesh model, we optimise the rendering of the impostors by precalculating and storing each of the key-frame's viewpoint data in a single VBO object. Since dynamically orientating the quad involves the computationally expensive $\cos^{-1}$ function (see Equation 4.5), we use a lookup-table (LUT) of $\cos^{-1}$ values instead. A LUT is typically an array used to replace a run-time computation with a simpler lookup operation and can provide a significant speed gain.

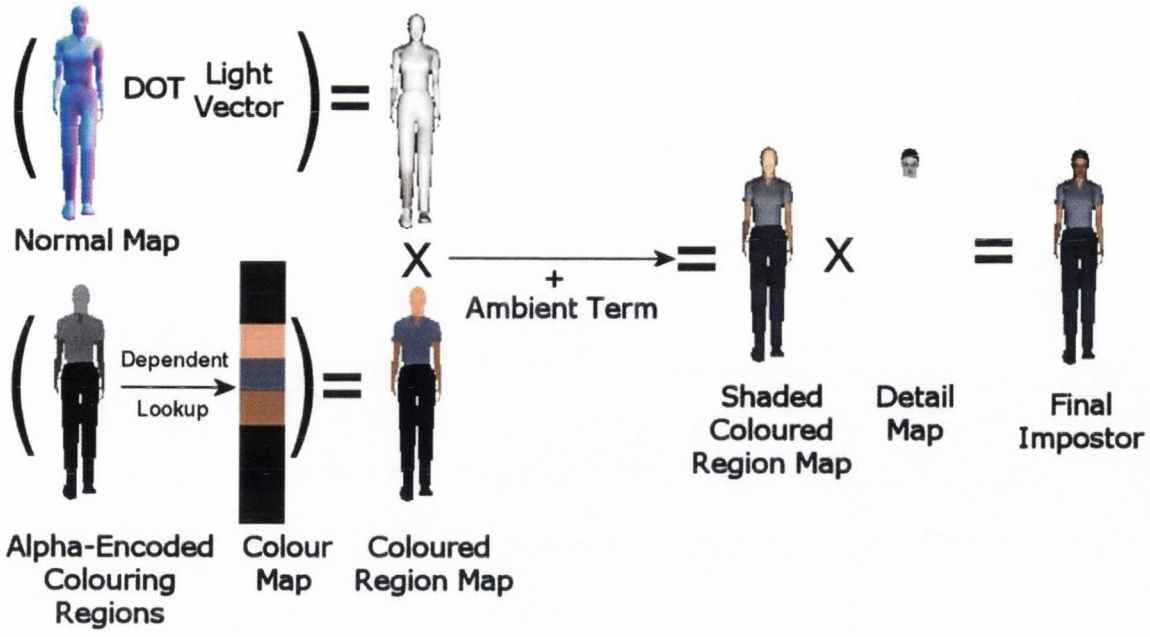


Figure 4.9: Impostor shading and colouring sequence.

Variation LOD: Adding Variety to the Impostor Model's Appearance

To improve the variety of the impostor, the same approach of changing the colour of skin and clothes was used. A multi-pass method, as described in [TLC02b], achieves this goal by performing a rendering pass for every different region of colour that needs to be changed. We exploit the programmability of graphics hardware to efficiently increase the variety and interest of each impostor. In order to match the virtual human's geometric representation, the impostors must also be able to change colour, depending on the human model and outfit materials. We achieve this by storing distinct material IDs in the alpha channel of the impostor detail image upon generation, and use these IDs to address a changeable colour map at run-time. We perform a lookup on the detail map, using the alpha-encoded material IDs to address a colour map texture that can be altered to match the outfit of the virtual human currently being rendered (Figure 4.10). It should be noted that, since the alpha channel of the impostor's detail map contains alpha encoded regions, nearest filtering needs to be used. Otherwise, linear filtering results in the linear interpolation of these values when the impostor representation is at a distance, causing shading artefacts due to the wrong outfit colour being looked up. However, nearest filtering can cause aliasing artefacts between pixels encoded with differing alpha values when the impostor is at a distance. This problem can be solved by using a high-level shader written in the OpenGL shading language to linear filter

the looked up color values [Gui05].

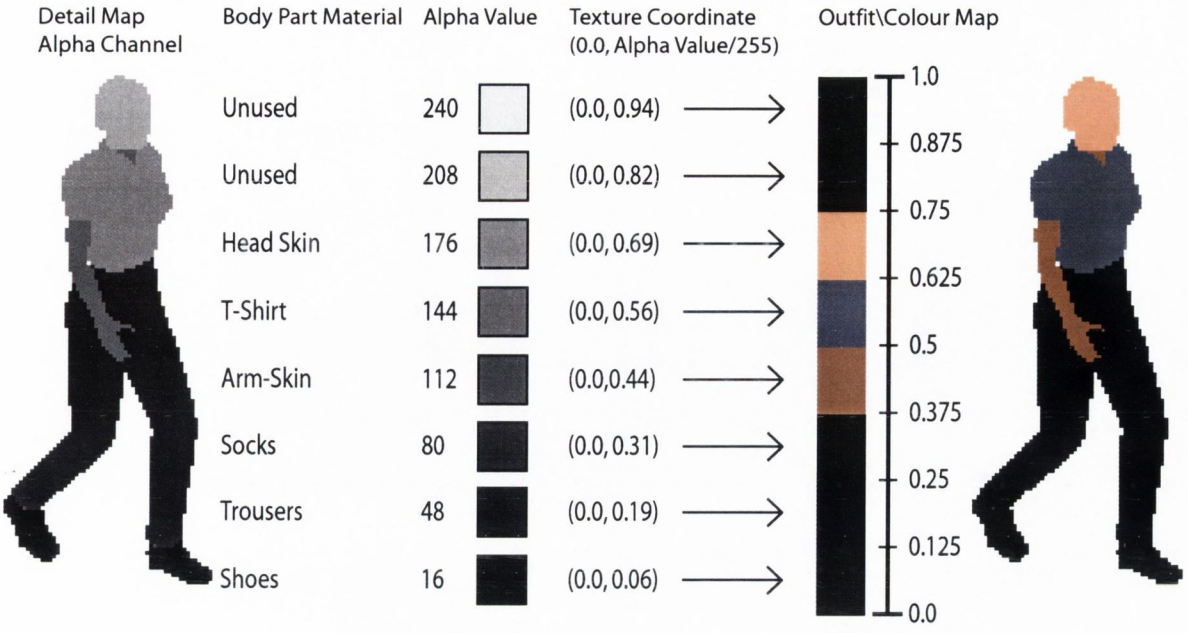


Figure 4.10: Using programmable texture addressing to add variety to the impostor representation.

Animation LOD: Adding Variety to the Impostor Model's Animation

Similar to the mesh model, we add variety to the animation at a lower level of detail by pre-generating the template model's impostor images using the same default animations, that can reflect the age and gender of the model. To avoid the impostors moving in step, the same method previously described in Section 4.1.1 is used, whereby each virtual human's animation is offset by a particular number of frames to achieve a more varied crowd motion. However, since each animation key-frame is stored in a separate texture, this type of variation is limited depending on the number of textures needed in a single frame.

Increasing an impostor representation's sense of individualism is a tricky problem, since it is limited to the animation used in the pre-generation of its images. Similar to the mesh model, we solve this problem by layering head and arm gestures on top of the default impostor animation. However, the main difference is that a particular body-part in the impostor image is replaced with a gesturing mesh representing the body-part. Since each body-part of the impostor is represented by a particular alpha value in the detail image's alpha channel, the impostor can be rendered without these body-parts by changing the alpha function accordingly. Using the method previously described in Section 4.1.1, the gesturing bones are

updated, and the affected part of the mesh is deformed and rendered. The main advantage of this approach is that it avoids the cost of deforming and rendering the entire mesh by replacing it with the impostor representation. Thus, only the triangles affected by the gesturing bones need to be rendered. While minor rendering artefacts can appear caused by the layering of the mesh on top of the impostor, these can be removed through blending.

The problem with this method is that, depending on the viewpoint being displayed, holes appear when a body part is not rendered since the body part may sometimes be occluding other areas of the impostor. When the virtual human performs a head gesture this artefact is not as much of a problem as when they are performing an arm gesture. Currently, virtual humans that are rendered with an impostor representation switch to a low resolution mesh representation when they request an arm animation. As a possible solution, dynamically generated impostors could be used to render the virtual human's body without its arms and this will be investigated in future work.

Table 4.1 and Table 4.2 respectively summarizes the Animation_{LOD} and Variation_{LOD} that can be employed for a crowd's impostor representation.

$\text{LOD}_{Animation} = 0$	$\text{LOD}_{Animation} = 1$	$\text{LOD}_{Animation} = 2$
Pre-generated textures using 1 default animation	Pre-generated textures using 1 default animation with layered gestures	Pre-generated textures using N default animations

Table 4.3: The different levels of detail for animation that can be used for a impostor representation (Geometry_{LOD} is LOW).

$\text{LOD}_{Variation} = 0$	$\text{LOD}_{Variation} = 1$	$\text{LOD}_{Variation} = 2$	$\text{LOD}_{Variation} = 3$
1 template impostor 1 outfit Uniform crowd motion	1 template impostor N outfits Uniform crowd motion	1 template mesh N outfits Varied crowd motion	M template impostor N outfits per model Varied crowd motion

Table 4.4: The different levels of detail for variation that can be used crowd's impostor representation (Geometry_{LOD} is LOW).

Short-Comings of the Pre-Generated Impostor Representation

While the impostor used in the Geopostor system is computationally efficient to render, the following short-comings are associated with this representation:

- Anti-Aliasing: Since the impostors are not rendered without anti-aliasing, this results in the silhouette being pixellated in appearance and is especially noticeable when the impostor is close to the viewer. Future work will investigate how anti-aliasing techniques would improve the impostor's visual appeal.

- Models and animations need to be symmetric: To reduce the number of viewpoint images needed, both the model and animation have to be symmetric in the XZ plane. If this is not possible then the impostor's texture will consume twice as much memory in order to fit the additional viewpoint images that are needed.

- No viewpoint images generated from directly above or below the ground-plane: No viewpoint images were generated from directly above the virtual human model or from below the ground-plane, resulting in parallax artefacts when the impostor is viewed from these camera angles. However, these viewpoints were not needed since the camera is not allowed to move below the ground plane in the city simulation system described in Section 4.2. The number of viewpoint images needed depends on what camera angles the impostors will be viewed from and this should be considered when generating the impostor's textures to minimize memory consumption.

- Pixellated shadows when the sun is low in the sky: Since the impostor texture are used in projecting ground-plane shadows (see Section 4.15), this results in the shadows being pixellated when the sun is low in the sky and is especially noticeable when the shadows are close to the viewer. In this case, the virtual human's mesh representation should be used in the projection of the shadow.

4.1.3 Switching Between Representations

The main aesthetic problem with using an impostor to represent a virtual human is that, once the human is close to the viewpoint, the impostor's flat and pixellated appearance becomes quite obvious (shown in Figure 4.11 (a)). As noted by Ulicny et al. [UdHCT04], while impostors may be a good approach for far-away humans, this problem means that they are not suited when more detailed views are necessary. However, our main contribution is to allow a virtual human to switch from an impostor to a higher LOD representation, based on some selection criteria, thus greatly improving the realism of the simulation.

Our approach is to switch between an impostor and the mesh that was used to generate the impostor at a specific distance (d_{switch}) (shown in Figure 4.11 (b)). At run-time, if the virtual human is within the view-frustum and the distance of the human from the viewer is less than d_{switch}, the system switches between the two representations. In order to achieve

a seamless transition between the representations, animation popping artefacts need to be eliminated. We use the virtual human's current animation frame to select the corresponding mesh pose or impostor texture depending on what representation we are switching to.

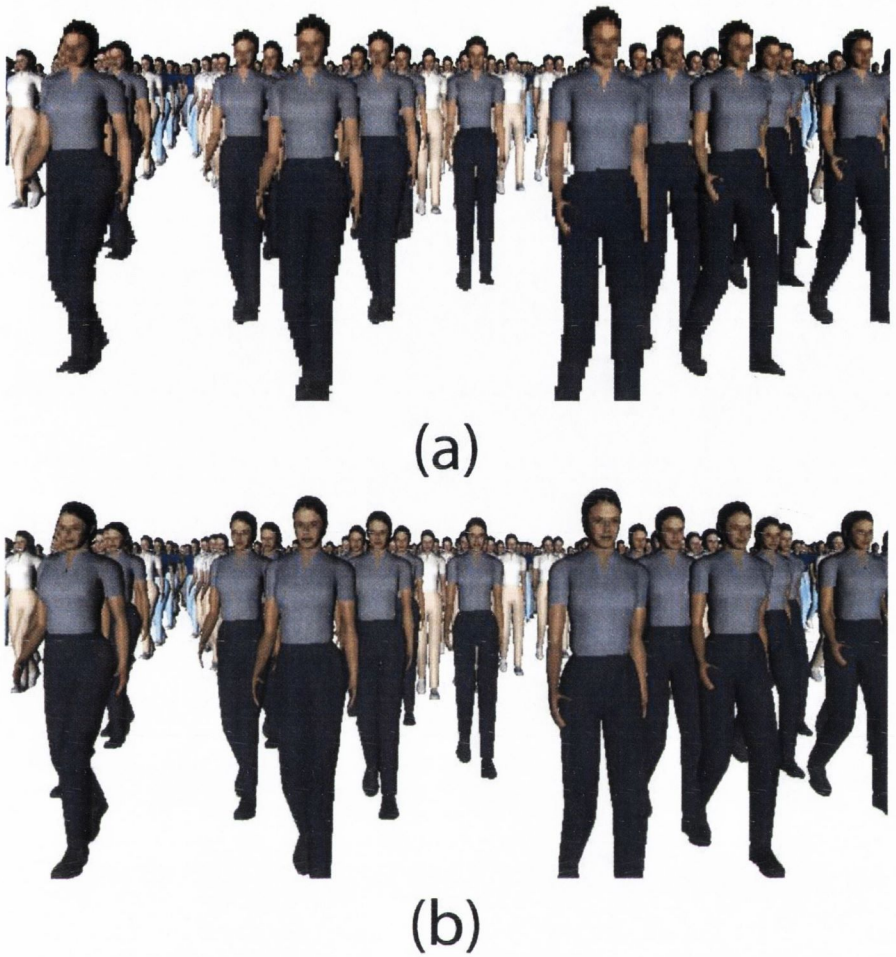


Figure 4.11: (a) Problem of an impostor's appearance being flat and pixellated when it is near to the viewer. (b) Problem solved by switching to a mesh representation at an appropriate distance.

We propose that the switch between representations should happen at a distance where the ratio of the screen size of an impostor quadrilateral to the size of the viewpoint image equals a certain threshold. We can calculate the size of both the quadrilateral and its viewpoint image in terms of the impostor's pixel size and texel size respectively. A texel (TEXtured ELement) is the basic unit of measurement when dealing with texture mapped 3-D objects. When a texture map is loaded into texture memory as an array of n×m texture elements, each element is referred to as a texel. When a 3-D texture-mapped object appears close to

the viewer so that the texture elements appear relatively large, there may be several pixels in each texel and the pattern of the texture map is easy to see. Therefore, the ideal pixel to texel ratio at which the switching should happen is at 1:1, as aliasing starts to occur when the size of the impostor's quadrilateral is greater than the size of the texture-mapped image, resulting in the stretching of the image on the impostor's quadrilateral. Perceptual experiments were carried out to test whether people could detect these representations switching at this ratio (see Section 5.3).

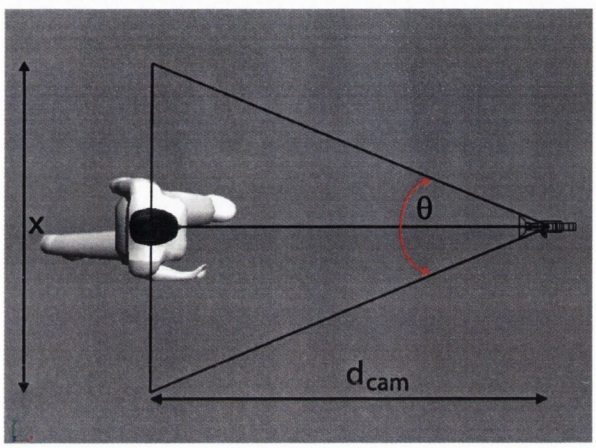


Figure 4.12: Calculating impostor image pixel size.

Given the set of viewing parameters used in the generation of the impostor images in 3D Studio MAX, we can compute the size of an impostor image's texel (Texel_{size}) as shown in Figure 4.12. In this figure, θ is the camera's field of view, d_{cam} is the distance along the viewing direction from the camera to the mesh, and x is the resolution of the impostor viewpoint image. Using these values, the size of an impostor image's texel can be calculated using Equation (4.10). Given the set of viewing parameters of the system and using the properties of similar triangles, the distance d_{switch} at which the pixel to texel ratio is equal to one, can be calculated using Equation 4.11 and Equation 4.12. Note that θ is the camera's field of view, $\text{d}_{nearplane}$ is the distance along the viewing direction from the camera to the near plane, and x is the resolution of the screen in pixels.

$$Texel_{Size} = \frac{tan(\frac{\theta}{2}) * 2 * d_{cam}}{x} \tag{4.10}$$

$$Pixel_{Size} = \frac{(2 * d_{nearplane} * tan(\frac{\theta}{2}))}{x} \tag{4.11}$$

$$d_{switch} = \frac{d_{nearplane} * Texel_{Size}}{Pixel_{Size}} \tag{4.12}$$

4.2 Integration of Geopostors into an Urban Simulation Environment

To examine the performance and usability of our crowd system, we incorporated it into the Virtual Dublin project, an existing virtual city simulation [HO03]. This is a highly detailed model of Dublin city centre, covering several square kilometres, implemented as an OpenGL first person perspective application (see Figure 4.13). The Virtual Dublin project aims to create an interactive model of the city of Dublin, and at the same time create a generic urban simulation engine which could be applicable to other cities.

The initial system was based on a CAD model of the grounds of Trinity College Dublin, which was converted to operate in a real-time application and then expanded to encompass the surrounding areas of the city. The system has been under development for four years, and has involved many undergraduate students in a data-acquisition capacity. The project has also served as a test-bed for mobile device research and the large scale crowd research presented here.

The system provides an OpenGL application allowing the user full mobility about the environment. Highly detailed models of landmarks and buildings are used to provide an immersive environment similar to that found in contemporary video games. The system's realism is enhanced by weather and time-of-day effects, and provides a level-of-detail based rendering of the environment that allows interactive frame rates to be maintained in all areas of the city. The system uses both culling techniques and impostor rendering techniques to maintain interactivity. Texture compression is also used extensively to reduce memory load and prevent thrashing due to page faults. Further details about the project and system may be found in [Ham05].

4.2.1 Virtual Human Pedestrian Behaviour

Steering Behaviour: Obstacle Avoidance

To control each virtual human's movement in the world, we use Reynold's approach where each human is simulated as a simple vehicle model whose properties are based on a point mass approximation [Rey99]. This model allows a very simple and computationally cheap physically-based model and for this reason it is well suited to controlling the movement of a

Figure 4.13: Virtual Dublin screen shots.

large number of virtual humans. Using this model, specific steering behaviours can be defined for the virtual human.

In an obstacle-free world, the new position of a walking virtual human (P') at each time-step can easily be calculated using its current position (P), its steering vector ($\overrightarrow{S}$) and the distance travelled (d) between its current keyframe and the last keyframe of its walk animation (see Equation 4.13).

$$P' = P + \overrightarrow{S} * d \tag{4.13}$$

To avoid a virtual human walking in the same direction, we can rotate its steering vector at every time-step by certain amount, thus generating a random steering or *wandering* behaviour. However, this results in the directional vector oscillating and the virtual human appears to be unable to decide on where it is going. In order to make the wandering behaviour more believable, the directional vector is not changed every time-step but only every 200 milliseconds and this results in the virtual human wandering around its environment in a more coherent manner. Additionally, by constraining the maximum amount the steering vec-

tor can be rotated to $\pm$ 2.8125 degrees, the virtual human does not perform large unrealistic directional changes while it walks.

Urban environments are far from being obstacle-free. Therefore, virtual humans wandering around this type of environment without obstacle avoidance would result in them walking through objects such as buildings and the overall realism of the simulation would be greatly reduced. Typically, obstacle avoidance involves the virtual human checking whether its steering vector intersects an object and, if so, it steers in some way to avoid it. While the performance of this method can be improved by checking only those objects within the same node as the virtual human, this method would become computationally demanding as the number of humans and objects in each node increases. Given that we want to populate an environment full of static obstacles with a large number of virtual humans, the method of obstacle avoidance must be efficient enough so as not to unduly impact on performance.

Due to the nature of the environment (a city filled with vertically rising structures), we perform collision detection in 2D using a top-down map of walkable areas similar to that described in [TLC02b] and [LMM03]. We pre-generate a set of maps of walkable areas or *walk-maps* by capturing an orthographic top-down view of the city model in the OpenGL depth buffer. We store the city's pavements (including tunnels through buildings and pavements occluded by over-passing bridges) as walkable, and buildings, roads, and other obstacles as blocked, by selecting appropriate near and far planes when rendering the city model into the depth buffer. Each map is 800×600 pixels, where a pixel is white if it is walkable otherwise black if it is blocked (Figure 4.14) and corresponds to a physical area of 33×33cm. These 1 bit data (walkable or blocked) maps require 60KB of memory and sixty-three walk-maps for the 4.5km^2 area covered by the virtual city were pre-generated, requiring a total of 3.6MB of memory for the city.

To perform obstacle avoidance, we perform a simple lookup on the walk-map to determine if the virtual human's newly calculated position is walkable. If it is, the human moves to its new position. However, if it is blocked, the human needs to steer away. The simplest way is to allow the virtual human to perform an *about turn* by rotating its steering vector by 180 degrees, but this provides unrealistic results.

Our approach is to allow the virtual human to check whether the positions to the left and right of its new blocked position are free and it uses this information to either:

- Randomly choose which direction to rotate its steering vector if both positions are walkable
- Rotate its steering vector to the right if it is walkable
- Rotate its steering vector to the left if it is walkable

- Perform an about-turn if both positions are blocked

While this provides a more realistic collision avoidance behaviour, where the human seldomly needs to perform the unrealistic about turn, the problem with this approach is that it only takes into account what is free ahead of the virtual human, which can result in the human walking very close to and sometimes through obstacles that are to the side of it. Since the edges of buildings and roads are linear in nature, we can improve the realism of the obstacle avoidance by making sure the humans walk along these edges without getting too close to them. We implement this by looking up the positions to the left and right of its current position every second. If one of these positions is blocked, the human steers away from this area by rotating its steering vector in the opposite direction.

Another type of map that we use for obstacle avoidance is a *potential field map.* The idea behind potential fields is that each obstacle is considered to have a repulsion force whose strength is inversely proportional to the distance from it. We pre-generate these maps for the buildings, roads and other obstacles in the environment by calculating the potential field force in the x-z plane at every walkable pixel in the walk map. These maps are stored as 24-bit data, thus requiring 1.4MB of memory per map and 88.2MB for the city.

For each walkable pixel, the total force at that pixel ($\sum \vec{F}$) is calculated by summing the repulsion forces of any blocked pixels in a 7×7 neighbourhood affecting the walkable pixel. The summed force's normalised x and z component $\vec{F}_x$ and $\vec{F}_z$ are mapped to a value between 0 and 255 and stored in the red and green component, respectively, of the potential field map (using Equation 4.14). To store the magnitude of the summed forces $|\sum \vec{F}|$, the maximum magnitude of a force $|\vec{F}_{MAX}|$ exerted on a pixel needs to be calculated by summing all of the forces exerted on a walkable pixel inhabiting a completely blocked neighbourhood. Using Equation 4.14, $|\sum \vec{F}|$ is mapped between 0 and 255 and finally stored in the blue component of the potential field map.

$$\begin{aligned} R &= ((\vec{F}_x * 0.5) + 0.5) * 255 \\ G &= ((\vec{F}_z * 0.5) + 0.5) * 255 \\ B &= \frac{|\sum \vec{F}|}{|\vec{F}_{MAX}|} * 255 \end{aligned} \tag{4.14}$$

Using these maps, the virtual human steers itself away from the obstacles by calculating the force, if any, at its newly calculated position and uses this to rotate its steering vector. The main advantage of potential fields maps is that they prevent the human walking too

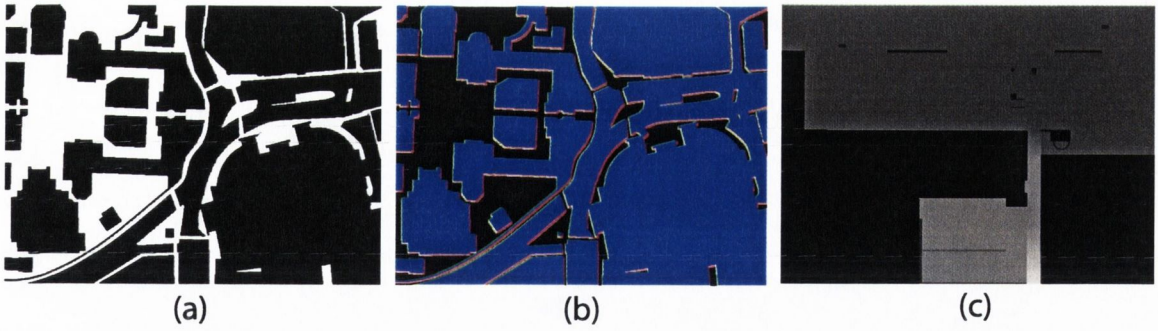


Figure 4.14: (a) Walk map. (b) Potential field map. (c) Height map.

close to obstacles to the side of it, since the obstacle's force field will push it away. However, the main problem is that these maps require a lot of memory and thrashing occurs due to the continuous paging in and paging out of this memory as a result of the camera moving around the environment. While the size of a potential field map can be reduced to 0.9MB by storing only the direction of the force and not its magnitude, the city still requires a total of 60.4MB of memory. Our solution to minimize this thrashing is to use a LOD approach, where the more simple walk-maps are used for the majority of the city and the potential field maps are only used for selected areas that are considered more important to the simulation.

Finally, in order to position the virtual humans so that they do not intersect with the ground, we use a *height-field map*. We pre-generate these images in a similar manner to the walk-maps, but the near and far planes are appropriately selected so that the height of the walkable areas are mapped to a value between 0 and 255 in the depth buffer, thus requiring 180KB of memory per map. By looking up this height value at its new position, the virtual human can use this to adjust its height above the ground plane. It should be noted that the ground is at the same height throughout the city and therefore height-field maps are not necessary. However, these maps have been successfully implemented with another virtual environment where the ground consisted of several planes at different heights connected by ramps.

