EG 2006 Course on Populating Virtual Environments with Crowds

Organiser: Daniel Thalmann

Address: EPFL-VRlab, CH 1015 Lausanne

e-mail: Daniel.Thalmann@epfl.ch

Phone: +41-21-693-5214

Fax: +41-21-693-5328

URL: http://vrlab.epfl.ch

Lecturers: Daniel Thalmann (EPFL), Carol O'Sullivan (Trinity College), Pablo de Heras Ciechomski (EPFL), Simon Dobbyn (Trinity College)

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

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

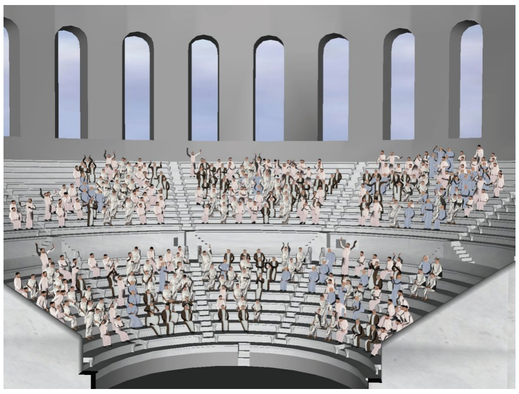
The necessity to model virtual populations appears in many applications of computer animation and simulation. Such applications encompass several different domains – representative or autonomous agents in virtual environments, human factors analysis, training, education, simulation-based design, and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

For many years, this 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.

The course will first explain in details the different approaches to create virtual crowds: particle systems with flocking techniques using attraction and repulsion forces, copy and pasting techniques, agent-based methods.

The course will explore essential aspects to the generation of virtual crowds. In particular, it will present the aspects concerning information (intentions, status and knowledge), behavior (innate, group, complex and guided) and control (programmed, autonomous and guided). It will emphasize essential concepts like sensory input (vision, audition, tactile), versatile motion control, and artificial intelligence level,. The course will survey methods for animating the individual members that make up crowds. It will survey a variety of approaches, with a focus on how example-based synthesis methods can be adapted for crowds. It will also discuss agent architectures for scalable crowd simulation.

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 simulation, AI, attention and conversational behaviour. Perceptual issues with respect to the appearance and movement of crowds of characters will be addressed.



The course will also explore essential aspects to the generation of virtual crowds. In particular, it will present the aspects concerning information (intentions, status and knowledge), behavior (innate, group, complex and guided) and control (programmed, autonomous and guided). It will emphasize essential concepts like sensory input (vision, audition, tactile), versatile motion control, artificial intelligence level, and rendering techniques. The course will also presents the new challenge in the production of real-time crowds for games, VR systems for training and simulation. Techniques for rendering a very large number of Virtual Humans will be emphasized. The course will be illustrated with a lot of examples from recent movies and real-time applications in Emergency situations and Cultural Heritage (like adding virtual audience in Roma 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 has been the leader of the Graphics group in Trinity College Dublin, Ireland, since 1999, where she has managed a range of projects with significant budgets and successfully supervised many researchers. Her research interests include perception, virtual humans, crowds, and physically-based animation. She has been a member of many IPCs, including the SIGGRAPH papers committee, and has published over 70 peer-reviewed papers. Carol has presented at SIGGRAPH several times, most recently a paper on impostor techniques for crowd simulation in the 2005 SI3D session. She has organised and co-chaired several conferences and workshops, including Eurographics 2005, the SIGGRAPH/EG Symposium on Computer Animation 2006 and the SIGGRAPH/EG Campfire on Perceptually Adaptive Graphics 2001.

Simon Dobbyn is a postdoctoral researcher at the Interaction, Simulation and Graphics Lab in Trinity College Dublin where he recently finished his PhD entitled "Hybrid Representations and Perceptual Metrics for Scalable Human Simulation". His research interests include the real-time rendering of virtual crowds, level of detail, and perception.

Pablo de Heras' goal in life is optimizing real-time rendering and exploration of and interaction with large collections of objects such as crowds of humans. He is a PhD student under the supervision of professor Daniel Thalmann at EPFL, VRlab in Switzerland where he started in 2002. He did his Master thesis at Massive Entertainment a game company in Sweden. He has been working on real-time rendering of crowds, novel tools for interaction with crowds, dynamics interaction with characters and variety editing for human characters in crowds.

Selected Publications

J. Pettre, P. de Heras Ciechomski, J. Maim, B.Yersin, J.P. Laumond, D.Thalmann, Real-Time Navigating Crowds: Scalable Simulation and Rendering, Proc. CASA 2006, Journal of Computer Animation and Virtual Worlds, July 2006

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, B. Ulicny, R. Cetre, and D. Thalmann, 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 Heritage, VAST2004

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.

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.

State-of-the-Art: Real-time Crowd Simulation

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, UT02b, 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 simu-

Figure 1: *A virtual crowd in a city*

lations 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 good 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 [DHOO05]. 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] [UT02b, 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, dHSMT05]

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, PLT05]

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. A 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 the 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 of 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 behavioral 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 pioneering 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 a 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, attractive and repulsive forces, analogous to physical electric ones, 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 boids, a more realistic motion is achieved by taking into account physical properties of 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, where 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 some actions associated. 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, UT02b] 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 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 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].

The majority of the population in the developed world lives in cities; it's 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 an 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, a sidewalk tells that it should be walked on, or a bench tells that it should be sat 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 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. Most recently, work presented by Pettre et al [PLT05] show how to automatically and robustly compute a multi-level navigation graph using three dimensional cylinders. This work also shows how to re-use the resulting path planning computation for a few hundred agents that can react to congestion along the path.

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 detail 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 pixels at the same time, where a 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. In the recent work of Hamill et al. [HMDO05] they pursue psychophysics, a discipline to decide perceptual limitations to the human vision system for example. Doing tests on how motion affects the perception of a human represented by an impostor or by a geometric structure, they were able to define distances of least noticable switching between models.

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 an 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 as in the work of Ulicny et al. [UdHCT04]. 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 was presented at I3d, where only a few actors are fully geometrical while the vast number of actors are made up of billboards [DHOO05]. A similar approach can be found in [CLM05]. A more recent approach to crowd rendering using geometry is through **dynamic meshes** as presented in the work of de Heras et al. [dHSMT05], where dynamic meshes use systems of caches to re-use skeletal updates which are typically costly. A hybrid of dynamic and baked meshes is found in [YMdHC*05] where the graphics programming unit (GPU) is used to its fullest.

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 the brain's decision, the motion is selected from an 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) were 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 the real-time strategy genre leading the way as increase of sizes of involved armies has direct effect on 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 movie production, computer games often inject realism into virtual worlds from the real world by using large libraries of animations, 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 characters that are visible or soon to be visible. 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 than handling 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 the quality of crowd rendering or the behavioral model, a virtual crowd simulation is not very useful, if it is too difficult to produce content for it. The authoring possibilities are an important factor influencing the usability of a 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 number 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 ways of how to author crowd scenes. Anderson et al. [AMC03] achieved interesting results for a particular case of flocking animation following constraints. Their method can be used, for instance, to create and animate flocks moving in shapes. Their algorithm generates a constrained flocking motion by iterating the 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 Crowd-Brush 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 the 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 representations of the world as it is easier to handle, to analyze and to validate. However, such a 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 motion, its dynamics and transition, or biomechanical constraints 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 a 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 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 in battle scenes, virtual armies have to be able to navigate around in 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 a 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 an increase of complexity of entities at all levels - whether it is visualization, animation, or behaviors. Both quantitative and qualitative improvements require novel methods, as in most cases it is not straightforward to apply the method that works for a 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 large investments due to increasing revenues from entertainment applications. For example, many movies with virtual crowds were blockbusters with revenues in the order of hundreds of millions of the dollars or 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 individuality of 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 an 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 also holds 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 still be possible, even if not practical, to use a 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.

6. Biographies

Branislav Ulicny is a research assistant and a PhD student at VRLab, Swiss Federal Institute of Technology (EPFL). His research interests include emergent crowd simulations for interactive virtual environments, behavioral animation, multi-agent simulations and artificial life. He is working on crowds and groups behavior simulation for several European projects in the areas of virtual heritage and training systems.

Pablo de Heras Ciechomski is a research assistant and PhD student at VRLab, Swiss Federal Institute of Technology (EPFL). His main research interest is in optimization techniques for realtime rendering of crowds of dynamic virtual humans. His Masters thesis was on human animation and dynamics for Massive Entertainment a game company in Sweden. He received his Master of Science degree in computer engineering at Lund Institute of Technology in Lund, Sweden.

Soraia Raupp Musse is an adjunct professor and researcher at PIPCA, UNISINOS in Brazil, where she coordinates CROMOS Lab (a Lab focused on virtual human and crowd simulation). Several projects with companies such as Petrobras, Legion, HP Brazil are been developed at CROMOS Lab. She obtained her Ph.D. degree at EPFL - Laboratoire d'Infographie, under supervision of Prof. Daniel Thalmann. Her research interests include crowd simulation, autonomous and life-like agents and human body animation.

Daniel Thalmann is a full professor at EPFL. He is director of Virtual Reality Laboratory at EPFL. His current research interests include Real-time Virtual Humans in Virtual Reality, Networked Virtual Environments, Artificial Life, and Multimedia. He has published more than 250 papers in graphics, animation, and virtual reality. He is coeditor of 25 books and coauthor of several books.

Proposers have authored and co-authored several dozens of publications on the topic of crowd simulations and related areas in various journals, conferences and workshops [FRMS*99] [ABT00] [Mus00] [MT00] [MT01] [UT01] [UT02b] [UT02a] [BMdOB03] [UdHCT04] [dHUDC04] [BdSM04] [AB05] [NC05] [DCP05] [BBOM03]. Proposers were contributing crowd and groups simulations to several ongoing and completed European (CROSSES, CAHRISMA, JUST, ERATO, eRENA, COVEN), Swiss National Fundation projects [CRO02] [JUS03] [ERA04] and other international projects with Petrobras, HP Brazil and LEGION [CRO05].

References

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

[AB05] A. Braun B. J. Bodman S. R. M.: Simulating virtual crowds in emergency situations. In *Proceedings of ACM SYmposium on Virtual Reality Software and Technology - VRST 2005* (Monterey, California, USA, 2005), ACM. 11

[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. 3, 7, 11

[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). 5

[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, 9

[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

[BBOM03] BRAUN A., BODMANN B. E. J., OLIVEIRA L. P. L., MUSSE S. R.: Modelling individual behavior in crowd simulation. In *Proceedings of Computer Animation and Social Agents 2003* (New Brunswick, USA, 2003), IEEE Computer Society, pp. 143–148. 11

[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

[BdSM04] BARROS L. M., DA SILVA A. T., MUSSE S. R.: Petrosim: An architecture to manage virtual crowds in panic situations. In *Proc. Computer Animation and Social Agents '04* (2004), pp. 111–120. 11

[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). 3, 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, 11

[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. 3, 9

[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

[CLM05] COIC J.-M., LOSCOS C., MEYER A.: Three LOD for the realistic and real-time rendering of crowds with dynamic lighting. *Research Report LIRIS, France* (2005). 7

[CRO02] CROSSES: CROwd Simulation System for Emergency Situations, 2002. project website, http://crosses.matrasi-tls.fr. 11

[CRO05] Cromos - crowd modelling and simulation laboratory, 2005. Lab website, http://www.inf.unisinos/ cromoslab. 11

[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

[DCP05] D. C. PAIVA R. VIEIRA S. R. M.: Ontology-based crowd simulation for normal life situations. In *Proceedings of Computer Graphics International 2005* (Stony Brook, USA, 2005), IEEE Computer Society. 11

[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). 2, 7

[dHSMT05] DE HERAS P., SCHERTENLEIB S., MAIM J., THALMANN D.: Real-time shader rendering for crowds in virtual heritage. In

Proc. 6th International Symposium on Virtual Reality, Archaeology and Cultural Heritage (VAST Š05) (2005). 3, 7

[dHUDC04] DE HERAS P., ULICNY B., D. T., CETRE R.: A case study of a virtual audience in a reconstruction of an ancient roman odeon in aphrodisias. In *Proc. 5th International Symposium on Virtual Reality, Archaeology and Cultural Heritage (VAST Š04)* (2004). 11

[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). 8

[ERA04] ERATO, 2004. project website, http://www.at.oersted.dtu.dk// erato. 11

[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. 3, 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. 3, 11

[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. 5

[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). 3

[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, 2

[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). 2

[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

[HMDO05] HAMILL J., MCDONNEL R., DOBBYN S., O'SULLIVAN C.: Perceptual evaluation of impostor representations for virtual humans and buildings. *Eurographics'05: Computer Graphics Forum 24*, 3 (September 2005), 581–590. 7

[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, 2, 4

[JUS03] JUST-in-time health emergency interventions - training of non-professionals by virtual reality and advanced it tools, 2003. project website, http://www.justweb.org. 11

[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. 3, 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). 8

[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. 3

[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). 3

[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. 10

[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. 8

[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, 10

[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, 4

[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

[MT00] MUSSE S. R., THALMANN D.: From one virtual actor to virtual crowds: Requirements and constraints. In *Proc. Agents'00* (2000). 11

[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, 11

[Mus00] MUSSE S. R.: *Human Crowd Modelling with Various Levels of Behaviour Control*. PhD thesis, EPFL, Lausanne, 2000. 2, 5, 11

[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

[NC05] N. COURTY S. R. M.: Simulation of large crowds in emergency situations including gaseous phenomena. In *Proceedings of Computer Graphics International 2005* (Stony Brook, USA, 2005), IEEE Computer Society. 11

[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. 6

[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

[PLT05] PETTRE J., LAUMOND J P., THALMANN D.: A navigation graph for real-time crowd animation on multilayered and uneven terrain. In *First International Workshop on Crowd Simulation (VCROWDS'05)* (2005). 3, 6

[Pow73] POWERS W. T.: *The Control of Perception*. Aldine, Chicago, 1973. 4

[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, 5

[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. 4

[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. 8

[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, 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, 10

[Sim95] SIME J.: Crowd psychology and engineering. *Safety Science 21* (1995), 1–14. 10

[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). 2

[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, 10

[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. 3

[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, 10

[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. 3, 7

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

[TT94] TU X., TERZOPOULOS D.: Artificial fishes: Physics, locomotion, perception, behavior. In *Proc. SIGGRAPH '94* (1994), pp. 43–50. 2, 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'04)* (2004), pp. 243–252. 3, 7, 9, 11

[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, 11

[UT02a] ULICNY B., THALMANN D.: Crowd simulation for virtual heritage. In *Proc. First International Workshop on 3D Virtual Heritage* (Geneva, 2002), pp. 28–32. 11

[UT02b] ULICNY B., THALMANN D.: Towards interactive real-time crowd behavior simulation. *Computer Graphics Forum 21*, 4 (Dec. 2002), 767–775. 1, 2, 5, 11

[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, 2, 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). 3, 7

[YMdHC*05] YERSIN B., MAIM J., DE HERAS CIECHOMSKI P., SCHERTENLEIB S., THALMANN D.: Steering a virtual crowd based on a semantically augmented navigation graph. In *First International Workshop on Crowd Simulation (VCROWDS'05)* (2005). 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. 10

EUROGRAPHICS 2006

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.

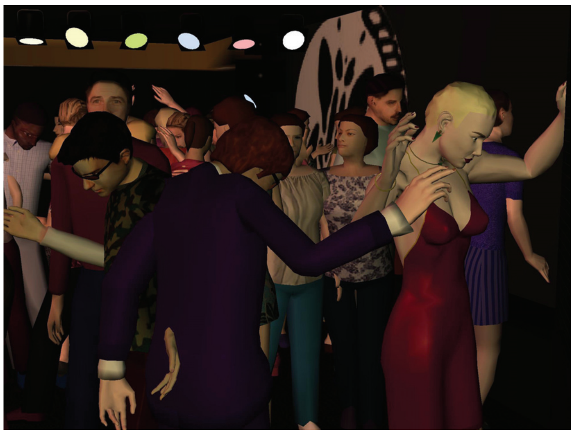


Figure 1. A group of Virtual Humans

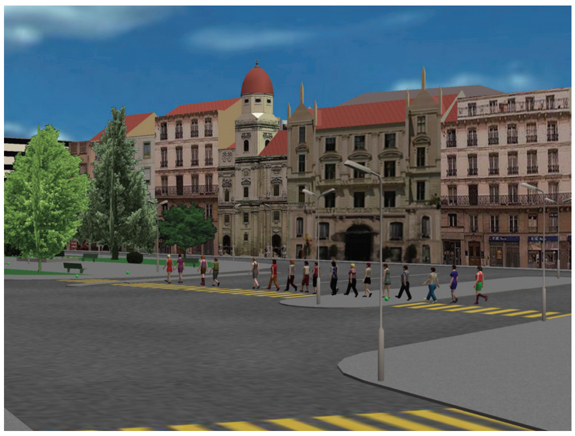


Figure 2. Virtual Humans in a city

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.

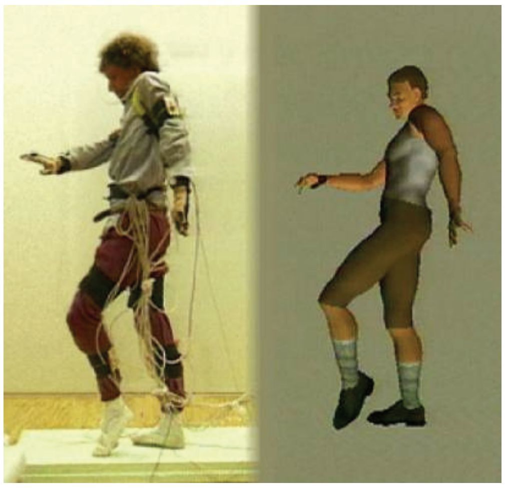


Figure 3. Motion capture

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.

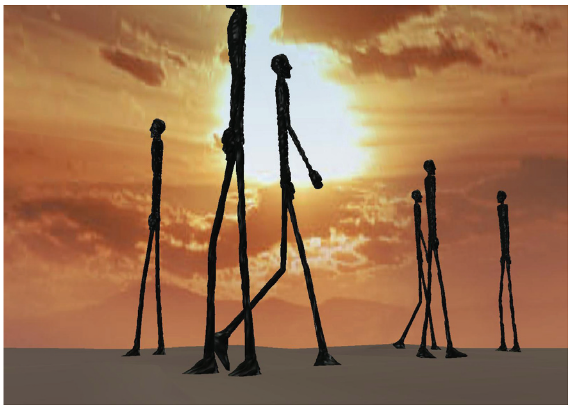


Figure 4. Individualized walking

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.

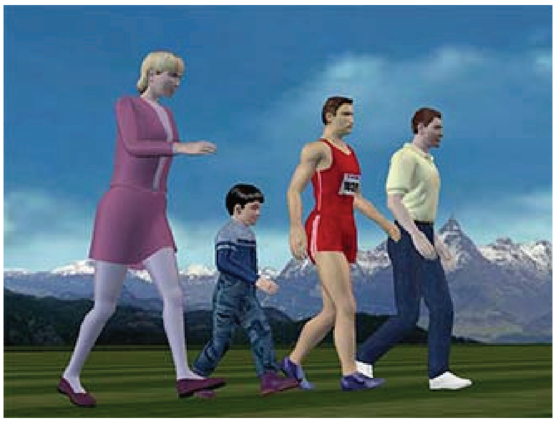


Figure 5. Examples of PCA-based walking humans

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.

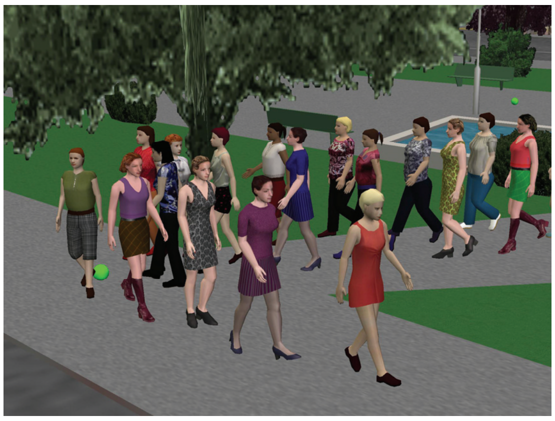


Figure 6. Crowd guided by a leader

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.

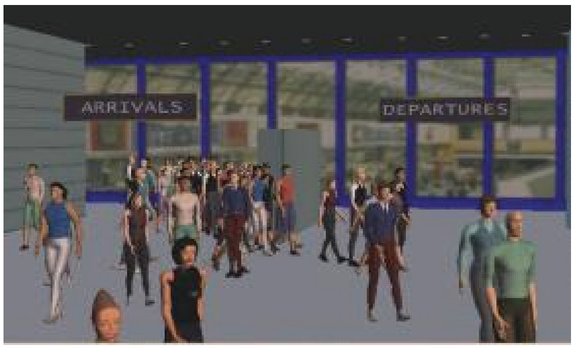


Figure 7. Train station simulation.

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.

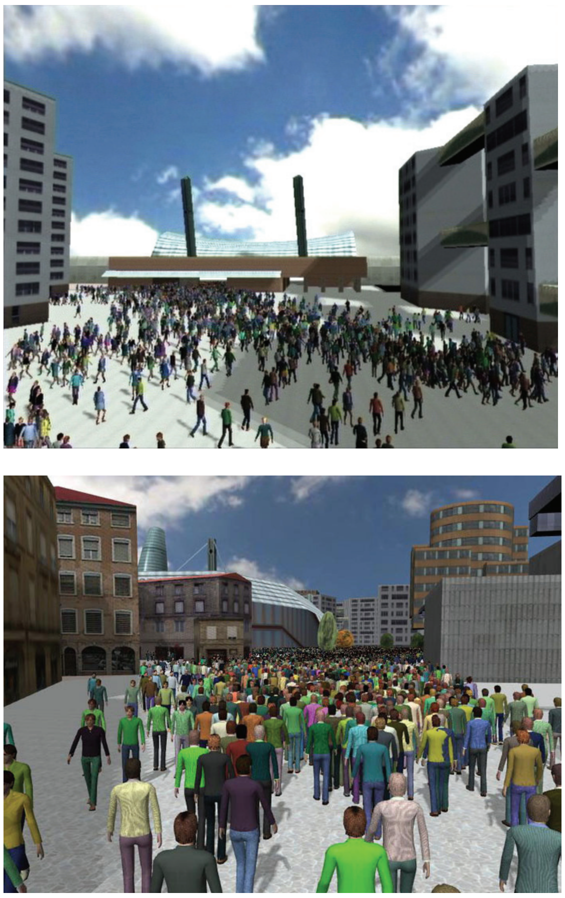


Figure 8. Examples of large crowds.

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].

