EG 2007 Course on Populating Virtual Environments with Crowds

Organisers

Daniel Thalmann
EPFL VRlab
Email : Daniel.Thalmann@epfl.ch
URL: http://vrlab.epfl.ch

Carol O'Sullivan
Trinity College, Dublin, Ireland
Carol.OSullivan@cs.tcd.ie

Lecturers: Daniel Thalmann (EPFL), Carol O'Sullivan (Trinity College), Barbara Yersin (EPFL), Jonathan Maïm (EPFL), Rachel McDonnell (Trinity College)

Abstract

For many years, it was a challenge to produce realistic virtual crowds for special effects in movies. Now, there is a new challenge: the production of real-time autonomous Virtual Crowds. Real-time crowds are necessary for games, VR systems for training and simulation and crowds in Augmented Reality applications. Autonomy is the only way to create believable crowds reacting to events in real-time. This course will present state-of-the-art techniques and methods.

Keywords: population, crowd simulation, informed virtual environments, autonomous agents, perception

Necessary background and potential target audience for the tutorial: experience with computer animation is recommended but not mandatory. The course is intended for animators, designers, and students in computer science.

Detailed outline of the tutorial

The necessity to model virtual populations occurs in many applications of computer animation and simulation. Such applications encompass several different domains – representative or autonomous agents in virtual environments, perceptual metrics and human factors, training, education, simulation-based design, and entertainment. Realistically reproducing dynamic life in the real-time simulation of virtual environments is also a great challenge.

In this course, we will first present in detail the different approaches to creating virtual crowds, including particle systems with flocking techniques using attraction and repulsion forces, copy and pasting techniques, and agent-based methods.

We will survey methods for animating the individual members that make up a crowd, encompassing a variety of approaches, with particular focus on how example-based synthesis methods can be adapted for crowds. Agent

architectures for scalable crowd simulation will also be presented.

The course will cover the topics of real-time crowd rendering, including image-based/impostor, polygonal and point-based techniques. The topic of Level of Detail (LOD) crowd animation will also be covered, not only for rendering, but also for animation. Perceptual issues with respect to the appearance and movement of crowds of characters will be addressed.

New challenges in the production of real-time crowds for games, VR systems for training and simulation will be presented and analysed, with an emphasis on techniques for highly scalable crowd rendering. The course will be illustrated with many examples from recent movies and real-time applications in Emergency Scenarios and Cultural Heritage (such as adding virtual audiences in Roman or Greek theaters).

Resume of the presenters

Daniel Thalmann is Professor and Director of The Virtual Reality Lab (VRlab) at EPFL, Switzerland. He is a pioneer in research on Virtual Humans. His current research interests include Real-time Virtual Humans in Virtual Reality, Networked Virtual Environments, Artificial Life, and Multimedia. He is coeditor-in-chief of the Journal of Computer Animation and Virtual Worlds and member of the editorial board of the Visual Computer and 4 other journals. Daniel Thalmann was member of numerous Program Committees, Co-chair, and Program Co-chair of several conferences including IEEE VR 2000. He has also organized 5 courses at SIGGRAPH on human animation and crowd simulation. Daniel Thalmann has published numerous papers in Graphics, Animation, and Virtual Reality. He is coeditor of 30 books included the recent “Handbook of Virtual Humans”, published by John Wiley and Sons and coauthor of several books. He received his PhD in Computer Science in 1977 from the University of Geneva and an Honorary Doctorate (Honoris Causa) from University Paul-Sabatier in Toulouse, France, in 2003.

Carol O'Sullivan is an Associate Professor and acting head of the Computer Science department in Trinity College Dublin, where she also leads the Graphics, Vision andVisualisation group (GV2). Her research interests include computer graphics, perception, virtual humans, crowds, and physically-based animation. She has managed a range of projects with significant budgets and successfully supervised many researchers. She has been a member of many IPCs, including the SIGGRAPH and Eurographics papers committees, and has published about 90 peer-reviewed papers. She is an associate editor of IEEE CG&A and Graphical models, and has organised and co-chaired several international conferences and workshops, including Eurographics 2005, the SIGGRAPH/EG Symposium on Computer Animation 2006 and the SIGGRAPH/EG Campfire on Perceptually Adaptive Graphics 2001.

Rachel McDonnell is a postdoctoral researcher at the Graphics, Vision and Visualization Group in Trinity College Dublin where she recently finished her PhD entitled "Realistic Crowd Animation: A Perceptual Approach". Her research interests include perceptually adaptive graphics, real-time crowd simulation, virtual human animation and cloth simulation.

Barbara Yersin is a PhD student at the VRLab, EPFL. She has achieved her Master project at the University of Montreal, after which she has received her Master in Computer Science in March 2005 from EPFL (Swiss Federal Institute of Technology in Lausanne). Her main interests are in computer graphics, particularly crowd simulation and animation.

Jonathan Maïm is PhD student at VRlab at the Swiss Federal Institute of Technology in Lausanne (EPFL). In April 2005, he receives a Master Degree in Computer Science from EPFL after achieving his Master Project at the University of Montréal. His research efforts are concentrated on building an architecture for simulating real-time crowds of virtual humans.

Selected Publications

S. Raupp Musse, D.Thalmann, A Behavioral Model for Real Time Simulation of Virtual Human Crowds, IEEE Transactions on Visualization and Computer Graphics, Vol.7, No2, 2001, pp.152-164.

B. Ulicny, P. de Heras Ciechomski, D. Thalmann, Crowdbrush: Interactive Authoring of Real-time Crowd Scenes, Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation ‘04, 2004, pp.243-252

B. Ulicny, D. Thalmann, Towards Interactive Real-Time Crowd Behavior Simulation, Computer Graphics Forum, 21(4):767-775, December 2002

P. de Heras Ciechomski, S. Schertenleib, J. Maïm, D. Maupu and D. Thalmann, Real-time Shader Rendering for Crowds in Virtual Heritage, VAST '05, 2005

N. Magnenat-Thalmann, D. Thalmann (eds), Handbook of Virtual Humans, John Wiley, 2004

C. O´Sullivan, J. Cassell, H. Vilhjalmsson, J. Dingliana, S. Dobbyn, B. McNamee, C. Peters and T. Giang, Levels of Detail for Crowds and Groups, Computer Graphics Forum, 21(4), 2002.

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

S. Dobbyn, R. McDonnell, L. Kavan, S. Collins, and C. O'Sullivan, Clothing the masses: Real-time clothed crowds with variation. In Eurographics Short Papers 2006, pp. 103–106.

R. McDonnell, S. Dobbyn, and C. O'Sullivan, Crowd Creation Pipeline for Games, In Proceedings of the 9th International Conference on Computer Games, CGames 2006, pp. 183-190.

J. Hamill, R. McDonnell, S. Dobbyn, and C. O'Sullivan, Perceptual Evaluation of Impostor Representations for Virtual Humans and Buildings. Computer Graphics Forum 24(3) (EUROGRAPHICS 2005 Proceedings) 2005.

R. McDonnell, S. Dobbyn, and C. O'Sullivan, LOD Human Representations: A Comparative Study. Proceedings of the First International Workshop on Crowd Simulation (V-CROWDS '05) 2005.

R. McDonnell, S. Dobbyn, S. Collins, and C. O'Sullivan, Perceptual evaluation of LOD clothing for virtual humans. In Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2006, pp. 117–126.

J. Pettré, P. de Heras Ciechomski, J. Maïm, B. Yersin, J.-P. Laumond, D. Thalmann. Real-time navigating crowds: scalable simulation and rendering. Computer Animation and Virtual Worlds. Volume 17, Issue 3-4 , Pages 445 – 455. Special Issue: CASA 2006.

B. Yersin, J. Maïm, P. de Heras Ciechomski, S. Schertenleib and D. Thalmann, Steering a Virtual Crowd Based on a Semantically Augmented Navigation Graph, Proceedings of the First International Workshop on Crowd Simulation (V-CROWDS '05) 2005.

Smooth Movers: Perceptually Guided Human Motion Simulation.Rachel McDonnell, Fional Newell and Carol O'Sullivan. Eurographics/ACM SIGGRAPH Symposium on Computer Animation (SCA'07). To appear (2007)

State-of-the-Art: Real-Time Crowd Simulations

B. Ulicny, P. de Heras Ciechomski, S. R. Musse[2] & D. Thalmann

VRLab, EPFL
CH-1015, Lausanne, Switzerland
branislav.ulicny, pablo.deheras, daniel.thalmann@epfl.ch
http://vrlab.epfl.ch

[2] CROMOS Lab, Universidade do Vale do Rio dos Sinos
Ciências Exatas e Tecnológicas - PIPCA
Av. Unisinos 950
93022-000 - São Leopoldo - RS, Brazil
soraiarm@exatas.unisinos.br
http://www.inf.unisinos.br/ cromoslab

Abstract

Crowds are part of our everyday experience; nevertheless, in virtual worlds they are still relatively rare. In the past, main reasons hindering a wider use of virtual crowds in the real-time domain were their high demands on both general and graphics performance coupled with high costs of content production. The situation is, though, changing fast; market forces are pushing performance of the consumer hardware up, reaching and surpassing performance of professional graphics workstations from just few years ago. With current consumer-grade personal computers it is possible to display 3D virtual scenes with thousands of animated individual entities at interactive framerates. In this report, we present the related works on the subject of groups and crowd simulation discussing several areas such as behavioral simulation, crowd motion control, crowd rendering and crowd scenario authoring.

Keywords: Autonomous agents, behavioral animation, computer graphics, crowd simulations, flocking, image-based rendering, multi-agent systems, impostors, virtual reality.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Three-Dimensional Graphics and Realism]: Animation I.3.3 [Picture/Image Generation]: Display algorithms I.2.11 [Distributed Artificial Intelligence]: Multiagent systems

1. Introduction to crowd simulations

Although collective behavior has been studied since as early as the end of the nineteenth century [LeB95], attempts to simulate it by computer models are quite recent, with most of the works done only in the mid and late nineties. In the past years researchers from a broad range of fields such as architecture [SOHTG99, PT01, TP02], computer graphics [BG96, HB94, MT01, Rey87, TLC02b, UT02, BMdOB03], physics [HM95, HFV00, FHV02], robotics [MS01], safety science [Sim04, Sti00, TM95a], training systems [Bot95, VSMA98, Wil95], and sociology [JPvdS01, MPT92, TSM99] have been creating simulations involving collections of individuals. Nevertheless, despite apparent breadth of the crowd simulation research basis, it can be noted that interdisciplinary exchange of ideas is rare; researchers in one field are usually not very aware of works done in the other fields.

Most approaches were application-specific, focusing on different aspects of the collective behavior, using different modeling techniques. Employed techniques range from those that do not distinguish individuals such as flow and network models in some of the evacuation simulations [TT92], to those that represent each individual as being controlled by more or less complex rules based on physical laws

[HFV00, HIK96], chaos equations [SKN98] or behavioral models in training systems [Wil95] or sociological simulations [JPvdS01].

We can distinguish two broader areas of crowd simulations. The first one is focusing on a **realism of behavioral aspects** with usually simple 2D visualizations like evacuation simulators, sociological crowd models, or crowd dynamics models. In this area, a simulated behavior is usually from a very narrow, controlled range (for example, people just flying to exit or people forming ring crowd structures) with efforts to quantitatively validate correspondence of results to real world observations of particular situations [TM95b]. Ideally, a simulation's results would be then consistent with data sets collected from field observations or video footage of real crowds either by human observers [SM99] or by some automated image processing method [CYC99, MVCL98]. Visualization is used to help to understand simulation results, but it is not crucial. In most cases, a schematic representation, with crowd members represented by colored dots, or sticky figures, is enough, sometimes even preferable as it allows highlighting important information.

In the second area, a main goal is **high quality visualization** (for example, in movie productions and computer games), but usually the realism of the behavior model is not the priority. What is important is a convincing visual result, which is achieved partly by behavior models, partly by human intervention in the production process. A virtual crowd should both look well and be animated in a believable manner, the emphasis of the research being mostly on rendering and animation methods. Crowd members are visualized as fully animated three dimensional figures that are textured and lit to fit into the environment [TLC02b]. Here, behavior models do not necessarily aim to match quantitatively the real world, their purpose is more in alleviating of human animators work, and to be able to respond to inputs in case of interactive applications.

Nevertheless, a recent trend seems to be a **convergence of both areas**, where visualization oriented systems are trying to incorporate better behaviors models to ease creation of convincing animations [Ant98, Cha04] and behavior oriented models are trying to achieve better visualization, especially in the domain of evacuation simulators [Exo04, STE]. We can expect that the most demanding applications would be training systems, where both valid replication of the behaviors and high quality visualization is necessary for a training to be effective.

1.1. Requirements and constrains for crowd modeling

Real-time crowds bring different challenges compared to the systems either involving small number of interacting characters (for example, the majority of contemporary computer games), or non-real-time applications (as crowds in movies, or visualizations of crowd evacuations after off-line model computations). In comparison with single-agent simulations, the main conceptual difference is the **need for efficient variety management** at every level, whether it is visualization, motion control, animation or sound rendering. As everyday experiences hint, virtual humans composing a crowd should look different, move different, react different, sound different and so forth. Even if assuming perfect simulation of a single virtual human would be possible, still creating a simulation involving multiple such humans would be a difficult and tedious task. Methods easing control of many characters are needed; however such methods should still preserve ability to control individual agents.

In comparison with non-real-time simulations, the main technical challenge is **increased demand on computational resources** whether it is general processing power, graphics performance or memory space. One of the foremost constraining factors for real-time crowd simulations is crowd rendering. Fast and scalable methods both to compute behavior, able to take into account inputs not known in advance, and to render large and varied crowds, are needed. While non-real-time simulations are able to take advantage of knowing a full run of the simulated scenario (and therefore, for example, can run iteratively over several possible options selecting the globally best solution), real-time simulations have to react to the situation as it unfolds in the moment.

2. Crowd simulation areas

In order to create a full simulation of the crowd in the virtual environment, many issues have to be solved. The areas of relevance for crowd simulation and some associated questions include:

Crowd behavior generation: How should a virtual crowd respond to changes in their surroundings? How should agents respond to behaviors of other agents? What is an appropriate way of modeling perception for many agents? [Rey87, TT94, HB94, BCN97, BH97, Rey99, Mus00] [UT02, NG03]

Crowd motion control: How should virtual entities move around and avoid collisions with both a static environment and dynamic objects? How can a group move in a coordinated manner? [ALA*01, GKM*01, AMC03, LD04]

Integration of crowds in virtual environments: Which aspects of the environment need to be modeled? Which representation of environmental objects is best suited for fast behavior computation? [FBT99, BLA02, KBT03, LMA03]

Virtual crowd rendering and animation: How to display many animated characters fast? How to display a wide variety of appearances? How to generate varied animations? [ABT00, LCT01, TLC02a, WS02]

Interaction with virtual crowds: How and which information should be exchanged between real and virtual

humans? What is the most efficient metaphor to direct crowds of virtual extras? [FRMS*99, UdHCT04]

Generation of virtual individuals: How to generate a heterogeneous crowd? How to create a population with desired distribution of features? [GKMT01, SYCGMT02]

Authoring of scenarios: How to author complex crowd scenes in an efficient way? How to distribute crowd members in designated areas? How to distribute features over a population? [Che04, UdHCT04]

Many of these aspects are to a greater or lesser extent intertwined. For example, efficiency of rendering constrains the possible variety of behaviors and appearances; higher-level behavior generation controls lower-level motion systems, but the behavior should also respond appropriately to collisions encountered while moving; the behavior model affects interaction possibilities; the environment representation affects possible behaviors; authoring tools allow handling of more complex behavior and environment representations and so on.

3. Overview of crowd simulations

3.1. Crowd evacuation simulators

One of the largest areas where crowd behaviors have been modeled is the domain of safety science and architecture with the dominant application of crowd evacuation simulators. Such systems model movements of a large number of people in usually closed and well-defined spaces like inner areas of buildings [TM95a], subways [Har00], ships [KMKWS00] or airplanes [OGLF98]. Their goal is to help designers to understand the **relation between the organization of space and human behavior** [OM93].

The most common use of evacuation simulators is the modeling of crowd behavior in case of forced evacuation from a confined environment due to some threat like fire or smoke. In such a situation, a number of people have to evacuate the given area, usually through a relatively small number of fixed exits. Simulations are trying to help with answering questions like: Can the area be evacuated within a prescribed time? Where do the hold-ups in the flow of people occur? Where are the likely areas for a crowd surge to produce unacceptable crushing pressure? [Rob99] The most common modeling approach in this area is the use of cellular automata serving both as a representation of individuals and a representation of the environment.

Simulex [TM95a, Sim04] is a computer model simulating the escape movement of persons through large, geometrically complex building spaces defined by 2D floor plans and connecting staircases. Each individual has attributes such as position, body size, angle of orientation and walking speed. Various algorithms as distance mapping, way finding, overtaking, route deviation and adjustment of individual speeds due to proximity of crowd members are used to compute egress simulation, where individual building occupants walk towards and through the exits.

K. Still developed a collection of programs named *Legion* for simulation and analysis of the crowd dynamics in evacuation from constrained and complex environments like stadiums [Sti00]. Dynamics of crowd motion is modeled by mobile cellular automata. Every person in the crowd is treated as an individual, calculating its position by scanning its local environment and choosing an appropriate action.

3.2. Crowd management training systems

The modeling of crowds has also been essential in police and military simulator systems used for training in how to deal with mass gatherings of people.

CACTUS [Wil95] is a system developed to assist in planning and training for public order incidents such as large demonstrations and marches. The software designs are based on a world model in which crowd groups and police units are placed on a digitized map and have probabilistic rules for their interactive behavior. The simulation model represents small groups of people as discrete objects. The behavioral descriptions are in the form of a directed graph where the nodes describe behavioral states (to which correspond actions and exhibited emotions) and transitions represent plausible changes between these states. The transitions depend on environmental conditions and probability weightings. The simulation runs as a decision making exercise that can include pre-event logistic planning, incident management and debriefing evaluation.

Small Unit Leader Non-Lethal Training System [VSMA98] is a simulator for training US Marines Corps in decision making with respect to the use of non-lethal munitions in peacekeeping and crowd control operations. Trainees learn rules of engagement, the procedures for dealing with crowds and mobs and ability to make decisions about the appropriate level of force needed to control, contain, or disperse crowds and mobs. Crowds move within a simulated urban environment along instructor-predefined pathways and respond both to actions of a trainee and to actions of other simulated crowds. Each crowd is characterized by a crowd profile - series of attributes like fanaticism, arousal state, prior experience with non-lethal munitions, or attitude toward Marines. During an exercise, the crowd behavior computer model operates in real time and responds to trainee actions (and inactions) with appropriate simulated behaviors such as loitering, celebrating, demonstrating, rioting and dispersing according to set of Boolean relationships defined by experts.

3.3. Sociological models of crowds

Despite being a field primary interested in studying collective behavior, only a relatively small number of works on crowd simulations have been done in sociology.

McPhail et al. [MPT92] studied individual and collective actions in temporary gatherings. Their model of the crowd is based on perception control theory [Pow73] where each separate individual is trying to control his or her experience in order to maintain a particular relationship to others: in this case it is a spatial relationship with others in a group. The simulation program called *GATHERING* graphically shows movement, milling, and structural emergence in crowds. The same simulation system was later used by Schweingruber [Sch95] to study the effects of reference signals common to coordination of collective behavior and by Tucker et al. [TSM99] to study formation of arcs and rings in temporary gatherings.

Jager et al. [JPvdS01] modeled clustering and fighting in two-party crowds. Crowd is modeled by a multi-agent simulation using cellular automata with rules defining approach-avoidance conflict. The simulation consists of two groups of agents of three different kinds: hardcore, hangers-on and bystanders, the difference between them consisting in the frequency with which they scan their surroundings. The goal of the simulation was to study effects of group size, size symmetry and group composition on clustering, and 'fights'.

3.4. Group behavior in robotics and artificial life

Researchers working in the field of artifical life are interested in exploring how group behavior emerges from local behavioral rules [Gil95]. Software models and groups of robots were designed and experimented with in order to understand how complex behaviors can arise in systems guided by simple rules. The main source of inspiration is nature, where, for example, social insects efficiently solve problems such as finding food, building nests, or division of labor among nestmates by simple interacting individuals without an overseeing global controller. One of the important mechanisms contributing to a distributed control of the behavior is *stigmergy*, indirect interactions among individuals through modifications of the environment [BDT99].

Dorigo introduced *ant systems* inspired by behaviors of real ant colonies [Dor92]. Ant algorithms have been successfuly used to solve a variety of discrete optimization problems including travelling salesman problem, sequential ordering, graph colouring or network routing [BDT00]. Besides insects, also groups of more complex organisms such as flocks of birds, herds of animals and schools of fish have been studied in order to understand principles of their organization. Recently, Couzin et al. presented a model how animals that forage or travel in groups can make decisions even with a small number of informed individuals [CKFL05].

Principles from biological systems were also used to design behavior controllers for autonomous groups of robots. Mataric studied behavior-based control for a group of robots, experimenting with a herd of 20 robots whose behavior repertoire included safe wandering, following, aggregation, dispersion and homing [Mat97]. Molnar and Starke have been working on assignment of robotic units to targets in a manufacturing environments using a pattern formation inspired by pedestrian behavior [MS01]. Martinoli applied swarm intelligence principles to autonomous collective robotics, performing experiments with robots that were gathering scattered objects and cooperating to pull sticks out of the ground [Mar99]. Holland and Melhuish experimented with a group of robots doing sorting of objects based on ant behaviors where ants sort larvae and cocoons [HM99]. In an interesting work a robot was used to control animal behavior, Vaughan et al. developed a mobile robot that gathers a flock of real ducks and manoeuvres them safely to a specied goal position [VSH*00].

3.5. Crowds in virtual worlds

In order to have a persuasive application using crowds in virtual environments, various aspects of the simulation have to be addressed, including behavioral animation, environment modeling and crowd rendering. If there is no satisfactory rendering, even the best behavior model will not be very convincing. If there is no good model of a behavior, even a simulation using the best rendering method will look dumb after only few seconds. If there is no appropriate model of the environment, characters will not behave believably, as they will perform actions at wrong places, or not perform at all.

The goal of behavioral animation is to **ease the work of designers** by letting virtual characters perform autonomously or semi-autonomously complicated motions which otherwise would require large amounts of human animators' work; or, in case of interactive applications, the behavioral models allow characters to **respond to user initiated actions**.

In order for a behavior to make sense, besides characters, also their surrounding environment has to be modeled, not just graphically but also semantically. Indeed, a repertoire of possible behaviors is very dependent on what is and what is not included in a model of the environment. It happens very often that the environment is visually rich, but the interaction of characters with it is minimal.

Finally, for interactive applications, it is necessary to display a varied ensemble of virtual characters in an efficient manner. Rendered characters should visually 'fit' into the environment, they should be affected by light and other effects in the same manner as their surroundings.

Next, we will present representative works for each of these topics grouped according to their main focus.

Behavioral animation of groups and crowds

Human beings are arguably the most complex known creatures, therefore they are also the most complex creatures

to simulate. A behavioral animation of human (and humanoid) crowds is based on foundations of group simulations of much more simple entities, notably flocks of birds [Rey87, GA90] and schools of fish [TT94]. The first procedural animation of flocks of virtual birds was shown in the movie by Amkraut, Girard and Karl called Eurhythmy, for which the first concept [AGK85] was presented at The Electronic Theater at SIGGRAPH in 1985 (final version was later presented at Ars Electronica in 1989). The flock motion was achieved by a global vector force field guiding a flow of flocks [GA90].

In his pioneer work, Reynolds [Rey87] described a distributed behavioral model for simulating aggregate motion of a flock of birds. The technical paper was accompanied by an animated short movie called "Stanley and Stella in: Breaking the Ice" shown at the Electronic Theater at SIGGRAPH '87. The revolutionary idea was that a **complex behavior** of the group of actors can be obtained by **simple local rules** for members of the group instead of some enforced global condition. The flock is simulated as a complex particle system, with the simulated birds (called *boids*) being the particles. Each boid is implemented as an independent agent that navigates according to its local perception of the environment, the laws of simulated physics, and the set of behaviors. The boids try to avoid collisions with one another and with other objects in their environment, match velocities with nearby flock mates and move towards a center of the flock. The aggregate motion of the simulated flock is the result of the interaction of these relatively simple behaviors of the individual simulated birds. Reynolds later extended his work by including various steering behaviors as goal seeking, obstacle avoidance, path following or fleeing [Rey99], and introduced a simple finite-state machines behavior controller and spatial queries optimizations for real-time interaction with groups of characters [Rey00].

Tu and Terzopoulos proposed a framework for animation of artificial fishes [TT94]. Besides complex individual behaviors based on perception of the environment, virtual fishes have been exhibiting unscripted collective motions as schooling and predator evading behaviors analogous to flocking of boids.

An approach similar to boids was used by Bouvier et al. [BG96, BCN97] to simulate human crowds. They used a combination of particle systems and transition networks to model crowds for the visualization of urban spaces. At the lower level, attraction and repulsion forces, analogous to physical electric forces, enable people to move around the environment. Goals generate attraction forces, obstacles generate repulsive force fields. Higher level behavior is modeled by transition networks with transitions depending on time, visiting of certain points, changes of local population densities and global events.

Brogan and Hodgins [BH97, HB94] simulated group behaviors for systems with **significant dynamics**. Compared to the boids, a more realistic motion is achieved by taking into account physical properties of the motion, such as momentum or balance. Their algorithm for controlling the movements of creatures proceeds in two steps: first, a perception model determines the creatures and obstacles visible to each individual, and then a placement algorithm determines the desired position for each individual given the locations and velocities of perceived creatures and obstacles. Simulated systems included groups of one-legged robots, bicycle riders and point-mass systems.

Musse and Thalmann [Mus00, MT01] presented a **hierarchical model** for real-time simulation of virtual human crowds. Their model is based on groups, instead of individuals: groups are more intelligent structures, individuals follow the groups specification. Groups can be controlled with different levels of autonomy: guided crowds follow orders (as go to certain place or play a particular animation) given by the user in run-time; programmed crowds follow a scripted behavior; and autonomous crowds use events and reactions to create more complex behaviors. The environment comprises a set of interest points, which signify goals and waypoints; and a set of action points, which are goals that have associated some actions. Agents move between waypoints following Bezier curves.

Recently, another work was exploring group modeling based on hierarchies. Niederberger and Gross [NG03] proposed an architecture of hierarchical and heterogeneous agents for real-time applications. Behaviors are defined through specialization of existing behavior types and weighted multiple inheritance for creation of new types. Groups are defined through recursive and modulo based patterns. The behavior engine allows for the specification of a maximal amount of time per run in order to guarantee a minimal and constant framerates.

Ulicny and Thalmann [UT01, UT02] presented a crowd behavior simulation with a modular architecture for multi-agent system allowing autonomous and scripted behavior of agents supporting variety. In their system, the behavior is computed in layers, where decisions are made by behavioral rules and execution is handled by hierarchical finite-state machines.

Perceived complexity of the crowd simulation can be increased by using **level of details** (LOD). O'Sullivan et al. [OCV*02] described a simulation of crowds and groups with level of details for geometry, motion and behavior. At the geometrical level, subdivision techniques are used to achieve smooth rendering LOD changes. At the motion level, the movements are simulated using adaptive levels of detail. Animation subsystems with different complexities, as a keyframe player or a real-time reaching module, are activated and deactivated based on heuristics. For the behavior, LOD is employed to reduce the computational costs of updating the behavior of characters that are less important. More complex characters behave according to their

motivations and roles, less complex ones just play random keyframes.

Environment modeling for crowds

Environment modeling is closely related to the behavioral animation. The purpose of the models of the environment is to facilitate simulation of entities dwelling in their surrounding environments. Believability of virtual creatures can be greatly enhanced if they behave in accordance with their surroundings. On the contrary, the suspense of disbelief can be immediately destroyed if they perform something not expected or not permitted in the real world, such as passing through the wall or walking on the water. The greatest efforts have been therefore directed to representations and algorithms preventing 'forbidden' behaviors to occur: till quite recently the two major artificial intelligence issues concerning game development industry were collision avoidance and path-planning [Woo99, DeL00].

Majority of the population in the developed world lives in cities; it is there where most of the human activities take place nowadays. Accordingly, most of the research have been done for **modelling of virtual cities**. Farenc et al. [FBT99] introduced an **informed environment** dedicated to the simulation of virtual humans in the urban context. The informed environment is a database integrating semantic and geometrical information about a virtual city. It is based on a hierarchical decomposition of a urban scene into environment entities, like quarters, blocks, junctions, streets and so on. Entities can contain a description of the behaviors that are appropriate for agents located on them; for example, sidewalk tells that it should be walked on, or bench tells that it should be sit on. Furthermore, the environment database can be used for a path-finding that is customized according to the type of the client requesting the path, so that, for example, a pedestrian will get paths using sidewalks, but a car will get paths going through roads.

Another model of a virtual city for a behavioral animation was presented by Thomas and Donikian [TD00]. Their model is designed with the main emphasis on a traffic simulation of vehicles and pedestrians. The environment database is split into two parts - a hierarchical structure containing a tree of polygonal regions, similar to the informed environment database; and a topological structure with a graph of a road network. Regions contain information on directions of circulation, including possible route changes at intersections. The agents then use the database to navigate through the city.

In a recent work, Sung et al. [SGC04] presented a new approach to control the behavior of a crowd by storing behavioral information into the environment using structures called **situations**. Compared to previous approaches, environmental structures (situations) can overlap; behaviors corresponding to such overlapping situations are then composed using probability distributions. Behavior functions define probabilities of state transitions (triggering motion clips) depending on the state of the environment features or on the past state of the agent.

On the side focused on more generic **path-planning** issues, several works have been done. Kallmann et al. [KBT03] proposed a fast path-planning algorithm based on a fully dynamic constrained Delaunay triangulation. Bayazit et al. [BLA02] used global roadmaps to improve group behaviors in geometrically complex environments. Groups of creatures exhibited behaviors such as homing, goal searching, covering or shepherding, by using rules embedded both in individual flock members and in roadmaps. Tang et al. [TWP03] used a modified A* algorithm working on grid overlayed over a hight-map generated terrain. Recently, Lamarche and Donikian [LD04] presented a topological structure of the geometric environment for a fast hierarchical path-planning and a reactive navigation algorithm for virtual crowds.

Crowd rendering

Real-time rendering of a large number of 3D characters is a considerable challenge; it is able to exhaust system resources quickly even for state of the art systems with extensive memory resources, fast processors and powerful graphic cards. 'Brute-force' approaches that are feasible for a few characters do not scale up for hundreds, thousands or more of them. Several works have been trying to circumvent such limitations by clever use of graphics accelerator capabilities, and by employing methods profiting from the fact that our perception of the scene as a whole is limited.

We can perceive in full details only a relatively small part of a large collection of characters. A simple calculation shows that to treat every crowd member as equal is rather wasteful. Modern screens can display around two millions of pixels at the same time, where fairly complex character can contain approximately ten thousand triangles. Even if assuming that every triangle would be projected to a single pixel, and that there would be no overlap of characters, the screen fully covered by a crowd would contain only around two hundred simultaneously visible characters. Of course, in reality the number would be much smaller, a more reasonable estimate is around a few dozen of fully visible characters, with the rest of the crowd being either hidden behind these prominent characters or taking significantly less screen space. Therefore it makes sense to take full care only of the foremost agents, and to replace the others with some less complex approximations. Level of details techniques then switch visualizations according to position and orientation of the observer.

Billboarded impostors are one of the methods used to speed up crowd rendering. Impostors are partially transparent textured polygons that contain a snapshot of a full 3D character and are always facing the camera. Aubel et al. [ABT00] introduced dynamically generated impostors to

render animated virtual humans. In their approach, an impostor creating process is running in parallel to full 3D simulations, taking snapshots of rendered 3D characters. These cached snapshots are then used over several frames instead of the full geometry until a sufficient movement of either camera or a character will trigger another snapshot, refreshing the impostor texture.

In another major work using impostors, Tecchia et al. [TLC02a] proposed a method for real-time rendering of animated crowd in a virtual city. Compared to the previous method, impostors are not computed dynamically, but are created in a preprocessing step. Snapshots are sampled from viewpoints distributed in the sphere around the character. This process is repeated for every frame of the animation. In run-time, images taken from viewpoints closest to the actual camera position are then used for texturing of the billboard. Additionally, the silhouettes of the impostors are used as shadows projected to a ground surface. Multitexturing is used to add variety by modulating colors of the impostors. In a later work they added lighting using normal maps [TLC02b]. Their method using precomputed impostors is faster than dynamical impostors, however it is very demanding on texture memory, which leads to lower image quality as size of textures per character and per animation frame have to be kept small.

A different possibility for a fast crowd display is to use **point-based rendering techniques**. Wand and Strasser [WS02] presented a multi-resolution rendering approach which unifies image based and polygonal rendering. They create a view dependant octree representations of every keyframe of animation, where nodes store either a polygon or a point. These representations are also able to interpolate linearly from one tree to another so that in-between frames can be calculated. When the viewer is at a long distance, the human is rendered using point rendering; when zoomed in, using polygonal techniques; and when in between, a mix of the two.

An approach that has been getting new life is the one of **geometry baking**. By taking snapshots of vertex positions and normals, complete mesh descriptions are stored for each frame of animation. Since current desktop PCs have large memories many such frames can be stored and re-played. A hybrid approach of both baked geometry and billboarding is to be presented at I3d, where only a few actors are fully geometrical while the vast number of actors are made up of billboards [DHOO05].

What is common to all approaches is instancing of template humans, by changing the texture or color, size, orientation, animation, animation style and position. This is carefully taken care of to smoothly transition from one representation to another so as not to create pops in representation styles. In the billboarding scenario this is done by applying different colors on entire zones such as torso, head, legs and arms. This way the texture memory is used more efficiently as the templates are more flexible. For the geometrical approaches these kind of differences are usually represented using entirely different textures as the humans are too close just to change basic colour for an entire zone [UdHCT04].

Crowds in non-real time productions

One of the domains with a fastest growth of crowd simulations in recent years are special effects. While only ten years ago, there were no digital crowds at all, nowadays almost every blockbuster has some, with music videos, television series and advertisements starting to follow. In comparison with crowds of real extras, virtual crowds allow to significantly reduce costs of production of massively populated scenes and allow for bigger creative freedom because of their flexibility. Different techniques, as replications of real crowd video footage, particle systems or behavioral animation, have been employed to add crowds of virtual extras to shots in a broad range of movies, from historical dramas [Tit97, Gla00, Tro04], through fantasy and science fiction stories [Sta02, The03, Mat03], to animated cartoons [The94, Ant98, A b98, Shr04].

The main factors determining the choice of techniques are the required visual quality and the production costs allowed for the project [Leh02]. It is common to use different techniques even in a single shot in order to achieve the best visuals - for example, characters in the front plane are usually real actors, with 3D characters taking secondary roles in the background.

Although a considerable amount of work was done on crowds in movies, only relatively little information is available, especially concerning more technical details. Most knowledge comes from disparate sources, for example, from "making-of" documentary features, interviews with special effect crew or industry journalist accounts. For big budget productions, the most common approach is **in-house development** of **custom tools** or suites of tools which are used for a particular movie. As the quality of the animation is paramount, large libraries of motion clips are usually used, produced mainly by motion capture of live performers. All production is centered around shots, most of the times only few seconds long. In contrast to real-time simulations, there is little need for continuity of the simulation over longer periods of the time. It is common that different teams of people work on parts of the shots which are then composited in post-processing.

The most advanced crowd animation system for non real-time productions is *Massive*; used to create battle scenes for *The Lord of the Rings* movie trilogy [Mas04]. In *Massive*, every agent makes decisions about its actions depending on its sensory inputs using a brain composed of thousands of logic nodes [Koe02]. According to brain's decision, the motion is selected from extensive library of motion captured clips with precomputed transitions. For example, in the second part of the trilogy over 12 millions of motion captured frames

(equivalent to 55 hours of animation) was used. Massive also uses rigid body dynamics, a physics-based approach to facilitating realistic stunt motion such as falling, or animation of accessories. For example, a combination of physics-based simulation and custom motion capture clips was used to create the scene of "The Flooding of Isengard" where orcs are fleeing from a wall of water and falling down the precipice [Sco03].

In comparison with real-time application, the quality of motion and visuals in non real-time productions is far superior, however it comes at a great cost. For example for *The Lord of the Rings: The Two Towers*, rendering of all digital characters took ten months of computations on thousands computer strong render farm [Doy03].

Crowds in games

In current computer games virtual crowds are still relatively rare. The main reason is that crowds are inherently costly, both in terms of real-time resources requirements and for costs of a production. Nevertheless, the situations is starting to change, with real-time strategy genre leading the way as increase of sizes of involved armies has direct effect on a gameplay [Rom04, The04a].

The main concern for games is the **speed of both rendering and behavior computation**. In comparison with non real-time productions, the quality of both motion and rendering is often sacrificed in a trade-off for fluidity. Similarly to movies production, computer games often inject into virtual world realism coming from real world by using large libraries of animation, which are mostly motion captured. The rendering uses level-of-details techniques, with some titles employing animated impostors [Med02].

To improve costs of behavior computations for games that involve a large number of simulated entities, simulation level-of-detail techniques have been employed [Bro02, Rep03]. In such techniques, behavior is computed only for a characters that are visible or near to be visible to a player. Characters are created in a space around the player with parameters set according to some expected statistical distributions, the player lives in a "simulation bubble". However, handling simulation LOD is much more complex as handling of rendering LOD. It is perfectly correct not to compute visualization for agents that are not visible, but not computing behaviors for hidden agents can lead to an incoherent world. In some games it is common that the player causes some significant situation (for example, traffic jam), looks away, and then after looking back, the situation is changed in an unexpected way (a traffic jam is "magically" resolved).

In case the scenario deals with hundreds or thousands of entities, many times the selectable unit with distinct behavior is a formation of troops, not individual soldiers. What appears to be many entities on the screen is indeed only one unit being rendered as several visually separated parts [Sho00, Med02, Pra03].

A special case are sport titles such as various football, basketball or hockey simulations, where there is a large spectator crowd, however only of very low details. In the most cases there is not even a single polygon for every crowd member (compared to individual impostors in strategy games). Majority of the crowd is just texture with transparency applied to stadium rows, or to a collection of rows, and only few crowd members, close to the camera can be very low polygon count 3D models.

Crowd scenario authoring

No matter how good is a crowd rendering or a behavior model, a virtual crowd simulation is not very useful, if it is too difficult to produce a content for it. The authoring possibilities are an important factor influencing the usability of crowd simulation system, especially when going beyond a limited number of "proof-of-concept" scenarios. When increasing the number of involved individuals it becomes more difficult to create unique and varied content of scenarios with large numbers of entities. Solving one set of problems for crowd simulation (such as fast rendering and behavior computation for large crowds) creates a new problem of how to create content for crowd scenarios in an efficient manner.

Only recently, researchers started to explore the ways how to author crowd scenes. Anderson et al. [AMC03] achieved interesting results for a particular case of flocking animation following constrains. Their method can be used, for instance, to create and animate flocks moving in shapes. Their algorithm generates a constrained flocking motion by iterating simulation forwards and backwards. Nevertheless, their algorithm can get very costly when increasing the number of entities and simulation time.

Ulicny et al. [UdHCT04] proposed a method to create complex crowd scenes in an intuitive way using a CrowdBrush tool. By employing a brush metaphor, analogous to the tools used in image manipulation programs, the user can distribute, modify and control crowd members in real-time with immediate visual feedback. This approach works well for creation and modification of spatial features, however the authoring of temporal aspects of the scenario is limited.

Sung et al. [SGC04] used a situation-based distributed control mechanism that gives each agent in a crowd specific details about how to react at any given moment based on its local environment. A painting interface allows to specify situations easily by drawing their regions on the environment directly like drawing a picture on the canvas. Compared to previous work where the user adds, modifies and deletes crowd members, here the interface operates on the environment.

Chenney [Che04] presented a novel technique for representing and designing velocity fields using flow tiles. He applied his method on a city model with tiles defining the flow

of people through the city streets. Flow tiles drive the crowd using the velocity to define the direction of travel for each member. The use of divergence free flows to define crowd motion ensures that, under reasonable conditions, the agents do not require any form of collision detection.

4. Discussion

We presented an overview of the works on crowd simulations done in different fields such as sociology, safety science, training systems, computer graphics or entertainment industry. Based on the analysis of published research works and data available on industry applications, we made following observations.

Domain specificity: While some of the know-how is transferable across the fields, each of the domains dealing with crowds poses unique challenges and requires different solutions. It is indeed the targeted application that drives most of the design choices while creating a simulation of the crowd. There is no "silver bullet" solution, the ultimate crowd simulation that would be fitting all purposes. Features that are advantageous for one purpose are disadvantages in the other and trade-offs have to be resolved in a different manner. For example, most of the crowd evacuation simulators use discrete 2D grid representation of the world as it is easier to handle, to analyze and to validate. However, such representation is too coarse for crowd simulations with 3D articulated bodies. The controller that drives a virtual humanoid in a movie or a computer game has to be more complex than the behavior model that drives 2D dots. It is not enough to decide global position and orientation of the entity; features like type of the motion, its dynamics and transition, or biomechanical constrains have to be taken into account. A simple re-application of evacuation models to 3D visualizations leads to awkward, unrealistic looking animations. Humans can get easily enchanted by seeing artificial objects performing behaviors that are not expected from them (such as geometrical primitives fleeing in 2D labyrinth), but are very critical at evaluating of (what are expected to be) the other people. Motions that look reasonable for 2D dots can look very artificial when applied to virtual humans. Even a relatively straightforward transition from segmented skeletons to fully skinned bodies in many cases reveals disturbing imperfections in the motion. For applications where the visual quality is the most important (as in movies or games), the behavior has to be constrained by availability of motions and transitions among the motions (for example, when using physically based simulation [HB94] or motion graphs [SGC04]).