Steering Behaviour: Group Formation

In the real world, while a large number of humans walk by themselves as they carry on with their everyday lives, it is also common for humans to form and walk in groups while conversing. To enhance the realism of a crowd populating a virtual city, virtual humans need to form groups and interact with each other. In our system, we allow virtual humans to form groups of two and we base our approach on the research by Reynolds [Rey99] on three steering behaviours related to groups of characters: cohesion, separation and alignment. We

use these behaviours to determine how a couple should react with each other when moving through the environment, while still avoiding static obstacles.

The cohesion steering behaviour allows two virtual humans to form a group and works by steering the couple towards their average position. By finding the couple's average position, each of the virtual human's steering vector is rotated towards that average position. The separation steering behaviour prevents the couple getting too close to each other. Based on their average position, if each virtual human is within a certain distance of each other, their steering vector is rotated away from this position. Finally, the alignment steering behaviour allows the couple to align their direction with each other. To align their direction, each virtual human's steering vector is rotated in the direction of the couple's average steering vector.

Gaze Behaviour

It is important for virtual humans to perceive the world they inhabit through virtual sensors. Various techniques related to the simulation of visual perception exist for small numbers of virtual humans but these techniques are too computationally intensive to use with large-scale crowds. However, depending on the virtual human's position with respect to the camera, they only need to act as if they are perceiving their virtual environment and a simpler, less expensive, approach can be used. We implement this by using the same approach described in Section 4.1.1, where we randomly generate a key-frame animation for the virtual human's head bone to simulate the virtual human looking around its environment.

We use a LOD approach, where we pre-define a maximum number of virtual humans that can implement this randomly generated gaze-behaviour. We limit the number of virtual humans performing these low-level behaviours, since they will be not noticeable at a distance. At run-time, depending on the current number of virtual humans performing this action, a virtual human is assigned a gaze behaviour when it is within the view-frustum and within a certain distance. Once the virtual humans can no longer be seen by the viewer, the behaviour is discarded.

Conversational Behaviour

To add variety and to enhance the realism of the simulation, it is important for virtual humans walking in groups to interact with each other. Using the same approach described in Section 3.4.3, we allow these virtual humans to converse. We plan to use the LOD approach already present in ALOHA system whereby, depending on a couple's position with respect to the viewer, the level of detail for the conversational behaviour is selected. The system currently implements low rated couples with a random behaviour utilizing low detailed key-

frame animations. Since virtual humans rendered with the impostor representation cannot perform arm gestures (see Section 4.1.2), a lower-resolution mesh is used for representing couples talking in the distance.

4.2.2 Virtual Human LOD Shadows

Our run-time system enhances the realism of the virtual humans and the environment they inhabit by creating shadows on the ground wherever the light is blocked. Our shadow technique is based on the planar projected shadow algorithm and is implemented in hardware using per-pixel stencil testing. This section will describe how this technique is used to render the virtual humans' shadows.

The planar projected shadow algorithm is used to cast a geometric model's shadow onto a ground plane based on the light's position. In order to achieve this, a planar projected shadow matrix can be constructed. Given the equation for a ground plane G: $\vec{N}$ + d = 0 and the homogenous position of the light $\vec{L}$, a 4×4 planar projected shadow matrix S can be constructed using Equation 4.15 (see [Bli88] [HMAM02] for the derivation of the matrix).

$$S = \begin{pmatrix} dot - L_x * N_x & -L_x * N_y & L_x * N_z & -L_x * d \\ -L_y * N_x & dot - L_y * N_y & -L_y * N_z & -L_y * d \\ -L_z * N_x & -L_z * N_y & dot - L_z * N_z & -L_z * d \\ -L_w * N_x & -L_w * N_y & -L_w * N_z & dot - L_w * d \end{pmatrix} \tag{4.15}$$

$$where\ dot = N_x * L_x + N_y * L_y + N_z * L_z + d * L_w$$

Stenciling works by tagging pixels in one rendering pass to control their update in subsequent rendering passes. It is an extra per-pixel test that uses the stencil buffer to track the stencil value of each pixel. When the stencil test is enabled, the frame buffer's stencil values are used to accept or reject rasterized fragments. When rendering the scene, the stencil buffer is cleared at the beginning and a unique non-zero stencil value is assigned to pixels belonging to the ground plane. In the first rendering pass, the shadow cast by each virtual human's geometric representation is rendered. Using the matrix S, the geometry is projected onto the ground plane and rendered into the stencil buffer, where each pixel is tagged with the ground plane's unique stencil value. In the subsequent rendering pass, each virtual human's representation is rendered and the appropriate areas of the stencil buffer are simultaneously cleared. This prevents an artefact whereby shadows might overwrite real objects, damaging the realism of the scene. Finally, a single semi-transparent quad is rendered over the whole scene (where the stencil buffer pixels have been set to the unique stencil value) resulting in

realistically blended shadows.

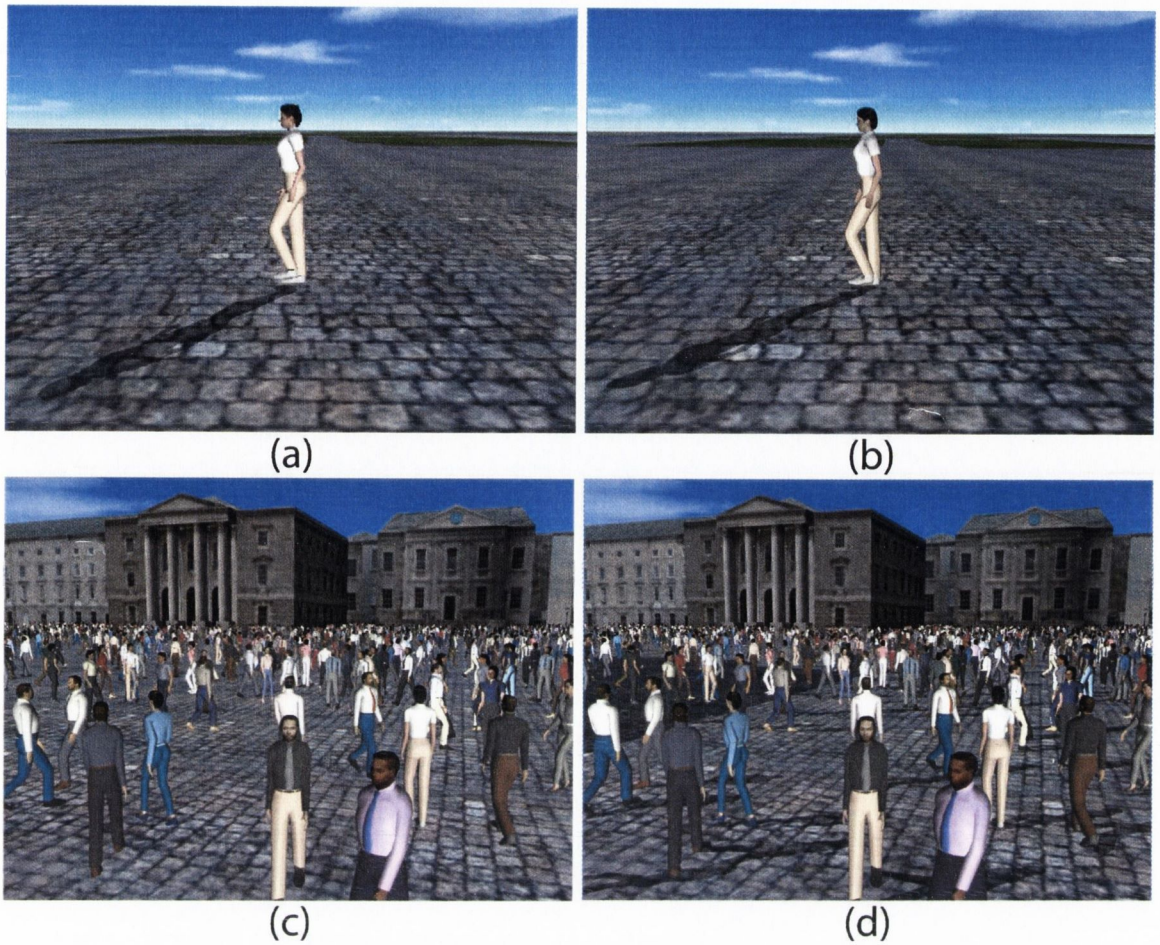


Figure 4.15: (a) Projected impostor shadow. (b) Projected mesh shadow. (c) Crowd and city without shadows. (d) Crowd and city with projected LOD shadows.

Our shadow technique uses a LOD approach, where either the impostor or mesh representation is projected onto the ground plane depending on which LOD representation the virtual human is currently using (see Figure 4.15 (a) and (b)). To render the virtual human's shadow using the impostor representation, we need to calculate which viewpoint image needs to be displayed with respect to the light's position and rotate its quadrilateral so that it always faces the light. Using the virtual human's position $\overrightarrow{H}$ and the light's position $\overrightarrow{L}$, the quadrilateral's normal vector $\overrightarrow{N}$ can be calculated using Equation 4.16. The projection of the impostor onto the ground plane with respect to the light position can be calculated using $\overrightarrow{N}$ and Equations 4.6 and 4.8 (previously described in Section 4.1.2). The impostor's shadow requires no more than a single textured quad, and therefore is extremely fast to render.

$$\vec{N} = \frac{\vec{H} - \vec{L}}{|\vec{H} - \vec{L}|} \tag{4.16}$$

While this method is similar to that employed by Loscos et al. [LTC01], our use of the stencil buffer instead of darkened textures results in shadows that blend realistically with both the underlying world and each other (see Figure 4.15 (d)). The main advantage of implementing this shadow algorithm with the stencil buffer is that it can avoid artefacts caused by double blending and can limit the shadow to an arbitrary ground plane surface. Unfortunately, unlike full geometric stencil shadows, our projection shadows are restricted to the ground plane and do not project onto nearby static objects, or other dynamic objects. While shadow mapping could be used to solve this problem (see Section 2.1.2), a LOD approach would be needed to deal with the many hundreds or thousands of shadows. It should be noted that shadow volumes were not considered in the system as this technique can decrease the pixel fill rate and the constructed shadow volume for an impostor is incorrect as a result of being a semi-transparent quadrilateral.

4.2.3 Performance Optimisations

Virtual Human Occlusion Culling

As a first step towards improving performance, view frustum culling can be used to eliminate those humans that are not potentially on screen. However, due to the densely occluded nature of an urban environment, large groups of humans may be in the frustum but occluded by buildings and therefore rendered unnecessarily. By avoiding the rendering of these humans using occlusion culling techniques, this should greatly improve the performance of the system [CT97, BHS98, SVNB99, WS99, Zha98].

We make use of hardware accelerated occlusion culling similar to the technique used by Saulters et al. [SF02] to cull large sections of the crowd. We utilise the *ARB_occlusion_query* extension to determine the visibility of an object. This extension defines a mechanism whereby an application can query the number of pixels drawn by a primitive or group of primitives. Typically, the major occluders are rendered and an occlusion query for the bounding box of an object in the scene is performed. If a pixel is drawn for that object's bounding box, then the object is not occluded and therefore should be displayed. The main performance advantage of this extension is that it allows for parallelism between the CPU and GPU, since many queries can be issued before asking for the result of any one. This means that more useful work, such as the rendering of other objects or other computations on the CPU, can be carried out while waiting for the occlusion query results to be returned.

Since the city is populated by several thousand humans, there could potentially be a large number of humans in the view frustum and therefore it would be computationally inefficient to perform a separate occlusion query for each human. To facilitate the occlusion culling of buildings, the virtual city is divided into a grid of regular-sized nodes. By re-using these nodes so that they store which virtual humans inhabit them, this can help to avoid performing separate occlusion culling queries for each human. Having initially rendered the static environment, we perform occlusion queries on the bounding volume of any nodes in the view-frustum, thus allowing us to rapidly discard those nodes hidden by the environment and the humans within them. With regards to the unoccluded nodes, we perform view-frustum culling on the virtual humans within these nodes, since parts of these nodes may not be within the view frustum. It should be noted that the height of each node's bounding volume is set to the height of the tallest virtual human used in the system to allow humans to still be displayed when they are behind an occluding object whose height is less (e.g., walls). This occlusion culling method could be extended so that the number of pixels drawn for a node could be used as a metric to decide on what level of detail the humans in the node should use, with regards to representation, behaviour, and animation.

Additionally, the occlusion results for each node are used to reduce the computational load of updating every human's position in the scene. This technique takes advantage of the fact that a large number of humans are occluded per frame and therefore their position in the world can remain unchanged without the viewer noticing. By storing the time each node was last unoccluded, the position of a human is only updated if the node it inhabits has been unoccluded for the last five seconds. However, simulation artefacts can arise when the camera's position remains static for a period of time and the humans move from an unoccluded node to an occluded node. This results in the congregating of humans on the boundary of these occluded nodes since their steering behaviour is not being updated. A potential solution to this problem would involve a LOD simulation approach whereby humans are updated at a frequency dependent on the last time the node was unoccluded.

Virtual Human Simulation LOD

While frustum and occlusion culling decrease the rendering workload, there are still overheads associated with updating the positions of thousands of humans in motion. To lighten the workload we pause humans within nodes that have not been visible for more than five seconds. This time delay prevents temporal artefacts becoming noticeable amongst the nearby humans when performing rapid camera rotation. In addition to this, checking whether a node is occlusion culled is only performed every 100 milliseconds if the camera has moved or rotated, since the same nodes will be occluded if the camera remains stationary. Since the humans

only move every 100 milliseconds, we reduce the number of times we check whether a human is within the view-frustum by performing this test every time the humans move instead of every frame.

Minimising OpenGL State Changes

OpenGL is a simple state machine with two operations: setting a state, and rendering utilizing that state. By minimizing the number of times a state needs to be set, this can maximize performance since it minimizes the amount of work the driver and the graphics card 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 attempt 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, rendering is optimized by sorting the virtual humans in the following order based on the most to least expensive state changes: binding a shader, binding a texture, and setting VBO data pointers. By organising the rendering of our crowd in this manner, our approach sorts each virtual human by LOD representation, then by template model, and finally by the current key-frame of animation. Sorting the virtual humans by LOD representation minimizes the number of times that the following states have to be changed: the setting of lighting parameters, alpha test enabling and disabling, and vertex and fragment programs. Next, sorting the LOD representations based on template model minimizes texture loads and binds. Finally, sorting virtual humans using the same template model by animation key-frame reduces the setting of VBO data pointers, since each VBO stores the data for a particular key-frame. In the case of rendering virtual humans using the same model and animated with the same key-frame, an extra step needs to be implemented to sort them based on the viewpoint required with respect to the camera. This is necessary, since certain viewpoints for the current key-frame are obtained by mirroring the same viewpoint for the symmetrical key-frame. By sorting impostors based on whether the viewpoint is mirrored, this minimizes texture loads and binds.

Minimising Texture Thrashing

Texture thrashing can become a serious problem when populating a virtual city with crowds using a number of different pre-generated impostor models. In addition to each impostor model requiring 1.5MB of texture memory every frame, the city model will also require a certain amount of texture memory. Therefore, as the number of template models within the virtual city increases, texture thrashing will occur much sooner as a result of the extra texture

memory being consumed by the city model. It should be noted that, in the case of real-time applications where the camera is fixed, say at eye-level, only 17 viewpoint images are needed for each frame of animation and therefore the consumption of texture memory is less of a problem. Since we wanted to implement a more generic system, where the camera can view the city from any height, 17 by 8 viewpoints are needed for the impostor representation.

However, as only a subset of the viewpoints in the impostor textures is being used every frame, we propose splitting the impostor detail and the normal map images into eight separate smaller *elevation* images containing the set of viewpoints pre-generated at each camera height. To facilitate the creation of these elevation images, an application was written in C to allow the positioning of viewpoint images within a larger image. The application reads in the 17 viewpoint images for a particular camera height and, based on the sum of these images' area, the minimum dimensions of the elevation image are calculated. Once the viewpoints have been loaded in, the application allows the user to organise the viewpoints within the new elevation image. Unfortunately, since the area of each viewpoint image varies, it is not guaranteed that they will all fit within the minimum dimensions and therefore have to be increased by a factor of two along a single dimension. Once the user has got all the 17 images to fit, the new elevation image is exported (shown in Figure 4.16).

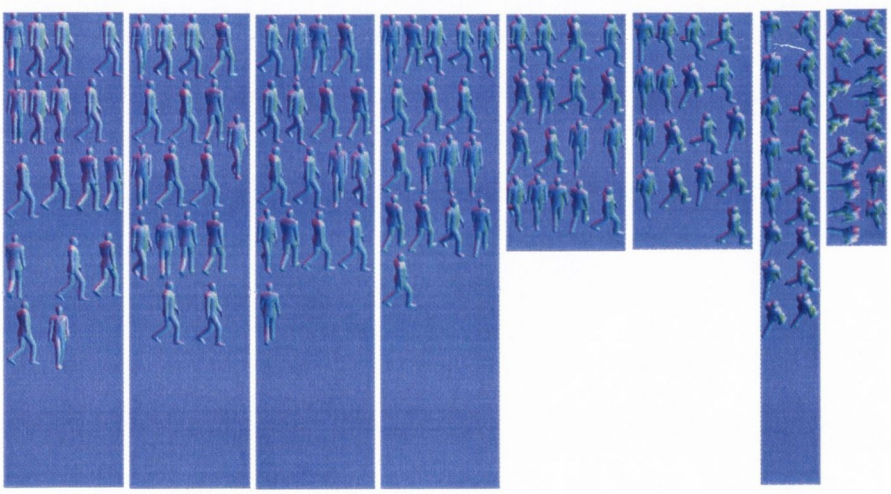


Figure 4.16: Normal map split into smaller elevation images.

The number of elevation images needed to render impostors using a particular human model type depends on the height of the camera and the distance of the camera from each impostor. Since buildings in a city environment generally occlude humans in the distance, all elevation images should never be needed simultaneously. The angle (θ_E) between the impostor and the camera around the horizontal axis, can be calculated using Equation (4.17), where

h_{cam} is the camera height and d_{xz} is the distance on the x-z plane from the camera to the impostor. Using θ_E, the elevation image needed for that impostor can be calculated. As the camera's height decreases, the number of elevation images needed is reduced dramatically (see Equation (4.17). Taking advantage of the occluding nature of city environments, this method of separating impostor and normal map images for each elevation permits greater variety, without texture thrashing, as a result of each human model type consuming less texture memory.

$$\theta_E = tan^{-1}(\frac{h_{cam}}{d_{xz}}) \tag{4.17}$$

4.3 Summary and Conclusion

In this chapter we have described the Geopostor system. We have detailed the various LOD representations that can be employed by the crowd, including our method of switching between them. Additionally, we have discussed the various levels of variation and animation that can be applied to our crowd in order to remove the disconcerting effect of a crowd made up of cloned individuals. The resulting system is capable of creating realistic crowds at real-time frame rates. From urban planning to computer games, the population of these types of applications with large-scale crowds is important.

In the following chapter we evaluate our crowd's representations with respect to their computational cost and test the performance of our system in the urban environment previously described. In order to maintain the crowd's realism, we carry out a detailed investigation into the perception of virtual human representations.

Chapter 5

Evaluation

In this chapter, we first evaluate the rendering cost of the LOD representations used in the Geopostor system, and test how this costs scales with the number of different template models used. Next, we test the performance of these representations in an urban environment, utilizing their rendering cost to calculate the maximum number of humans that can populate the environment, and compare the results with our Geopostor system switching between representations. Additionally, we test how the performance of the crowd simulation can be improved using various techniques. While the computational evaluation of human representations is a common method for deciding on when they should be used in a real-time applications, the area of perceptually evaluating them has been largely ignored. To deal with this issue, we introduce a method for perceptually evaluating these representations, based on psychophysical experimental methods. Subject to the computational and perceptual test results, we discuss the advantages and disadvantages of each representation, and apply this to our system to provide a balance between visual quality and performance.

5.1 Computational Evaluation of Virtual Human Representations

Two sets of tests were performed on the following virtual human geometric LOD representations used in our Geopostor system: deformable mesh representation and pre-generated impostor representation. These tests were performed in order to evaluate how the realism and variety of the simulated crowd affected the rendering cost of the representation used. This cost is important, as it allows developers of crowd systems to estimate the number of humans, using a particular LOD representation, that can be displayed at a desired frame rate.

5.1.1 Computational Evaluation Test Conditions

An OpenGL application was used to perform the tests, where the test environment consisted of a white background and displayed a number of virtual humans using the tested representation. Each virtual human walked on the spot using the default walk animation described in Section 4.1.1. The crowd's individuals were positioned in a regular grid-based manner so that they were always fully on-screen, and therefore no occlusion culling or frustum culling tests were performed. All of our tests were performed using a PentiumIV 3.6Ghz processor, with 2.0GB RAM and an ATI Radeon X850 XT Platinum Edition graphics card with 256MB of video memory. It should be noted that hardware accelerated anti-aliasing was disabled for all of these tests.

We performed 5 tests containing a crowd of 100, 250, 500, 750 and 1,000 individuals, and calculated each test's average frame rate (FPS_{Test}), which involved averaging the frame rate recorded for 1000 frames. The rendering cost for each test ($Cost_{LOD\ Test}$) was calculated using Equation 5.1, where N_T is the size of the crowd, and the rendering cost for the representation ($Cost_{LOD}$) was subsequently calculated as the average of the tests' FPS_{Test} values. The number of humans ($N_{Threshold}$), that can be displayed at a desired frame rate ($FPS_{Threshold}$), can be estimated using Equation 5.2.

$$Cost_{LOD\ Test} = \frac{1}{(FPS_{Test}) \times N_T} \tag{5.1}$$

$$N_{Threshold} = \frac{1}{FPS_{Threshold} \times Cost_{LOD}} \tag{5.2}$$

For each representation, we evaluated the effect of the various levels of detail on its rendering cost, as well as other factors specific to the representation. The main levels of detail tested were the following:

- **Constant Shading: ($\text{Shading}_{LOD} = 0$)**
  One LOD approach for reducing the cost of a representation is by disabling lighting or using constant shading for virtual humans that are far from the viewer (see [dHCUCT04]). The representations were lit with constant shading and without any variation, as this is the least computationally expensive method for rendering them.

- **Gouraud Shading: ($\text{Shading}_{LOD} = 1$)**
  The representations were rendered with Gouraud shading and the following Variation_{LOD} were tested to see how the simulated crowd's level of variety affected the rendering cost (Table 4.2 and Table 4.4 explain each Variation_{LOD} for the mesh and the impostor representation, respectively):

  - **$\text{Variation}_{LOD} = 0$:**
    The representations were rendered without any variation to provide a baseline.
  - **$\text{Variation}_{LOD} = 0$, with projected ground-plane shadows:**
    The representations' were rendered with their shadows projected on the ground plane using the approach described in Section 4.15.
  - **$\text{Variation}_{LOD} = 1$:**
    The representations were rendered with colour variation using the approach described in Section 4.1.1.
  - **$\text{Variation}_{LOD} = 2$:**
    The representations were rendered with colour variation and displayed varied crowd motion, using the approach described in Section 4.1.1.
  - **$\text{Variation}_{LOD} = 3$**
    The representations were rendered using 5 and 10 different template models. A maximum of 10 template meshes was chosen, since crowds in games typically consist of no more than 10 different models.

5.1.2 Set I Tests: Deformable Mesh Representation

A high resolution mesh (100%) and 4 low resolution meshes (60%, 45%, 30%, and 15%) were used in these tests, where the corresponding number of vertices and triangles are shown in Table 5.1. All of the triangles were texture mapped using a single decal RGBA image of 256×256 pixels (262KB). As described in Section 4.1.1, the deformation of each mesh resolution was precalculated and stored in a vertex buffer object. This results in a larger amount of video memory being consumed than storing the character's mesh once and deforming it at run-time. Since memory is more limited in game applications, it is common practise to

use the latter method. However, this increases the rendering cost of the mesh since it needs to be deformed on a per-frame basis for every human and therefore the pre-calculated poses were used in these tests to improve the rendering speed.

Mesh Resolution	Vertex Percentage	Number of Vertices	Number of Triangles
High	100%	1383	2170
Low 4	60%	854	1258
Low 3	45%	653	937
Low 2	30%	456	612
Low 1	15%	245	298

Table 5.1: The vertex percentage, number of vertices and number of triangles for each mesh resolution tested.

Set I Tests: LOD Settings

Using each mesh resolution, the crowd was rendered with the following settings:

- **Constant Shading Test**:
  - $\text{Shading}_{LOD} = 0$, $\text{Variation}_{LOD} = 0$.
  - 1 template model.
  - 1 outfit texture bound per frame.
  - Uniform Crowd Motion (1 VBO bound per frame).
- **Gouraud Shading Test**:
  - $\text{Shading}_{LOD} = 1$, $\text{Variation}_{LOD} = 0$.
  - 1 template model.
  - 1 outfit texture bound per frame.
  - Uniform Crowd Motion (1 VBO bound per frame).
- **100 Outfits Test**:
  - $\text{Shading}_{LOD} = 1$, $\text{Variation}_{LOD} = 1$.
  - 1 template model.
  - 100 outfit textures bound per frame.
  - Uniform crowd motion (1 VBO bound per frame).
- **100 Outfits with Motion Variation Test**:

- $\text{Shading}_{LOD} = 1$, $\text{Variation}_{LOD} = 2$.
- 1 template model.
- 100 outfit textures bound per frame.
- Varied crowd motion (10 VBOs bound per frame).

- **Shadows Test**:
  - $\text{Shading}_{LOD} = 1$, $\text{Variation}_{LOD} = 0$.
  - 1 template model.
  - 1 outfit texture bound per frame.
  - Uniform crowd motion (1 VBO bound per frame).
  - Ground-plane shadows.

- **Template Models Tests**:
  - $\text{Shading}_{LOD} = 1$, $\text{Variation}_{LOD} = 3$.
  - N template models used (where N = 1, 5, and 10).
  - N*100 outfit textures bound per frame.
  - Varied crowd motion (N*10 VBOs bound per frame).

Set I Tests: Results

Both the rendering cost and the size of the crowd that can be displayed at 30 frames per second were calculated for each mesh resolution (see Figure 5.1).

As expected, Gouraud shading is more expensive than constant shading, since it requires additional per vertex computations. However, these computations result in the variation in shade across the mesh's triangles, and greatly improving the realism of the mesh in comparison to constant shading, which results in the same colour used across the triangle. In the case of visual realism not being an important factor e.g., an application simulating crowds in an emergency simulation, then constant shading should be used due to its low rendering cost. In the case of all settings, the rendering cost of the 100% mesh is proportionally higher than all other resolutions. This suggests that the 100% mesh is vertex/geometry bound due to the higher number of polygons. Further timing tests investigating this bottleneck were not carried out since the rest of the data provided sufficiently correct results.