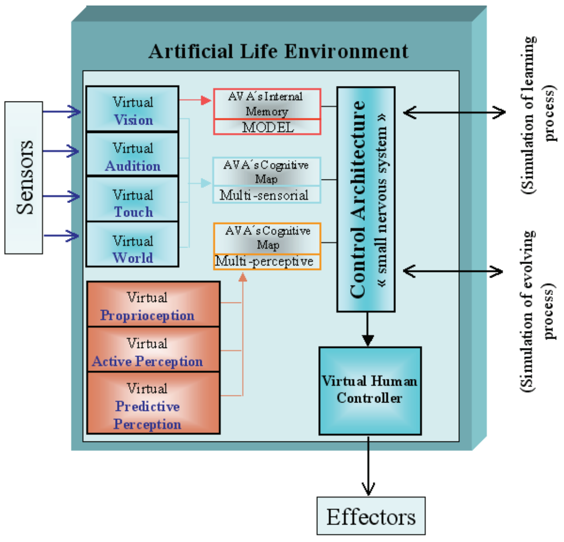


Figure 9: A schematic representation of our *ALifeE*. Virtual Vision discovers the VE, constructs the different types of Perception and updates the AVA's Cognitive Map to obtain a multi-perceptive mapping. Then the Control Architecture uses both the "cognitive maps" and the "memory model" to interact 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 signal processing. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, 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, August 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

EUROGRAPHICS 2006

Populating Virtual Environments with Crowds: Rendering Pipeline Optimizations

P. de Heras Ciechomski and D. Thalmann,

Swiss Federal Institute of Technology, VRLab, Lausanne, Switzerland

Abstract

Rendering a large crowd in real time requires optimizations at all levels of a graphics rendering engine. This tutorial notes goes through high-level optimizations ranging from the pipeline architecture down to details concerning data transmission and specific graphics hardware considerations.

1. Introduction

The structure of the notes is as follows:

Section 2 overviews the pipeline to get a high level prespective of the different rendering passes
Section 3 introduces gemetrical rendering methods
Section 4 goes into depth about deformed geometry rendering and animation caching schemes
Section 5 introduces GLSL hardware shaders and several accelerations and memory transfer considerations
Section 6 describes static mesh rendering
Section 7 concludes with results

2. Pipeline

The goal of the real-time crowd visualizer is to render a large crowd according to their current simulation state, providing the position, orientation and animation for each individual. System constraints are believability, real-time updates (25 frames per second) and a number of digital actors ranging in the tens of thousands. Actors are made believable by varying their appearance (textures and colors) and animation. Their graphical representation is derived from a *template* which contains all the possible variations. Using only a limited set of templates, a varied crowd is achieved, leading to considerable time savings for designers.

2.1. Templates

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

- a mesh with at least three static pre-computed levels-of-detail (LOD)

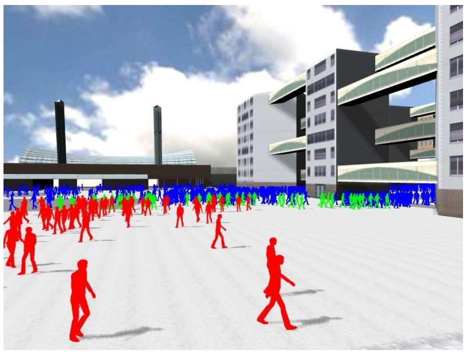


Figure 1: *Rendering fidelity levels exposed: dynamic meshes in red, static meshes in green and billboards in blue.*

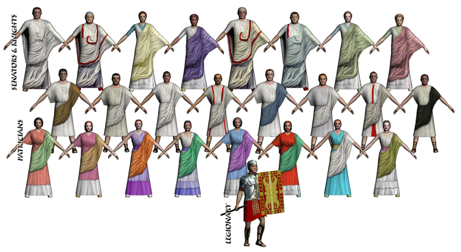


Figure 2: *The ERATO templates with combinations of textures and meshes.*

- one or several base textures in gray scale (except or the skin) identifying color modulation areas (pants, shirt, hair etc.)
- a skeleton with optional complexity reduction using pruning (Section 4.1)
- a set of animations as skeletal orientations

Each human in the visualization system is called an *instance* and is derived from a template. Individualization comes from assigning a specific base texture and a color combination for each identifiable region and an animation which can be a combination of several interpolated key frames.

2.2. Rendering Passes

The rendering pipeline advances consecutively in four steps starting with culling which determines visibility and rendering representation or *fidelity* for each simulated human, as shown in Figure 1. By re-using the information stored in existing navigation graphs of the simulation system, culling is not done on each individual but at node level, thereby determining fidelities for a set of characters at once. Only nodes that are between fidelity levels where each individual has to be inspected are processed more precisely. During culling, humans are separated into three vectors, storing their indices for the coming rendering steps, which ensures that each visible human of a specific fidelity is visited only once. Figure 3 shows how the rendering fidelities are partitioned from the viewer.

The second step in the pipeline is rendering of dynamic meshes (Section 4) which is the most detailed *fidelity* and can play back interpolated animations based on skeletal postures. An animation is constructed based on the walking style of the human and its speed as in [PdHCM*06], yielding a smoothly interpolated walking animation adapted to the simulation system's positional updates. A hardware *vertex shader* and *fragment shader* are used to deform and render the human on the graphics card.

Static meshes (aka *baked* or pre-deformed) constitute the second rendering *fidelity* which keeps a pre-transformed set of animations usually in the range of two or three animations, using the lowest resolution mesh of the deformed ones in the previous step. By pre-computing the deformations substantial gains in speed are achieved as presented in Section 6.

The third and final rendering fidelity is the billboard which uses a simplified scheme of sampling and lighting. World-aligned billboards [MH02] are used, with the assumption that the camera will never hover directly above the crowd. Thus, only the sample images at waist level of the character are needed. In this case, each template is sampled at 20 different angles, for each of the 25 key-frames composing a walk animation. When constructing the resulting texture, the bounding box of each sampled frame is detected to pack them tightly together. When rendering billboarded pedestrians, a specificity of the technique is to apply cylindrical lighting instead of using normal maps: each vertex normal is set to point in the positive z-direction, plus a small offset on the x-axis, so that it points slightly outside the frame, see [PdHCM*06]. The lighting intensity is then interpolated for each pixel in the fragment shader.

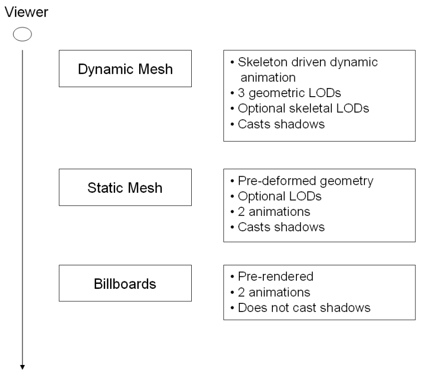


Figure 3: *The fidelities seen from the top.*

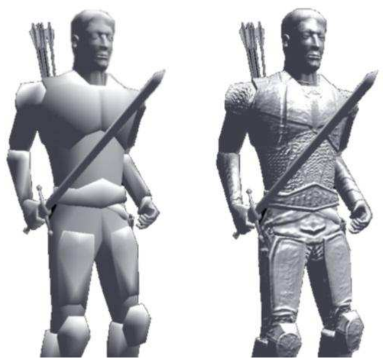


Figure 4: *Geometry with Gouraud shading (left) and normal mapping (right). The same amount of triangles is present in both.*

3. Geometrical Methods

Geometrical rendering of virtual humans uses a triangular mesh, which is textured and lit, see Figure 5. The thesis concentrates on meshes where the triangle count is in the range of a few hundred to a thousand. The mesh is surrounding a skeleton with no more than 80 or so bones that drive the vertices in the mesh to move and bend. This technique is known as skeletal subspace deformation or "skinning" (Lander [Lan98]). Adding more triangles is often not interesting as they will merely increase the computational size and not add enough detail where it is due. It is then better to use a technique such as normal mapping (Cohen *et al.* [COM98])

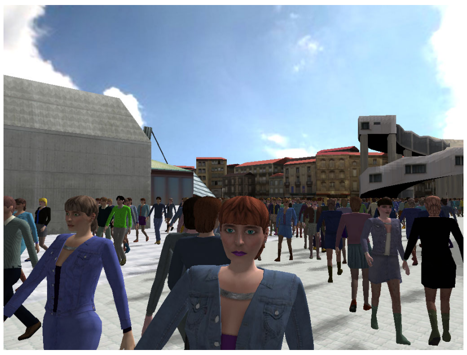


Figure 5: *Hybrid of dynamic and static geometry and billboards. The closest character consists of dynamic mesh connected to 78 bones, which makes it possible to have an expressive face.*

to add more detail in a mesh, thereby keeping a low triangle count and still keeping a high detail level. As it is shown in Figure 4 a normal mapped mesh with 1000 triangles, has the same representational fidelity as the one using one million, by the usage of surface extraction details which are then saved in the normal map texture. A more thorough description of the normal mapping can be found in [COM98], concluding that a virtual human can be represented with good detail using 1000 or so triangles.

The first distinction made is between a dynamically deformed and a static pre-computed mesh. In the latter case all vertex positions and normals are pre-computed for a full animation cycle such as walking, while for dynamic meshes, any animation can be used such as inverse kinematics, animation blending and mixing or any dynamically updated animation. These two problems can be attacked using a CPU based approach as well as a graphics hardware approach utilizing programmable shaders.

4. Deformed Meshes

A deformed mesh rendering approach of a virtual human is based on having a skeleton that drives the deformation of the human. A good survey of different deformation algorithms can be found in the Master thesis of de Heras Ciechomski [dHC01] and in the work of Kavan *et al.* [Kt05]. The algorithm used in this work is called Skeleton Subspace Deformation (SSD), where each vertex is deformed by the weighted transformation of one or several attached bones or joints. The equation that deforms each vertex is

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

where v(t) is the deformed vertex at time t, X_i^t is the global transform of bone i at time t, X_i^{-ref} is the inverse global transform of the bone in the reference position and v^{ref} is the vertex in the reference position. It is possible to pre-compute the transform in the reference position, which requires as many pre-computations as there are affecting bones per vertex. Instead what is usually done is that the transform pair $X_i^t X_i^{-ref}$ is concatenated.

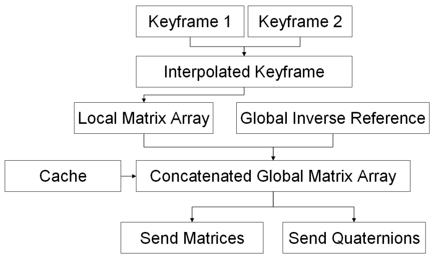


Figure 6: *Deformed mesh operations, from top to bottom. First minimum two animation key frames are interpolated, which results in a new quaternion key frame that is then converted into matrices. The matrix key frame is made global by concatenating it using the hierarchical information and is multiplied by the inverse reference posture. In the case of an animation caching scheme the previous operations are not done but the resulting matrix array is fetched instead. Finally the matrix array is sent to the graphics card for deforming the mesh in the hardware* vertex shader.

A deformed mesh approach has three costly operations (see Figure 6). The first is to animate and update the skeleton itself into a specific position and can require several matrix and quaternion operations. The second operation is the deformation of geometry such as vertex positions, normals and perturbed texture spaces in the case of normal maps. Finally the third cost is in the rendering of the character triangle by triangle on the graphics card and in the case of using color variety this also poses consideration and will be discussed in more detail in tutorial notes on variety.

In this section a skeleton simplification schema is described, followed by a CPU based approach for geometrical rendering, which is then enhanced by a GPU based acceleration.

4.1. Skeletal Pruning

There are two main reasons to optimize the skeleton that deforms characters, one is speed and the other is memory space. In the database of animations a template holds all the possible animations that it can perform in the form of quaternion tables. A table entry is an entire animation, while a row in this entry is a key frame. A key frame consists of all the

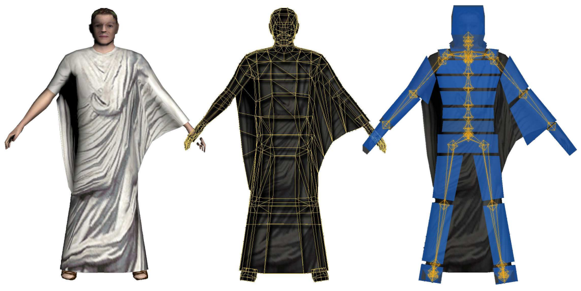


Figure 7: *Textured model (left), triangle outlines (middle), underlying skeleton as oriented bounding boxes (right).*

joint transforms relative their hierarchically preceding transform or father, in quaternion form. This is a performing form of storage to get an in-between frame of animation by spherically interpolating two neighboring key frames, if the animation was recorded using a low sampling frequency. Quaternions are also performant for transitioning between animations such as walking and running or any two such linked animation states. Since most dynamic mesh characters will be interpolating between animations and doing transitions, the animations should be stored in form of quaternions. Only in the special case of no interpolation or in-betweening is done, it is of interest to keep all animations as matrices. In that case all key frames would be pre-concatenated global transforms readily available to the deformation pipeline such that the transform pair $X_i^t X_i^{-ref}$ is stored. From here on it is assumed that key frames are stored as quaternions.

Besides the concept of rendering fidelities previously introduced, there are also sub-fidelities; Dynamic characters can have moving fingers, expressive faces, even moving eye sockets; This is possible to see when up close to the character but virtually impossible at a distance. A character with an expressive face can have 78 bones of which 45 are used exclusively for the bones in the hands and the face. It is of interest to decrease the number of bones or to use skeletal levels of details in the case these details are no longer visible. The most straightforward method is to cut off or to *prune* bones or entire sub-trees of bones that are at the end of the skeleton. The reason for pruning at the end of the skeleton is that otherwise it would be necessary to compute the in-between transforms that were removed in the reduction. For example, reducing three bones in the middle of the spine of a highly detailed skeleton, would affect the transformation in a hierarchical manner where the bone transforms are expressed as

$$X_i^0 = X_1^0 X_2^1 \dots X_i^{i-1} \qquad (2)$$

Where X_i^0 is the transform that changes coordinate system for a point described in reference system i to reference system 0 (the root). If bones are removed in this matrix stack one would have to compute their impact on the final matrix, which is their concatenated transform. If bone j is removed it would be necessary to decide on a heuristic for vertices that were connected to it, so as to choose a new bone to connect to or to simply discard its influence, depending on its weight. Also the transform X_{j-1}^j is lost and will have to be concatenated into the next bones, either in run-time or be pre-computed for all key-frames, for all animations. This would create extra storage for each removed bone, something that raises the complexity of the animation engine and requires low-level changes at storage structures. These extra calculations and extra storage requirements can be completely circumvented if only the removal of complete end-hierarchies is considered, which means that for the hand, only the wrist would be left and for the face only the bone deforming the neck would be the one left. This is interesting since it means all transforms would simply be dropped, as no bone is left at the end of the hierarchy. It also simplifies the heuristic for re-assigning bones to vertices that were connected to the excluded bone, as the bone on top of the removed hierarchy is used exclusively for its transforms. This is how skeletal pruning works and requires no extra transforms during run-time or any storage for pre-calculation. The only requirement being a re-assignment of influencing bones for the vertices affected by the ones that were pruned.

Pruning is now used in four separate stages in the engine: when the mesh is loaded a pruned version of it is created that accounts for the re-assignment of bones. When an animations key is fetched from the animation data base only the active quaternions are retrieved, consequently only the active joints are interpolated when transitioning between two animations or performing a blending. Finally, before deforming the mesh the concatenated matrix stack is only calculated for the active bones. The optimization benefits are presented in the Results section.

4.2. Vertex and Animation Caching

Once the animation data is computed and stored in a concatenated matrix stack, the mesh is ready to be deformed and displayed. Doing these operations on each and every individual character in the scene each frame, and even with the use of skeletal pruning, is computationally expensive. One approach of reducing the cost is by simply not doing any deformations and re-using the previous calculation, which is the basis of a caching scheme. When a character is close to the camera his animations are updated frequently up to a rate of 50 Hz and when far away from the camera the update rate is at minimum 4 Hz. Equation 1 can be written as

$$v(t) = \sum_{i=1}^{n} X_i^{tDef} v^{ref} \qquad (3)$$

Figure 8: *Scene from the game Doom 3^{TM} by id Software, with extensive use of vertex and pixel shaders.*

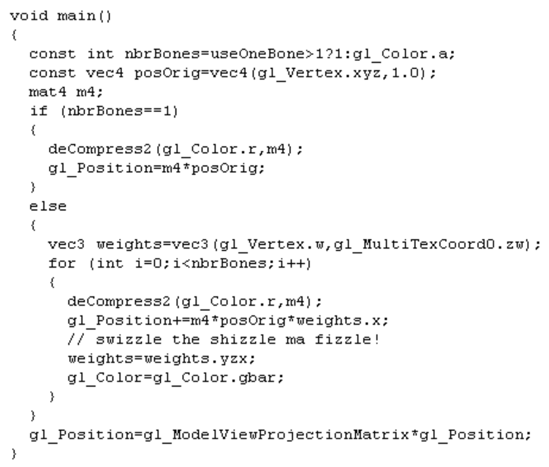
where $X_i^{tDef} = X_i^t X_i^{-ref}$ is the concatenated form, which is stored in the animation cache. This is a considerable decrease in skeletal animation updates. If a software deformation pipeline of the character is used, caching the deformed vertices and normals is of interest. Normal deformation uses the same expression as Equation 1 but instead of the full 4×4 matrix resulting from the multiplication of the pair $X_i^t X_i^{-ref}$ only the rotational part is used, which is the upper 3×3 matrix. This is done either explicitly by using a smaller matrix or by setting the fourth component of the normal to zero, thereby achieving the same result. When working in a software deformation mode, it is better to use the rotational matrix explicitly. There might be a contra-indication to this if SSE matrix multiplication are used, which is optimized for 4×4 floating point operations, see [Str02]. By deforming the geometry only at the same intervals as the animation, computations can be greatly decreased, see de Heras Ciechomski *et al.* [dHCSMT05] and is shown in the Results section of this chapter. The reason for separating animation and vertex caching, is that the mesh can change levels of detail faster than an animation update. The caching scheme can also be re-used for hardware deformation shaders, but only for the animation update, since that is the only part being computed on the CPU. In a hardware shader the mesh is deformed by a vertex program and the resulting transformed vertex cannot be retrieved so it cannot be cached. However this is not a problem as hardware shaders are more performant as is shown in the next section.

5. Shaders

This section emphasizes new graphics hardware with programmable shaders. Since the engine is using OpenGL, the focus is on the OpenGL Shading Language (GLSL) aka GLSLang. Shaders replace the normal fixed function pipeline of OpenGL, which does vertex transformations and per-pixel rasterizations, called *vertex shaders* and *pixel shaders* also known under the name of *fragment shaders*. These programs are executed on the graphics hardware prior to being written in a C-like language and compiled at runtime by the graphics card driver. This on demand compilation means that benefits can be gained from the newest possible optimizations, in compilation and execution from the manufacturer. It also makes the programs portable to any graphics hardware, which implements a similar shader model, differing for example in the support for *while*-loops, *if*-statements, *for*-loops and data transfer modes. A program must contain one of each program type, where the vertex shader is the first to be executed and can communicate with the pixel shader in a one-way fashion. The reason for this simplified communication model is that hardware is equipped with parallel vertex and pixel shaders, so the programs are set up to be easily parallelized, as not to wait for the previous stage to complete for example. Some graphics hardware implement the *if*-statement as two forked executions and depending on the final boolean value the correct fork is chosen for changing the data, all in the name of parallelism. This might change as pixel shaders start to communicate with the vertex shader using vertex textures as is already possible on some graphics hardware. Discovering the power of graphics shaders one sees the benefit of switching to using them and the engine is completely relying on shaders. Doing a clean break from a software deformation path is beneficial in terms of less code paths to maintain. A demo from ATI was released with crowds running on a steep terrain (Gosselin *et al.* [GSM04]), showing how new graphics cards can handle deformation of geometry.

A vertex shader gets information from the per vertex stream such as position, texture coordinates (which can be several, usually up to eight), normal, color and vertex attributes. Instead of using multi-texture coordinates directly for data sending, vertex attributes wrap the necessary steps to do so, hiding the details from the developer. This can be misleading and that is why it is preferable to use standard OpenGL attributes such as color, multi-texture coordinates and so forth for sending the necessary data to the vertex shader. Other data that is re-used over several vertices are called *uniforms* that can be for example, the matrices used for the weighted deformation of vertices or the incoming light direction. When a vertex shader is finished with processing, it sends the data to the pixel shader in form of *varying* variables, which are interpolated over the triangle. Some varyings that are usually sent, are the interpolated surface normal to be used with the incoming light direction, in case of Phong shading or the interpolated intensity scalar in case of Gouraud shading (Gouraud [Gou71]).

The *pixel shaders* or as it is sometimes called - the *fragment shader* - is responsible for rasterizing the final pixel, taking the necessary information from the vertex shader interpolated variables (varyings) and optional user data (uniforms). In this stage many different pixel operations are done such as alpha checks, reading texture data from one or sev-



```
void main()
{
  const int nbrBones=useOneBone>1?1:gl_Color.a;
  const vec4 posOrig=vec4(gl_Vertex.xyz,1.0);
  mat4 m4;
  if (nbrBones==1)
  {
    deCompress2(gl_Color.r,m4);
    gl_Position=m4*posOrig;
  }
  else
  {
    vec3 weights=vec3(gl_Vertex.w,gl_MultiTexCoord0.zw);
    for (int i=0;i<nbrBones;i++)
    {
      deCompress2(gl_Color.r,m4);
      gl_Position+=m4*posOrig*weights.x;
      // swizzle the shizzle ma fizzle!
      weights=weights.yzx;
      gl_Color=gl_Color.gbar;
    }
  }
  gl_Position=gl_ModelViewProjectionMatrix*gl_Position;
}
```

Figure 9: *Minimal deforming vertex shader.*

eral textures, coloring the pixel and other frame-buffer operations. Techniques such as normal mapping, multi-texturing and lighting equations for materials are executed in the pixel shader.

The next section continues with shaders used to deform character meshes and color them according to their modifiers.

5.1. Deformation Shaders

After having shown the basics for deforming a mesh using a skeleton in Section 4 and how to optimize the calculations for a software pipeline, the possibilities of the hardware graphics processing unit (GPU) are explained. First it is described how the character is deformed using a vertex shader and then how lighting and texturing is applied in the pixel shader. After this how to send data to the graphics card in an optimal way is elaborated. A typical vertex shader used is listed in Figure 9.

The deformation shaders use *Shader Model 3* (SM3) which includes *if*-statements, *for*-loops and other logical branching structures. These branches are good if it is known for example, that the character is so far away, that he only needs to use one influencing bone per vertex, which can be seen in Figure 9. In this listing a check is made if the mesh uses more than one bone to influence each vertex, stored in the *nbrBones*-variable. In that case, the costly *for*-loop is ignored, as is the initial zeroing of variables and the scalar multiplications that go with it. An *if*-statement can be circumvented, by compiling two programs for each separate case and switching them when necessary from the CPU-side. Since an *if*-statement shouldn't cost more than 2 clock cycles, on nVidia hardware of the 6800 series and up, this was not considered. One might wonder why the *useOneBone*-variable in Figure 9 is stored in the alpha color component (glColor.a) and this thirst for knowledge is quenched in the following section.