Application focus: The consequence of the crowd models being domain specific is that in the majority of cases the applications are focusing either on the realism of behavioral aspects, or on the quality of the visualization. The most representative examples of the former category are evacuation simulations, which are usually validated on a large scale statistical parameters such as the number of the people passing through a particular exit in a defined time interval. Behaviors of individuals are not detailed and not defined beyond the narrow scope of the simulation; for example, before or after the incident people are either static or have random Brownian motion. The examples of the latter category are crowds in movies and games, where the goal of the behavior model is to alleviate designers from the tedious tasks of orchestration of animation for large number of entities or to respond to the actions of the user. The repertoire of behaviors is larger; for example, as the most common use of virtual crowds are battles scenes, virtual armies have to be able to navigate around the environment, to attack using different weapons and to defend themselves against various enemies. The most challenging area for crowd simulation are training simulations as there is a need for both behavior realism and persuasive visualization. Present crowd management training systems have been focusing on training strategical skills therefore giving more emphasis on behavioral simulation with visualization being only schematic. Tactical on-site training of crowd management with the trainee immersed in the virtual world is not yet explored.

Crowd models: It is difficult to transfer current knowledge about real crowds from social sciences into crowd simulations. Most of the sociology work on crowds is about macroscopic behavior, not directly dealing with actions of a particular person in particular situation in particular time instance. Methodological observations about microscopic behaviors are sparse: sociological models based on collected real data have a limited scope. The quality of the crowd behavior model is prominent in safety science applications; however, despites calls for including more knowledge about psychology into evacuation models [Sim95], most of the current applications still model behavior of the crowd based more on physical than on psychological principles. Demands on crowd models are different for entertainment industry applications. For production purposes it is preferable to be able to control the crowd instead of just observing the results of the model. Emergent behavior has sense as far as it alleviates designers from tedious tasks. The crowd can be controlled "top-down" where the group behavior is imposed by design, or "bottom-up", where the collective behavior emerges from the behavior of individuals. Group based approaches have the advantage of easier handling when group membership does not need to change, however they bring the disadvantage of overhead when group membership changes often.

Trends: Virtual crowds are a relatively new topic, with majority of the research and commercial applications done in the past few years, especially concerning real-time crowds. The most visible trend is the increase of the number of simulated entities; new techniques together with rapidly evolving hardware allows to handle bigger crowds. Another recently appearing trend is about going beyond simple quantitative improvements towards increasing of complexity of entities at all levels - whether it is visualization, anima-

tion, or behaviors. Both quantitative and qualitative improvements require novel methods, as in the most cases it is not straightforward to apply the method that work for small number of entities to a large crowd. Similarly to other areas in computer graphics and virtual reality simulations, the major driving force of the innovation starts to be the entertainment industry resulting from the large investments due to increasing revenues from entertainment applications. For example, many movies with virtual crowds were blockbusters with revenues in order of hundreds of millions of the dollars and more [Sta02, Mat03, The03, Shr04] allowing to finance large internal research and development (R&D) teams. Even the military, which used to be one of the largest traditional sponsors of the simulation research, starts to use in some cases commercial entertainment technologies for its training instead of costly own R&D [Mac01, ZHM*03].

5. Future challenges and conclusions

We see several possible directions for future research in the area of interactive crowd simulations:

Heterogeneity: In current crowd simulation systems, the whole crowd is constituted by the same type of agents. Even while creating the individuality of the agents by varying parameters, the principle of the behavior computation is the same for every entity. It is possible to create a heterogeneous crowd simulation, where different agents can have completely different behavior computation engines. Such architecture could, for example, lead to an increase of the behavioral variety, while keeping individuals simpler compared to a homogeneous simulation with the same variety.

Scalability: In order to increase the number of simulated entities, the crowd simulation should be scalable [SGC04]. This can be achieved, for example, by using behavior and animation level-of-details[ACF01, AW04], where there are a different computational demands for agents, depending on their relative position to the observer. The behavior model should then allow to work with different fidelity, for example, by using iterative algorithms, or also heterogeneous crowds could be employed.

Variety: The variety of the virtual crowd can be enhanced by adapting methods, capable of producing higher levels of the variety, for the crowd simulations. The natural candidates are methods, which deal with variety sources in the real people, such as parametric generation of bodies [Seo03] or faces [BV99].

Parallelization: The computation of the crowd simulation can be speeded up by using parallelization [QMHZ03]. However, the parallelization of the agents becomes practical only for the hardware that supports a parallel execution of more threads than there are potentially parallelizable application components. For example, recently US military experimented with a combat simulation running on 128 node Linux cluster handling 100.000 entities (which means that each sequential node took care of on average 780 entities) [The04b].

The rapid adoption of the crowd simulation in movies and other non real-time productions in recent years shows that there is a great demand for virtual crowds. It is not so difficult to imagine why - humans are social creatures and real world reflects this fact, most of the people are surrounded by other people. It is therefore expected to see crowds in the works of both fact and fiction.

A similar reasoning holds also for interactive virtual environments such as computer games, training systems or educational applications - we expect to see them populated. However, while in movies it can be still possible, even if not practical, to use crowd of real extras, interactive applications have to rely fully on the virtual crowds. As already current generation personal computers are capable of handling thousands of real-time virtual characters, we believe that in coming years there will be more and more interactive virtual crowds.

We can expect to see a convergence between non real-time and real-time domains, in a manner similar to other areas in computer graphics. The convergence will be fueled both by increases in the power of both general purpose and graphics processors and by the development of novel methods and algorithms. In non real-time applications, the real-time methods can be used to improve the productivity for creating crowd scenes because of shorter production cycles and immediacy of the changes allowing new ways of authoring. On the other hand, in real-time applications, there will be improvements in quality of both rendering and behaviors moving towards the results possible before only by lengthy computations in non-real time productions.

References

[A b98] A bug's life, 1998. movie homepage, http://www.pixar.com/featurefilms/abl. 7

[ABT00] AUBEL A., BOULIC R., THALMANN D.: Real-time display of virtual humans: Levels of detail and impostors. *IEEE Transactions on Circuits and Systems for Video Technology 10*, 2 (2000), 207–217. 2, 6

[ACF01] ARIKAN O., CHENNEY S., FORSYTH D. A.: Efficient multi-agent path planning. In *Computer Animation and Simulation '01* (2001), Magnenat-Thalmann N., Thalmann D., (Eds.), SpringerComputerScience, Springer-Verlag Wien New York, pp. 151–162. Proc. of the Eurographics Workshop in Manchester, UK, September 2–3, 2001. 10

[AGK85] AMKRAUT S., GIRARD M., KARL G.: Motion studies for a work in progress entitled "Eurythmy". *SIGGRAPH Video Review*, 21 (1985). (second item, time code 3:58 to 7:35). 4

[ALA*01] ASHIDA K., LEE S.-J., ALLBECK J. M., SUN H., BADLER N. I., METAXAS D.: Pedestrians: Creating agent behaviors through statistical analysis of observation data. In *Proc. Computer Animation '01* (2001), IEEE Press. 2

[AMC03] ANDERSON M., MCDANIEL E., CHENNEY S.: Constrained animation of flocks. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'03)* (2003), pp. 286–297. 2, 8

[Ant98] AntZ, 1998. movie homepage, http://www.pdi.com/feature/antz.htm. 2, 7

[AW04] AHN J., WOHN K.: Motion level-of-detail: A simplification method on crowd scene. In *Proc. Computer Animation and Social Agents '04* (2004), pp. 129–137. 10

[BCN97] BOUVIER E., COHEN E., NAJMAN L.: From crowd simulation to airbag deployment: particle systems, a new paradigm of simulation. *Journal of Electrical Imaging 6*, 1 (January 1997), 94–107. 2, 5

[BDT99] BONABEAU E., DORIGO M., THERAULAZ G.: *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999. 4

[BDT00] BONABEAU E., DORIGO M., THERAULAZ G.: Inspiration for optimization from social insect behaviour. *Nature 406* (2000), 39–42. 4

[BG96] BOUVIER E., GUILLOTEAU P.: Crowd simulation in immersive space management. In *Proc. Eurographics Workshop on Virtual Environments and Scientific Visualization '96* (1996), Springer-Verlag, pp. 104–110. 1, 5

[BH97] BROGAN D., HODGINS J.: Group behaviors for systems with significant dynamics. *Autonomous Robots 4* (1997), 137–153. 2, 5

[BLA02] BAYAZIT O. B., LIEN J.-M., AMATO N. M.: Better group behaviors in complex environments using global roadmaps. In *Proc. Artificial Life '02* (2002). 2, 6

[BMdOB03] BRAUN A., MUSSE S. R., DE OLIVEIRA L. P. L., BODMANN B. E. J.: Modeling individual behaviors in crowd simulation. In *Proc. Computer Animation and Social Agents '03* (2003). 1

[Bot95] BOTTACI L.: A direct manipulation interface for a user enhanceable crowd simulator. *Journal Of Intelligent Systems 5*, 2-4 (1995), 249–272. 1

[Bro02] BROCKINGTON M.: Level-of-detail AI for a large role-playing game. In *AI Game Programming Wisdom* (2002), Rabin S., (Ed.), Charles River Media. 8

[BV99] BLANZ B., VETTER T.: A morphable model for the synthesis of 3d faces. In *Proc. Siggraph '99* (1999), pp. 187–194. 10

[Cha04] Character Studio, 2004. http://www.discreet.com/products/cs. 2

[Che04] CHENNEY S.: Flow tiles. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'04)* (2004), pp. 233–245. 2, 8

[CKFL05] COUZIN I. D., KRAUSE J., FRANKS N. R., LEVIN S. A.: Effective leadership and decision-making in animal groups on the move. *Nature 433* (2005), 513–516. 4

[CYC99] CHOW T. W. S., YAM J. Y.-F., CHO S.-Y.: Fast training algorithm for feedforward neural networks: application to crowd estimation at underground stations. *Artificial Intelligence in Engineering 13* (1999), 301–307. 2

[DeL00] DELOURA M. (Ed.): *Game Programming Gems*. Charles River Media, 2000. 6

[DHOO05] DOBBYN S., HAMILL J., O'CONOR K., O'SULLIVAN C.: Geopostors: A real-time geometry/impostor crowd rendering system. In *Proc. ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games* (2005). 7

[Dor92] DORIGO M.: *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992. 4

[Doy03] DOYLE A.: The two towers. *Computer Graphics World* (February 2003). 7

[Exo04] Exodus, 2004. the evacuation model for the safety industry, homepage, http://fseg.gre.ac.uk/exodus. 2

[FBT99] FARENC N., BOULIC R., THALMANN D.: An informed environment dedicated to the simulation of virtual humans in urban context. In *Proc. Eurographics'99* (1999), Blackwell, pp. 309–318. 2, 6

[FHV02] FARKAS I., HELBING D., VICSEK T.: Mexican waves in an excitable medium. *Nature 419* (2002), 131–132. 1

[FRMS*99] FARENC N., RAUPP MUSSE S., SCHWEISS E., KALLMANN M., AUNE O., BOULIC R., THALMANN D.: A paradigm for controlling virtual humans in urban environment simulations. *Applied Artificial Intelligence Journal - Special Issue on Intelligent Virtual Environments 14*, 1 (1999), 69–91. 2

[GA90] GIRARD M., AMKRAUT S.: Eurhythmy: Concept and process. *The Journal of Visualization and Computer Animation 1*, 1 (1990), 15–17. Presented at The Electronic Theater at SIGGRAPH '85. 4

[Gil95] GILBERT N.: Simulation: an emergent perspective. In *New technologies in the social sciences* (Bournemouth, UK, 27-29th October 1995). 4

[GKM*01] GOLDENSTEIN S., KARAVELAS M., METAXAS D., GUIBAS L., AARON E., GOSWAMI A.: Scalable nonlinear dynamical systems for agent steering and crowd simulation. *Computers & graphics 25*, 6 (2001), 983–998. 2

[GKMT01] GOTO T., KSHIRSAGAR S., MAGNENAT-THALMANN N.: Automatic face cloning and animation. *IEEE Signal Processing Magazine 18*, 3 (2001). 2

[Gla00] Gladiator, 2000. movie homepage, http://www.dreamworks.com. 7

[Har00] HAREESH P.: Evacuation simulation: Visualisation using virtual humans in a distributed multi-user immersive VR system. In *Proc. VSMM '00* (2000). 3

[HB94] HODGINS J., BROGAN D.: Robot herds: Group behaviors for systems with significant dynamics. In *Proc. Artificial Life IV* (1994), pp. 319–324. 1, 2, 5, 9

[HFV00] HELBING D., FARKAS I., VICSEK T.: Simulating dynamical features of escape panic. *Nature 407* (2000), 487–490. 1

[HIK96] HOSOI M., ISHIJIMA S., KOJIMA A.: Dynamical model of a pedestrian in a crowd. In *Proc. IEEE International Workshop on Robot and Human Communication '96* (1996). 1

[HM95] HELBING D., MOLNAR P.: Social force model for pedestrian dynamics. *Phys. Rev. E 51* (1995), 4282–4286. 1

[HM99] HOLLAND O. E., MELHUISH C.: Stigmergy, self-organisation, and sorting in collective robotics. *Artificial Life 5* (1999), 173–202. 4

[JPvdS01] JAGER W., POPPING R., VAN DE SANDE H.: Clustering and fighting in two-party crowds: Simulating the approach-avoidance conflict. *Journal of Artificial Societies and Social Simulation 4*, 3 (2001). 1, 4

[KBT03] KALLMANN M., BIERI H., THALMANN D.: Fully dynamic constrained delaunay triangulations. In *Geometric Modelling for Scientific Visualization* (2003), Brunnett G., Hamann B., Mueller H., Linsen L., (Eds.), Springer-Verlag, pp. 241–257. 2, 6

[KMKWS00] KLÜPFEL H., MEYER-KÖNIG M., WAHLE J., SCHRECKENBERG M.: Microscopic simulation of evacuation processes on passenger ships. In *Theoretical and Practical Issues on Cellular Automata* (2000), Bandini S., Worsch T., (Eds.), Springer, London, pp. 63–71. 3

[Koe02] KOEPPEL D.: Massive attack. *Popular Science* (November 2002). 7

[LCT01] LOSCOS C., CHRYSANTHOU Y., TECCHIA F.: Real-time shadows for animated crowds in virtual cities. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST'01)* (New York, Nov. 15–17 2001), Shaw C., Wang W., (Eds.), ACM Press, pp. 85–92. 2

[LD04] LAMARCHE F., DONIKIAN S.: Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum 23*, 3 (2004). (Proc. Eurographics '04). 2, 6

[LeB95] LEBON G.: *Psychologie des Foules*. Alcan, Paris, 1895. 1

[Leh02] LEHANE S.: Digital extras. *Film and Video Magazine* (July 2002). 7

[LMA03] LOSCOS C., MARCHAL D., A.MEYER: Intuitive crowd behavior in dense urban environments using local laws. In *Proc. Theory and Practice of Computer Graphics (TPCG'03)* (2003). 2

[Mac01] MACEDONIA M.: Games, simulation, and the military education dilemma. In *The Internet and the University: 2001 Forum* (2001), Devlin M., Larson R., Meyerson J., (Eds.), Educause. 9

[Mar99] MARTINOLI A.: *Swarm Intelligence in Autonomous Collective Robotics: From Tools to the Analysis and Synthesis of Distributed Collective Strategies*. PhD thesis, EPFL, Lausanne, 1999. 4

[Mas04] MASSIVE, 2004. Crowd animation software for visual effects, http://www.massivesoftware.com. 7

[Mat97] MATARIC M.: Behavior-based control: Examples from navigation, learning, and group behavior. *Journal of Experimental and Theoretical Artificial Intelligence* (1997), 323–336. 4

[Mat03] Matrix, 2003. movie homepage, http://whatisthematrix.warnerbros.com. 7, 9

[Med02] Medieval: Total War, 2002. game homepage, http://www.totalwar.com. 8

[MPT92] MCPHAIL C., POWERS W., TUCKER C.: Simulating individual and collective actions in temporary gatherings. *Social Science Computer Review 10*, 1 (Spring 1992), 1–28. 1, 3

[MS01] MOLNAR P., STARKE J.: Control of distributed autonomous robotic systems using principles of pattern formation in nature and pedestrian behavior. *IEEE Trans. Syst. Man Cyb. B 31*, 3 (June 2001), 433–436. 1, 4

[MT01] MUSSE S. R., THALMANN D.: A hierarchical model for real time simulation of virtual human crowds. *IEEE Transactions on Visualization and Computer Graphics 7*, 2 (April-June 2001), 152–164. 1, 5

[Mus00] MUSSE S. R.: *Human Crowd Modelling with Various Levels of Behaviour Control*. PhD thesis, EPFL, Lausanne, 2000. 2, 5

[MVCL98] MARANA A. N., VELASTIN S. A., COSTA L. F., LOTUFO R. A.: Automatic estimation of crowd density using texture. *Safety Science 28*, 3 (1998), 165–175. 2

[NG03] NIEDERBERGER C., GROSS M.: Hierarchical and heterogenous reactive agents for real-time applications. *Computer Graphics Forum 22*, 3 (2003). (Proc. Eurographics'03). 2, 5

[OCV*02] O'SULLIVAN C., CASSELL J., VILHJÁLMSSON H., DINGLIANA J., DOBBYN S., MCNAMEE B., PETERS C., GIANG T.: Levels of detail for crowds and groups. *Computer Graphics Forum 21*, 4 (Nov. 2002), 733–741. 5

[OGLF98] OWEN M., GALEA E. R., LAWRENCE P. J., FILIPPIDIS L.: The numerical simulation of aircraft evacuation and its application to aircraft design and certification. *The Aeronautical Journal 102*, 1016 (1998), 301–312. 3

[OM93] OKAZAKI S., MATSUSHITA S.: A study of simulation model for pedestrian movement with evacuation and queuing. In *Proc. International Conference on Engineering for Crowd Safety '93* (1993). 3

[Pow73] POWERS W. T.: *The Control of Perception*. Aldine, Chicago, 1973. 3

[Pra03] Praetorians, 2003. game homepage, http://praetorians.pyrostudios.com. 8

[PT01] PENN A., TURNER A.: Space syntax based agent simulation. In *Pedestrian and Evacuation Dynamics*, Schreckenberg M., Sharma S., (Eds.). Springer-Verlag, Berlin, 2001. 1

[QMHZ03] QUINN M. J., METOYER R. A., HUNTER-ZAWORSKI K.: Parallel implementation of the social forces model. In *Proc. Pedestrian and Evacuation Dynamics'03* (2003), Galea E., (Ed.). 10

[Rep03] Republic: the Revolution, 2003. game homepage, http://www.elixir-studios.co.uk/nonflash/republic/republic.htm. 8

[Rey87] REYNOLDS C. W.: Flocks, herds, and schools: A distributed behavioral model. In *Proc. SIGGRAPH '87* (1987), pp. 25–34. 1, 2, 4

[Rey99] REYNOLDS C. W.: Steering behaviors for autonomous characters. In *Proc. Game Developpers Conference '99* (1999), pp. 763–782. 2, 5

[Rey00] REYNOLDS C. W.: Interaction with groups of autonomous characters. In *Proc. Game Developpers Conference '00* (2000), pp. 449–460. 5

[Rob99] ROBBINS C.: Computer simulation of crowd behaviour and evacuation. *ECMI Newsletter*, 25 (March 1999). 3

[Rom04] Rome: Total War, 2004. game homepage, http://www.totalwar.com. 8

[Sch95] SCHWEINGRUBER D.: A computer simulation of a sociological experiment. *Social Science Computer Review 13*, 3 (1995), 351–359. 3

[Sco03] SCOTT R.: Sparking life: Notes on the performance capture sessions for 'The Lord of the Rings: The Two Towers'. *ACM SIGGRAPH Computer Graphics 37*, 4 (2003), 17–21. 7

[Seo03] SEO H.: *Parameterized Human Body Modeling*. PhD thesis, University of Geneva, 2003. 10

[SGC04] SUNG M., GLEICHER M., CHENNEY S.: Scalable behaviors for crowd simulation. *Computer Graphics Forum 23*, 3 (2004). Proc. Eurographics '04. 6, 8, 9, 10

[Sho00] Shogun: Total War, 2000. game homepage, http://www.totalwar.com. 8

[Shr04] Shrek 2, 2004. movie homepage, http://www.shrek2.com. 7, 9

[Sim95] SIME J.: Crowd psychology and engineering. *Safety Science 21* (1995), 1–14. 9

[Sim04] Simulex, 2004. evacuation modeling software, product information, http://www.ies4d.com. 1, 3

[SKN98] SAIWAKI N., KOMATSU T., NISHIDA S.: Automatic generation of moving crowds in the virtual environments. In *Proc. AMCP '98* (1998). 1

[SM99] SCHWEINGRUBER D., MCPHAIL C.: A method for systematically observing and recording collective action. *Sociological Methods & Research 27*, 4 (May 1999), 451–498. 2

[SOHTG99] SCHELHORN T., O'SULLIVAN D., HAKLAY M., THURSTAIN-GOODWIN M.: Streets: an agent-based pedestrian model. In *Proc. Computers in Urban Planning and Urban Management* (1999). 1

[Sta02] Star Wars, 2002. movie homepage, http://www.starwars.com/. 7, 9

[STE] STEPS, Simulation of Transient Evacuation and Pedestrian movements. http://www.fusion2.mottmac.com/html/06/software.cfm. 2

[Sti00] STILL G.: *Crowd Dynamics*. PhD thesis, Warwick University, 2000. 1, 3

[SYCGMT02] SEO H., YAHIA-CHERIF L., GOTO T., MAGNENAT-THALMANN N.: Genesis : Generation of e-population based on statistical information. In *Proc. Computer Animation '02* (2002), IEEE Press. 2

[TD00] THOMAS G., DONIKIAN S.: Modelling virtual cities dedicated to behavioural animation. In *Proc. Eurographics '00* (2000), vol. 19, pp. 71–80. 6

[The94] The Lion King, 1994. movie homepage, http://disney.go.com/disneyvideos/animatedfilms/lionking. 7

[The03] The Lord of the Rings, 2003. movie homepage, http://www.lordoftherings.net. 7, 9

[The04a] The Lord of the Rings, The Battle for Middle Earth, 2004. game homepage, http://www.eagames.com/pccd/lotr_bfme. 8

[The04b] The wars of the virtual worlds,, 2004. University of Southern California, http://www.isi.edu/stories/96.html. 10

[Tit97] Titanic, 1997. movie homepage, http://www.titanicmovie.com. 7

[TLC02a] TECCHIA F., LOSCOS C., CHRYSANTHOU Y.: Image-based crowd rendering. *IEEE Computer Graphics and Applications 22*, 2 (March-April 2002), 36–43. 2, 6

[TLC02b] TECCHIA F., LOSCOS C., CHRYSANTHOU Y.: Visualizing crowds in real-time. *Computer Graphics Forum 21*, 4 (November 2002), 753–765. 1, 2, 7

[TM95a] THOMPSON P., MARCHANT E.: A computer-model for the evacuation of large building population. *Fire Safety Journal 24*, 2 (1995), 131–148. 1, 3

[TM95b] THOMPSON P., MARCHANT E.: Testing and application of the computer model 'simulex'. *Fire Safety Journal 24*, 2 (1995), 149–166. 2

[TP02] TURNER A., PENN. A.: Encoding natural movement as an agent-based system: An investigation into human pedestrian behaviour in the built environment. *Environment and Planning B: Planning and Design 29* (2002), 473–490. 1

[Tro04] Troy, 2004. movie homepage, http://troymovie.warnerbros.com. 7

[TSM99] TUCKER C., SCHWEINGRUBER D., MCPHAIL C.: Simulating arcs and rings in temporary gatherings. *International Journal of Human-Computer Systems 50* (1999), 581–588. 1, 4

[TT92] TAKAHASHI T. S. H.: Behavior simulation by network model. *Memoirs of Kougakuin University*, 73 (1992), 213–220. 1

[TT94] TU X., TERZOPOULOS D.: Artificial fishes: Physics, locomotion, perception, behavior. In *Proc. SIGGRAPH '94* (1994), pp. 43–50. 2, 4, 5

[TWP03] TANG W., WAN T. R., PATEL S.: Real-time crowd movement on large scale terrains. In *Proc. Theory and Practice of Computer Graphics* (2003), IEEE. 6

[UdHCT04] ULICNY B., DE HERAS CIECHOMSKI P., THALMANN D.: Crowdbrush: Interactive authoring of real-time crowd scenes. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'03)* (2004), pp. 243–252. 2, 7, 8

[UT01] ULICNY B., THALMANN D.: Crowd simulation for interactive virtual environments and vr training systems. In *Proc. Eurographics Workshop on Animation and Simulation* (2001), Springer-Verlag, pp. 163–170. 5

[UT02] ULICNY B., THALMANN D.: Towards interactive real-time crowd behavior simulation. *Computer Graphics Forum 21*, 4 (Dec. 2002), 767–775. 1, 2, 5

[VSH*00] VAUGHAN R. T., SUMPTER N., HENDERSON J., FROST A., CAMERON S.: Experiments in automatic flock control. *Robotics and Autonomous Systems 31* (2000), 109–177. 4

[VSMA98] VARNER D., SCOTT D., MICHELETTI J., AICELLA G.: UMSC Small Unit Leader Non-Lethal Trainer. In *Proc. ITEC'98* (1998). 1, 3

[Wil95] WILLIAMS J.: *A Simulation Environment to Support Training for Large Scale Command and Control Tasks*. PhD thesis, University of Leeds, 1995. 1, 3

[Woo99] WOODCOCK S.: Game AI: The state of the industry. *Game Developer Magazine* (August 1999). 6

[WS02] WAND M., STRASSER W.: Multi-resolution rendering of complex animated scenes. *Computer Graphics Forum 21*, 3 (2002). (Proc. Eurographics'02). 2, 7

[ZHM*03] ZYDA M., HILES J., MAYBERRY A., WARDYNSKI C., CAPPS M., OSBORN B., SHILLING R., ROBASZEWSKI M., DAVIS M.: Entertainment R&D for defense. *IEEE Computer Graphics and Applications* (January / February 2003), 2–10. 9

Computerized Models for Virtual Humans and Crowds

Daniel Thalmann,

EPFL VRlab, Switzerland

Abstract

Interactive systems, games, VR and multimedia systems require more and more flexible Virtual Humans with individualities. There are mainly two approaches:
1) Recording the motion using motion capture systems, then to try to alterate such a motion to create this individuality. This process is tedious and there is no reliable method at this stage.
2) Creating computational models which are controlled by a few parameters. One of the major problem is to find such models and to compose them to create complex motion. Such models can be created for walking, grasping, but also for groups and crowds.

1. Introduction

Virtual humans simulations are becoming each time more popular. Nowadays many systems are available to animate virtual humans. Such systems encompass several different domains, as: autonomous agents in virtual environments, human factors analysis, training, education, virtual prototyping, simulation-based design, and entertainment. In the context of Virtual Humans, a Motion Control Method (MCM) specifies how the Virtual Human is animated and may be characterized according to the type of information it privileged in animating this Virtual Human. For example, in a keyframe system for an articulated body, the privileged information to be The problem is basically to be able to generate variety among a finite set of motion requests and then to apply it to either an individual or a member of a crowd. A single autonomous agent and a member of the crowd present the same kind of 'individuality'. The only difference is at the level of the modules that control the main set of actions. With this formulation, one can also see that the personality of an agent (i.e. the set of noisy actions) can be preserved whenever it is in a crowd, alone. Figure 1 shows a group of Virtual Humans in a room and Figure 2 Virtual Humans in city.

The problem is basically to be able to generate variety among a finite set of motion requests and then to apply it to either an individual or a member of a crowd. A single autonomous agent and a member of the crowd present the same kind of 'individuality'. The only difference is at the level of the modules that control the main set of actions. With this formulation, one can also see that the personality of an agent (i.e. the set of noisy actions) can be preserved whenever it is in a crowd, alone.

Figure 1. A group of Virtual Humans

Figure 2. Virtual Humans in a city

To create this flexible Virtual Humans with individualities, there are mainly two approaches:

- Recording the motion using motion capture systems (magnetic or optical), then to try to alterate such a motion to create this individuality. This process is tedious and there is no reliable method at this stage.

- Creating computational models which are controlled by a few parameters. One of the major problem is to find such models and to compose them to create complex motion. Such models can be created for walking, running, grasping, but also for interaction, groups, and crowds.

2. Motion Capture and Retargeting

The first approach consists in recording the motion (Fig. 3) using motion capture systems (magnetic or optical), then to try to alterate such a motion to create this individuality. This process is tedious and there is no reliable method at this stage. Even if it is fairly easy to correct one posture by modifying its angular parameters (with an Inverse Kinematics engine, for instance), it becomes a difficult task to perform this over the whole motion sequence while ensuring that some spatial constraints are respected over a certain time range, and that no discontinuities arise. When one tries to adapt a captured motion to a different character, the constraints are usually violated, leading to problems such as the feet going into the ground or a hand unable to reach an object that the character should grab. The problem of adaptation and adjustment is usually referred to as the Motion Retargeting Problem.

Figure 3. Motion capture

Witkin and Popovic [WP95] proposed a technique for editing motions, by modifying the motion curves through warping functions and produced some of the first interesting results. In a more recent paper [PW99], they have extended their method to handle physical elements, such as mass and gravity, and also described how to use characters with different numbers of degrees of freedom. Their algorithm is based on the reduction of the character to an abstract character which is much simpler and only contains the degrees of freedom that are useful for a particular animation. The edition and modification are then computed on this simplified character and mapped again onto the end user skeleton. Bruderlin and Williams [BW95] have described some basic facilities to change the animation, by modifying the motion parameter curves. The user can define a particular posture at time t, and the system is then responsible for smoothly blending the motion around t . They also introduced the notion of motion displacement map, which is an offset added to each motion curve. The Motion Retargeting Problem term was brought up by Michael Gleicher [G98]. He designed a space-time constraints solver, into which every constraint is added, leading to a big optimisation problem. He mainly focused on optimising his solver, to avoid enormous computation time, and achieved very good results. Bindiganavale and Badler [BB98] also addressed the motion retargeting problem, introducing new elements: using the zero-crossing of the second derivative to detect significant changes in the motion, visual attention tracking (and the way to handle the gaze direction) and applying Inverse Kinematics to enforce constraints, by defining six sub-chains (the two arms and legs, the spine and the neck). Finally, Lee and Shin [JS99] used in their system a coarse-to-fine hierarchy of B-splines to interpolate the solutions computed by their Inverse Kinematics solver. They also reduced the complexity of the IK problem by analytically handling the degrees of freedom for the four human limbs

Lim and Thalmann [LT00] have addressed an issue of solving customers' problems when applying evolutionary computation. Rather than the seemingly more impressive approach of wow-it-all-evolved- from-nothing, tinkering with existing models can be a more pragmatic approach in doing so. Using interactive evolution, they experimentally validate this point on setting parameters of a human walk model for computer animation while previous applications are mostly about evolving motion controllers of far simpler creatures from scratch.

Given a captured motion associated to its Performer Skeleton, we decompose the problem of retargeting the motion to the End User Skeleton into two steps

- First, computing the Intermediate Skeleton matrices by orienting the Intermediate Skeleton bones to reflect the Performer Skeleton posture (Motion Converter).
- Second, setting the End User Skeleton matrices to the local values of the corresponding Intermediate Skeleton matrices.

The first task is to convert the motion from one hierarchy to a completely different one. We introduce the Intermediate Skeleton model to solve this, implying three more subtasks: manually set at the beginning the correspondences between the two hierarchies, create the Intermediate Skeleton and convert the movement. We are then able to correct the resulting motion and make it enforce Cartesian constraints by using Inverse Kinematics. When considering motion conversion between different skeletons, one quickly notices that it is very difficult to directly map the Performer Skeleton values onto the End User Skeleton, due to their different proportions, hierarchies and axis systems. This raised the idea of having an Intermediate Skeleton: depending on the Performer Skeleton posture, we reorient its bones to match the same directions. We have then an easy mapping of the Intermediate Skeleton values onto the End User Skeleton. The first step is to compute the Intermediate Skeleton (Anatomic Binding module). During the animation, motion conversion takes two passes, through the Motion Converter and the Motion Composer (which has a graphical user interface).

3. Creating Computational Models

The second approach consists in creating computational models which are controlled by a few parameters. One of the major problem is to find such models and to compose them to create complex motion. Such models can be created for example for walking.

Walking has global and specific characteristics. From a global point of view, every human-walking has comparable joint angle variations. However, at a close-up, we notice that individual walk characteristics are overlaid to the global walking gait.

We use the walking engine described in [BMT90] which has been extended in the context of a european project on virtual human modeling [BCH95]. Our contribution consists in integrating the walking engine as a specialized action in the animation framework. Walking is defined as a motion where the center of gravity alternatively balances from the right to the left side. It has the following characteristics

- at any time, at least one foot is in contact with the floor, the 'single support' duration *(d*s).
- there exists a short instant during the walk cycle, where both feet are in contact with the floor, the 'double support' duration *(dd*s).
- it is a periodic motion which has to be normalized in order to adapt to different anatomies.

The joint angle variations are synthesized by a set of periodic motions which we briefly mention here:

- sinus functions with varying amplitudes and frequencies for the humanoid's global translations (vertical, lateral and frontal) and the humanoid's pelvic motions (forward/backward, left/right and torsion)
- periodic functions based on control points and interpolating hermite splines.They are applied to the hip flexion, knee flexion, ankle flexion, chest torsion, shoulder flexion and elbow flexion.

The parameters of the joint angle functions can be modified in a configuration file in order to generate personalized walking gaits, ranging from tired to energetic, sad to happy, smart to silly. The algorithm also integrates an automatic speed tuning mechanism which prevents sliding on the supporting surface. Many high level parameters can be adjusted dynamically, such as linear and angular velocity, foot step locations and the global walk trajectory. The walk engine has been augmented by a specialized action interface and its full capacity is therefore available within the animation framework. The specialized action directly exports most common high level parameter adjustment functions. For fine-tuning, it is still possible to explicitly access the underlying motion generator. The walk animation engine has been developed in the early nineties. However it suffered from not being easily combined with other motions, for example a walking human giving a phone call with a wireless phone was hardly possible. Now, that the walking engine is integrated as a specialized action, a walking and phoning human is easily done, simply by performing the walk together with a 'phone'-keyframe for example. In Figure 4, we show an example of parameterized.

Figure 4. Individualized walking

More recently, Glardon et al. [GBT04] have proposed a novel approach to generate new generic human walking patterns using motion-captured data, leading to a real-time engine intended for virtual humans animation. The method applies the PCA (Principal Component Analysis) technique on motion data acquired by an optical system to yield a reduced dimension space where not only interpolation, but also extrapolation are possible, controlled by quantitative speed parameter values. Moreover, with proper normalization and time warping methods, the generic presented engine can produce walking motions with continuously varying human height and speed with real-time reactivity. Figure 5 shows examples.

Figure 5. Examples of PCA-based walking humans

4. Crowds and Groups

Animating crowds [MT01] is challenging both in character animation and a virtual city modeling. Though different textures and colors may be used, the similarity of the virtual people would be soon detected by even non-experts, say, "everybody walks the same in this virtual city!". It is, hence, useful to have a fast and intuitive way of generating motions with different personalities depending on gender, age, emotions, etc., from an example motion, say, a genuine walking motion. The problem is basically to be able to generate variety among a finite set of motion requests and then to apply it to either an individual or a member of a crowd. It also needs very good tools to tune the motion [EBM00].

The proposed solution addresses two main issues: i) crowd structure and ii) crowd behavior. Considering crowd structure, our approach deals with a hierarchy composed of crowd, groups and agents, where the groups are the most complex structure containing the information to be distributed among the individuals. Concerning crowd behavior, our virtual agents are endowed with different levels of autonomy. They can either act according to an innate and scripted crowd behavior (programmed behavior), react as a function of triggered events (reactive or autonomous behavior) or be guided by an interactive process during simulation (guided behavior). We introduced the term <guided crowds> to define the groups of virtual agents that can be externally controlled in real time [MBC98]. Figure 6 shows a crowd guided by a leader.

Figure 6. Crowd guided by a leader

In our case, the intelligence, memory, intention and perception are focalized in the group structure. Also, each group can obtain one leader. This leader can be chosen randomly by the crowd system, defined by the user or can emerge from the sociological rules. Concerning the crowd control features, The crowd aims at providing autonomous, guided and programmed crowds. Varying degrees of autonomy can be applied depending on the complexity of the problem. Externally controlled groups, <guided groups>, no longer obey their scripted behavior, but act according to the external specification. At a lower level, the individuals have a repertoire of basic behaviors that we call innate behaviors. An innate behavior is defined as an "inborn" way to behave. Examples of individual innate behaviors are goal seeking behavior, the ability to follow scripted or guided events/reactions, the way trajectories are processed and collision avoided. While the innate behaviors are included in the model, the specification of scripted behaviors is done by means of a script language. The groups of virtual agents whom we call <programmed groups> apply the scripted behaviors and do not need user intervention during simulation. Using the script language, the user can directly specify the crowd or group behaviors. In the first case, the system automatically distributes the crowd behaviors among the existing groups. Events and reactions have been used to represent behavioral rules. This reactive character of the simulation can be programmed in the script language (scripted control) or directly given by an external controller. We call the groups of virtual agents who apply the behavioral rules <autonomous groups>.

The train station simulation (Figure 7) includes many different actions and places, where several people are present and doing different things. Possible actions include "buying a ticket", "going to shop", "meeting someone", "waiting for someone", "making a telephone call", "checking the timetable", etc. This simulation uses external control to guide some crowd behaviors in real time.

Figure 7. Train station simulation.

More recently, we developed a new crowd engine allowing to display up to 50'000 thousands virtual humans in real-time. This makes Computational models even more important. Figure 8 shows two examples.

Figure 8. Examples of large crowds.

5. Perception

Let's now consider the simulation of a referee during a tennis match. He has to decide if the ball is out or in. One solution is to calculate the intersection between the impact point of the ball and the court lines. Such an analytical

calculation will lead to the decision that the ball is out for 0.01 millimeters. Ridiculous, nobody in reality could take such an objective decision, this is not believable. The decision should be based on the evaluation of the visual aspect of the scene as perceived by the referee.

In a more general context, it is tempting to simulate perception by directly retrieving the location of each perceived object straight from the environment. This is of course the fastest solution (and has been extensively used in video-games until the mid-nineties) but no one can ever pretend that it is realistic at all (although it can be useful, as we will see later on). Consequently, various ways of simulating visual perception have been proposed, depending on whether geometric or semantic information (or both) are considered. Renault et al. introduced first the concept of synthetic vision [RMT90] then extended by Noser et al..[NRT95]. Tu and Terzopoulos [TT94] implemented a realistic simulation of artificial fishes. Other authors [KL99] [BG95] [PO02] also provided synthetic vision approaches. In the next section, we are going to compare now rendering-based vision, geometric vision and database access.

5.1 Synthetic Vision

Rendering-based vision from Noser and Renault et al. [NRT95] is achieved by rendering of-screen the scene as viewed by the agent. During the process, each individual object in the scene is assigned a different colour, so that once the 2D image has been computed, objects can still be identified: it is then easy to know which object is in sight by maintaining a table of correspondences between colours and objects' IDs. Furthermore, highly detailed depth information is retrieved from the view z-buffer, giving a precise location for each object. An other application of synthetic vision is real-time collision avoidance for multiple agents: in this case, each agent is perceiving the others, and dynamically creates local goals so that it avoids others while trying to reach its original global goal.

Rendering-based vision is the most elegant method, because it is the more realistic simulation of vision and addresses correctly vision issues such as occlusion for instance. However, rendering the whole scene for each agent is very costly and for real-time applications, one tend to favour geometric vision.

One problem is how to decide that an object is in the field of view of the Virtual Human and that he/she can identify it. We can imagine for example that the Virtual Human's wife is in front of the VH but hidden by a wardrobe and on the computed 2D image contains only one pixel for the wife, can he recognize his wife based on such a detail ?

Bordeux et al. [BBT99] has proposed a perception pipeline architecture into which filters can be combined to extract the required information. The perception filter represents the basic entity of the perception mechanism. Such a filter receives a perceptible entity from the scene as input, extracts specific information about it, and finally decides to let it pass through or not.

The criteria used in the decision process depends on the perception requirements. For virtual objects, they usually involve considerations about the distance and the relative direction of the object, but can also be based on shape, size, colour, or generic semantic aspects, and more generally on whatever the agent might need to distinguish objects. Filters are built with an object oriented approach: the very basic filter for virtual objects only considers the distance to the object, and its descendants refine further the selection.