An interesting result is that rendering the meshes with a Variation_{LOD} of 1 or 2 has little effect on the rendering cost. This indicates that, while the mesh representation is expensive

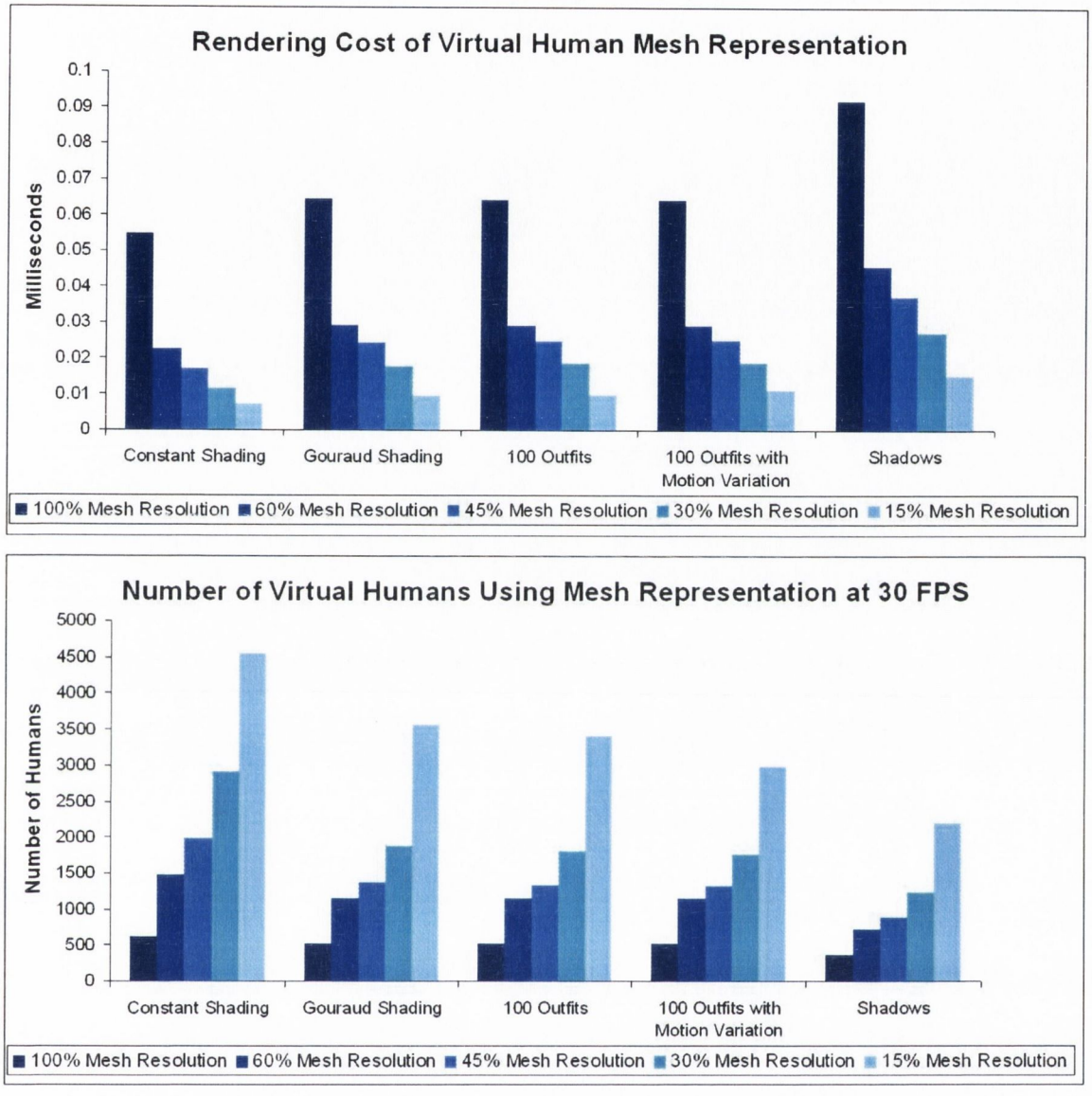


Figure 5.1: (a) Rendering cost and (b) number of humans displayed at 30 frames per second for a mesh at a resolution of 100%, 60%, 45%, 30% and 15%.

to render due to its geometric complexity, adding colour and motion variation, which requires several VBOs bound per frame, has little effect on the rendering cost. Therefore, colour and motion variation should be employed when rendering a crowd utilizing a deformable mesh representation, since it results in a more visually interesting crowd for minimal extra cost.

Casting shadows on the ground-plane comes at a great cost, since twice as many triangles are being rendered per-frame as a result of rendering the model into the stencil buffer. Therefore, while virtual environments that require a greater realism should use projected ground-plane shadows for the virtual humans, they should be used only when noticeable as only a limited number of polygons can be displayed in each frame of a simulation.

Finally, the cost of rendering each set of template meshes was calculated (shown in Figure 5.2) and the results show that the number of template meshes has no effect on the rendering cost, in the same way that there was no difference between rendering the mesh representation with either a Variation_{LOD} of 0 or 2. Both of these results indicate that the cost of binding the additional VBOs per frame is not a computationally expensive operation.

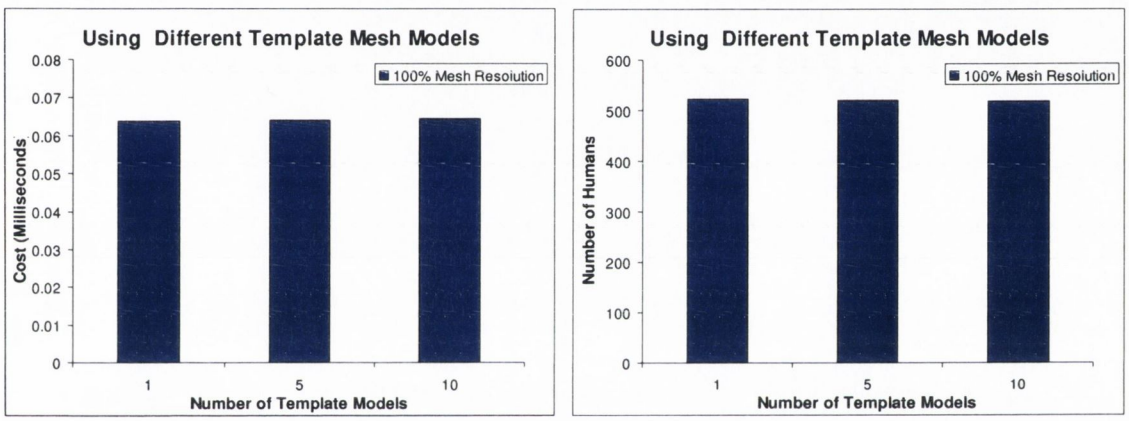


Figure 5.2: (a) The cost of using 1, 5 and 10 different template meshes. (b) The number of humans that can be rendered at 30 frames per second using 1, 5 or 10 different meshes.

Adding Variation Using OpenGL Fixed Function Pipeline Vs. Programmable Graphics Hardware

This extra test was carried out to test whether there was a difference in the rendering cost between adding variation using the OpenGL fixed function pipeline or graphics hardware. These tests were carried out for each mesh resolution, but for only 1,000 virtual humans. In both cases, each individual's mesh was rendered with Gouraud shading with no variation (i.e, a single outfit and the entire crowd walk in step), thus requiring a single outfit and VBO bind per frame.

In the case of using the fixed function pipeline, the mesh was organised into five batches of triangles (see Section 4.1.1 for more details), where each batch represented the group of triangles to be shaded with a particular diffuse colour (defined by the outfit). In order to add variation through graphics hardware, a fragment program utilizing texture indirection to lookup the outfit's texture (see Section 4.1.1) was implemented.

From these results (see Figure 5.3(a)), adding colour variation using the OpenGL fixed function pipeline causes a bottleneck for the lower mesh resolutions. This is caused by application-limited code, where the diffuse colour for each batch of triangles needs to be changed each time the mesh is rendered (e.g., to render 1,000 meshes where each mesh uses 5 materials would involve changing the diffuse colour 5,000 times in total). Since adding colour variation through programmable graphics hardware does not need to change the diffuse colour based on the triangle batches and therefore allows the entire mesh to be rendered in a single draw call, this allows more virtual humans to be displayed, especially for the lower resolution meshes (see Figure 5.3(b)). Another advantage of adding variation through programmable graphics hardware is that colour modulation depends on per pixel alpha values, thus allowing for more detailed variation in comparison to the more limited per triangle-based OpenGL approach.

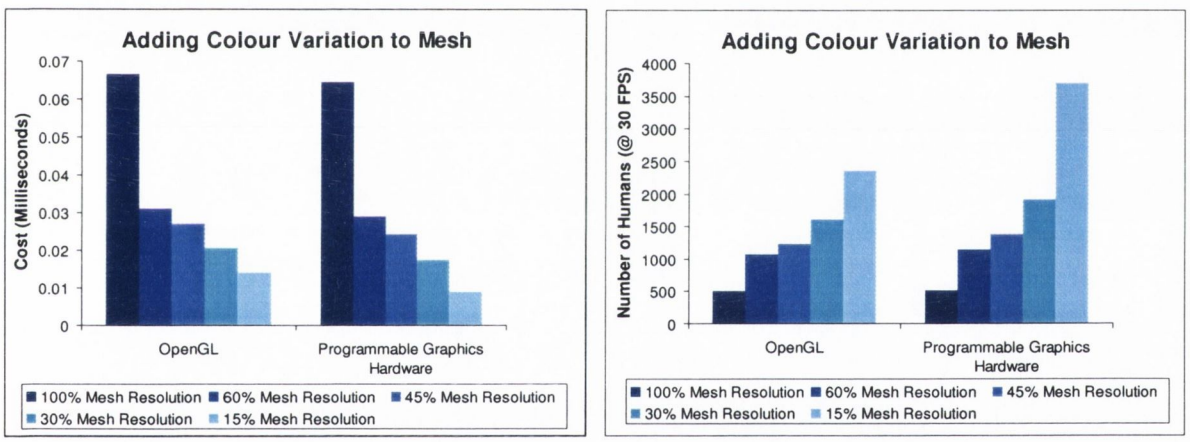


Figure 5.3: (a) The cost of adding variation using the OpenGL fixed function pipeline and programmable graphics hardware. (b) The number of humans that can be coloured at 30 frames per second using either programmable graphics hardware or OpenGL's fixed function pipeline.

5.1.3 Set II Tests: Pre-Generated Impostor Representation

The same pre-generated impostor representation described in Section 4.1.2 was used.

Set II Tests: LOD Settings

The framerate was recorded for the 5 tests (100 to 1000 virtual humans) using the following settings:

- **Constant Shading Test**:
  - $\text{Shading}_{LOD} = 0$, $\text{Variation}_{LOD} = 0$.
  - 1 template model
  - 1 outfit texture bound per frame.
  - Uniform crowd motion (1 detail map image bound per frame).
- **Gouraud Shading Test**:
  - $\text{Shading}_{LOD} = 1$, $\text{Variation}_{LOD} = 0$.
  - 1 template model
  - 1 outfit texture bound per frame.
  - Uniform crowd motion (1 detail map and 1 normal map image bound per frame).
- **100 Outfits Test**:
  - $\text{Shading}_{LOD} = 1$, $\text{Variation}_{LOD} = 1$.
  - 1 template model
  - 1 outfit texture bound per frame.
  - Uniform crowd motion (1 detail map and 1 normal map image bound per frame).
- **100 Outfits with Motion Variation Test**:
  - $\text{Shading}_{LOD} = 0$, $\text{Variation}_{LOD} = 2$.
  - 1 template model
  - 100 outfit textures bound per frame.
  - Varied crowd motion (10 detail map and 10 normal map images bound per frame).
- **Shadows Test**:
  - $\text{Shading}_{LOD} = 1$, $\text{Variation}_{LOD} = 0$.
  - 1 template model
  - 1 outfit texture bound per frame.

- Uniform crowd motion (1 detail map and 1 normal map image bound per frame).
- Ground-plane shadows.

- **Template Models Tests:**
  - $\text{Shading}_{LOD} = 1$, $\text{Variation}_{LOD} = 3$.
  - N template models used (where N = 1, 5, and 10).
  - N*100 outfit textures bound per frame.
  - Varied crowd motion (N*10 detail map and N*10 normal map images bound per frame).

Set II Tests: Results

For each setting, both the rendering cost and the number of virtual humans that can be displayed at 30 frames per second were calculated (Figure 5.4 and Figure 5.5).

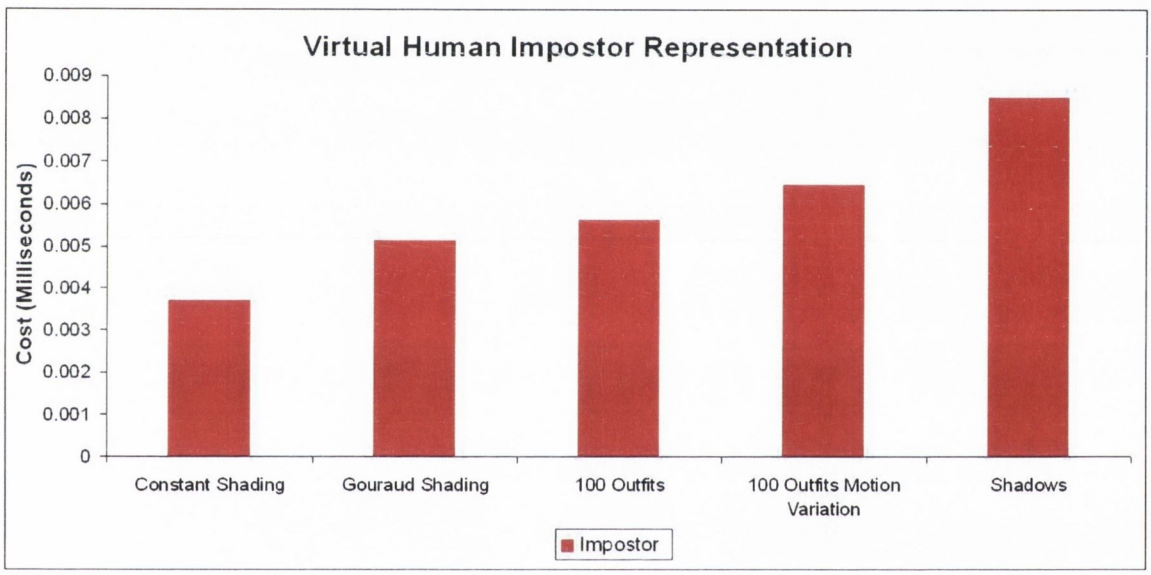


Figure 5.4: Rendering cost of using the impostor representation.

Again, constant shading is less computationally expensive than Gouraud shading since it does not require the binding of the normal-map image and the extra computations to shade each pixel based on the position of the environment's light. However, this comes at the price of realism, as the impostor not being dynamically lit. While adding colour variation has a small effect on the rendering cost of the impostor, adding motion variation has a greater impact due to the the extra impostor textures being loaded in from texture memory and

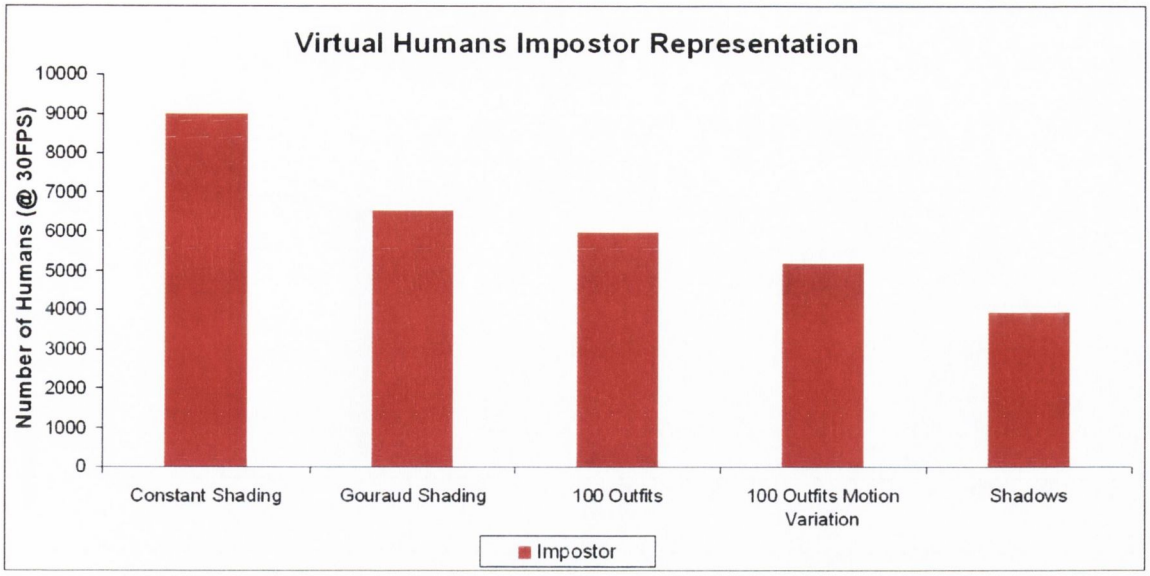


Figure 5.5: Number of humans displayed at 30 frames per second using the impostor representation.

bound to the impostor's quadrilateral. Finally, casting shadows on the ground plane almost doubles the rendering cost due to the impostor's quadrilateral being rendered into the stencil buffer, and therefore should only be used when the shadows are noticeable. The difference between the number of humans that can be rendered at 30 frames per second using the mesh and the impostor representation is shown in Figure 5.6.

It should be noted that there is a small difference between using the impostor and 15% mesh resolution even though the mesh contains almost 300 more polygons. This is because the rendering of the impostor representation requires more per-pixel operations and each quadrilateral is rendered with a separate API call unlike the 15% mesh resolution where a single API call is used to draw 300 batched triangles. However, the impostor's rendering cost can be further reduced by using a technique similar to Gosselin et al. [GSM05] that would involve rendering multiple impostors per draw call, thus reducing driver overhead.

Unlike the mesh representation, increasing the number of template models used affects the rendering cost due to the extra impostor textures needed (Figure 5.7). Since these tests were performed using a graphics card with 256MB of memory, texture thrashing due to the loading in and out of textures is therefore not an issue. However, in the case of graphics cards with less memory (i.e., 128MB), texture thrashing would be a problem in these tests once the number of template models increased.

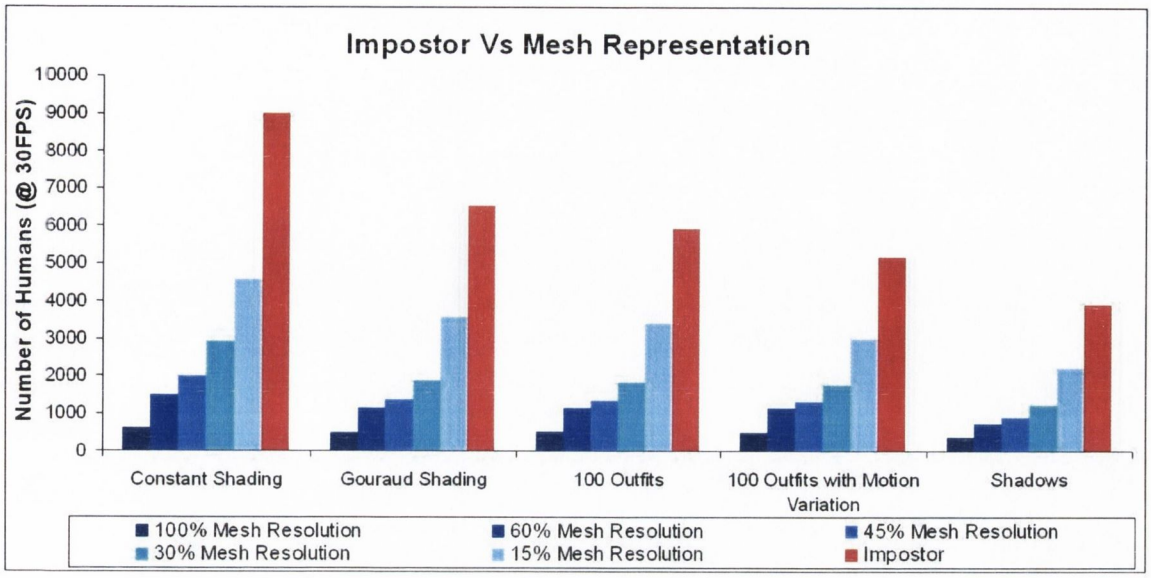


Figure 5.6: Number of humans displayed at 30 frames per second using the impostor and mesh representation.

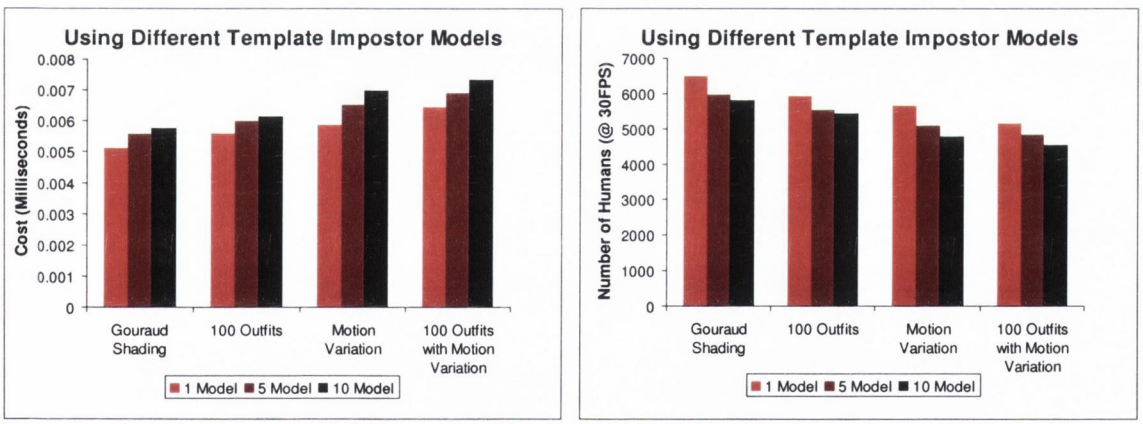


Figure 5.7: (a) The cost of using 1, 5 and 10 different template impostors. (b) The number of humans that can be rendered at 30 frames per second using 1, 5 or 10 different impostors.

Hybrid Impostor/Mesh Model Tests

This additional test was performed to calculate the rendering cost of displaying the hybrid impostor/mesh model (described in Section 4.1.2), whereby the impostor's head was replaced with a gesturing deformable mesh. Each virtual human's head mesh was animated with a randomly generated head animation, while the impostor representation performed the walk animation. From the head's original high resolution mesh, two low resolution meshes were generated at 50% and 25% and Cost_{LOD} was calculated for each resolution. During these tests, each virtual human hybrid representation was fully-lit with Gouraud shading, used the same outfit texture, and walked in step with the other virtual humans. Furthermore, virtual humans utilising three different mesh resolutions (100%, 50% and 25%) and performing the same head animation (based on the method described in Section 4.1.1) were tested for comparison (see Table 5.3 for each resolution's corresponding number of vertices and triangles).

Head Mesh Resolution	Vertex Percentage	Number of Vertices	Number of Triangles
High	100%	317	580
Low 2	50%	167	296
Low 1	25%	85	146

Table 5.2: The vertex percentage, number of vertices and number of triangles for each head mesh resolution used in the hybrid representation's tests.

Mesh Resolution	Vertex Percentage	Number of Vertices	Number of Triangles
High	100%	1383	2170
Low 2	50%	725	1043
Low 1	25%	385	499

Table 5.3: The vertex percentage, number of vertices and number of triangles for each mesh resolution used in the hybrid representation's tests for comparison.

From these results (Figure 5.8), due to the extra triangles in the head mesh, the rendering cost of the hybrid representation is more expensive than an impostor just performing the walk animation. However, the hybrid representation is cheaper than a mesh performing the same head behaviour since the mesh representation's body is replaced by a texture-mapped quadrilateral. It should be also noted that, by reusing the pre-computed VBO data, only the vertices in the head mesh need to be deformed and therefore the difference between rendering the mesh representation with and without a head animation is marginal.

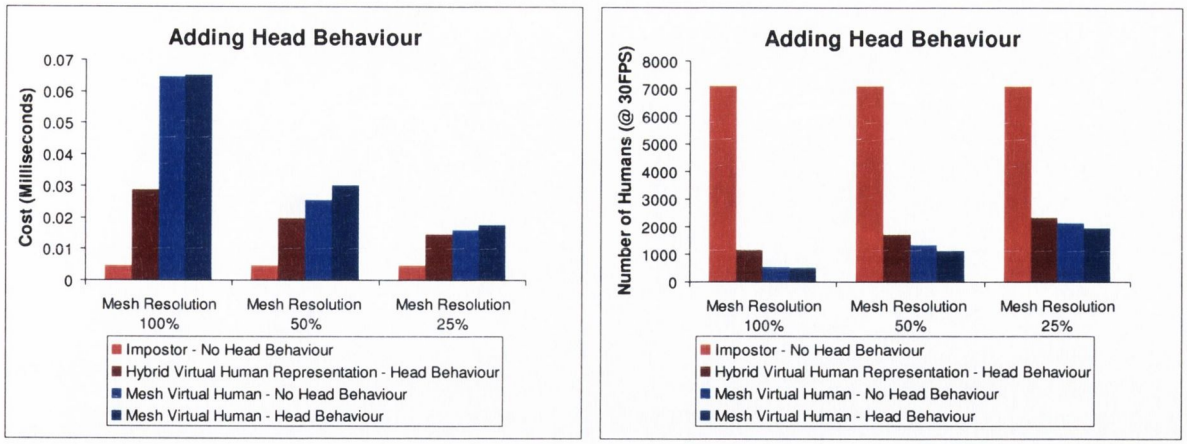


Figure 5.8: (a) The cost of adding head behaviour. (b) The number of humans with head behaviour displayed at 30 frames per second.

5.1.4 Video Memory Consumption

The consumption of video memory needs to be considered when dealing with real-time graphics. Video memory is used to store the various buffers (e.g., frame buffer, depth buffer, and stencil buffer), geometric data and texture maps used by an application. The amount of memory consumed by a buffer is typically several megabytes (depending on its resolution and color depth) and therefore is negligible in comparison to the amount consumed by geometric data and texture maps needed to simulate a realistic environment. Since video memory is limited, it needs to be budgeted and this affects various factors in the Geopostor system (e.g., number of individuals, animations and viewpoints). Otherwise, the application's interactivity will suffer due to the thrashing of memory.

In the case of the impostor representation, video memory is consumed by its textures and the vertex buffer object used to store the geometry for each viewpoint's billboard. The amount of memory consumed by an impostor's detail and normal map for a single frame of animation is listed in Table 5.4. Since only a subset of viewpoints are needed per frame, splitting the textures into smaller elevation textures greatly reduces the amount of memory consumed (see Section 4.2.3). Depending on the elevation at which the viewpoints are generated from, the viewpoints can be fitted into either a 128×1024 or a 256×1024 image. While mip-mapping increases the memory consumed by the textures by a third, this is necessary to remove visual artefacts (see Section 4.1.2).