5.2. Memory Transfer Optimizations

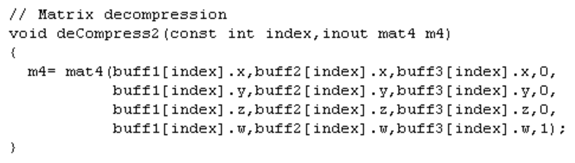
One of the most important optimization areas of a high performant shader program is data transfer, such that the data is easy to digest which means the operations that work on it are fast and that the amount of data sent is minimized. The data needed to be sent to the vertex deformation shader is as follows:

- vertex position, normal and texture coordinates (*attribute*)
- an array of deformation matrices shared by all vertices (*uniform*)
- indices and weights of deformation matrices (*attribute*)
- a variable indicating if more than one bone is used (*uniform*)

Before embarking on this data sending journey, one has to be aware of the limitations of the graphics card, which has a specific memory allocation and transfer model. The atomic structure of the graphics card is based around four floating point vector entries of a total of 256 such atomic entries, giving a grand total of 1024 aligned single float scalars. This holds for the graphics cards such as nVidias 6800 and 7800 series and will most probably expand in size with newer and more powerful cards. The bone matrices for the deformation equation need to be sent (Section 4), being 4×4 floating point (float) vectors or 16 single float scalars. In GLSL there is a built-in variable called Matrix4 which suits the need of characters of 60 bones or less. Even though 60 matrixes make up only 240, 4 float vectors, since all variables that are used in a shader, share this same data window, all of the 256 atomic vectors cannot be used. Variables such as the OpenGL model view matrix, projection matrix, lights, active color, and others that are utilized by the shader are loaded into these 256, 4 float vectors. That is why the use of bones should be restrictive, by for example cutting the mesh into sub-meshes, where each is only affected by a maximum of 60 number of bones, or to use skeletal pruning as is described in Section 4.1. The other way of getting around this memory problem, is through compacting the memory required for the transfer of the bone matrices. In any case it is a problem that needs a solution as some applications use humans with around 80 bones with detailed finger and facial movements. The following section details data compaction procedures.

5.2.1. Matrix Compression

The first approach to matrix compression, is to have a look at the vector of homogenous transforms X_i^{tDef}, that are interesting to send, namely



```
// Matrix decompression
void deCompress2(const int index,inout mat4 m4)
{
  m4= mat4(buff1[index].x,buff2[index].x,buff3[index].x,0,
           buff1[index].y,buff2[index].y,buff3[index].y,0,
           buff1[index].z,buff2[index].z,buff3[index].z,0,
           buff1[index].w,buff2[index].w,buff3[index].w,1);
}
```

Figure 10: *A shader function to decompress a 3 row matrix.*

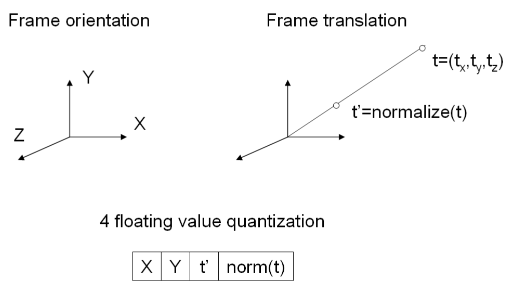


Figure 11: *Quantization of a transformation matrix into 4 float values.*

$$\begin{bmatrix} x_x & y_x & z_x & t_x \\ x_y & y_y & z_y & t_y \\ x_z & y_z & z_z & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

which by the first look doesn't need a full four by four matrix and using a Matrix4 variable seems like a waste as only 12 float values are needed. This entire matrix can be compressed down to a single 4 float vector (one atomic structure), by using various assumptions on the data. There are however other intermediate solutions available.

The first solution is to send the three rows in three vectors as in Gosselin *et al.* [GSM04]. This way a maximum of 80 bones is reached, which is an improvement of 25% over using straight Matrix4 variables. A good point of this method is that compression on the CPU is easy and fast as well as the GPU decompression, something that is appreciated in terms of code simplicity and maintenance and has also reached the necessary size of 80 bones for the character. The shader function decompressing the matrix can be seen in Figure 10, where *buff1*, *buff2* and *buff3* are *uniform* vectors holding the 3 matrix rows. The discussion could stop here but if there is a need to send for example more bones or if the data is shared by other structures more elaborate approaches are necessary.

The matrix in Equation 4 is orthonormal, which means that each column is orthogonal to the others and that each column has a unit norm. This means that the z-column can be determined by taking the cross product of the x and y-columns or vectors *e.g.* $z = x \times y$. This brings the size down to 9 floats, which is closer to the goal of decreasing the size per bone matrix, but still takes three float vectors because of the alignment to four floats. Practically this creates a space improvement of 25%, which is the same as the previous approach, even though three floats less are used. Further investigation is necessary.

By re-writing the attributes in other representational spaces, more valuable floats can be saved. A normalized three sized vector can be compressed into one float value, by using a discretization of 8 bits for each component, multiplying them into a 32 bit float value. This of course makes the assumption that the graphics hardware is able to handle 32 bit floats internally, which is the case for nVidia^{TM} hardware. With this representation two floats are used to describe the x and y vectors but the translation still needs to be represented. A point in space can be described as a length and a normalized direction vector, which is the same as factoring out the length from the point. Thus encoding the length of the translation in one float value and the direction vector in a second is possible. This gives a grand total of four float values, such that the entire transformation matrix is encoded in one atomic structure, as shown in Figure 11. It is an improvement of 75% in space used to describe the transform. It does require more operations on both the CPU side and on the GPU side, but these operations can be written down as cross-products, with constant vectorized expressions. The only drawback of this method is that the rotational space is quantized quite roughly and thus a very smooth resulting animation should not be expected.

Normally the first solution using a decomposition in three rows using three atomic structures is the one to prefer, as it's the easiest to implement and works well for characters with less than 80 bones.

5.2.2. Quaternion Compression

If the graphics engine is representing every transform using quaternions and a separate translation, instead of matrices, there is another way of sending the data to the graphics card. Instead of using quantizations of columns going as far down as one atomic structure for the entire transform, quaternions can be used, written as four real values,

$$\hat{q} = (q_x, q_y, q_z, q_w) \tag{5}$$

As the translation is using only three float values without vector compression there are, two atomic structure vectors, which gives a 50% space improvement. The precision is also higher than using a matrix and encoding it into a rough vector compression, as was described above. However there is a slight drawback, because a matrix-to-quaternion transformation is needed on the CPU side and a quaternion-to-matrix transform in the vertex shader (unless direct rotation

```
// Quaternion decompression
void deCompress(const int index,inout mat4 m4)
{
  const vec4 q=buff1[index];
  vec4 x=vec4(q.x*q.x,q.x*q.y,q.x*q.z,q.x*q.w);
  vec4 y=vec4(      0,q.y*q.y,q.y*q.z,q.y*q.w);
  vec4 z=vec4(      0,      0,q.z*q.z,q.z*q.w);
  m4[0]=vec4(1-2*(y.y+z.z),2*(x.y-z.w),2*(x.z+y.w),0);
  m4[1]=vec4(2*(x.y+z.w),1-2*(x.x+z.z),2*(y.z-x.w),0);
  m4[2]=vec4(2*(x.z-y.w),2*(y.z+x.w),1-2*(x.x+y.y),0);
  m4[3]=buff2[index];
}
```

Figure 12: *Decompressing the quaternion in the vertex shader.*

with the quaternion is used). These conversions are a bit more costly than using a matrix, as can be seen in Figure 12 where the decompression is listed. The conversion to a matrix costs 20 multiplications and 12 additions, while on the CPU the conversion from matrix to quaternion costs on average one square root operation, 4 multiplications and 10 additions, compared to no such operations for the matrix sending path. There are a number of possible accelerations to using quaternions and all of them have not been explored. The least costly way of sending them is if the engine keeps rotations in quaternion form through all stages of computations, even keeping the global deformation transforms as quaternion and translation pairs, instead of matrices. The shader would also need to transform the vertex using this decomposition.

5.2.3. Mesh Segmentation

If the mesh is segmented into sections, the step of compressing and decompressing the matrices can be completely skipped. Each section of the mesh uses a subset of the bones and the necessary bones for each sub-mesh are sent. There is no good metric for how many sub-meshes would be needed, or how to choose which triangles are contained within a specific sub-mesh. Both problems are linked and the more sub-meshes are made, the more drawing calls and bone transformation uploads to the graphics hardware have to be done. Mesh segmentation was never implemented because of the overhead it would presumably generate; for all sub-meshes, modified *uniform* variables would have to be re-sent for example.

5.3. Geometry transmission

Each character in the engine consists of exactly one mesh and needs to send for each vertex its position, texture coordinates, normal, indices to deformation matrices and weights to these matrices. This data can be sent in attribute vectors, but if more control is preferred it is more convenient to use the values embedded in the standard components. The per vertex data, is listed as follows:

- glPosition (p_x, p_y, p_z, w_1)
- glColor (i_1, i_2, i_3, w_{nbr})

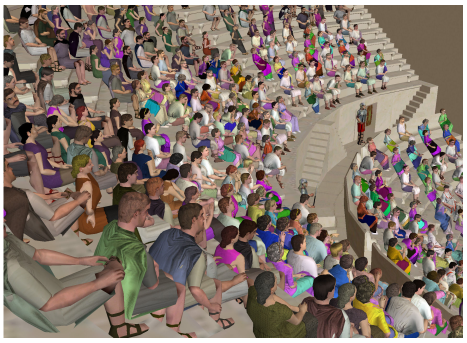


Figure 13: *ERATO color variety applied to the crowd using the pixel shader.*

```
uniform sampler2D    baseTex; // texture unit 0
uniform sampler2D varietyTex; // texture unit 1
uniform float     varietyRow;

void main()
{
  N=normalize(N);
  vec4 texel=texture2D(baseTex,gl_TexCoord[0].st);
  texel*=texture2D(varietyTex,vec2(texel.a,varietyRow));
  texel.a=1.0;
  const float lightIntensity=max(0.0,dot(L,N))*0.8;
  const float ambient=0.4;
  gl_FragColor=(lightIntensity+ambient)*texel;
}
```

Figure 14: *Pixel shader doing lighting and color variety modification.*

- glTextureCoordinate (u, v, w_2, w_3)
- glNormal (n_x, n_y, n_z)

Each vertex needs matrix indices (i_1, i_2, i_3), matrix weights (w_1, w_2, w_3), a normal (n_x, n_y, n_z), a position (p_x, p_y, p_z) and a texture coordinates (u, v). By assigning these values into the standard OpenGL components as listed above, they can be stored in a display list, which will for most cases put the geometrical data in the graphics card's memory. The first reason for having no more than three influencing bones is that more than this number is usually not needed, so even if the character LOD might have some vertices that are connected to more bones, the least influencing ones are discarded and the weight sum is re-normalized to one. The second reason for having three bones is that it neatly fits inside the standard data, just that their largest versions are used, which is the four dimensional texture coordinate, color and position. When the position is sent re-setting the fourth component to 1.0 before transforming it has to be taken into account, as seen in Figure 9.

5.4. Pixel Shader Usage

All geometrical characters, no matter whether they are dynamic or static, send their interpolated normals to the pixel

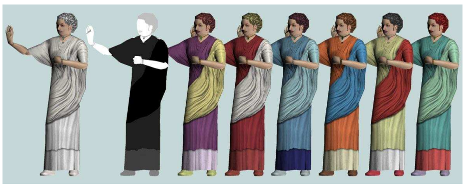


Figure 15: *Geometrical texture variety.*

shader for a diffuse Phong lighting effect. In the pixel shader per-texel accurate color modification is applied, using a texture as a look-up table (LUT), which is explained in depth in [dHC06] and is similar to the way it is done in the thesis of Dobbyn [Dob05]. Having a look at Figure 14, the LUT-texture is kept in the varietyTex variable, and that varietyRow determines from which row to sample from in the LUT. The pixel shader is quite light weight in comparison to the vertex shader, which is a good trade-off as there are generally more pixels than vertices per virtual human. The pixel shader usage is more extensibly described in the tutorial notes on variety.

6. Static Meshes

Until now the discussion has been almost exclusively dealing with deformable meshes that can play back any animation. Now the focus will be on rendering geometry in the case there is a very limited set of animations. This limited amount of animations is usually between 10 and 15 and has the constraint that the geometry isn't too complex in terms of number of triangles, so as not to take large amounts of memory or become prohibitive in terms of pre-processing time. A static mesh is a pre-computed version of the deformed mesh, where the vertices and normals have been stored for the selected animations. These geometrical snapshots are called key frames, which are usually using a cyclic animation and need to be computed with a software pipeline. Even if this thesis advocates a complete change in the pipeline to use shaders exclusively, there are still uses for software deformation. Storing these pre-deformed meshes in OpenGL display lists, the data transfer to the card is minimized as it includes doing one drawing call for the entire object. The geometric data will most probably already reside in the graphics hardware memory.

Storing the data on the graphics card can be done in numerous ways. The most simple of them mentioned above is through the use of display lists, which in itself is usually a good performing storage. If an even faster rendering is desired, on some graphics cards it pays off to create optimal storage for vertex data. A first good step is to use indexed geometry, resulting in less storage space and profiting from vertex caches existing on graphics cards for a few years already. If a further increase in speed of the storage and its transmission is required, the vertex data and the triangles indexed need to be close to each other so that they fit nicely into the vertex caching scheme as in Hoppe [Hop96]. Indexed geometry can be put in interleaved arrays which was tried, but are equivalent in speed, see de Heras Ciechomski *et al.* [dHCUDC04].

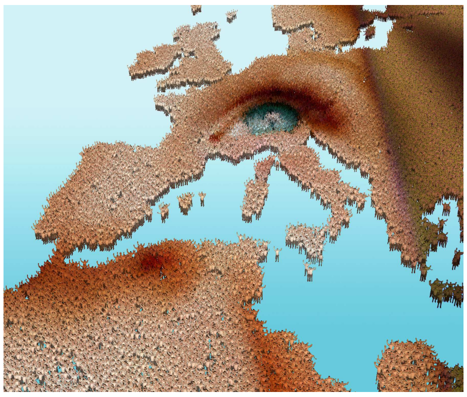


Figure 16: *CrowdBrushed Europe, containing 30 000 animated humans forming the landscape, from [UdHCC05].*

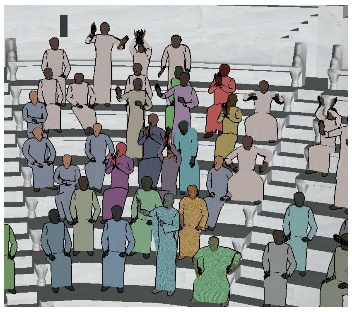


Figure 17: *Cartoon shading of the crowd.*

Approximately the data size of unpacked models ready for rendering is computed in the following manner. Each triangle has three vertices, three normals and three texture coordinates, all together with the size of approximately one hundred bytes. For each frame, the size requirement in bytes is one hundred times the number of triangles. An average human that consists of one thousand triangles requires around one hundred kBytes per key frame. Since the animations are sampled at 32 fps, this amounts to approximately 3.2 megaBytes per second.

The benefit of using display lists is that one can take advantage of the full OpenGL pipeline, without any changes to lighting, see Figure 17. The characters look well from all camera directions and zooms, as opposed to billboards which are usually low resolution and need pre-processing for all different camera angles. The memory usage is lower than with billboards; however more polygons need to be rendered. The data storage requirements on disk are very low,

Figure 18: *Spotlight on the crowd.*

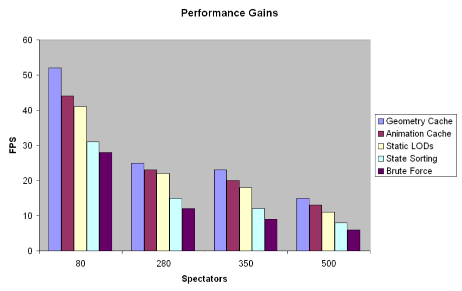


Figure 19: *Performance impact of animation and geometry caches on a software pipeline deformation.*

since only a basic mesh and animations of the skeleton are stored.

7. Results

This section sums up the results of the previously described rendering optimization techniques with regard to crowd rendering.

7.1. Caching

The testbed for experimentation of caching optimizations is the following: an Intel Pentium IV running at 3 GHz with 1 GB of main memory and an nVidia Geforce 6600 with 256 MB of onboard DDR2 memory. The scene consists of an Odeon filled with a crowd of Romans watching and reacting to a play. Each human has his own individual behavior in terms of different animation clips being interpolated and transitioned, thus creating animations on-the-fly. Dynamically deformed meshes are used with a software pipeline so that the resulting vertices and normals can be re-used in the next frame in a dedicated cache per individual. This is also described in de Heras Ciechomski *et al.* [dHCSMT05].

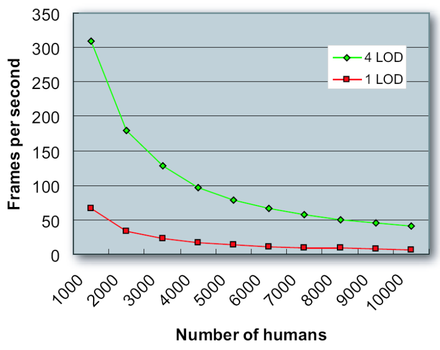


Figure 20: *Using 4 levels-of-detail meshes greatly improves performance.*

7.1.1. Geometrical Caching

The first test caches vertices and normals after each mesh update depending on the distance to the viewer. It is added on top of a LOD scheme and the resulting performance is shown in Figure 19, where it goes under the name of animation caching. Compared to a pure LOD scheme geometrical caching improves the performance by 27% as can be seen in the top bars when done on a crowd of 80 characters. As the number of characters increase the impact of the caching increases since the bottleneck becomes the mesh deformation more and more, until it become 36%. For a software pipeline the improvement in performance becomes conservatively 30% on average.

7.1.2. Animation Caching

Animation caching works by storing the concatenated form of the matrix array (see Figure 6) before it is sent to the vertex deformation phase. Its impact on performance compared to LOD meshes can be seen in Figure 19 and amounts to 7% for a low amount of meshes and 18% for a high amount of meshes, being a 10% on average.

7.2. Data Transmission

Using the matrix compression technique described in Section 5.2.1 the high bone count humans are possible to send to the hardware shader, which was not possible without it. Putting the entire human inside of a display list also speeds up rendering; however no tests were done with the data outside of the display lists.

7.3. Skeletal Pruning

When bone pruning is done on the skeleton, by removing bone operations from the animation interpolation and concatenation, its impact has global and local effects. Locally the effects are directly quantifiable through a mathematical

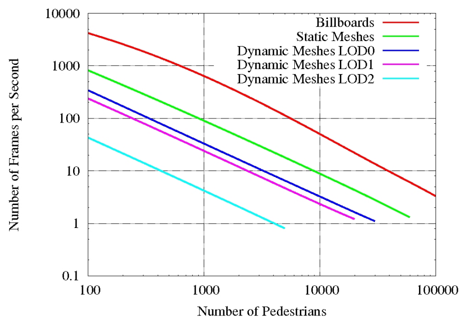


Figure 21: *Three rendering fidelities, dynamic meshes using 3 geometrical LODs, static meshes and simplified billboards.*

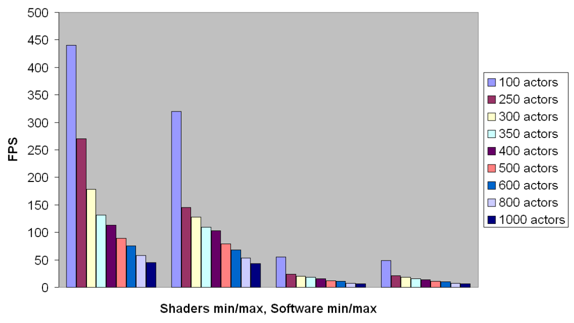


Figure 22: *Dynamic mesh deformation using hardware shader programs in comparison to a pure software deformation pipeline.*

formulation as the number of bones removed is linearly related to the number of operations done in form of quaternions interpolations and matrix multiplications. Globally the effect can be less or more depending on the graphics card as it does the vertex interpolation step and the final rendering. If the final rendering is slow, for example if the number of vertices increases, the number of transforms will depend on the ratio between the number of triangles in the hand and face to the rest of the body. When the vertices in the hand use less bones because of the pruning step (which collapses them onto the single wrist bone) the amount of transforms done in the *for*-loop in Figure 9 also decreases by minimum 2 or three times. Globally the amount increases conservatively minimum by 10% in fps on a graphics card of the nVidia 7800 series even though theoretically the number should be higher. Since the amount of bones sent to the graphics card is still the same as before the pruning (78) even though they are not filled with any data, this could maybe also impact. The reason for not changing the number of bones sent to the card is that it would require a new data type in form of a skeletal LOD in the system.

7.4. Static vs Dynamic Meshes

Static meshes can be used in scenarios when a very limited set of animations is used for the entire animation repertoire. They can also be used as an intermediate rendering fidelity between dynamic meshes and billboards as is proposed in this work. To know their performance compared to dynamic meshes a scene of up to 100 000 humans is used. The static meshes are built using 2 animation cycles, one walking and one idle animation, while the dynamic meshes use an animation bank of 1000 animations played back on the mesh, so they have the extra cost of updating the animation which is necessary to include having real-world statistics. Dynamic mesh LOD 0 in Figure 21 (blue) has the same triangle complexity as the static mesh (green) which is consistently three times faster. Static meshes definitely have a place in a crowd engine because of their speed as long as they are used with a limited set of animations.

7.5. Shader Programs vs. Fixed Function Pipeline

The next test is comparing the performance of shader programs versus a fixed function pipeline and is executed on an AMD64 4000+ with 2GB of memory and an nVidia SLI 6800 Ultra graphics card. In the tests one template is used consisting of 8 geometrical LODs, starting at 1026 triangles down to 490 triangles. Two animations are mixed at quaternion level using skeletons of 33 bones each and consisting of three (512 pixels wide and high) textures, for the texture variety. All humans are in view and the average of minimum and maximum frames per second is shown in Figure 22, where the maximum fps corresponds to a view far from the crowd so that all humans are 490 triangles, while the minimum fps is obtained by zooming in on the crowd until the user is close enough to engage the most detailed geometrical meshes. From the graph can be deduced that an average speed-up of 700% is gained from using programmable hardware shaders. This acceleration can also be due to how data is sent to the graphics card since for vertex shaders all data is sent in one OpenGL display list call for each human. This gives an advantage in terms of less rendering primitive calls comparing to the software approach which does up to 16 glDrawElements calls for 8 coloring areas even with a mesh subdivision approach as described in the tutorial notes on variety.

References

[COM98] Cohen J., Olano M., Manocha D.: Appearance-preserving simplification. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques, SIGGRAPH '98* (New York, NY, USA, 1998), ACM Press, pp. 115–122.

[dHC01] DE HERAS CIECHOMSKI P.: *Parametric Dynamics - A Method for Addition of Dynamic Motion to Computer Animated Human Characters*. Master's thesis, Lund Institute of Technology, jul 2001.

[dHC06] DE HERAS CIECHOMSKI P.: Rendering massive real-time crowds. *PhD Thesis at Swiss Federal Institute of Technology, VRLab, Lausanne, Swizerland* (June 2006).

[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. *In proceedings of the 11th International Conference on Virtual Systems and Multimedia (VSMM'05)* (2005), 601–610.

[dHCUDC04] DE HERAS CIECHOMSKI P., ULICNY B., D. T., CETRE R.: A case study of a virtual audience in a reconstruction of an ancient roman odeon in aphrodisias. *In Proceedings of the 5th International Symposium on Virtual Reality, Archeology and Cultural Heritage (VAST'04)* (2004).

[Dob05] DOBBYN S.: Hybrid representations and perceptual metrics for scalable human simulation, July 2005.