Actually, the structure transmitted to a filter contains, along with the object to perceive, a reference to the agent itself and previously computed data about the object. The filter can extend the structure with the results of its own computation, for example the relative position and speed of the object, a probable time to impact or the angular extension of the object from the agent s point of view. Since a perception filter does not store data concerning the objects that passed through it, it is fully reentrant and can be used by several agents at the same time. This allows the creation of a common pool of filters at the application, each agent then referencing the filters it needs, thus avoiding useless duplication.

However, the major problem with Geometric vision is to find the proper formulas when intersecting volumes (for instance, intersecting the view frustum of the agent with a volume in the scene). One can use bounding boxes to reduce the computation time, but it will always be less accurate than Synthetic vision. Nevertheless, it can be sufficient for many applications and, as opposed to rendering-based vision, the computation time can be adjusted precisely by refining the bounding volumes of objects.

Database access makes maximum use of the scene data available in the application, which can be distributed in several modules. For instance, the objects position, dimensions and shape are maintained by the rendering engine whereas semantic data about objects can be maintained by a completely separate part of the application. Due to scalability constraints as well as plausibility considerations, the agents generally restrain their perception to a local area around them instead of the whole scene. This method is generally chosen when the number of agents is high. In Musse's [MT01] crowd simulation, human agents directly know the position of their neighbours and compute coherent collision avoidance trajectory. As said before, the main problem with the method is the lack of realism, which can only be alleviated by using one of the other methods.

These various approaches to visual perception have their advantages and disadvantages dependent essentially of the complexity and the context of the scenes. But, finally no approach can solve common problematics as the following one: What makes a little girl to be lost in a crowd ? The child will be lost if she just does not know where is her family. Now imagine a virtual crowd where each individual is indexed. It will be extremely easy fo find where is the girl (index 345) and the parents (index 748). At this stage, we could just activate a function making the girl walking towards his parents. This is completely unrealistic from a behavioural point of view.

5.2 Memory

Noser et al. [NRT95] made a few years ago a character trying to find the exit from a maze. To simulate the memory process, they used an octree structure to store the information see by the character. The results were that the second time, it was straightforward for the character to find the exit. Again, this is not so convincing as never somebody could remember all the paths inside a maze. This kind of memory can then easily be linked to the synthetic vision: the 2D rendering and the corresponding z-buffer data are combined in order to determine whether the corresponding voxel of the scene is occupied by an object or not. By navigating through the environment, the agent will progressively construct a voxel-based representation of it. Of course, a rough implementation of this method would suffer from dramatic memory cost, because of the high volume required to store all voxels. Noser proposed to use octrees instead which successfully reduces the amount of data. Once enough information has been gathered through exploration, the agent is then able to locate things and find its way.

Peters and O'Sullivan [PO02] propose a system of memory based on what is referred to a "stage theory" by Atkinson and Shiffrin [AS68]. They propose a model where information is processed and stored in 3 stages: sensory memory, short-term memory, and long-term memory.

Although these approaches are quite interesting, they do not solve the following simple problematics. Imagine now a Virtual Human inside a room containing 100 different objects. Which objects can we consider as memorized by the Virtual Human ? Can we decide that when an object is seen by the actor, it should be stored in his memory. To answer this question, we have just to consider the popular family game consisting in showing 20 objects during 2 minutes to people and asking them to list the objects. Generally nobody is able to list the 20 objects. Now, how to model this inability to remember all objects ?

5.3 Integration of Virtual Sensors

The modelling of an AVA gaining its independence with regard to its virtual representation remains an important theme in research and is very close to autonomous robotics. It helps also to understand and model human behaviour.

The AVA collects information only through the virtual sensors described earlier (Figure 9). We assume that vision is the main canal of information between the AVA and its environment as indicated by the standard theory in neuroscience for multi-sensorial integration [E98].

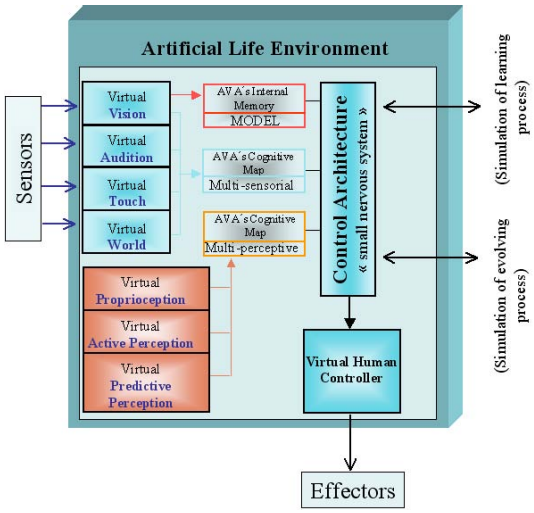


Figure 9: A schematic representation of our *ALifeE*. Virtual Vision discovers the VE, constructs the different types of Perception and updates the AVA's Cognitive Map to obtain a multi-perceptive mapping. Then the Control Architecture uses both the "cognitive maps" and the "memory model" to interact with the learning, development, and control processes of the AVA (Virtual Human Controller).

The sensorial modalities update the AVA's cognitive map to obtain a multi-sensorial mapping. For example, visual memory in the AVA's internal memory is used for a global move from point A to point B. Should obstacles be present, it would have to be replaced for a local move by direct vision of the environment.

In our approach, we tried to integrate all the multi-sensorial information from the AVA's virtual sensors. In fact, an AVA in a VE may have different degrees of autonomy and different sensorial canals depending on the environment. For instance, an AVA moving in a VE represented by a well-lit room will use primarily the sensorial information of vision. However if the light is turned off, the AVA will appeal to the acoustic or tactile sensorial information in the same way a human would move around in a dark room [SKA02].

From this observation we derive the hypotheses underlying our ALifeE framework approach. They are backed up by the latest research in neuroscience [P02], which describes a partial re-mapping at the behavioural level of the human including:

- *Assignment*: the prediction of the acoustic position of an object from its visual positions requires a transformation from its *eye-centred* (vision sensor) coordinates to its *head-centred* ones (auditory sensor). The comparison of these two types of results can be used to determine whether the acoustic and visual signals are directly connected to the same object.
- *Recoding*: the choice of the reference frame to integrate the sensorial signals.

6. Conclusion

In order to develop truly interactive multimedia systems with Virtual Humans, games, and interactive movies, we need a flexible way of animating these Virtual Humans. Altering motion obtained from a motion capture system is not the best solution. Only computational models can offer this flexibility unless powerful motion retargeting methods are developed, but in this case they will look similar to computational models.

Acknowledgments

The author would like to thank all people who have contributed to these projects especially Luc Emering, Soraia Musse, Ik Soo Lim, Pascal Glardon, Mireille Clavien, Toni Conde, and Pablo de Heras. Research has been partly funded by the Swiss National Foundation for Research and the Federal Office for Education and Science in the framework of the CROSSES project.

References

[AS68] ATKINSON R., SHIFFRIN R., Human Memory : a Proposed System and its Control Processes, in: *K.Spence and J.Spence, the Psychology of Learning and Motivation: Advances in Research and Theory*, Vol.2, NY, Academic Press, 1968.

[BB98] BINDIGANAVALE R., BADLER N.I.. Motion abstraction and mapping with spatial constraints. In N. Magnenat-Thalmann and D. Thalmann, editors, Modeling and Motion Capture Techniques for Virtual Environments, *Lecture Notes in Artificial Intelligence*, pages 70–82. Springer, November 1998. held in Geneva, Switzerland, November 1998.

[BBT99] BORDEUX C., BOULIC R., THALMANN D., An Efficient and Flexible Perception Pipeline for Autonomous Agents, *Proc. Eurographics '99*, Milano, Italy, pp.23-30.

[BCH95] BOULIC R., CAPIN T., HUANG Z., MOCCOZET L., MOLET T., KALRA P., LINTERMANN B., MAGNENAT-THALMANN N., PANDZIC I., SAAR K., SCHMITT A., SHEN J., THALMANN D., The HUMANOID Environment for Interactive Animation of Multiple Deformable Human Characters, *Proc. Eurographics `95*, Maastricht, August 1995, pp.337-348.

[BG95] BLUMBERG B.M., GALYEAN T.A,, Multi-Level Direction of Autonomous Creatures for Real-Time Virtual Environments, *Proc. SIGGRAPH 95*, 1995, pp.47-54.

[BMT90] BOULIC R., MAGNENAT-THALMANN N., THALMANN D., A Global Human Walking Model with Real-time Kinematics Personification,*The Visual Computer*, Vol.6, No6, December 1990, pp.344-358.

[BW95] BRUDERLIN A., WILLIAMS L. Motion sig-nal processing. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Se-ries, pages 97–104. ACM SIGGRAPH, Addison Wes-ley, August 1995. held in Los Angeles, California, 06- 11 August 1995.

[E98] Elfes G. Occupancy Grid: A Stochastic Spatial Representation for Active Robot Perception. In *6th Conference on Uncertainly in AI*, 1990

[EBM00] L.EMERING, R.BOULIC, T.MOLET, D.THALMANN, Versatile Tuning ofHumanoid Agent Activity, *Computer Graphics Forum*, 2000

[G98] GLEICHER G. Retargeting motion to new characters. In Michael Cohen, editor, *SIGGRAPH 98 Con-ference Proceedings*, Annual Conference Series, pages 33–42. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.

[GBT04] GLARDON P., R. BOULIC R., THALMANN D., PCA-based Walking Engine using Motion Capture Data, *Computer Graphics International, June 2004*, pp.292-298.

[JS99] JEHEE L., SHIN S.Y.. A hierarchical approach *Proceedings of SIGGRAPH 99*, pages 39–48, Au-gust 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.

[KL99] KUFFNER J., LATOMBE J.C., Fast Synthetic Vision, Memory, and Learning Models for Virtual Humans, *Proc. Computer Animation 1999*, IEEE CS Press, pp.118-127.

[LT00] LIM I.S., THALMANN D., Solve Customers' Problems: Interactive Evolution for Tinkering with Computer Animation, *Proc. 2000 ACM Symposium on Applied Computing (SAC2000)*, pp. 404-407

[MBC98 MUSSE, S.R., BABSKI, C., CAPIN, T. AND THALMANN, D. Crowd, Modelling in Collaborative Virtual Environments. *ACM VRST '98*, Taiwan

[MT01] MUSSE S.R., THALMANN D., A Behavioral Model for Real-Time Simulation of Virtual Human Crowds, *IEEE Transactions on Visualization and Computer Graphics*, Vol.7, No2, 2001, pp.152-164.

[NRT95] NOSER H., O. RENAULT O., D. THALMANN, N. MAGNENAT THALMANN, Navigation for Digital Actors based on Synthetic Vision, Memory and Learning, *Computers and Graphics*, Pergamon Press, Vol.19, No1, 1995, pp.7-19.

[P02] POUGET A., A computational perspective on the neural basis of multi-sensory spatial representations. *Nature Reviews/Neuroscience*, 2002; 3:741-747.

[PO02] PETERS C., O'SULLIVAN C., A Memory Model for Autonomous Virtual Humans, *Proc. Third Irish Eurographics Workshop on Computer Graphics*, Dublin, pp. 21-26.

[PW99] POPOVIC Z., WITKIN A.. Physically based motion transformation. *Proceedings of SIGGRAPH 99*, pages 11–20, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.

[RMT90 RENAULT O., MAGNENAT-THALMANN N., THALMANN D., A Vision-based Approach to Behavioural Animation, *Journal of Visualization and Computer Animation*, Vol.1, No1, 1990, pp.18-21.

[SKA02] Strösslin Th, Krebser Ch, Arleo A, Gerstner W. Combining Multimodal Sensory Input for Spatial Learning. In Proceedings of ICANN, 2002; 87-92. *LNCS 2415*.Springer-Verlag.

[TT94] TU X., TERZOPOULOS D., Artificial Fishes, Physics, Locomotion, Perception, Behaviour, *Proc. SIGGRAPH '94*, pp.43-50.

[WP95] WITKIN A., POPOVIC Z.. Motion warping. *Proceedings of SIGGRAPH 95*, pages 105–108, August 1995. ISBN 0-201-84776-0. Held in Los Angeles, California.

Real-Time Crowds: Architecture, Variety, and Motion Planning

Jonathan Maïm[1] Barbara Yersin[1] Daniel Thalmann[1]

[1] Virtual Reality Laboratory, EPFL, Switzerland

Abstract

To simulate realistic virtual crowds in real time, three main requirements need satisfaction. First of all, quantity, i.e., the ability to simulate thousands of characters in real time. Secondly, quality, because each virtual human composing a crowd needs to look unique in its appearance and animation. Finally, realism in terms of crowd motion and navigation. In this tutorial, we explain how all objectives can be reached together. We first detail an efficient and versatile architecture able to simulate thousands of characters in real time. Then, state-of-the-art techniques to transform similar instances of a crowd into unique individuals are introduced. Finally, a hybrid motion planning approach, able to manage navigation and obstacle avoidance in real time, is presented. Overall, we show that it is possible to combine these three aspects to simulate large, realistic, and visually appealing crowds in real time.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - Animation

1. Introduction

In these turorial notes, we focus on technical aspects for creating an architecture sustaining real-time crowd simulation composed of several thousand of varied individuals planning for their path and avoiding collision. We begin the tutorial with a description of the different virtual human representations commonly used in crowd simulation in Section 2 and how each of them can cast shadows. Then, in Section 3, we introduce our crowd architecture, and the pipeline developed to process crowds in real time. In Section 4, we detail several techniques that can be efficiently used to vary the appearance of similar characters, instantiated from a same human template. Finally, we also present in Section 5 a hybrid and scalable motion planning architecture able to manage thousands of characters in complex environments in real time. Our conclusion is presented in Section 6.

2. Virtual Human Representations

In an ideal world, graphic cards would be able, at each frame, to render an infinite number of triangles with an arbitrary complex shading on them. To visualize crowds of virtual humans, we would simply use thousands of very detailed meshes, *e.g.*, capable of hand and facial animation. Unfortunately, in spite of the recent programmable graphics hardware advances, we are still compelled to stick to a limited triangle budget per frame. This budget is spent wisely to be able to display dense crowds without too much perceptible degradations. The concept of levels of detail (LOD), extensively treated in the literature (see Luebke *et al.* [LWC*02]) is exploited to meet our real-time constraints. For a crowd of virtual humans specifically, and depending on the location of the camera, a character is rendered with a particular representation, resulting from the compromise of rendering cost and quality. In this Section, we first introduce the data structure we use to create and simulate virtual humans: the human template. Then, we describe the three levels of detail a human template uses: the deformable mesh, the rigid mesh, and finally the impostor.

2.1. Human Template

A type of human such as a woman, man, or child is described as a human template, which consists of :

- A skeleton, composed of joints, representing articulations,

- A set of meshes, all respresenting the same virtual human, but with a decreasing number of triangles,
- Several appearance sets, used to vary its appearance,
- A set of animation sequences which it can play.

Each rendered virtual human is derived from a human template, *i.e.*, it is an instance of a human template. In order for all the instances of a same human template to look different, we use several appearance sets, that allow to vary the texture applied to the instances, as well as modulate the colors of the texture (see Section 4).

2.2. Deformable Mesh

A deformable mesh is a representation of a human template composed of triangles. It is enveloping a skeleton of 78 joints, used for animation: when the skeleton moves, the vertices of the mesh follow smoothly its joint movements, similarly to our skin. We call such an animation a skeletal animation. Each vertex of the mesh is influenced by one or a few joints. Thus, at every keyframe of an animation sequence, a vertex is deformed by the weighted transformation of the joints influencing it. The corresponding equation is:

$$v(t) = \sum_{i=1}^{n} \chi_i^t \chi_i^{-ref} v^{ref} \quad (1)$$

where $v(t)$ is the deformed vertex at time t, χ_i^t is the global transform of joint i at time t, χ_i^{-ref} is the inverse global transform of the joint in the reference position, and v^{ref} is the vertex in its reference position. This technique is known as skeletal subspace deformation, or skinning.

The skinning can be efficiently performed by the GPU: the deformable mesh sends the joint transformations of its skeleton to the GPU, that takes care of moving each vertex according to its joint influences. However, it is important to take into account the limitations of graphic cards (Shader Model 2 & 3 [nvi06]), that can store only up to 256 atomic values, *i.e.*, 256 vectors of four floating points. The joint transformations of a skeleton can be sent to the GPU as 4x4 matrices, *i.e.*, four atomic values. This way, the maximum number of joints a skeleton can have reaches:

$$\frac{256}{4} = 64 \quad (2)$$

When wishing to perform hand and facial animation, 64 joints are not sufficient. Our solution is to send each joint transformation to the GPU as a unit quaternion and a translation, *i.e.*, two atomic values. This allows to double the number of joints possible to send. Note that one usually does not wish to use all the atomic structures of a GPU exclusively for the joints of a skeleton, since it is usually exploited to process other data.

Rendering deformable meshes is very costly, due primarily to a pipeline flush occuring each time a new virtual human is rendered, and also to the expensive vertex skinning and joint transmission. Nevertheless, it would be a great quality drop to do without them, indeed :

- They are the most flexible representation to animate, allowing even for facial and hand animation (if using a sufficiently detailed skeleton),
- Such animation sequences, called skeletal animations, are cheap to store: for each keyframe, only the transformation of deforming joints, *i.e.*, those moved in the animation, need to be kept. Thus, a tremendous quantity of those animations can be exploited in the simulation, increasing crowd movement variety,
- Procedural and composited animations are suited for this representation, *e.g.*, idle motions can be generated on-the-fly (see for example Egges *et al.* [EGMT06]),
- Blending is also possible for smooth transitions between different skeletal animations.

Unfortunately, the cost of using deformable meshes as the sole representation of virtual humans in a crowd is too prohibitive. We therefore use them in a limited number and only at the fore-front of the camera. Note that before switching to rigid meshes, we use several deformable meshes, keeping the same animation algorithm, but with a mesh of a decreasing number of triangles.

Skinned and textured deformable meshes require skilled designers. But once finished, they are automatically used as the raw material to derive all subsequent representations: the rigid meshes and the impostors.

2.3. Rigid Meshes

A rigid mesh is a precomputed geometric posture of a deformable mesh, thus sharing the very same appearance. A rigid animation sequence is always inspired from an original skeletal animation, and from an external point of view, both look alike. However, the process to create them is different. To compute a keyframe of a rigid animation, the corresponding keyframe for the skeletal animation is retrieved. It provides a skeleton posture (or joint transformations). Then, as a preprocess, each vertex is deformed on the CPU, in opposition to a skeletal animation, where the vertex deformation is achieved online, and on the GPU. Once the rigid mesh is deformed, it is stored as a keyframe, in a table of vertices, normals (3D points), and texture coordinates (2D points). This process is repeated for each keyframe of a rigid animation. At runtime, a rigid animation is simply played as the succession of several postures or keyframes. There are several advantages in using such a rigid mesh representation:

- It is much faster to display, because the skeleton deformation and vertex skinning stages are already done and stored in keyframes. The communication between the CPU and the GPU is kept to a minimum, since no joint transformation needs to be sent, and pipeline flushing is significantly reduced.

- It looks exactly the same as the skeletal animation used to generate it.

The gain in speed brought by this new representation is considerable. It is possible to display about 10 times more rigid meshes than deformable meshes (see Section 3.5 for detailed results). However, the rigid meshes need to be displayed farther from the camera than deformable meshes, because they allow for neither procedural animations, nor blending, and no composited, facial, or hand animation is possible.

2.4. Impostor

An impostor is the less detailed representation, and extensively exploited in the domain of crowd rendering [TLC02, DHOO05, MR06]. An impostor represents a virtual human with only two textured triangles, forming a quad, which is enough to give the wanted illusion at long range from the camera. Similarly to a rigid animation, an impostor animation is a succession of postures, or keyframes, inspired from an original skeletal animation. The main difference with a rigid animation is that it is only a 2D image of the posture that is kept for each keyframe, instead of the whole geometry. Creating an impostor animation is complex and time consuming. Thus, its construction is achieved in a preprocess, and the result is then stored into a database in a binary format (see Section 3.4), similarly to a rigid animation. We detail here how each keyframe of an impostor animation is developed. The first step when generating such a keyframe for a human template is to create two textures, or atlas:

- A normal map, storing in its texels the 3D normals as RGB components. This normal map is necessary to apply the correct shading to the virtual humans rendered as impostors. Indeed, if the normals were not saved, a terrible shading would be applied to the virtual human, since it is represented with only two triangles. Switching from a rigid mesh to an impostor would thus lead to awful popping artefacts.
- A UV map, storing in its texels the 2D texture coordinates as RG components. This information is also very important, because it allows to apply correctly a texture to each texel of an impostor. Otherwise, we would need to generate an atlas for every texture of a human template.

Since impostors are only 2D quads, we need to store normals and texture coordinates from several points of view, so that, at runtime, when the camera moves, we can display the correct keyframe from the correct camera view point. In summary, each texture described above holds a single mesh posture for several points of view. This is why we also call such textures atlas. We illustrate in Figure 1 a 1024x1024 atlas for a particular keyframe. The top of the atlas is used to store the UV map, and its bottom the normal map. The main advantage of impostors is that they are very efficient, since only two triangles per virtual human are displayed. Thus, they constitute the biggest part of the crowd. However, their

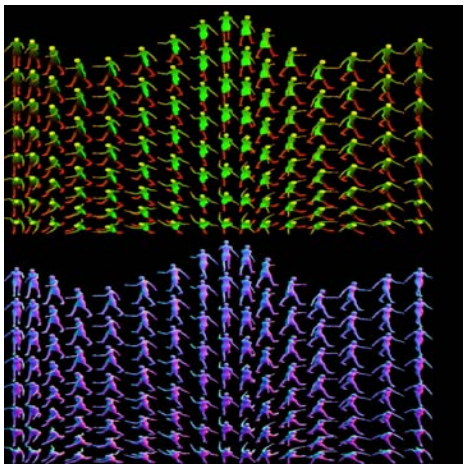
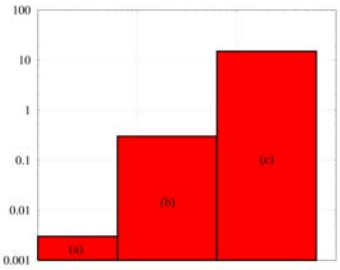


Figure 1: *A 1024x1024 atlas storing the UV map (above) and the normal map (below) of a virtual human performing a keyframe of an animation from several points of view.*

rendering quality is poor, and thus they cannot be exploited close to the camera. Moreover, the storage of an impostor animation is very costly, due to the high number of textures that need to be saved. We summarize in Table 2 the perfor-



	Max Displayable Number @ 30 Hz	Animation Frequency [Hz]	Animation Sequence Storage [Mb/s]	Memory Location
(a)	~200	25	~0.03	CPU
(b)	~2'000	20	~0.3	GPU
(c)	~20'000	10	~15	GPU

Figure 2: *Storage space in [Mb] for one second of an animation clip of (a) a deformable mesh, (b) a rigid mesh, and (c) an impostor.*

mance and animation storage for each virtual human representation. We observe that each step down the representation hierarchy allows to increase by an order of magnitude the number of displayable characters. We also note that the faster the display of a representation the bigger the animation storage. Finally, rigid meshes and impostors are stored in GPU memory, which is usually much smaller than CPU memory. Figure 3 summarizes the shared resources inside a human template.

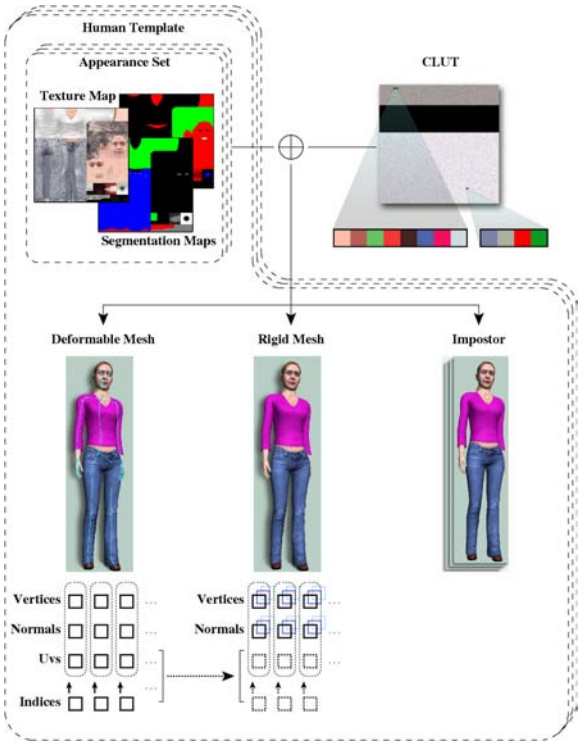


Figure 3: *shared resources between representations inside a human template.*

2.5. Shadows

In our architecture, illumination ambiances are set from four directional lights, whose direction and diffuse and ambient colors are prealably (or interactively) defined by the designer. The light coming from the sun is the only one provoking shadows. As we lack a real-time global illumination system, the three other lights are present to provide enough freedom for the designer to give a realistic look to the scene. This configuration has given us satisfaction as we mainly work on outdoor scenes. See Figure 6 (left) and 7 for results.

Virtual humans cast shadows on the environment and, reciprocally, the environment casts shadows on them. This is achieved using a shadow mapping algorithm [Wil78] implemented on the GPU. At each frame, virtual humans are rendered twice:

- The first pass is from the directional light view perspective, *i.e.*, the sun. The resulting z-buffer values are stored in the shadow map.
- The second pass is from the camera view perspective. Each pixel is transformed into light perspective space and its z value is compared with the one stored in the shadow map. Thus, it is possible to know if the current pixel is in shadow or not.

So, we need to render twice the number of virtual humans really present. Though with modern graphics hardware, rendering to a z-only framebuffer is twice as fast as rendering to a complete framebuffer, one expects a certain drop in the frame rate. Moreover, standard shadow mapping suffers from important aliasing artefacts located at shadow borders. Indeed, the resolution of the shadow map is finite, and the bigger the scene, the more aliasing artefacts appear. To alleviate this limitation, several strategies are used:

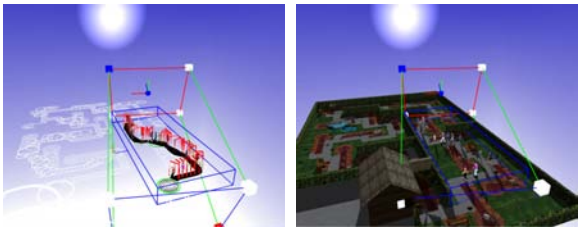


Figure 4: *Shadowed scene with apparent directional light frustum.*

- Dynamically constrain the shadow map resolution to visible characters, and
- Combine percentage closer filtering [RSC87] with stochastic sampling [Coo86], to obtain fake soft shadows [Ura05].

We now further describe how to dynamically constrain the shadow map resolution to visible characters. A directional light, as its name indicates, is defined only by a direction. Rendering from a directional light implies using an orthographic projection, *i.e.*, its frustum is a box, as depicted in Figure 4. An axis-aligned bounding box (AABB) is a box whose faces have normals that coincide with the basis axes [MH99]. They are very compact to store; only its two extreme points are necessary to determine the whole box. AABB are often used as bounding volumes, *e.g.*, in a first pass of a collision detection algorithm, to efficiently eliminate simple cases.

A directional light necessarily has an orthographic frustum aligned along its own axes. So, we can consider this frustum as an AABB. The idea is, at each frame, to compute the box englobing all the visible virtual humans, so that it is as tight as possible. Indeed, using an AABB as small as possible allows to have a less stretched shadow map. At each frame, we compute this AABB in a four-step algorithm:

1. The crowd AABB is computed in world coordinates, using visible navigation graph vertices. By default, the AABB height is set to two meters, in order to bound the characters at their full height.
2. The light space axes are defined, based on the light normalized direction L_z:
   L_x = normalize($(0,1,0)^T \hat{} L_z$).
   L_y = normalize($L_z \hat{} L_x$).

3. The directional light coordinate system is defined as the 3x3 matrix $M_l = [L_x, L_y, L_z]$.
4. The eight points composing the AABB (in world coordinates) are multiplied by M_l^{-1}, *i.e.*, the transpose of Ml. This operation expresses these points in our light coordinate system.

Note that remultiplying the obtained points by M_l would express the crowd AABB back into world coordinates. In Figure 4 are illustrated the shadows obtained with this algorithm. Practically, to be able to choose an adequate resolution given the situation, *e.g.*, detailed shadows for characters close to the camera, we use three different shadow maps: one for the shadows cast by the environment, one for the people (deformable and rigid meshes) near the camera, and one for people far from it (impostors).

3. Architecture

The main problem when dealing with thousands of characters is the quantity of information that needs to be processed for each one of them. Such a task is very demanding, even for modern processors. Simple approaches, where virtual humans are processed one after another, in no specific order, provokes costly state switches for both the CPU and GPU. For an efficient use of the available computing power, and to approach hardware peak performance, data flowing through the same path need to be grouped. In this Section, we present an architecture able to handle, early in its pipeline, the sorting of virtual human related data into grouped slots, allowing the simulation of thousands of characters. Moreover, it is versatile enough to be stressed in very different scenarii, *e.g.*, in confined environments like an auditorium or a classroom, as well as in large-scale environments like a crowded fun fair or city.

The Section is divided as follows: first, in Section 3.1,we briefly introduce the human data structure our architecture employs. Then, we delve into each of the pipeline stages in Section 3.2. In Section 3.3, motion kits, a data structure specifically developed for managing the different levels of detail at the animation stage are described. Concerning efficiency of storage and data management, we mainly employ a database to store all the virtual human related data, as detailed in Section 3.4. Finally, in Section 3.5, we show the overall performance of our architecture.

3.1. Human Data Structures

Virtual human instances are shared in several data structures, and a unique identifier is associated to each one of them. Our crowd data structure is mainly composed of two arrays. An array of body entities, and an array of brain entities. The unique identifier of each virtual human is used to index these arrays and retrieve specific data, which is distributed in a body and brain entity. Body data consists in all the parameters used at every frame, like the position and orientation of the virtual human. Brain data is more related to behavior parameters, and is less regularly exploited. By separating these parameters from the body entity, we tighten the storage of very often used data. Indeed, such a regrouping improves performance: in a recent work [PdHCM*06], while experimenting different steering methods, we observed that with a varying number of characters in a very large scale (tens of thousands), the performance of the different methods remained about the same. Memory latency to jump from an instance to the other was the bottleneck when dealing with big crowds.

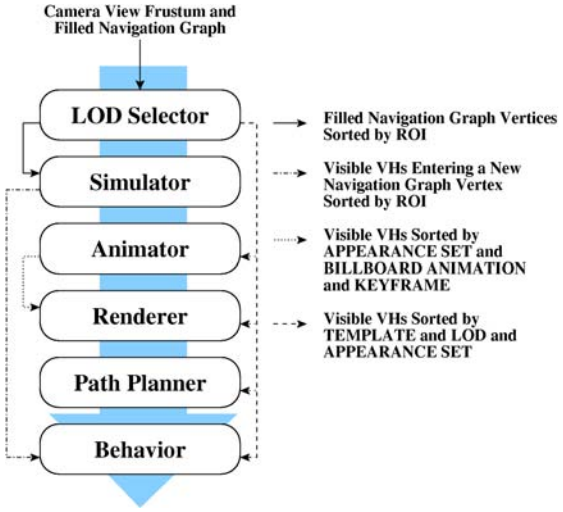


Figure 5: *Crowd architecture pipeline.*

3.2. Pipeline Stages

In this Section, we first provide a short reminder on the navigation graph structure, which is used for crowd motion planning (see Section 5), but also provides a very convenient structure to process the virtual humans hierarchically instead of individually. Then, we detail the stages of the pipeline illustrated in Figure 5.

For a given scene, a navigation graph is provided and used to steer virtual humans along predefined paths. The graph is composed of a set of vertices, represented in the scene as vertical cylinders where no collision with the environment can occur. Two vertices can be connected by an edge, represented as a gate between two overlapping cylinders (see Figure 8). When several cylinders overlap, their consecutive gates delimit a corridor. In a scene, a path to follow is defined as a sequence of gates to reach one after the other, *i.e.*, simple sub-goals for the chosen steering method (See Section 5 for more details). During simulation, each vertex keeps a list of the ids of virtual human currently travelling

Figure 6: *(left) Virtual humans navigating in a complex environment. (right) Similar image with apparent levels of detail; in red: the rigid meshes, in green: the impostors.*

Figure 7: *Dense crowd in a large environment.*

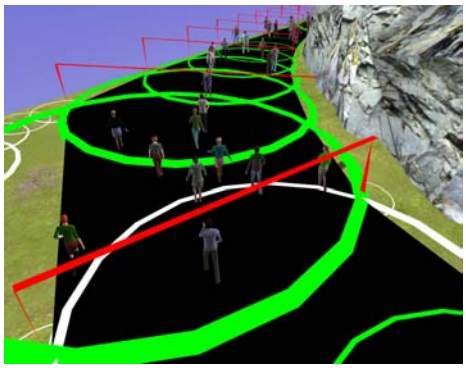


Figure 8: *Virtual humans steering along a path sustained by a navigation graph structure (in green and white). Overlapping vertices form gates (in red). Consecutive gates on the path form corridors (in black).*

through it. Pettre *et al.* [PdHCM*06, PGT07] fully detail the necessary steps to construct navigation graph from an arbitrary 3D scene. Here follows the detailed description of each pipeline stage.

The **LOD Selector** is the first stage of the pipeline. It receives as input a navigation graph filled with virtual human ids and the camera view frustum. The role of the LOD Selector entity is to categorize graph vertices, *i.e.*, to score each one of them for further processing. We have two different scores to attribute to each vertex. Firstly, a level of detail (LOD), determined by finding the distance from the vertex to the camera and its eccentricity from the middle of the screen. This LOD score is then used to choose the appropriate virtual human representation inside the vertex. Secondly, the LOD Selector associates with each vertex a score of interest, resulting in an environment divided into regions of different interest (ROI). For each region, we choose a different motion planning algorithm. Regions of high interest use accurate, but more costly techniques, while regions of lower interest may exploit simpler methods (See Section 5 for more details).

The LOD Selector uses the navigation graph as a hierarchical structure to avoid testing individually each character. The processing of data is achieved as follows: firstly, each vertex of the graph is tested against the camera view frustum, *i.e.*, frustum culled. Empty vertices are not even scored, nor further held in the process for the current frame; indeed, there is no interest to keep them in the subsequent stages of the pipeline. On the other hand, vertices filled with at least one character and outside the camera view are kept, but they are not assigned any LOD score, since they are outside the view frustum, and thus, their virtual humans are not displayed. As for their ROI score, they get the lowest one: no dynamic collision avoidance between pedestrians need be achieved. However, even if they are not in the camera field, virtual humans contained in these vertices need a minimal simulation to sporadically move along their path. Without care, when they quit the camera field, they immediately stop moving, and thus, when the camera changes its point of view, packages of stagnant characters suddenly move again, causing a disturbing effect for the user. Finally, the vertices that are filled and visible are assigned a higher ROI score, and then are further investigated to sort their embedded virtual humans by human template, LOD, and appearance set.

At the end of this first stage, we obtain two lists. The first one contains all virtual human ids, sorted by human template, by LOD, and finally by appearance set. The second list contains occupied vertices, sorted by ROI. Obtaining such lists takes some time. However, it is very useful to group data and process through the next stages of the pipeline. We illustrate in the following pseudo-code how the first list is typically used in the next stages of the pipeline:

```
For each human template:
   apply human template common data
   operations, e.g., get its skeleton,
   For each LOD:
      apply LOD common data operations,
      e.g., enable LOD specific shader program,
      For each appearance set:
         apply appearance set common data
         operations, e.g., bind textures,
         For each virtual human id:
            get body or brain structure from the id,
            apply specific operations on it.
```

The second stage is the **Simulator**, which uses the second

list to iterate through all ROI slots and obtain the corresponding filled vertices. At this stage, virtual humans are considered as individual 3D points, and depending on the ROI, the proper motion planning method is applied. Please, refer to Section 5 for more details on the techniques used for each ROI.

The **Animator** is responsible for the animation of the characters, whichever the representation they are using. The slots of visible virtual humans, sorted by human template, LOD, and appearance set in the LOD Selection phase, are the main data structure used in this stage. Below is described the specific tasks that are achieved for the deformable meshes:

```
For each human template:
   get its skeleton,
   For each deformable mesh LOD:
      For each appearance set:
         For each virtual human id:
            get the corresponding body,
            update the animation time (between 0.0 and 1.0),
            perform general skeletal animation,
            perform facial skeletal animation,
            perform hand skeletal animation.
```

Since the virtual humans are also sorted by LOD, we can iterate over the deformable meshes without having to check that they actually are deformable. Performing a skeletal animation, whether it is for the face, the hands or all the joints of a virtual human, can be summarized in four steps. First, the correct keyframe, depending on the animation time, is retrieved. Note that at this step, it is possible to perform a blending operation between two animations. The final keyframe used is then the interpolation of the ones retrieved from each animation. The second step is to duplicate the original skeleton relative joint matrices in a cache. Then, in the cache, the matrices of the joints modified by the keyframe are overwritten. Finally, all the relative matrices (including those not overwritten) are multiplied to obtain global matrices, and each of them is post-multiplied by the inversed global matrices of the skeleton. Note that optional animations, like facial animation, are usually performed only for the best deformable mesh LOD, *i.e.*, the most detailed mesh, at the fore-front.

For the rigid meshes, the role of the Animator is much reduced, since all the deformations are pre-computed (see Section 2):

```
For each human template:
   For each rigid mesh LOD:
      For each appearance set:
         For each virtual human id:
            get the corresponding body,
            update the animation time (between 0.0 and 1.0).
```

Note that we do not iterate over all LOD slots, since we are only concerned with the rigid meshes. Once again, the sorting achieved in the LOD Selection stage ensures that we are exclusively iterating over rigid meshes, without costly tests.

Finally, for the impostors, since a keyframe of an impostor animation is only represented by two texture atlases, no specific deformation needs to be achieved. However, we assign the animator a special job: to update a new list of virtual human ids, specifically sorted to allow a fast rendering of impostors. Indeed, at initialization, and for each human template, a special list of virtual human ids is created, sorted by appearance set, impostor animation, and keyframe. The first task achieved by the Animator is to reset the impostor specific list in order to refill it accordingly to the current state of the simulation. Then, to refill this list, an iteration is performed over the current up-to-date list, the one sorted by human template, LOD, and appearance set (updated in the LOD Selection stage):

```
For each human template:
   get its impostor animations,
   For the only impostor LOD:
      For each appearance set AS:
         For each virtual human id:
            get the corresponding body,
            update the animation time (between 0.0 and 1.0),
            get body's current impostor animation id a,
            get body's current impostor keyframe id k,
            put virtual human id in special list[AS][a][k].
```

This way, the impostor specific list is updated every time the data passes through the Animator stage, and is thus ready to be exploited at the next stage, the Renderer.

The **Renderer** represents the phase where draw calls are issued to the GPU to display the crowd. As detailed in Section 2.5, rendering shadows is a two-pass algorithm, and achieved in this stage: first, deformable and rigid meshes, and impostors are sequentially rendered from the point of view of the sun, *i.e.*, the main directional light. Then, they are consecutively rendered from the point of view of the camera. To diminish state change overhead, the number of draw calls are minimized, thanks to our slots of visible humans sorted by human template, LOD and appearance set. In the following pseudo-code, we show the second pass in the deformable mesh rendering process:

```
For each human template:
   For each deformable mesh LOD:
      bind vertex, normal, index, and texture buffer,
      send to the GPU the joint ids influencing each vertex,
      send to the GPU their corresponding weights,
      For each appearance set:
         send to the GPU  texture specular parameters,
         bind texture and segmentation maps,
         For each virtual human id:
            get the corresponding body,
            send the joint orientations from cache,
            send the joint translations from cache.
```

This second pass is preceded by another pass, used to compute the shadows. Note that in this first pass, the process is quite similar, although data useless for shadow computation is not sent, *e.g.*, normal and texture parameters. In this rendering phase, one can see the full power of the sorted lists: all the instances of a same deformable mesh have the same vertices, normals and texture coordinates, Thus, these coordinates need to be binded only once per deformable mesh LOD. The same applies for the appearance sets: even though they are used by several virtual humans, each needs to be sent only once to the GPU. Note that each joint transformation is sent to the GPU as two vectors of four floating points

(see Section 2.2), retrieved from the cache filled in the Animation phase.

For the rigid meshes, the process is quite different, since all the vertex deformations have been achieved in a preprocess. We develop here the second pass in pseudo-code:

```
For each human template:
   For each rigid mesh LOD:
      bind texture coordinate buffer,
      bind indices buffer,
      For each appearance set:
         send to the GPU texture specular parameters,
         bind texture and segmentation maps,
         For each virtual human id:
            get the corresponding body,
            get the correct rigid animation keyframe,
            bind its vertex and normal buffer.
```

In the rendering phase of the rigid meshes, only the texture coordinates and indices can be binded at the LOD level, in opposition to the deformable meshes, where all mesh data is binded at this level. The reason is obvious: for a deformable mesh, all the components representing its mesh information (vertices, normals, *etc.*) are the same for all instances. It is only later, on the GPU, that the mesh is deformed to fit the skeleton posture of each individual. For a rigid mesh, its texture coordinates, along with its indices (to access the buffers), remain the same for all of their instances. However, since the vertices and normals are displaced in a preprocess and stored in the keyframes of a rigid animation, it is only at the individual level, where we know the animation played, that their binding can be achieved. Note that since the vertices sent to the GPU are already deformed, there is no specific work to be achieved in the vertex shader. Concerning the shadow computation phase, *i.e.*, the first pass, the pseudo-code is the same, but without sending useless data, like normal and texture information.