The amount of memory consumed by the impostor's vertex buffer object for a single frame of animations is 0.002 MB for 17 viewpoints and 0.015 MB for 17 by 8 viewpoints (using Equation 5.3). It should be noted that there is no need to store vertex normals in

Image	Format	Viewpoints	Dimensions	Compression Ratio	Memory Size
Detail Map	RGBA (4 bytes)	17×8	1024×1024	4:1 (DXT3)	$(4\times1024\times1024\times\frac{1}{4})$ = 1.0 MB
Normal Map	RGB (3 bytes)	17×8	1024×1024	6:1 (DXT3)	$(3\times1024\times1024\times\frac{1}{6})$ = 0.5 MB
TOTAL (no mip-mapping)					1.5MB
TOTAL (mip-mapping)					2.0MB
Detail Map	RGBA (4 bytes)	17	(128→256) ×1024	4:1 (DXT3)	$(4\times(128\rightarrow256)\times1024\times\frac{1}{4})$ = 0.125→0.25 MB
Normal Map	RGB (3 bytes)	17	(128→256) ×1024	6:1 (DXT3)	$(3\times(128\rightarrow256)\times1024\times\frac{1}{6})$ = 0.0625→0.125 MB
TOTAL (no mip-mapping)					0.1875→0.375MB
TOTAL (mip-mapping)					0.25→0.5MB

Table 5.4: The amount of video memory consumed by the impostor's detail map and normal map for a single frame of animation.

the impostor vertex buffer object since the billboards are dynamically oriented towards the camera and therefore the normals are always the same.

$$\begin{aligned} \textit{Impostor VBO (bytes)} = N_{Viewpoints} \times (\underbrace{2}_{N_{Triangles}}) \times (\underbrace{3}_{N_{TriangleVertices}}) \\ \times (\underbrace{3}_{Position_{xyz}} + \underbrace{2}_{TexCoord_{uv}}) \times (\underbrace{4\ bytes}_{Size\ of\ float}) \end{aligned} \tag{5.3}$$

In the case of the mesh representations, video memory is consumed by its diffuse texture and the vertex buffer object used to store the precomputed poses. The amount of memory consumed by the diffuse texture is 0.167 MB (see Table 5.5 for the diffuse texture's details). The amount of memory consumed by the vertex buffer object depends on the number of poses it stores and on the number of triangles in the mesh (see Equation 5.4). Table 5.6 lists the

size of a vertex buffer object used to store a single pose for the same 100% mesh resolution used in Section 5.1.2.

Image	Format	Dimensions	Compression	Memory Size
Detail Map (with mip-mapping)	RGBA (4 bytes)	256×256	4:1 (DXT3)	$(4\times256\times256\times\frac{1}{4})$ = 0.0625 MB ≈ 0.083 MB

Table 5.5: The amount of video memory consumed by the mesh's diffuse texture.

LOD	$\text{Number}_{Triangles}$	Number of Poses	Memory Consumed by VBO
Mesh	2170	1	0.198 MB

Table 5.6: The amount of video memory consumed by a mesh's vertex buffer object storing a single pose.

$$
\begin{aligned}
Mesh\ VBO\ (bytes) \;=\; & (N_{Poses}) \times (N_{Triangles}) \times (\underbrace{3}_{N_{TriangleVertices}}) \\
& \times (\underbrace{3}_{Position_{xyz}} + \underbrace{3}_{Normal_{xyz}} + \underbrace{2}_{TexCoord_{uv}}) \times (\underbrace{4\ bytes}_{Size\ of\ float})
\end{aligned}
\tag{5.4}
$$

The total amount of video memory consumed by an impostor in the Geopostor system ranges from a minimum of 2.52 MB or 5.02 MB (depending on the size of the elevation texture) to 20.15 MB. The total memory consumed by the corresponding mesh representation is 2.063 MB. Therefore, a single human model in the Geopostor system requires 4.583 MB to 22.583 MB of video memory for both representations (see Table 5.7). It should be noted that the consumption of video memory by the human's color map is not considered as a result of being one-dimensional textures and therefore have negligible memory usage.

With a memory budget e.g. 25%, on a graphics card with 256 MB of video memory for the Geopostor system, this would allow 64MB of video memory to be used by the system for storing the different human models or animations. In the case of using the larger impostor textures, this would only allow almost 3 different animated human models to be stored or

LOD	Number of frames of animation	Memory Consumed by VBO	Memory Consumed by Texture Maps (with mip-mapping)	Total Memory Consumed
Impostor (17×8 Viewpoints)	10	0.15MB	20.0MB	20.15 MB
Impostor (17 Viewpoints)	10	0.02MB	2.5→5.0MB	2.52→5.02MB
Mesh	10	1.98MB	0.083	2.063 MB

Table 5.7: The total amount of video memory consumed by a single human model's impostor and mesh representation.

a single human model with 3 different animations. This is greatly improved by using the smaller elevation textures where between 9 and 14 different human models can be stored or between 3 and 4 human models with 3 different animations each (depending on the elevation at which the impostors are being shown at). It should be noted that recent graphics cards (e.g., ATI Radeon X1900 XTX, NVIDIA Geoforce 7800 GTX) and next generation console architectures such as the Playstation 3 feature 512 MB of video memory allowing the number of human model types to be doubled for the same memory budget percentage.

5.1.5 Geopostor System's Bottlenecks

The Geopostor system was tested using NVIDIA's *NVPerfKit* [NVIa] to establish how the system is using the GPU and to identify performance issues. NVPerfKit allow graphics application developers gain access to low-level performance counters inside the driver and GPU [NVIb]. Since a NVIDIA graphics card is needed to use NVPerfKit, the tests were performed using a PentiumIV 3.4Ghz processor, with 1.0GB RAM and an NVIDIA 6600 GT graphics card with 128MB of video memory. The same OpenGL environment (as described in Section 5.1.1) was populated with 1,000 virtual humans using the 5 resolution meshes and the impostor. All representations were rendered using Gouraud shading implemented in hardware. The frame rate (Table 5.8) and various GPU counters (see Figure 5.9 (a) and (b)) were recorded.

Ideally, the percentage of time the GPU is idle should be as small as possible as the GPU is otherwise being wasted. This is a case for all of the mesh representations, indicating that the bottleneck is GPU and not CPU (see GPU Idle % in Figure 5.9 (a)). However, GPU

Idle % is higher for the impostor suggesting that it is partially CPU-limited as a result of rendering each billboard using a separate API call.

LOD Geometric Representation	100% Mesh	60% Mesh	45% Mesh	30% Mesh	15% Mesh	Impostor
FPS	12	20	22	42	83	331

Table 5.8: Frame-rate of the geometric representations used in the Geopostor system measured using NVPerfKit.

In the case of all the mesh representations, the percentage of time that vertex shader unit 0 was busy (i.e., Vertex Shader Utilization %) accounted for a large percentage of the GPU time while the percentage of time that pixel shader unit 0 was busy (i.e., Pixel Shader Utilization %) was relatively small (see Figure 5.9 (b)). This indicates the meshes are vertex/geometry bound. The number of unique vertices transformed by the geometry (Vertex Count) is approximately three times the number of the triangles processed in the geometry subsystem (Triangle Count) (see Figure 5.9 (b)). The meshes' performance could be improved by using triangle strips or fans, since the GPU's post T&L cache would reuse the transformation of cached vertices and therefore reduce the overall number of vertices transformed. An interesting result is that the test involving the 45% mesh resolution only achieved 22 frames per second (see Table 5.8). This is possibly due to VBOs under a certain number of triangles become CPU-limited [Wlo03].

Due to the impostor requiring more per-pixel operations and its geometry consisting of only two triangles, the impostor's Pixel Shader Utilization percentage was substantially higher and its Vertex Shader Utilization percentage was substantially lower, indicating that, unlike the mesh representations, it is pixel-bound and not vertex/geometry bound. Additionally, the per-pixel operations are also applied to the background of the impostor textures resulting in a higher Pixel Count (see Figure 5.9 (b)). The percentage of time the pixel shader is stalled by raster operations (ROP) is significantly higher due to alpha testing the impostor's pixels (see ROP Stalls % in Figure 5.9 (a)). As expected, the percentage of time that pixel shader unit was stalled waiting for a texture fetch (i.e., *Shader Stalls* %) was high for the impostor in comparison to any of the meshes. Since the textures do have mipmips and do not use anisotropic filtering, this suggests that there is bad coherency when accessing textures.

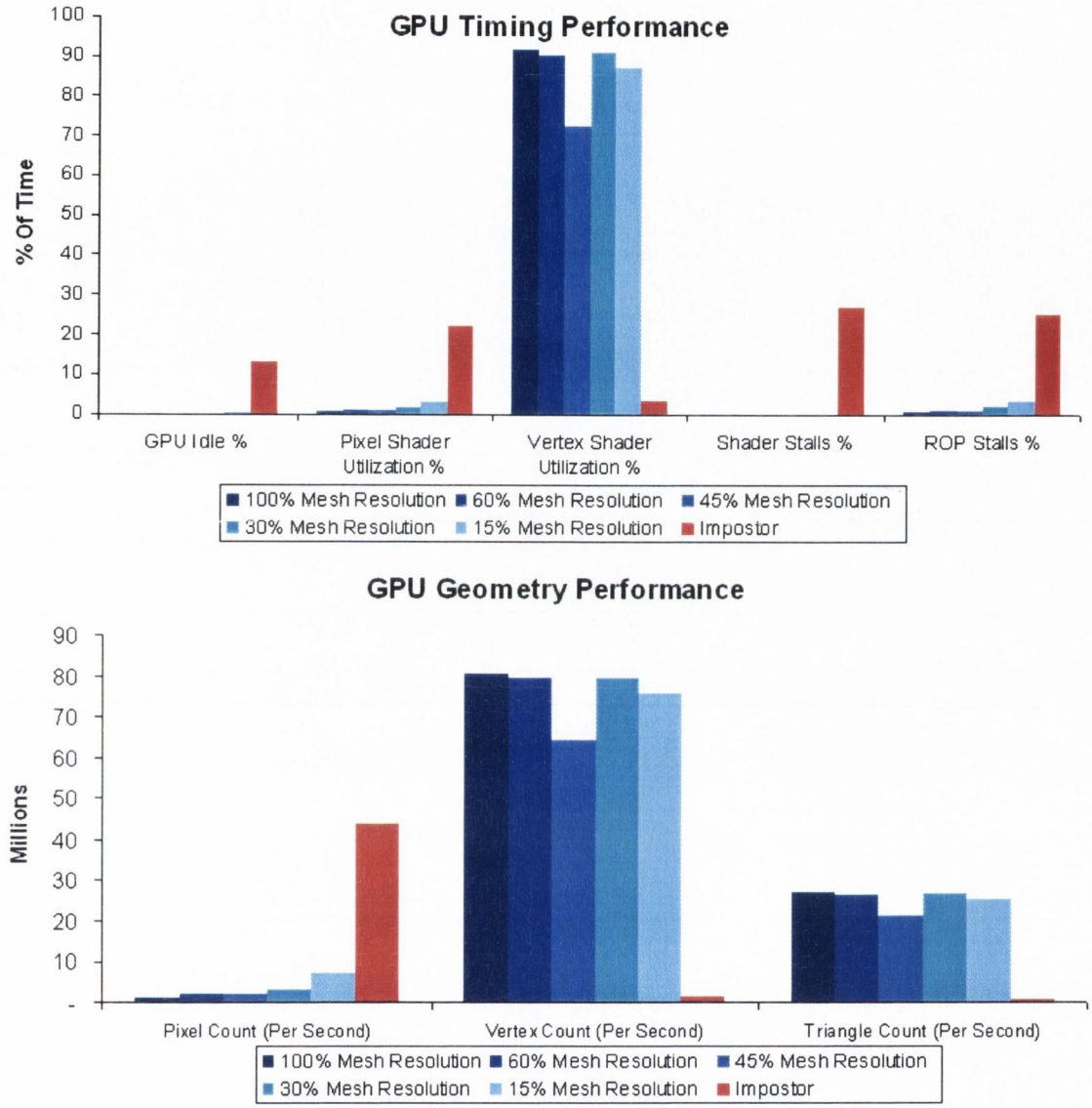


Figure 5.9: (a) GPU timing performance of the Geopostor system. (b) GPU Geometry Performance of the Geopostor system.

5.2 Performance of Crowd System in an Urban Simulation System

The city application contains over 100,000 polygons and uses over 200MB of texture data, and so provides a good real-world application in which to test our crowd system. The aim of these tests is to compare the number of humans that can populate the city at a real-time frame rate based on the virtual human's geometric, animation, and variation LOD. To provide a baseline, we recorded the frame rates during a 2000 frame walkthrough of the virtual city containing no humans and averaged the frame rate every 20 frames (Figure 5.10(a)).

Instead of populating the environment with various sized crowds until the frame rate drops below a certain threshold, we precalculate the maximum number that could populate the environment at this threshold for each virtual human representation LOD. Since only small parts of the environment are inhabitable (i.e., pavement and grass areas), populating the city by randomly positioning the virtual humans in these areas (using the walk-maps discussed in Section 4.2.1) results in an uneven distribution of humans per frame. Therefore, bottlenecks will occur at particular frames when large inhabitable areas are within the view frustum. To precalculate where these bottlenecks will occur, the number of humans drawn at each frame was recorded during the walkthrough of the city populated with 10,000, 25,000, 50,000, 75,000 and 100,000 virtual humans (Figure 5.10(b)). It should be noted that the random number generator was seeded with the same number before each test so that the position of the humans was constant amongst further tests.

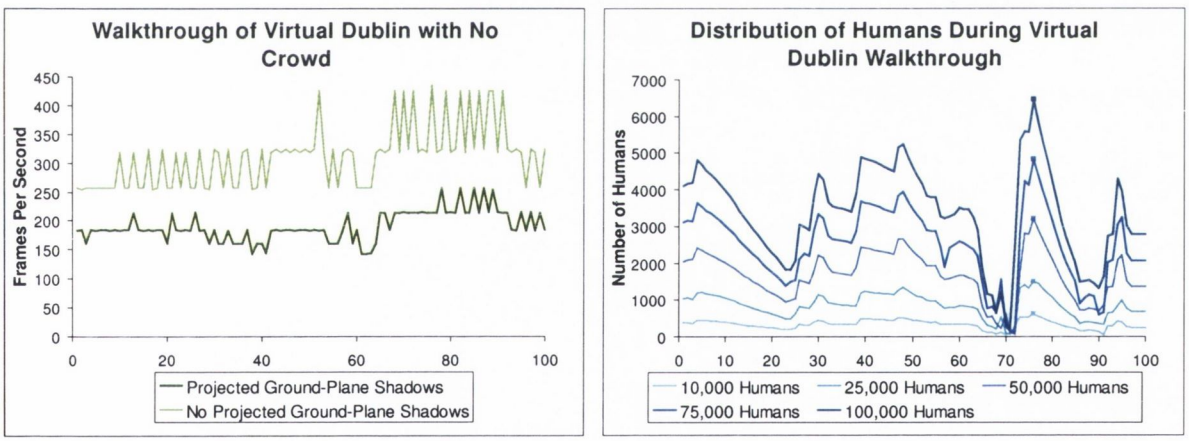


Figure 5.10: (a) Walkthrough of Virtual Dublin containing no crowds. (b) Distribution of humans within Virtual Dublin populated with 10,000, 25,000, 50,0000, 75,0000 and 100,000 humans

From these tests, the maximum number of unoccluded humans within the view-frustum

occurred at frame 76 of each populated walkthrough (see Figure 5.10(b)). To estimate the maximum number of humans (Max_H) that can be displayed at this frame so that the frame rate does not drop below $\text{FPS}_{Threshold}$, we utilize the various LOD rendering costs (Cost_{LOD}), calculated in Section 5.1, in Equation 5.5. From Figure 5.10(a), the frames per second at frame 76 (FPS_{76}) of the walkthrough populated with no virtual humans is 212. To estimate the number of humans that need to populate the city (City_{Pop}) so that there is Max_H at frame 76, we can interpolate this value from the distribution of humans in the various walkthroughs in Figure 5.10(b).

$$Max_H = (1 - \frac{FPS_{Threshold}}{FPS_{76}}) \times (\frac{1}{Cost_{LOD} \times FPS_{Threshold}}) \tag{5.5}$$

From Section 5.1, since a deformable mesh at the highest LOD for variation had no affect on the rendering cost (where $\text{Variation}LOD = 3$ in Table 4.2), five walkthroughs using a mesh resolution of 100%, 60%, 45% and 15% were recorded with 10 template meshes. Virtual humans using a particular template mesh had a choice of 100 outfits, performed the default walk animation ($\text{Animation}_{LOD} = 0$) and did not move in step with each other to allow for varied crowd motion. In the case of the crowd's lowest LOD representation, the impostor's rendering cost varied depending on its Variation_{LOD}. As Variation_{LOD} increases, so does texture memory consumption. Since the environment has its own texture memory requirements, the chances of texture thrashing occurring increases with these levels of detail. To test this, four walkthroughs of the city populated with impostors were recorded using the following crowd levels of detail:

- **Impostor Crowd Test 0** ($\text{Variation}_{LOD} = 1$, $\text{LOD}_{Animation} = 0$):
  1 template model with 100 outfits with uniform crowd motion
- **Impostor Crowd Test 1** ($\text{Variation}_{LOD} = 2$, $\text{LOD}_{Animation} = 0$):
  1 template model with 100 outfits with varied crowd motion
- **Impostor Crowd Test 2** ($\text{Variation}_{LOD} = 3\text{a}$, $\text{LOD}_{Animation} = 0$):
  10 template models with 100 outfits with uniform crowd motion
- **Impostor Crowd Test 3** ($\text{Variation}_{LOD} = 3\text{b}$, $\text{LOD}_{Animation} = 0$):
  10 template models with 100 outfits with varied crowd motion

To evaluate each representation's impact on the frame rate of these walkthroughs, so that we only take the rendering cost into account, the virtual humans did not perform col-

lision avoidance with the environment's static objects and walked on the spot. However, $FPS_{Threshold}$ was set to 35 frames per second to allow for collision avoidance in future tests without the frame rate becoming non-interactive. Table 5.9 summarizes each representation tested, and the estimated number of humans needed to populate the city, so that there are MAX_H at frame 76. It should be noted that, in these tests, the virtual humans were both occlusion and frustum culled, and lighting was enabled. The results for these walkthroughs are in Figure 5.11 and Figure 5.12, and show that the size of the crowd can be dramatically increased at interactive frame rates when using the impostor representation.

Crowd Model	$Variation_{LOD}$	$Animation_{LOD}$	$Cost_{LOD}$(ms)	Max_H	$City_{Pop}$
High Res Mesh	3	1	0.064526	369	6,000
Low Res Mesh 4	3	1	0.029238	816	13,000
Low Res Mesh 3	3	1	0.025209	946	15,000
Low Res Mesh 2	3	1	0.018985	1,257	20,000
Low Res Mesh 1	3	1	0.011160	369	35,000
Impostor Test 0	1	0	0.005617	4,250	67,000
Impostor Test 1	2	0	0.006458	3,696	58,000
Impostor Test 2	3a	0	0.006141	3,887	61,000
Impostor Test 3	3b	0	0.007339	3,252	51,000

Table 5.9: Population of the city for each LOD representation so that the frame rate does not drop below a threshold of 35fps.

To compare these results with our Geopostor system's switching approach, four walkthroughs were performed using the same settings as the tests involving the impostors (Figure 5.13). The selection criteria used for switching between representations was the hypothesised pixel to texel ratio of 1:1 and the number of impostors and meshes drawn at each frame is shown in Figure 5.14(a) and (b), respectively. The results show that our system has no

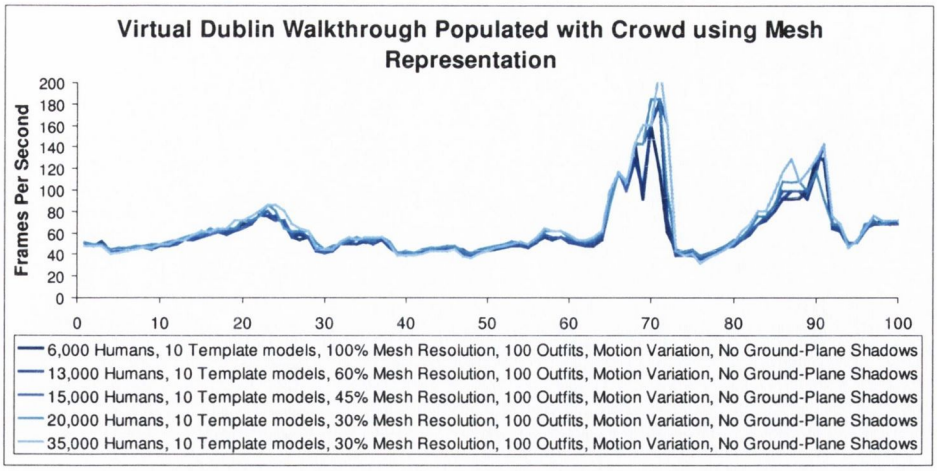


Figure 5.11: Walkthrough of Virtual Dublin containing crowds using different mesh resolutions.

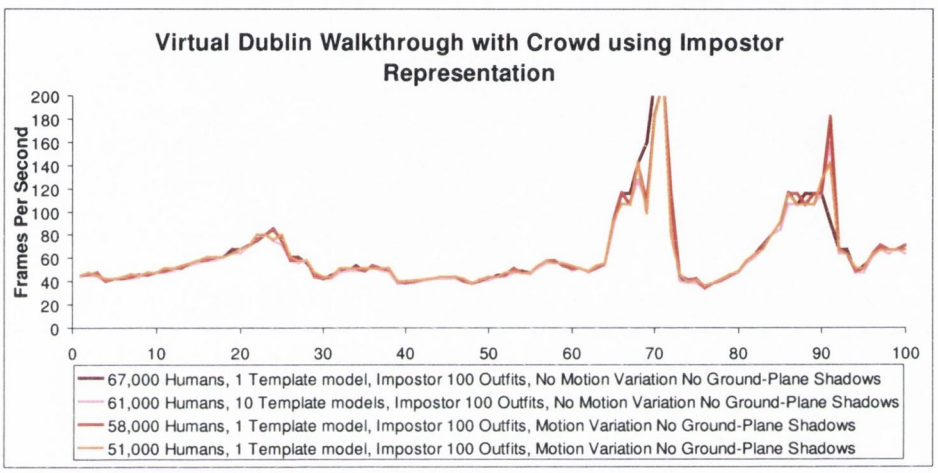


Figure 5.12: Walkthrough of Virtual Dublin containing crowds using the impostor model.

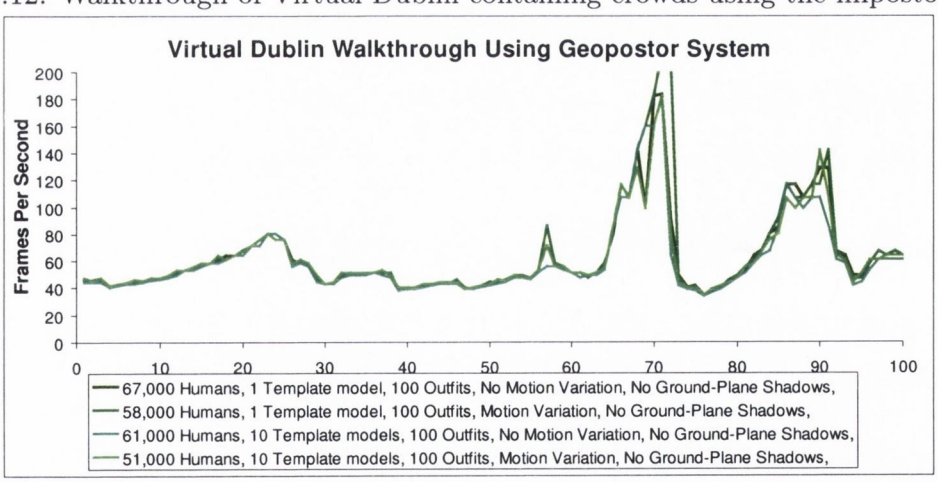


Figure 5.13: Walkthrough of Virtual Dublin using the Geopostor system

significant impact on the frame rate in comparison to the various walkthroughs using solely impostors. It was also found that texture thrashing did not occur, even at the highest LOD for variation. However, this is due to the fact that the graphics card had 256MB of memory.

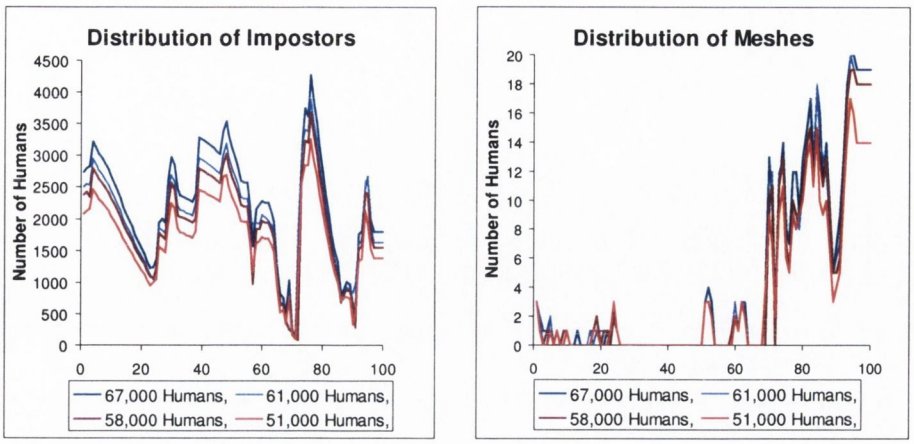


Figure 5.14: (a) Number of impostors drawn each frame. (b) Number of meshes drawn each frame.

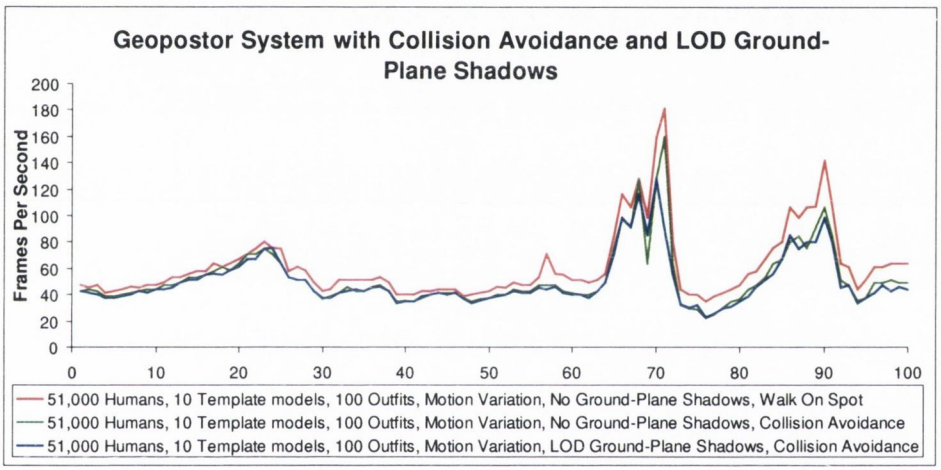


Figure 5.15: Geopostor system with collision avoidance and projected ground-plane shadows.