[Gou71] GOURAUD H.: Continuous shading of curved surfaces. *IEEE Transactions on Computers* (1971), 623Ű628.

[GSM04] GOSSELIN D., SANDER P., MITCHELL J.: Rendering a crowd. In *ShaderX3 : Advanced Rendering with DirectX and OpenGL* (2004), Charles River Media, 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.

[Kt05] KAVAN L., *et al.*: Spherical blend skinning: a real-time deformation of articulated models. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2005), ACM Press, pp. 9–16.

[Lan98] LANDER J.: Skin them bones: Game programming for the web generation. *Game Developer Magazine* (May 1998), 11–16.

[MH02] MÖLLER T. A., HAINES E.: *Real-Time Rendering*. AK Peters, Ltd, 2002.

[PdHCM*06] PETTRE J., DE HERAS CIECHOMSKI P., MAÏM J., YERSIN B., LAUMOND J.-P., THALMANN D.: Real-time navigating crowds: Scalable simulation and rendering. *Computer Animation and Virtual Worlds (CAVW), special issue of CASA 2006* (2006).

[Str02] STRATTON C.: Optimizing for sse: a case study. *Hugi*, 25 (2002).

[UdHCC05] ULICNY B., DE HERAS CIECHOMSKI P., CLAVIEN M.: Crowdbrushed europe. *Winner of the Computer Graphics Forum Competition* (2005).

Populating Virtual Environments with Crowds: Variety Creation and Editing

P. de Heras Ciechomski and D. Thalmann,

Swiss Federal Institute of Technology, VRLab, Lausanne, Switzerland

Abstract

Variety is part of the believability aspect of a crowd, as the more differentiation there is between characters in form of morphology, clothing, color combinations, behavior and animations the more life-like it will appear. Using a few building blocks called templates artist resources are utilized optimally by instantiating humans from a common mold. This template variety is edited in real-time with a user interface exposing the coloring parameters as an HSB scale.

1. Introduction

The goal of this course notes is to show how to create a varied crowd, using few basic building blocks, such that different combinations are possible by employing little artistic resources. A human in the system is assumed to have a basic mesh, a single exchangeable texture with specific color areas and a set of animations to play, as described in the tutorial course notes on pipeline optimizations.

The variety in rendering extends the way Tecchia *et al.* [TLC02] create color variety from a single texture for billboards to dynamically animated 3D virtual humans. A wide spectrum of colors and appearances is achieved by combining textures and color variety (see Section 2). Each mesh has a set of interchangeable textures and the alpha-channel of each texture is segmented in several zones: one for each body part or piece of clothing. This segmentation is done using a desktop publishing software (in this case Adobe PhotoShop^{TM}). Two possible approaches for creating color variety are presented using the alpha layer information. The first approach is software-based and runs on graphics cards with a fixed function pipeline. The second approach uses a hardware based fragment shader, followed by a presentation of variety editing and color constraints.

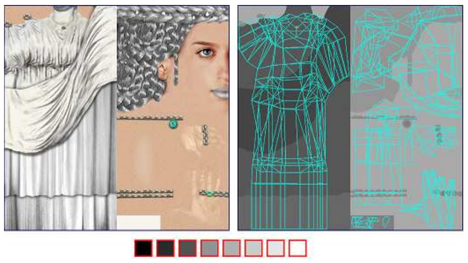


Figure 1: *Texture and alpha zone map of a patrician woman*

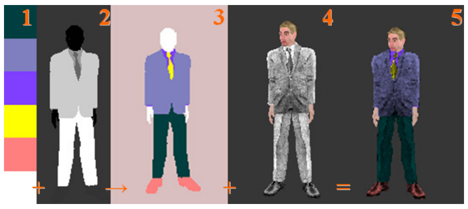


Figure 2: *Billboard variety coloring steps from left to right: acquiring the modification colors (1), selecting which area to modulate using the alpha channel (2), adding the diffuse color (4) and the result in (5), from Maupu [Mau05].*

1.1. A Software Approach: Fixed Function Pipeline

An alpha zone is denoting a part of the texture of the character that is to be modulated with a certain color. For example, an alpha zone is a specific part of a dress or some jewelry

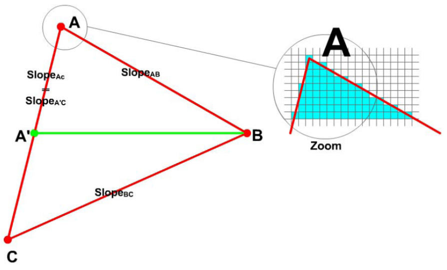


Figure 3: *Rasterizing the triangle over the alpha layer, from Maupu [Mau05].*

that the user wants to be able to color to differentiate crowd members from each other. Some triangles, denoted as *dirty* triangles, can span several such alpha zones that have to be colored differently. Examples of such triangles can be seen in Figure 1, where those close to the border between the face skin color and the hair are overlapping. A solution consists in re-triangulating *dirty* triangles, such that each zone is covered with more triangles. Even if this could be achieved using a subdivision algorithm, the solution wouldn't be appropriate since there would be no control on the amount of new triangles generated and a nonsensical situation could appear: a lower LOD mesh having more triangles than a higher LOD mesh.

The presented approach consists of using alpha tests on a split mesh to reduce the work of the graphics card. Meshes are quickly split at startup in several sub-meshes: one per group of uniform triangles, plus one for the *dirty* triangles. This splitting is achieved by plotting every triangle over the texture's alpha layer, as in Figure 3. During the plotting, if a change in the alpha value appears, the triangle is considered as *dirty*. Once the mesh is split, multi-pass rendering is done only on the *dirty* triangle sub-mesh, consisting usually of a small number of triangles. In the case of the patrician woman only 40 triangles out of 1000 are considered *dirty*.

The main drawback of the software approach is that the rendering complexity depends on the amount of *dirty* triangles. If some alphas overlap a large amount of triangles, the rendering will slow down accordingly, which makes the frame rate depend considerably on the chosen texture. Moreover, it also depends on the way alpha zones are designed. Thus, when intending to utilize this splitting mesh technique, this issue should be taken into account, in particular trying to reduce as much as possible the amount of *dirty* triangles when creating textures and mapping characters. A way to ensure this is to have zones with uniform alpha values connected by a large pixel neighborhood. However, this approach is able to run on a large install-base of machines.

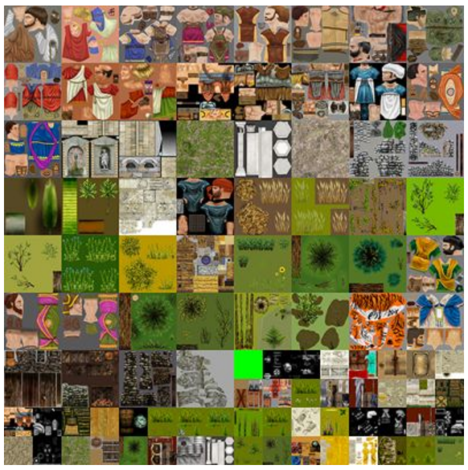


Figure 4: *A typical texture atlas.*

```
uniform sampler2D    baseTex; // texture unit 0
uniform sampler2D varietyTex; // texture unit 1
uniform float      varietyRow;

void main()
{
  N=normalize(N);
  vec4 texel=texture2D(baseTex,gl_TexCoord[0].st);
  texel*=texture2D(varietyTex,vec2(texel.a,varietyRow));
  texel.a=1.0;
  const float lightIntensity=max(0.0,dot(L,N))*0.8;
  const float ambient=0.4;
  gl_FragColor=(lightIntensity+ambient)*texel;
}
```

Figure 5: *Pixel shader doing lighting and color variety modification.*

1.2. A Hardware Approach: Shaders for Color Variety

Since the color modification information is encoded in a color texture as the alpha component is has 8 bits of precision, giving 256 possible coloring areas in theory. In a software approach the mesh is separated in triangles per coloring group and one *dirty* triangle group, which would in the worst case require 255 additive coloring passes plus 256 alpha check passes to render a character. This is of course completely unacceptable because more time would be spent on sending OpenGL commands and transferring data than on actual rendering. Since a designer has to manage these coloring regions it is more convenient to keep the number of regions down to 16 or less. Usually they are no more than eight, unless specific small details are designed which can be seen on the noble Roman virtual humans in form of jewelry shown in the ERATO project [ERA05].

The fragment shader determines for each pixel which color modification area it is part of and then colors it with the appropriate modulating color. In the first implementation of the fragment shader, an *if-else*-statement determined

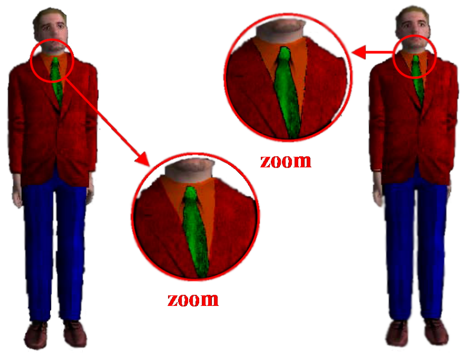


Figure 6: *Bilinear filtering artifacts in the alpha layer can be seen in the right zoomed-in version, near the borders of the orange shirt, the green tie and the red vest, from Maupu [Mau05].*

the correct color to apply with a limitation of 8 alpha zones. Since *Shader Model 3.0* allows nested *if*-statements, three *if*-evaluations per pixel are enough. Another way to program the fragment shader is to send a one dimensional texture and map the alpha value to the color to be modulated, for a specific character. In order to batch drawing calls (Wloka [Wlo03]), all individual one dimensional textures are put into one 2D texture of 1024 × 256 size, called a texture atlas whose generic example with 2D textures of individuals is shown in Figure 4. Each row in this texture atlas maps from an alpha value to a color value and one extra variable has to be sent to the mesh in the form of an identifier for which row to sample from, see listing in Figure 5.

1.3. Filtering

The color variety rests on a precise control of alpha key values. While uploading textures to the graphics card, such filtering known as *nearest* filtering is preferable to a *linear* one. Indeed a *linear* filtering would create new alpha values at the border of two alpha zones and thus, pixels at these borders would not be drawn, shown in Figure 6. However, nearest filtering is gross and to soften the texture Mipmaps are required (Williams *et al.* [Wil83]). OpenGL's Mipmap creation tool cannot be used here since it does a bilinear filtering on all layers of the texture. In fact, the alpha layer itself has to be nearest neighbor filtered separately, while the RGB layer must be bi-linearly filtered when the Mipmaps are being built.

1.4. Color

The color variety presented here is based on texture color modulation. Each fragment is colored by modulating the pixel color by the texture color: thus, the value produced by the texture function is given by:

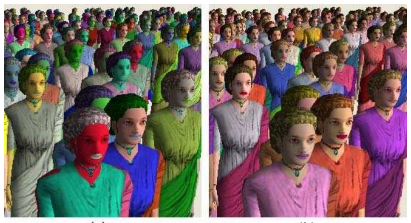


Figure 7: *Random color system (a) versus HSB control (b).*

$$C_v = C_t C_f \tag{1}$$

where f refers to the incoming fragment and t to the texture image. Colors C_v, C_t, and C_f can take values in the interval between 0 and 1. In order to have a large panel of reachable colors, C_t should be as light as possible, *i.e.*, near to 1. Indeed, if C_t is too dark, the modulation by C_f will give only dark colors. On the other hand, if C_t is a light color, the modulation by C_f 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. However passing the whole texture as luminance is not necessary as there is no gain in memory: OpenGL will emulate an RGB texture based on luminance values, since graphics cards are optimized for RGB textures. The drawback of passing the main parts of the texture to luminance is that *funky* colors can be generated *i.e.* agents are dressed in colors that do not match. Some constraints have to be added when modulating colors randomly. With the RGB color system, it is hard to constrain colors effectively. That is why the crowd rendering engine uses the HSB system [Smi78], also called HSV, meaning Hue, Saturation and Brightness or Value. This model is linked to the human color perception and is more user-friendly than the RGB system. In the next section a graphical user interface created for helping designers to set constraints on colors is presented.

2. Designing Variety

In the process of designing Romans in the ERATO project and more generally human color variety, localized constraints are dealt with : some body parts need very specific colors. For instance, roman skin colors are taken from a specific range of unsaturated shades with red and yellow 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. A tool is needed that allows us to control the randomness of color parameters for each body part of each roman (see Figure 7).

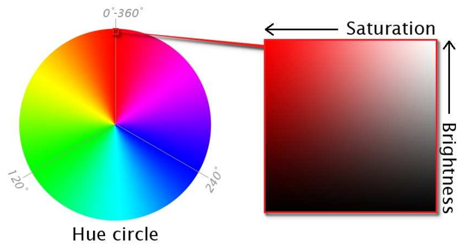


Figure 8: *HSB color space. Hue is represented by a circular region. A separate square region may be used to represent saturation and brightness, i.e., the vertical axis of the square indicates brightness, while the horizontal axis corresponds to saturation.*

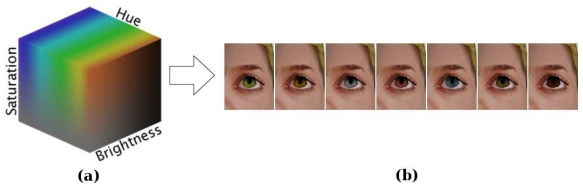


Figure 9: *The HSB space is constrained to a three dimensional color space with the following parameters (a): hue from 20 to 250, saturation from 30 to 80 and brightness from 40 to 100. Colors are then randomly chosen inside this space to add variety on the eyes texture of a character (b).*

2.1. Color Models

The standard RGB color model representing additive color primaries of red, green, and blue is mainly used for specifying color on computer screens. In order to quantify and control the color parameters applied to the Roman crowd, a user-friendly color is used. Smith [Smi78] proposes a model that deals with everyday life color concepts *i.e.* hue, saturation and brightness. This system is the HSB (or HSV) color model. 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. The brightness measures how light or dark a color is, as a value in the interval between 0 and 100. The color space is thus represented by the HSB model shown in Figure 8.

2.2. HSB Color Model as a Tool for Designing Variety

The HSB color model enables control of color variety in an intuitive and flexible manner. Indeed, as shown in Figure 9, by specifying a range for each of the 3 parameters, it is possible to define a three-dimensional color space, called the HSB map.

A GUI was built so that designers can easily load, modify and save HSB maps for different human templates (see Figure 11). This GUI provides all the necessary tools to efficiently :

- change the number of virtual humans rendered
- select the desired virtual human template
- choose which texture of the selected template one wishes to work with
- choose on which body part (alpha value) of the selected texture color ranges should be defined
- select a saturation and a brightness range between 0 and 100, and choose a range in the hue circle between 0 and 360 (cycles are allowed), for the currently selected alpha value

The result of using these features is visualized in real time *i.e.* every change directly affects each rendered human.

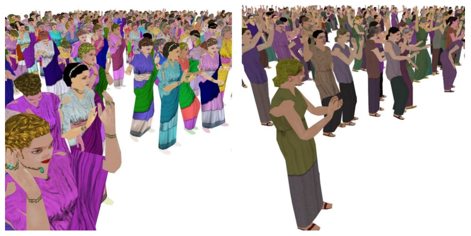


Figure 10: *Decurion women with saturation from 40 to 80 and brightness from 50 to 70 (left); plebeian women with saturation from 10 to 50 (right) .*

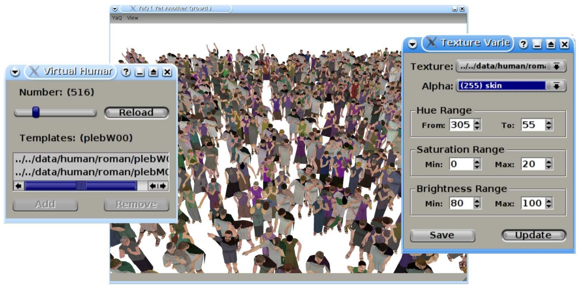
2.3. Variety Case Study : Roman Society

To further illustrate the use of the GUI, a case study in the framework of the ERATO project is presented [ERA05]. In order to simulate Roman society, social classes had to be differentiated. These differences were shown through clothing, where colors and color patterns defined the rank of individuals. For instance, decurion women (rich elite) wore fine fabric with rich colors, while lower class citizens wore simple garments made of usually dark raw material. HSB maps allowed to specify these significant differences by setting saturation and brightness values for rich garments and lower for modest ones (see Figure 10).

3. Results

3.1. Variety Pixel Shaders versus Software

For software rendering without shader capability it is important to use few passes for alpha testing as each alpha channel requires an extra rendering pass. In this case a segmented mesh and dirty triangle separation gives a 250% performance improvement. However if shader programs can be executed it wins with no contest with a 700% speed improvement even over segmented meshes (see Results section of the pipeline optimizations course notes or [dHC06]).



Number: (516)
Templates: (plebW00)
../../data/human/roman/plebW
../../data/human/roman/plebM
Texture: ../../data/human/rom
Alpha: (255) skin
Hue Range From: 305 To: 55
Saturation Range Min: 0 Max: 20
Brightness Range Min: 80 Max: 100

Figure 11: *Real-time texture variety design of plebeian Roman social classes. Dialog to select the template to edit and to choose the number of characters (left); the results displayed on the crowd (middle); dialog to design the variety of the selected template with the possibility to choose the body part along with the texture of the template to edit (right).*

```
#alpha name %     Hmin  Hmax  Smin  Smax  Bmin  Bmax
0      shoes 90   20    45    64    95    30    60
15     hair  100  6     44    39    51    25    75
25     skin  100  25    41    25    35    85    95
35     pants 90   0     360   30    50    20    40
50     suit  90   0     360   30    50    20    30
65     shirt 70   0     360   0     80    50    100
80     tie   50   0     360   40    100   0     60
100    mouth 0    0     0     0     0     0     0
```

Figure 12: *Color variety editing per texture in an ASCII file. Each row describes three intervals in the HSV space.*

3.2. Variety Editing

Using the HSV color variety interface for the humans, per variety texture color variations are stored in text files, shown in Figure 12. This file is editable directly inside of a text editor or can be exported from the interface shown in Figure 11.

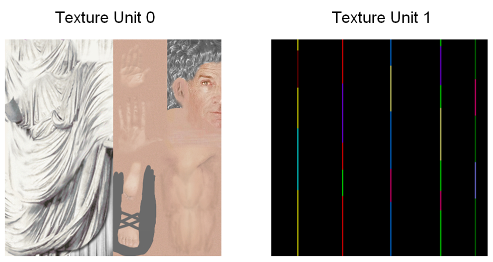


Figure 13: *Texture units and how they change active textures. On the left is the color and alpha (RGBA) base texture and on the right in the second texture unit is the color variety texture.*

Visually the clothes and accessories were subjectively easier to constrain for the lab designers as well as gave a credible way of randomizing colors within selected intervals. At startup of the application for each variety texture a color variety texture is created which is then shared by all instances of a template. One row in this texture corresponds to a sample of each of the HSV intervals, converted to RGB colors, shown on the right in Figure 13.

4. Discussion

A method for segmenting a mesh into regions to speed up software rendering of the alpha test path was shown and was efficient. Hardware shaders were shown to be more powerful, easier to program and more performant than a software solution. An easy and intuitive HSB coloring interface was used and did improve the design process of human garment colorings using constrained randomized intervals. Hopefully more visual GUIs for crowd editing will adopt this type of coloring parametrization as it is relatively easy to implement.

References

[dHC06] DE HERAS CIECHOMSKI P.: Rendering massive real-time crowds. *PhD Thesis at Swiss Federal Institute of Technology, VRLab, Lausanne, Swizerland* (June 2006).

[ERA05] ERATO - identification, evaluation and revival of the acoustical heritage of ancient theatres and odea, 2005. project website, http://www.at.oersted.dtu.dk// erato.

[Mau05] MAUPU D.: Creating variety - a crowd creator tool. *Semester project at Swiss Federal Institute of Technology - EPFL, VRLab* (2005).

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

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

[Wil83] WILLIAMS L.: Pyramidal parametrics. In *SIGGRAPH '83: Proceedings of the 10th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1983), ACM Press, pp. 1–11.

[Wlo03] WLOKA M.: Batch, batch, batch: What does it really mean. *GDC* (2003).

Populating Virtual Environments with Crowds: Interaction

P. de Heras Ciechomski and D. Thalmann,

Swiss Federal Institute of Technology, VRLab, Lausanne, Switzerland

Abstract

Interacting with a crowd is an easy task if the interface is spray-like as in a PhotoShop application as it is something intuitive and easy to relate to. Extending the spray paradigm beyond assignment of emotional states and simple animation states is what has been puzzling us as researchers the last months. With the advent of new powerful path planning algorithms serving thousands of path queries simultaneously the spray interface finally found virgin ground. We show how to not only assign indivudual atomic emotional states to virtual humans through a spray interface but also how to interact with path quieries for a thousand agents at a time.

1. Introduction

When increasing the number of individuals constituting a crowd it becomes more difficult to create unique and varied scenario content. Modifying features of every single individual one by one, soon becomes laborious. If, on the other hand, a set of features (either uniform, or patterned) are applied to many individuals at once, unwanted artifacts on a larger scale could emerge, resulting in an "army-like" appearance with too uniform, or periodic distributions of individuals or characteristics. Random distributions can alleviate such problems; however, it can be very difficult to capture the desired constraints into a set of mathematical equations, especially considering integration into common art production pipelines.

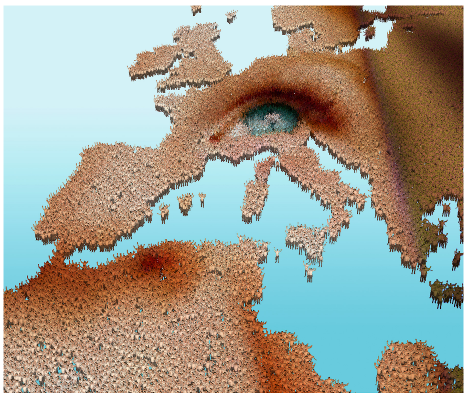


Figure 1: *CrowdBrushed Europe.*

The challenge is how to create complex scenes resembling a variety-rich look of the real world. Here it is understood that complexity is analogous to a notion of complexity of patterns generated by cellular automata as in Wolfram [Wol02]: not uniform, not periodical, nor just fully random.

2. Related Work

Bottom-up approaches, such as local rule based flocking as in Reynolds [Rey87] can create such complexity. However they are difficult to control if particular end configurations are to be achieved, namely how to set local rules to get a global result. In the work of Anderson *et al.* [AMC03] interesting results for a particular case of flocking animation are obtained. Nevertheless, the algorithm can get very costly when increasing the number of entities and simulation time.

Major 3D content creation packages used by the media industry now offer tools to improve working with a large number of virtual characters, such as Character Studio^{TM} or Softimage^{TM}. The production of massively populated scenes is still in the majority of cases a lengthy and manual-intervention intensive process, operating mostly in a non-real-time mode. An interesting approach to add sets of objects (as clouds, trees, flowers, or buildings) to the scene is used in Maya Paint Effects^{TM}, where a designer can paint pseudo-3D objects in the scene using 2D brush strokes. SuchE objects are not fully integrated into the 3D scene: they are rendered in a special buffer with separate shading and are further composed into the final image as a Z-buffer based post process. Other work done on direct multi-agent interaction is presented in Millan *et al.* [MR05].