Rendering impostors is fast, thanks to the virtual human id list sorted by human template, appearance set, animation, and keyframe, that is updated at the Animation phase. Here follows the corresponding pseudo-code:

```
For each human template:
   get its impostor animations,
   For each appearance set:
      bind texture and segmentation maps,
      For each impostor animation:
         For each keyframe:
            bind normal map,
            bind UV map,
            For each virtual human id:
               get the corresponding body,
               get the correct point of view,
               send to GPU texture coordinates where
               to get the correct virtual human posture
               and point of view.
```

The **Path Planner** is performing the collision avoidance between virtual humans. It is at the Simulator stage that subgoals are set several frames ahead, and that the followed directions are interpolated by steering methods. The Path Planner cares only for collision avoidance, and runs at a lower frequency than the other presented stages. Note that we put this stage and the next one, the Behavior, after the Renderer, because the GPU is paralelly rendering. So, instead of waiting for the frame to finish being rendered, we concurrently use the CPU. The different algorithms used by the Path Planner are detailed in Section 5.

The **Behavior** is the phase exploiting the slots of virtual humans reaching new navigation graph vertices. All along the entire pipeline, virtual humans cannot change their current animation or steering, because it would invalidate our various sorted slots. This last stage is thus the only one which is allowed to change the steering and current animation sequence of virtual humans. It is always achieved at the end of the pipeline, one frame ahead. Basically, each time a character is entering a new graph vertex (detected at the Simulator phase), we apply a probability to change the steering and / or animation. For instance, a character entering a new vertex with a walk animation clip has a probability to start playing another animation sequence, *e.g.*, an idle one.

3.3. Motion Kits

We have developed three levels of representation for the virtual humans: the deformable meshes, the rigid meshes, and the impostors (see Section 2). When playing an animation sequence, a virtual human is treated differently depending on its current distance and eccentricity to the camera, *i.e.*, the current level of detail it uses. For clarity purpose, we recall giving an animation clip a different name depending on which level of detail it applies to. An animation clip intended for a deformable mesh is a skeletal animation, one for a rigid mesh is a rigid animation, and finally, an animation clip for an impostor is an impostor animation.

We have already shown that the main advantage of using less detailed representations is the speed of rendering. However, for the memory, the cost of storing an animation sequence for a deformable, a rigid mesh, or an impostor is impressively growing (see Figure 2). From this, it is obvious that the number of animation sequences stored must be limited for the less detailed representations. It is also true that we want to keep as many skeletal animation clips as possible for the deformable meshes, firstly, because their storage requirement is cheap, and secondly, for variety purpose. Indeed, deformable meshes are at the forefront, close to the camera, and several virtual humans playing the same animation clip are immediately noticed.

The issue arising is then the switching from a level of representation to another. For instance, what should happen if a deformable mesh performing a walk cycle reaches the limit at which it switches to the rigid mesh representation? If a rigid animation with the same walk cycle (same speed) has been precomputed, switching is done smoothly. However, if the only rigid animation available is a fast run cycle, the virtual human will “pop” from a representation to the other, greatly disturbing the user. We therefore need each skeletal animation to be linked to a ressembling rigid animation, and similarly to an impostor animation. For this reason, we have

developed the motion kit data structure. We first describe the motion kit data structure in Section 3.3.1 and then its implementation in Section 3.3.2

3.3.1. Data Structure

A motion kit holds several items:

- A name, identifying what sort of animation it represents, *e.g.*, walk_1.5ms,
- Its type, determined by four identifiers: action, subaction, left arm action, and right arm action,
- A link to a skeletal animation,
- A link to a rigid animation,
- A link to an impostor animation.

Each virtual human knows only the current motion kit it uses. Then, at the Animator stage, depending on the distance of the virtual human to the camera, the correct animation clip is used. Note that there is always a 1:1 relation between the motion kits and the skeletal animations, *i.e.*, a motion kit is useless if there is no corresponding skeletal animation. As for the rigid and impostor animations, their number is much smaller than for skeletal animations, and thus, several motion kits may point to the same rigid or impostor animation. For instance, imagine a virtual human using a motion kit representing a walk cycle at 1.7 m/s. The motion kit has the exact skeletal animation needed for a deformable mesh (same speed). If the virtual human is a rigid mesh, the motion kit may point to a rigid animation at 1.5 m/s, which is the closest one available. And finally, the motion kit also points to the impostor animation with the closest speed. The presented data structure is very useful to easily switch from a representation to another. In Figure 9, we show a schema representing a motion kit and its links to different animation clips. All the motion kits and the animations are stored in a database, along with the links joining them (see Section 3.4). One may wonder what the four identifiers are for. They are used as categories to sort the motion kits. With such a classification, it is easy to randomly choose a motion kit for a virtual human, given certain constraints. Firstly, the action type describes the general kind of movement represented by the motion kit. It is defined as either:

- stand: for all animations where the virtual human is standing on its feet,
- sit : for all animations where the virtual human is sitting,
- walk : for all walk cycles, or
- run : for all run cycles.

The second identifier is the subaction type, which more restrains the kind of activity of the motion kit. Its list is nonexhaustive, but it contains descriptors such as talk, dance, idle, *etc.* We have also added a special subaction called none, which is used when a motion kit does not fit in any of the other subaction types. Let us note that some action / subaction couples are likely to contain no motion kit at all. For instance, a motion kit categorized as a sit action and a dance subaction is not likely to exist. The third and fourth identifiers: left and right arm actions are used to add some specific animation to the arms of the virtual humans. For instance, a virtual human can walk with the left hand in its pocket and the right hand holding a cellphone. For now, we have three categories that are common to both identifiers: none, pocket, and cellphone. However, this list can be extended to other possible arm actions. For instance, holding an umbrella, pull a caster suitcase, or scratch one's head.

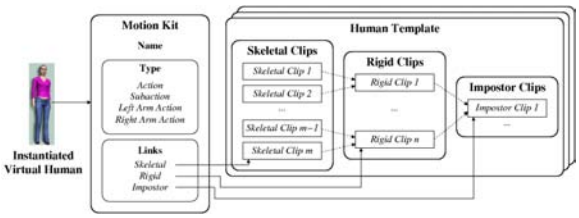


Figure 9: *Example of motion kit structure. On the left, a virtual human instantiated from a human template point to the motion kit it currently uses. In the center, a motion kit with its links identifying the corresponding animations to use for all human templates.*

When one creates a varied crowd, it is simple for each virtual human to ask randomly for one of all the motion kits available. If the need is more specific, like a crowd following a path, it is easy to choose only the adequate walk / run motion kits, thanks to the identifiers.

3.3.2. Implementation

In our architecture, the motion kits are stored in a four-dimensional table:

```
Table[ action id][ subaction id]
     [left arm action id][right arm action id].
```

For each combination of the four identifiers, a list of motion kits corresponding to the given criteria is stored. As previously mentioned, not all combinations are possible, and thus, some lists are empty. In Figure 10, a virtual human is playing a skeletal animation, linked to a motion kit with the following identifiers: walk, none, cellphone, pocket. In our architecture, an animation (whatever its level of detail) is dependent on the human template playing it : for a deformable mesh, a skeletal animation sequence specifies how its skeleton is moved, which causes the vertices of the mesh to get deformed on the GPU. Since each human template has its own skeleton, it is impossible to share such an animation with other human templates. Indeed, it is easy to imagine the difference there is between a child and an adult skeleton. For a rigid animation, it is the already deformed vertices and normals that are sent to the GPU, thus such an animation is specific to a mesh, and can only be performed by a virtual human having this particular set of vertices, *i.e.*, issued from

Figure 10: *A virtual human using a motion kit with identifiers: walk, none, cellphone, pocket.*

the same human template. Finally, an impostor animation clip is stored as a sequence of pictures of the virtual human. It is possible to modify the texture and color used for the instances of the same human template, but it seems obvious that such animation pictures cannot be shared by different human templates. This specificity is reflected in our implementation, where three lists of skeletal, rigid, and impostor animations are stored for each human template.

It follows that each motion kit should also be human template-dependent, since it has a physical link to the corresponding animation triplet. However, this way of managing the data is far from optimal, because usually, an animation (whatever its level of detail) is always available for all the existing human templates. It means that, for instance, if a template possesses an animation imitating a monkey, all other human templates are likely to have it. Thus, making the information contained in a motion kit human template-dependent would be redundant. We introduce 2 simple rules that allow us to keep a motion kit independent from a human template:

1. For any motion kit, all human templates have the corresponding animations.
2. For all animations of all human templates, there is a corresponding motion kit.

We now explain how, thanks to these assertions, we can keep a motion kit independent from the human templates and still know to which animation triplet it should link. First, note that each human template contains amongst other things:

- A list of skeletal animations,
- A list of rigid animations,
- A list of impostor animations.

Now, following the two rules mentioned above, all human templates contain the same number of skeletal animations, the same number of rigid animations, and the same number of impostor animations. If we manage to sort similarly these animation lists for all human templates, we can link the motion kits with them by using their index in the lists. We show a simple example in Figure 9, where a structure representing the human templates is depicted: each human template contains a list of skeletal, rigid, and impostor animations. On the left of the image, a motion kit is represented, with all its parameters. Particularly, it possesses three links that indicate where the corresponding animations can be found for all human templates. These links are represented with arrows in the figure, but in reality, they are simply indices that can be used to index each of the three animation lists for all human templates.

With this technique, we are able to treat all motion kits independently from the human templates using them. The only constraint is to respect the rules (1) and (2).

3.4. Database Management

We use the locomotion engine of Glardon *et al.* [GBT04b, GBT04a] to generate various locomotion cycles. Although this engine is fast enough to generate a walk or run cycle in real-time, it cannot keep up that rythm with thousands of virtual humans. When this problem first occured, the idea of precomputing a series of locomotion cycles and store them in a database came up. Since then, this system has proved very useful for storing other unchanging data. The main tables that can be found in the database are the following:

- Skeletal animations,
- Rigid animations,
- Impostor animations,
- Motion kits,
- Human templates, and
- Accessories.

In this Section, we detail what advantages and drawbacks we meet by using such a database, and what kind of information we can safely store there.

As previously mentioned, all the skeletal, rigid and impostor animations can neither be generated online, nor at the initialization phase of the application, because the user would have to wait during an important amount of time before the simulation launch. This is why the database is used. With it, the only work that needs to be done at initialization is to load the animation sequences, so that they are ready when needed at runtime. Although this loading phase may look time consuming, it is quite fast, since all the animation data is serialized into a binary format. Within the database, the animation tables have four important fields [†]: unique id, motion kit id, template id and serialized data. For each animation entry A, its motion kit id is later used to create the

[†] By field, understand a column in the database that allows for queries.

necessary links (see previous Section), while its template id is needed to find to which human template *A* belongs. It also allows to restrain the number of animations to load to the strict minimum, *i.e.*, only those needed for the human templates used in the application. It is mainly the serialized data that allows to distinguish a skeletal from a rigid or a impostor animation. For a skeletal animation, we mainly serialize all the information concerning the orientation of each joint for each keyframe. With a rigid animation, for each keyframe, a set of already deformed vertices and normals are saved. Finally, for a impostor animation, two series of images of the human template are kept (the normal and the UV map) for several keyframes and points of view.

Another table in the database is used to store the motion kits. It is important to note that since they are mainly composed of simple data, like integers and strings (see previous Section), they are not serialized in the database. Instead, each of their elements is introduced as a specific field: unique id, name, speed, four identifiers (action id, subaction id, left arm action id, right arm action id), and two special motion kit ids (rigid motion kit id, impostor motion kit id). When loading a motion kit *M* from the database, its basic information, *i.e.*, speed, name, *etc.*, are directly extracted to be saved in our application. Each of the two special motion kit ids is an index referring to another motion kit. This reference is necessary to complete the linking between *M* and its corresponding rigid and impostor animations.

We have introduced in the database a table in order to store the unchanging data of the human templates. Indeed, we have some human templates already designed and ready to be used in the crowd simulation. This table has the following fields: unique id, name, skeleton hierarchy, and skeleton posture. The skeleton hierarchy is a string summarizing the skeleton features, *i.e.*, all the joint names, ids, and parent. When loading a human template, this string is used to create its skeleton hierarchy. The skeleton posture is a string giving the default posture of a skeleton : with the previous field, the joints and their parents are identified, but they are not placed. In this specific field, we get for each joint its default position and orientation, relatively to its parent. As one can notice, for now the human template table is incomplete, *e.g.*, the appearance sets are missing, and no information is serialized, similarly to the motion kits. This is mainly due to a lack of time (indeed, as of today, our crowd simulator is still being developed). But it certainly is an advantage to further fill this table with more data in a binary format, so that the loading of human templates is faster at initilization.

Finally, the database possesses two tables dedicated to accessories. An accessory is a mesh used to add variety and believability to the appearance of the virtual humans. For instance, it can be a hat, a pair of glasses, a bag, *etc.* (see Section 4.3 for more details). In a first table, we store the elements specific to an accessory, independently from the human template wearing it : unique id, name, type, serialized data. In the serialized data is stored all the vertex, normal and texture information to make an accessory displayable. The second table is necessary to share information between the accessories and the human templates. As specified in Section 4.3, the displacement of a specific accessory relatively to a joint is different for each human template. This displacement is stored as a matrix. So, in this second table, we employ a field template id and a field accessory id to know exactly where the field matrix must be used. Thus, for each accessory / human template couple, corresponds an entry within this table. Note that we also store there the joint to which the accessory needs to be attached. This is because in some special cases, they may differ from a skeleton to another. For instance, when we attach a back pack to a child template, the joint used is a vertebra that is different from the one for an adult template.

Using a database to store serialized information has proven to be very useful, because it greatly accelerates the initialization time of the application. The main problem is its size, which increases each time a new element is introduced into it. However, with real-time constraints, we allow ourselves to have a sufficiently large database within reasonable limits to obtain varied crowds.

3.5. Performance

We have just detailed the different necessary steps to create and exploit a fast architecture for simulating crowds. We first showed the interest of using several representations, *i.e.*, deformable meshes, rigid meshes, and impostors. Then, we fully detailed each step of our pipeline for fast animation and rendering of thousands of virtual humans. Through the use of motion kits, we allowed for switching smoothly from a representation to another, limiting animation popping artefacts. We exposed how a database can be exploited to store all unchanging data, and finally, we introduced a shadow map algorithm adapted to crowds.

We now expose the performance obtained with this architecture. In Figure 2, the various storage requirements, depending on the animation types are exposed. In Figure 11, we compare the frame rates obtained in two cases. Firstly, when sorted virtual human lists are exploited, as detailed in Section 3.1. Secondly, when the Animator and Renderer stages do not use sorted lists, but directly each virtual human, one after another, in no spescific order. With such a process, all the information needed by the GPU has to be sent for each virtual human, independently from the data that may be shared by several of them. As one can observe in Figure 11, when using highly detailed deformable meshes, the results obtained with or without sorted lists are almost similar. This can be explained by the communications sent from the CPU to the GPU (joint transmission): such transmissions imply a pipeline flush for each rendered virtual human, thus becom-

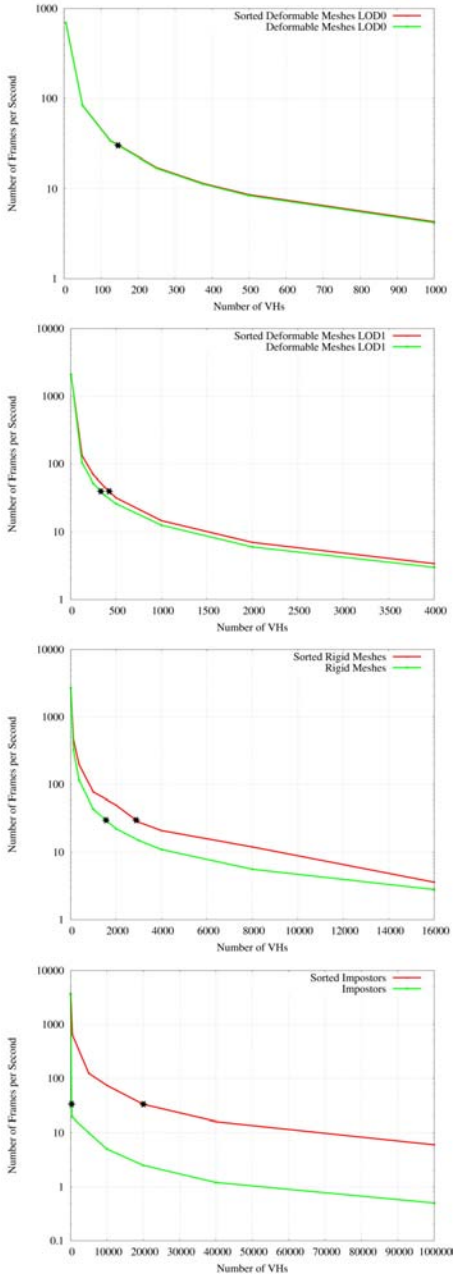


Figure 11: *Frames per second obtained for (a) highly detailed deformable meshes, (b) simple deformable meshes, (c) rigid meshes, and (d) impostors. The red lines show the results obtained when working with sorted lists, the green ones with a naive approach. The stars indicate the results for 30 frames per second. (e) Conditions in which the tests have been achieved: five human templates, steering and animation enabled, no shadows, no accessories, no collision avoidance.*

ing the bottleneck of the application. However, when less detailed representations are exploited, the advantage of sorting the lists becomes clear. A few images directly obtained from our running architecture are shown in Figure 6(left) and 7(c). In Figure 6(right), one can observe the distance at which the virtual humans switch to lower representations: in red are the rigid meshes, and in green the impostors.

4. Crowd Variety

When simulating a small group of virtual humans, it is easy to make them look singularly different: one can use several human templates and textures for each virtual human present in the scene, and assign them different animations. However, when the group extends to a crowd of thousands of people, this solution becomes unfeasible. First, in terms of design, it is unimaginable to create one mesh and series of animations per individual. Moreover, the memory space required to store all the data would be far too demanding. There is no direct solution to this problem, but it is however possible to achieve good results by multiplying the levels where variety can be introduced. First of all, several human templates can be used. Secondly, for each template, several textures can be designed. Thirdly, the color of each part of a texture can be varied so that two virtual humans issued from the same template and sharing the same texture have not the same clothes / skin / hair color. Finally, we also develop the idea of accessories, which allows a human mesh to be "augmented" with various objects such as a hat, a watch, a back pack, glasses, etc. Variety can also be achieved through animation. We mainly concentrate on the locomotion domain, where we vary the movements of the virtual humans in two ways. Firstly, by generating in a preprocess several locomotion cycles (walk and run) at different speeds, that are then played by the virtual humans online. Secondly, we use offline inverse kinematics to enhance the animation sequences with particular movements, like having a hand in the pocket, or at the ear as if making a phone call. In the following Section, we further develop each necessary step to vary a crowd in appearance: in Section 4.1, we show the three levels where variety can be achieved. Then, in Section 4.2, we detail how we segment the texture of a virtual human in order to apply varied colors to each identified body part. Moreover, accessories are fully explained in Section 4.3. We also describe animation variety in Section 4.4.

4.1. Variety at Three Levels

When referring to appearance variety, we mean how we modulate the rendering aspect of each individual of a crowd. This term is completely independent from the animation sequences played, the motion planning or the behavior of the virtual humans. First of all, let us remind that a human template is a data structure containing:

- A skeleton, defining what and where are its joints,

- A set of meshes, representing its different levels of detail,
- Several appearance sets, *i.e.*, textures and their corresponding segmentation maps,
- A set of animation sequences that can only be played by this human template.

For further indications on the human template structure, the reader is invited to refer to Section 2. We apply appearance variety at three different levels. The first, coarsest level is simply the number of human templates used. It seems obvious that the more human templates, the more variety. In Figure 12, we show five different human templates to illustrate this. The main issue when designing many human templates is the time required to design them and the memory requirements to store them. Their number needs thus to be limited. In order to mitigate this problem, we further vary the human templates by creating several textures and segmentation map sets for each one of them. For simplification, we designate a texture and its associated segmentation maps as an appearance set. The second level of variety is represented by the texture of an appearance set. Indeed, once an instance of a human template is provided with an appearance set, it automatically assumes the appearance of the corresponding texture. Of course, changing appearance set, and thus, texture, does not change the shape of the human template. For instance, if its mesh contains a pony tail, it will remain whatever the texture applied. However, it can impressively modify the appearance of the human template. In Figure 13, we show five different textures applied to the same human template. Finally, at the third level, we can play with color variety on each body part of the texture, thanks to the segmentation maps of the appearance set. We fully dedicate the next Section to this particular level. In Figure 14, we show several color modulated instances of a single mesh and appearance set.

Figure 12: *Five different human templates.*

4.2. Color Variety

Human templates possess several textures, improving the sense of variety. But too often, characters sharing the same texture, *i.e.*, looking exactly the same, appear in the vicinity of the camera, breaking the feeling of uniqueness of the spectator. Differentiating character body parts and then applying a unique combination of colors to each of them is a way to obtain variation inside a single texture.

Figure 13: *Five different textures of a single human template.*

Figure 14: *Several color varied instances of a single mesh and texture.*

4.2.1. Principles of the Method

Previous work on increasing the variety in color appearance for the characters composing a crowd share the common idea of storing the segmentation of body parts in a single alpha layer, *i.e.*, each body part is represented by a defined level of intensity of the alpha channel. Tecchia *et al.* [TLC02] use multi-pass rendering and the alpha channel to select parts to render for impostors. Dobbyn *et al.* [DHOO05] and De Heras *et al.* [dHCSM*05] avoid multi-pass rendering by using programmable graphics hardware. They also extend the method for being usable by 3D virtual humans too. Figure 15 depicts a typical texture and its associated alpha zone map. The method is based on texture color modulation: the final color C_b of each body part is a modulation of its texture color C_t by a random color C_r:

$$C_b = C_t C_r. \quad (3)$$

Colors C_b, C_t, and C_r can take values between 0.0 and 1.0. In order to have a large panel of reachable colors, C_t should be as light as possible, *i.e.*, near to 1.0. Indeed, if C_t is too dark, the modulation by C_r will give only dark colors. On the other hand, if C_t is a light color, the modulation by C_r will provide not only light colors, but also dark ones. This explains why part of the texture has to be reduced to a light luminance, *i.e.*, the shading information and the roughness of the material. The drawback of passing the main parts of the texture to luminance is that funky colors can be generated, *i.e.*, characters are dressed in colors that do not match. Some constraints have to be added when modulating colors randomly.

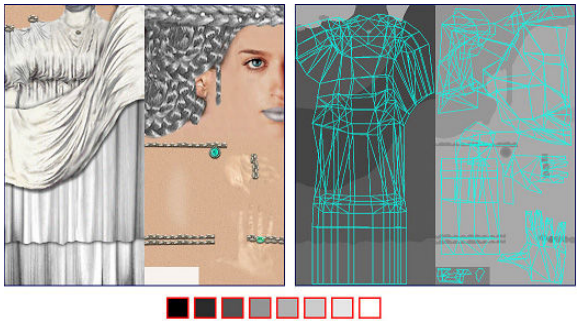


Figure 15: *Typical RGBA image used for color variety. The RGB part composes the texture and the alpha the segmentation map.*

4.2.2. HSB Color Spaces

The standard *RGB* color model representing additive color primaries of red, green, and blue is mainly used for specifying color on computer screens. With this system, it is hard to constrain colors effectively (see Figure 16). In order to quantify and control the color parameters applied to the crowd, a user-friendly color is used. Smith [Smi78] proposed a model that deals with everyday life color concepts, *i.e.*, hue, saturation and brightness, which are more linked to the human color perception than the *RGB* system. This system is called the *HSB* (or *HSV*) color model (see Figure 17):

- the hue defines the specific shade of color, as a value between 0 and 360 degrees,
- the saturation denotes the purity of the color, *i.e.*, highly saturated colors are vivid while low saturated colors are washed-out, like pastels. Saturation can take values between 0 and 100, and
- the brightness measures how light or dark a color is, as a value between 0 and 100.

In the process of designing virtual human color variety, localized constraints are dealt with: some body parts need very specific colors. For instance, skin colors are taken from a specific range of unsaturated shades with red and yellow

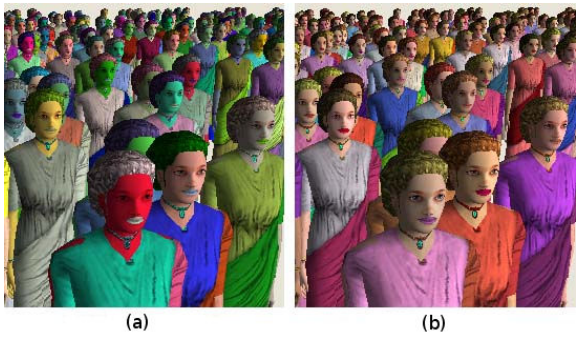


Figure 16: *Random color system (a) versus HSB control (b).*

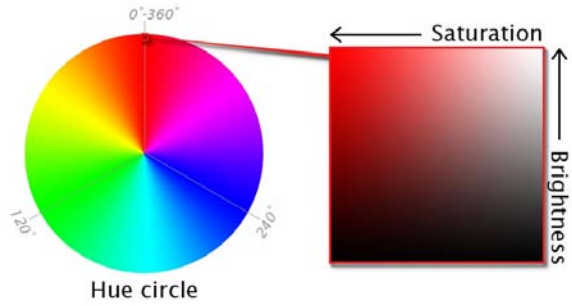


Figure 17: *HSB color space. Hue is represented by a circular region. A separate square region may be used to represent saturation and brightness, i.e., the vertical axis of the square indicates brightness, while the horizontal axis corresponds to saturation.*

dominance, almost deprived of blue and green. Eyes are described as a range from brown to green and blue with different levels of brightness. These simple examples show that one cannot use a random color generator as is. The *HSB* color model enables control of color variety in an intuitive an flexible manner. Indeed, as shown in Figure 18, by specifying a range for each of the three parameters, it is possible to define a 3D color space, called the *HSB* map.

Figure 18: *The HSB space is constrained to a three dimensional color space with the following parameters (a): hue from* 20 *to* 250*, saturation from* 30 *to* 80 *and brightness from* 40 *to* 100*. Colors are then randomly chosen inside this space to add variety on the eyes texture of a character (b).*

4.2.3. Segmentation Maps

The method presented in Section 4.2.1 is perfectly adequate when viewing crowds at far distances. However, when some individuals are close to the camera, the method tends to have too sharp transitions between body parts. There is no smooth blending between different parts, *e.g.*, the transition between skin and hair, as depicted in Figure 19. Also, character closeups bring the need for a new method capable of handling detailed color variety, for instance, subtle make-up effects for female characters. Moreover, at short distances, materials should be illuminated differently to obtain realistic characters at the forefront. To obtain a detailed color variety method, we propose, for each appearance set, to use segmentation maps.

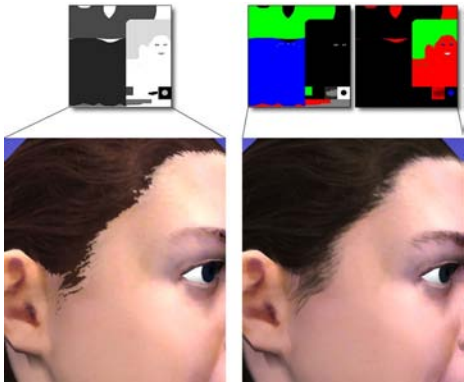


Figure 19: *Closeup of the transition between skin and hair: artifacts in previous methods when segmenting body parts in a single alpha layer (left), smooth transitions between parts with our method (right).*

A segmentation map is a four channel image, delimiting four body parts (one per channel) and sharing the same parameterization as the texture of the appearance set. The intensity of each body part is thus defined throughout the whole body of each character, *i.e.*, 256 levels of intensity are possible for each part, 0 meaning it is not present at this location, and 255 meaning it is fully present. For our virtual humans, we have made experiments with eight body parts, *i.e.*, two *RGBA* segmentation maps per appearance set. The results are satisfying for our specific needs, but the method can be used with more segmentation maps if more parts are needed. For instance, it would be possible to use the method for adding color variety to a city by creating segmentation maps for buildings. Using segmentation maps to efficiently distinguish body parts also provides two advantages over previous methods:

- Possibility to apply different illumination models to each body part. With previous methods, achieving such effects requires costly fragment shader branching.

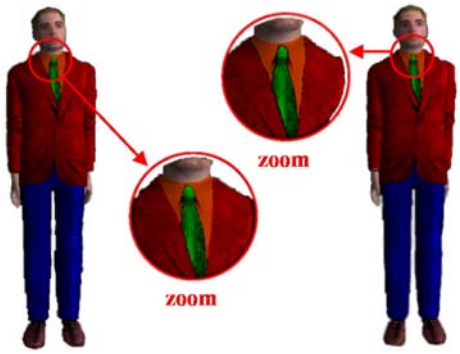


Figure 20: *Bilinear filtering artifacts in the alpha layer can be seen in the right zoomed-in version, near the borders of the orange shirt, the green tie and the red vest [Mau05].*

- Possible mipmapping activation and use of linear filtering, which greatly reduce aliasing. Since previous methods use the alpha channel of the texture to segment their body parts, they cannot benefit from this algorithm, which causes the appearance of artefacts at body part seams (see Figure 20).

Figure 21 depicts the different effects achievable with our color variety method: make-up, cloth patterns, freckles, etc, and localised specular parameters. The segmentation maps are designed manually. Ideally, for a given pixel, we wish the sum of the intensity of each body part to reach 255. When designing the segmentation maps with a software like Adobe Photoshop, unwanted artefacts may later appear within the smooth transitions between body parts. Indeed, some pixel sums of intensity levels may not reach 255. For instance, imagine the transition between the hair and the skin of a virtual human. A pixel of the segmentation map may reach a contribution of 100 for the skin part, while the hair part contribution is of 120. Their sum amounts to 220. Although this is not an issue while designing the segmented body parts in Photoshop, it leads to problems when trying to normalize the contributions in the application. Indeed, with simple normalization, such pixels compensate the uncomplete sum with a black contribution, thus producing a final color much darker than expected. This is illustrated in Figure 22. The proposed solution is to compensate this lack with white instead of black, to get a real smooth transition without unwanted dark zones. The results obtained with our three levels of appearance variety are illustrated in Figure 23, where several instances of a single human template are displayed, taking full advantage of all available appearance sets and color variety.

4.2.4. Color Variety Storage

Each segmentation map of a human template is divided into four different body parts. Each of these parts has a specific color range, and specularity parameters. The

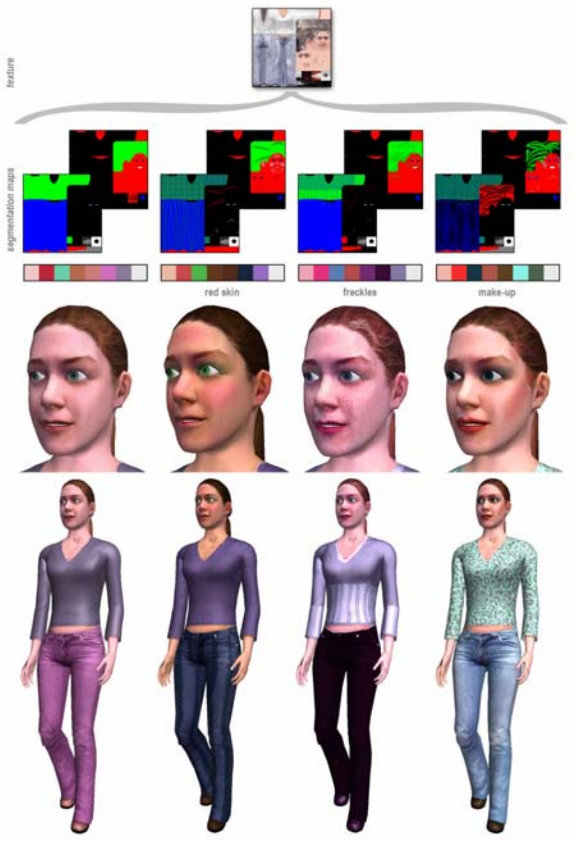


Figure 21: *Examples of achievable effects through appearance sets (make-up, freckles, clothes design, etc), and per body part specular parameters (shiny shoes, glossy lips, etc).*

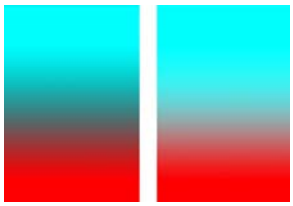


Figure 22: *A blue to red gradient. (a) The sum of the red and blue contributions does not reach 255 in some pixels, causing the gradient to suffer from an unwanted black contribution, (b) A white contribution is added so that the sum of contributions is always 255.*

Figure 23: *Several instances of a single human template, exploiting all its appearance sets and color variety.*

eight body parts we need are designed in two different segmentation maps, *i.e.*, two *RGBA* images, each containing four channels and thus four body parts. At its birth, each character is assigned a unique set of eight random colors from the constrained color spaces, similarly to De Heras *et al* [dHCSM*05]. These eight colors are stored in eight contiguous *RGB* texels, starting at the top-left of a 1024×1024 image, called Color Look Up Table (CLUT). We show an illustration of a CLUT in Figure 24. Therefore, if a 1024×1024 image is used for storing the CLUT, it is possible to store a set of up to:

$$\frac{1024 \cdot 1024}{8} = 131,072 \quad (4)$$

unique combinations of colors. Note that illumination parameters are set per body part and thus not saved within the CLUT, but directly sent to the GPU.

4.3. Accessories

We have already described how to obtain varied clothes and skin colors by using several appearance sets. Unfortunately, even with these techniques, the feeling of watching the same person is not completely overcome. The main reason is the lack of variety in the human templates used. Indeed, it is very often the same human template (or a small number of them) that is used for the whole crowd, resulting in large groups of similarly shaped humans. We cannot increase too much the number of human templates, because it requires a lot of work for a designer: create the human template, its textures, its skinning, its different levels of detail, *etc.* Note that the number of human templates is also limited by The storage. However, in real life, people have different haircuts, they wear hats or glasses, carry bags, etc. These particularities may look like details, but it is with the sum of those details that we are able to distinguish anyone. In this Section, we first explain what exactly are accessories. Then, we show from a technical point of view the different kinds of accessories we have identified, and how to develop each of them

Figure 24: *A CLUT image used to store the color of each virtual human body parts and accessories.*

in a crowd application. An accessory is a simple mesh representing any element that can be added to the original mesh of a virtual human. It can be a hat as well as a handbag, or glasses, a clown nose, a wig, an umbrella, a cellphone, etc. Accessories have two main purposes: firstly, they allow to easily add appearance variety to virtual humans. Secondly, they make characters look more believable: even without intelligent behavior, a virtual human walking around with a shopping bag or a cellphone looks more realistic than the one just walking around. The addition of accessories allows a spectator to identify himself to a virtual human, because it performs actions that the spectator himself does everyday. We basically distinguish two different kinds of accessories that are incrementally complex to develop. The first group is composed of accessories that do not influence the movements of a virtual human. For instance, whether someone wears a hat or not will not influence the way he walks. The second group gathers the accessories requiring a small variation in the animation clip played, *e.g.*, a virtual human moving with an umbrella or with a bag still walks the same way, but the arm in contact with the accessory needs an adapted animation sequence.

4.3.1. Simple Accessories

The first group of accessories does not necessitate any particular modification of the animation clips played. They simply need to be correctly "placed" on a virtual human. Each accessory can be represented as a simple mesh, independent from any virtual human. First, let us lay the problem for a single character. The issue is to render the accessory at the correct position and orientation, accordingly to the movements of the character. To achieve this, we can "attach" the accessory to a specific joint of the virtual human. Let us take a real example to illustrate our idea : imagine a walking person wearing a hat. Supposing that the hat has the correct size and does not slide, it basically has the same movement as the head of the person as he walks. Technically, this means that the series of matrices representing the head movement are the same for the hat movement. However, the hat is not placed at the exact position of the head. It usually is on top of the head and can be oriented in different ways, as shown in Figure 25. Thus, we also need the correct displacement between the head joint position and the ideal hat position on top of it. In summary, to create a simple accessory, our needs are the following:

- For each accessory:
  - A mesh (vertices, normals, texture coordinates),
  - A texture,
- For each human template / accessory couple:
  - The joint to which it must be attached,
  - A matrix representing the displacement of the accessory, relatively to the joint.

Note that the matrix representing the displacement of the accessory is not only specific to one accessory, but specific to each human template / accessory couple. This allows us to vary the position, the size, and the orientation of the hat depending on which virtual human mesh we are working with. This is depicted in Figure 25, where the same hat is worn differently by two human templates. It is also important to note that the joint to which the accessory is attached is also dependent on the human template. This was not the case at first : a single joint was specified for each accessory, independently from the human templates. However, we have noticed that depending on the size of a virtual human, some accessories may have to be attached to different joints. For instance, a backpack is not attached to the same vertebra if it is for a child or a grown up template. Finally, with this information, we are able to assign each human template a different set of accessories, increasing greatly the feeling of variety.

4.3.2. Complex Accessories

The second group of accessories we have identified is the one that requires slight modifications of the animation sequences played. Concerning the rendering of the accessory, we still keep the idea of attaching it to a specific joint of the virtual human. The additional difficulty is the modification of the animation clips to make the action realistic. For instance, if we want to add a cellphone accessory, we also need the animation clips allowing the virtual human to make a phone call. We focus only on locomotion animation sequences. Our raw material is a database of motion captured walk and run cycles that can be applied to the virtual humans. From each animation clip, an adjustment of the arm

Figure 25: *Two human templates wearing the same hat, in their default posture. The pink, yellow and blue points represent the position and orientation of the root, the head joint* $(m1)$*, and the hat accessory* $(m2)$*, respectively.*

motion is performed in order to obtain a new animation clip integrating the wanted movement, *e.g.*, hand close to the ear. These animation modifications can be generalized to other movements that are independent from any accessory, for instance, hands in the pockets. This is why we fully detail the animation adaptation process in Section 4.4.2.

4.3.3. Loading and Initialization

In this Section, we focus on the architectural aspect of accessories, and how to assign them to all virtual humans. First of all, each accessory has a type, *e.g.*, "hat" or "back pack". We differentiate seven different types, but this number is arbitrary. In order to avoid the attribution of, for instance, a cowboy hat and a cap on the same head, we never allow a character to wear more than one accessory of each type. To distribute accessories to the whole crowd, we need to extend the following data structures (introduced in Section 2):

- **Human template**: each human template is provided with a list of accessory ids, sorted by type. This way, we know which template can wear which accessory. This process is necessary, since all human templates cannot wear all accessories. For instance, a school bag would suit the template of a child, but for an adult template, it would look much less believable,
- **Body entity**: each body entity possesses one accessory slot per existing type. This allows to later add up to seven accessories (one of each type) to the same virtual human.

We also create two data structures to make the accessory distribution process efficient:

- **Accessory entity**: each accessory itself possesses a list of body ids, representing the virtual humans wearing it. They are sorted by human template.
- **Accessory repository**: an empty repository is created to receive all accessories loaded from the database. They are sorted by type.

At initialization, the above data structures are filled. We detail this process in the following pseudo-code:

```
For each accessory in database:
   load its data contained in the database,
   create its vertex buffer (for later rendering),
   insert it into the accessory repository (sorted by type).
   For each human template h:
      For each accessory a suitable to h :
         insert a's id into h's list l (sorted by type).
   For each body b:
      get human template h of b,
      get accessory id list l of h,
      For each accessory type t in l:
         choose randomly an accessory a of type t,
         assign a to the correct accessory slot of b,
         push b's id in a's body id list (sorted by hu-
man template).
```

The process of filling these data structures is done only once at initialization, because we assume that once specific accessories have been assigned to a virtual human, they never change. However, it would be easy to change online the accessories worn, through a call to the last loop. Note that a single vertex buffer is created for each loaded accessory, independently from the number virtual humans wearing it.

4.3.4. Rendering

Since the lists introduced in the previous Section are all sorted accordingly to our needs, the rendering of accessories is much facilitated. We show in the following pseudo-code our pipeline:

```
1 For each accessory type t of the repository:
2    For each accessory a of type t:
3       bind vertex buffer of a,
4 send a's appearance parameters to the GPU,
5 get a's list l of body ids (sorted by human template).
6 For each human template h in l:
7          get the joint j of h to which a is attached,
8    get the original position matrix m1 of j,
9    get the displacement matrix m2 of couple [a,h],
10    For each body b of h:
11       get matrix m3 of b's current position,
12       get matrix m4 of j's current deformation for b,
13       multiply current modelview matrix by mi (i=1..4),
14       call to vertex buffer rendering.
```

Although this pseudo-code may seem complex at first sight, it is quite simple and well optimized to minimize state switches. First of all, at line (3), each accessory has its vertex buffer binded. We can process this way, independently from the bodies, because an accessory never changes its shape or texture. Then, we process through each accessory's body id list (5). This list is sorted by human template (6), allowing us to retrieve information common to all its instances, *i.e.*, the joint j to which is attached the accessory (7), along with its original position matrix $m1$ in the skeleton (8), and

the original displacement matrix $m2$ between $m1$ and the desired position of the accessory (9). An example with a hat attached to the head joint of two human templates is illustrated in Figure 25. Once the human template data is retrieved, we iterate over each body wearing the accessory (10). A body entity also has specific data that is required: its position for the current frame (11), and the displacement of its joint, relatively to its original position, depending on the animation played (12). Figure 26 illustrates the transformation represented by these matrices. Finally, by multiplying the matrices extracted from the human template and body data, we are able to define the exact position and orientation of the accessory (13). The rendering of the vertex buffer is then called and the accessory is displayed correctly (14).