Two more tests were performed to measure the effect of collision avoidance and projected ground-plane shadows on the frame rate. For the first test, the virtual human no longer walked on the spot but navigated through their environment by performing collision avoidance with static objects, using the method described in Section 4.2.1. The Geopostor system was used in this test, where $\text{LOD}_{Variation}$ was set to a level of 4 with 10 template models. Finally, the effect of projecting the virtual human's shadow onto the ground-plane was tested. In this test, all of the settings were the same as above. A LOD approach was utilized, whereby

each virtual human's $LOD_{Geometry}$ representation was rendered into the stencil buffer only if it was within an arbitrary distance of 65 meters from the camera which was selected based on preliminary observations. Perceptual experiments to evaluate the optimal shadow distance could be carried out as future work. From the recorded results (see Figure 5.15), collision avoidance and rendering projected shadows can be performed using our system while maintaining the application's interactivity.

5.2.1 Minimizing Texture Thrashing Evaluation

Finally, to test the effect of splitting the impostor textures into separate elevation textures (see Section 4.2.3) as opposed to using the original textures of 1024*1024 pixels, we recorded the frame rates during a 2000 frame walkthrough of the virtual city, containing 10,000 individuals, and averaged the frames per second every 40 frames. The walkthrough used was different to the previous tests. Since we want to test how the size of the pre-generated impostor texture affects texture thrashing as the number of different template impostor models increases, all of these tests were performed using a GeForce 4 Ti4600 3D card with 128MB of video memory instead of the 256MB card used in other test. Additionally, the buildings in the urban environment were rendered without textures to avoid texture thrashing due to the city's models. These tests were run with 4, 6, 8, and 10 different human models for both types of textures. As shown in Figure 5.16, once the number of human models exceeds four, texture thrashing becomes a problem when using the original textures. However, up to ten human models can be used without texture thrashing affecting the frame rate when the impostor textures are split into separate elevation textures.

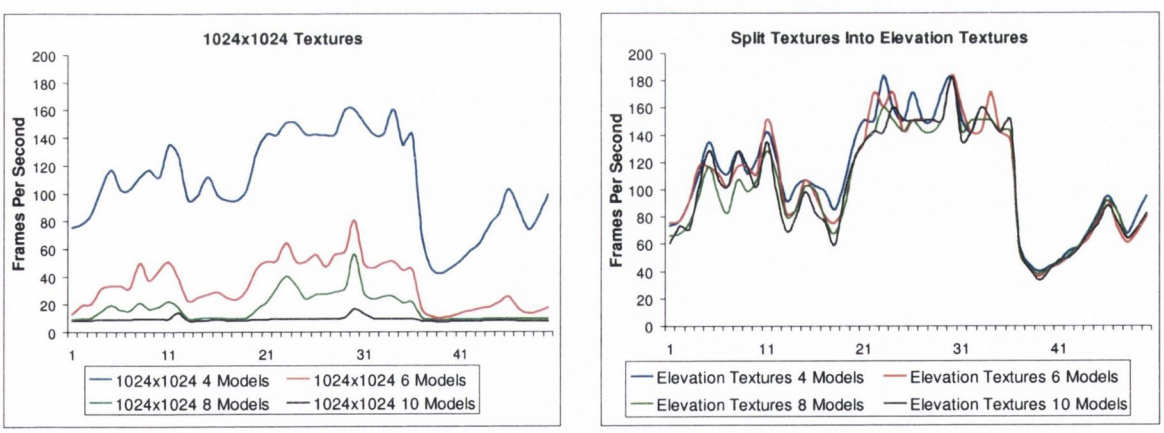


Figure 5.16: Frame rates using 1024x1024 textures Vs. using elevation textures

5.3 Perceptual Evaluation of Virtual Human Representations

Usually developers of real-time crowd systems decide on the virtual human representation they will use based on three factors: the size of the crowd being rendered, each representation's rendering cost and its visual appeal. While there has been extensive research on the numerous ways of graphically representing virtual humans (including their associated rendering cost), there has been no research conducted on perceptually evaluating them. However, evaluating these representations based on the plausibility of visual appearance and motion would provide a useful metric to help developers of LOD-based crowd systems improve its visual realism while maintaining real-time frame rates. With regards to improving our crowd system, we carried out perceptual evaluation experiments on various virtual human representations using experimental procedures from the area of *psychophysics*.

5.3.1 Psychophysics

Psychophysics is the science of human sensory perception and is used to explore two general perceptual problems involving the measurement of sensory thresholds: discrimination and detection [LHEJ01]. Discrimination is the ability to tell two stimuli apart, where each differ by a small amount, usually along a single dimension. Detection is a special case of the discrimination problem, where the reference stimulus is a null stimulus. Typically, both perceptual problems can be investigated using either a classical *yes-no* or a *forced choice* experiment design [Tre95]. A yes-no design involves experiment participants deciding on whether the stimuli are the "same" (no response) or "different" (yes response) while forced choice designs consist of the participant identifying a specific target stimulus given a number of choices. Using these designs, the participants responses for each stimulus level can be collected and analyzed to estimate discrimination or detection thresholds. In order to measure these thresholds, the participant's cumulative responses are plotted as a graph of *percentage yes* responses (using a yes-no design) or *percentage correct* responses (using a forced choice design) for each stimulus level. An S-shaped curve termed a *Psychometric Function* is fitted to the cumulative responses, where the percentage yes or percentage correct is plotted as a function of stimulus.

For a yes-no design, the sensitivity threshold is specified by the stimulus intensity required for a person to reach a 50% yes point i.e., the point where same and different responses are equally likely. This threshold is known as the *Point of Subjective Equality* (PSE). For this design, a simple Ogive inverse normal distribution function (see Equation 5.6) can be use to plot a curve that fits the participant's data (shown in Figure 5.17) and, from this curve, the PSE can be estimated as the 50% point and calculated using Equation 5.7. The inverse

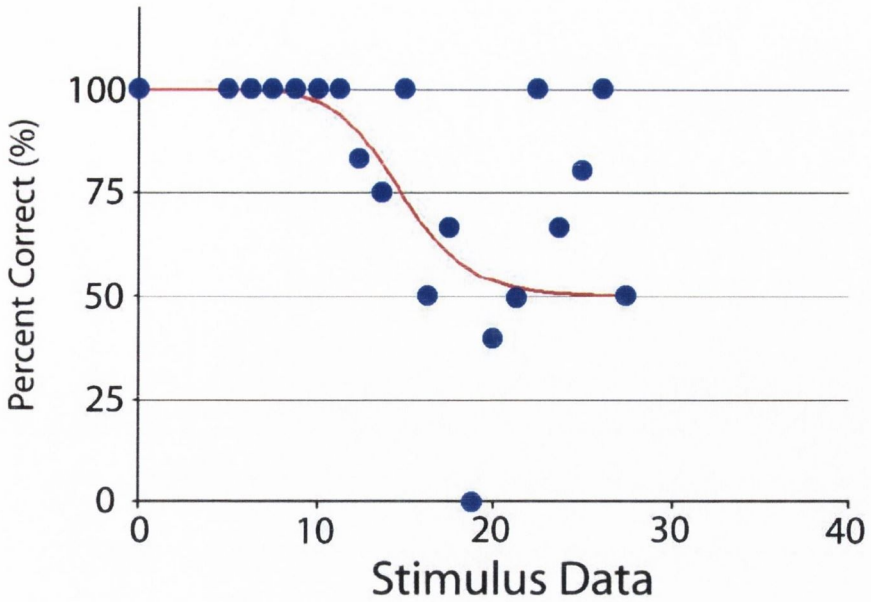


Figure 5.17: An Ogive function fitted to a participant's data for a yes-no design.

normal distribution function computes the stimulus intensity (x) for a given probability (P).

$$P_{Ogive}(x) = \frac{1}{\sigma\sqrt{2\Pi}} \exp \frac{-(x-\mu)^2}{2\sigma^2} \tag{5.6}$$

$where:$ σ is the mean,
μ is the standard deviation, and
μ^2 is the variance.

$$PSE_{Ogive} = P_{Ogive}(0.5) \tag{5.7}$$

For forced choice designs, the threshold is often chosen as a halfway point between chance and 100% correct [Tre95]. For example, for a two alternative forced choice (2AFC) paradigm, the target stimulus is one of two choices. Therefore, the sensitivity threshold is the midpoint between chance (50% point in the case of 2 choices) and 100% correct, which is the 75% point. For experimental data using a 2AFC paradigm, a logistical function is normally used to fit a suitable curve to the participant's data and estimate the PSE using the 75% point. In our experiments we use a slightly modified version of the logistical function (given in Equation 5.8). The PSE for an experiment using a 2AFC design can be calculated using Equation 5.9

$$P_{Logistic}(x) = 1 - \gamma(\frac{1}{1 + (\frac{x}{\alpha})^{-\beta}}) \tag{5.8}$$

$where:$ α is the stimulus at the halfway point,
β is the steepness of the curve, and
γ is the probability of being correct by chance.

$$PSE_{Logistic} = P_{Logistic}(0.75) \tag{5.9}$$

Another interesting threshold that can be estimated from these curves is the *difference threshold* or the *just noticeable difference* (JND). The JND is the smallest difference in intensity required for a person to distinguish two stimuli and this can be estimated as the amount of additional stimulus needed to increase a participant's detection rate from 50% to 75% (for a yes-no design) or from 75% to 87.5% (for a 2AFC design) on the fitted psychometric function. Equation 5.10 and Equation 5.11 are used to calculate the JND for a experiment using a yes-no and 2AFC experiment, respectively. Finally, ANalysis of Variance (ANOVA) is used to test the null hypothesis that two means are equal. The null hypothesis is rejected if there are significant differences between the means.

$$JND_{Ogive} = P_{Ogive}(0.75) - P_{Ogive}(0.5) \tag{5.10}$$

$$JND_{Logistic} = P_{Logistic}(0.875) - P_{Logistic}(0.75) \tag{5.11}$$

The main problem with measuring thresholds of perception is that participants do not always respond in the same way when presented with identical stimuli in an ideal, noise-free experimental setup. This is mainly due to the fact that the neurosensory system is somewhat noisy, but other reasons such as attentional differences, learning, and adaptation to the experimental setup can also have an effect. To reduce some of these problems, many psychophysical techniques for collecting data have been developed [Tre95]. With regards to our experiments, we use a staircase experimental procedure.

A simple *up-down staircase* procedure involves setting the stimulus level to a pre-defined intensity and presenting the stimulus to the participant [Cor62, Lev71]. Depending on the participant's response, the stimulus level is decreased (for a positive response) or increased (for a negative response) by a fixed amount or *step-size* and the altered stimulus is presented to the participant again. The experiment is terminated once the participant's response changes from positive to negative and vice versa (called a *reversal*) a certain number of times. Figure 5.18

illustrates the stepping procedure for an up-down staircase terminated after four reversals. It should be noted that care is needed when selecting the step-size. Too large a step-size results in inaccurate threshold estimates and the possibility of *outliers* in the data. Alternatively, too small a step-size may result in an accurate threshold estimate but the risk of participants becoming bored, tired or losing their attention is high. Normally, the appropriate step-size is selected based on the results from preliminary experiments testing several different step-sizes.

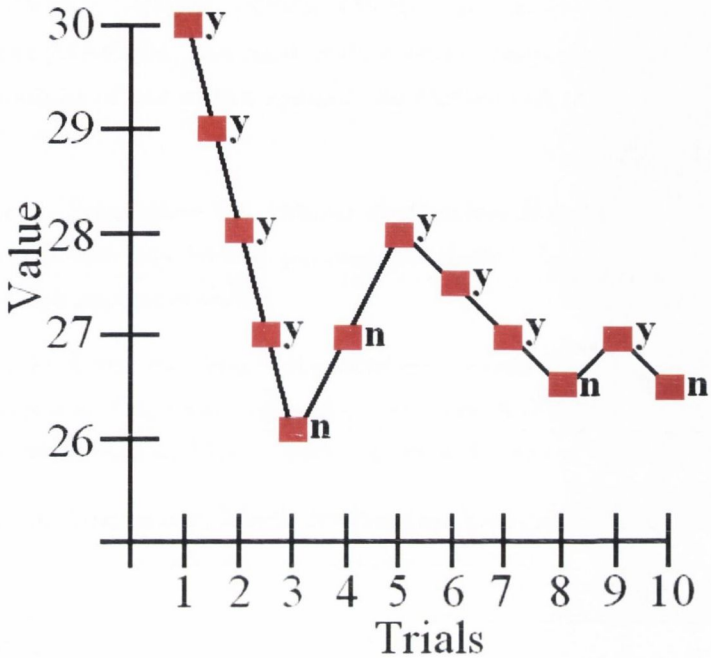


Figure 5.18: Example of the stepping procedure for an up-down staircase terminated after four reversals.

To eliminate response bias caused by participants learning how the experimental procedure works, a pair of randomly interleaved staircases can be used [ODGK03]. This involves setting up *ascending* and *descending* staircases, where their respective stimulus level is initialised to a maximum and minimum intensity. These two staircases are then presented to the participant in a randomly interleaved manner to eliminate the participant guessing the direction of change of the stimulus intensity. To avoid data being sampled at too high or too low stimulus levels, adaptive procedures can be used to specify how to adapt the stimulus level depending on the participant's response. As a result of this, data sampling is concentrated around the participant's threshold on the psychometric function. Levitt provides an overview of adaptive staircase procedures [Lev71] such as the *transformed up-down* method and the *weighted up-down* method. With transformed up-down methods (used in [MAEH04]), the stimulus is altered depending on the outcome of two or more preceding trials. For example, a

three-up one-down (3U-1D) stepping procedure involving the stimulus level is increased only after three successive incorrect responses and decreased with each correct response. With weighted up-down methods, different step-sizes for upward and downward steps are used.

While there has been little previous work related to the perception of virtual human representations, research using psychophysical experiment techniques has been conducted on the perception of other computer graphics related areas such as the animation of virtual humans [HOT98] [WB03] [HRD04] [RP03] [OHJ00], sensitivity to head tracking latency in virtual environments [MAEH04] and rigid-body collision responses [ODGK03]. With the goal of improving the realism of our crowd system, we carried out the following three perceptual experiments:

1. **Experiment 1: Impostor Vs. Mesh Detection Experiment**
   At what distance can experiment participants detect that a virtual human is using an impostor or mesh representation?

2. **Experiment 2: Low Vs. High-Resolution Mesh Discrimination Experiment**
   At what distance and at what resolution can experiment participants discriminate between a high resolution and low resolution mesh representation?

3. **Experiment 3: Impostor/Mesh Switching Detection Experiment**
   At what distance can experiment participants detect an impostor switching to a mesh?

5.3.2 Experiment 1: Impostor Vs. Mesh Detection Experiment

Aim

While a pre-generated impostor is significantly faster to render than the corresponding mesh, its main aesthetic problem is that, once the impostor is close to a viewer, certain artefacts are quite noticeable and the viewer is able to perceive the difference between the two representations, especially when displayed side-by-side. These artefacts may be caused either by aliasing, loss of depth information, or using a fixed number of pre-generated viewpoint images.

In this experiment, we aimed to establish the distance at which a virtual human's pre-generated impostor is perceptually equivalent to its mesh. In order to establish this distance threshold, we simultaneously presented two virtual humans using the impostor and mesh model at various distances to the experiment participant, and tested the participant's ability to detect which virtual human was using which representation. By recording the participant's responses at each distance the virtual humans were displayed at and plotting a psychometric function to this data, this distance threshold was estimated from the fitted curve using

the PSE. This PSE signifies the distance at which the participant is likely to choose either representation with equal likelihood, and therefore provides a good estimate to the distance at which the impostor is perceptually equivalent to the mesh (for that person).

The goal in establishing such a threshold was to provide us with a guide to the distance at which both representations could be displayed in our system without a user detecting the impostor. This distance can be calculated in terms of a pixel to texel ratio, and it was hypothesised that beyond the point of one-to-one pixel to texel ratio (see Section 4.1.3), the participants would be unable to detect the impostor. Often it is difficult to find the exact experimental parameters and variables for a staircase procedure (such as step-size, maximum and minimum stimulus levels etc.). A study was first carried out using a weighted up-down experimental procedure followed by a second study on a different set of participants using a transformed up-down experimental procedure. In order to validate and fine tune results found in the first study, which produced approximate threshold ranges, we exploited our earlier findings to find the exact pixel to texel ratio in the second study.

Apparatus and Participants

The equipment used was a high end commodity PC with an NVidia GeForce graphics accelerator card. For the first study, a 19-inch monitor was used, while a 21-inch monitor was used for the second study. Both monitors were at a resolution of 800x600 pixels with a screen refresh-rate of 85 Hertz.

Eleven participants (2 females, 9 males, aged between 22 and 39) took part in the first study, while 38 participants (13 females, 25 males, aged between 17 and 35) took part in the second. All participants were drawn from the staff and students of the authors' institution, had normal or corrected to normal vision and were both familiar and unfamiliar with graphics. All of the experimental participants were positioned approximately 28"-30" from the screen at zero elevation, so the full display subtended a visual angle of approximately 26°. User input for the experiments was provided by a USB gamepad featuring two trigger buttons to allow the participant to make their selection.

Visual Content and Procedure

An OpenGL test application was used to present the experimental stimuli to the participants. The experiment environment consisted of a black grid with a white background (for the ground plane). The 3D world was configured for a standard 45° field of view of the environment. All representations in the test application were lit by a directional light source pointing towards them and positioned directly behind the camera.

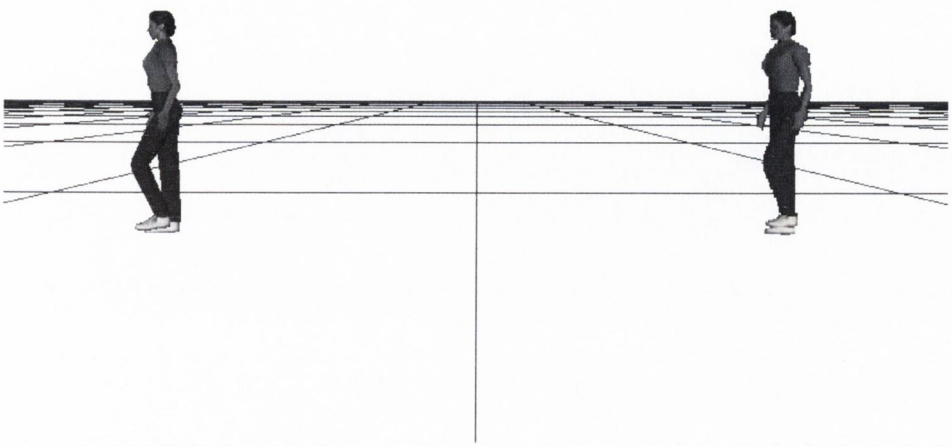


Figure 5.19: Virtual human impostor Vs. mesh detection experiment.

The participants were shown the mesh (described in Section 4.1.1) and impostor model (described in Section 4.1.2) side by side at different distances from the participant (as shown in Figure 5.19) for 5 seconds. The virtual humans were separated by a fixed number of screen pixels to keep the distance between the representations constant. Both representations were animated with the same one second walk-cycle consisting of one keyframe of animation every 100 milliseconds. Since applications containing virtual humans would typically involve displaying them from multiple viewpoints, both virtual humans were rotated at 5.625 degrees every 100 milliseconds in a randomised direction around the y-axis so that the participant was not comparing the representations based on a single viewpoint and therefore eliminating directional bias. It should be noted that all models were displayed in grey-scale, as in these experiments we wished to determine people's ability to detect detail. Colour would complicate this issue by introducing further confounding factors, so we left this aspect for future examination.

This experiment employed a two alternative forced choice (2AFC) design, whereby the participant was asked to choose which virtual human "looked better" at each distance. We considered the virtual human using the mesh representation to be the "correct" response. Depending on the participant's response, the distance at which to display the virtual humans

was either increased (for a correct response) or decreased (for a incorrect response). To avoid the participant guessing the direction of change in the virtual humans' distance, we used randomly interleaved ascending and descending staircases. Each staircase terminated after twelve reversals, where a reversal occurred whenever the participant changed his response from correct to incorrect and vice-versa.

To concentrate the data sampling around the participant's distance threshold, an adaptive step-size was employed with a adaptive procedure. The set of experiments in the first study employed a 3:1 weighted up-down adaptive procedure. The initial step-size for each staircase was set to 2.5 units, and was halved after each of the first four reversals, resulting in a final step-size of 0.15625 units. The ascending staircase started at the initial distance of 5 units, and the descending staircase at an initial distance of 31 units. By using a 3:1 Weighted Up-Down procedure, this resulted in the virtual humans being moved closer by three times the current step-size for every "incorrect" guess, otherwise they were moved away by just the current step-size for each "correct" response. The second study employed a Three-Up, One-Down (3U-1D) stepping procedure i.e., each time the participant indicated 3 consecutive correct responses, the distance at which the virtual humans were displayed was increased by the step-size, otherwise one incorrect response caused the distance to decrease by the step-size. The initial step-size was set to 4 units, and after the first four reversals the final step-size was 0.25 units. The ascending staircase began with the virtual humans displayed at the furthest stimulus distance of 29 units, while the descending staircase started at the closest distance of 9 units.

Results

For each staircase, we recorded the participants' responses at each distance, as well as the distances at which the 12 reversals occurred. A participant's results were accepted if both staircases converged to approximately the same answer. To check this, we compared the minimum and maximum distance at which the last 4 reversals occurred for the ascending and descending staircase. If they did not overlap with each other, then the participant's data was considered to be diverging and therefore unusable. After eliminating any diverging results, the experimental data from nine out of the eleven participants and sixteen out of the thirty-eight participants experimental data converged properly for the first and second study, respectively.

The large amount of diverging data is thought to be a result of a flaw related to using a 2AFC task, whereby a series of lucky guesses at low stimulus levels i.e., when the representations are far away from the viewer, can erroneously drive the staircase to levels that are too low [Kle01]. On further analysis of this diverging data, it was discovered that the ascending

staircases were not able to recover from a string of lucky guesses. However, it was found that, for the majority of the descending staircases, the minimum and maximum distances of the last 4 reversals overlapped with the results of both staircases for the converging data.

A psychometric curve ranging from 100% to 50% was fitted to each participant's experimental data using Equation 5.8. Using Equation 5.9 and Equation 5.11, the PSE and JND for each study were calculated as the 75% and 87.5% levels on the curve, respectively. At this PSE, the participant will judge the representations with equal likelihood as looking better. The corresponding pixel:texel ratio values were calculated for the PSE and JND using Equation 5.12. The one-to-one pixel to texel ratio equivalent distance for the virtual human model used in this experiment is listed in Table 5.10 along with the two-to-one distance for comparison.

$$Pixel : TexelRatio(distance) = \frac{distance \times PixelSize}{TexelSize} \tag{5.12}$$

Pixel:Texel Ratio	Distance
1:1	11.0 units/meters
2:1	22.0 units/meters

Table 5.10: One-to-one pixel to texel ratio distances for the virtual human with a 45° Field of View at 800x600 Resolution

The mean PSE and JND for the 9 participants in the first study were 1.4 ± 0.142 and 0.492 ± 0.152 (see Figure 5.20(a)). The mean PSE showed that users perceived the impostor representation of human models at a distance greater than the hypothesized ratio (a pixel to texel ratio of 1.4:1). However, the mean JND is quite large indicating that the participants were not sensitive to small changes to the pixel to texel ratio at which the impostor was being displayed. The mean PSE and JND for the 16 participants in the second study were respectively 1.164 ± 0.064 and 0.1 ± 0.01 (see Figure 5.20(b)). The mean PSE is close to the hypothesized value of one-to-one, and this result represents an improvement on the first study. This change in results is attributable to learning from the first study's experimental results and adapting the psychophysical procedure accordingly. Since only one virtual human model was used for this experiment there were no means to compare, hence an ANOVA was not performed.

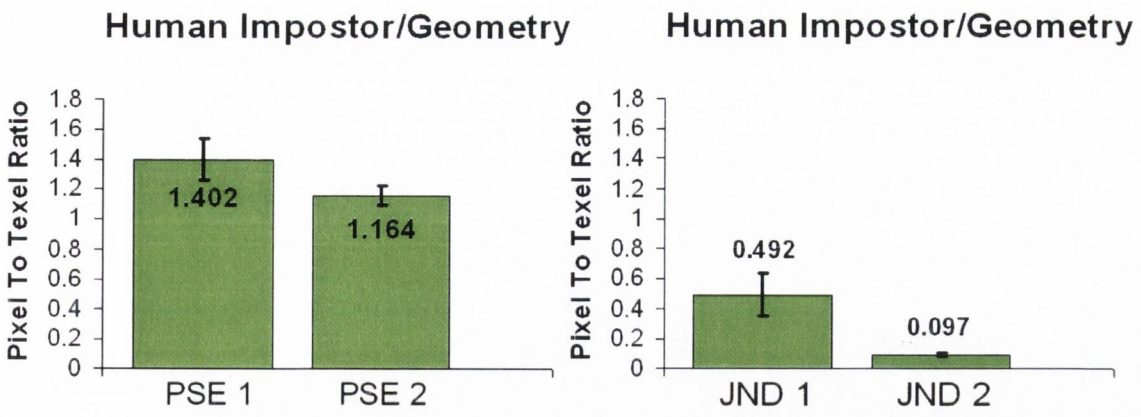


Figure 5.20: Results of the impostor Vs. mesh detection experiments (showing PSE and JND in terms of pixel to texel Ratio) using: (a) a 3:1 weighted up-down procedure (PSE 1, JND 1) and (b) a 3U-1D procedure (PSE 2, JND 2).

5.3.3 Experiment 2: Low Vs. High-Resolution Mesh Discrimination Experiment

Aim

A common LOD approach for reducing the computational cost associated with rendering a high detailed mesh, is to render a simpler mesh containing less triangles when the loss of detail will be imperceptible to the viewer of the system. However, care has to be taken in generating the low resolution mesh, as removing too much detail can result in blocky shaped meshes with animation artefacts caused by not enough joint vertices. Due to these artefacts, the overall visual realism of the virtual human is reduced.