The approach presented here gives full creative power to designers using metaphors of artistic tools, operating on a two-dimensional canvas, familiar from image manipulation programs working in what-you-see-is-what-you-get-mode (WYSIWYG), with a real-time view of the authored scene, such as Photoshop^{TM}. The advantages of immediate feedback, an intuitive interface and familiarity allows to better express the artist's vision at the same time leading to an increase in productivity. It is also presented in the thesis of Branislav Ulicny [Uli05] and de Heras Ciechomski [dHC06].

The structure is as follows: first, the overall design of the system discussing the requirements needed for interactive authoring is presented, followed by a detailed account of the concept of brushes. Furthermore present results are shown, where a prototype of the crowd brush application is used, to create a scene of a virtual audience in a reconstruction of an ancient theatre.

Then the spray interface is extended to interact with path planning the route of thousands of agents at the same time as in the work of Pettre *et al.* [PdHCM*06]. In this work virtual human paths are assigned to several thousands of agents at the samte time by re-using the same path solution thereby reducing the cost a thousandfold. The spray interface is used to select key location where the agents have to pass through.

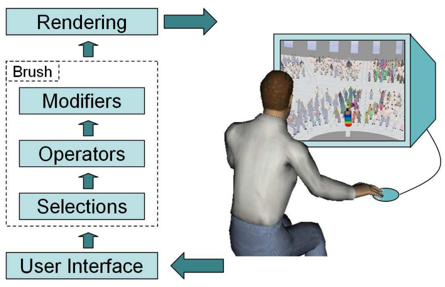


Figure 2: *Overview of the system design.*

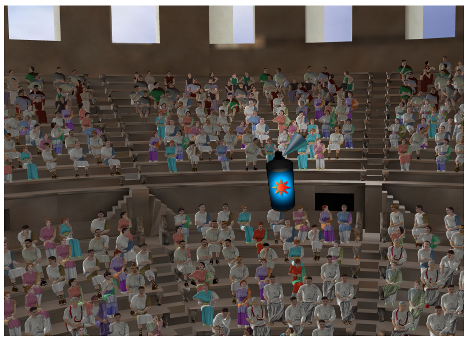


Figure 3: *CrowdBrush used on the Roman crowd of the Aprhrodisias Odeon.*

3. System overview

The goal is to create a system that allows authoring of freely navigable real-time 3D scenes, composed of a large number of varied animated individuals in a virtual environment. The authoring should be simple, intuitive and usable by non-programmers.

Inspiration comes from the domain of image and word processing, where most applications use WYSIWYG approaches, with immediate feedback of the resulting manipulations. In computer graphics such an approach was used, for example, for direct painting on models, Hanrahan *et al.* [HH90] and Kalnins *et al.* [KMM*02], sketching of 3D models out of 2D drawings, Zelznik *et al.* [ZHH96], or painting 2D images with a virtual brush, Xu *et al.* [XLTP03].

The idea is simple: the designer manipulates virtual tools, working in a two-dimensional screen space, with a mouse and a keyboard. These tools then affect the corresponding objects in a three-dimensional world space, as shown in Figure 3. Different tools have different visualizations and perform different effects on the scene including creation and deletion of crowd members, modifying their appearances, triggering of various animations or setting high-level behavioral parameters.

Briefly, experiments were made with a fully three-dimensional interface, where tools existed in a 3D world space. Nevertheless it appeared to be not very practical, at least not when using standard input devices operating in two dimensions as a mouse or a trackball. The usability of a 3D interface could be improved using some truly 3D input devices such as a spaceball, a 3D mouse, or magnetic sensors. However, it would limit the number of potential users as such devices are not common.

In order to create an authoring tool as outlined above, the system, on which it will operate, should fulfill the following requirements:

Individuality: The system must allow for each individual instance to have a set of attributes, and not share them among many individuals, as they can potentially have unique values, and can be individually selectable.

Responsiveness: It must be fast enough for real-time editing to allow for an immediate feedback loop. Therefore, it must not involve lengthy preprocessing stages (at least for features targeted for authoring) and also the system's responses to changes must be fast.

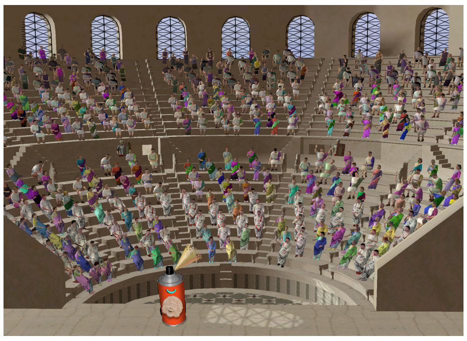


Figure 4: *Laughter CrowdBrush used on the Roman crowd of the Aphrodisias Odeon.*

The requirements are close to those of any interactive application. Indeed, boundaries between authoring and interaction are getting more and more fuzzy. A recent trend in game development is to use the same application for part of the authoring and actual game play (or to integrate play-test ability in authoring tools), such as in RenderWare Studio^{TM} and Gamebryo^{TM}. The player interaction is, then, the subset of possible interactions for a designer. Such designs came from the necessity of constantly switching between "play" and "create" modes while developing the game, in order to increase productivity.

Figure 2 shows an overview of the system design. The user controls the application using a mouse and a keyboard. The mouse moves the visual representation of the brush tool (an icon of a spray can is used) on the screen, with the mouse buttons triggering different actions. The keyboard selects different tools and switches between "navigate" and "paint" modes. In the "navigate" mode, the mouse controls position and orientation of the camera. In the "paint" mode, the camera control is suspended and different areas on screen are selected depending on triggered actions. These areas are then further processed by the brush according to their particular configurations, explained in the next section.

4. Brushes

Brushes are tools with a visual representation that affect crowd members in different manners. For example, a brush can create new individuals in the scene, or it can change their appearances or behaviors. Selected visualizations of brushes intuitively hint on their specific function. For example, the creation brush has an icon of a human, the orientation brush has an icon of a compass, the deletion brush has an icon of a crossed over human, and so on, as in Figure 4.

A brush is processed in three stages. First, a selection of the affected area in 2D screen space is performed according to a triggered action, with subsequent picking of entities in the 3D world space. Then, the operator modifies the manner of execution of the brush in the selected area. Finally, the brush changes the values of the modifiers for the affected individuals, or in case of the creation brush, new population members are created.

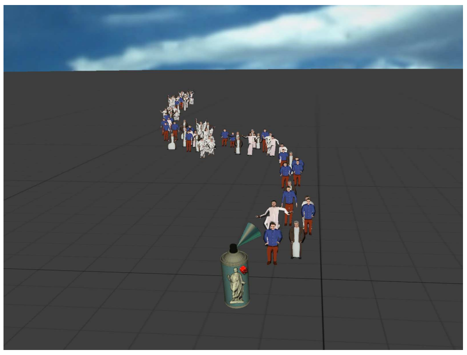


Figure 5: *Creation brush with random operator.*

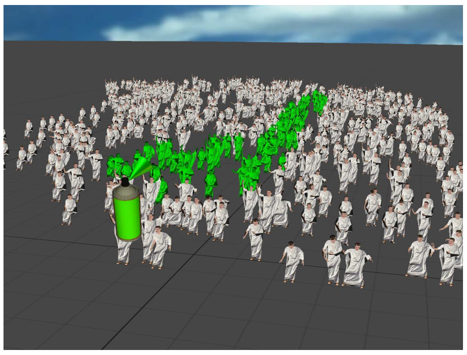


Figure 6: *Color brush with uniform operator.*

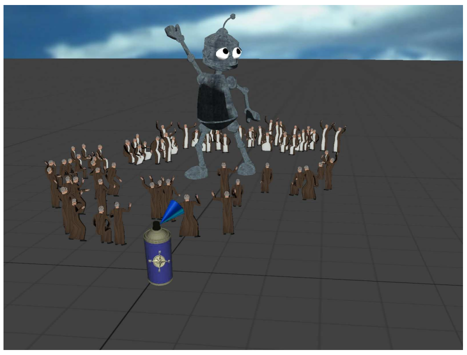


Figure 7: *Orientation brush with gradient operator (Hugo model by Laurence Boissieux ©INRIA 2003).*

Selections are defined in screen-space. A selection can be a single point at the location of a cursor, or an area around a cursor. If the selection is a single point, picking in the 3D world is performed by computing the intersection of a line segment with the scene. If the selection is an area, picking is performed on a random sample of points from that area, following a "spray" metaphor. The size of the selected area in world space, changes with the level of zoom into the 3D scene. This provides an intuitive control of focus: if one wants to work on a large part of the crowd, zooms out of the 3D view are performed; if focus is on a smaller group or individual, a zoom-in is performed.

Operators define how selections are affected. For example, a stroke of the creation brush with the random operator creates a random mix of entities (see Figure 5); a stroke of the uniform color brush sets colors of affected individuals to the same value, as shown in Figure 6; and a stroke of the orientation brush with the gradient operator lets the individuals turn in the direction of the gradient as in Figure 7.

Modifiers are non-sharable properties, giving uniqueness to every individual member of the crowd. Modifiers encapsulate low-level features influencing both appearance and animations of virtual humans. Spatial configuration is qualified by modificators of position and orientation. Appearance is influenced by modifiers of color, texture, material and scale. Execution of actions is determined by animation selection, shift, and speed modifiers. High-level features can use a combination of several low-level features accessed through their modifiers. For example, a particular emotional state sets animations from a predefined set with some specific speed, or clothing style selects a set of appropriate textures and colors for different body parts.

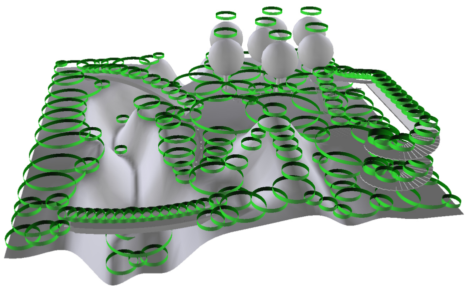


Figure 8: *Bounding cylinder graph.*

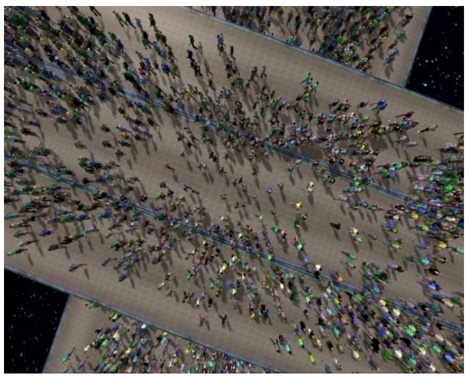


Figure 9: *PLanet Eight. Crowd members can walk on multiple levels above and below each other.*

5. Path Construction

Scalable path planning was introduced in our system with the work of Pettre *et al* [PdHCM*06] where cylindrical volumes (Figure 8) define walkable paths in the environment. A fully automatic scene partitioning is presented and allows multi-level paths to exist at the same time, such as paths on top and below a stairway for example, see Figure 9. When a scene has been partitioned which takes a few minutes of computing, paths between cylinders have been defined such that if two cylinders intersect, the plane created by their intersection is used as a door or bridge for humans to go from one to the other as in Figure 11.

5.1. Path Sharing

When a cylindrical map of the environment with all walkable intersections has been computed a sharable path is computed as in Figure 10. Computation is done using a straightforward Dijkstra algorithm, however since the cylinders bounding the path are big and define intersectional planes several humans can use the same path without it looking similar. In Figure 11 one can see that humans can choose any path between two gates from one in-cylinder to an out-cylinder. Using this path contruction and sharing, humans adapt their distribution according to the available path width, since cylinders bound as much available space as possible. Each chosen path gives rise to a certain number of possible routes, so using only two clicks of the mouse a few thousand different routes are made available.

6. Path Editing

Path editing is intuitive and easy to perform, using the mouse pointing at the part of the city where the humans should go. When the intersection of the mouse picking point and the affected bouding cylinder is established the end-point has to be chosen by the user. An example city with cylinders is

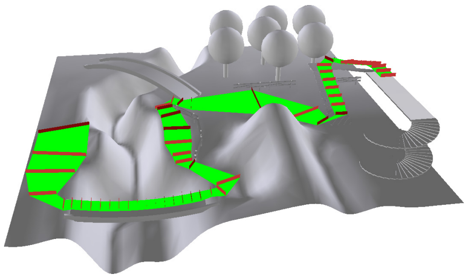


Figure 10: *A shareable path is computed.*

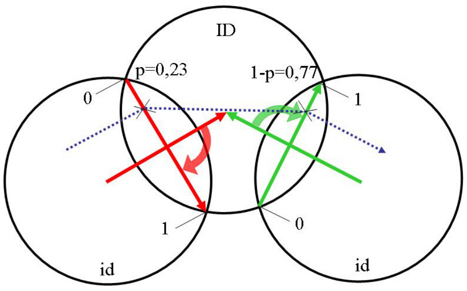


Figure 11: *The intersection between two cylinders is called a gate.*

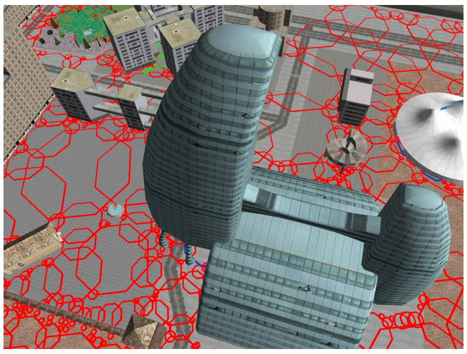


Figure 12: *A city with bounding cylinders drawn.*

depicted in Figure 12. Using the start and end-point a shareable path is computed as illustrated in the previous section. No collision avoidance is done at the moment in-between humans for large distances from the camera, only when very close to the viewer, the active nodes do collision-avoidance for their crowd members.

7. Results and Discussion

7.1. CrowdBrush

Usability and responsiveness of authoring applications is tested by producing the scenario of a virtual audience as shown in Figure 4. Given existing models of a theatre and template humans, the task is to fill the building with an audience and distribute the animations according to the requirements of the designer. Four different template humans, each represented with around one thousand triangles are used. The theater model has around fourteen thousand triangles.

In order to facilitate positioning of the humans, a grid of valid positions in the theatre following the distribution of the seats is created. The creation brush is restricted to operate only on this grid, instead of free picking. Using the grid the humans are positioned without caring about collisions, for example, if two humans happen to materialize very close to each other. The correct position and orientation of the audience is automatic, in this way, the same scene has a certain expected behavior when a user interacts with it, in a similar way as in a paint program such as PhotoShop^{TM}, where pixel positions are placed in a grid.

Interactive frame rates are kept at all times, like in cases, when adding and removing humans from the scene, or when modifying attributes, such as animation, color or orientation. When increasing the number of humans in the audience, performance was dropping, yet even with full capacity of the theatre, which is around 700 places, an acceptable rate of 27 frames per second with lighting enabled is maintained. Due to the responsiveness of the crowd engine and by consequence the interface, scene manipulation is immediate and intuitive, as changes appear on-the-fly.

Besides scene creation, interacting with the audience is enabled by the use of brushes. As soon as humans start to appear in the scene, their behaviors are changed by using the "emotion" spray that makes humans play different kinds of animations from predefined sets, like happy animations or sad animations.

The proposed authoring approach can work in conjunction with different crowd rendering systems if they fulfill the requirements defined in Section 3. The brush metaphor is also compatible with work done on crowd behavior simulation, where the spray could be used, for example, to send events to activate behavioral rules [MT01], [UT02] as is done in the ERATO project [ERA05].

7.2. Path Sharing

Using the bounding cylinder approach and editing the paths from a high-level interface such as CrowdBrush the city is quickly and interactively populated. Using the density distributions described in [PdHCM*06] paths are uniformly filled with humans directly and over time. The only drwaback of the method is that humans can sometimes be congested in cylindrical paths athat are too narrow. The adaptive density functions try to alleviate this problem but they still remain in some places. A solution is to simply remove cylinders from possible paths that are narrower than a certain width.

Sharing of the same path for 10 000 humans is possible and looks good, see Figure 13 and 14.

References

[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.

[dHC06] DE HERAS CIECHOMSKI P.: Rendering massive real-time crowds. *PhD Thesis at Swiss Federal Institute of Technology, VRLab, Lausanne, Swizerland* (June 2006).

[ERA05] ERATO - identification, evaluation and revival of

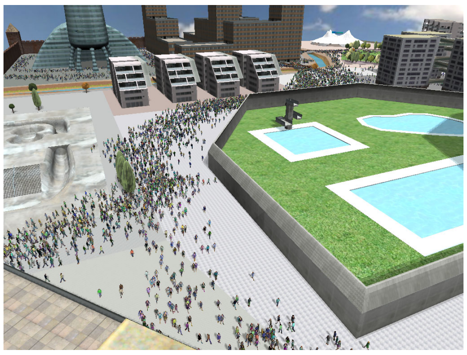


Figure 13: *A city with 35 000 humans following only 6 paths.*

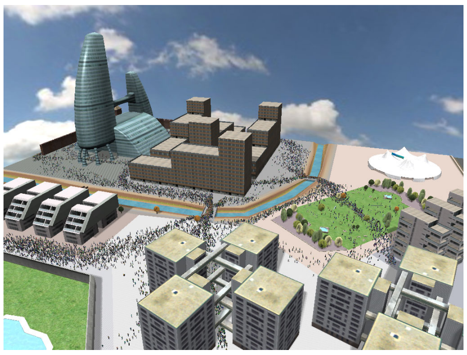


Figure 14: *A city with 35 000 humans following only 6 paths.*

the acoustical heritage of ancient theatres and odea, 2005. project website, http://www.at.oersted.dtu.dk// erato.

[HH90] HANRAHAN P., HAEBERLI P. E.: Direct WYSIWYG painting and texturing on 3D shapes. *In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques (SIGGRAPH'90)* (1990), 215–223.

[KMM*02] KALNINS R. D., MARKOSIAN L., MEIER B. J., KOWALSKI M. A., LEE J. C., DAVIDSON P. L., WEBB M., HUGHES J. F., FINKELSTEIN A.: WYSIWYG NPR: Drawing strokes directly on 3D models. In *Proc.SIGGRAPH'02* (2002), pp. 755–762.

[MR05] MILLAN E., RUDOMIN I.: Agent paint: Intuitive specification and control of multiagent animations. *In Proceedings of Computer Animation and Social Agents (CASA'05)* (2005).

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

[PdHCM*06] PETTRE J., DE HERAS CIECHOMSKI P., MAÏM J., YERSIN B., LAUMOND J.-P., THALMANN D.: Real-time navigating crowds: Scalable simulation and rendering. *Computer Animation and Virtual Worlds (CAVW), special issue of CASA 2006* (2006).

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

[Uli05] ULICNY B.: Crowds for interactive virtual environments. *Ph.D. Dissertation at Swiss Federal Institute of Technology, Lausanne (EPFL), Switzerland* (2005).

[UT02] ULICNY B., THALMANN D.: Towards interactive real-time crowd behavior simulation, Dec. 2002.

[Wol02] WOLFRAM S.: *A New Kind of Science*. Wolfram Media, Inc., 2002.

[XLTP03] XU S., LAU F. C. M., TANG F., PAN Y.: Advanced design for a realistic virtual brush. *Computer Graphics Forum 22*, 3 (2003), 533–542. (Proc. Eurographics'03).

[ZHH96] ZELEZNIK R. C., HERNDON K. P., HUGHES J. F.: SKETCH: An interface for sketching 3D scenes. In *Proc. SIGGRAPH '96* (1996), ACM SIGGRAPH, Addison Wesley, pp. 163–170.

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

S. Dobbyn and C. O'Sullivan

Interaction, Simulation and Graphics (ISG) 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, animation, and behaviour of virtual crowds in the following manner:

- We first introduce general **character visualisation** techniques using the fixed function graphics pipeline, and show how recent improvements in graphics hardware has greatly improving the realism of characters in computer games. Furthermore, we describe **acceleration techniques** for the rendering of large crowds which can be subdivided into three categories: **visibility culling** methods, **geometrical *level of detail* (LOD)** and sample-based rendering techniques such as using **image-based** and **point-based** representations.
- Next, we describe **character animation techniques**, including how a character's model is animated using the **layered approach**, and the various techniques for generating character animations such as **kinematics**, **physically-based animation** and **procedural animation**. We also describe how **animation and simulation level of detail** provides a computationally efficient solution for the simulation of crowds.
- Finally, we detail **behavioural techniques** used to endow virtual characters with artificial intelligence (AI), including **agent-object interaction techniques** to simulate characters interacting with objects. We also provide an overview of the research on **intelligent virtual agents** and present how the technique of **level of detail AI** has been employed in computer games.

2. 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.

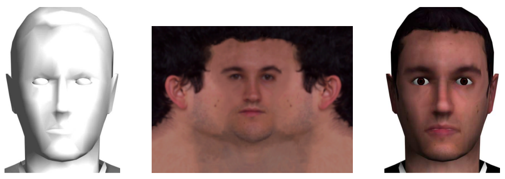


Figure 1: *Simple Texturing-Mapping: (a) Mesh without texture-mapping, (b) Texture Map (c) Texture-mapped mesh.*

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.

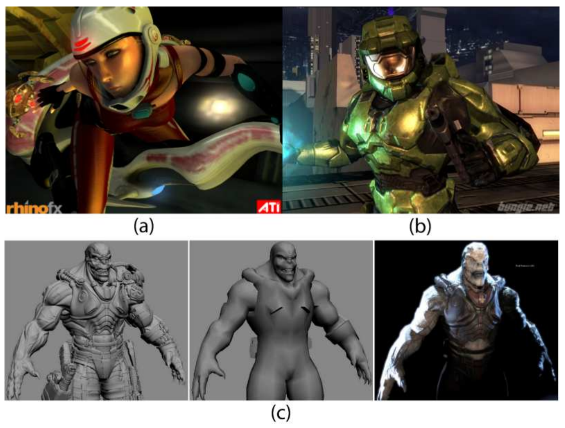


Figure 2: *Per-pixel lighting effects such as environment mapping in (a) Ruby Demo ((© ATI Technologies) and (b) Halo 2 (© 2004 Microsoft Corporation), and (c) Normal mapping in Unreal Engine 2003 (© 2005 Epic Games Inc).*

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 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.

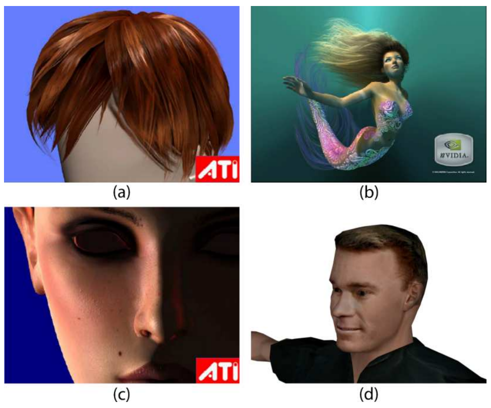


Figure 3: *Real-time hair rendering based on the light scattering of human hair fibres [MJC*03] and a fur rendering model [KK89] using a (a) polygonal model [Sch04] (b) a particle system [Wlo04]. (c) Real-time skin rendering based on subsurface scattering [SGM04]. (d) Hair and skin rendered with simple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

More recently, more realistic character effects borrowed from the film industry have been implemented in real-time. Based on the technique used to light the face of digital characters in the film *The Matrix Reloaded*, Sander et al. [SGM04] produced realistic looking skin in real-time. Scheuermann et al. [Sch04] improved the rendering of real-time hair using a polygonal model, where the hair shading is based on the work on light scattering of human hair fibers by Marschner et al. [MJC*03] and on Kayiya et al.'s fur rendering model [KK89]. While this technique has greatly improved the realism of real-time hair, in addition to using low geometric complexity, it assumes little or no hair animation and is not suitable for all hair styles. Wloka [Wlo04] uses a similiar rendering approach for underwater hair which is animated by treating it as a particle system. Unfortunately, these techniques can only be used for a limited number of characters, since they are computationally intensive, and therefore simple texture-mapped triangles are typically used for an individual's skin and hair detail within large crowds (Figure 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].