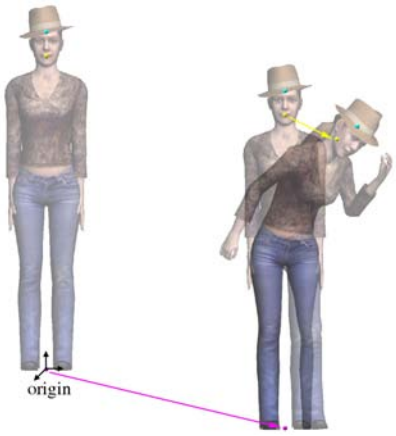


Figure 26: *Left: a human template in default posture. Right: the same human template playing an animation clip. The displacement of the body, relatively to the origin* ($m3$) *is depicted in pink, the displacement of the head joint due to the animation clip* ($m4$) *in yellow.*

4.3.5. Empty Accessories

We have identified seven different accessory types. And, through the accessory attribution pipeline, we assign seven accessories per virtual human. This number is important and the results obtained can be unsatisfying: indeed, if all characters wear a hat, glasses, jewelry, a back pack, etc, they look more like christmas trees than believable people. We need the possibility to have people without accessories too. To allow for this, we could simply randomly choose for each body accessory slot, whether it is used or not. This solution works, but a more efficient one can be considered. Indeed, at the rendering phase of a large crowd, testing each slot of each body to know whether it is used or not implies useless code branching, *i.e.*, precious computation time. We therefore propose a faster solution to this problem by creating empty accessories. An empty accessory is a fake one, possessing no geometry nor vertex buffer. It only possesses a unique id, similarly to all other accessories. At initialization, before loading the real accessories from the database, the following pseudo-code is executed:

```
For each accessory type $t$:
   create one empty accessory e of type $t$,
   put $e$ in the accessory repository (sorted by type),
   For each human template $h$:
      put $e$'s id in $h$'s accessory id list.
```

The second loop over the human templates is necessary in order to make all empty accessories compatible with all human templates. Once this preprocess done, the loading and attribution of accessories is achieved as detailed in Section 4.3.3. This fore introduction of empty accessories causes later their possible insertion in some of the accessory slots of the bodies. Note that if, for instance, a body entity gets an empty accessory for hat, reciprocally, the id of this body will be added to the empty accessory's body id list. This is illustrated with an example in Figure 27. One may wonder how the rendering is achieved. If keeping the same pipeline as detailed in Section 4.3.4, we meet troubles when attempting to render an empty accessory. Moreover, some useless matrix computation would be done. Our solution is simple. Since the empty accessories are the first ones to be inserted into the accessory repository (sorted by type), we only need to skip the first element of each type to avoid their computation and rendering. The pseudo code given in Section 3.4 only needs a supplementary line, which is:

```
1b skip first element of t.
```

With this solution, we take full advantage of accessories, obtaining varied people, not only through the vast choice of accessories, but also through the possibility of not wearing them. And there is no need for expensive tests within the rendering loop. In Figure 28, we show the results obtained when using accessories in addition to the appearance variety detailed in Section 2.

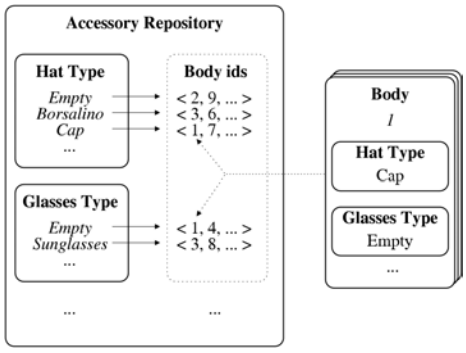


Figure 27: *Left: a representation of the accessory repository, sorted by type. Each accessory possesses its own list of body ids. Reciprocally, all bodies possess slots filled with their assigned accessories. Right: illustrated example of the accessory slots for body with id 1.*

Figure 28: *Several instances of a single human template, varied through the appearance sets, color variety, and accessories.*

4.3.6. Color Variety Storage

In Section 4.2.4, we detail how to apply color variety to the different body parts of a texture. The same method can be applied to the accessories. A human texture is segmented in eight body parts, each having its specific color range. At initialization, for each instantiated virtual human and each body part, a color is randomly chosen in a range to modulate the original color of the texture. Since accessories are smaller and less complex than virtual humans, we only use four different parts, *i.e.*, one segmentation map per appearance set. Then, similarly to the characters, each instance of each accessory is randomly assigned four colors within the *HSB* ranges defined for each part. These four random colors have also to be stored. We reemploy the CLUT used for storing the virtual humans color variety to save the colors of the accessories. In order not to confuse the color variety of the body parts and those of the accessories, we store the latter contiguously from the bottom-right of the CLUT (see Figure 24). Each character thus needs eight texels for its own color variety and $7*4$ other texels for all its potential accessories. This sums up to 36 texels per character. A $1024x1024$ CLUT is therefore able to roughly store more than 29000 unique color variety sets.

4.3.7. scalability

We can simulate a high number of virtual humans, thanks to our different representations. It is important to note that the above description of accessories solves only the case of dynamically animated virtual characters, *i.e.*, deformable meshes. However, if we want to ensure continuity when switching from a representation to another, it is important to also find a solution for the other LOD : a hat on the head of a virtual human walking away from the camera cannot suddenly disappear when the virtual human is switching to a lower representation. We develop here how to make accessories scalable. First, let us detail how accessories can be scaled to fit rigid meshes. An accessory has an animation clip of its own, similar to the animation of a particular joint of a virtual human. If we wanted to simply apply the rigid mesh principle to accessories, we would have to store an important quantity of information:

```
For each rigid animation:
   For each keyframe:
      For each vertex of the accessory:
         save its new position which is found through
         the animation matrices,
         save its corresponding normal, which is found
         through the animation matrices.
```

As one can see, this pipeline corresponds to the one used to store the vertices and normals of a rigid mesh at each keyframe of a defined animation clip. If we analyze this pipeline, we can observe that there is a clear redundancy in the information stored: firstly, an accessory is never deformed, which means that its vertices do not move, relatively to each other. They can be considered as a single group transformed by the animation matrices. The same applies to the normals of the accessory. Secondly, as detailed in Section 3.3, it is impossible to store in a database a rigid and an impostor animation clip for each existing skeletal animation. It follows that creating all the rigid / impostor versions of an animation clip for each possible accessory cannot be considered. In order to drastically diminish the information to store for an accessory in a rigid animation, we propose a solution in two steps: Firstly, as previously detailed, there is no need to store all the vertices and all the normals at each keyframe of an animation sequence, since the mesh is not deformed. It is sufficient to keep a single animation matrix per keyframe, valid for all vertices. Then, at runtime, the original mesh representing the accessory is transformed by the stored animation matrices. Secondly, we can regroup all accessories depending on the joint they are attached to. For instance, all hats and all glasses are attached to the head. So, basically, they all have the same animation. The only difference between a pair of glasses and a hat is the position where they are rendered, relatively to the head position (the hat is above the head, the glasses in front of it). So, we only need to keep this specific displacement for each accessory relatively to its joint. This corresponds to a single matrix per human template / accessory couple, which is completely independent from the animation clip played (see Section 4.3.1 and 4.3.4). In summary, with this solution, we only need:

```
For each rigid animation:
   For each keyframe:
      For each joint using an accessory:
         a single matrix representing
         the transformation of the joint at this keyframe,
```

and

```
For each human template / accessory couple (indepen-
dent of the animation):
   a matrix representing the accessory's displacement,
   relatively to the joint.
```

Scaling the accessory principle to impostors proves to be complicated. Once again, a naive approach would be as follows:

```
For each original impostor animation (without accessories):
   For all possible combinations of accessories:
      create a similar impostor animation directly
      containing these accessories.
```

One can quickly imagine the explosion the memory would endure, even when starting with only a few original impostor animations. We cannot afford to generate one impostor animation for each possible combination of accessories. The first possible simplification is to let the unnoticeable accessories disappear. Indeed, impostors are usually employed when the virtual humans are far from the camera, and thus, small details, taking only a few pixels can be ignored. Such accessories would be watches, jewelry, and others. Of course, it is also dependent on the distance from the camera where the impostors are used, and whether such disappearances are noticeable or not. As for larger accessories, like hats or bags, we are still working to find the best solution, but this work is in progress, and as of today, we have no finite solution to expose.

4.4. Animation Variety

As explained in previous Sections, it is possible to vary the appearance of individuals, even when issued from the same human template. However, we introduced in Section 3.3 the necessity to also provide a large variety of animation clips to the simulation. Virtual humans can be visually as different as possible, if they all perform the same animation, the result is not realistic at all. In this Section, we detail two techniques we employ to vary the animation of characters, while remaining in the domain of navigating crowds, *i.e.*, working with locomotion animations.

4.4.1. Locomotion

First of all, in order to obtain variety in animation, there is a great need for a huge set of raw animation cycles that can then be further varied. We recall here the locomotion engine of Glardon *et al.* that we have used to generate our original set of walk and run cycles. Glardon *et al.* have introduced a PCA-based walk engine capable of animating on the fly human-like characters of any size and proportions by generating complete locomotion cycles [GBT04b,GBT04a]. They have captured walk and run motions from several people, from which they have created a normalized model. There are mainly three high-level parameters which allow to modulate these motions:

- Personification weights: five people, different in height and gait have been captured while walking and running. This variable allows the user to choose how he wishes to parametrize these different styles.
- Speed: the five subjects have been captured at many different speeds. This parameter allows to choose at which velocity the walk/run cycle should be generated.
- Locomotion weights: this parameter defines whether the cycle is a walk or a run animation.

Thus, the engine is able to generate a whole range of varied locomotion cycles for a given character. To efficiently animate the locomotion of each individual, we generate in a preprocess a certain number of locomotion cycles for each human template. We have used this engine to generate over 100 different locomotion cycles per human template: for each one of them, we sample walk cycles at speeds varying from 0.5 m/s up to 2 m/s and similarly for the run cycles between 1.5 m/s and 3 m/s. Each human template is also assigned a particular personification weight so that it has its own style. With such a high number of animations, we are already able to perceive a sense of variety in the way the crowd is moving. Virtual humans walking together with different locomotion styles and speeds add to the realism of the simulation. Once provided with a large set of animation clips, the issue becomes to store and use them in an efficient way. In Section 3.4, we fully detail how the whole data is managed.

4.4.2. Accessory Movements

Variety in movement is one necessary condition for achieving believable synthetic crowds as individuals are seldom unrolling the sole locomotion cycle while moving from one place to another. The upper limb movements being not compulsory in locomotion, hands are most of the time exploited for accessory activities such as holding an object (cell phone, bag, umbrella, *etc.*) or are simply protected by remaining in the pocket of some cloth (see Figure 29). These activities constitute alternate coordinated movements that have to match the continuously changing constraints issued from the primary locomotion movement. Indeed, constantly reusing the same arm posture through the locomotion cycle leads to a loss a believability; for example a hand "in-the-pocket" should follow the pelvis forward-backward movement when large steps are performed. For these reasons, a specific animation cycle has to be defined also for an accessory movement that is to be exploited with locomotion. We achieve the accessory movement design stage after the design of the individual locomotion cycles for a set of discretized speeds. We exploit a Prioritized Inverse Kinematics solver [BB04] that allows combining various constraints with a priority level if necessary. The required input is:

- The set of locomotion cycles,
- One "first guess" posture of the hand and arm, possibly with the clavicle, designed with the skinned target character,
- The set of "effector" points to be constrained on the hand or arm, (see Figure 30, the three coloured cubes on the hand),
- For each effector, its corresponding target goal location expressed in other local frames of the body; for example relative to the head for a cell-phone conversation, or to the pelvis and thigh for a hand in a trousers' pocket (see Figure 30, the three corresponding coloured cubes attached to the pelvis),
- If an effector is more important than the others, the user can associate it with a greater priority level. Our solver ensures that the achievement of other effectors goals does not perturb the high priority one.

All the additional elements to the original locomotion cycles can be specified by an animator by locating them on the target character mesh in a standard animation software. The resulting set of parameters can be saved in a configuration file for the second stage of running the Inverse Kinematics adjustment of the posture for all frames of the locomotion cycles (Figure 31). The resulting accessorized locomotion cycles are saved in files for a further storage optimization stage. Figure 32 shows successive postures from such a movement.

Figure 29: *Examples of accessory movements (hands in the pocket, phone call, hand on hip, ...).*

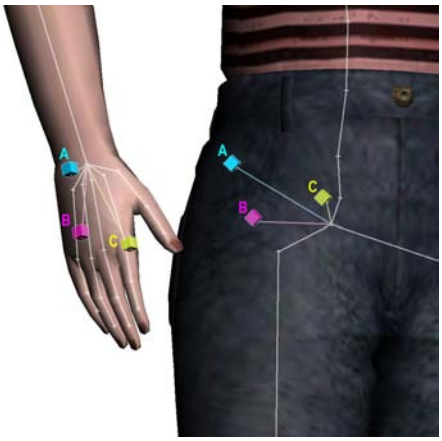


Figure 30: *Set of controlled effectors attached to the hand and corresponding goal positions attached to the pelvis.*

5. Motion Planning

Realistic real-time motion planning for crowds has become a fundamental research field in the Computer Graphics community. The simulation of urban scenes, epic battles, or other

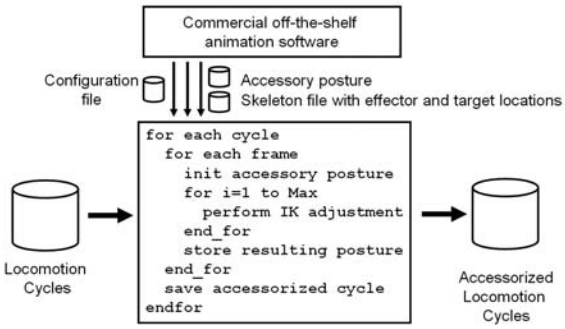


Figure 31: *Overview of the two-stage process for producing accessorized locomotion cycles.*

Figure 32: *Example of posture from an accessorized locomotion cycle.*

environments that show thousands of people in real time require fast and realistic crowd motion. Domains of application are vast: video games, psychological studies, architecture, and many others. We present a novel architecture offering a hybrid, scalable solution for real-time motion planning of thousands of characters in complex environments.

Real crowds are formed by thousands of individuals that move in a bounded environment. Each pedestrian has individual goals in space that he wants to reach, avoiding obstacles. People perceive their environment, and use this information to choose the shortest path in time and space that leads to their goal. Emergent behaviors can also be observed in crowds. For example, in places where the space is small and very crowded, people form lanes to maximize their speed. Also, when dangerous events such as fires occur, pedestrians tend to react in very chaotic ways to escape.

Planning crowd motion in real time is a very expensive task, which is often decoupled into two distinct parts: path planning and obstacle avoidance. Path planning consists in finding the best way to reach a goal. Obstacles can either be other pedestrians or objects that compose the environment. The path selection criteria are the avoidance of congested zones, and minimization of distance and travel time. Path

planning must also offer a variety of paths to spread pedestrians in the whole scene. Avoidance, on the other hand, must inhibit collisions of pedestrians with obstacles. For real-time simulations, such methods need to be efficient as well as believable.

Figure 33: *Pedestrians using our hybrid motion planning architecture to reach their goal and avoid each other.*

Multiple motion planning approaches for crowds have been introduced. As of today, several fast path planning solutions exist. Avoidance however, remains a very expensive task. Agent-based methods offer realistic pedestrian motion planning, especially when coupled with global navigation. This approach gives the possibility to add individual and cognitive behaviors for each agent, but becomes too expensive for a large number of pedestrians. Potential field approaches handle long and short-term avoidance. Long term avoidance predicts possible collisions and inhibits them. Short term avoidance intervenes when long-term avoidance alone cannot prevent collisions. These methods offer less believable results than agent-based approaches, because they do not provide the possibility to individualize each pedestrian. However, this characteristic also implies much lower computational costs.

We present a hybrid architecture to handle realistic crowd motion planning in real time. In order to obtain high performance, our approach is scalable. As briefly introduced in Section 3.2, we divide the scene into multiple regions of varying interest, defined at initialization and modifiable at runtime. According to its level of interest, each region is ruled by a different motion planning algorithm. Zones that attract the attention of the user exploit accurate methods, while computation time is saved by applying less expensive algorithms in other regions. Our architecture also ensures that no visible disturbance is generated when switching from an algorithm to another.

Our results shows that it is possible to simulate up to ten thousand pedestrians in real time with a large variety of goals. Moreover, the possibility to introduce and interactively modify the regions of interest in a scene offers a way for the user to select the desired performance and to distribute the computation time accordingly. A simulation of pedestrians taking advantage of our architecture to plan their motion in a city environment is illustrated in Figure 33.

The remainder of this Section is organized as follows: first, in Section 5.1, we introduce previous work in crowd motion planning. Then, in Section 5.2, we describe at a high-level our motion planning architecture, and how we exploit it to distribute regions of three different levels of interest. In Section 5.3, the integration of the various approaches employed and the optimizations applied to keep high frame rates are detailed. Finally, in Section 5.4, we run several tests in different conditions and environments to assess our architecture. Finally, limitations are discussed in Section 5.5.

5.1. Crowd Motion Planning Background

Crowd behavior and motion planning are two topics that have long been studied in fields such as Robotics and Sociology. More recently however, and due to the technology improvements, these domains have aroused the interest of the Computer Graphics community as well.

The first studied approach, *i.e.*, agent-based, represents a natural way to simulate crowds as independent individuals interacting with each other. Such algorithms usually handle short distance avoidance, and navigation remains local. Reynolds [Rey99] proposed to use simple rules to model crowds of interacting agents. Heïgeas *et al.* [HLTC03] introduced a model based on cellular automata and the physical properties of the environment, while Kirchner and Shadschneider [KS01] used static potential fields to rule a cellular automaton. Metoyer and Hodgins [MH03] proposed an avoidance algorithm based on a bayesian decision process. Nevertheless, the main problem with agent-based algorithms is their low performance. With these methods, simulating thousands of pedestrians in real time requires the use of particular machines supporting heavy parallelizations [Rey06]. Moreover, such approaches forbid the construction of autonomous adaptable behaviors, and can only manage crowds of pedestrians with local objectives.

To solve the problems inherent in local navigation, some behavioral approaches have been extended with global navigation. Bayazit *et al.* [BLA03] stored global information in nodes of a probabilistic roadmap to handle navigation. Sung *et al.* [SKG05] combined probabilistic roadmaps with motion graphs to find paths and animations to steer characters to a goal, while Lau and Kuffner [LK06] used precomputed search trees of motion clips to accelerate the search for the best paths and motion sequences to reach an objective. Lamarche and Donikian [LD04] used automatic topological model extraction of the environment for navigation. Another method, introduced by Kamphuis and Overmars [KO04], allows a group of agents to stay together while trying to reach

a goal. Although these approaches offer appealing results, they are not fast enough to simulate thousands of pedestrians in real time. Loscos *et al.* [LMM03] presented a behavioral model based on a 2D map of the environment. Their method is suited for simulating wandering crowds, but does not provide high level control on pedestrian goals. As introduced in Section 3.2 Pettré *et al.* [PdHCM*06, PGT07] presented a novel approach to automatically extract a topology from a scene geometry and handle path planning using a *navigation graph* (see Figure 34). The main advantage of this technique is that it handles uneven and multi-layered terrains. Nevertheless, it does not treat inter-pedestrian collision avoidance. Finally, Helbing *et al.* [HMS94, HFV00] used agent-based approaches to handle motion planning, but mainly focused on emergent crowd behaviors in particular scenarii.

Another approach for motion planning is inspired from fluid dynamics. Such techniques use a grid to discretize the environment into cells. Hughes [Hug02, Hug03] interpreted crowds as density fields to rule the motion planning of pedestrians. The resulting potential fields are dynamic, guiding pedestrians to their objective, while avoiding obstacles. Chenney [Che04] developed a model of flow tiles that ensures, under reasonable conditions, that agents do not require any form of collision detection at the expense of precluding any interaction between them. More recently, Treuille *et al.* [TCP06] proposed realistic motion planning for crowds. Their method produces a potential field that provides, for each pedestrian, the next suitable position in space (a *waypoint*) to avoid all obstacles. Compared to agent-based approaches, these techniques allow to simulate thousands of pedestrians in real time, and are also able to show emergent behaviors. However, they produce less believable results, because they require assumptions that prevent treating each pedestrian with individual characteristics. For instance, only a limited number of goals can be defined and assigned to groups of pedestrians. The resulting performance depends on the size of the grid cells and the number of groups.

The work presented in this Section introduces a new hybrid architecture offering a realistic and scalable solution for real-time crowd motion planning. Based on a navigation graph, we divide the environment into regions of varying interest. In regions of high interest, we exploit a potential field-based approach. Since we only use it locally, we can plan motion for many more groups and with finer grid cells than with an algorithm purely based on it. In other regions, motion planning is ruled by the navigation graph and short-term collision avoidance algorithms. Our local use of potential field-based approach allows us to plan motion for many more groups and with finer grid cells than with a purely potential field algorithm.

5.2. Motion Planning Architecture

The foundation of our motion planning architecture is based on navigation graphs, automatically extracted from the mesh of an arbitrary environment. This approach has the advantage of robustly handling path planning. Vertices represent cylindrical zones of the walkable space, while edges are the gates where pedestrians can cross the space from one vertex to another. To connect two distant vertices, it is possible to generate a *navigation flow*, composed of a set of varied paths. We show an example of such a flow in Figure 34. Thanks to this approach, pedestrian spreading is ensured. During simulation, pedestrians are assigned one navigation flow, and one direction. When they reach an extremity of the flow, they reverse their direction, and choose a new path, minimizing their travel time, *e.g.*, avoiding congested areas. Vertices offer a suitable structure of the walkable space. Indeed, they can be exploited to classify different regions of the scene. For instance, Pettré *et al.* [PdHCM*06, PGT07] used them to define several levels of simulation, each updated at different frequencies.

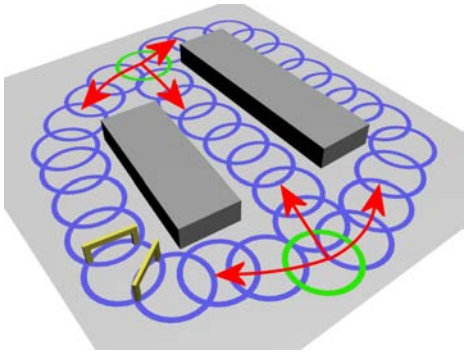


Figure 34: *A navigation graph composed of a single navigation flow (in blue) connecting two distant vertices (in green). The navigation flow is composed of three different paths that can be followed in either direction (red arrows). Two edges are also represented as gates (in yellow).*

The goal of our architecture is to handle thousands of pedestrians in real time. To achieve this result, we exploit the above mentioned vertex structure to divide the environment into regions ruled by different motion planning techniques. We classify these regions with a level of interest. The most interesting zones are ruled by realistic but expensive techniques, while others use simpler and faster solutions. Regions of interest (ROI) can be defined in any number and anywhere in the walkable space with high-level parameters. Moreover, it is possible to dynamically modify these parameters at runtime. Such flexibility is indeed desirable, because it allows the user to first choose the desired performance, and then distribute ROI, *i.e.*, computation time, as wished.

We observe that by defining only three different ROI, we obtain a simple and flexible architecture for realistic results:

- ROI 0 is composed of vertices of **high** interest.

- ROI 1 regroups vertices of **low** interest.
- ROI 2 contains all other vertices, of **no** interest.

Practically, we position the ROI with respect to the camera position and field of view. ROI 0 is directly in front of it, and/or in zones where important visible events occur. ROI 1 covers the remaining visible space, while ROI 2 includes all vertices outside the view frustum. Note that this choice is arbitrary, and our architecture is versatile enough to satisfy any other environment decomposition.

For regions of no interest (ROI 2), path planning is ruled by the navigation graph. Pedestrians use linear steering to follow the list of waypoints on their path edges. To use the minimal computation resources, obstacle avoidance is not handled.

Path planning in regions of low interest (ROI 1) is also ruled by the navigation graph. To steer pedestrians to their waypoints, an approach similar to Reynolds' is used [Rey99]. In these regions, for obstacle avoidance, an agent-based short-term algorithm (detailed in Section 5.3.4) is exploited. Although agent-based, this algorithm works at a low level, and thus stays simple and efficient.

In the regions of high interest (ROI 0), path planning and obstacle avoidance are both ruled by a potential field-based algorithm, similarly to Treuille *et al.* [TCP06]. Compared to agent-based approaches, potential fields are less expensive, and still offer results more realistic than the ones of ROI 1 and 2, because collision avoidance is planned in the long-term. Nevertheless, in certain situations, this approach fails to avoid collisions. To overcome this problem, the same short-term algorithm as in ROI 1 is also activated in ROI 0.

An important concern when dealing with regions ruled by different motion planning algorithms is to keep smooth and unnoticeable transitions at their borders. The way we place ROI implicitely solves this issue. Firstly, ROI 2 is always outside the view frustum, and thus does not require any specific attention. Secondly, passing the borders between ROI 0 and ROI 1 is always smooth, because they both use the same short-term avoidance algorithm.

5.3. Implementation

In this Section, the details of our hybrid architecture implementation are presented. We mainly focus on the initialization and runtime operations to construct and manage the scalable crowd motion planning. Firstly, in Section 5.3.1, the initialization phase is detailed, *i.e.*, the grid construction over the graph space, the initialization of the structure of neighbor cells and of the ROI. Then, we describe each step of the runtime pipeline, composed of five stages:

- Classification of graph vertices in correct ROI (Section 5.3.2).
- Potential field computation (Section 5.3.3).
- Short-term avoidance algorithm computation (Section 5.3.4).
- Pedestrian steering (Section 5.3.5).
- Continuity maintenance between grid and navigation graph (Section 5.3.6).

5.3.1. Initialization

First of all, for the given environment, a navigation graph is created, and navigation flows generated. We maintain a list of all active vertices, *i.e.*, of all vertices belonging to at least one path. The others are simply discarded, since no pedestrian will ever pass on them during simulation. Then, a grid is disposed on the scene, its size limited by the bounding rectangle containing all graph vertices. This grid is composed of an array of cells, each containing the link to its neighbor cells, and intrisic parameters used to compute the potential.

Many of the cells that compose the grid are not needed in the simulation, because they represent zones that are not covered by graph vertices, and thus indicate static obstacles. Moreover, some vertices are not used by any navigation flow, and thus are not exploited by pedestrians, as illustrated in Figure 35. Thus, we test whether each cell center is inside a vertex that composes a path. If not, the cell is deactivated. The main advantage of this preprocess is the reduction of the number of cells in which the potential field computation is necessary. Finally, each cell is linked only to its active neighbors.

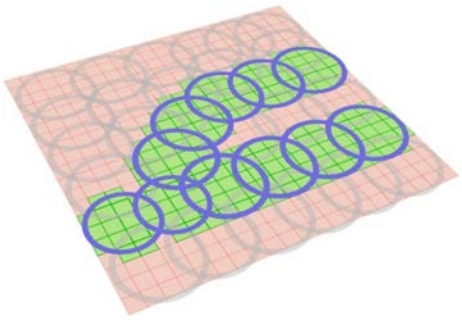


Figure 35: *The grid is placed on top of the graph, and only cells within a vertex that is part of a path stay active (in green).*

5.3.2. Classification of Graph Vertices in ROI

To define a ROI, the user specifies three parameters: a position, a radius, and a level of interest. All vertices whose center is contained whithin this region are assigned the specified level. These parameters can be modified at any moment, implying a re-classification of vertices.

In our practical use of ROI, we create three lists corresponding to our three levels of interest. At runtime, we first automatically detect vertices that are outside the view frustum, and insert them into the list with the lowest level of

interest (ROI 2). We then iterate over the remaining vertices, testing whether they are inside a ROI 0. If it is the case, the vertex is classified as of high interest and put in the corresponding list. Otherwise, it is put in the remaining list, of low interest (ROI 1). In the next two sections, we detail how pedestrian motions are planned in ROI 0 and 1.

5.3.3. Potential Field Computation

To accelerate the potential field computation, it is possible to group pedestrians, as suggested by Treuille *et al.* [TCP06]. In our case, pedestrians in ROI 0 having the same navigation flow and direction, *i.e.*, having the same goal, are grouped together. Thus, for each of these navigation flows, there are two groups. Groups are recomputed at each time step, to correctly classify pedestrians that change ROI.

Once this is achieved, for each group, a potential field is computed. At the goal, the potential is set to 0, and increased while spreading over the grid. Given the potential gradient, each pedestrian is assigned a new waypoint, corresponding to the center of a neighbor cell. The potential field computation itself is not further discussed (for details, see [TCP06]), but, taking advantage of our architecture, we introduce two techniques to reduce its computation time. First of all, we have observed no visual alteration when lowering the potential computation frequency to a reasonable value, as opposed to every time step. We thus have empirically set it to 5 Hz. Secondly, with our approach, the potential computation is only required in regions of high interest (ROI 0). These regions only cover part of the scene, and thus part of the grid. By computing the potential only for the cells located inside ROI 0, it is possible to drastically decrease computation time. However, goals are often outside these regions, and thus, it is impossible to initiate the potential computation. For each group, we therefore create subgoals, situated just outside ROI 0, as illustrated in Figure 36. We use the navigation flow structure to identify them: for every path of every flow leaving ROI 0, the first vertex met in the direction of the goal, is a subgoal. The potential computation is initiated in the central cell of every subgoal, and spread over all cells inside ROI 0 vertices. To obtain the same behavior as if the potential was computed all over the grid, we do not initiate the potential of the subgoal cells to 0, but approximate it. For each subgoal cell c inside vertex v_c, the potential ϕ_c is computed as:

$$\phi_c = C \cdot \sum_{v \in P(v_c)} (v.density + 1) \cdot v.radius \qquad (5)$$

Where v is a vertex of path $P(v_c)$, starting at v_c and leading to the final goal. The density of v is given by the number of pedestrians inside it per square meter. With Equation 5, the contribution to the potential of each vertex v is defined as its radius, weighted by its degree of occupation. To avoid having a null contribution from an empty vertex, we always add 1 to the computed density. Constant C is used to weight the sum so that values for ϕ_c are in the same range as if the potential was computed from the goal. Note that vertex v_c may be part of several paths at the same time. In this case, we compute Equation 5 for each path, and assign the lowest result to ϕ_c.

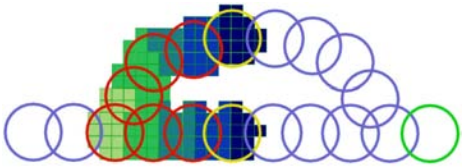


Figure 36: *Potential is computed for vertices in ROI 0 (in red) and vertices that have been identified as subgoals (in yellow). The final goal is displayed in green. Potential starts in the central cells of the subgoals with an approximated value.*

5.3.4. Short-Term Avoidance Algorithm

In this Section, we detail our short-term avoidance algorithm, which is a simplified low-level agent-based approach. It is used to efficiently avoid local inter-pedestrian collisions in both ROI 0 and 1. Particularly, in ROI 0, it complements the potential field approach, which may fail when the available space is too small and too crowded.

Algorithm 1 details step by step how we manage short-term avoidance. First of all, we need to find pedestrians that can potentially collide. To avoid an exhaustive search, we take advantage of the grid structure covering the whole environment: at runtime, every pedestrian in ROI 0 or 1 is registered in its current grid cell (line 3). This way, we can reduce the search for possible collisions to a small set of neighbor cells. Although this simplification does not cut down the order of complexity in $O(n^2)$, it significantly decreases n, as compared to a brute force approach [Rey87]. To keep the algorithm fast, the two steps mentioned above are alternated during simulation (line 1) : we first register the pedestrians to their cell at one time step, while the search for potential collisions and their avoidance is achieved at the next step (line 4). Given the low distance covered by a pedestrian in such a short time lapse, the algorithm robustness is guaranteed.

The avoidance itself is based on two values: a distance of *security* α, fixed at 2 m, and a distance of *emergency* β, at 0.5 m. For each pedestrian p in ROI 0 or 1, we start by searching for its neighbor cells in an area of radius α (line 6). Then, for each pedestrian $p_{neighbor}$ contained in a neighbor cell, we test the angle between the heading direction of p and its distance vector to $p_{neighbor}$. If this angle is too small, the current waypoint of p is rotated away from its neighbor (line 11). An illustration of this situation is shown in Figure 37. To make sure the pedestrian still reaches its goal, note that the waypoint is set back to its original position at

the next time step. There are still some cases when this approach fails, *e.g.*, in overcrowded places. If p and $p_{neighbor}$ are at an emergency distance (line 8), p is gently slided aside its neighbor.

Data: set of pedestrians in $\{ROI\ 0 \cup ROI\ 1\}$, set of grid cells, security distance α, emergency distance β
Result: updated set of pedestrians in $\{ROI\ 0 \cup ROI\ 1\}$
if *isEven(frameNumber)* **then**
  for *each pedestrian* $p \in \{ROI\,0 \cup ROI\,1\}$ **do**
    register p in its current cell c_p
  end
end
else
  for *each pedestrian* p *in cell* c_p **do**
    $Set_{neighbors} = findNeighbors(c_p, \alpha)$
    for *each pedestrian* $p_{neighbor}$ *in* $Set_{neighbors}$ **do**
      if $distance(p, p_{neighbor}) < \beta$ **then**
        slide p away from $p_{neighbor}$
      end
      else if $angle(p, p_{neighbor}) \in [-\frac{\pi}{4}, \frac{\pi}{4}]$ **then**
        $rotateWaypoint(p, p_{neighbor})$
      end
    end
  end
end

Algorithm 1: Short-term avoidance algorithm.

Figure 37: *Two pedestrians are closer than the security distance. An angle γ is computed between the first pedestrian's heading vector (in blue) and the two characters distance vector (in red).* ***(left)*** *The angle γ is in the range $[-\frac{\pi}{4}, \frac{\pi}{4}]$ (green zone) and a collision avoidance is attempted by rotating the first pedestrian's waypoint.* ***(right)*** *The second character is outside the green zone, and no avoidance procedure is yet required.*

5.3.5. Steering

Both navigation graph and potential field approaches provide waypoints toward which pedestrians have to move. A smooth steering algorithm is necessary to obtain a fluid movement toward these points. The seek behavior of Reynolds [Rey99] has the advantage of producing a believable steering toward a target point in space. We use this steering model for pedestrians of both ROI 0 and 1. For ROI 2, a linear steering is employed.

5.3.6. Continuity Maintenance

In our motion planning architecture, we work with two approaches based on different spaces: a navigation graph defined by its vertices and edges, and a grid composed of cells. This duality brings up two issues when switching from one space to the other. More precisely, when a pedestrian passes from ROI 0 to ROI 1.

The first issue arises when a pedestrian enters the active grid space (ROI 0). Its position is then only updated in the grid, but no longer in the graph. It implies that this character stays registered in the same vertex while progressing in the grid. Thus, its next waypoint on the graph also remains the same. When the pedestrian eventually exits ROI 0, it turns back to meet the graph waypoint it has long since passed. To avoid this problem, we keep updating the pedestrian position in the graph, even in ROI 0: if a pedestrian enters this region, we keep track of its distance to its next graph waypoint. When the distance is under a given threshold, the pedestrian is registered in the next vertex.

The second issue occurs when two or more paths of the same navigation flow are present in ROI 0. Since path planning in that area is ruled by the potential field, a pedestrian chooses the path where the potential is the lowest, as in Figure 38 (right). However, this path does not necessarily correspond to the one it is registered to in the graph (Figure 38 (left)). In the worst case, the pedestrian becomes completely lost when exiting ROI 0: it is within a vertex that does not belong to the path it should follow. To solve this problem, when any pedestrian exits ROI 0, we test whether it still is on the same graph path. If not, we look for a new path using this vertex and register the pedestrian to it.

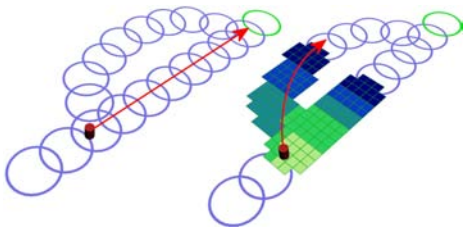


Figure 38: *(left) In graph space, the path followed by the pedestrian is the right one. (right) In grid space, the potential field is lower on the left path. High potential is represented in light green and low potential in dark blue.*

5.4. Performance Tests

We have run several tests in different crowded environments with an Athlon64 4000+, with 2 *GB* of memory and two NVidia 6800 ultra in SLI mode. For all tests, pedestrians are represented with two human templates using several

textures, and exploiting color variety techniques. They are rendered as impostors, and use a walk animation, sampled at a frequency of 20 Hz. Note that in all the following tests, we observe interesting emergent behaviors, *e.g.*, lane formations or panic effects, that make the crowd motion planning more realistic.

We use a first environment, representing a city pedestrian area, to test the performance of our motion planning architecture, compared with our implementation of the purely potential field-based approach of Treuille *et al.* [TCP06]. In this scene, the camera position is fixed at a predefined position. For our tests, we define three regions in the environment. The one with the highest level of interest (ROI 0) has a radius of 15 m, and is static, positioned at the center of the scene. Note that we have voluntarily set this region in the center of the scene, where the eye is naturally attracted, rather than in front of the camera. The remaining space inside the view frustum is of low interest (ROI 1), and the other zones are classified in ROI 2. We have tested the efficiency of both approaches with cells of $3 \times 3\ m^2$, and an increasing number of pedestrians and groups, starting from 2 groups and 200 pedestrians up to 12 groups, totaling $1,200$ characters. Figure 39 shows the results of this comparison. We can see that the performance of our approach logically decreases with the increasing number of groups, but much more slowly than with the purely potential field-based approach. There are two reasons. Firstly, our technique only computes the potential field in a limited region of high interest (ROI 0). Secondly, only a subset of the total number of groups passes in this region, minimizing the number of potential fields to compute. This test has also been performed with the ROI 0 dynamically moving on the city place. Even so, the obtained results remain similar to those illustrated in Figure 39.

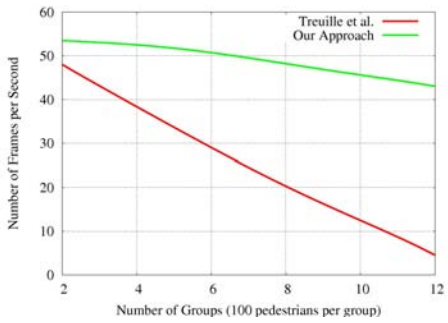


Figure 39: *Comparison between our approach and our implementation of the purely potential field-based approach of Treuille et al. [TCP06] for a varying number of groups. Each group is composed of a hundred pedestrians.*

Our second test is achieved with a crowd of $10,000$ pedestrians in a large scene with 12 navigation flows, *i.e.*, 24 groups, spread over the whole environment, as demonstrated in Figure 40. For this scenario, the different regions of interest are placed according to the camera position. If the camera moves, the regions are also displaced. The cell size is set to $4 \times 4\ m^2$, and the obtained performance is of about 20 fps.

For our third scenario, we use the same city pedestrian area as in the first test, but extend it with several surrounding streets and buildings. There are $5,000$ pedestrians and some cars navigate on the roads. We illustrate this scenario in Figure 41. Each cell of the grid covers a $3 \times 3\ m^2$ area. Since the user attention is mainly drawn by the cars, which threaten to hit pedestrians at every moment, a region of high interest (ROI 0) is set around each of them. Moreover, to make pedestrians flee the potential collision, a high discomfort and speed increase are set in front of the cars, as in [TCP06]. As a result, pedestrians close to a car are always in a region of high interest, and thus ruled by a potential field. In front of cars particularly, the pedestrians flee the zone of danger, demonstrating an emergent panic behavior. The remaining visible environment is classified as a region of low interest (ROI 1), so that pedestrians still take care to avoid each other. Finally, the zone outside the view frustum is set as of no interest (ROI 2). The resulting fps varies between 15 and 30, depending on the number of visible cars (1 to 3), and the size of their surrounding ROI 0, (10 to 15 m radius).

Figure 40: $10,000$ *pedestrians planning their motion in a large landscape of fields. There are* 12 *navigation flows and the cell size is set to* $4 \times 4\ m^2$.