In this second experiment, we aimed at establishing the resolution, in terms of both the percentage of vertices and the distance at which a virtual human's low resolution mesh is perceptually equivalent to a high resolution mesh. In order to establish this resolution threshold, we simultaneously presented each experiment participant with two virtual humans using the high and a low-resolution mesh at a particular distance, and tested their ability to discriminate whether the two resolution models were identical or not. The participant's responses for each low resolution mesh displayed were recorded, and a psychometric function was plotted to this data for each distance at which the virtual humans were displayed.

The goal in establishing such a threshold was to provide a guide to when a low-resolution mesh could be used in place of the high resolution mesh in our system without a user detecting the reduction of detail in the character's appearance.

Apparatus and Participants

The equipment used was a high end commodity PC with an NVidia GeForce graphics accelerator card. A 21-inch monitor was used at a resolution of 800x600 pixels with a screen refresh-rate of 85 Hertz.

18 participants (5 females, 13 males, aged between 17 and 28) took part in the second experiment. All participants were drawn from the staff and students of the authors' institution, had normal or corrected to normal vision and were both familiar and unfamiliar with graphics. All of the experimental participants were positioned approximately 28"-30" from the screen at zero elevation and so the full display subtended a visual angle of approximately 26°. User input for the experiments was provided by a USB gamepad featuring two trigger buttons to allow the participants to indicate their response.

Visual Content and Procedure

The same OpenGL test application as in the experiment in Section 5.3.2 was used to present the two virtual humans using the high resolution and a low resolution mesh to the participants for 5 seconds. The high resolution mesh consisted of 2170 triangles, and nineteen low resolution meshes were generated from this original mesh by hand. Solely using automatic simplification would result in losing important vertices needed to maintain the appearance of the model under motion, especially for the very low resolution models, which would subsequently bias the experiment's results. Therefore, the low resolution meshes were automatically simplified with manual intervention to keep the integrity of the really low resolution meshes. Using the 3D Studio MAX *multires* modifier, nineteen low resolution meshes were generated in this manner, ranging from a reduced vertex percentage of 60% to 15% at intervals of 2.5%. Preliminary observations were used for setting the appropriate range of resolutions for the low LOD mesh. A minimum vertex percentage of 15% was selected, as this was the maximum amount which the mesh's detail could be reduced while still retaining the general shape and motion of the virtual human's high resolution mesh. The corresponding number of vertices and triangles for each resolution model are shown in Table 5.11.

The participants were simultaneously shown two virtual humans side by side using the high resolution mesh and a low resolution mesh at 3 specific distances from the participant. The virtual humans were separated by a fixed number of screen pixels to keep the distance between the representations constant. As in the experiment in Section 5.3.2, both models were animated with the same one second walk-cycle and were rotated by 5.625 degrees every 100 milliseconds in a randomised direction around the y-axis, to eliminate directional bias. All models were displayed in grey-scale, as in these experiments we only wished to determine

Mesh Resolution	Vertex Percentage	Number of Vertices	Number of Triangles
High LOD	100%	1,383	2,170
Low LOD 18	60%	854	1,258
Low LOD 17	57.5%	819	1,204
Low LOD 16	55%	786	1,148
Low LOD 15	52.5%	754	1,096
Low LOD 14	50%	725	1,043
Low LOD 13	47.5%	688	990
Low LOD 12	45%	653	937
Low LOD 11	42.5%	617	881
Low LOD 10	40%	584	824
Low LOD 9	37.5%	553	773
Low LOD 8	35%	520	719
Low LOD 7	32.5%	489	665
Low LOD 6	30%	456	612
Low LOD 5	27.5%	417	559
Low LOD 4	25%	385	499
Low LOD 3	22.5%	345	453
Low LOD 2	20%	313	397
Low LOD 1	17.5%	278	352
Low LOD 0	15%	245	298

Table 5.11: Low resolution mesh details

people's ability to discriminate loss of detail.

This experiment consisted of 3 pairs of ascending and descending staircases randomly interleaved, where each pair displayed the virtual humans at one of the three distances from the participant. A yes-no design was employed, whereby the participants were asked to indicate whether the virtual humans looked the "same" (no response) or "different"(yes response) by pressing the respective left or right trigger button on a USB gamepad. For each staircase, a simple up-down stepping procedure was employed i.e., each time the participant indicated a "same" response, the resolution of the low LOD mesh was decreased by the step-size, otherwise a "different" response increased the resolution by the step-size. Each staircase ran for twelve reversals i.e., each time the participant's response changed. An adaptive step-size was used, where the initial step-size was only halved once after the first reversal for each staircase.

The ascending staircase began with the low LOD mesh displayed at the highest resolution of 60%, and the descending staircase started with the low LOD mesh displayed at the lowest resolution of 15%. The initial resolution step-size was set to 5% and, after the first reversal, the final step-size was 2.5%. As mentioned previously, 3 distances at which to display the representations from the viewer were chosen. The first distance was 5 units, and the other

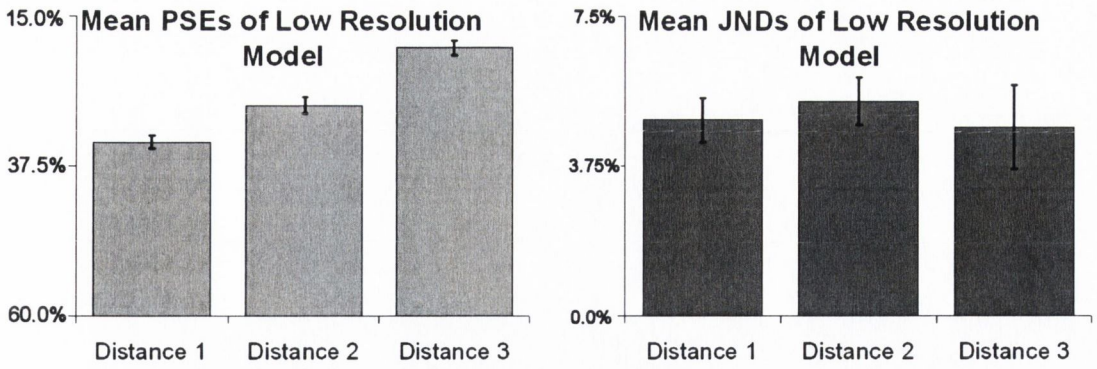


Figure 5.21: Mean PSE and JND values for low resolution models.

2 distances were calculated based on a percentage of the representation's initial screen-space size at the first distance. The second distance was 8 units and the third distance was 16 units which corresponded to 66.6% and 33.3% of the representation's initial screen space size.

Results

For each staircase, we recorded each participant's response for each mesh resolution displayed, as well as the resolution at which the 12 reversals occurred. We eliminated any diverging data using the same method as in Section 5.3.2. In this experiment, out of the 18 participants, 16 converged for the first distance, 13 converged for the second distance and finally 12 converged for the third distance.

Using the converging data, for each pair of staircases, we calculated the percentage of yes responses for each resolution displayed, and plotted this as a function of the resolution. For this experiment, a psychometric function based on Equation 5.6 was used to fit a curve to the data set and we subsequently calculated each participant's PSE and JND. The mean PSE and JND for each distance were calculated and are shown in Figure 5.21. The corresponding number of vertices and polygons for the PSE are shown in Table 5.12.

From this study, we found that, for this human mesh model, a low resolution mesh is perceptually equivalent to the high resolution mesh at a vertex percentage of 36.4% for a distance of 5 units, 31.7% for a distance of 8 units, and 22.5% for a distance of 16 units. A single factor ANOVA comparing the mean PSE averaged over 12 participants for the 3 distances, revealed a statistically significant difference between the mean PSE values for the 3 distances (see Table 5.13). This difference shows that distance affected perception of the low resolution mesh's visual appearance, with participants being able to discriminate better between different resolution meshes at closer distances. There was no significant difference

between the mean JNDs, which indicates that the same amount of stimulus change had to be added to the stimulus level at each distance in order for the participant to notice a difference. What is interesting about this result is that people were equally sensitive to the amount of vertex percentage difference, irrespective of distance.

Distance	5.0	8.0	16.0
PSE Vertex %	36.4±1.0	31.7±1.0	22.5±1.0
JND Vertex %	4.9±0.6	5.3±0.6	4.7±1.0
PSE Vertices	456	397	282
PSE Polygons	700	605	409

Table 5.12: Mean PSE and JND for low resolution geometry

PSE Comparisons	$F_{1,22}$	P
Distance 1 vs Distance 2	18.45	< 0.0005
Distance 1 vs Distance 3	103.28	≈ 0
Distance 2 vs Distance 3	24.54	≈ 0

Table 5.13: PSE comparisons for distance

5.3.4 Experiment 3: Impostor/Mesh Switching Discrimination Experiment

Experiment 3: Aim

Typically developers use the LOD approach of switching between a detailed mesh representation and a lower detailed model based on some selection criteria, to help maintain the interactivity of their system. It is important that the switching between models is imperceptible to the viewer, otherwise the overall believability of the system is reduced. While the selection of the model's resolution can be based on several switching criteria, usually this is based on some distance threshold from the viewer of the system. With respect to our system, we achieve interactive frame rates by using an impostor representation that can be displayed at a fraction of the rendering cost of the mesh and switch between these representations in order to maintain the realism of the crowd. While having thresholds for the believability of an impostor is useful when displayed beside its equivalent mesh representation, popping artifacts often manifest during the transition from impostor to geometry. These sudden popping artefacts during this transition may be caused either by differences in aliasing, depth information, or using a fixed number of pre-generated viewpoint images which can also cause shading differences.

In this experiment, we aimed to establish the distance at which the transition from a pre-generated impostor to a mesh is noticeable. In order to establish this distance threshold at which to switch, we presented a virtual human switching from the impostor representation to the mesh at various distances to each experiment participant, and tested the participant's ability to detect any popping artefacts. By recording the participant's responses at each distance the switch occurred and plotting a psychometric function to this data, this switching distance threshold can be estimated from the fitted curve using the PSE value. This PSE signifies the distance at which the participant is equally likely to notice or not notice the transition between representations, and therefore provides a good estimate to the distance at which such transitions will be acceptable. It should be noted that once the logistical function has been computed, other data points (for example, when people cannot notice 90% of the time) can be simply extrapolated.

The goal in establishing such a threshold was to provide us with a guide to the distance at which the switching between our impostor and mesh representation should occur in order to reduce any noticeable popping artefacts and therefore maintain the realism of our crowd. This distance can be calculated in terms of a pixel to texel ratio (see Section 4.1.3), and it was hypothesised that beyond the point of one-to-one pixel to texel ratio, the participants would be unable to detect the transition.

Experiment 3: Apparatus

The equipment used was a high end commodity PC with an NVidia GeForce graphics accelerator card. A 19-inch monitor was used, at a resolution of 800x600 pixels, with a screen refresh-rate of 85 Hertz. All of the experimental participants were positioned approximately 28"-30" from the screen at zero elevation and so the full display subtended a visual angle of approximately 26°. User input for the experiments was provided by a USB gamepad featuring two trigger buttons for the participants to indicate their response.

Experiment 3: Visual Content and Procedure

The same OpenGL test application as in the experiment in Section 5.3.2 was used to present the virtual human, switching between the impostor and mesh representation, to the participants. For each trial, the same model used in the first experiment was displayed, starting at a specific distance from the viewer, then moving at a constant speed towards the camera, and finally stopping at a specific distance. At some point during the interval the model switched from an initial impostor representation to a mesh representation. The virtual human was horizontally positioned at the center of the screen, animated with the same walk cycle used

in the other experiments, and again displayed in grey-scale.

A yes-no design was employed, whereby the participants were asked to indicate whether they noticed a "definite change" in the model, by pressing the left or right trigger buttons of the gamepad to indicate their respective yes/no response. The experiment consisted of a single pair of ascending and descending staircases randomly interleaved. For each staircase, a simple up-down stepping procedure was employed i.e., each time the participant indicated a "yes" response, the distance at which the switch occurred was increased by the step-size, otherwise a "no" response decreased the distance by the step-size. Each staircase ran for twelve reversals i.e., each time the participant's response changed. An adaptive step-size was used, where the initial step-size was only halved once after the first reversal for each staircase.

Two separate experiments were carried out, with the model either facing the user or spinning on the spot at a rate of 5.625° every 100 milliseconds in a randomised direction. For both experiments, the model started at a range of 36 units, and then moved at a speed of 6 units/sec toward the screen. The stopping point was a range of 1 unit from the screen. After the first four reversals, the final step-size was 0.3125 units. The virtual human switched from its impostor to its geometric representation at a switching distance ranging from 6 to 31 units.

The results of pilot experiments were used for setting the speed of the camera. It was found that, when the virtual human approached the camera too quickly, the resulting rate of change in the texture detail of the geometric representation (since mipmapping was not employed for its texture), caused the participants to perceive a switch where there was none. While the effect of popping artifacts may be reduced by blending, such as in Ebbesmeyer [Ebb98], we aimed to establish baseline thresholds were this would not be necessary. For urban simulations (which generally are constrained to the ground plane), transitions typically occur at the distance where the change in depth information is small due to perspective, and for virtual humans the overall change of depth information is similarly small. A further investigation of the effect of blending on transition detection is desirable.

Experiment 3: Results

For the first case, where the virtual human faced the viewer, there were seventeen experimental participants (13M-4F, ages 12-39), 10 of whose experimental data converged properly. For the second case, where the virtual human spun, there were 10 experimental participants (8M-2F, ages 12-39), nine of whose experimental data converged properly. All participants had normal or corrected to normal vision, and were both familiar and unfamiliar with graphics. For each staircase, we recorded the participants' responses for each trial's switching distance, as well as the distance at which the 12 reversals occurred. We eliminated any diverging data

using the same method as in Section 5.3.2.

A psychometric curve ranging from 100% to 50% was fitted to each participant's experimental data using Equation 5.8 where $\gamma = 0.5$. The mean PSE calculated (shown as PSE1 in Figure 5.22), was approximately the predicted one-to-one value with a small mean JND (shown as JND1), indicating that the participants were quite sensitive to subtle changes in the pixel to texel ratio at which the popping occurred. The mean PSE calculated for the second experiment (shown as PSE2), was less than for PSE1, suggesting that the spinning was a distracting factor. However, the differences were not significant for the PSE ($F_{1,17} = 1.46, p > 0.3$) or the JND values ($F_{1,17} = 0.22, p > 0.7$). The large number of diverging results in the first case, however, suggests that the participants noticed other artefacts, which were masked in the second case when the virtual human was spinning.

It should be noted that the results from this experiment are predicated on the texel size the impostor was pre-generated at. The texel size of the impostor used in this experiment was selected to ensure that all 17 by 8 pre-generated viewpoints fitted into a 1024 by 1024 image which is an image size commonly used in these type of applications. While the switching was not detected at a ratio of one-to-one for this texel size, it is hypothesised that this ratio will no longer be valid for impostors generated at a larger texel size due to aliasing artefacts being more noticeable. In order to establish at what texel size the switching is detectable at a one-to-one ratio, this would involve pre-generating impostors at various texel sizes, presenting a virtual human switching from each impostor to the mesh at the one-to-one distance, and evaluating at what texel size the participants is capable of detecting any popping artefacts.

5.4 Discussion of Evaluation Results

In this section, we will discuss the advantage and disadvantage of each virtual human LOD representation based on the results from Section 5.1 to Section 5.3. We hope that the following suggestions will provide developers of real-time crowd systems with a guide of when to use these LOD representations, in order to balance realism with interactivity:

1. **Crowd's Geometric LOD Representations:** At the highest level of geometric detail, the high resolution mesh is the most realistic in appearance. Additionally, since the mesh is deformed based on the underlying skeleton motion, this representation's level of detail for animation can be easily increased using various animation techniques (e.g., inverse kinematics, physically based animation, procedural animation etc...). However, this visual and animation detail comes at a great rendering cost and therefore this representation cannot solely be employed in real-time crowd simulations. This representation should only be used for simulating characters considered important in the

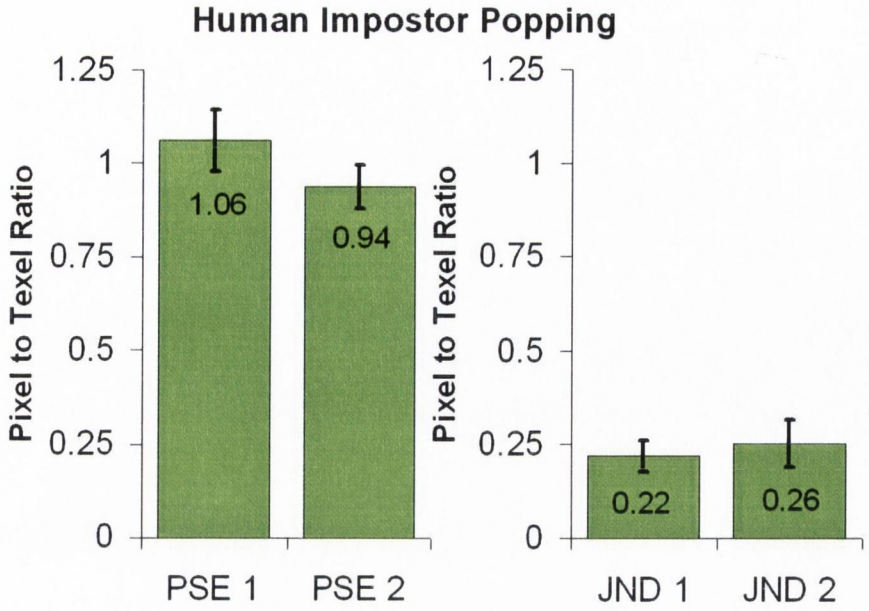


Figure 5.22: Results of the popping detection experiments (showing the PSE and the JND in terms of pixel to texel ratio) for humans facing viewer (1) and spinning (2).

application and not for characters in the background. For characters that are rated important but are far from the viewer, a lower resolution model should be used if the characters are required to perform a specific animation. Since the impostor's rendering cost is the least expensive, this representation allows the size of a simulated crowd to be greatly increased. However, virtual humans using this representation are limited to performing the animation used in the generation of the impostor's images. Therefore, impostor representations should be used for virtual humans equivalent to a film's scene extras, where their cyclical default animation could be masked by using the hybrid representation presented in Section 4.1.2, and more detailed characters should be used in the foreground of the scene.

2. **Crowd Shading Model:** While the rendering cost of a model can be reduced using constant shading, this results in a great reduction in visual realism, since the representation is not shaded. We suggest that constant shading could be used for virtual humans in a crowd that are far from the viewer of the application. However, switching between constant and Gouraud shading will cause popping artefacts due to the difference in colour, and perceptual experiments would need to be carried out to detect at what distance people notice the difference between the shading models. In the case

where the visual realism of the crowd is not important, we recommend using constant shading.

3. **Crowd Motion:** We consider uniform crowd motion to be quite disconcerting, and therefore suggest that it is important that individuals using the same template model should not move in step. While this has little or no effect on the rendering cost of the mesh representation, it comes at a greater cost for the impostor representation. However, we consider varied crowd motion worth the extra rendering cost in order to maintain the crowd's realism. Since current generation graphic cards and next generation consoles have at least 256MB of video memory, texture thrashing should not be a problem for several template models.

4. **Perceptual Factors:** In Section 5.3.2, it was found that an impostor and its corresponding mesh representation are not perceptually equivalent at a distance of less than a 1.16:1 pixel to texel ratio when simultaneously displayed side by side. Additionally, in Section 5.3.4, it was found that people could detect a virtual human switching between its impostor and mesh representation at a distance of less than a 1.04:1 pixel to texel ratio. This is lower than in the previous experiment, probably because the two representations are never compared side by side. These results provide developers with a metric of approximately a 1:1 pixel to texel ratio for the use of a pre-generated impostor representation. Also, these test results could be considered conservative, since the complexity of the test scene was extremely basic. We hypothesise that, in more complex scenes containing several hundred humans, switching between an impostor and a mesh representation could occur at a closer distance and we plan to test this hypothesis.

   Low resolution meshes can be generated that are perceptually equivalent to the high resolution mesh at particular distances (Section 5.3.3). Since these models consist of fewer triangles, the rendering cost of these resolutions is substantially less when compared to that of the original high resolution mesh. Using the results in Section 5.1.2, the associated rendering cost for each low resolution mesh that is perceptually equivalent to the high resolution mesh can be estimated (see Table 5.14). These results suggest that the high resolution mesh should be replaced in our system with a simpler model (depending on the distance from the viewer), since the extra detail in the high resolution mesh is not perceived by the viewer, and therefore is unnecessary. The results also suggest that we are using a mesh that is too detailed (2,170 triangles) for the highest LOD at a distance less than 5 units.

   It was also found in Section 5.3.2, that impostors are perceptually equivalent to the high resolution model at a pixel to texel ratio of approximately 1.16, which corresponds

to a distance of 12.416 virtual world units. However, low resolution meshes can be perceptually equivalent to their high resolution mesh at a closer distance. By using the results from Section 5.3.3, we can estimate the percentage of vertices at which to generate a low resolution mesh that is indistinguishable from the high resolution model at the same distance as the impostor. This corresponds to a low resolution mesh of approximately 27.5%. It should be noted that these rendering costs are taken from the tests where $\text{LOD}_{Variation}$ is equal to 3 (i.e., 10 template models using 100 outfits each with varied crowd motion). Due to the rendering cost of each model (see Table 5.14), we suggest that it would be advantageous to use the impostor instead of a low resolution mesh for virtual humans being displayed at a distance greater than the 1.16 ratio or acting as scene extras. The distances at which different LOD representations are perceptually equivalent to the highest resolution mesh is illustrated in Figure 5.23.

$\text{LOD}_{Geometry}$	Distance (units)	Cost_{LOD}(ms)	Crowd Size @ 30FPS
High Res 100%	< 5.0	0.0645	370
Low Res 36.4%	> 5.0	0.0206	1,615
Low Res 31.7%	> 8.0	0.0185	1,804
Low Res 27.5%	> 12.416	0.0163	2,044
Low Res 22.5%	> 16.0	0.0135	2,464
Impostor	12.416	0.00697	4,777

Table 5.14: The distance at which $\text{LOD}_{Geometry}$ models are perceptually equivalent and their associated rendering cost (where $\text{Variation}_{LOD} = 3$) using the results from Section 5.1.

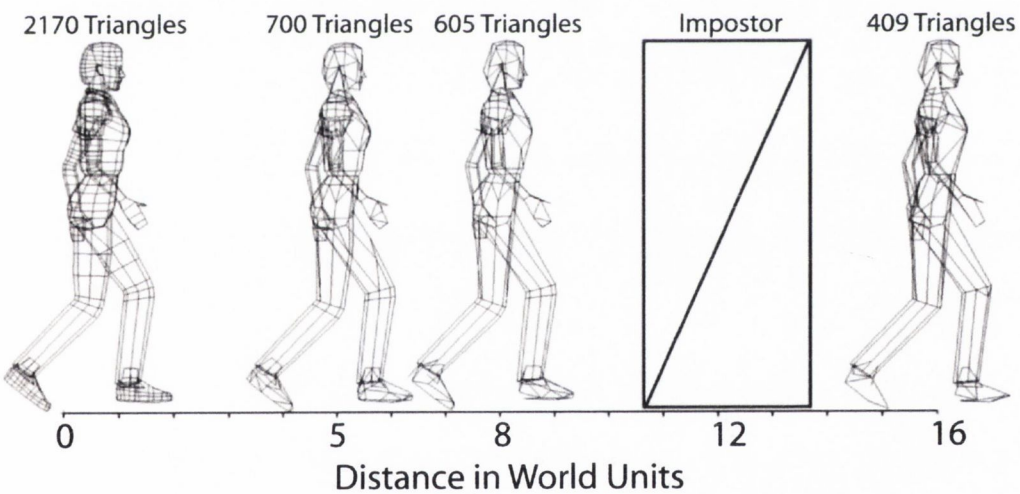


Figure 5.23: Distances at which different LOD representations are perceptually equivalent to the highest resolution mesh.

Chapter 6

Conclusions and Future Work

In this thesis we have presented various techniques for the domain of scalable virtual human simulation. The contributions of this thesis will be highlighted in this chapter and a number of topics for further research will be presented.

6.1 Summary of Contributions

The main contributions of this chapter can be summarized as follows:

1. **The Enhancement of the ALOHA System:**
   While the aim of the original ALOHA framework was to animate and render virtual humans in real-time based on an LOD approach, only the basic features of the system were implemented. Since we wished to simulate autonomous virtual humans within a 3D dynamic environment, various plug-ins were created at the initial stages of the system's development to improve the scene's content. Collaborative work allowed us to achieve our goal of simulating autonomous characters, which involved the following contributions to this domain:

   - **Implementation of Innate Behaviours:** Navigation, agent-object interaction and conversational behaviours were implemented to allow the virtual humans control their low-level motions, thus reducing the constraints of the existing system's pre-scripted approach. These behaviours complemented the use of different levels of detail to provide a viable solution for the simulation of small groups.
   - **Integration of the PPA Architecture:** By integrating the PPA architecture into the ALOHA system, this system enabled the creation of a simulation of an

environment populated by intelligent agents. These agents are able to carry out their daily lives, behaving believably in a wide range of situations.

2. **The Geopostor System:**
   We have created a system capable of rendering large crowds of characters in real-time, where each individual can exhibit various levels of variation. The crowd system has been integrated into a large city environment and allows the city to be populated with up to tens of thousands individuals. The main contribution is that it allows for imperceptible switching between an impostor and a mesh representation, based on a pixel to texel ratio. In this way, our system allows visual quality and performance to be balanced. Our results so far have convinced us that human impostors are an excellent substitute for geometry, not only because of proven rendering efficiency gains, but also in terms of visual fidelity. At certain distances, it is virtually impossible to determine whether the high-resolution model or the impostor is being rendered. The visual realism of our impostor representations is one of the clear advantages this method has over other approaches, such as using low polygon models.

   - **Impostor Rendering Techniques**:
     We have improved upon previous pre-generated impostor rendering techniques with respect to two factors: performance and realism. Firstly, the advances in graphics hardware allowed us to dynamically light and add colour variation to the impostors in a single pass. By providing an application to allow an artist to create 1-D outfit textures for the virtual humans, variation of the crowd was substantially increased with negligible memory usage. In addition to using DXT3 texture compression, we minimised texture thrashing by splitting the impostor textures into separate elevation images. We further contributed to the impostor's realism by providing realistic shadows that blend with the world and each other.
   - **Hybrid Representation**:
     We introduced a new hybrid representation that combines the impostor performing its default animation with a gesturing mesh in order to provide some animation variation, whilst avoiding the deformation of the entire mesh.