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

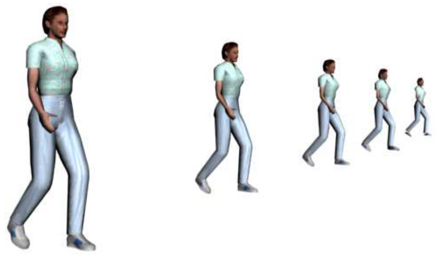


Figure 4: *Five discreet mesh models containing (a) 2,170 (b) 1,258 (c) 937 (d) 612 and (e) 298 triangles.*

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.

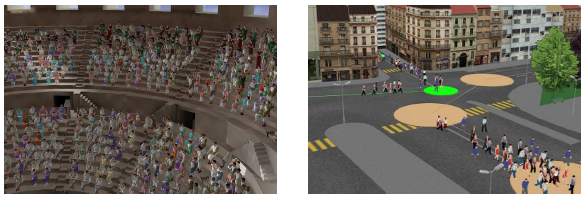


Figure 5: *Rendering crowds using a discrete LOD approach [dHCUCT04].*

Gosselin et al. [GSM05] present an efficient technique for rendering large crowds while taking variety into account. Their approach involves reducing the number of API calls need to draw a character's geometry by rendering multiple characters per draw call, each with their own unique animation. This is achieved by packing a number of instances of character vertex data into a single vertex buffer and implementing the skinning of these instances in a vertex shader. As vertex shading is generally the bottleneck of such scenes containing a large number of deformable meshes, they minimize the number of vertex shader operations that need to be performed.

In their simulation, they use one directional light to simulate the sun, and three local diffuse lights. The shading of each character's mesh is performed by per-pixel shading and a normal map generated from a high resolution model is used. Specular lighting is calculated for the sun and is attenuated using a gloss map to allow for parts of the character to have differing shininess. Realism is further increased by using an ambient occlusion map generated from the high resolution model. This map approximates the amount of light that could reach the model from the external lighting environment and provides a realistic soft look to the character's illumination. Finally, using a ground occlusion texture which represents the amount of light a character should receive from the sun based on their position in the world, the illusion that the terrain is shading the characters as they move within the environment is created. So that the characters are not a carbon copy of each other, they use a colour lookup texture, which specifies 16 pairs of colours that can be used to modulate the character, with a mask texture to specify which portions should be modulated. In addition to this, decal textures to add other various details to the character's model, such as badges, are applied (see Figure 6).

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

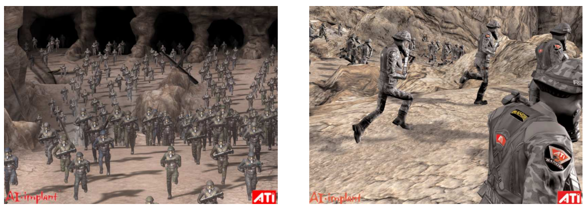


Figure 6: *Geometric-based representations rendered with various per-pixel shading effects [GSM05] (© 2005 ATI Technologies).*

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).

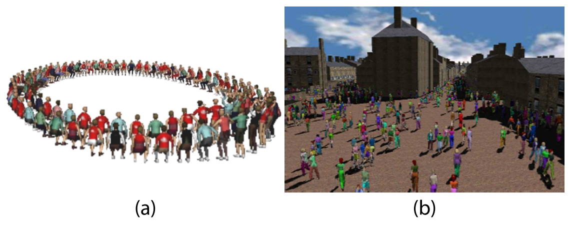


Figure 7: *Image-based crowds: (a) Dynamically generated image-based crowds [ABT00] (b) Pre-generated image-based crowds [TC00].*

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.

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 reference viewpoint is selected and displayed on a quadrilateral, which is dynamically orientated towards the viewer. To allow for the dynamic lighting of the impostor representation, Techia et al. [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).

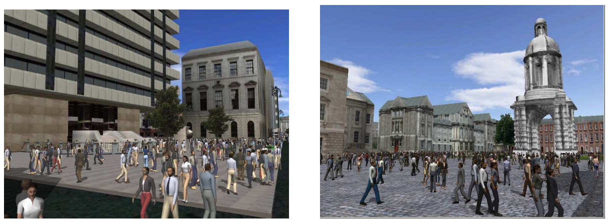


Figure 8: *Geopostor system.*

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.

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.

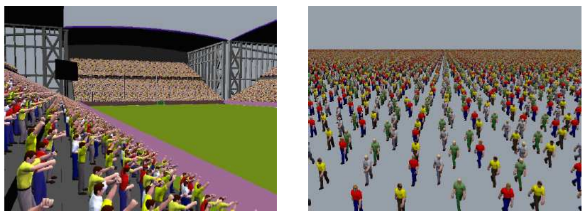


Figure 9: *Point-based crowds: (a) Stadium populated with animated 16,000 fans and (b) Crowd of 90,000 humans walking on the spot.*

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 ($(TM_i{}^{-1})^T$).

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

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.

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.

4. Behavioural Techniques for Characters

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

4.1. Agent-Object Interaction

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

Smart objects extend the idea of object specific reasoning, whereby objects contain more information than just intrinsic object properties [Lev96]. A smart object is an object that is modelled with its interaction features, which are all parts, movements and descriptions of an object that have some important role when interacting with an agent. Smart objects provide the necessary parameters for motion generation. Features are identified in such a way as to provide important information to the motion generator. Smart object applications provide a number of advantages over more commonplace approaches: they decentralise animation control, separate high level planning from low level object reasoning and allow the same object to be used in multiple applications. They also allow behaviours to be easily connected with high-level planners, and provide for Object Oriented Design since each object encapsulates data. Extensive research has been conducted by Kallman and Thalmann [KT98, KT99a, KT99b] on agent-object interactions using smart objects.

4.2. Intelligent Virtual Agents

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

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

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

The ViCrowd system [MT01] was created to generate and model crowds with various degrees of autonomy. The crowd is modelled as a hierarchy of groups and individuals. Depending on the complexity of the simulation, a range of behaviours, from simple to complex rule-driven, are used to control the crowd motion with different degrees of autonomy (see Figure 10. The crowd behaviour is controlled in three different ways:

1. Using innate and scripted behaviours.
2. Defining behavioural rules, using events and reactions.
3. Providing an external control to guide crowd behaviours in real time.

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

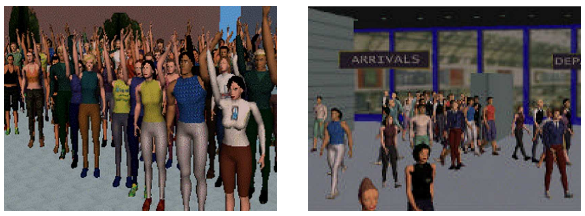


Figure 10: *ViCrowd system.*

4.3. Level of Detail Artificial Intelligence

Modelling a scene populated with large crowds is a challenging process. However, the simulation does not need to process every agent in order to present a believable experience. For example, agents that are not in the view-frustum should not worry about detecting future collisions with other agents. Hence, a virtual world needs to present a believable world, but this does not mean that it has to be an accurate model of a real world. Level of detail AI (LODAI) reduces high CPU demands by approximating the behaviour of agents who are rated with a low-level of importance (e.g., if the agent is not in the view-frustum or is at a great distance from the viewpoint) with minimal perceptual degradation.

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

4.3.1. Role-Passing System

In character-centric video games i.e., games which focus on character interaction, rather than action, there is a trend for computer controlled NPCs to be very simplistic in their behaviour. Usually, no modelling of NPCs is performed until the player reaches the location in which an NPC is based. When the player arrives at this location, NPCs typically wait to be involved in some interaction, or play through a predefined script. This leads to very predictable, and often jarring behaviour. For example, a player might enter a room and meet an NPC who would perform a set of actions based on some script. However, if the player were to leave that room and re-enter, the NPC would play through the same script again.

MacNamee et al. [MC01] developed the Proactive Persistent Agent (PPA) architecture and its technique of role-passing which allows intelligent agents to take on different roles depending on the situation in which they are found. Agents based on this architecture are proactive in the sense that they can take the initiative and follow their own goals, irrespective of the actions of the player. Persistence refers to the fact that, at all times, all NPCs in a virtual world are modelled at least to some extent, regardless of their location relative to that of the player. This complements the use of level of detail AI (LODAI), whereby the characters can be controlled at higher or lower levels of sophistication based on their position, with respect to the player, in a virtual world.

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

The main advantage of role passing is that it offers a simple and efficient technique for the control of agents which move competently, and believably, between situations (Figure 11). Not only does this allow believable agent behaviour, it promotes the idea of situational intelligence, whereby issues unrelated to an agent's current situation are not taken into account, thus reducing its processing load. Role-passing also makes populating a virtual world with agents a straightforward process. Placing agents within novel situations involves simply defining new roles, easing some of the complications involved in attempting to design very general agents. Furthermore designing a role involves simply drawing a fuzzy cognitive map (FCM), and it is possible that this is more amenable to game designers than writing script or plans.

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.

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

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

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

[BBB*99] BADLER N. I., BINDIGANAVALE R., BOURNE J., ALLBECK J., SHI J., PALMER M. S.: Real-time virtual humans. *International Conference on Digital Media Futures* (1999).

[BID*01] BURKE R., ISLA D., DOWNIE M., IVANOV Y., BLUMBERG B.: Creature smarts: The art and architecture of a virtual brain. *Proceedings of the Game Developers Conference* (2001), 147–166.

[BMdOB03] BRAUN A., MUSSE S. R., DE OLIVEIRA L. P. L., BODMANN B. E. J.: Modeling individual behaviours in crowd simulation. *16th International Conference on Computer Animation and Social Agents (CASA)* (2003), 143–148.

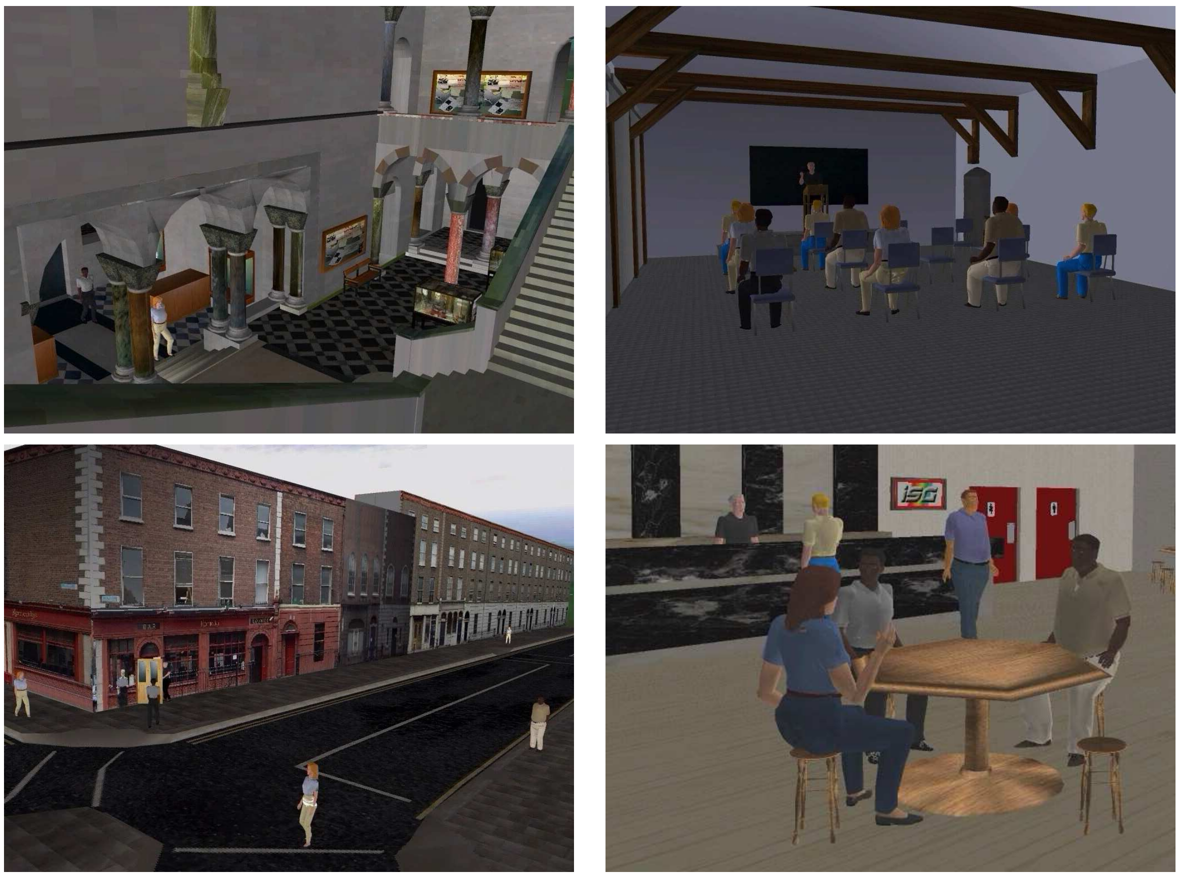


Figure 11: *University campus simulation using the PPA architecture: (a) Students going to lectures. (b) Academic presenting the lecture. (c) Academic and students going to the pub. (d) People socialising in the pub.*

[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.

[CT00] CAICEDO A., THALMANN D.: Virtual humanoids: Let them be autonomous without losing control. *The Fourth International Conference on Computer Graphics and Artificial Intelligence 3IAŠ2000* (2000).

[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.

[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.

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

[FvdPT01] FALOUTSOS P., VAN DE PANNE M., TERZOPOULOS D.: The virtual stuntman: Dynamic characters with a repertoire of autonomous motor skills. *Computers and Graphics 25*, 6 (December 2001), 933–Ű953.

[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.

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

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

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

[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.

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

[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.

[MC01] MACNAMEE B., CUNNINGHAM P.: Proposal for an agent architecture for proactive persistent non player characters. *Proceedings of the 12th Irish Conference on AI and Cognitive Science* (2001), 221–Ű232.

[MDCO03] MACNAMEE B., DOBBYN S., CUNNINGHAM P., O'SULLIVAN C.: Simulating virtual humans across diverse situations. *Intelligent Virtual Agent Conference* (2003), 159–163.

[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.

[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.

[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 (2001), 152–164.

[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).

[PBC*01] PANDZIC I., BABSKI C., CAPIN T., LEE W., MAGNENAT-THALMANN N., MUSSE S. R., MOCCOZET L., SEO H., THALMANN D.: Simulating virtual humans in networked virtual environments. *Presence: Teleoperators and Virtual Environments 10*, 6 (2001), 632–646.

[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.

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

[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.: Realtime 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.

Populating Virtual Environments with Crowds: Real-Time Crowd Rendering with Pre-Generated Impostors

S. Dobbyn and C. O'Sullivan

Interaction, Simulation and Graphics (ISG) 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

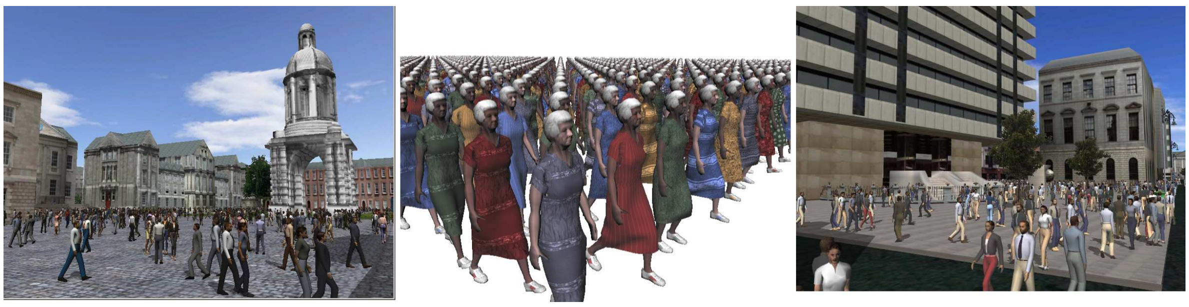


Figure 1: *Screenshots of the Geopostor system.*

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.

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

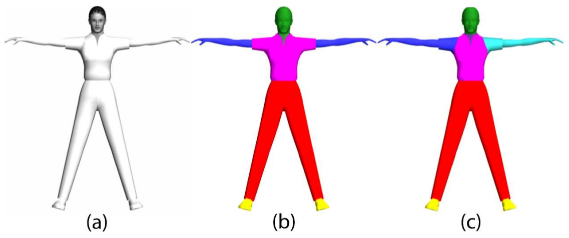


Figure 2: *(a) High LOD mesh representation in 3D Studio MAX. (b) The grouping of triangles based on material used (shown by the different colours). (c) The grouping of triangles based on the body part it represents (shown by the different colours).*

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.

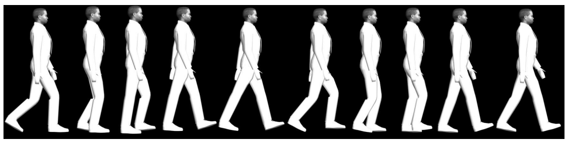


Figure 3: *Precalculating and storing the deformation of a mesh performing a walk animation for 10 key-frames.*

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. We assign barycentric texture coordinates to each triangle's vertices to provide an easy way to interpolate the normal at a specific point on a triangle. For each pixel, the triangle's ID (T_{id}) and its interpolated texture coordinate (u_i,v_i) at that pixel are stored in the *G-buffer*. Using the triangle's vertex normals ($\overrightarrow{NV_1}$, $\overrightarrow{NV_2}$ and $\overrightarrow{NV_3}$ which are accessed using T_{id}), the interpolated normal $\overrightarrow{N}$ at that pixel is calculated using Equation 1 and converted into a RGB colour ($Pixel_{RGB}$), using Equation 2.

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

$$\begin{aligned} Pixel_R &= ((0.5 * N_x) + 0.5) * 255 \\ Pixel_G &= ((0.5 * N_y) + 0.5) * 255 \\ Pixel_B &= ((0.5 * N_z) + 0.5) * 255 \end{aligned} \qquad (2)$$

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

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].

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.

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 3.

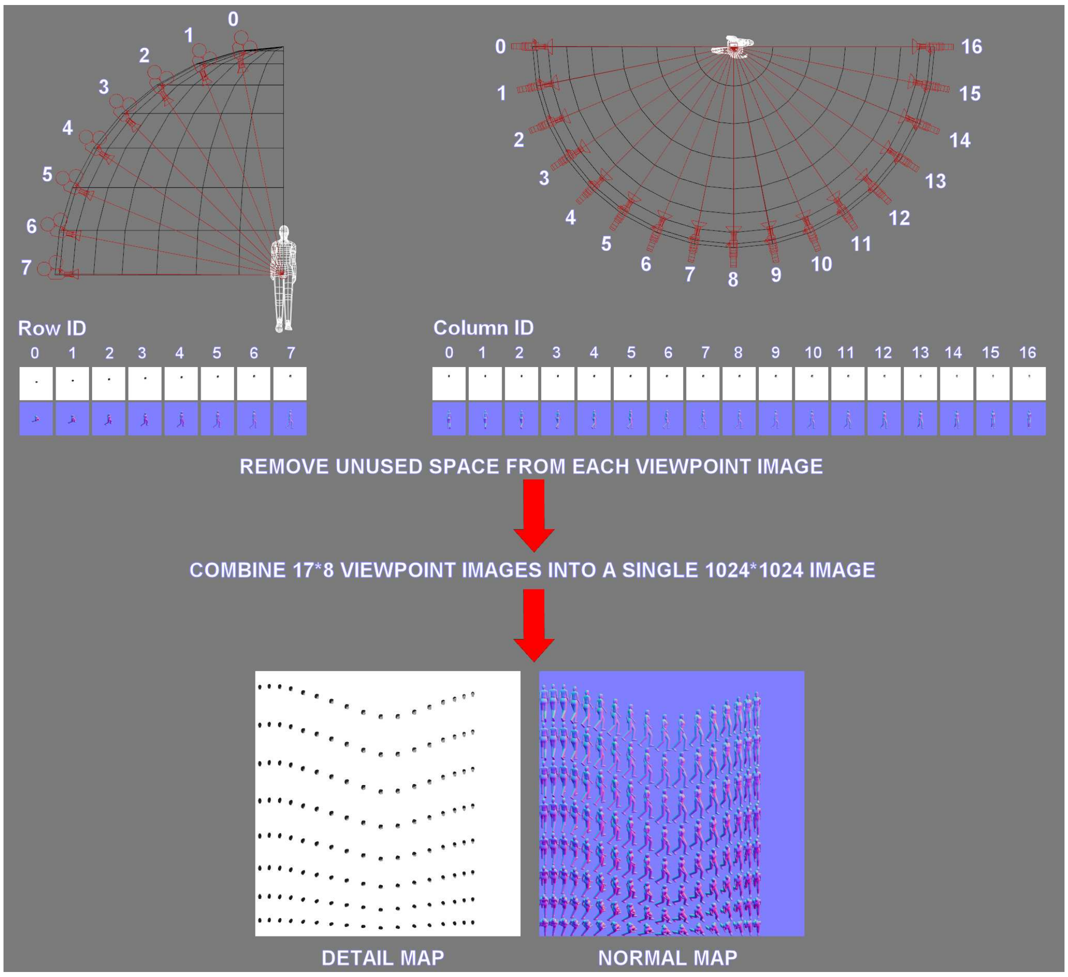

Row ID
Column ID
REMOVE UNUSED SPACE FROM EACH VIEWPOINT IMAGE
COMBINE 17*8 VIEWPOINT IMAGES INTO A SINGLE 1024*1024 IMAGE
DETAIL MAP
NORMAL MAP

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.*

$$\vec{N} = \frac{\vec{H} - \vec{C}}{|\vec{H} - \vec{C}|} \quad (3)$$

The vector from the camera to the human projected onto the ground plane $\overrightarrow{CH}$ can be calculated (Equation 4) using $\vec{N}$. It should be noted that in Equation 3, 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.

$$\overrightarrow{CH} = \frac{(N_x, 0, N_z)}{|(N_x, 0, N_z)|} \quad (4)$$

The amount by which to rotate the quadrilateral around the x-axis θ_x and y-axis θ_y is calculated using Equation 5. The viewpoint's row and column ID (V_{Row} and V_{Column}) can be used to lookup which viewpoint to render using Equation 6, where N_x and N_y are the number of viewpoint images pre-generated around the x- and y-axis.

$$\begin{aligned} \theta_x &= \cos^{-1}(N_y) \\ \theta_y &= \cos^{-1}(CH_z) \end{aligned} \quad (5)$$

$$\begin{aligned} V_{Row} &= \theta_x \times \frac{N_x}{90} \\ V_{Column} &= \theta_y \times \frac{N_y}{180} \end{aligned} \quad (6)$$

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 7.

$$\begin{aligned} Pixel_{Colour} = {} & DetailTexture_{RGB} * \\ & (Ambient_{LightModel} * Ambient_{Material} + \\ & (MAX(Vector_{Light} \cdot Normal_{Vertex}), 0) * \\ & (Diffuse_{Light} * Diffuse_{Material})) \end{aligned} \tag{7}$$

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 8 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.

$$\begin{aligned} Pixel_{Colour} = {} & DetailMap_{RGB} * \\ & (Ambient_{LightModel} * Ambient_{Material} + \\ & (MAX((Vector_{Light} \cdot NormalMap_{RGB}, 0) * \\ & (ColourMap\,[DetailMap\ \alpha]) * Diffuse_{Light})) \end{aligned} \tag{8}$$

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 3), 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 an OpenGL application that allows the artist to select particular colors for each body material from a colour palette (see Figure 6).