Finally, we have tested the evolution of the frame rate with a fixed number of groups and an increasing number of pedestrians. The test has been achieved in a large scene with 24 groups, a cell size of $3 \times 3\ m^2$, and 1 to 5 regions of high interest, distributed over the scene. Each of them has a fixed radius of 15 m. For the remaining of the scene, ROI 2 is not exploited; all vertices are classified as ROI 1. During the test, the rendering of the scene and pedestrians was deactivated to analyze the sole motion planning cost. The results, in Figure 42, show that even with 5 different regions of interest, our architecture still manages the motion planning of $10,000$ pedestrians at interactive frame-rates (between 10 and 15 fps). Note that the increasing number of pedestrians

does not as much influence the potential computation, which is more sensible to the number of groups, than the short-term avoidance, which has a complexity in $O(n^2)$.

Figure 41: *A city scene where pedestrians avoid a car surrounded by a ROI 0.*

All tests show that our motion planning architecture offers high performance for large crowd motion planning. The possibility to select and distribute regions of interest as wished gives the opportunity to easily tune the simulation for the desired frame rate, and to define many groups, *i.e.*, many different goals. Additionally, compared to a purely potential-field based technique, much smaller cells can be used for obtaining better visual results in long-term avoidance cases.

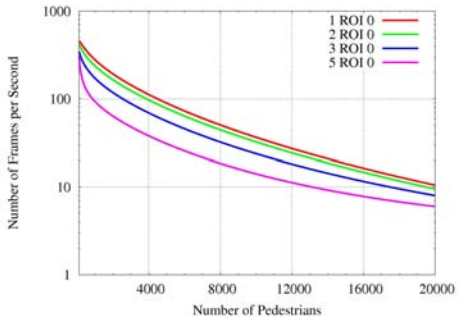


Figure 42: *Performance obtained for a number of groups fixed to* 24 *and an increasing number of pedestrians, without rendering.* 1 *to* 5 *ROI 0 with a radius of* 15 *m each are placed in the scene, while the remaining space is entirely in ROI 1.*

5.5. Limitations

There are some limitations to our motion planning architecture. Firstly, in too crowded narrow environments, severe bottlenecks may appear, making the use of our potential field-based approach a waste of computational time. However, it is possible to force such regions to always keep a predefined lower level of interest, *e.g.*, ruled by a short-term avoidance algorithm. Another limitation is the use of a group-based approach. Indeed, we are constrained to assign general goals for groups of pedestrians. Assigning one different goal to each pedestrian would be too prohibitive for real-time applications. Yet, we note that our architecture is able to handle many more groups than previous potential field-based methods. This is mainly due to our massive reduction of the number of cells in which the potential actually needs to be computed, and implies the possibility to refine the grid for more accurate results.

6. Conclusion

In this tutorial, we have detailed the numerous aspects that need to be taken into account when simulating crowds in real time. We have shown how to efficiently exploit the computational time with a complete description of our architecture and pipeline. In order to obtain a large variety of characters, we have described several techniques, fast and robust, based on the use of a limited number of human templates. Means to obtain variety in animation have also been introduced, while our hybrid scalable motion planning algorithm has been thoroughly detailed. Tests and results have also been presented to estimate the achieved performance for different parts of the complete architecture.

7. Acknowledgements

We would like to thank Mireille Clavien for her great job on designing virtual humans and creating several figures of this tutorial. We acknowledge Antoine Schmid and Ronan Boulic for their work on accessory movements, Fiorenzo Morini for his work on motion planning. This research was sponsored by the Swiss National Research Foundation.

References

[BB04] BAERLOCHER P., BOULIC R.: An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. *Vis. Comput. 20*, 6 (2004), 402–417.

[BLA03] BAYAZIT O. B., LIEN J.-M., AMATO N. M.: Better group behaviors in complex environments using global roadmaps. In *ICAL 2003: Proceedings of the eighth international conference on Artificial life* (Cambridge, MA, USA, 2003), MIT Press, pp. 362–370.

[Che04] CHENNEY S.: Flow tiles. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2004), Eurographics Association, pp. 233–242.

[Coo86] COOK R. L.: Stochastic sampling in computer graphics. *ACM Trans. Graph. 5*, 1 (1986), 51–72.

[dHCSM*05] DE HERAS CIECHOMSKI P., SCHERTENLEIB S., MAÏM J., MAUPU D., THALMANN D.: Real-time Shader Rendering for Crowds in Virtual Heritage. In *VAST '05, 2005* (2005).

[DHOO05] DOBBYN S., HAMILL J., O'CONOR K., O'SULLIVAN C.: Geopostors: a real-time geometry / impostor crowd rendering system. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2005), ACM Press, pp. 95–102.

[EGMT06] EGGES A., GIACOMO T. D., MAGNENAT-THALMANN N.: Synthesis of realistic idle motion for interactive characters. In *Game Programming Gems 6* (2006).

[GBT04a] GLARDON P., BOULIC R., THALMANN D.: A coherent locomotion engine extrapolating beyond experimental data. In *Proc. of Computer Animation and Social Agent* (2004).

[GBT04b] GLARDON P., BOULIC R., THALMANN D.: Pca-based walking engine using motion capture data. In *Proc. of Computer Graphics International* (2004).

[HFV00] HELBING D., FARKAS I., VICSEK T.: Simulating dynamical features of escape panic. *Nature 407*, 6803 (September 2000), 487–490.

[HLTC03] HEIGEAS L., LUCIANI A., THOLLOT J., CASTAGNÉ N.: A physically-based particle model of emergent crowd behaviors. In *Graphicon* (2003).

[HMS94] HELBING D., MOLNÁR P., SCHWEITZER F.: Computer simulations of pedestrian dynamics and trail formation. *Evolution of Natural Structures* (1994), 229–234.

[Hug02] HUGHES R.: A continuum theory for the flow of pedestrians. *Transportation Research Part B: Methodological 36* (July 2002), 507–535(29).

[Hug03] HUGHES R. L.: The flow of human crowds. *Annual Review of Fluid Mechanics 35*, 1 (2003), 169–182.

[KO04] KAMPHUIS A., OVERMARS M. H.: Finding paths for coherent groups using clearance. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2004), Eurographics Association, pp. 19–28.

[KS01] KIRCHNER A., SHADSCHNEIDER A.: Simulation of evacuation processes using a bionics-inspired cellular automaton model for pedestrian dynamics. In *Physica A* (2001), pp. 237–244.

[LD04] LAMARCHE F., DONIKIAN S.: Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum 23*, 3 (2004), 509–518.

[LK06] LAU M., KUFFNER J. J.: Precomputed search trees: Planning for interactive goal-driven animation. In *2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (Sept. 2006), pp. 299–308.

[LMM03] LOSCOS C., MARCHAL D., MEYER A.: Intuitive crowd behaviour in dense urban environments using local laws. In *TPCG '03: Proceedings of the Theory and Practice of Computer Graphics 2003* (Washington, DC, USA, 2003), IEEE Computer Society, p. 122.

[LWC*02] LUEBKE D., WATSON B., COHEN J. D., REDDY M., VARSHNEY A.: *Level of Detail for 3D Graphics*. Elsevier Science Inc., New York, NY, USA, 2002.

[Mau05] MAUPU D.: Creating variety - a crowd creator tool, 2005.

[MH99] MÖLLER T., HAINES E.: *Real-time rendering*. A. K. Peters, Ltd., Natick, MA, USA, 1999.

[MH03] METOYER R. A., HODGINS J. K.: Reactive pedestrian path following from examples. In *CASA '03: Proceedings of the 16th International Conference on Computer Animation and Social Agents (CASA 2003)* (Washington, DC, USA, 2003), IEEE Computer Society, p. 149.

[MR06] MILLAN E., RUDOMIN I.: Impostors and pseudo-instancing for gpu crowd rendering. In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia* (New York, NY, USA, 2006), ACM Press, pp. 49–55.

[nvi06] NVIDIA: Nvidia geforce 8800 gpu architecture overview, 2006.

[PdHCM*06] PETTRÉ J., DE HERAS CIECHOMSKI P., MAÏM J., YERSIN B., LAUMOND J.-P., THALMANN D.: Real-time navigating crowds: scalable simulation and rendering. *Journal of Visualization and Computer Animation 17*, 3-4 (2006), 445–455.

[PGT07] PETTRÉ J., GRILLON H., THALMANN D.: Crowds of moving objects: Navigation planning and simulation. In *Proceedings of IEEE International Conference on Robotics and Automation* (2007).

[Rey87] REYNOLDS C. W.: Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), ACM Press, pp. 25–34.

[Rey99] REYNOLDS C.: Steering behaviors for autonomous characters, 1999.

[Rey06] REYNOLDS C.: Big fast crowds on ps3. In *sandbox '06: Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames* (New York, NY, USA, 2006), ACM Press, pp. 113–121.

[RSC87] REEVES W. T., SALESIN D. H., COOK R. L.:

Rendering antialiased shadows with depth maps. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), ACM Press, pp. 283–291.

[SKG05] SUNG M., KOVAR L., GLEICHER M.: Fast and accurate goal-directed motion synthesis for crowds. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2005), ACM Press, pp. 291–300.

[Smi78] SMITH A. R.: Color gamut transform pairs. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1978), ACM Press, pp. 12–19.

[TCP06] TREUILLE A., COOPER S., POPOVIC; Z.: Continuum crowds. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), ACM Press, pp. 1160–1168.

[TLC02] TECCHIA F., LOSCOS C., CHRYSANTHOU Y.: Image-based crowd rendering. *IEEE Computer Graphics and Applications 22*, 2 (2002), 36–43.

[Ura05] URALSKY Y.: Efficient soft-edged shadows using pixel shader branching. In *GPU Gems 2* (2005).

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1978), ACM Press, pp. 270–274.

Populating Virtual Environments with Crowds: Level of Detail for Real-Time Crowds

S. Dobbyn and C. O'Sullivan

Graphics, Vision and Visualisation (GV2) Lab, Trinity College Dublin, Ireland

Abstract

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.

1. Introduction

This part of the tutorial describes the main research related to the real-time visualisation and animation 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.
- Then, 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.

2. Character Visualisation

2.1. 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 several hundred 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. Additionally, 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 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 1: *Simple Texturing-Mapping: (a) Mesh without texture-mapping, (b) Texture Map (c) Texture-mapped mesh.*

2.2. 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). Environment mapping [BN76] can be used to simulate an object reflecting its environment. For characters such as soldiers wearing shiny

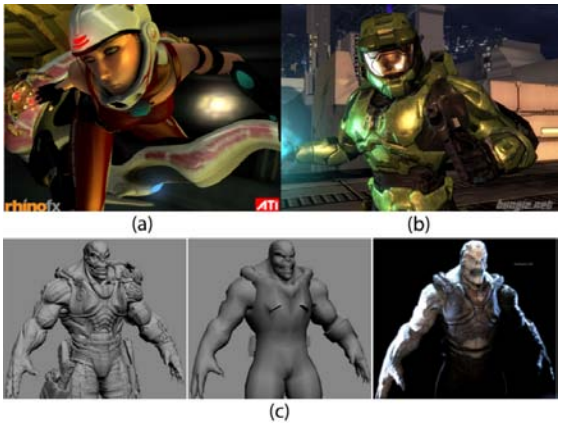


Figure 2: *Per-pixel lighting effects such as environment mapping in (a) Ruby Demo ((© ATI Technologies) and (b) Halo 2 (© 2004 Microsoft Corporation), and (c) Normal mapping in Unreal Engine 2003 (© 2005 Epic Games Inc).*

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 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.

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 ren-

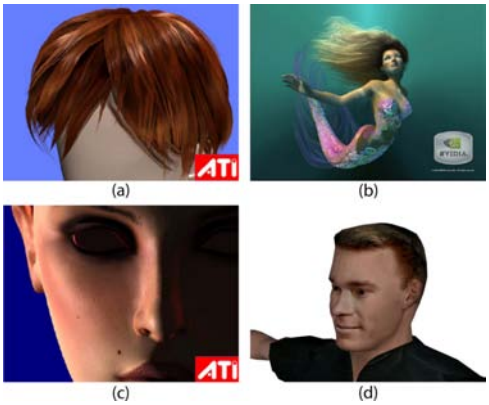


Figure 3: *Real-time hair rendering based on the light scattering of human hair fibres [MJC*03] and a fur rendering model [KK89] using a (a) polygonal model [Sch04] (b) a particle system [Wlo04]. (c) Real-time skin rendering based on subsurface scattering [SGM04]. (d) Hair and skin rendered with simple texturing.*

dering 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 3).

2.3. 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.

2.3.1. 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.

2.3.2. 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].

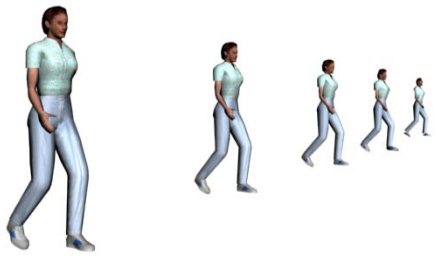


Figure 4: *Five discreet mesh models containing (a) 2,170 (b) 1,258 (c) 937 (d) 612 and (e) 298 triangles.*

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 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.

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

Figure 5: *Rendering crowds using a discrete LOD approach [dHCUCT04].*

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 6).

Figure 6: *Geometric-based representations rendered with various per-pixel shading effects [GSM05] (© 2005 ATI Technologies).*

While Gosselin et al. provide techniques to improve the rendering performance of multiple character meshes, the

crowd is homogeneous in nature since it is made of individuals that are using the same template model and are animated with the same running motion. Recently, Dudash et al. [Dud07] has extended this work and provided an efficient way of adding variation to the crowd's animation and appearance. Their approach involves using instancing through the DirectX 10 API in an attempt to reduce the number of draw calls, state changes and buffer updates.

They achieve mesh variation by breaking a character into a collection of mesh sub-sections. For each character, each sub-section is initialised with an alternate mesh randomly selected from multiple template meshes at load time. At run-time, they make a list for each sub-mesh containing the per-instance data of the characters that are using that particular piece and then draw each instance. To provide for a more heterogeneous crowd animation, they avoid the usual technique of using the limited number of shader constants for the animation data. Instead, they encode all the animations' frame data into a texture, from which the vertex shader looks up the bone matrices for each character's current frame of animation. In this way, their characters can perform any frame of any animation defined in the texture. Additionally, they implement a discrete LOD system where characters in the distance use mesh sub-sections of lower resolution.

2.3.3. Image-Based Crowd Rendering

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. Consequently, due to this planar impostor 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 [DSSD99], billboard clouds [DDSD03], and texture depth images [JW02] for rendering acceleration of various applications. A survey of these different types, including their application and their advantages and disadvantages, can be found in [JWP05]. To represent a virtual human, Tecchia et al. [TC00] 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 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.

Figure 7: *Image-based crowds: (a) Dynamically generated image-based crowds [ABT00] (b) Pre-generated image-based crowds [TC00].*

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 3, 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. [TC00] 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 animated 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 refer-

ence 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. [TLC02] 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.

Dobbyn et al. [DHOO05] developed the Geopostor system, which provides for a hybrid combination of pre-generated impostor and detailed geometric rendering techniques for virtual humans. By switching between the two representations, based on a "pixel to texel" ratio, their system allows visual quality and performance to be balanced. They 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 (see Figure 8).

Figure 8: *Geopostor system.*

Recently, Millán et al. [MR06b] described a LOD system which takes advantage of existing programmable graphics hardware in order to improve the simulation and rendering performance of their crowd system. They simulate several hundred thousand characters in real-time by storing each character's position and orientation in a pixel buffer which is updated by a fragment program. Once the pixel buffer is updated, this data is subsequently used by graphics hardware to render the characters using a particular representation selected based on a LOD map. The LOD map is a 2D grid rendered on a per-frame basis, where each pixel represents a position in the world and stores a specific encoded value representing its distance from the camera. Once the LOD map is generated, it is stored in a pixel buffer which is looked up by the vertex processor to select a LOD representation for each character instance. Similar to Dobbyn et al. [DHOO05], they use a geometry/impostor based LOD system. However, they use the vertex processor to rotate each impostor's billboard towards the camera view and calculate the texture coordinates of the most suitable viewpoint to be mapped.

2.3.4. 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.

Figure 9: *Point-based crowds: (a) Stadium populated with animated 16,000 fans and (b) Crowd of 90,000 humans walking on the spot.*

For smaller crowds, consisting of several thousands of objects, each object is represented 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 9), 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. For a comparison between point-based models and impostors see [MR06a].

3. 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.

3.1. 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 1). When calculating the deformation of the normals, only the rotational component is used by getting the inverse transpose of the global transformation matrix ($(TM_i^{-1})^T$).

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

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].

3.2. 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 animations 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.

3.2.1. 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.

3.2.2. 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 algorithms 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].

3.2.3. 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.

3.3. 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 2.3.2) 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.

Ahn et al. [AOW06] detail a motion simplification framework for articulated characters which attempts to speed up animation performance through posture simplification whilst conserving the features of the original motion. The framework involves extracting key postures from a motion and then automatically generating the priority of joint reductions in order to guide the posture simplification process. The experimental results shows that the proposed motion simplification can be successfully applied to a crowd of a thousand articulated characters in real-time.

3.4. 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. [UT02] 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.

References

[ABC96] AMAYA K., BRUDERLIN A., CALVERT T.: Emotion from motion. *GI '96: Proceedings of the Conference on Graphics Interface* (1996), 222–229.

[ABT00] AUBEL A., BOULIC R., THALMANN D.: Real-time display of virtual humans: Levels of details and impostors. *IEEE Transactions on Circuits and Systems for Video Technology 10*, 2 (2000), 207–217.

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

[AOW06] AHN J., OH S., WOHN K.: Optimized motion simplification for crowd animation. *Computer Animation and Virtual Worlds 17*, 3-4 (2006), 155–165.

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

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

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

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

[COCSD03] COHEN-OR D., CHRYSANTHOU Y., SILVA C. T., DURAND F.: A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics 9*, 3 (2003), 412–431.

[COM98] COHEN J., OLANO M., MANOCHA D.: Appearance-preserving simplification. *SIGGRAPH '98: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive techniques* (1998), 115–122.

[DDSD03] DÉCORET X., DURAND F., SILLION F., DORSEY J.: Billboard clouds for extreme model simplification. *ACM Transactions on Graphics (TOG) 22*, 3 (2003), 689–696.

[dHCSMT05] DE HERAS CIECHOMSKI P., SCHERTENLEIB S., MAÏM J., THALMANN D.: 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* (2005), 601–610.

[dHCUCT04] DE HERAS CIECHOMSKI P., ULICNY B., CETRE R., THALMANN D.: 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* (2004), 9–17.

[DHOO05] DOBBYN S., HAMILL J., O'CONOR K., O'SULLIVAN C.: Geopostors: A real-time geometry / impostor crowd rendering system. *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games* (April 2005), 95–102.

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

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

[DSSD99] DÉCORET X., SCHAUFLER G., SILLION F., DORSEY J.: Multi-layered impostors for accelerated rendering. *Computer Graphics Forum (Proceedings of Eurographics '99) 18*, 3 (1999), 61–73.

[Dud07] DUDASH B.: Skinned instancing, NVIDIA Corporation. *http://developer.download.nvidia.com/SDK//direct3d/samples.html* (2007).

[EDD*95] ECK M., DEROSE T., DUCHAMP T., HOPPE H., LOUNSBERG M., STUETZLE W.: Multiresolution analysis for arbitrary meshes. *SIGGRAPH '95: Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques* (1995), 173–182.

[FS93] FUNKHOUSER T. A., SÉQUIN C. H.: 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* (1993), 247–254.

[FST92] FUNKHOUSER T. A., SÉQUIN C. H., TELLER

S.: Management of large amounts of data in interactive building walkthroughs. *SI3D '92: Proceedings of the 1992 symposium on Interactive 3D graphics* (1992), 11–20.

[GMPO00] GIANG T., MOONEY R., PETERS C., O'SULLIVAN C.: Aloha: Adaptive level of detail for human animation towards a new framework. *Eurographics 2000 Short Paper Programme* (2000), 71–77.

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

[GSM05] GOSSELIN D., SANDER P., MITCHELL J.: *ShaderX3 - Drawing a Crowd*. Charles River Media, 2005, pp. 505–517.

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

[HWBO95] HODGINS J. K., WOOTEN W. L., BROGAN D. C., O'BRIEN J. F.: Animating human athletes. *SIGGRAPH '95: Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques* (1995), 71–78.

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

[JWP05] JESCHKE S., WIMMER M., PURGATHOFER W.: Image-based representations for accelerated rendering of complex scenes. *In Eurographics 2005 STAR Reports* (2005), 1–20.

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. *ACM Transactions on Graphics (TOG) 21*, 3 (2002), 473–482.

[Kil00] KILGARD M. J.: *A Practical and Robust Bump-mapping Technique for Today's GPUs*. Tech. rep., NVIDIA Corporation, 2000.

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

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

[Lan98] LANDER J.: Skin them bones. *Game Developer Magazine* (May 1998), 11Ű–16.

[Lan99] LANDER J.: Over my dead, polygonal body. *Game Developer Magazine* (October 1999), 17Ű–22.

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

[LW85] LEVOY M., WHITTED T.: *The Use of Points as a Display Primitive*. Tech. rep., University of North Carolina at Chapel Hill, 1985.

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

[Map] MAPPER A. N.:. http://www.ati.com.

[MJC*03] MARSCHNER S. R., JENSEN H. W., CAMMARANO M., WORLEY S., HANRAHAN P.: Light scattering from human hair fibers. *ACM Transactions on Graphics (TOG) 22*, 3 (2003), 780–791.

[MR06a] MILLÁN E., RUDOMÍN I.: A comparison between impostors and point-based models for interactive rendering of animated models. In *In Proceedings International Conference in Computer Animation and Social Agents (CASA) 2006* (2006).

[MR06b] MILLÁN E., RUDOMÍN I.: Impostors and pseudo-instancing for gpu crowd rendering. In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia* (2006), pp. 49–55.

[MS95] MACIEL P., SHIRLEY P.: Visual navigation of large environments using textured cluster. *SI3D '95: Proceedings of the 1995 Symposium on Interactive 3D Graphics* (1995), 95–102.

[OCV*02] O'SULLIVAN C., CASSELL J., VILHJÁLMSSON H., DINGLIANA J., DOBBYN S., MCNAMEE B., PETERS C., GIANG T.: Levels of detail for crowds and groups. *Computer Graphics Forum 21*, 4 (2002), 733–742.

[Ope04] OPENGL S. G. I.: The OpenGL Graphics System: A Specification. *http://www.opengl.org /documentation/specs/version2.0/glspec20.pdf* (October 2004).

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

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

[SBMTT99] SANNIER G., BALCISOY S., MAGNENAT-THALMANN N., THALMANN D.: Vhd:a system for directing real-time virtual actors. *The Visual Computer 15*, 7/8 (1999), 320–329.

[Sch04] SCHEUERMANN T.: Practical real-time hair rendering and shading. *SIGGRAPH 2004 Sketch* (2004).

[Sco03] SCOTT R.: Sparking life: notes on the performance capture sessions for the *Lord of the Rings: the Two*

Towers. *SIGGRAPH Computer Graphics 37*, 4 (2003), 17–21.

[SGM04] SANDER P., GOSSELIN D., MITCHEL J.: Real-time skin rendering on graphics hardware. *SIGGRAPH 2004 Sketch* (2004).

[SKvW*92] SEGAL M., KOROBKIN C., VAN WIDENFELT R., FORAN J., HAEBERLI P.: Fast shadows and lighting effects using texture mapping. *SIGGRAPH '92: Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques 26*, 2 (1992), 249–252.

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

[TC00] TECCHIA F., CHRYSANTHOU Y.: Real-time rendering of densely populated urban environments. *Proceedings of the Eurographics Workshop on Rendering Techniques* (2000), 83–88.

[TLC02] TECCHIA F., LOSCOS C., CHRYSANTHOU Y.: Visualizing crowds in real-time. *Computer Graphics Forum 21*, 4 (2002), 753–765.

[TT94] TU X., TERZOPOULOS D.: Artificial fishes: Physics, locomotion, perception, behavior. *SIGGRAPH '94: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques* (1994), 43–50.

[UdHCT04] ULICNY B., DE HERAS CIECHOMSKI P., THALMANN D.: Crowdbrush: Interactive authoring of real-time crowd scenes. *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/EUROGRAPHICS Symposium on Computer Animation* (2004), 243–252.

[UT02] ULICNY B., THALMANN D.: Towards interactive real-time crowd behaviour simulation. *Computer Graphics Forum 21*, 4 (2002), 767–775.

[Web00] WEBER J.: Run-time skin deformation. *In Proceedings of Game Developers Conference* (2000).

[Wlo04] WLOKA M.: Advanced rendering techniques. *EUROGRAPHICS 2004 Tutorial* (2004). http://developer.nvidia.com/object/eg_2004_presentations.html.

[WS02] WAND M., STRASSER W.: Multi-resolution rendering of complex animated scenes. *Computer Graphics Forum 21*, 3 (2002), 483–491.

Optimising and Evaluating the Realism of Virtual Crowds: Perceptual Experiments and Metrics

R. McDonnell, S. Dobbyn, and C. O'Sullivan

Graphics, Vision & Visualisation Lab (GV2), Trinity College Dublin, Ireland

Abstract

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), only recently have researchers become interested in perceptually evaluating them. 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 to improve 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.

Introduction

While there has been little previous work related to the perception of virtual human representations [HMDO05, MDO05], research has been conducted on perception of human motion in the context of computer graphics and has mainly been focused on the effect of animation quality on user perception. Wang et al. [WB03] conducted a set of experiments to evaluate a cost function proposed by Lee et al. [LCR02] for determining the transition quality between motion clips. Other recent work by Harrison et al. [HRvdP04] examined the perceptual impact of dynamic anomalies in human animation. Reitsma and Pollard [RP03] conducted a study, developing a metric to evaluate the perceived error introduced during motion editing. Harrison et al. [HBF02] focused on higher-level techniques for specifying and modifying human motions. Oesker et al. [OHJ00] investigated the extent to which observers perceptually process the LOD in naturalistic character animation. The study most related to our work is by Hodgins et al. [HOT98]. They performed a series of perceptual experiments, the results of which indicated that a viewer's perception of motion characteristics is affected by the geometric model used for rendering. Participants were shown a series of paired motion sequences and asked if the two motions in each pair were the "same" or "different". The motion sequences in each pair were rendered using the same geometric model. For the three types of motion variation tested, sensitivity scores indicated that subjects were better able to observe changes when viewing the polygonal model than they were with a stick figure model.

With the goal of improving the realism of our crowd system, we carried out the following four sets of perceptual experiments:

1. **Perception of Human Appearance**
   Experiment 1: Impostor Vs. Mesh Detection
   At what distance can experiment participants detect that a virtual human is using an impostor or mesh representation?
   Experiment 2: Low Vs. High-Resolution Mesh Discrimination
   At what distance and at what resolution can experiment participants discriminate between a high resolution and low resolution mesh representation?
   Experiment 3: Impostor/Mesh Switching Discrimination
   At what distance can experiment participants detect an impostor switching to a mesh?

2. **Perception of Motion**
   Experiment 4: Perception of Human Motion
   How well do different virtual human representations replicate motion?
   Experiment 5: Perception of Cloth Motion
   How well do different representations replicate deformable clothing?

Experiment 6: Applying Motion Capture
Is the sex of a motion captured actor important when applying his/her motion to virtual characters?

3. **Perceptual Metrics for Smooth Animation**
Experiment 7: Impostor Update Frequency
What is the optimal sampling rate (i.e, the number of viewpoints) for impostors?
Experiment 8: Animation Update Frequency
How many pose changes per second are needed for smooth animation?
Experiment 9: Simulation Level of Detail
Can we save memory by displaying foreground characters at higher update rates than background characters, with no loss of visual fidelity?

4. **Evaluation of Metrics**
Experiment 10: Impostor Update and Mesh Detection Metric Evaluation
To evaluate the effectiveness of the metrics discovered in Experiment 1, 3 and 7 by placing the characters in crowds of different sizes.

Psychophysics

We will begin with a broad overview of some psychophysical techniques that were used for gathering information for the experiments that we performed.
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 analysed 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 1) can be use to plot a curve that fits the participant's data (shown in Figure 1) and, from this curve, the PSE can be estimated as the 50% point and calculated using Equation 2. The inverse normal distribution function computes the stimulus intensity (x) for a given probability (P).

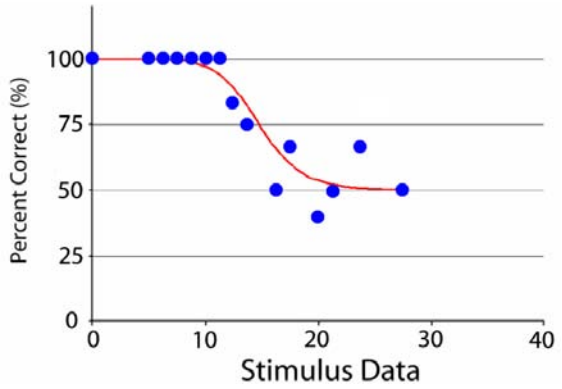


Figure 1: *An Ogive function fitted to a participant's data for a yes-no design.*

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

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

$$PSE_{Ogive} = P_{Ogive}(0.5) \quad (2)$$

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 3). The PSE for an experiment using a 2AFC design can be calculated using Equation 4.

$$P_{Logistic}(x) = 1 - \gamma\left(\frac{1}{1+(\frac{x}{\alpha})^{-\beta}}\right) \quad (3)$$

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.

$$PSE_{Logistic} = P_{Logistic}(0.75) \quad (4)$$

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 and Equation 6 are used to calculate the JND for an 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.

$$JND_{Ogive} = P_{Ogive}(0.75) - P_{Ogive}(0.5) \quad (5)$$

$$JND_{Logistic} = P_{Logistic}(0.875) - P_{Logistic}(0.75) \quad (6)$$

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 2 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.

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

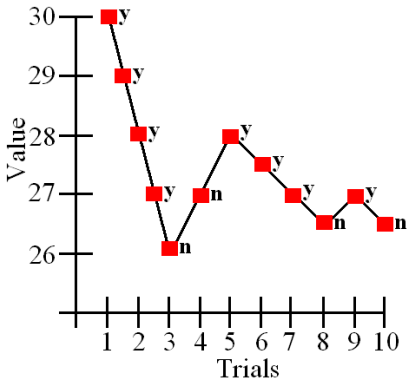


Figure 2: *Example of the stepping procedure for an up-down staircase terminated after four reversals.*

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.

1. Perception of Human Appearance

The main aim of this experiment is to establish if and when various virtual human representations are perceptually equivalent. This is especially important in LOD crowd systems displaying different representations simultaneously. Using a psychophysical approach, we attempt to derive a perceptual metric to aid in deciding when to use a particular representation based on the virtual human's appearance. Hamill et al. [HMDO05] and McDonnell et al. [MDO05] detail these experiments in full.

1.1. Experiment 1: Impostor Vs. Mesh Detection

This experiment aimed to establish the distance at which participants were able to discriminate between a virtual human's high resolution mesh and an impostor. In [DHOO05], it was hypothesised that humans should be able to detect the impostor once the size of a texel is bigger than the size of the impostor image's pixel, as aliasing artifacts then start to occur as a result of stretching the impostor's image onto the quadrilateral. Based on this hypothesis, the system switched between a virtual human's impostor and mesh representation when the impostor image pixel size to impostor texel

size was a ratio of 1:1. In this experiment, we try to find the exact pixel to texel ratio at which the participants are able to discriminate between these representations.

1.1.1. Visual Content and Procedure

Participants were shown the virtual human's geometric and impostor representations at different distances for 5 seconds (Figure 3). Each representation was animated with a 1-second cyclical walk animation. The participants were asked to indicate on which side the virtual human "looked better" by pressing the corresponding trigger button on a USB gamepad. Because applications containing virtual humans would typically involve displaying them from multiple viewpoints, both representations 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.

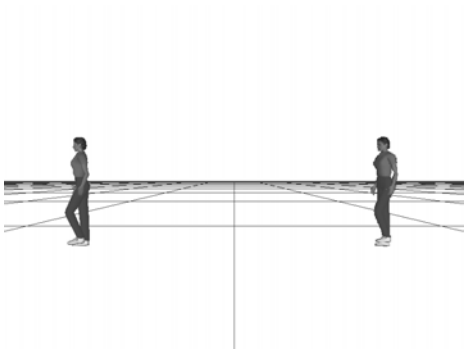


Figure 3: *Test application environment with mesh on left and impostor on right.*

1.1.2. Results

We recorded the participant's response at each distance and calculated the percentage of correct responses for each ratio at which the representations were displayed. We then plotted this as a function of the ratio and fitted a psychometric curve to the data.

Subsequently the PSE was calculated for each participant. The mean PSE value for all participants was 1.164 ± 0.064. At this PSE level, the participant will judge the representations with equal likelihood as "looking better". The mean PSE is close to the hypothesised value of 1. This means that when an impostor is directly beside its mesh counterpart, users should not notice the difference between them when the impostor is displayed at a pixel to texel ratio of 1 to 1.

$$Pixel : TexelRatio(distance) = \frac{distance \times PixelSize}{TexelSize} \quad (7)$$

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

A common LOD approach for reducing the computational cost associated with rendering a high detailed mesh is to replace it with a simpler, lower resolution mesh containing fewer triangles, where the loss of detail should be imperceptible to the viewer of the system (see [dHCUCT04]). However, care has to be taken when generating these low resolution meshes, as removing too much detail can produce blocky shaped results, along with animation artifacts due to the loss of joint vertices, and the overall visual realism of the virtual human is reduced (Figure 4). The second experiment was aimed at establishing the resolution, in terms of percentage of vertices, at which participants were able to discriminate between a virtual human's high resolution mesh and a selection of simplified low resolution meshes for three different distances.

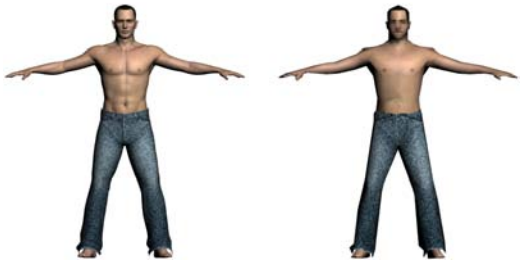


Figure 4: *(left) high resolution model, (right) blocky shaped low resolution version.*

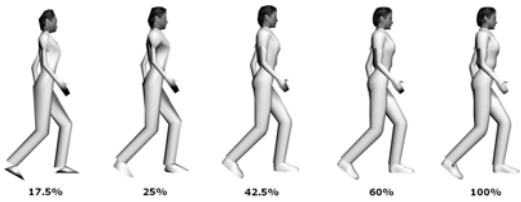


Figure 5: *Illustration of some of the simplified models used for* Experiment 2.

1.2.1. Visual Content and Procedure

In these experiments, a female model of 2170 polygons was used. Using the 3D Studio Max *Multires* modifier, a selection of low resolution meshes were generated in this manner, ranging from a reduced vertex percentage of 60% to 15% (2170 polygons to 262 polygons) at intervals of 2.5% (see Figure 5).

A high resolution mesh and a low resolution mesh were displayed alongside each other for comparison. The participants were asked to indicate whether the representations looked the "same" or "different" by pressing the respective left or right trigger button on a USB gamepad (Figure 6). Each time the participant indicated a "same" response, the resolution of the low LOD mesh was decreased, otherwise a "different" response increased the resolution.

As mentioned previously, three distances at which to display the representations from the viewer were chosen. The first distance was 5 metres, the second distance was 8 metres and the third distance was 16 metres, which corresponded to

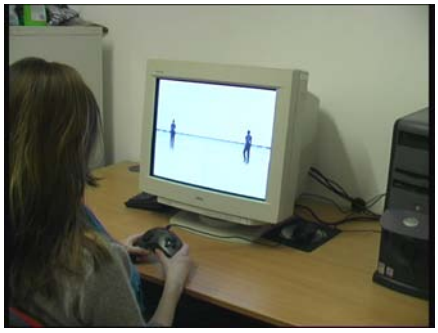


Figure 6: *Participant taking part in* Experiment 2.

66.6% and 33.3% of the representation's initial screen space size.

1.2.2. Results

We recorded the participant's response for each mesh resolution displayed and calculated the percentage of correct responses for each resolution at which the representations were displayed, and plotted this as a function of the resolution.

It was found 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.

The mean PSE values give us a good indication as to the percentage of vertices necessary for a low resolution mesh to be considered the same as its higher counterpart 50% of the time. However, this value is not very practical for developers, so we also included a further analysis. After much consideration, we chose the 80% probability of acceptance as the level to explore further as a practical level for developers. The 80% probability of acceptance is the percentage of vertices necessary for an observer to say that they found the low resolution mesh equivalent to the high resolution 80% of the time. We felt that this level was high enough to ensure that the difference between the low and high resolution meshes would not be noticed in most situations, such as in a game. Also, this level was considered low enough to be of practical use in real-time, as a 90% probability would result in a mesh almost as detailed as the high resolution mesh, and would not represent a significant saving.

The mean 80% probability of acceptance values are presented in Figure 7, along with the mean PSE values. The corresponding number of vertices and polygons for the 80% probability of acceptance are shown in Table 1.

From these results, a low resolution mesh generated at 42.2% is acceptable to use as a replacement for the high resolution, at the closest distance, as observers will consider them to be same 80% of the time. This suggests that we are using a mesh that is too detailed for the highest LOD in our crowd system, and a less detailed model could be used without the user noticing, while improving the system's performance.

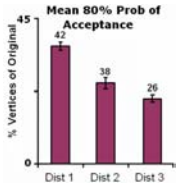


Figure 7: *(left) Mean PSE values, (right) Mean 80% probability of acceptance.*

Distance	5.0	8.0	16.0
Vertex %	42.2±1.0	37.6±1.0	26.2±1.0
Vertices	487	434	302
Triangles	871	770	524

Table 1: *Developer guidelines from 80% probability of acceptance results.*

$LOD_{Geometry}$	Distance (metres)	$Cost_{LOD}$ (ms)	Crowd Size @ 30FPS
High Res 100%	< 5.0	0.0645	517
Low Res 42.2%	> 5.0	0.0241	1,385
Low Res 37.6%	> 8.0	0.0221	1,507
Low Res 31.9%	> 12.416	0.0198	1,686
Low Res 26.2%	> 16.0	0.0135	1,961
Impostor	12.416	0.00697	4,777

Table 2: *The distance at which $LOD_{Geometry}$ models are perceptually equivalent (using 80% probability of acceptance values) and their associated rendering cost.*

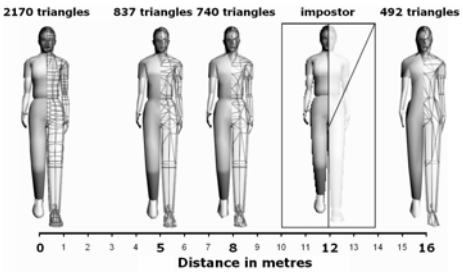


Figure 8: *Summary of distances at which to display representations. In this example 12 metres is equivalent to the distance at which an impostor achieves the acceptable pixel to texel ratio.*

1.2.3. Developer Guidelines for Experiment 1 and 2

The results from *Experiment 1* showed that participants could not discriminate between impostors and their high resolution model at a pixel to texel ratio of approximately 1.16, which corresponds to a distance of 12.416 metres. However, low resolution meshes can be perceptually equivalent to their high resolution mesh at a closer distance. By using the results from *Experiment 1*, 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

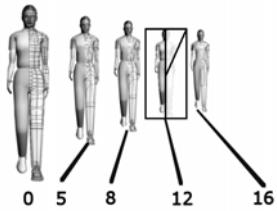


Figure 9: *Summary of distances at which to display representations, where representations are shown at the actual distance used.*

distance as the impostor. This corresponds to a low resolution mesh of approximately 31.9%, or 621 triangles. Due to the rendering cost of each model (Table 2), we suggest that it would be advantageous to use the impostor instead of a low resolution mesh for virtual humans being displayed at a ratio greater than 1.16. To summarise, Figures 8 and 9 illustrate the distances at which low resolution meshes and impostors are perceptually equivalent to a high resolution mesh.

1.3. Experiment 3: Impostor/Mesh Switching Discrimination

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 a high resolution mesh only for humans in the front, and impostor representation that can be displayed at a fraction of the rendering cost of the mesh. We 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 it is 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 Experiment 3, we aimed to establish the distance at which the transition from a pre-generated impostor to a mesh is noticeable. 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 Experiment 2), 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.

1.3.1. Visual Content and Procedure

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 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. Each time the participant indicated a "yes" response, the distance at which the switch occurred was increased, otherwise a "no" response decreased the distance.

Two separate experiments were carried out, with the model either facing the user or spinning on the spot. The virtual human switched from its impostor to its geometric representation at a switching distance ranging from 6 to 31 units.

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.

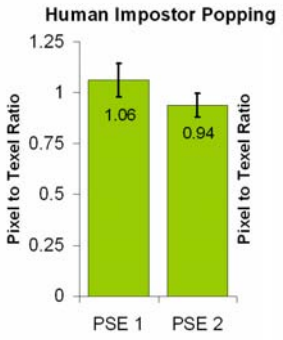


Figure 10: *Results of the popping detection experiments (showing the PSE in terms of pixel to texel ratio) for humans facing viewer (1) and spinning (2).*

1.3.2. Results

We recorded the participants' responses for each trial's switching distance. A psychometric curve was fitted to each participant's experimental data. The mean PSE calculated (shown as PSE1 in Figure 10), was approximately the predicted one-to-one value. The mean PSE calculated for the

second experiment (shown as PSE2), was less than for PSE1, suggesting that the spinning was a distracting factor.