3. **Perceptual Evaluation of LOD Representations**:
   The Geopostor system employs various geometrical LOD representations in order to visualize crowds at a real-time frame rate. We improved the visual performance of the system by perceptually evaluating the effectiveness of impostors and low resolution meshes at replicating a highly detailed model's visual appearance. The results pro-

duced helpful metrics for guiding the switching between LOD representations without being perceived by users of our system. Although it is common for developers of crowd systems to utilize a LOD framework to accelerate the rendering of their application, they typically neglect the fundamental process of perceptually evaluating their techniques. This can lead to the incorrect choice of LOD selection metrics, causing the users' suspension of disbelief to suffer dramatically as a result of perceiving a change in the model. While the goal of carrying out these experiments was to improve the realism of our crowd system, it is hoped that these experiments will provide a methodology that developers could use to perceptually evaluate other LOD representations.

6.2 Future Work

6.2.1 Integration of the Geopostor System into the ALOHA System

The Geopostor system was built upon a simplified version of the ALOHA system and allows the simulation of large crowds. However, since the PPA architecture was removed, the current behaviour of the virtual crowds is fairly simplistic. With our crowd rendering system in place, we plan to integrate our work fully into the ALOHA system, which would result in a scalable system that provides our virtual humans with more believable behaviours using a variety of motions. Although the ALOHA system allows the simulation of autonomous agents that exhibit situational intelligence, the character models that were used contained a large number of triangles, resulting in the system's main bottleneck. This would be removed by utilizing the LOD representations employed by the Geopostor system. Since it would be impossible to simulate each individual in the crowd as an autonomous virtual human, a new LOD framework is proposed. This framework would involve the Geopostor system controlling the low-level simulation and rendering of the crowd in the background, while the ALOHA system would manage the higher LOD techniques for the characters that are the focus of the viewer's attention.

6.2.2 Future Enhancements to the Mesh Representation

By implementing matrix palette skinning using programmable hardware, this would allow subtle but important variations in the motions used to animate the geometric models with minimum overhead, taking a similar approach to Gosselin et al. [GSM05]. Additionally, per-pixel lighting effects involving various textures such as normal maps, gloss maps and ambient occlusion maps could be implemented in order to improve the mesh's realism. Olano et al. [OKS03] propose the idea of shader simplification or *shader LOD*, where they reduce the

complexity of a shader in order to increase the number of shaded objects that can be rendered at interactive rates. Future work will also investigating this technique for the shading of a character's mesh.

6.2.3 Future Enhancements to the Impostor Representation

Inclusion of a UV Coordinate Map

The impostor's detail map could be replaced with a UV coordinate map, where each pixel stores the texture coordinates of the triangle at that pixel. This image's alpha channel would contain the same alpha encoded regions as the detail map, as this distinguishes which pixels need to be transparent when displaying the impostor at run-time. To render the impostor representation using this image, this would involve replacing the detail map term (DetailMAP) in Equation 4.9 with the UV map (UVMAP) looking up the same detail texture (DetailTexture) used by the mesh's representation (see in Equation 6.1).

$$\begin{aligned} Pixel_{Colour} = \; & (DetailTexture_{RGB}\,[UVMap_R, UVMap_G]) * \\ & (Ambient_{LightModel} * Ambient_{Material} + \\ & (MAX((Vector_{Light} \cdot NormalMap_{RGB}), 0)) * \\ & (ColourMap\,[UVMap_{\alpha}]) * Diffuse_{Light})) \end{aligned} \tag{6.1}$$

The first advantage of the UV map is that it can be used to improve the impostor's realism by looking up various textures utilized in the per-pixel lighting of the impostor's associated mesh. For example, an impostor representation could be generated for the mesh used in ATI's crowd demo (see Section 2.1.3), where the UV map could be used to look up the following textures to match the shading of the mesh representation: a gloss map for specular lighting, an ambient occlusion map for the external lighting environment, and a decal texture to add detail, such as badges. The second advantage is that the UV map could be used to render an impostor using the detail texture of different template meshes. In the case of limited texture memory, impostor textures could be generated for a single model and variation could be added by allowing the UV map to look up several different detail textures. Additionally, each individual could be scaled so that they are not the same size.

Dynamically Generated Impostors

Previously, dynamically generated impostors involved rendering a model into an off-screen pixel buffer or *pbuffer*, requiring a computationally expensive context switch. With the recent introduction of the *EXT_framebuffer_object* extension into the OpenGL specification, this provides a much better and more simplified method. The main advantage of the *framebuffer object* is that it only requires a single GL context, resulting in the switching between framebuffers being much faster than switching between pbuffers. Although this extension would improve the performance of generating an impostor image for a character's mesh at run-time, this impostor technique relies heavily on re-using the current impostor image for it be efficient, since the character's full geometry needs be deformed and rendered each time the image is updated.

Another approach would be to render several pre-generated impostors to the framebuffer object, and dynamically generate a single impostor containing these characters. 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 may 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 potentially 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. However, issues that may cause artefacts will need to be investigated such as the depth of the humans being incorrect.

Anti-Aliasing

Although our impostors are rendered without anti-aliasing, the representation's pixellated silhouette is predominantly masked by the crowded nature of the simulated scenes and switching to the associated mesh before it becomes noticeable. However, future work would involve investigating how anti-aliasing techniques would improve the impostor's visual appeal, taking its cost into account.

6.2.4 Hybrid Representation

Due to the occlusion nature of the virtual human's body parts, the hybrid representation is limited to performing head gestures, or otherwise holes appear. Future work will investigate how this problem could be solved. One solution is to pre-generate the impostor textures without the virtual human's arms and pre-generate a second set of impostor textures just for the arms performing the default animation. However, the effect on the impostor's rendering cost and texture memory consumption would need to be investigated. In the case of the latter being a problem, this could be reduced by limiting arm gestures to virtual humans at eye-level. The use of dynamically generated impostors will also be investigated in order to provide a cheap representation that is not limited to a cyclical default animation.

6.2.5 Perceptual Evaluation

The perceptual evaluation of the LOD representations used in our system could be considered conservative, since these experiments consisted of either one or two characters on-screen and rendered in grey-scale. Future experiments would involve investigating whether the complexity of the scene (i.e., a crowd of characters, colour) affects these metrics.

Recent PC games and next generation console games are increasingly employing low-resolution meshes with normal map images to render visually detailed characters. While these characters are aesthetically pleasing at a fraction of the rendering cost, the detail of the low-resolution mesh needs to be carefully chosen in order to avoid shading artefacts. Future work will investigate the ability of humans to perceive these artefacts with the aim of establishing a useful metric.

Appendix A

Simulation Example using the PPA Architecture Integrated into the ALOHA System

For demonstration purposes, a simulation of a university campus has been created. As the simulation progresses, a population of virtual humans moves between the different parts of the campus assuming and discarding a range of different roles. Implemented roles include students attending lectures, academics presenting these lectures, people relaxing in the college bar, and a barman serving customers (see Figure 2). The basic agent's FCM employed in this simulation is shown in Figure 1 and the four different roles layered on top of this FCM are shown in Figure 3.

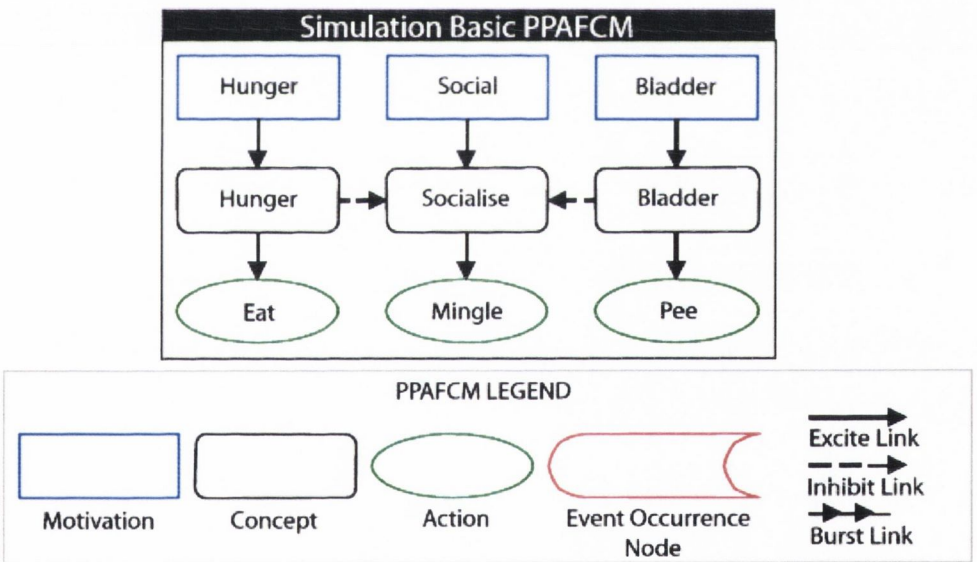


Figure 1: The PPAFCM used by the basic agent in the simulation of a university campus.

Figure 3 shows the adoption of the *lecturer* role used to simulate academics lecturing.

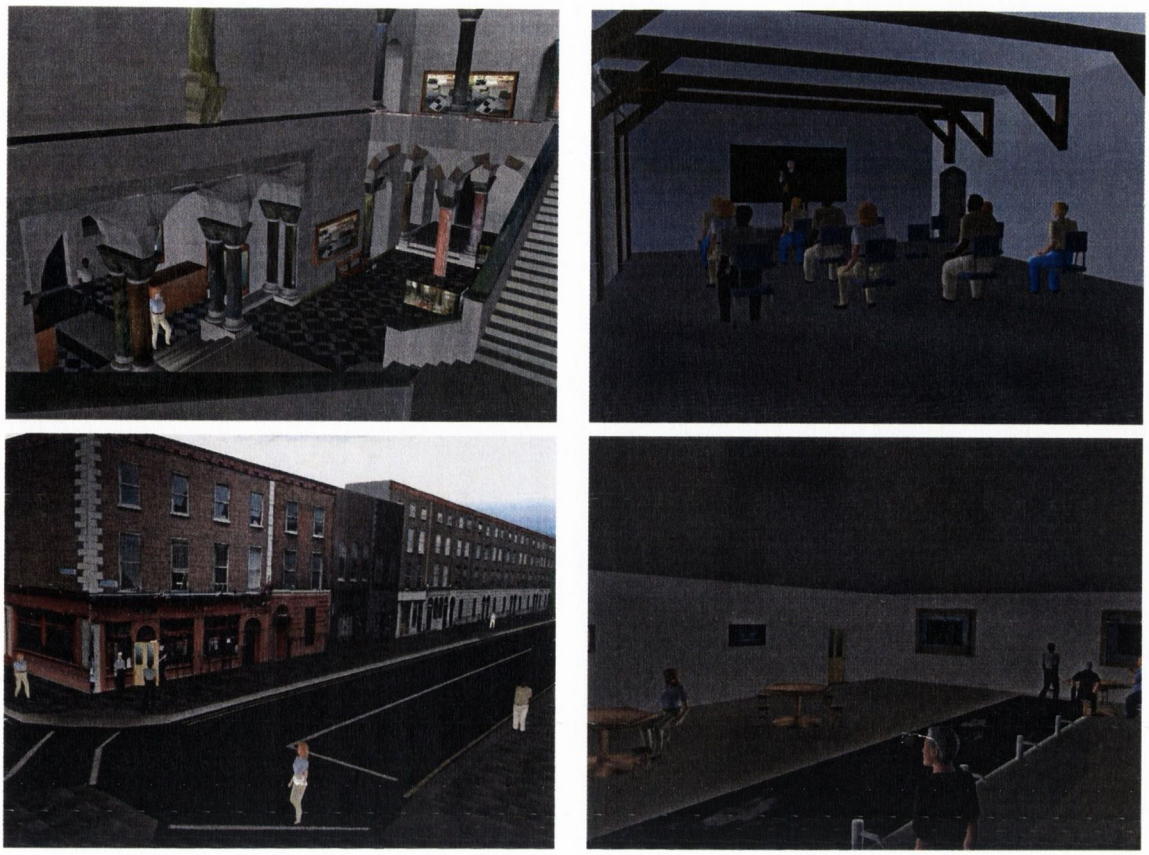


Figure 2: University campus simulation using the PPA architecture: (a) Students going to lectures. (b) Academic presenting the lecture. (c) Academic and students going to the pub (d) People socialising in the pub.

This role involves new "Occupy Lecture Podium" and "Lecture" concepts, where the respective motivations are of the same name. Activation of the "Occupy Lecture Podium" concept results in "Use Lecture Podium" action where the agent uses the *lecture-podium* object with its agent-object interaction behaviour. This concept is also linked to a persistent flag node, which is used to maintain knowledge on whether or not particular states of the virtual world have been satisfied. For example, the "At Lecture Podium" flag keeps track of whether the agent is currently at the podium or not. When the agent has finished the "Use Lecture Podium" action, the "At Podium" flag is turned on, which in turn inhibits the agent's "Occupy Lecture Podium" concept. Once the agent is finished giving its lecture, the flag is turned off. The activation of the "Lecture" concept leads to the "Give Lecture" action, implemented by the agent's conversational behaviour, whereby only the lecturer is allowed to speak. The "Occupy Lecture Podium" inhibits the "Lecture" concept so that the agent only

starts lecturing when he is at the podium. To ensure that the lecturer does not socialise with the students, and leave the lecture to go to the toilet or get some food, the role respectively increases the thresholds of the basic "Socialise", "Bladder" and "Hunger" concepts.

Figure 3 shows the adoption of the *student* role used to simulate students attending lectures. This role involves a new "Attend Lecture" concept, where the motivation is of the same name, and activation leads to the "Sit In Lecture" action which simulates the agents sitting at their scheduled lecture. The "Sit In Lecture" action is simply an agent-object interaction behaviour used in conjunction with a *lecture-room-desk* object. The student role's PPAFCM contains rule nodes which allow agents to check the occurrence of particular states within the virtual world. A rule node stores an associated rule which is evaluated regularly to determine the node's activation e.g., when the "Lecturer Nearby" rule is satisfied, the "Attend Lecture" concept is increased (to hurry the students to sit down) and the "Socialise" concept is implicitly inhibited (to stop the students talking during the lecture). To ensure that the agent does not leave the lecture to go to the toilet or get some food, the role increases the thresholds of the basic "Bladder" and "Hunger" concepts. Similar to the lecturer role's "At Podium" flag, the "Sitting" flag keeps track of whether the student is currently sitting or not. When the student has finished the "Sit In Lecture" action, the "Sitting" flag is turned on, which in turn inhibits the student's "Attend Lecture" concept. Once the student stands up, the flag is turned off.

Once the lectures are finished, the agents layered with a Student or Lecturer role, discard their role and assume a *bar customer* role (see Figure 3) to simulate the agents going for a drink. The "Time to Change Role" and "Not Ready To Change Role" nodes are used to control the smooth transition between roles so that the agents finish what they are currently doing and do not drop everything when its schedule indicates it is time to change role. The important concepts for bar customer role are "Get Drink" and "Sit" which have motivations of "Beer Lust" and "Sit" as their respective primary inputs. Each concept has attached to it an action node indicating, respectively, that the customer should get a drink from the bar ("Use Bar"), or sit down at one of the bar's tables ("Sit Down"). Both these actions result in the agent-object interaction behaviour respectively utilizing a *bar-counter* object (described in Section 3.4.2) and a *bar-table* object, to step the customer through how it should use these objects. In addition to the implicit inhibitions from the basic agent concepts to those in the assumed role, a inhibition link exists between the "Get Drink" and "Sit" concepts to stop the customer trying to sit down while it is getting a drink. A "Sitting" flag is used to maintain whether the customer is sitting or not.

The final role employed in this simulation is the *bar keep* role which is used to simulate a barman serving drinks to the customer. The important concept for this role is "Serve"

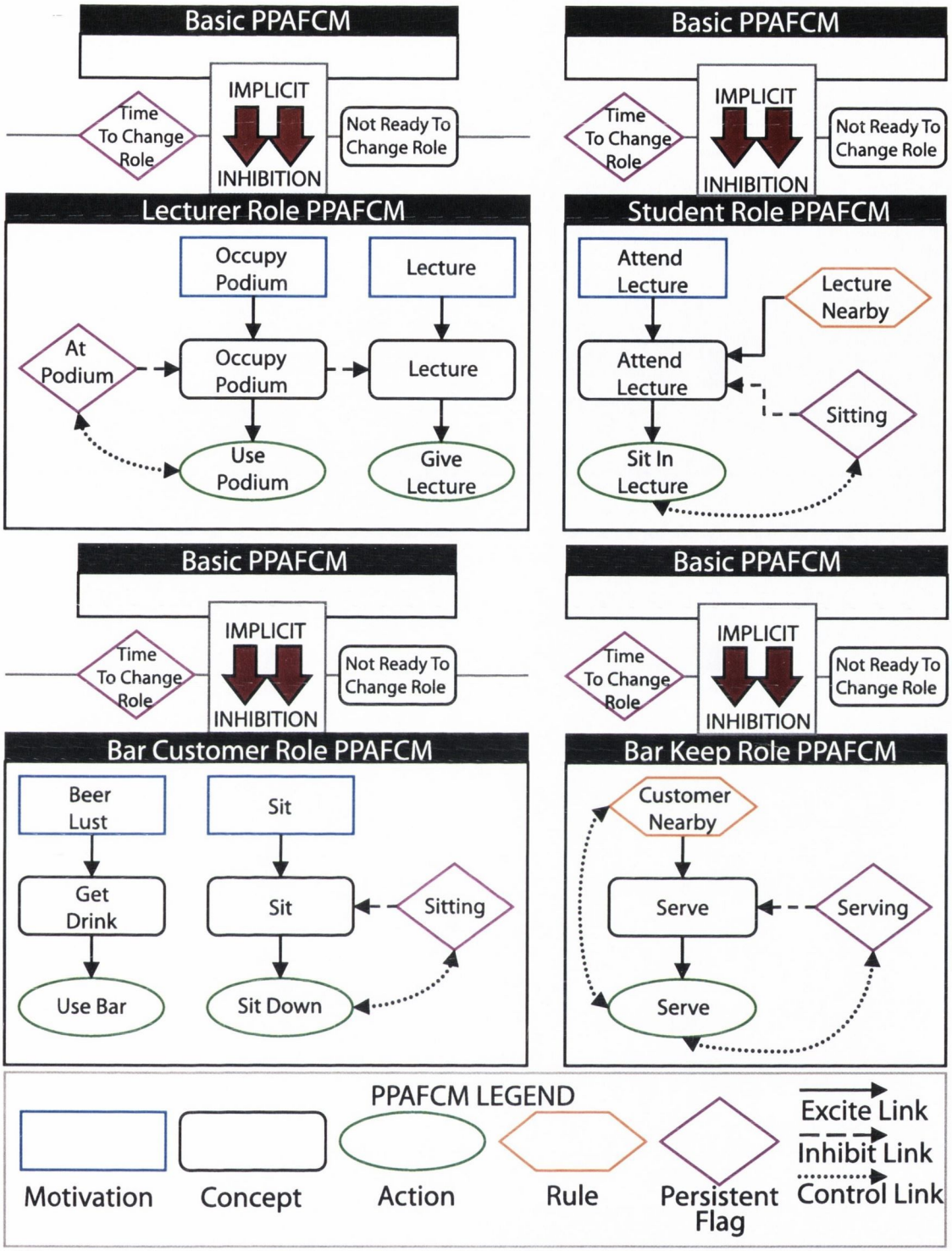


Figure 3: Basic agent with the following roles layered on top of it: (a) LECTURER Role. (b) STUDENT Role. (c) BAR CUSTOMER Role. (d) BAR KEEP Role.

which has a "Customer Nearby" rule node as its primary input, which is set once an agent layered with the bar customer role uses the bar-counter object. The "Serve" concept's linked action node results in the agent-object interaction behaviour utilizing the *bar-counter* object, and subsequently sets the "Serving" flag which inhibits the barman from serving another customer until it finishes with the current customer. To ensure that the barman does not leave the bar to either go to the toilet or get some food, the role increases the thresholds of the basic "Bladder" and "Hunger" concepts.

Appendix B

ALOHA Specific File Formats

Extension	Description
.ASF	ALOHA Skeletal Format
.AMF	ALOHA Mesh Format
.ASMF	ALOHA Skinned Mesh Format
.AKF	ALOHA Keyframe Format
.IDF	Impostor Data File
.SDF	Scene Descriptor File

Table 1: ALOHA specific file formats.

Bibliography

[ABC96] K. Amaya, A. Bruderlin, and T. Calvert. Emotion from motion. *GI '96: Proceedings of the Conference on Graphics Interface*, pages 222–229, 1996.

[ABT00] 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.

[AI.] AI.implant. Biographic technologies. www.ai-implant.com.

[AL00] R. Aylett and M. Luck. Applying artificial intelligence to virtual reality: Intelligent virtual environments. *Applied Artificial Intelligence*, 14(1):3–32, 2000.

[AM00] U. Assarsson and T. Möller. Optimized view frustum culling algorithms for bounding boxes. *Journal of Graphics Tools: JGT*, 5(1):9–22, 2000.

[BB88] R. Barzel and A. H. Barr. A modeling system based on dynamic constraints. *SIGGRAPH '88: Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, pages 179–188, 1988.

[BBB+99] N. I. Badler, R. Bindiganavale, J. Bourne, J. Allbeck, J. Shi, and M. S. Palmer. Real-time virtual humans. *International Conference on Digital Media Futures*, 1999.

[BH02] D. Brogan and J. Hodgins. Simulation level of detail for multiagent control. *AAMAS '02: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 199–206, 2002.

[BHS98] J. Bittner, V. Havran, and P. Slavík. Hierarchical visibility culling with occlusion trees. pages 207–219, 1998.

[BID+01] R. Burke, D. Isla, M. Downie, Y. Ivanov, and B. Blumberg. Creature smarts: The art and architecture of a virtual brain. *Proceedings of the Game Developers Conference*, pages 147–166, 2001.

[Bli78] J. F. Blinn. Simulation of wrinkled surfaces. *SIGGRAPH '78: Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, pages 286–292, 1978.

[Bli88] J. Blinn. Me and my (fake) shadow. *IEEE Comput. Graph. Appl.*, 8(1):82–86, 1988.

[BMdOB03] A. Braun, S. R. Musse, L. P. L. de Oliveira, and B. E. J. Bodmann. Modeling individual behaviours in crowd simulation. *16th International Conference on Computer Animation and Social Agents (CASA)*, pages 143–148, 2003.

[BMH98a] D. Bartz, M. Meißner, and T. Hüttner. Extending graphics hardware for occlusion queries in OpenGL. *HWWS '98: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, pages 97–104, 1998.

[BMH98b] D. Brogan, R. Metoyer, and J. Hodgins. Dynamically simulated characters in virtual environments. *IEEE Computer Graphics and Applications*, 18(5):59–69, 1998.

[BN76] J. F. Blinn and M. E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542–546, 1976.

[BPB99] N. I. Badler, M. S. Palmer, and R. Bindiganavale. Animation control for real-time virtual humans. *Communications of the ACM*, 42(8):64–73, 1999.

[Bro02] M. Brockington. *AI Game Programming Wisdom - Level of Detail AI for a Large Role-Playing Game*, pages 419–425. Charles River Media, 2002.

[BY95] C. Beardon and V. Ye. Using behavioral rules in animation. *Computer Graphics: Development in Virtual Environments*, pages 217–234, 1995.

[Cat74] E. E. Catmull. *A subdivision algorithm for computer display of curved surfaces.* PhD thesis, Dept. of Computer Science, University of Utah, December 1974.

[Ceb01] C. Cebenoyan. Efficient animation. *Game Developer Conference (NVIDIA Corporation Presentation)*, 2001. http://developer.nvidia.com/object/gdc2001_efficientanimation.html.

[CH97] D. A. Carlson and J. K. Hodgins. Simulation levels of detail for real-time animation. *GI '97: Proceedings of the Conference on Graphics Interface*, pages 1–8, 1997.

[Cla76] J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, 1976.

[CMVA99] R. Cozot, F. Multon, B. Valton, and B. Arnaldi. Animation levels of detail design for real-time virtual human. *Eurographics Workshop on Animation and Simulation '99*, pages 35–44, 1999.

[COCSD03] D. Cohen-Or, Y. Chrysanthou, C. T. Silva, and F. Durand. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):412–431, 2003.

[COFHZ98] D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Computer Graphics Forum*, 17(3):243–254, 1998.

[COM98] J. Cohen, M. Olano, and D. Manocha. Appearance-preserving simplification. *SIGGRAPH '98: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive techniques*, pages 115–122, 1998.

[Cor62] T. Cornsweet. The staircase method in psychophysics. *American Journal of Psychology*, 75:485–491, 1962.

[Cro] Crowd Dynamics Ltd. *http://www.crowddynamics.com.*

[Cro77] F. Crow. Shadow algorithms for computer graphics. *In Computer Graphics (SIGGRAPH 77 Proceedings)*, pages 242–248, 1977.

[CT97] S. Coorg and S. Teller. Real-time occlusion culling for models with large occluders. *SI3D '97: Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pages 83–90, 1997.

[CT00] A. Caicedo and D. Thalmann. Virtual humanoids: Let them be autonomous without losing control. *The Fourth International Conference on Computer Graphics and Artificial Intelligence 3IA2000*, 2000.

[CVB01] J. Cassell, H. Vilhjálmsson, and T. Bickmore. BEAT: the behavior expression animation toolkit. *SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 477–486, 2001.

[CVC+99] J. Cassell, H. Vilhjálmsson, K. Chang, T. Bickmore, L. Campbell, and H. Yan. Requirements for an architecture for embodied conversational characters. *Computer Animation and Simulation 99 (Eurographics Series)*, pages 109–120, 1999.

[DDSD03] X. Décoret, F. Durand, F. Sillion, and Julie Dorsey. Billboard clouds for extreme model simplification. *ACM Transactions on Graphics (TOG)*, 22(3):689–696, 2003.

[dHCSMT05] P. de Heras Ciechomski, S. Schertenleib, Jonathan Maïm, and D. Thalmann. Reviving the roman odeon of aphrodisias: Dynamic animation and variety control of crowds in virtual heritage. *VSMM '05: Proceeding of the 11th International Conference on Virtual Systems and Multimedia*, pages 601–610, 2005.

[dHCUCT04] P. de Heras Ciechomski, B. Ulicny, R. Cetre, and D. Thalmann. A case study of a virtual audience in a reconstruction of an ancient roman odeon in aphrodisias. *VAST '04: The 5th International Symposium on Virtual Reality, Archaeology and Cultural Heirtage*, pages 9–17, 2004.

[DHOO05] S. Dobbyn, J. Hamill, K. O'Conor, and C. O'Sullivan. Geopostors: A real-time geometry / impostor crowd rendering system. *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 95–102, April 2005.

[DK96] J. Dickerson and B. Kosko. Virtual worlds as fuzzy dynamical systems. *Technology for Multimedia*, pages 567–603, 1996.

[Dob01] S. Dobbyn. Scripting system for real-time animation. *Final Year Project, Trinity College Dublin, Ireland*, 2001.