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].

6. 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,

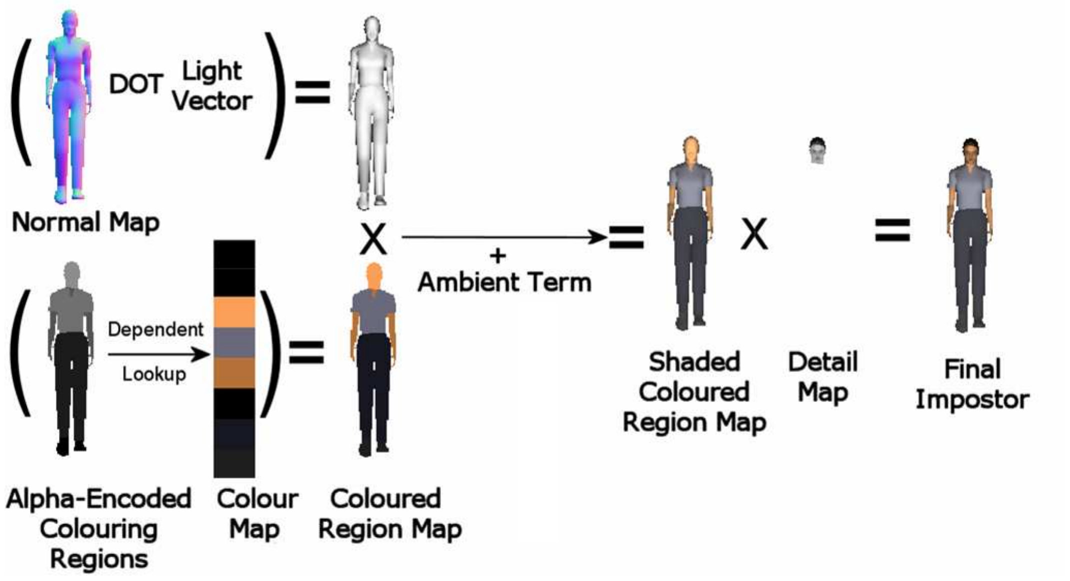


Figure 5: *Impostor shading and colouring sequence.*

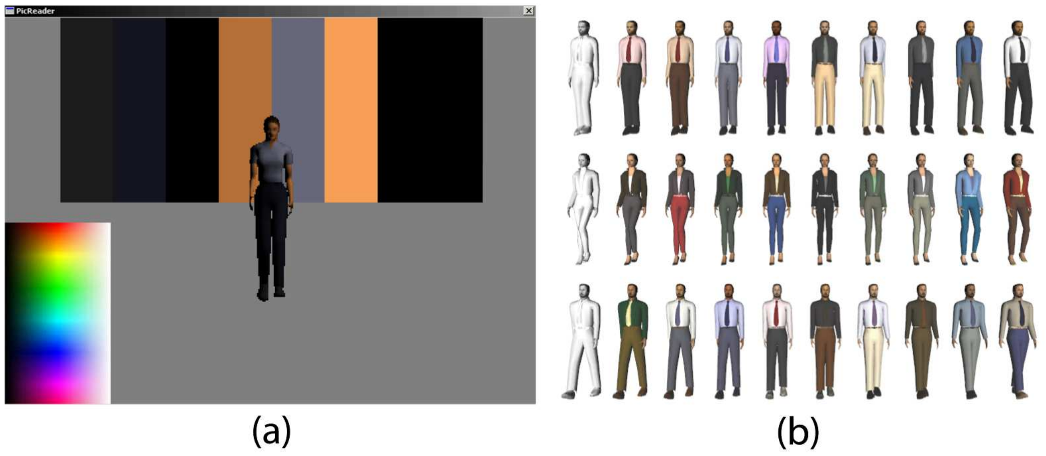


Figure 6: *(a)Example of an artist in progress of generating an outfit for a model using the 'Outfit Editor' application. (b) Nine outfits for three template meshes.*

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 8). 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

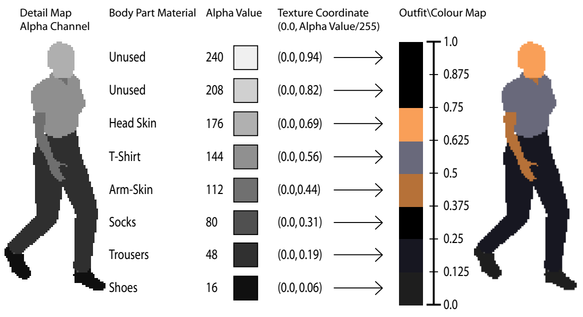


Figure 7: *Using programmable texture addressing to add variety to the impostor representation.*

can appear caused by the layering of the mesh on top of the impostor, these can be removed through blending.

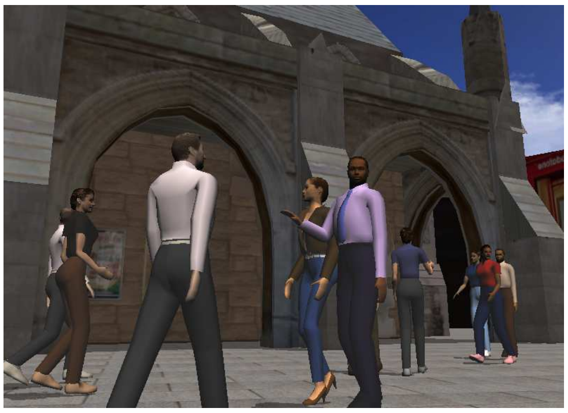


Figure 8: *Adding variety to the virtual human model's animation by layering head and arm gestures on top of the default walk animation.*

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

7. 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 9 (see [Bli88] [HMAM02] for the derivation of the matrix).

$$S = \begin{pmatrix} D - L_x * N_x & -L_x * N_y & L_x * N_z & -L_x * d \\ -L_y * N_x & D - L_y * N_y & -L_y * N_z & -L_y * d \\ -L_z * N_x & -L_z * N_y & D - L_z * N_z & -L_z * d \\ -L_w * N_x & -L_w * N_y & -L_w * N_z & D - L_w * d \end{pmatrix} \quad (9)$$

$$\textit{where } D = N_x * L_x + N_y * L_y + N_z * L_z + d * L_w$$

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 9 (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 10. The projection of the impostor onto the ground plane with respect to the light position can be calculated using $\overrightarrow{N}$ and Equations 4 and 6 (previously

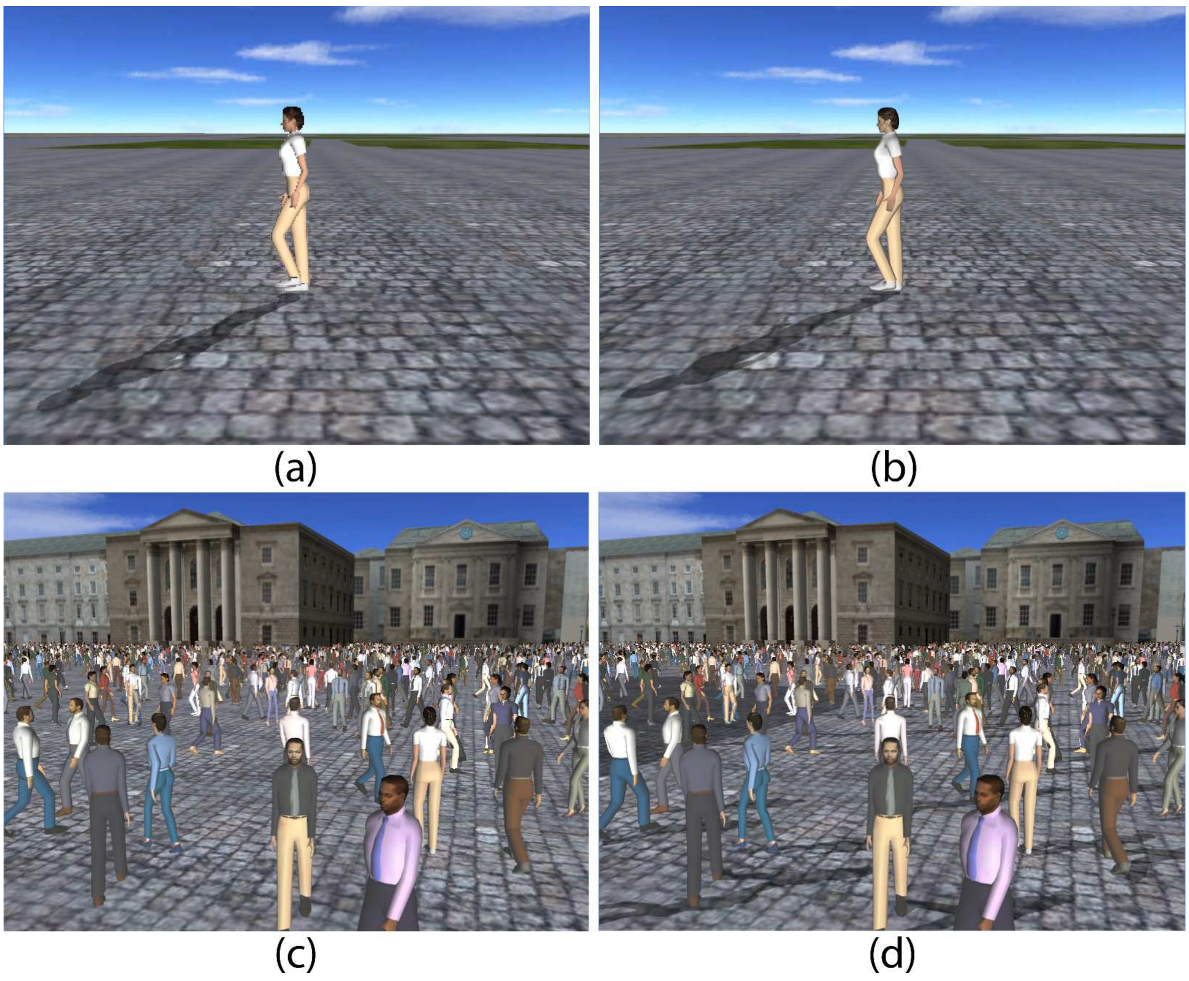


Figure 9: *(a) Projected impostor shadow. (b) Projected mesh shadow. (c) Crowd and city without shadows. (d) Crowd and city with projected LOD shadows.*

described in Section 4). The impostor's shadow requires no more than a single textured quad, and therefore is extremely fast to render.

$$\vec{N} = \frac{\vec{H} - \vec{L}}{|\vec{H} - \vec{L}|} \tag{10}$$

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 9 (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 the constructed shadow volume for an impostor is incorrect as a result of being a semi-transparent quadrilateral.

8. Performance Optimisations

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

8.2. 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.

8.3. 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.

8.4. 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 10).

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 11, 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

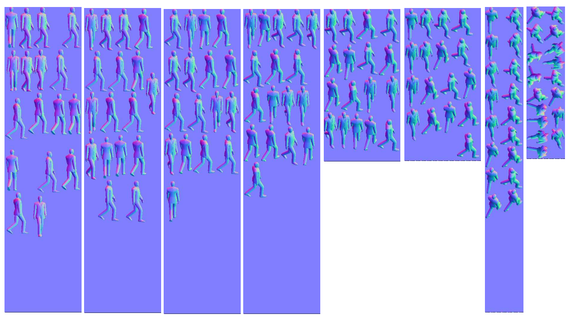


Figure 10: *Normal map split into smaller elevation images.*

reduced dramatically (see Equation 11). Taking advantage of the occluding nature of city environments, this method of separating impostor and normal map images for each elevation permits greater variety, without texture thrashing, as a result of each human model type consuming less texture memory.

$$\theta_E = tan^{-1}(\frac{h_{cam}}{d_{xz}}) \qquad (11)$$

9. 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 9), 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.*

[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.

[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.

[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).

[HMAM02] HAINES E., MÖLLER T., AKENINE-MOLLER T.: *Real-Time Rendering*. A.K. Peters, 2002.

[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.

[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).

[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.

Populating Virtual Environments with Crowds: Perceptual Evaluation of Virtual Human Models

S. Dobbyn and C. O'Sullivan

Interaction, Simulation and Graphics (ISG) Lab, 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), there has been no research conducted on perceptually evaluating them. However, evaluating these representations based on the plausibility of visual appearance and motion would provide a useful metric to help developers of LOD-based crowd systems improve its visual realism while maintaining real-time frame rates. With regards to improving our crowd system, we carried out perceptual evaluation experiments on various virtual human representations using experimental procedures from the area of psychophysics*.*

1. Psychophysics

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

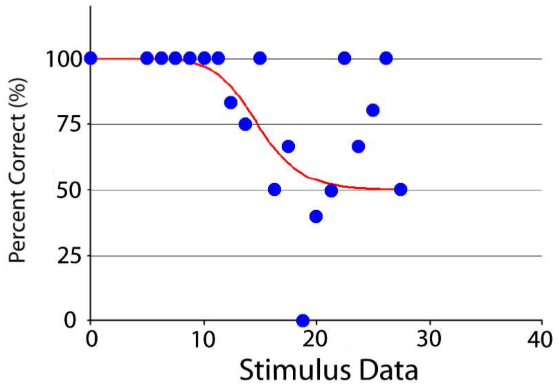


Figure 1: *An Ogive function fitted to a participant's data for a yes-no design.*

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).

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

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

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

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 a experiment using a yes-no and 2AFC experiment, respectively. Finally, ANalysis of Variance (ANOVA) is used to test the null hypothesis that two means are equal. The null hypothesis is rejected if there are significant differences between the means.

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

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

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 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

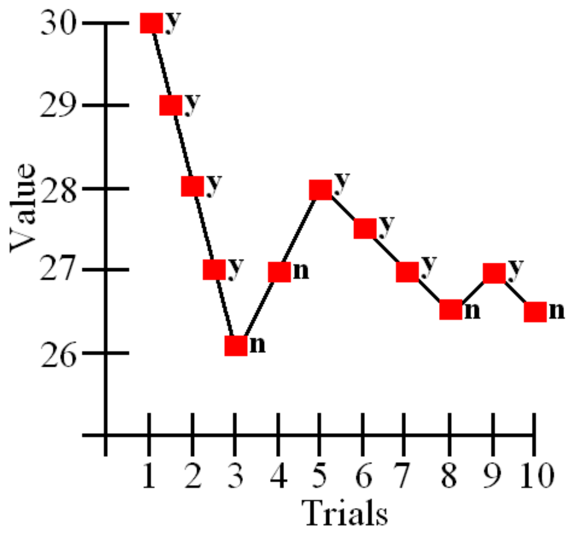


Figure 2: *Example of the stepping procedure for an up-down staircase terminated after four reversals.*

each correct response. With weighted up-down methods, different step-sizes for upward and downward steps are used.

While there has been little previous work related to the perception of virtual human representations [HDMO05, 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. [HRD04] 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 three perceptual experiments:

1. **Experiment 1: Impostor Vs. Mesh Detection Experiment**
   At what distance can experiment participants detect that a virtual human is using an impostor or mesh representation?
2. **Experiment 2: Low Vs. High-Resolution Mesh Discrimination Experiment**
   At what distance and at what resolution can experiment participants discriminate between a high resolution and low resolution mesh representation?
3. **Experiment 3: Impostor/Mesh Switching Detection Experiment**
   At what distance can experiment participants detect an impostor switching to a mesh?
4. **Experiment 4: Perception of Human Motion**
   How well do different virtual human representations replicate motion?

2. Experiment 1: Impostor Vs. Mesh Detection Experiment

2.1. Experiment 1: Aim

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

In this experiment, we aimed to establish the distance at which a virtual human's pre-generated impostor is perceptually equivalent to its mesh. In order to establish this distance threshold, we simultaneously presented two virtual humans using the impostor and mesh model at various distances to the experiment participant, and tested the participant's ability to detect which virtual human was using which representation. By recording the participant's responses at each distance the virtual humans were displayed at and plotting a psychometric function to this data, this distance threshold was estimated from the fitted curve using the PSE. This PSE signifies the distance at which the participant is likely to choose either representation with equal likelihood, and therefore provides a good estimate to the distance at which the impostor is perceptually equivalent to the mesh (for that person).

The goal in establishing such a threshold was to provide us with a guide to the distance at which both representations could be displayed in our system without a user detecting the impostor. This distance can be calculated in terms of a pixel to texel ratio i.e., where the ratio of the screen size of an impostor quadrilateral to the size of the viewpoint image equals a certain threshold [DHOO05]. A texel (TEXtured ELement) is the basic unit of measurement when dealing with texture mapped 3-D objects. When a texture map is loaded into texture memory as an array of n×m texture elements, each element is referred to as a texel. When a 3-D texture-mapped object appears close to the viewer so that the texture elements appear relatively large, there may be several pixels in each texel and the pattern of the texture map is easy to see. Therefore, it was hypothesised that beyond the point of one-to-one pixel to texel ratio, the participants would be unable to detect the impostor, as aliasing starts to occur when the size of the impostor's quadrilateral is greater than the size of the texture-mapped image, resulting in the stretching of the image on the impostor's quadrilateral.

Often it is difficult to find the exact experimental parameters and variables for a staircase procedure (such as step-size, maximum and minimum stimulus levels etc.). A study was first carried out using a weighted up-down experimental procedure followed by a second study on a different set of participants using a transformed up-down experimental procedure. In order to validate and fine tune results found in the first study, which produced approximate threshold ranges, we exploited our earlier findings to find the exact pixel to texel ratio in the second study.

2.2. Experiment 1: Apparatus and Participants

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

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

2.3. Experiment 1: Visual Content and Procedure

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

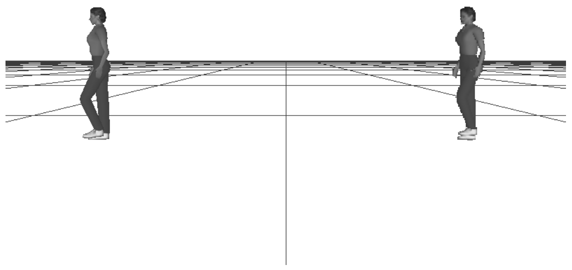


Figure 3: *Virtual human impostor Vs. mesh detection experiment.*

The participants were shown the mesh and pre-generated impostor model side by side at different distances from the participant (as shown in Figure 3) for 5 seconds. The virtual humans were separated by a fixed number of screen pixels to keep the distance between the representations constant. Both representations were animated with the same one second walk-cycle consisting of one keyframe of animation every 100 milliseconds. Since applications containing virtual humans would typically involve displaying them from multiple viewpoints, both virtual humans were rotated at 5.625 degrees every 100 milliseconds in a randomised direction around the y-axis so that the participant was not comparing the representations based on a single viewpoint and therefore eliminating directional bias. It should be noted that all models were displayed in grey-scale, as in these experiments we wished to determine people's ability to detect detail. Colour would complicate this issue by introducing further confounding factors, so we left this aspect for future examination.

This experiment employed a two alternative forced choice (2AFC) design, whereby the participant was asked to choose which virtual human "looked better" at each distance. We considered the virtual human using the mesh representation to be the "correct" response. Depending on the participant's response, the distance at which to display the virtual humans was either increased (for a correct response) or decreased (for a incorrect response). To avoid the participant guessing the direction of change in the virtual humans' distance, we used randomly interleaved ascending and descending staircases. Each staircase terminated after twelve reversals, where a reversal occurred whenever the participant changed his response from correct to incorrect and vice-versa.

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

2.4. Experiment 1: Results

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

The large amount of diverging data is thought to be a result of a flaw related to using a 2AFC task, whereby a series of lucky guesses at low stimulus levels i.e., when the representations are far away from the viewer, can erroneously drive the staircase to levels that are too low [Kle01]. On further analysis of this diverging data, it was discovered that the ascending staircases were not able to recover from a string of lucky guesses. However, it was found that, for the majority of the descending staircases, the minimum and maximum distances of the last 4 reversals overlapped with the results of both staircases for the converging data.

A psychometric curve ranging from 100% to 50% was fitted to each participant's experimental data using Equation 3. Using Equation 4 and Equation 6, the PSE and JND for each study were calculated as the 75% and 87.5% levels on the curve, respectively. At this PSE, the participant will judge the representations with equal likelihood as "looking better". The corresponding pixel:texel ratio values were calculated for the PSE and JND using Equation 7. The one-to-one pixel to texel ratio equivalent distance for the virtual human model used in this experiment is listed in Table 1 along with the two-to-one distance for comparison.

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

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

Table 1: *One-to-one pixel to texel ratio distances for the virtual human with a 45° Field of View at 800x600 Resolution.*

The mean PSE and JND for the 9 participants in the first study were 1.4 ± 0.142 and 0.492 ± 0.152 (see Figure 4(a)). The mean PSE showed that users perceived the impostor representation of human models at a distance greater than the hypothesized ratio (a pixel to texel ratio of 1.4:1). However, the mean JND is quite large indicating that the participants were not sensitive to small changes to the pixel to texel ratio at which the impostor was being displayed. The mean PSE and JND for the 16 participants in the second study were respectively 1.164 ± 0.064 and 0.1 ± 0.01 (see Figure 4(b)). The mean PSE is close to the hypothesized value of one-to-one, and this result represents an improvement on the first study. This change in results is attributable to learning from the first study's experimental results and adapting the psychophysical procedure accordingly. Since only one virtual human model was used for this experiment there were no means to compare, hence an ANOVA was not performed.

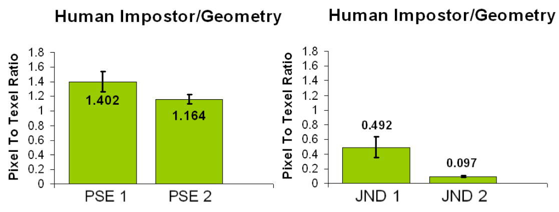


Figure 4: *Results of the impostor Vs. mesh detection experiments (showing PSE and JND in terms of pixel to texel Ratio) using: (a) a 3:1 weighted up-down procedure (PSE 1, JND 1) and (b) a 3U-1D procedure (PSE 2, JND 2).*

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

3.1. Experiment 2: Aim

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

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

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

3.2. Experiment 2: Apparatus and Participants

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

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

3.3. Experiment 2: Visual Content and Procedure

The same OpenGL test application as in the experiment in Section 2 was used to present the two virtual humans using the high resolution and a low resolution mesh to the participants for 5 seconds. The high resolution mesh consisted of 2170 triangles, and nineteen low resolution meshes were generated from this original mesh by hand. Solely using automatic simplification would result in losing important vertices needed to maintain the appearance of the model under motion, especially for the very low resolution models, which would subsequently bias the experiment's results. Therefore, the low resolution meshes were automatically simplified with manual intervention to keep the integrity of the really low resolution meshes. Using the 3D Studio MAX *multires* modifier, nineteen low resolution meshes were generated in this manner, ranging from a reduced vertex percentage of 60% to 15% at intervals of 2.5%. Preliminary observations were used for setting the appropriate range of resolutions for the low LOD mesh. A minimum vertex percentage of 15% was selected, as this was the maximum amount which the mesh's detail could be reduced while still retaining the general shape and motion of the virtual human's high resolution mesh. The corresponding number of vertices and triangles for each resolution model are shown in Table 2.