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 17×8 pre-generated viewpoints fitted into a 1024×1024 image which is an image size commonly used in these types 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 participant is capable of detecting any popping artefacts.

2. Perception of Motion

In a LOD crowd system that simultaneously displays different model representations, as described in [DHOO05], it is important that the quality of the motion of the lower LODs is not significantly different from that of the high resolution. Hodgins et al. [HOT98] showed that model type affected user perception of human motion, when a stick figure model's motion was compared to a polygonal model. Here, we test whether or not the lower detail representations replicate the motion of the high resolution mesh, and we also test their ability to replicate deformable cloth motion. Finally, we look at how motion capture data is perceived when applied to different models. More detailed descriptions of these experiments can be found in [MDO05, MDCO06, MJH*07].

2.1. Experiment 4: Perception of Human Motion

The first experiment in this section aims to analyse the ability of low resolution geometry and impostors to replicate the motion of the higher resolution human mesh that they were generated from.

2.1.1. Visual Content and Procedure

Three different representations of a male model were used. Two of the models were polygonal models with deformable meshes which were manipulated by an underlying skeleton; the high resolution polygon model had a deformable mesh of 2022 polygons, while the low resolution polygon model had only 808 polygons for a deformable mesh. We automatically simplified the mesh as much as possible, without making the simplified version look different from the original. Impostors were the third type evaluated and the same pre-generated approach was used as in the other experiments.

A reference motion *R* was created which consisted of 10 frames of a key-framed walk motion. The 10 frames of *R* were modified a number of times to create the arm, leg, and torso motion variation sequences.

Firstly, the performance of the participants in distinguishing smaller and larger dynamic arm motions was examined. Assessing the arm motion variation involved comparing *R* to a set of motions which altered the distance of the arm from the body at certain keyframes. The modifications were made by rotating the upper left arm joint in *kAl* at the shoulder along the positive horizontal axis by a fixed number of degrees (Figure 12 (left)). The right arm was altered by the same amount in the reverse direction.

A similar test was conducted to test the ability of the participants in distinguishing larger and smaller leg motions for all representations. The leg was altered by iterative translations along the longitudinal and vertical axes (Figure 12 (middle)). Finally, the ability of the participants to distinguish modifications to the torso was tested. In this instance, the alterations were made by iteratively rotating the lower spine of the skeleton by a fixed number of degrees around the longitudinal axis (Figure 12 (right)).

Figure 12: *(left) Top view of min and max arm variation, (middle) Perspective view of min and max leg variation, (right) Top view of min and max torso variation.*

2.1.2. Experiment Procedure

Participants viewed pairs of movies, and were asked to specify whether they thought that the motion of the character in the movies was the "same" or "different" (Figure 13). After the first 4-second movie was viewed, the participant pressed a "view next" button on the screen using the mouse. The next movie was then presented for 4 seconds and the participant had to decide whether they thought that the motions were the same or different and press the corresponding on-screen button.

Figure 13: *Participant taking part in the Experiment 4.*

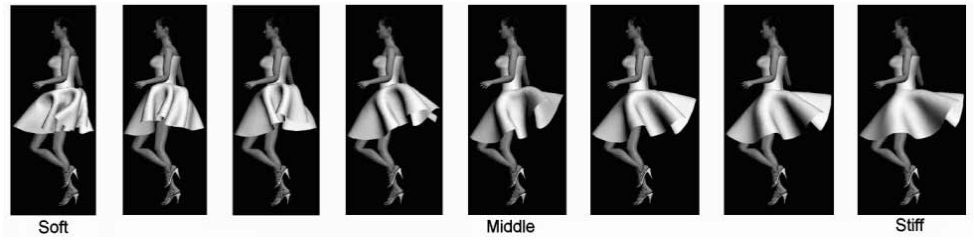


Figure 11: *Eight different materials used in order of increasing stiffness*

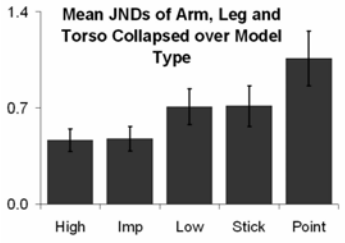


Figure 14: *Mean JND values for all motion variations collapsed over model type.*

2.1.3. Results

For each participant, the number of times that they viewed a pair of motions at each stimulus level was recorded, along with the number of correct responses that they gave at that level. The percentages of correct responses were then plotted against the stimulus level values. As we are interested in sensitivities to differences in motion for the different model representations, and not actual threshold values we examined the JND values rather than the PSE values in this experiment. Psychometric curves were then fitted to the datasets and, for each participant, the JND were calculated from these curves. The JND was then found by calculating the difference between the PSE and the stimulus level value that corresponded to 75% correct responses on the psychometric curve.

An ANOVA was used to compare mean JND values across all of the participants and showed that there was a significant difference in their sensitivities with respect to the changes viewed. The significances for the differences between model types indicate that the motion of the impostor was closer to that of the high resolution polygon model than that of the low resolution model (Figure 14).

We suggest that this is due to the fact that, even though the impostor appears perceptually different to the high resolution model at the distance shown in the experiments, it replicated the motion of the high resolution model accurately. The low resolution model may not replicate this motion as effectively because there are fewer vertices on the mesh, and even though it is the same skeleton used to deform this mesh, the deformation loses subtle motion information.

2.2. Experiment 5: Perception of Cloth Motion

Displaying large crowds of high quality virtual characters with deformable clothing is not possible in real-time at present because of the cost of rendering the thousands of polygons necessary to depict the subtle motion of the cloth. Current real-time crowd systems are capable of displaying thousands of skinned characters by using lower quality representations instead of high quality to achieve their goal. Hybrid systems that switch between high and lower quality models depending on the distance from the camera, are also a solution to this problem.

Our aim was to extend the hybrid crowd system described in [DHOO05] to include clothed characters. However, we were not certain at the outset which representation (i.e., low detail geometry or impostor) would be most suited to displaying the deformations of simulated cloth, although we hypothesised that low geometry would not be sufficient to reproduce the required deformations. We address the questions: Can impostors and low resolution geometry display a range of different cloth materials? How well can they reproduce individual material types? If compared directly, which representation would resemble a higher quality representation more closely?

2.2.1. Stiffness Sorting Condition

This condition aimed to establish whether the real-time lower detail cloth representations could produce distinguishable levels of cloth stiffness. After some experimentation, we found eight cloth motions which ranged from very soft to very stiff (Figure 11). Eight movies, each showing one of the representations, were placed randomly on a 21-inch widescreen monitor, each playing in a loop. Participants were asked to place the 8 movies in order of stiffness, with the least stiff on the left and the most stiff on the right. The order in which the participants placed the movies was recorded and compared to the actual sequence.

Figure 15 shows us that overall the participants found the low detail geometry cloth more difficult to sort than the high detail, and the impostor representation. There was no statistical significance between sorting the impostor and sorting the high resolution movies. These results indicate that the perception of subtle differences in cloth motions using the impostor is closer to that of the high detail geometry cloth

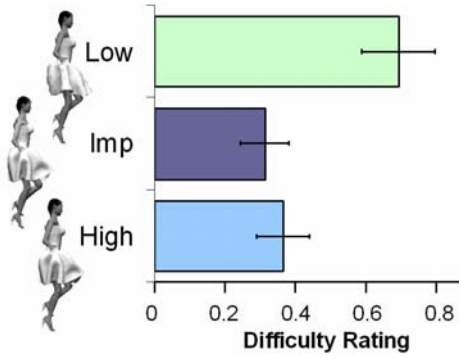


Figure 15: *Results of Stiffness Sorting Condition.*

simulation than the low geometry simulation. This also supports the findings in Experiment 4 that impostors are better at depicting small differences in human motion.

2.2.2. Stiffness Forced Choice Condition

The next condition we tested was to determine how well the high and low detail geometry and impostor reproduced the stiffness levels of a gold standard cloth (rendered offline with non-realtime rendering and lighting). Participants were shown 2 gold standard movies (that of the stiff and that of the soft skirts) beside each other. They were asked after every trial to indicate which of the two gold standard animations (stiff or soft) was more similar to the current target animation, playing on an adjacent monitor.

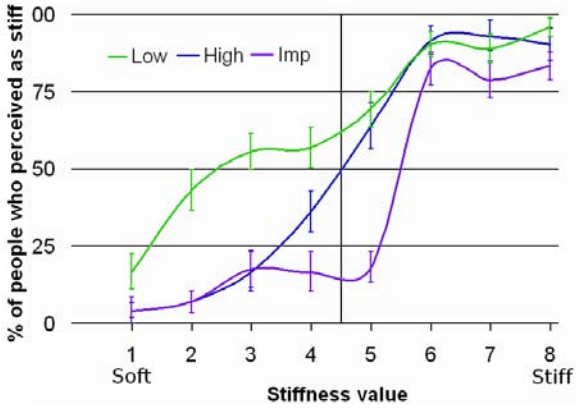


Figure 16: *Perceived stiffness for different LOD cloths. Standard error bars are shown.*

The interpretation of these results is evident from Figure 16. Participants found that the perceived stiffness of the cloth motion for the impostor was closer to that of the high resolution than the low resolution for low stiffness levels (i.e., soft materials). This suggests that the impostor better matches the high detail geometry motion at low stiffness levels. There is a divergence at the middle stiffness levels, where most participants rated the impostors to be soft, and few found the low detail to be soft. At the high stiffness levels, more participants' perception of the low resolution cloth motion was closer to the high detail geometry than the impostor. Overall, there seems to be a tendency for the participants to perceive the low detail geometry cloth motion to be stiffer than the high geometry, and the impostor to be less stiff than the high geometry.

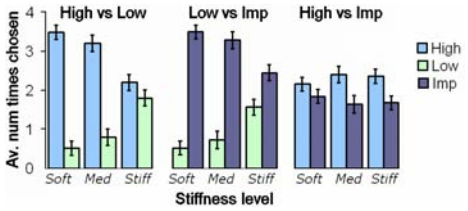


Figure 17: *Results for LOD Comparison Condition.*

2.2.3. LOD Comparison Condition

A third condition was tested in order to see how well each representation matched the gold standard. Participants were first shown pairs of the most rigid cloth and were asked which cloth animation was most similar to the gold standard rigid cloth. Participants pressed left or right buttons on the gamepad to choose the most similar simulation. They were then shown pairs of the cloth with stiffness approximately halfway on our estimated scale (i.e., the fifth image in Figure 11), and were asked to compare them with the corresponding gold standard cloth. Finally, the participants viewed pairs of the most soft cloth and were asked to compare them to the gold standard as before. The number of times that a participant preferred each LOD representation over each of the others was recorded.

Our results (Figure 17) suggested that, when viewing these representations in a hybrid system that simultaneously displays virtual humans using two types of representation, switching intermittently between them, the low resolution will not resemble the high resolution as closely as the impostor does, thus resulting in significant artifacts.

2.3. Experiment 6: Applying Motion Capture

Crowds simulated with synthetic walking motions can lack personality, so motion captured data can be used to add realism. In this experiment, we investigate some factors that affect the perceived sex of walking virtual humans, with a view to increasing the realism of pedestrians in real-time crowd simulations. We cannot simulate everyone in a crowd with their own personal motion captured walk, as the more motions we use, the greater the demands on potentially limited computational and memory resources (e.g., a games console or hand-held device). Therefore, the challenge is to optimise quality and variety with the resources available. Specifically, we ask the question whether, if there is a clear visual indicator of sex (i.e., a highly realistic, unambiguously female or male model, as shown in Figure 18), will motion or form information dominate our perception of the sex of the character? If motion information alone always determines per-

ceived sex, then we would always need to create templates of every different motion for both males and females. However, if we find that form dominates, or that simulated neutral motions are as good as captured natural motions under some circumstances, then such duplication may not always be necessary. Perhaps some actors' walks can be equally effectively applied to both male and female models. Any of these results would allow us to create "canonical" motions to which variety could later be added, irrespective of sex.

2.3.1. Visual Content and Procedure

Six undergraduate students (3M, 3F) volunteered to be motion captured, each in a separate session per actor. We captured some of the walks without their knowledge to ensure they were walking naturally, then applied the motion capture data to characters in 3D Studio Max and kept one natural walk per actor. We also generated three different neutral walk motions, with neither male nor female characteristics (such as hip sway or shoulder movement).

Four different models were used to display the different motions (Figure 18): highly detailed woman and man models of approximately 35000 polygons each, an androgynous character, and a point light walker. The woman and man were chosen as typical characters that would be used in a computer simulation of natural crowds. The androgynous figure was chosen as it did not appear particularly male or female and so could serve as a control. The point light walker was generated from a generic neutral skeleton and so contained minimal shape information.

Figure 18: *Four model representations were animated with real female, real male or synthetic neutral motions. From left to right: Woman model, Man model, Androgynous figure and Point light walker.*

Each of the different motion types (3) from each of the actors (3) were applied to each of the model types (4), with two repetitions for each condition, resulting in a total of 72 movies. Participants viewed the movies and were were told to take both motion and form/shape into account and they marked their selections on an answer sheet. Participants categorised the character they just saw on a five-point scale of 1: *very male*, 2: *male*, 3: *ambiguous*, 4: *female* or 5: *very female*.

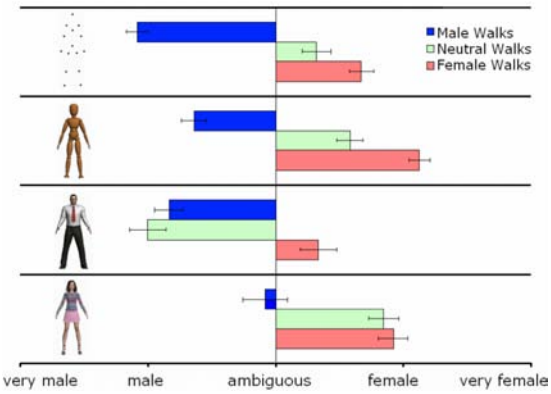


Figure 19: *The interacting effects of model and walk type.*

2.3.2. Results

Results show that male walks on the woman are rated as ambiguous, as are female walks on the man (Figure 19). This implies that applying motion captured from actors of the opposite sex to the character will produce confusing or unsatisfactory results in general. Interestingly, neutral walks were considered male when viewed on the man and female when viewed on the woman. This implies that for neutral walks, the appearance of the character takes precedence over the motion in determining the sex of the character. This result has implications for computer graphics applications where resources are limited, as re-using the same neutral walks on male and female characters would appear to produce the desired effect.

Also, for a character with androgynous appearance, the motion information is most important when determining the sex (as without motion, the androgynous figure was consistently rated to be ambiguous).

3. Perceptual metrics for smooth animation

In this section, we describe a series of experiments designed to provide metrics to developers for smooth animations. The first experiment in this section finds the optimal number of viewpoint images necessary for smooth impostor animations. The next experiment aims to find the optimal pose update rate for characters performing different animations. Finally, we look at simulation LOD and establish whether different pose updates would be acceptable if displayed together in one scene. These experiments are described in full in [MDCO06, MNO07].

3.1. Experiment 7: Impostor Update Frequency

From the results recorded in previous experiments, we can see that impostors are good at representing the deforming folds of cloth and are a good substitute for high resolution geometry clothed models at a 1:1 pixel to texel ratio distance from the camera. As mentioned above, impostors are generated by rendering multiple images of the human from different viewpoints for every frame of animation. The appropriate viewpoint is selected with respect to the camera in the real-time system. Typically, these viewpoints are generated at regular intervals around a sphere, so the sampling

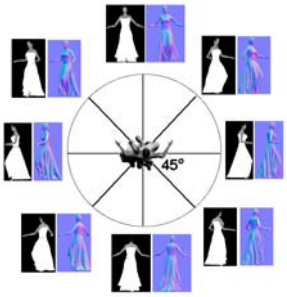


Figure 20: *Generating impostor images from a camera positioned on the circumference of a circle every 45°.*

density can be described by the number of degrees difference between each segment of the sampled sphere (e.g., Figure 20 shows impostor images generated every 45°).
Ideally, impostors would be generated at very small intervals around a sphere, the same number of times that a polygonal model would be updated, which would allow seamless transitions between the images. However, as we are using pre-generated textures, texture memory consumption prevents choosing such a dense sampling, so there is a need to pick an optimal number of viewpoint images to generate. Dobbyn et al. [DHOO05] and Tecchia et al. [TLC02] report rendering 17 and 16 viewpoints of their impostors from one side of the human and mirroring the impostors for the reverse angle. This corresponds to an update rate of 10.58° (180° divided by 17) and 11.25° respectively. However, they do not specify their reasons for choosing these numbers of viewpoints. With the addition of cloth to the impostors, mirroring is no longer possible due to the non-symmetric nature of cloth (the cloth on one side is usually not identical to the other side, due to the folds occurring in different places). Also, it was not clear that this directional sampling density would be appropriate when clothing was added to the impostors. As in Tecchia et al. and Dobbyn et al., interpolation was not used between different views, as it would be computationally intensive and could introduce visual artifacts.

Figure 21: *Six different characters used in impostor update frequency test*

3.1.1. Visual Content and Procedure

We chose 6 characters as stimuli for this experiment (Figure 21). We pre-generated 256 impostors for every frame of the 10-frame animation, which corresponded to an *optimal* sampling density of 1.4° (i.e., 360° divided by 256). The

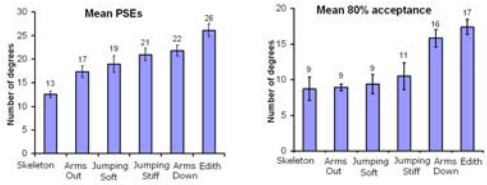


Figure 22: *(a) Mean PSE values for all models, (b) Mean values for 80% probability of acceptance*

characters moved on a circular path at a normal walk pace, with the closest point to the viewer on the circle being the pixel to texel ratio distance reported in Experiment 1, as impostors would not be viewed any closer than this in a crowd system. The character was placed at a random point on this path, and walked for 3 seconds, in either a clockwise or an anticlockwise direction. Participants were asked to specify, using the gamepad, whether they perceived the change in orientation of the character as jerky or smooth. The next trial they saw depended on their previous response.
For each participant, a psychometric curve was fitted to their dataset as described previously. This allowed us to find the Point of Subjective Equality (PSE). The PSE is the point at which participants were equally likely to find an animation smooth or jerky, i.e., where they have a 50% chance of considering that motion smooth (Figure 22). A psychometric probability curve for each of the characters, derived from all of the data, was then created, using the average of all participants' PSE and standard deviations (Figure 23).

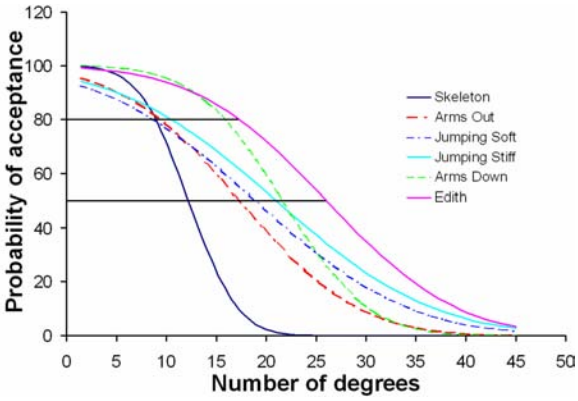


Figure 23: *Probability of acceptance curve derived from psychometric data. This shows the number of degrees necessary for each of the characters to be considered smooth for a range of different probability of acceptance values. For example, at 10 degrees, 80% of the time participants found* Edith*, a typical pedestrian, to be smooth.*

3.1.2. Developer Guidelines

For normal walking characters, with either stiff or soft clothing, a viewpoint update rate of 17° is necessary to guarantee with 80% likelihood that users will not notice viewpoint

Figure 24: *Character 1, character 2 and character 2 with deformable clothing.*

changes of the impostors. This corresponds to 21 images that need to be generated at equal spacing around the character. We suggest rounding to the nearest even number of images (22) in order to include the direct front and the direct back view images, particularly in applications where a front-on view would be most noticeable. For other characters whose width to depth ratios are large, a viewpoint update rate of 9° is advised. This corresponds to 40 images around the character. In [DHOO05] and [TLC02], updates of 10.58° and 11.25° were used. We can now see that these rates were underestimates for complex characters but overestimates for normal walking characters.

3.2. Experiment 8: Animation Update Frequency

Pose update rate can be defined as the frequency of individual simulation steps displayed when animating a character. For most real-time crowd simulation, the pose update rate is largely constrained by the available hardware and overall scene complexity. Individual mesh or image-based (impostor) keyframes can be "pre-baked" for background characters, which reduces rendering and simulation costs but increases memory consumption (thus the pps must remain low).

In this experiment, we identify some factors and thresholds for the perceived smoothness of animated virtual characters. In a Baseline Condition, we first determined whether different pose update rates are in fact needed for human motions with different character and motion properties. Our detailed Movement Condition examined the impact and interactions of various motion properties.

3.2.1. Baseline Condition

We first tried to identify baseline factors that affected the perceived smoothness of animated human motion. The goal was to find the threshold among 15 update rates (ranging from 4pps to 63pps) at which the participants found the different animations smooth, for each of the conditions tested.

The conditions were *character type* (a male 'character 1' and a female 'character 2' shown in Figure 24) and *motion type* (motion captured kungfu kick, jumping jack, walking and jogging), and *character cloth type* (i.e., character 2 as depicted in Block 1 with simple skinned clothing, and character 2 with physically simulated deformable clothing). Each condition was shown at each update rate a number of times

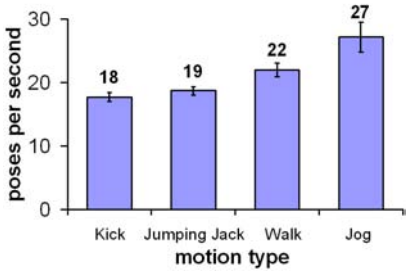


Figure 25: *Mean 50% threshold values for different animation types. Error bars show the standard error of the mean.*

and the participant pressed the left or right mouse button on the laptop to indicate "smooth" or "jerky" for each movie.

3.2.2. Analysis

We found that character type did not affect update rate in our experiments. Surprisingly, deformable clothing did not have an effect. However, motion type did have a significant effect, with motions moving further across the screen needing more updates than other motions (Figure 25).

We felt that the distance that the character moved across the screen must have been a factor, as the walk and jog motion moved much more across the screen than the other two motions. Furthermore, the amount of activity in the motion clip seemed to have an effect. Therefore we designed our next experiment condition to focus on these two factors in particular.

3.2.3. Movement Condition

The second condition examined more formally the effect of different motion types and their interactions.

3.2.4. Visual Content and Procedure

Two motion complexities were chosen: *Normal walk* with arms by the side, and *Complex walk*, the same walk motion with added activity in the arms, torso and head, each moving in time with the legs of the walk cycle. Three different cycle rates were chosen: *Lo* (1.5 cycles/sec), *Med* (2.72 cycles/sec) and *Hi* (3.75 cycles/sec). Four different linear velocities were chosen: *V0* (walking on the spot), *V1* (walking 1/3rd of the distance across the screen, i.e., 7.75screen centimetres/sec), *V2* (walking 4/6ths of the distance across the screen, 15.5cm/sec) and *V3* (walking the full distance across the screen, 23cm/sec).

We split the 4 linear velocities into four separate experiment blocks. Participants viewed all four blocks, with a one minute break in between each block. As before, they were asked to indicate whether the animation looked "smooth" or "jerky" at each trial.

3.2.5. Analysis

As in the baseline experiment, we fitted psychometric curves to participants' data for each of the conditions, and were thus able to calculate their 50% threshold values.

To summarize our findings for the Movement experiment, in Figure 27 we show a chart of the 80% acceptance thresholds

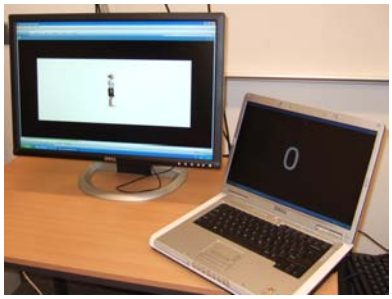


Figure 26: *System Setup for Movement Condition.*

(i.e., the level at which observers will say 'smooth' 80% of the time). We felt that these values could be of practical use to developers since the thresholds at this level were reasonable for real-time performance. Figure 28 depicts these values in a more useful way for developers to choose the correct update rates for their particular animations.

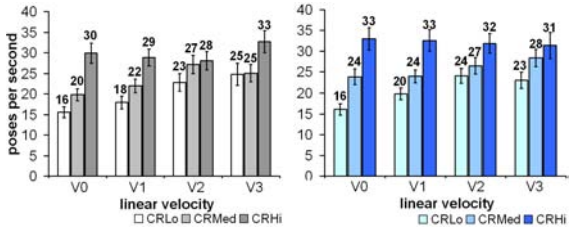


Figure 27: *(l) Thresholds for 80% probability of acceptance for normal walk. (r) Thresholds for 80% probability of acceptance for complex walk. Error bars show the standard error of the mean. CR=Cycle Rate.*

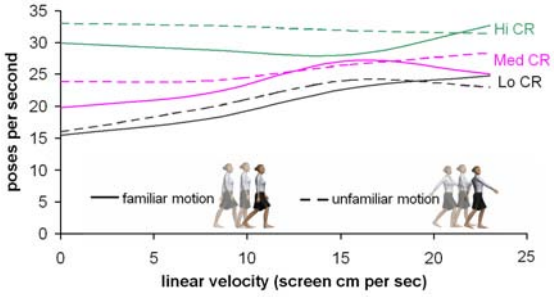


Figure 28: *Summary of Movement Experiment results. For the familiar motion, the legs were the fastest moving body parts. We calculated that the fastest pixel was moving at 7 screen cm/sec (Lo CR), 12cm/sec (Med CR), and 15cm/sec (Hi CR). For the unfamiliar motion, the arms were the fastest and we calculated that the fastest moving pixel was moving at 13cm/sec, 23cm/sec and 32cm/sec for Lo, Med and Hi CR.*

These results will perhaps be of most use (and immediately applicable) for real-time character simulation, particularly when the characters have cyclical motions. In a preprocessing step, each motion could be labeled with an optimal update rate, based on the cyclical update rate and complexity of the motion. At run-time, this rate for all characters could change priority when the camera is moving fast, to account for the added jerkiness which occurs with fast camera motion.

3.3. Experiment 9: Simulation Level of Detail

For memory critical systems such as real-time crowds using impostors, the fewer poses required to make an animation appear smooth the better. In Experiment 8, we have provided thresholds of acceptability for different characters, depending on their cycle rate, complexity and the amount of camera motion in the scene. However, we have not yet established whether different pose updates would be acceptable if displayed together in one scene, as our previous experiments showed one image at a time. Therefore, in this experiment, we consider simulation level of detail by examining the effect of its implementation on perceived motion smoothness. We give participants a discrimination task, in which they can view two pose update rates simultaneously and make their decisions based on a comparison of the two.

3.3.1. Visual Content and Procedure

We used the same character as in the previous experiments (character 2). Each movie displayed one character in the front and a group of characters in the back (Figure 29). The characters in the background were updated at 5 different rates, ranging from 5 to 30pps, and this was the stimulus level - again, we wished to determine the thresholds among the 5 update rates at which all characters appeared smooth, for each set of conditions.

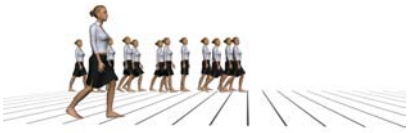


Figure 29: *Crowd in experiment setup.*

Figure 30: *Crowd in game setting used in SLOD experiment 2. Characters have different form and colour from one another and background is not white.*

This experiment had a 3-way design. The first condition was walk type: *in step* (i.e., all characters started at the same pose, and moved like an army) or *out of step* (i.e., all characters started at different poses which represented a more natural setting for a crowd of pedestrians.); the second condition was background group size: *small* (1 character), *medium* (6 characters) or *large* (12 characters); and the third condition was foreground character update rate: *update1* (30pps, which from our previous experiment we know to be the threshold at which 99% of participants perceived the motion presented in this experiment to be smooth) or *update2* (20pps, the threshold with a 75% probability of being perceived smooth).

We also wished to examine how a more natural scenario, such as that found in games, would affect our results. Therefore, we compared the 'out of step' large group to a new 'out of step' large crowd where all characters were different and appeared in a more complex background scene (Figure 30).

3.3.2. Analysis

In order for our SLOD experiment results to be of use to developers, we analysed the 80% probability of acceptance values, estimated from participants' psychometric curves. We found that 16pps was considered sufficient for all background characters that we tested (at 80% probability of acceptance there was no significant main effect of crowd size, walk type, or foreground character update rate). This result could be of great benefit to LOD crowd systems in particular. At present, in hybrid geometry/impostor crowds (e.g., [DHOO05]), 10pps are used for both foreground and background characters due to the memory consumption of impostors. However, this rate results in jerky looking animation. Using 30pps for the geometry and 10pps for the impostors resulted in noticeable differences, and using 30pps for the impostors is too costly in terms of memory. If, as our results suggest, it is possible to display geometry at 30pps and impostors at 16pps without observers noticing the difference, this will result in the ability to store double the number of characters in memory than would be possible if the impostors were being displayed at 30pps.

In order to evaluate our SLOD metrics, we plugged the value of 16pps for background characters into a simple geometry/impostor crowd scenario and found the differences in SLOD to be imperceptible. Although the differences were imperceptible in the example we tested, all of the characters were walking on the spot, so switching between update rates as the characters moved from foreground to background was not present.

4. Evaluation of Metrics

In this section, we evaluate the effectiveness of the previous perceptual metrics. This experiment is described in detail in [MDCO06].

4.1. Experiment 10: Impostor Update and Mesh Detection Metric Evaluation

This experiment aimed to test the validity of the results of the previous experiments in a real crowd scenario. The crowd scene was populated with the female jumping model used in the previous cloth experiments. The characters in the scene were either all wearing the most stiff skirt from the experiments, or the most soft skirt.

The experiment included three typical crowd systems: full geometry, hybrid high polygon/impostor and hybrid high polygon/low polygon. In the full geometry crowd system, all characters were high resolution polygonal models of 8983 polygons each (6172 for the human model and 2811 for the skirt). The hybrid high polygon/impostor system contained the high resolution polygon models nearest to the camera, and the impostor representations at the back (Figure 31 bottom). The latter were displayed at the pixel-to-texel ratio at which they are perceptually equivalent to high resolution meshes. This pixel-to-texel ratio was found for individual characters, but was never tested on a large crowd. We used the results of the previous experiment to choose the number of viewpoint images necessary for the two models.

The hybrid high/low resolution polygon system contained high resolution characters at the front, and low resolution at the back (Figure 31 middle). Five hundred and thirty polygons were chosen for the low resolution skirt.

In a typical hybrid crowd system, the LOD choice depends on the distance of a character from the camera. As the camera moves through the scene, switching between representations will occur, due to the camera distance changing. To examine this effect in our experiments, the camera zoomed up and down through a corridor between the characters at a speed of 4m/s, in order to ensure that LOD switching occurred. Switching between impostor viewpoints also occurred in this case.

The effect of switching between impostor viewpoints was then examined independently by allowing the camera to only pan from left to right at a speed of 2m/s, where the impostor distance was fixed. In this case, the impostor viewpoints were changing but no switching back and forth to high resolution geometry occurred.

One hundred and eight 4-second movies were created in total: Three types of system: *All Hi*, *Hi/Lo*, *Hi/Imp* $\times$ 2 skirt types: most stiff and least stiff $\times$ 3 crowd sizes: small (50 humans), medium (100 humans) and large (1000 humans) $\times$ 2 conditions: camera panning from left to right, camera moving up and down a fixed corridor $\times$ 3 random placings: 3 different random placings of characters in the scene.

Each participant viewed the sequence, and was asked for every trial, whether all of the characters in the scene were the same or if they noticed that some of the characters looked different in any way.

We first analysed the effect of camera panning or zooming to determine how effective or metrics for impostor viewpoint switching (Experiment 7) and LOD switching (Experiment 3) were in a crowd scenario. An ANOVA was performed

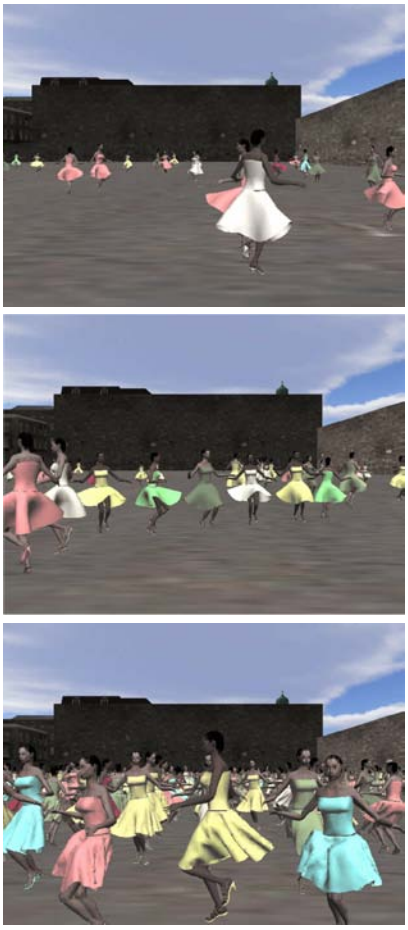


Figure 31: *(top) Small crowd of All Hi with stiff cloth. (middle) Medium crowd of Hi/Lo with stiff cloth. (bottom) Large crowd of Hi/Imp with soft cloth.*

on the dataset and it was found that there was no significant main effect of camera motion on the ability of participants to tell the differences between the representations. This implied that participants were unaware of the switching between representations or switching between impostor viewpoints, which is very good news for hybrid systems.

We then analysed the effect of stiffness (Figure 32). For the stiff cloth, participants noticed the difference between the hybrid *Hi/Lo* and *All Hi* movies significantly more times. Also, they noticed the difference between *Hi/Lo* more times than the *Hi/Imp*. There was no statistical significance between the number of times that they noticed a difference in *All Hi* compared to *Hi/Imp*.

For the soft cloth, there was a difference between *All Hi* and *Hi/Lo*, and between *Hi/Lo* and *Hi/Imp*. Again, there was no difference between *All Hi* and *Hi/Imp*. This implied that having a hybrid crowd using impostors and high resolution geometry will introduce less artifacts than a hybrid crowd with low resolution geometry.

As expected, there was a statistically significant difference between *Hi/Lo* stiff and soft, with the soft cloth low geometry being noticed more times than the stiff cloth. There was also a difference between impostor stiff and soft - with differences in the soft being noticed fewer times than differences in the stiff cloth. Similar differences in stiffness were present for *All Hi*.

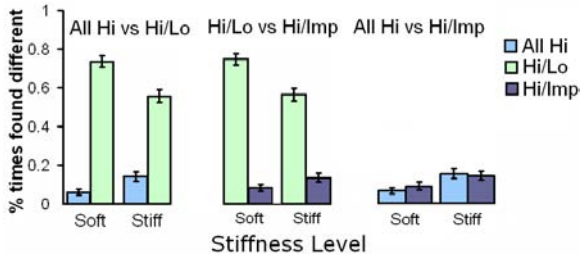


Figure 32: *Experiment 10: LOD vs Stiffness*

The previous experiments all depicted scenes with only 1 or 2 characters. This represents the worst-case scenario, as the character was being analysed directly, with no surrounding distractions. Surprisingly, it was found that there was no overall effect of crowd size, implying that differences could be noticed just as easily in small crowds as large crowds.

5. Conclusions and Future Work

We have gained new insights into the effects of different level of detail representations with respect to human motion, appearance and secondary motion. Although these results are useful in terms of helping to improve real-time crowd systems, this is still an area of research that would benefit from more perceptual studies. The ultimate goal is to create a framework for highly detailed crowds, driven by perceptual metrics.

In Experiment 10 we found that the size of the crowd did not affect perception of background character LOD. While this was a surprising result, the participants only had the task of trying to perceive differences between the foreground and background characters. We feel that if the viewer was asked to perform a different task, these background differences might not be perceived as often, and the crowd size may have an effect in this case. We base this assumption on the fact that previous research by Cater et al. [CCW03] has shown that humans fail to notice degradations in image quality in parts of the scene unrelated to their assigned task. Also, Harrison et al. [HRvdP04] noticed that expectation about the task affected perception of motion. Varying the task is certainly something we would like to examine in the future, as it is likely that viewers of crowd scenes will be involved in game-play in the foreground of the scene, or navigating through a city.

We presented guidelines for developers on the number of impostor viewpoints needed in order to produce imperceptible switching, for one elevation of impostors (i.e., those on the ground plane). Using the same number for higher elevations

might be wasteful, so it would be interesting to determine the number of updates needed for all elevations, using similar psychophysical methods.

References

[CCW03] CATER K., CHALMERS A., WARD G.: Detail to attention: Exploiting visual tasks for selective rendering. In *Proceedings of the Eurographics Symposium on Rendering* (2003), pp. 270–280.

[Cor62] CORNSWEET T.: The staircase method in psychophysics. *American Journal of Psychology 75*, 3 (1962), 485–491.

[dHCUCT04] DE HERAS CIECHOMSKI P., ULICNY B., CETRE R., THALMANN D.: A case study of a virtual audience in a reconstruction of an ancient roman odeon in aphrodisias. *The 5th International Symposium on Virtual Reality, Archaeology and Cultural Heirtage (VAST)* (2004), 9–17.

[DHOO05] DOBBYN S., HAMILL J., O'CONOR K., O'SULLIVAN C.: Geopostors: a real-time geometry / impostor crowd rendering system. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games* (2005), pp. 95–102.

[Ebb98] EBBESMEYER P.: Textured virtual walls - achieving interactive frame rates during walkthroughs of complex indoor environments. In *Proceedings of the Virtual Reality Annual International Symposium* (1998), pp. 220–228.

[HBF02] HARRISON J., BOOTH K. S., FISHER B. D.: Experimental investigation of linguistic and parametric descriptions of human motion for animation. *Computer Graphics International* (2002), 154–155.

[HMDO05] HAMILL J., MCDONNELL R., DOBBYN S., O'SULLIVAN C.: Perceptual evaluation of impostor representations for virtual humans and buildings. *Computer Graphics Forum 24*, 3 (2005), 623–633.

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

[HRvdP04] HARRISON J., RENSINK R. A., VAN DE PANNE M.: Obscuring length changes during animated motion. *ACM Transactions on Graphics 23*, 3 (2004), 569–573.

[LCR02] LEE J., CHAI J., REITSMA P.: Interactive control of avatars animated with human motion data. *International Conference on Computer Graphics and Interactive Techniques* (2002), 491–500.

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

[LHEJ01] LINSCHOTEN M., HARVEY L., ELLER P., JAFEK W.: Fast and accurate measurement of taste and smell thresholds using a maximum-likelihood adaptive staircase procedure. *Perception and Psychophysics 63*, 8 (2001), 1330–1347.

[MAEH04] MANIA K., ADELSTEIN B. D., ELLIS S. R., HILL M. I.: Perceptual sensitivity to head tracking latency in virtual environments with varying degrees of scene complexity. In *APGV '04: Proceedings of the 1st Symposium on Applied perception in graphics and visualization* (2004), pp. 39–47.

[MDCO06] MCDONNELL R., DOBBYN S., COLLINS S., O'SULLIVAN C.: Perceptual evaluation of LOD clothing for virtual humans. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2006), pp. 117–126.

[MDO05] MCDONNELL R., DOBBYN S., O'SULLIVAN C.: LOD human representations: A comparative study. *Proceedings of the First International Workshop on Crowd Simulation* (2005), 101–115.

[MJH*07] MCDONNELL R., JÖRG S., HODGINS J. K., NEWELL F., O'SULLIVAN C.: Virtual shapers & movers: Form and motion affect sex perception. In *Proceedings of the ACM Siggraph/Eurographics Symposium on Applied Perception in Graphics and Visualisation (to appear)* (2007).

[MNO07] MCDONNELL R., NEWELL F., O'SULLIVAN C.: Smooth movers: Perceptually guided human motion simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (to appear)* (2007).

[ODGK03] O'SULLIVAN C., DINGLIANA J., GIANG T., KAISER M. K.: Evaluating the visual fidelity of physically based animations. *ACM Transactions on Graphics 22*, 3 (2003), 527–536.

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

[RP03] REITSMA P., POLLARD N.: Perceptual metrics for character animation: Sensitivity to errors in ballistic motion. *ACM Transactions on Graphics 22*, 3 (2003), 537–542.

[TLC02] TECCHIA F., LOSCOS C., CHRYSANTHOU Y.: Visualizing crowds in real-time. *Computer Graphics Forum (Eurographics 2002) 21*, 4 (2002), 753–765.

[Tre95] TREUTWEIN B.: Minireview: Adaptive psychophysical procedures. *Vision Research 35*, 17 (1995), 2503–2522.

[WB03] WANG J., BODENHEIMER B.: An evaluation of a cost metric for selecting transitions between motion segments. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), pp. 232–238.

Populating Virtual Environments with Crowds: Real-Time Crowd Rendering with Pre-Generated Impostors

S. Dobbyn, R. McDonnell, and C. O'Sullivan

Graphics, Vision and Visualisation (GV2) Lab, Trinity College Dublin, Ireland

Abstract

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 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. This part of the tutorial describes the impostor representation used in our crowd rendering system, detailing our programmable hardware based method for lighting and adding variation.