[Dom] S. Dominé. Mesh skinning. *http://developer.nvidia.com/object/skinning.html.*

[Don97] S. Donikian. Vuems: A virtual urban environment modelling system. *CGI '97: Proceedings of the 1997 Conference on Computer Graphics International*, pages 127–133, June 1997.

[Don05] W. Donnelly. *GPU Gems 2 - Per-Pixel Displacement Mapping with Distance Functions*, pages 123–136. Addison-Wesley, 2005.

[DSSD99] X. Décoret, G. Schaufler, F. Sillion, and J. Dorsey. Multi-layered impostors for accelerated rendering. *Computer Graphics Forum (Proceedings of Eurographics '99)*, 18(3):61–73, 1999.

[DSV97] L. Darsa, B. Costa Silva, and A. Varshney. Navigating static environments using image-space simplification and morphing. *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics and games*, pages 25–34, 1997.

[Ebb98] P. Ebbesmeyer. Textured virtual walls - achieving interactive frame rates during walkthroughs of complex indoor environments. *VRAIS '98: Proceedings of the Virtual Reality Annual International Symposium*, pages 220–228, 1998.

[EDD+95] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsberg, and W. Stuetzle. Multiresolution analysis for arbitrary meshes. *SIGGRAPH '95: Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pages 173–182, 1995.

[EW96] P. K. Egbert and S. H. Winkler. Collision-free object movement using vector fields. *IEEE Computer Graphics and Applications*, 16(4):18–24, 1996.

[FFMC01] C. Fairclough, M. Fagan, B. MacNamee, and P. Cunningham. Research directions for AI in computer games. *Proceedings of the Twelfth Irish Conference on Artificial Intelligence and Cognitive Science*, pages 333–344, 2001.

[FIY+95] T. Fuji, K. Imamura, T. Yasuda, S. Yokoi, and J. Toriwaki. A virtual scene system for city planning. *Computer Graphics: Development in Virtual Environments*, pages 485–496, 1995.

[FS93] T. A. Funkhouser and C. H. Séquin. Adaptive display algorithm for interactive frame rate during visualisation of complex virtual environments. *SIGGRAPH '93: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, pages 247–254, 1993.

[FST92] T. A. Funkhouser, C. H. Séquin, and S. Teller. Management of large amounts of data in interactive building walkthroughs. *SI3D '92: Proceedings of the 1992 symposium on Interactive 3D graphics*, pages 11–20, 1992.

[FTT99] J. Funge, X. Tu, and D. Terzopoulos. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. *SIGGRAPH '99: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 29–38, 1999.

[Fun99] J. Funge. *AI for Games and Animation: A Cognitive Modeling Approach.* A.K. Peters, 1999.

[FvdPT01] P. Faloutsos, M. van de Panne, and D. Terzopoulos. The virtual stuntman: Dynamic characters with a repertoire of autonomous motor skills. *Computers and Graphics*, 25(6):933–953, December 2001.

[Gla86] A. Glassner. Adaptive precision in texture mapping. *SIGGRAPH '86: Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, pages 297–306, 1986.

[GLM98] S. Goldstein, E. Large, and D. Metaxas. Dynamical autonomous agents: Game applications. *Proceedings of Computer Animation '98*, pages 25–33, 1998.

[GLM99] S. Goldstein, E. Large, and D. Metaxas. Non-linear dynamical system approach to behaviour modeling. *The Visual Computer*, 15:349–369, 1999.

[GMPO00] T. Giang, R. Mooney, C. Peters, and C. O'Sullivan. Aloha: Adaptive level of detail for human animation towards a new framework. *Eurographics 2000 Short Paper Programme*, pages 71–77, 2000.

[Gof63] E. Goffmann. *Behavior in public places; notes on the social organization of gatherings.* New York, Free Press of Glencoe, 1963.

[Goo81] C. Goodwin. *Conversational Organization: Interaction between speakers and hearers.* New York, Academic Press, 1981.

[Gou71] H. Gouraud. Continuous shading of curved surfaces. *IEEE Transactions on Computers*, 20(6):623–628, 1971.

[Gra99a] Silicon Graphics. Nv_register_combiners extension. *http://oss.sgi.com/projects/ogl-sample/registry/NV/register_combiners.txt*, 1999.

[Gra99b] Silicon Graphics. Nv_texture_shader extension. *http://oss.sgi.com/projects/ogl-sample/registry/NV/texture_shader.txt*, 1999.

[Gra02a] Silicon Graphics. Arb_fragment_programs extension. *http://oss.sgi.com/projects/ogl-sample/registry/ARB/fragment_program.txt*, 2002.

[Gra02b] Silicon Graphics. Arb_vertex_programs extension. *http://oss.sgi.com/projects/ogl-sample/registry/ARB/vertex_program.txt*, 2002.

[GSM05] D. Gosselin, P. Sander, and J. Mitchell. *ShaderX3 - Drawing a Crowd*, pages 505–517. Charles River Media, 2005.

[Gui05] J. Guinot. Image filtering with GLSL - convolution kernels. *http://www.ozone3d.net/tutorials/image_filtering.php*, 2005.

[Ham05] J. Hamill. *Level of Detail Techniques for Real-Time Urban Simulation.* PhD thesis, University of Dublin, Trinity College, 2005.

[HDMO05] J. Hamill, S. Dobbyn, R. McDonnell, and C. O'Sullivan. Perceptual evaluation of impostor representations for virtual humans and buildings. *Computer Graphics Forum (Proceedings of Eurographics '05)*, 24:623–633, 2005.

[Hei91] T. Heidmann. Real shadows, real time. *Iris Universe*, (18):23–31, 1991.

[HLTC03] L. Heïgeas, A. Luciani, J. Thollot, and N. Castagné. A physically-based particle model of emergent crowd behaviors. *Graphicon*, 2003.

[HMAM02] E. Haines, T. Möller, and T. Akenine-Moller. *Real-Time Rendering.* A.K. Peters, 2002.

[HMB98] T. Hüttner, M. Meißner, and D. Bartz. OpenGL-assisted visibility queries of large polygonal models. Technical report, Dept. of Computer Science (WSI), University of Tübingen, 1998.

[HO03] J. Hamill and C. O'Sullivan. Virtual dublin - a framework for real-time urban simulation. *Proceedings of the Winter Conference on Computer Graphics (WSCG)*, 11(1-3):221–225, 2003.

[Hop96] H. Hoppe. Progressive meshes. *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 99–108, 1996.

[HOT98] J. K. Hodgins, J. F. O'Brien, and J. Tumblin. Perception of human motion with different geometric models. *IEEE Transactions on Visualization and Computer Graphics*, 4(4):307–316, 1998.

[HP88] D. Hamann and R. Parent. The behavioral test-bed: Obtaining complex behaviours from simple rules. *The Visual Computer*, 4(6):332–347, 1988.

[HP97] J. K. Hodgins and N. S. Pollard. Adapting simulated behaviors for new characters. *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 153–162, 1997.

[HRD04] J. Harrison, R. Rensink, and M. V. DePanne. Obscuring length changes during animated motion. *ACM Transactions on Graphics (TOG)*, 23(3):569–573, 2004.

[HS02] G. Hjelstrom and P. Smith. Polygon soup for the programmer's soul: 3d pathfinding. *Proceedings of the Game Developers Conference*, 2002.

[HWBO95] J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien. Animating human athletes. *SIGGRAPH '95: Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pages 71–78, 1995.

[IBB96] R. Ingram, S. Benford, and J. Bowers. Building virtual cities: applying urban planning principles to the design of virtual environments. *VRST '96: Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 83–91, 1996.

[JLF96] W. Jepson, R. Liggett, and S. FriedMan. Virtual modeling of urban environments. *Presence: Teleoperators and Virtual Environments*, 5(1):72–86, 1996.

[JW02] S. Jeschke and M. Wimmer. Textured depth meshes for real time rendering of arbitrary scenes. *EGRW '02: Proceedings of the 13th Eurographics Workshop on Rendering*, pages 181–190, 2002.

[JWP05] S. Jeschke, M. Wimmer, and W. Purgathofer. Image-based representations for accelerated rendering of complex scenes. *In Eurographics 2005 STAR Reports*, pages 1–20, 2005.

[KGP02] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. *ACM Transactions on Graphics (TOG)*, 21(3):473–482, 2002.

[Kil99] M. J. Kilgard. Improving shadows and reflections via the stencil buffer. *http://developer.nvidia.com/attach/6641*, 1999.

[Kil00] M. J. Kilgard. A practical and robust bump-mapping technique for today's GPUs. Technical report, NVIDIA Corporation, 2000.

[KK89] J. T. Kajiya and T. L. Kay. Rendering fur with three dimensional textures. *SIGGRAPH '89: Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*, pages 271–280, 1989.

[Kle01] S. A. Klein. Measuring, estimating and understanding the psychometric function:a commentary. *Perception and Psychophysics*, 63(8):1421–1455, 2001.

[KT98] M. Kallmann and D. Thalmann. Modeling objects for interaction tasks. *Proceedings of the 9th Eurographics Workshop on Animation and Simulation (EGCAS)*, pages 73–86, 1998.

[KT99a] M. Kallmann and D. Thalmann. A behavioral interface to simulate agent-object interactions in real time. *CA '99: Proceedings of Computer Animation*, pages 138–146, 1999.

[KT99b] M. Kallmann and D. Thalmann. Direct 3d interaction with smart objects. *VRST '99: Proceedings of ACM Virtual Reality Software and Technology*, pages 124–130, 1999.

[KZ05] L. Kavan and J. Zara. Spherical blend skinning: A real-time deformation of articulated models. *SI3D '05: Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, pages 9–16, 2005.

[Lan98] J. Lander. Skin them bones. *Game Developer Magazine*, pages 11–16, May 1998.

[Lan99] J. Lander. Over my dead, polygonal body. *Game Developer Magazine*, pages 17–22, October 1999.

[LB98] D. Loeb and A. Bustanoby. Titanic and digital character animation. *SIGGRAPH '98: ACM SIGGRAPH 98 Conference abstracts and applications*, page 309, 1998.

[LCR02] J. Lee, J. Chai, and P. Reitsma. Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics (TOG)*, 21(3):491–500, 2002.

[Lee02] W. Leeson. *Games Programming Gems III - Subdivision Surfaces for Character Animation*, pages 372–383. Charles River Media, 2002.

[Lev71] H. Levitt. Transformed up-down methods in psychoacoustics. *Journal of the Acoustical Society of America*, 49(2):467–477, 1971.

[Lev96] L. Levison. *Connecting Planning and Acting via Object-Specific Reasonings.* PhD thesis, Dept. of Computer and Information Science, University of Pennsylvania, 1996.

[LGC94] Z. Liu, S. J. Gortler, and M. F. Cohen. Hierarchical spacetime control. *SIGGRAPH '94: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 35–42, 1994.

[LHEJ01] M. Linschoten, L. Harvey, P. Eller, and W. Jafek. Fast and accurate measurement of taste and smell thresholds using a maximum-likelihood adaptive staircase procedure. *Perception and Psychophysics*, pages 1330–1347, 2001.

[LMM03] C. Loscos, D. Marchal, and A. Meyer. Intuitive crowd behaviour in dense urban environments using local laws. *Theory and Practice of Computer Graphics*, page 122, 2003.

[LS97] J. Lengyel and J. Snyder. Rendering with coherent layers. *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 233–242, 1997.

[LTC01] C. Loscos, F. Tecchia, and Y. Chrysanthou. Real-time shadows for animated crowds in virtual cities. *VRST '01: Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 85–92, 2001.

[LW85] M. Levoy and T. Whitted. The use of points as a display primitive. Technical report, University of North Carolina at Chapel Hill, 1985.

[LWC+02] D. Luebke, B. Watson, J. D. Cohen, M. Reddy, and A. Varshney. *Level of Detail for 3D Computer Graphics.* Elsevier Science Inc, 2002.

[Mac93] P. Maciel. *Visual Navigation of largely unoccluded environments using textured clusters.* PhD thesis, Indian University, Bloomington, January 1993.

[Mac04] B. MacNamee. *Persistent Proactive Agents: Using Situational Intelligence To Create Support Characters in Character-Centric Computer Games.* PhD thesis, University of Dublin, Trinity College, 2004.

[MAEH04] K. Mania, B. Adelstein, S. R. Ellis, and M. Hill. Perceptual sensitivity to head tracking latency in virtual environments with varying degrees of scene complexity. *APGV '04: Proceedings of the 1st Symposium on Applied Perception in Graphics and Visualization*, pages 39–47, 2004.

[Map] ATI Normal Mapper. http://www.ati.com.

[MAS] MASSIVE Software. *http://www.massivesoftware.com/.*

[MC01] B. MacNamee and P. Cunningham. Proposal for an agent architecture for proactive persistent non player characters. *Proceedings of the 12th Irish Conference on AI and Cognitive Science*, pages 221–232, 2001.

[MC03] B. MacNamee and P. Cunningham. Creating socially interactive non-player characters: The μ-sic system. *International Journal of Intelligent Games & Simulation*, 2(1):28–35, 2003.

[MDCO02] B. MacNamee, S. Dobbyn, P. Cunningham, and C. O'Sullivan. Men behaving appropriately: Integrating the role passing technique into the aloha system. *Proceedings of the AISB'02 Symposium: Animating Expressive Characters for Social Interactions*, pages 59–62, 2002.

[MDCO03] B. MacNamee, S. Dobbyn, P. Cunningham, and C. O'Sullivan. Simulating virtual humans across diverse situations. *Intelligent Virtual Agent Conference*, pages 159–163, 2003.

[MJC+03] S. R. Marschner, H. W. Jensen, M. Cammarano, S. Worley, and P. Hanrahan. Light scattering from human hair fibers. *ACM Transactions on Graphics (TOG)*, 22(3):780–791, 2003.

[MS95] P. Maciel and P. Shirley. Visual navigation of large environments using textured cluster. *SI3D '95: Proceedings of the 1995 Symposium on Interactive 3D Graphics*, pages 95–102, 1995.

[MT00] S. R. Musse and D. Thalmann. From one virtual actor to virtual crowds: requirements and constraints. *AGENTS '00: Proceedings of the Fourth International Conference on Autonomous Agents*, pages 52–53, 2000.

[MT01] S. R. Musse and D. Thalmann. A hierarchical model for real time simulation of virtual human crowds. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):152–164, 2001.

[Nil87] N. J. Nilsson. *Principles of Artificial Intelligence.* Springer Verlag, 1987.

[NRTT95] H. Noser, O. Renault, D. Thalmann, and N. Thalmann. Navigation for digital characters based on synthetic vision, memory and learning. *Computer and Graphics*, 19(1):7–19, 1995.

[NVIa] NVIDIA. NVPerfKit. *http://developer.nvidia.com/object/nvperfkit_home.html.*

[NVIb] NVIDIA. NVPerfKit User Guide. *http://download.nvidia.com/developer/Tools/NVPerfKit/User_Guide_NVPerfKit.pdf.*

[NVIc] NVIDIA. Using vertex buffer objects. *http://developer.nvidia.com/object/using_VBOs.html.*

[OCV+02] C. O'Sullivan, J. Cassell, H. Vilhjálmsson, J. Dingliana, S. Dobbyn, B. McNamee, C. Peters, and T. Giang. Levels of detail for crowds and groups. *Computer Graphics Forum*, 21(4):733–742, 2002.

[ODGK03] C. O'Sullivan, J. Dingliana, T. Giang, and M. K. Kaiser. Evaluating the visual fidelity of physically based animations. *ACM Transactions on Graphics (TOG)*, 22(3):527–536, 2003.

[OHJ00] M. Oesker, H. Hecht, and B. Jung. Psychological evidence for unconscious processing of detail in real-time animation of multiple characters. *Journal of Visualization and Computer Animation*, 11(2):105–112, 2000.

[OKS03] M. Olano, B. Kuehne, and M. Simmons. Automatic shader level of detail. *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 7–14, 2003.

[Ope04] Silicon Graphics Inc. OpenGL. The OpenGL graphics system: A specification. *http://www.opengl.org/documentation/specs/version2.0/glspec20.pdf*, October 2004.

[Pap] 3Dc White Paper. ATI Technologies. *http://www.ati.com/products/radeonx800/3DcWhitePaper.pdf.*

[PBC+01] I. Pandzic, C. Babski, T. Capin, W. Lee, N. Magnenat-Thalmann, S. R. Musse, L. Moccozet, H. Seo, and D. Thalmann. Simulating virtual humans in networked virtual environments. *Presence: Teleoperators and Virtual Environments*, 10(6):632–646, 2001.

[PBG98] L. Prasso, J. Buhler, and J. Gibbs. The pdi crowd system for antz. *SIGGRAPH '98: ACM SIGGRAPH 98 Conference Abstracts and Applications*, page 313, 1998.

[PBM94] N. A. Aufaure Portier, P. Berthet, and J. L. Moreno. Optimised network modelling for route planning. *EGIS '94: Proceedings of the Fifth European Conference and Exhibition on Geographic Information Systems*, (2):1807–1816, 1994.

[PDMO03] C. Peters, S. Dobbyn, B. MacNamee, and C. O'Sullivan. Smart objects for attentive agents. *In Short Paper Proceedings of the Winter Conference on Computer Graphics (WSCG)*, 2003.

[PG96] K. Perlin and A. Goldberg. Improv: a system for scripting interactive actors in virtual worlds. *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 205–216, 1996.

[PO02a] C. Peters and C. O'Sullivan. Synthetic vision and memory for autonomous virtual humans. *Computer Graphics Forum*, 21(4):743–752, 2002.

[PO02b] C. Peters and C. O'Sullivan. Vision-based reaching for autonomous virtual humans. *Proceedings of the AISB02 symposium: Animating Expressive Characters for Social Interactions*, 2002.

[PO03a] C. Peters and C. O'Sullivan. Attention-driven eye gaze and blinking for virtual humans. *GRAPH '03: Proceedings of the SIGGRAPH 2003 conference on Sketches & applications*, page 1, 2003.

[PO03b] C. Peters and C. O'Sullivan. Bottom-up visual attention for virtual human animation. *16th International Conference on Computer Animation and Social Agents (CASA)*, pages 111–117, 2003.

[Pot99] D. Pottinger. Coordinated unit movement. *Game Developer Magazine*, 3(3):207–217, January 1999.

[PPB+97] D. Pratt, S. Pratt, Paul T. Barham, R. Barker, M. Waldrop, J. Ehlert, and C. Chrislip. Humans in large-scale, networked virtual environments. *Presence: Teleoperators and Virtual Environments*, 6(5):547–564, 1997.

[Proa] The CAHRISMA Project. European Commission Fifth Framework INCO MED Programme. http://www.dat.dtu.dk/cahrisma.htm.

[Prob] The ERATO Project. http://www.at.oersted.dtu.dk/~erato/index.htm.

[PRT01] M. Parenthon, P. Reignier, and J. Tisseau. Put fuzzy cognitive maps to work in virtual worlds. *10th International Conference on Fuzzy Systems (Fuzz-IEEE'01)*, pages 252–255, 2001.

[RCB98] C. Rose, M. F. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics Applications*, 18(5):32–40, 1998.

[Ren] Renderware A.I. *http://www.renderware.com.*

[Rey87] C. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *SIGGRAPH '87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, pages 25–34, 1987.

[Rey99] C. Reynolds. Steering behaviors for autonomous characters. *Proceedings of the Game Developers Conference*, pages 763–782, 1999.

[RM04] I. Rudomín and E. Millán. Point based rendering and displaced subdivision for interactive animation of crowds of clothed characters. *VRIPHYS: Virtual Reality Interaction and Physical Simulation Workshop*, pages 139–148, 2004.

[RMS+96] T. A. Russ, R. M. MacGregor, B. Salemi, K. Price, and R. Nevatia. Veil: Combining knowledge with image understanding. *Proceedings of the DARPA Image Understanding Workshop*, pages 373–380, 1996.

[RP03] P. Reitsma and N. Pollard. Perceptual metrics for character animation; sensitivity to errors in ballistic motion. *ACM Transactions on Graphics (TOG)*, 22(3):537–542, 2003.

[SBMTT99] G. Sannier, S. Balcisoy, N. Magnenat-Thalmann, and D. Thalmann. Vhd:a system for directing real-time virtual actors. *The Visual Computer*, 15(7/8):320–329, 1999.

[Sch97] G. Schaufler. Nailboards: A rendering primitive for image caching in dynamic scenes. *Proceedings of the Eurographics Workshop on Rendering Techniques '97*, pages 151–162, 1997.

[Sch98a] G. Schaufler. Image-based object representation by layered impostors. *VRST '98: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 99–104, 1998.

[Sch98b] G. Schaufler. Per-object image warping with layered impostors. *Rendering Techniques '98: Proceedings of the Eurographics Workshop*, pages 145–156, 1998.

[SCH03] P. Sen, M. Cammarano, and P. Hanrahan. Shadow silhouette maps. *ACM Transactions on Graphics (TOG)*, 22(3):521–526, 2003.

[Sch04] T. Scheuermann. Practical real-time hair rendering and shading. *SIGGRAPH 2004 Sketch*, 2004.

[Sco03] Remington Scott. Sparking life: notes on the performance capture sessions for the *Lord of the Rings: the Two Towers*. *SIGGRAPH Computer Graphics*, 37(4):17–21, 2003.

[SD02] M. Stamminger and G. Drettakis. Perspective shadow maps. In *SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 557–562, 2002.

[SDB97] F. Sillion, G. Drettakis, and B. Bodelet. Efficient impostor manipulation for real-time visualization of urban scenery. *Computer Graphics Forum (Proc. of Eurographics '97)*, 16(3):207–218, 1997.

[SF02] S. Saulters and R. Ferguson. Real-time rendering of dynamically variable scenes using hardware occlusion queries. *Proceedings of the Eurographics Ireland Rendering Workshop*, pages 47–52, 2002.

[SGM04] P. Sander, D. Gosselin, and J. Mitchel. Real-time skin rendering on graphics hardware. *SIGGRAPH 2004 Sketch*, 2004.

[SKvW+92] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. Haeberli. Fast shadows and lighting effects using texture mapping. *SIGGRAPH '92: Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, 26(2):249–252, 1992.

[SLS+96] J. Shade, D. Lischinski, D. Salesin, T. DeRose, and J. Synder. Hierarchical image caching for accelerated walkthroughs of complex environments. *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 75–82, 1996.

[SM01] H. C. Sun and D. N. Metaxas. Automating gait generation. *SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 261–270, 2001.

[SS96] G. Schaufler and W. Sturzlinger. A three dimensional image cache for virtual reality. *Computer Graphics Forum*, 15(3):227–236, 1996.

[Sto96] B. W. Stout. Smart moves: Intelligent path-finding. *Game Developer Magazine*, pages 28–35, October 1996.

[SVNB99] C. Saona-Vázquez, I. Navazo, and P. Brunet. The visibility octree: a data structure for 3d navigation. *Computers & Graphics*, 23(5):635–643, 1999.

[TC00a] F. Tecchia and Y. Chrysanthou. Real-time rendering of densely populated urban environments. *Proceedings of the Eurographics Workshop on Rendering Techniques*, pages 83–88, 2000.

[TC00b] F. Tecchia and Y. Chrysanthou. Real-time visualisation of densely populated urban environments: a simple and fast algorithm for collision detection. *Eurographics UK*, 2000.

[TGKS04] T. True, B. Grantham, B. Kuehne, and D. Shreiner. Performance OpenGL: Platform-independent techniques. *GRAPH '04: Proceedings of the conference on SIGGRAPH 2004 course notes*, page 17, 2004.

[TK96] J. Torborg and J. Kajiya. Talisman: Commodity real-time 3d graphics for the pc. *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 353–364, 1996.

[TLC01] F. Tecchia, C. Loscos, and Y. Chrysanthou. Agent Behaviour Simulator (ABS):a platform for urban behaviour development. *GTEC '01: The First International Game Technology Conference and Idea Expo*, 2001.

[TLC02a] F. Tecchia, C. Loscos, and Y. Chrysanthou. Image based crowd rendering. *IEEE Computer Graphics and Applications*, 22(2):36–43, 2002.

[TLC02b] F. Tecchia, C. Loscos, and Y. Chrysanthou. Visualizing crowds in real-time. *Computer Graphics Forum*, 21(4):753–765, 2002.

[Tre95] B. Treutwein. Adaptive psychophysical procedures. *Vision Research*, 35(17):2503–2522, 1995.

[TT94] X. Tu and D. Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. *SIGGRAPH '94: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 43–50, 1994.

[UdHCT04] B. Ulicny, P. de Heras Ciechomski, and D. Thalmann. Crowdbrush: Interactive authoring of real-time crowd scenes. *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/EUROGRAPHICS Symposium on Computer Animation*, pages 243–252, 2004.

[UT01] B. Ulicny and D. Thalmann. Crowd simulation for interactive virtual environments and vr training systems. *Proceedings of Eurographics Workshop on Computer Animation and Simulation*, pages 163–170, 2001.

[UT02a] B. Ulicny and D. Thalmann. Crowd simulation for virtual heritage. *Proceedings of First International Workshop on 3D Virtual Heritage*, pages 28–32, 2002.

[UT02b] B. Ulicny and D. Thalmann. Towards interactive real-time crowd behaviour simulation. *Computer Graphics Forum*, 21(4):767–775, 2002.

[Vin97] S. Vincke. Real-time pathfinding for multiple objects. *Game Developer Magazine*, June 1997.

[WB03] J. Wang and B. Bodenheimer. An evaluation of a cost metric for selecting transitions between motion segments. *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/EUROGRAPHICS Symposium on Computer Animation*, pages 232–238, 2003.

[Web00] J. Weber. Run-time skin deformation. *In Proceedings of Game Developers Conference*, 2000.

[Wil78] L. Williams. Casting curved shadows on curved surfaces. *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 270–274, 1978.

[Wil83] L. Williams. Pyramidal parametrics. *SIGGRAPH '83: Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques*, pages 1–11, 1983.

[WJ95] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[WK88] A. Witkin and M. Kass. Spacetime constraints. *SIGGRAPH '88: Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, pages 159–168, 1988.

[Wlo03] M. Wloka. "Batch, Batch, Batch": What does it really mean? *Game Developers Conference*, 2003. http://developer.nvidia.com/docs/IO/8230/BatchBatchBatch.pdf.

[Wlo04] M. Wloka. Advanced rendering techniques. *EUROGRAPHICS 2004 Tutorial*, 2004. http://developer.nvidia.com/object/eg_2004_presentations.html.

[WS99] P. Wonka and D. Schmalstieg. Occluder shadows for fast walkthroughs of urban environments. *Computer Graphics Forum (Eurographics '99)*, 18(3):51–60, 1999.

[WS02] M. Wand and W. Straßer. Multi-resolution rendering of complex animated scenes. *Computer Graphics Forum*, 21(3):483–491, 2002.

[Zha98] H. Zhang. *Effective Occlusion Culling for the Interactive Display of Arbitrary Models*. PhD thesis, 1998.