1. Introduction

This part of the tutorial describes the impostor representation used in our Geopostor system, a real-time geometry/impostor crowd rendering system (Figure 1). The Geopostor system has been developed to solve the challenging problem of large-scale crowds 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.

Our main contribution is that our system 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 [DHOO05], 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.

2. Real-Time Crowd Rendering with Pre-Generated Impostors

While a deformable mesh was the obvious choice for the virtual human's highest LOD in our crowd system, 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

Figure 1: *Screenshots of the Geopostor system.*

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.

3. 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. [TC00]. A set of template mesh models were used in the pre-generation of the necessary impostor images in 3D Studio MAX. 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 2(b) and (c)) and was assigned a specific pre-defined material. It should be noted that the diffuse colour of each material is set to white (as shown in Figure 2(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.

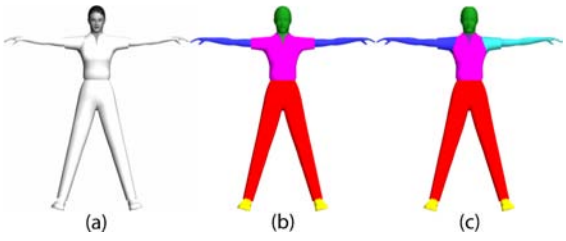


Figure 2: *(a) High LOD mesh representation in 3D Studio MAX. (b) The grouping of triangles based on material used (shown by the different colours). (c) The grouping of triangles based on the body part it represents (shown by the different colours).*

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. 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 3 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.

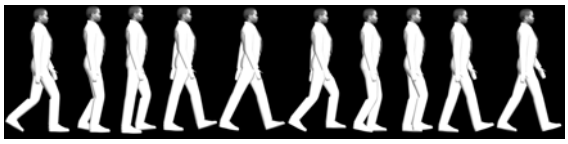


Figure 3: *Precalculating and storing the deformation of a mesh performing a walk animation for 10 key-frames.*

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. 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. The normal maps in [DHOO05] took a considerable time to generate, as per-pixel look ups and operations were needed, so we improved the algorithm by using a less computationally intensive technique. A copy of the character's mesh at the current frame was first needed. Each vertex normal was first converted into eye-space coordinates, to find the normal with respect to the camera, and then converted into an RGB colour (using Equation 1). Per-vertex colouring was then used to paint the RGB colours onto the vertices of the copied meshes (3D Studio Max's VertexPaint modifier was used to do this). These vertex colours were interpolated over the polygons, creating a character mesh with normal map colours. The normal map image was then generated by rendering an image of this mesh, from the current viewpoint. Per-vertex colouring and interpolation are operations that are performed very quickly, as they are supported by graphics hardware. This meant that the image could be produced almost immediately, without the need for slow per-pixel operations.

$$\begin{aligned} Pixel_R &= ((0.5 * N_x) + 0.5) * 255 \\ Pixel_G &= ((0.5 * N_y) + 0.5) * 255 \\ Pixel_B &= ((0.5 * N_z) + 0.5) * 255 \end{aligned} \tag{1}$$

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 runtime 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.

4. 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 [3Dc]. Another way of reducing these artefacts is to store one of the components in the alpha channel and then use DXT5 compression, which compresses the alpha values independently at a higher accuracy [ATI].

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 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 3), 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.

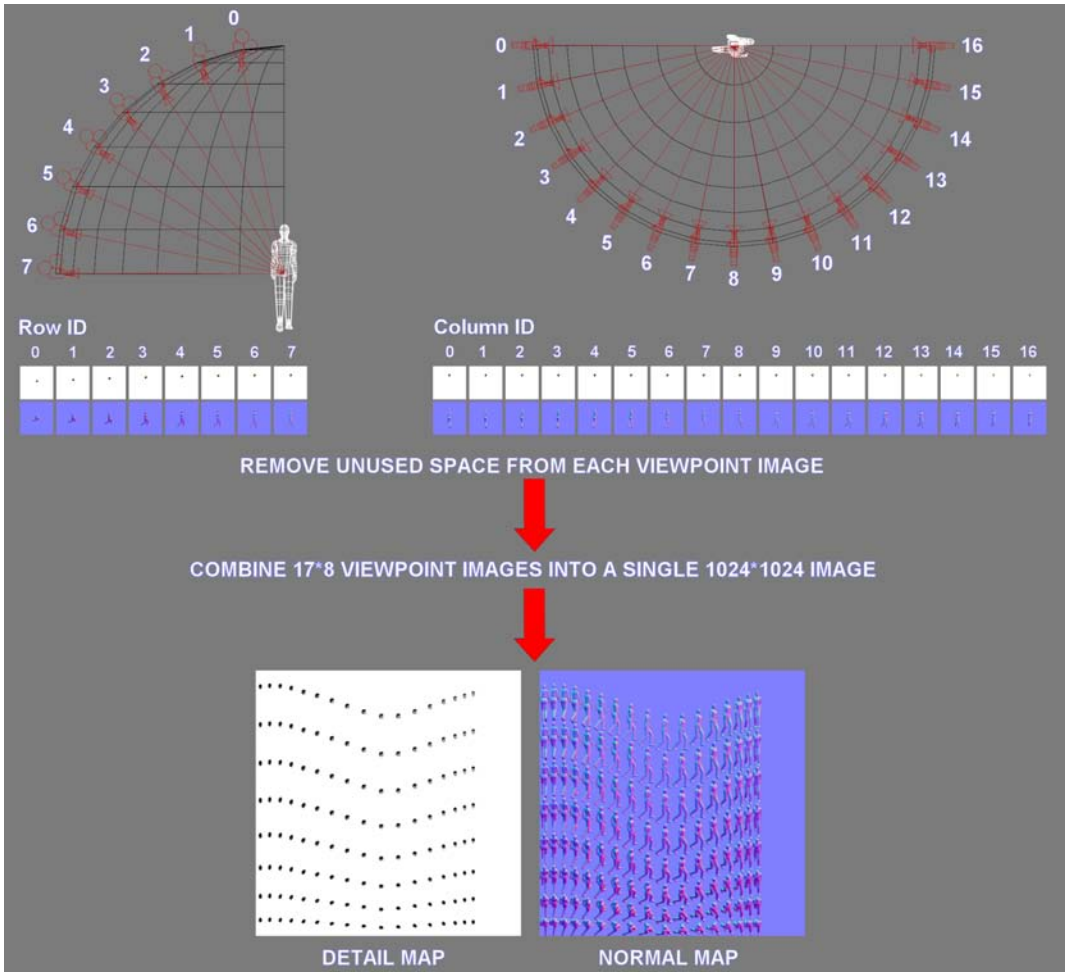


Figure 4: *A MaxScript plug-in removes unused space from each viewpoint image and combines 17*8 viewpoint images into a single 1024x1024 image for a particular frame.*

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 2.

$$\overrightarrow{N} = \frac{\overrightarrow{H} - \overrightarrow{C}}{|\overrightarrow{H} - \overrightarrow{C}|} \tag{2}$$

The vector from the camera to the human projected onto the ground plane $\overrightarrow{CH}$ can be calculated (Equation 3) using $\overrightarrow{N}$. It should be noted that in Equation 2, 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.

$$\overrightarrow{CH} = \frac{(N_x, 0, N_z)}{|(N_x, 0, N_z)|} \tag{3}$$

The amount by which to rotate the quadrilateral around the x-axis θ_x and y-axis θ_y is calculated using Equation 4. The viewpoint's row and column ID (V_{Row} and V_{Column}) can be used to lookup which viewpoint to render using Equation 5, where N_x and N_y are the number of viewpoint images pre-generated around the x- and y-axis.

$$\begin{aligned} \theta_x &= \cos^{-1}(N_y) \\ \theta_y &= \cos^{-1}(CH_z) \end{aligned} \tag{4}$$

$$V_{Row} = \theta_x \times \frac{N_x}{90}$$
$$V_{Column} = \theta_y \times \frac{N_y}{180} \quad (5)$$

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. [TLC02] 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, and thus the shading of the impostor is based on Equation 6.

$$\begin{aligned} Pixel_{Colour} = {} & DetailTexture_{RGB} * \\ & (Ambient_{LightModel} * Ambient_{Material} + \\ & (MAX(Vector_{Light} \cdot Normal_{Vertex}), 0) * \\ & (Diffuse_{Light} * Diffuse_{Material})) \end{aligned} \quad (6)$$

Similar to the mesh representation, the lighting of the impostor representation has been implemented in hardware using both texture shaders and register combiners [NVR99], and vertex and fragment programs [Ver02, Fra02]. This involves implementing Equation 7 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 5.

$$\begin{aligned} Pixel_{Colour} = {} & DetailMap_{RGB} * \\ & (Ambient_{LightModel} * Ambient_{Material} + \\ & (MAX((Vector_{Light} \cdot NormalMap_{RGB}, 0) * \\ & (ColourMap\,[DetailMap\,\alpha]) * Diffuse_{Light})) \end{aligned} \quad (7)$$

Similar to the mesh model, we optimise the rendering of the impostors by precalculating and storing each of the keyframe's viewpoint data in a single VBO object. Since dynamically orientating the quad involves the computationally expensive $\cos^{-1}$ function (see Equation 2), 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.

5. Variation LOD: Adding Variety to the Impostor Model'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 a 3D Studio MAX plug-in that allows the artist to select particular colors for each body material from a colour palette (see Figure 6). The impostor can be rendered in 3D Studio MAX's viewport in real-time using a shader written in HLSL to give the artist immediate feedback.

A multi-pass method, as described in [TLC02], 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 7). 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. 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].

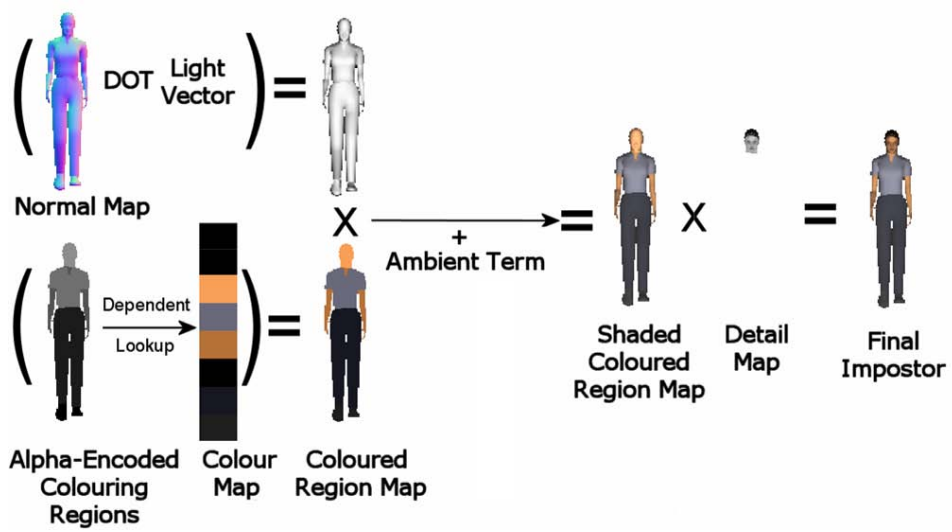


Figure 5: *Impostor shading and colouring sequence.*

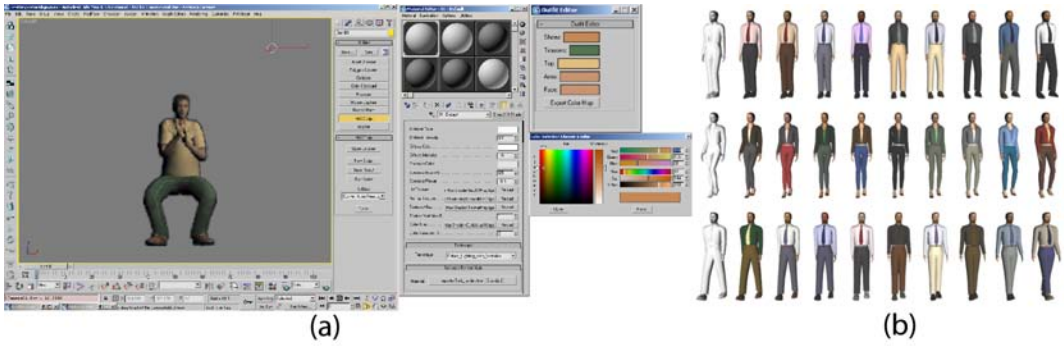


Figure 6: *(a)Example of an artist in progress of generating an outfit for a model using the 'Outfit Editor' plug-in. (b) Nine outfits for three template meshes.*

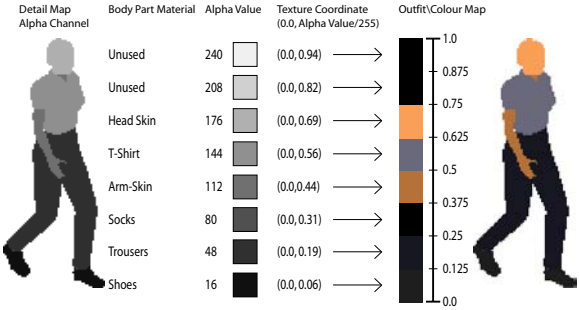


Figure 7: *Using programmable texture addressing to add variety to the impostor representation.*

6. Real-time Clothed Crowds with Pattern Variation

In the past, skinned meshes for the clothing of individuals in a simulated crowd were used, often resulting in rigid and unnatural motion. While there have been advances in the area of cloth simulation, both offline and in real-time, interactive cloth simulation for hundreds or thousands of clothed characters would not be possible with current methods. In [DMK*06], we addressed this problem and devised a system for animating large numbers of clothed characters.

The majority of games that implement cloth dynamics on current generation hardware do so in a highly constrained manner, often ignoring the issues of collision detection by allowing the cloth surface to penetrate nearby surfaces. Game developers favour cloth that is highly controllable and tend to use more traditional methods of bone based skinning and pre-simulated vertex mesh animations if the performance of the cloth is critical to the game. In simulated crowd scenes for games, cloth is rarely if ever used due to the large numbers of polygons required to accurately capture the deformation of the cloth. However, deformable clothing adds greatly to the realism of the characters.

We have added realism to our crowd simulations by dressing the individuals in realistically simulated clothing, using an offline commercial cloth simulator, and integrating this into our real-time hybrid geometry/impostor rendering system. Additionally, we developed a technique for generating cyclical motion for pre-simulated cloth, which therefore moves in a fluid, realistic manner. Furthermore, we have added variety to our impostor representation by developing a new hardware rendering technique for adding pattern variety to the same cloth for different humans in a crowd. Our results show a system capable of rendering large realistic clothed crowds at interactive frame rates.

6.1. Brief overview of Cloth Simulation

Implementing realistic cloth dynamics in real-time game applications still represents a significant challenge for game developers. Simulating cloth deformation is a complex process both in terms of dynamics simulation and collision detection for the changing cloth shape. Many game developers have relied on more tractable but approximate solutions including the Verlet method [Jak01], and have restricted the simulation's complexity by only using a subset of the mesh to represent the cloth. Another method to introduce complex clothing is to generate the folds using cloth simulation and then use skinning to attach the clothing to the character. This method works well in some cases, but often results in the unrealistic bending of folds in the cloth, as the folds have to deform according to the skeleton of the character. This can make a long flowing skirt look like trousers with folds. Also, the important secondary motion of the cloth is lost in this case. In the animation research community, Cordier and Magnenat-Thalmann [CMT05] use a data-driven approach for real-time processing of clothes. Vassilev et al. [VSC01] developed an efficient technique for dynamic cloth simulation using a mass-spring model.

6.2. Cyclical Cloth

In a real-time crowd system, the characters' animations are often cyclical in nature, so that they can be smoothly linked to allow them to move in a fluid manner. Cyclical animations are commonly obtained by manually altering the underlying skeletal motion so that they loop in a realistic looking manner. However, making looping animations using characters with pre-simulated clothing is a more difficult task, as manual cleanup of the cloth to make it cyclical is very time-consuming, particularly for very deformable items of clothing like skirts, and can result in unrealistic effects.

We wanted a more automatic way of generating cyclical cloth and began by creating a very long animated sequence, repeating the animation of the human many times and simulating the cloth in response to the repeating animation, in the expectation that it would at some point become periodic. On viewing these long sequences, it was found that the cloth did not always settle to a periodic state, particularly for highly deformable clothing such as long flowing skirts. A more robust method was needed in order to obtain a good cycle in the cloth. In a good cloth cycle, the cloth at the start frame *Fs* and at the end frame *Fe* of the animation cycle of length *l* should be the same, and be travelling at the same velocity. The long cloth sequence needed to be searched using a distance metric that took into account all of the vertices on the cloth mesh between the two frames of animation, in order to find one correctly cyclical loop.

Figure 8: *Edge Images taken from 5 different viewpoints.*

We used a distance metric similar to Kovar et al. [KGP02] to compute the differences between all frames that were of length *l* apart in the sequence, and chose a set of candidate cycles whose distances were below a user set threshold. Usually, we chose the 5 cycles with the lowest distance metric as candidate cycles. In the case of stiff clothing, where the motion is very restricted, picking the cycle with the smallest distance metric was often enough to produce a good cyclical motion. For other more deformable, flowing clothing, this metric was often insufficient, as it did not weight the importance of the folds in the cloth, but rather, weighted all points equally.

Bhat et al. [BTH*03] showed that the human perceptual system is sensitive to moving edges, and used this to compare the folds and silhouettes of simulated cloth to that of video cloth sequences to find the difference between them. We based the second pass of our algorithm on this idea of matching folds and silhouettes, and devised a metric for comparing the candidate cloth cycles at *Fs* and at *Fe*. For each of the candidate cycles, we generated images of the cloth at *Fs* and *Fe* from 5 different viewpoints around the skirt. The images at *Fs* and at *Fe* for each of the viewpoints were then converted into edge images (Figure 8), using the standard Canny edge detection algorithm [Can86]. The mean distance between edges in the corresponding images of *Fs* and *Fe* were then found using an edge distance estimator, and the resulting differences in the 5 images were summed together, to give a final difference metric for the candidate cycle. The cloth cycle with the smallest edge difference was chosen as the final cycle. In most cases, this final cloth cycle was good enough for use, but in certain cases, an extra linear blend step between *Fe* and *Fs* was needed to produce the final cycle. We tried to avoid linear blending where possible, as it often resulted in the cloth intersecting with the human model, and we also felt that the results were more natural when blending was not used.

This method produced cloth that appeared cyclical from all viewpoints for all of the clothing that we tested. Cyclical appearance was judged by whether or not it was possible to notice a discontinuity in the cloth motion at the start and end of

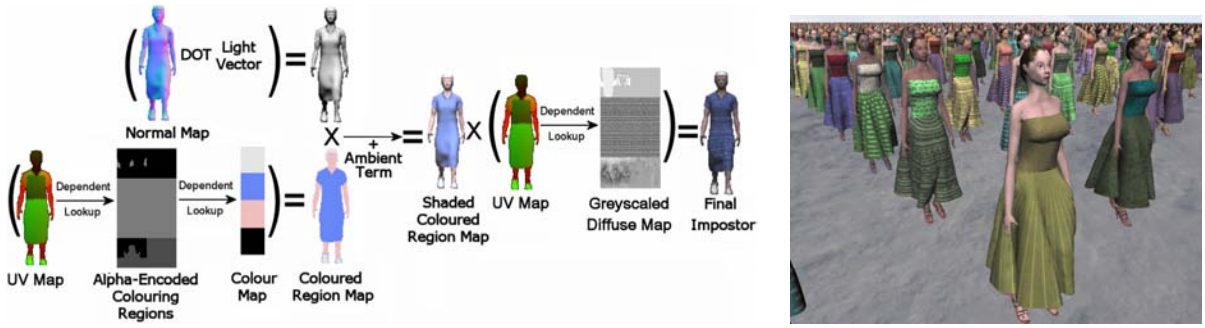


Figure 9: *(a) UV mapped impostor rendering sequence and (b) an example of adding variation to one character using 8 different diffuse textures.*

the cycle from all viewpoints (i.e., the looped animation did not exhibit a flicker in the cloth). We found that tighter clothing needed few animation cycles to produce cyclical cloth (sometimes as few as 4 cycles), as the cloth settled to a near periodic state quickly. Whereas looser cloth needed up to 40 animation cycles to produce a nice cyclical animation. The cyclical cloth and human animations could then be exported into a real-time system and replayed, and impostors were also generated as described next.

6.3. Cloth Geopostors

Our clothed crowd system builds on our Geopostor system [DHOO05], described earlier. Furthermore, our system includes clothed characters by using commercial software [CloFX] to obtain our cloth simulations, but any high quality offline simulator could be used to produce the cloth animation. We pre-simulate the deformation of both the virtual human's skin mesh using linear blend skinning and its cloth mesh using the physical simulator, based on the motion of its underlying skeleton. However, while the secondary motion of the character's cloth greatly adds to our crowd's visual realism, cyclical cloth motion is necessary to avoid any jerky motion artefacts and we present a technique to solve this in Section 6.2.

Once the character's meshes are pre-simulated, they are then exported and stored in separate keyframed meshes or "poses". By pre-calculating and storing the deformation of the skin and cloth mesh in poses, this avoids the cost of deforming the character's body and simulating its clothes at run-time. Generating the impostor representation of our clothed character involves capturing two types of images from a number of viewpoints around the model: a detail map image to capture the detail of the model's diffuse texture and a normal map image whereby the model's surface normals are encoded as an RGB value. At run-time, the clothed virtual humans switch between the two level of detail (LOD) representations depending on their position with respect to the viewer.

Adding colour variation to an impostor representation has already been achieved through the encoding of colouring regions in the alpha channel of the detail image, as described above. This is used at run-time to address a colour or "outfit" map through programmable texture addressing. In the case of a mesh, another method to add texture variation is to provide it with a set of different diffuse textures with which it can be texture mapped. However, applying this type of variation to the impostor would require exporting a set of detail maps for each different diffuse texture used, resulting in the rapid consumption of texture memory. To solve this problem, we propose replacing the detail map with a UV map.

6.4. UV Mapping Technique

We improve upon existing impostor techniques for adding variety by replacing the detail map images (described in Section 6.3) with a texture coordinate map or *UV map*. This is similar to a normal map whereby it is generated for each viewpoint and contains the texture coordinates of the model's surface encoded as a RGB value. At run-time, these values are used to lookup the same set of diffuse textures used by the mesh, allowing texture variation for both the human's skin and cloth. To also allow for colour variation, the alpha channel of the mesh's diffuse textures is encoded with alpha encoded regions which are used to lookup the colour map. The overall sequence for shading and adding both colour and texture variation to the impostor representation is shown in Figure 9. Before the impostor's UV mapped images are pre-generated, texture seams should be kept to a minimum when texture mapping the associated mesh. Otherwise, these seams result in incorrect texture coordinates being stored in the UV map, which causes rendering artefacts to arise at run-time due to the wrong pixels in the diffuse texture being addressed. A similar type of artefact also occurs when linear filtering is used, causing the background pixels and the impostor's silhouette to be linearly interpolated and generating incorrect texture coordinates. However, this is only noticeable when the impostor is close to the viewer.

This new type of image allows the texture variety and interest of each clothed impostor to be greatly increased. Replacing the detail map with the UV map image ameliorates the problem of trying to add the same type of variation using detail maps, which results in the consumption of large amounts of texture memory (see Section 6.5). Additionally, these UV maps could be further utilised to enhance the impostor's realism by applying various per-pixel lighting tex-

tures as specular maps, which are commonly used by high resolution game characters.

6.5. Results

Frame rate tests were carried out on the clothed crowd system to investigate how many clothed humans could be displayed using different LOD representations at 30 fps (see Figure 10). 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. In each test, all of the humans were on the screen walking on the spot, and were dynamically lit.

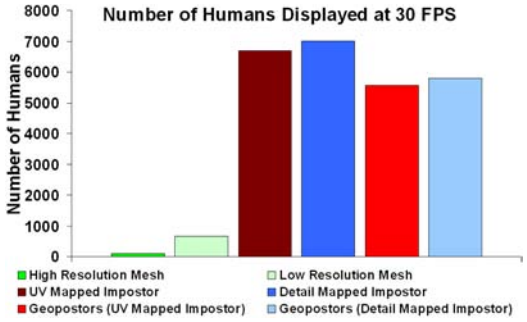


Figure 10: *Number of humans displayed at 30 fps using different LOD representations.*

It was found that 90 high resolution geometric models of clothed characters (13,056 triangles each) could be displayed onscreen at one time. With loss of visual quality, this number could be increased to 655 when low resolution geometry clothed characters were displayed (1,899 triangles each). Approximately 6,600 UV mapped impostors could be displayed at the same frame rate, but the closer humans appeared very pixellated. As expected, the number of detail mapped impostors displayed is higher due to the new UV map approach requiring extra texture lookups and per-pixel operations. When we used a hybrid geometry/impostor approach as in [DHOO05], up to 6,000 humans could be displayed (15 high resolution, 5,985 UV mapped impostors) which allowed visual quality and real-time performance to be maintained. Visual fidelity was maintained as impostors were displayed at the 1:1 pixel-to-texel ratio, where they are perceptually equivalent to high resolution meshes [Ham05].

Furthermore, in the case of switching between a clothed character's mesh and impostor representation, adding variation to the cloth using the detail mapped approach requires a substantially larger amount of texture memory in comparison to the UV mapped impostor (see Figure 11). These calculations use a single clothed character and assume that the detail mapped and UV mapped impostor consist of 10 frames of animation, each normal map being a 1024x1024 sized RBG image and both the detail map and UV map are 1024x1024 sized RGBA images. Additionally, each diffuse texture used by the clothed character's mesh representation is a 1024x1024 RGBA sized image. DXT3 texture compression is used in these calculations to reduce the memory requirements by 4 for all RGBA images and by 6 for all RGB images.

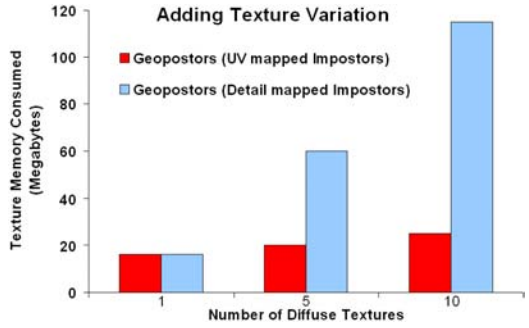


Figure 11: *Texture memory consumed by adding pattern variation to the Geopostor system using UV mapped and detail mapped impostors for a single clothed character.*

7. 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, 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. We solve this problem by layering head and arm gestures on top of the default impostor animation, whereby 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 corresponding mesh's skeleton, the gesturing bones are updated and the affected part of the mesh is deformed and rendered (Figure 12). 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

Figure 12: *Adding variety to the virtual human model's animation by layering head and arm gestures on top of the default walk animation.*

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.

8. 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: $\overrightarrow{N}$ + d = 0 and the homogenous position of the light $\overrightarrow{L}$, a 4×4 planar projected shadow matrix S can be constructed using Equation 8 (see [Bli88] [HMAM02] for the derivation of the matrix).

$$S = \begin{pmatrix} D - L_x * N_x & -L_x * N_y & L_x * N_z & -L_x * d \\ -L_y * N_x & D - L_y * N_y & -L_y * N_z & -L_y * d \\ -L_z * N_x & -L_z * N_y & D - L_z * N_z & -L_z * d \\ -L_w * N_x & -L_w * N_y & -L_w * N_z & D - L_w * d \end{pmatrix} \quad (8)$$

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

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.

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 13 (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 9. The projection of the impostor onto the ground plane with respect to the light position can be calculated using $\overrightarrow{N}$ and Equations 3 and 5 (previously described in Section 4). The impostor's shadow requires no more than a single textured quad, and therefore is extremely fast to render.

$$\overrightarrow{N} = \frac{\overrightarrow{H} - \overrightarrow{L}}{|\overrightarrow{H} - \overrightarrow{L}|} \quad (9)$$

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 13 (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, 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

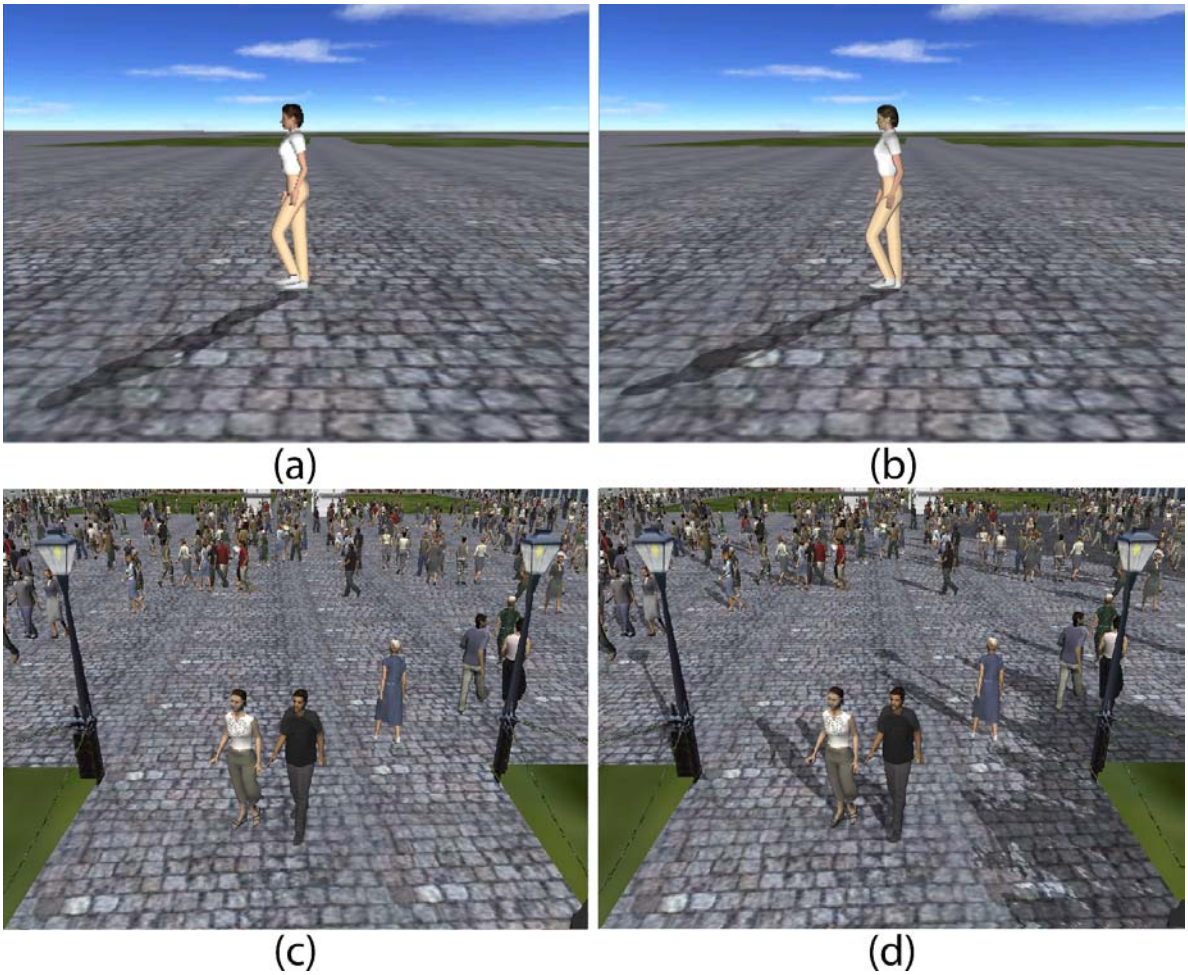


Figure 13: *(a) Projected impostor shadow. (b) Projected mesh shadow. (c) Crowd and city without shadows. (d) Crowd and city with projected LOD shadows.*

the constructed shadow volume for an impostor is incorrect as a result of being a semi-transparent quadrilateral.

9. Performance Optimisations

9.1. 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

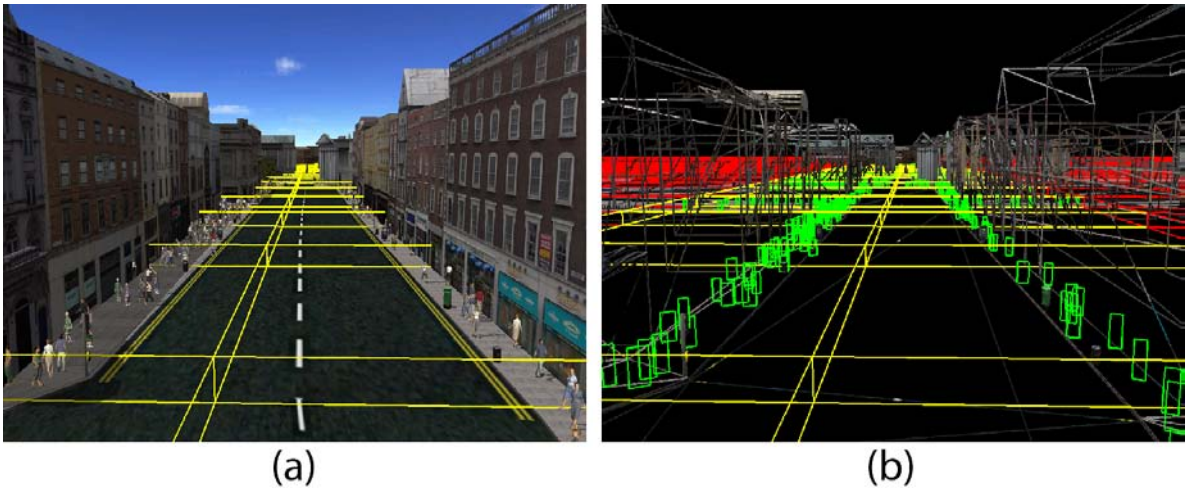


Figure 14: *Occlusion Culling: (a) Environment is divided into nodes to facilitate occlusion culling. (b) Characters that are in unoccluded nodes (shown in yellow) are drawn while those that are in occluded nodes (shown in red) are discarded.*

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 (see Figure 14(a)). By re-using these nodes so that they record 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 (as shown in Figure 14(b)). 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.

9.2. 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 a certain number of seconds. 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. 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.

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.

9.3. 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.

9.4. 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 15).

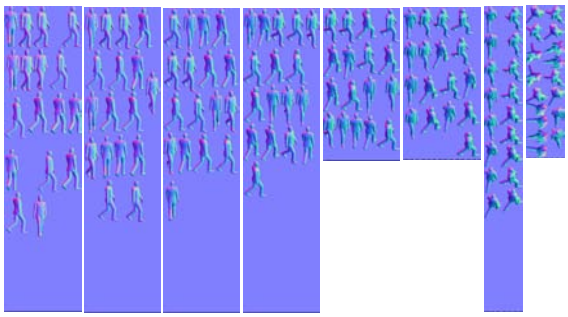


Figure 15: *Normal map split into smaller elevation images.*

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 10, 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 10). 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. It should be noted that in order for the transitions between viewpoints to appear smooth, the perceptual metrics detailed in [MDCO06] should be employed. These metrics are dependent on character dimensions, with characters of large width to depth ratios requiring more viewpoint images than those with small width to depth ratios.

$$\theta_E = tan^{-1}(\frac{h_{cam}}{d_{xz}}) \tag{10}$$

9.5. Optimisations For Spectator Crowds

In the case of crowds that do not move within the virtual environment, such as those found in sports games, viewpoint selection and the orienting of the billboard for each character can be done on the vertex processor. For each individ-

Figure 16: *(a) Rendering thousands of characters in a single draw call. (b) Frame rate results.*

ual, the billboard's vertices are set to the individual's position and the corresponding texture coordinates are set to the character's directional vector. These values are subsequently used by the vertex processor to dynamically rotate the vertices towards the camera view and lookup the texture coordinates for the most suitable viewpoint image. This means that the billboards of individuals can be batched together in a single vertex buffer object and rendered in a single draw call. For more details see [MR06]. It should be noted that smaller numbers of animation frames will result in bigger batch sizes, since only individuals using the same impostor texture can be batched together. Additionally, since the camera is typically limited to the playing field in sports games, pre-generating viewpoints from behind the character is not necessary, thus reducing the amount of texture memory consumed by the impostors.

As shown in Figure 16, rendering multiple instances in a single draw call greatly improves performance resulting in tens of thousands of characters drawn in real-time (see Figure 16). Two scenarios were tested to show the effect of batch size. The first test scenario involved one template model performing a single animation. The second scenario involved smaller batches as a result of using 2 template models performing one of 3 different animations. In both cases each animation was one second long and consisted of 30 key-frames. All of our tests were performed using a PentiumIV 3.6Ghz processor, with 2.0GB RAM and an NVidia Quadro FX 4400 graphics card with 512MB of video memory.

10. 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. 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 8), 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.

References

[3Dc] 3dc white paper, ATI Technologies. *http://www.ati.com/products/radeonx800/3DcWhitePaper.pdf*.

[ATI] Normal map compression, ATI Technologies. *http://ati.de/developer/NormalMapCompression.pdf*.

[BHS98] BITTNER J., HAVRAN V., SLAVÍK P.: Hierarchical visibility culling with occlusion trees. pp. 207–219.

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

[BTH*03] BHAT K. S., TWIGG C. D., HODGINS J. K., KHOSLA P. K., POPOVIC Z., SEITZ S. M.: Estimating cloth simulation parameters from video. *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on computer animation* (2003), 37–51.

[Can86] CANNY J.: A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI 8* (1986), 679–698.

[CloFX] Cloth simulation software, ClothFX.

[CMT05] CORDIER F., MAGNENAT-THALMANN N.: A data-driven approach for real-time clothes simulation. *Computer Graphics Forum 24*, 2 (2005), 173–183.

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

[DHOO05] DOBBYN S., HAMILL J., O'CONOR K., O'SULLIVAN C.: Geopostors: A real-time geometry / impostor crowd rendering system. *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games* (April 2005), 95–102.

[DMK*06] DOBBYN S., MCDONNELL R., KAVAN L., COLLINS S., O'SULLIVAN C.: Clothing the masses: Real-time clothed crowds with variation. In *Eurographics Short Papers (EG'06)* (September 2006), pp. 103 – 106.

[Fra02] Arb_fragment_programs extension, Silicon Graphics. *http://oss.sgi.com/projects/ogl-sample/registry /ARB/fragment_program.txt* (2002).

[Gui05] GUINOT J.: Image filtering with GLSL - convolution kernels. *http://www.ozone3d.net/tutorials/image _filtering.php* (2005).

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

[HMAM02] HAINES E., MÖLLER T., AKENINE-MOLLER T.: *Real-Time Rendering*. A.K. Peters, 2002.

[Jak01] JAKOBSEN T.: Advanced character physics. In *Proceedings of the Game Developers Conference* (2001).

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. *ACM Transactions on Graphics (TOG) 21*, 3 (2002), 473–482.

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

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

[MDCO06] MCDONNELL R., DOBBYN S., COLLINS S., O'SULLIVAN C.: Perceptual evaluation of LOD clothing for virtual humans. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2006), pp. 117–126.

[MR06] MILLÁN E., RUDOMÍN I.: Impostors and pseudo-instancing for gpu crowd rendering. In *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia* (2006), pp. 49–55.

[NVR99] NV_register_combiners extension,Silicon Graphics. *http://oss.sgi.com/projects/ogl-sample/registry/ NV/register_combiners.txt* (1999).

[OCV*02] O'SULLIVAN C., CASSELL J., VILHJÁLMSSON H., DINGLIANA J., DOBBYN S., MCNAMEE B., PETERS C., GIANG T.: Levels of detail for crowds and groups. *Computer Graphics Forum 21*, 4 (2002), 733–742.

[SF02] SAULTERS S., FERGUSON R.: Real-time rendering of dynamically variable scenes using hardware occlusion queries. *Proceedings of the Eurographics Ireland Rendering Workshop* (2002), 47–52.

[SVNB99] SAONA-VÁZQUEZ C., NAVAZO I., BRUNET P.: The visibility octree: a data structure for 3d navigation. *Computers & Graphics 23*, 5 (1999), 635–643.

[TC00] TECCHIA F., CHRYSANTHOU Y.: Real-time rendering of densely populated urban environments. *Proceedings of the Eurographics Workshop on Rendering Techniques* (2000), 83–88.

[TLC02] TECCHIA F., LOSCOS C., CHRYSANTHOU Y.: Visualizing crowds in real-time. *Computer Graphics Forum 21*, 4 (2002), 753–765.

[Ver02] Arb_vertex_programs extension, Sillicon Graphics. *http://oss.sgi.com/projects/ogl-sample/registry/ARB/ vertex_program.txt* (2002).

[VSC01] VASSILEV T., SPANLANG B., CHRYSANTHOU Y.: Fast cloth animation on walking avatars. *Computer Graphics Forum 20*, 3 (2001), 260–267.

[Wil83] WILLIAMS L.: Pyramidal parametrics. *SIGGRAPH '83: Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques* (1983), 1–11.

[WS99] WONKA P., SCHMALSTIEG D.: Occluder shadows for fast walkthroughs of urban environments. *Computer Graphics Forum (Eurographics '99) 18*, 3 (1999), 51–60.

[Zha98] ZHANG H.: *Effective Occlusion Culling for the Interactive Display of Arbitrary Models*. PhD thesis, 1998.