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

Table 2: *Low resolution mesh details*

The participants were simultaneously shown two virtual humans side by side using the high resolution mesh and a low resolution mesh at 3 specific distances from the participant. The virtual humans were separated by a fixed number of screen pixels to keep the distance between the representations constant. As in the experiment in Section 2, both models were animated with the same one second walk-cycle and were rotated by 5.625 degrees every 100 milliseconds in a randomised direction around the y-axis, to eliminate directional bias. All models were displayed in grey-scale, as in these experiments we only wished to determine people's ability to discriminate loss of detail.

This experiment consisted of 3 pairs of ascending and descending staircases randomly interleaved, where each pair displayed the virtual humans at one of the three distances from the participant. A yes-no design was employed,

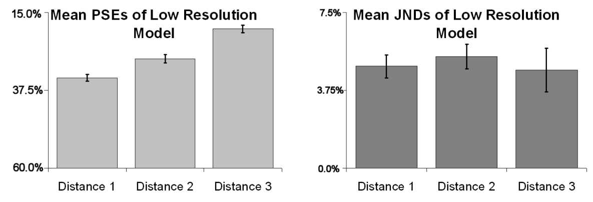


Figure 5: *Mean PSE and JND values for low resolution models.*

whereby the participants were asked to indicate whether the virtual humans looked the "same" (no response) or "different" (yes response) by pressing the respective left or right trigger button on a USB gamepad. For each staircase, a simple up-down stepping procedure was employed i.e., each time the participant indicated a "same" response, the resolution of the low LOD mesh was decreased by the step-size, otherwise a "different" response increased the resolution by the step-size. Each staircase ran for twelve reversals i.e., each time the participant's response changed. An adaptive step-size was used, where the initial step-size was only halved once after the first reversal for each staircase.

The ascending staircase began with the low LOD mesh displayed at the highest resolution of 60%, and the descending staircase started with the low LOD mesh displayed at the lowest resolution of 15%. The initial resolution step-size was set to 5% and, after the first reversal, the final step-size was 2.5%. As mentioned previously, 3 distances at which to display the representations from the viewer were chosen. The first distance was 5 units, and the other 2 distances were calculated based on a percentage of the representation's initial screen-space size at the first distance. The second distance was 8 units and the third distance was 16 units which corresponded to 66.6% and 33.3% of the representation's initial screen space size.

3.4. Experiment 2: Results

For each staircase, we recorded each participant's response for each mesh resolution displayed, as well as the resolution at which the 12 reversals occurred. We eliminated any diverging data using the same method as in Section 2. In this experiment, out of the 18 participants, 16 converged for the first distance, 13 converged for the second distance and finally 12 converged for the third distance.

Using the converging data, for each pair of staircases, we calculated the percentage of yes responses for each resolution displayed, and plotted this as a function of the resolution. For this experiment, a psychometric function based on Equation 1 was used to fit a curve to the data set and we subsequently calculated each participant's PSE and JND. The mean PSE and JND for each distance were calculated and are shown in Figure 5. The corresponding number of vertices and polygons for the PSE are shown in Table 3.

From this study, we found that, for this human mesh model, a low resolution mesh is perceptually equivalent to the high resolution mesh at a vertex percentage of 36.4% for a distance of 5 units, 31.7% for a distance of 8 units, and 22.5% for a distance of 16 units. A single factor ANOVA comparing the mean PSE averaged over 12 participants for the 3 distances, revealed a statistically significant difference between the mean PSE values for the 3 distances (see Table 4). This difference shows that distance affected perception of the low resolution mesh's visual appearance, with participants being able to discriminate better between different resolution meshes at closer distances. There was no significant difference between the mean JND values, which indicates that the same amount of stimulus change had to be added to the stimulus level at each distance in order for the participant to notice a difference. What is interesting about this result is that people were equally sensitive to the amount of vertex percentage difference, irrespective of distance.

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

Table 3: *Mean PSE and JND for low resolution geometry*

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

Table 4: *PSE comparisons for distance*

4. Experiment 3: Impostor/Mesh Switching Discrimination Experiment

4.1. Experiment 3: Aim

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

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

The goal in establishing such a threshold was to provide us with a guide to the distance at which the switching between our impostor and mesh representation should occur in order to reduce any noticeable popping artefacts and therefore maintain the realism of our crowd. This distance can be calculated in terms of a pixel to texel ratio (see Section 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.

4.2. Experiment 3: Apparatus

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

4.3. Experiment 3: Visual Content and Procedure

The same OpenGL test application as in the experiment in Section 2 was used to present the virtual human, switching between the impostor and mesh representation, to the participants. For each trial, the same model used in the first experiment was displayed, starting at a specific distance from the viewer, then moving at a constant speed towards the camera, and finally stopping at a specific distance. At some point during the interval the model switched from an initial impostor representation to a mesh representation. The virtual human was horizontally positioned at the center of the screen, animated with the same walk cycle used in the other experiments, and again displayed in grey-scale.

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

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

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

4.4. Experiment 3: Results

For the first case, where the virtual human faced the viewer, there were seventeen experimental participants (13M-4F, ages 12-39), 10 of whose experimental data converged properly. For the second case, where the virtual human spun, there were 10 experimental participants (8M-2F, ages 12-39), nine of whose experimental data converged properly.

All participants had normal or corrected to normal vision, and were both familiar and unfamiliar with graphics. For each staircase, we recorded the participants' responses for each trial's switching distance, as well as the distance at which the 12 reversals occurred. We eliminated any diverging data using the same method as in Section 2.

A psychometric curve ranging from 100% to 50% was fitted to each participant's experimental data using Equation 3 where $\gamma = 0.5$. The mean PSE calculated (shown as PSE1 in Figure 6), was approximately the predicted one-to-one value with a small mean JND (shown as JND1), indicating that the participants were quite sensitive to subtle changes in the pixel to texel ratio at which the popping occurred. The mean PSE calculated for the second experiment (shown as PSE2), was less than for PSE1, suggesting that the spinning was a distracting factor. However, the differences were not significant for the PSE ($F_{1,17} = 1.46, P > 0.3$) or the JND values ($F_{1,17} = 0.22, P > 0.7$). The large number of diverging results in the first case, however, suggests that the participants noticed other artefacts, which were masked in the second case when the virtual human was spinning.

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

5. Experiment 4: Perception of Human 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. We found in Hamill [HDMO05] that the motion of the impostor accurately replicated the motion of the high resolution model. We now test whether or not the low resolution polygon mesh replicates the motion of the high resolution mesh as accurately as the impostor, using the same psychophysical procedure. We also test the performance of a stick figure model to compare our results to those of Hodgins et al. [HOT98] and a point light source model as a baseline test.

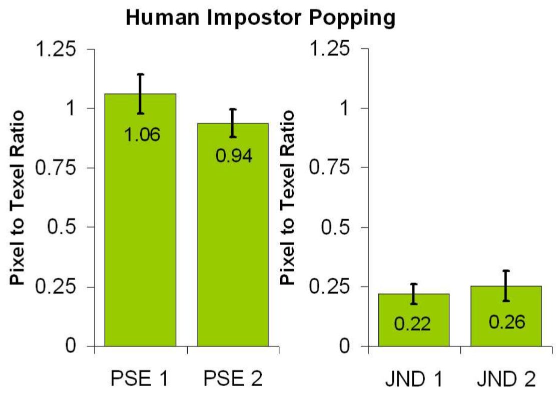


Figure 6: *Results of the popping detection experiments (showing the PSE and the JND in terms of pixel to texel ratio) for humans facing viewer (1) and spinning (2).*

5.1. Model Types

Five different representations of a male model were used (Figure 7). 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. The low resolution model was created by applying the 3D Studio Max *multires* modifier to the high resolution model. The modifier allows one to manually maintain part of the mesh at full resolution while reducing the LOD of the rest. Initially, we chose this option in order to keep a high number of polygons around the areas that would be deformed most by the joints. However, this manual selection is only necessary when simplifying meshes to very low resolutions, so we used automatic simplification. We automatically simplified the mesh as much as possible, without making the simplified version look different from the original, resulting in a mesh of 40% of the number of vertices of the original.

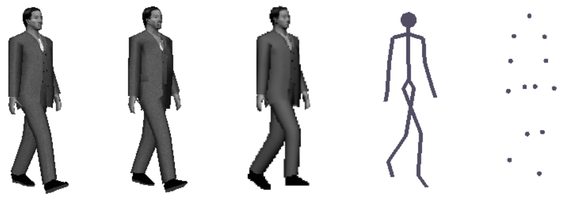


Figure 7: *High resolution, low resolution, impostor, stick figure and point light source model.*

Impostors were the third type evaluated and the same pre-generated approach was used as in the other experiments. All geometric and impostor representations were dynamically lit in the experiments. A stick figure was the next type and was created by drawing lines between the joints of the underlying skeleton. This representation was used in order to compare our findings with those of Hodgins et al. [HOT98]. Studies have shown that 13 moving light points, attached to the joints of the body, suffice to create a rich perception of a moving human figure [Joh73]. Using only 13 dots to display a human is the simplest representation and it is also the least computationally expensive of the 5 models, so we included this representation as the lowest LOD.

5.2. Method and Participants

We used a between-groups design for this experiment, where each group viewed a different model representation. This approach was chosen as we felt that, if allowed to view all of the models, the participants might base their judgments on characteristics of the models rather than the actual differences in motion, e.g., the impostor images contain artefacts along the edges which may cause the participant to focus on the artefacts instead of the overall motion if they had already seen the geometric model which showed less artefacts. Sixty-five participants (23 females, 42 males, aged between 17 and 35) took part in the experiment and were given book tokens as a reward for participation. They ranged from college staff, postgraduate students, undergraduate students and professionals and were all from different educational backgrounds. All participants were naive as to the purpose of the experiment and had normal or corrected to normal vision. The experiment was displayed on a 21 inch flat screen C.R.T. monitor. A grey-scale checkerboard floor plane was used so that the movement of the model could be seen clearly. All models were rendered in grey-scale in order to eliminate any bias due to colour that may have occurred. Lighting and rendering conditions were constant throughout the experiment.

5.3. Creating the Motion Variation

A reference motion *R* was created which consisted of 10 frames of a key-framed walk motion. This motion was cyclic and was repeatedly looped until 4 seconds of animation were recorded. The 10 frames of *R* were modified a number of times to create the arm, leg, and torso motion variation sequences. The arm and the torso variations were chosen as they were also used by Hodgins et al. [HOT98].

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. *kAl* and *kA2* were the keyframes in *R* where the arms were furthest away from the body (Figure 8). *kA1* and *kA2* were modified by a fixed amount 10 times, and the resulting 10 motions represented

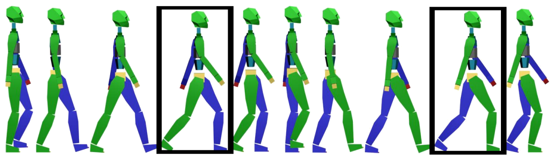


Figure 8: *Ten frames of animation of the reference motion R. kA1 is the frame highlighted on the left, kA2 is the frame highlighted on the right.*

the 10 different steps in the staircase analysis. 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. The right arm was altered by the same amount in the reverse direction. The pose of the skeleton at *kA1* was then copied and the inverse pose was pasted onto the skeleton at *kA2*.

The 10 altered biped motion sequences were then exported and loaded into an OpenGL rendering system and applied to our high and low resolution models with deformable meshes. The stick figure and point light source models were also rendered using these motion sequences. All of the altered motions were cyclic and looped until 4-second movies could be recorded. Forty of these movies were recorded (10 for each model type). The 10 impostor sequences were then rendered from the high resolution polygon model and recorded as movies. A similar test was conducted to test the ability of the participants in distinguishing larger and smaller leg motions for all representations. A further set of 50 motion sequence movies was created in a similar manner to the arm motions, except that the leg was altered by iterative translations along the longitudinal and vertical axes. Finally, the ability of the participants to distinguish modifications to the torso was tested. A further 50 movies were created by making kinematic alterations to *R*. 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.

5.4. 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 9). They were told to judge difference based on the overall motion of the character and not to focus on colour, speed or any other factors which were constant throughout the experiment. 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. Five groups of 13 participants took part, with each group seeing a different model representation.

Figure 9: *Participant taking part in the experiment.*

Experimental data was gathered using a staircase procedure. This experiment consisted of 3 ascending staircases and 3 descending staircases randomly interleaved, i.e., an ascending and descending staircase for each of the motion variation types. The ascending staircases began with a comparison of the reference motion *R* to itself, and the descending staircases began with a comparison of *R* with the most exaggerated motion sequence (i.e., step 10 of the staircase). For the ascending staircases, a simple up-down staircase was employed so that, for every correct response, 2 steps were added to the current step, and for every incorrect response, 1 step was subtracted from the current step. A reversal occurred when the participant made a different decision on the comparison from the decision they made about the previous comparison. We adapted the step-size after the first reversal so that only 1 step was added or subtracted. We felt that this refinement would keep the comparisons occurring in the area of interest i.e., close to the point at which they began to distinguish the differences in motion. Once the refinement to the step-size was made, the procedure was continued until 8 reversals were recorded. For the descending staircases, the same procedure was employed, but with the steps decreasing in the opposite direction. Staircases were randomly interleaved, and participants were randomly shown either *R* or the motion sequence at the current step-size. This gave a 50% detection threshold which is used to estimate the PSE.

5.5. Joint Weighting

The animations were created by altering certain joints by discrete amounts, but it was not immediately obvious how to compare the changes made to the arms with those of the legs or torso. Originally we used steps 0-10 for the arm, leg, and torso step-sizes. This scale told us nothing about the actual differences in motion, e.g., the arm moved only a small amount between steps whereas the legs moved a much larger amount. In order to be able to compare the performance of the different models used for rendering across all motion variation types, a scheme was needed to scale the variations and consequently the step-sizes. This involved finding a distance metric to compute the actual amount of motion change made at each step for the arm, leg and torso animations.

We took the approach of comparing the pose of the skeleton at the keyframe with the most exaggerated pose between two steps of the motion variation. The distance metric described in [LCR02] was used to compute the difference between the 2 frames of animation. This distance metric sums the weighted differences of joint orientations and velocities.

For the arm animations, we computed the distance metric between the most exaggerated arm pose of the skeleton at step θ_i, $0 \leq i \leq 10$, and the corresponding pose at step θ_{i+1}. Similarly for the leg and torso motion, we chose the most exaggerated animation keyframe at step θ_i and compared it to the corresponding frame in step θ_{i+1}.

Lee et al.'s cost function contains a parameter *wk* which adjusts the transition cost by weighting the differences in orientation of the joints. They report setting the weights to one for the important joints: shoulders, elbows, hips, knees, spine and pelvis; and all others are set to zero. We propose that the joints should be weighted depending on how much of a change is observed when that joint is moved. An empirical approach was taken to weighting the joints in an attempt to capture this observed change. We approximated the importance of the joints based on their projected pixel area. This area was computed by counting the number of pixels for each joint of the skeleton that was applied to all of the models at the appropriate camera angle and pose. The distance metric was computed for arm variation, leg variation and torso variation, and was used to scale the step-sizes of the experiments. It was found that the distance of the arm variations was ($dA = 0.049$), the torso ($dT = 0.483$) and the leg ($dL = 1$). The steps for the arm were then set to dA, $2dA$, $3dA$ up to $10dA$, and similarly for the leg and torso variations.

5.6. 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. The data for the ascending and descending staircases were combined, and a separate curve was created for each motion variation type for each participant. Psychometric curves were then fitted to the datasets and, for each participant, a PSE and JND were calculated from these curves. The PSE was the stimulus level value at the 50% detection level. 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.

Firstly, we will look at the performance for the model types over the whole dataset in order to get an overall picture. We will then look in more detail at the results for each of the motion variation types in the hope of gaining further insights. A two-factor ANOVA with replication was performed on the full dataset by collapsing all the recorded JND and PSE values over model type.

Results showed no significant differences between the mean PSE values of the participants when viewing different model types (Figure 10). This implied that the point at which people could notice differences in motion was the same for all model types. However, this measure gave no indication of their uncertainty. Similarly, 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 (Table 5). 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 11).

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. As expected, the perception of the lowest LOD model (the point light source model) was furthest from the high resolution model. The stick figure was a closer match, as it retained the links between the joints of the skeleton.

We then looked closer at the results for the arm, leg and torso motion variations. The arm motion was an example of a very subtle motion, as it was altered by only a very small amount each step, the torso was altered more than the arm and the leg motions were altered by a large amount each step. We used single factor ANOVAs to compare the means of the PSE and JND values between the model types. A summary of the ANOVA results can be seen in Tables 6 and 7. Figures 10 and 11 show an illustration of these results. The granularity chosen for the steps was due to the inability to dynamically create the impostor motions, each walk cycle at each level took approximately 1 hour to compute with a further 30 minutes of manual cleanup to recapture lost viewpoints in frames. As discussed previously, the results for the mean JND values of the full dataset highlighted a trend where the performance for the high resolution and the impostor were most similar, with the stick figure and the low resolution at the next level and the point light source model eliciting the worst scores. This same trend can be seen with the JND values for the Leg motion variation, indicating that the participants found it most difficult to notice changes in the leg motion on the lowest LOD model.

JND comparison for all model types	$F_{2,72}$	P
High vs. Low	4.3	< 0.05
High vs. Stick	4.8	< 0.05
High vs. Point	17.7	≈ 0
Impostor vs. Low	4.3	< 0.05
Impostor vs. Stick	4.8	< 0.05
Impostor vs. Point	18.0	≈ 0
Low vs. Point	4.6	< 0.05

Table 5: *ANOVA results of mean JND comparisons for model types (F_{crit} = 4).*

Leg PSE Comparision	$F_{1,24}$	P
High vs. Low	4.71	< 0.04
High vs. Stick	4.39	< 0.05
Impostor vs. Low	4.6	< 0.04
Impostor vs. Stick	4.6	< 0.04
Torso PSE Comparison	$F_{1,24}$	P
High vs. Impostor	18.86	< 0.0005
Impostor vs. Low	12.14	< 0.005
Impostor vs. Stick	7.11	< 0.01
Impostor vs. Point	13.35	< 0.001

Table 6: *ANOVA results of mean PSE comparisons (F_{crit} = 4.3).*

Arm JND Comparision	$F_{1,24}$	P
Impostor vs. High	7.52	< 0.01
Impostor vs. Low	7.3	< 0.01
Impostor vs. Stick	3.61	< 0.07
Impostor vs. Point	8.88	< 0.01
Leg JND Comparision	$F_{1,24}$	P
High vs. Stick	4.42	< 0.05
High vs. Point	11.31	< 0.003
Impostor vs. Point	9.55	< 0.005
Low vs. Point	5.25	0.03
Torso JND Comparison	$F_{1,24}$	P
High vs. Low	4.53	< 0.04
High vs. Point	10.83	≈ 0
Impostor vs. Low	5.68	< 0.03
Impostor vs. Point	13.08	< 0.001
Stick vs. Point	5.19	< 0.03

Table 7: *ANOVA results of mean JND comparisons (F_{crit} = 4.3).*

A similar overall trend is present for the arm and torso, as illustrated by the mean JND values. However, surprisingly, the impostor appears to be the most perceptible for the arm motion variation, which may be due to the fact that it is planar and may have made the subtle arm motions more noticeable. Also, the response to the torso motion variation does

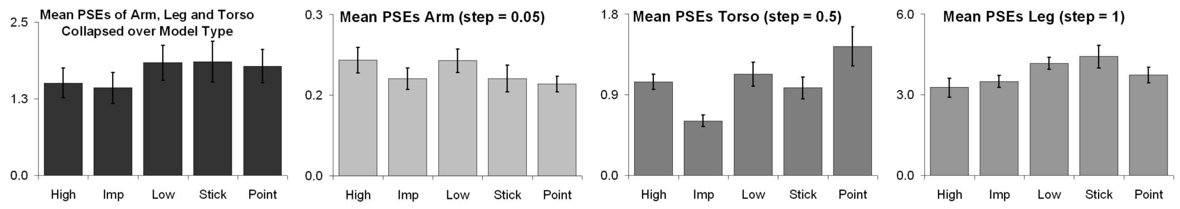


Figure 10: *(a) Mean PSE values for all motion variations collapsed over model type, (b) Mean PSE values for arm motion variation, (c) Mean PSE values for torso motion variation, (d) Mean PSE values for leg motion variation. The vertical axis shows differences in motion as estimated in Section 5.5.*

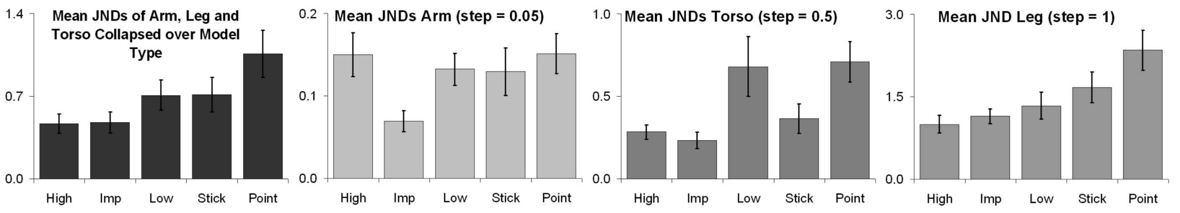


Figure 11: *(a) Mean JND values for all motion variations collapsed over model type, (b) Mean JND values for arm motion variation, (c) Mean JND values for torso motion variation, (d) Mean JND values for leg motion variation.*

not follow the overall trend, because participants were less able to notice these changes on the low resolution model than any other model representation. We attribute this to the fact that the torso motion is the motion which moves the most number of bones of the skeleton, and as the low resolution model had fewer vertices to move than the high resolution model, the resulting poor deformation had an effect on the perception of the motion.

6. Discussion of Evaluation Results

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

In Section 2, it was found that an impostor and its corresponding mesh representation are not perceptually equivalent at a distance of less than a 1.16:1 pixel to texel ratio when simultaneously displayed side by side. Additionally, in Section 4, it was found that people could detect a virtual human switching between its impostor and mesh representation at a distance of less than a 1.04:1 pixel to texel ratio. This is lower than in the previous experiment, probably because the two representations are never compared side by side. These results provide developers with a metric of approximately a 1:1 pixel to texel ratio for the use of a pre-generated impostor representation. Also, these test results could be considered conservative, since the complexity of the test scene was extremely basic. We hypothesise that, in more complex scenes containing several hundred humans, switching between an impostor and a mesh representation could occur at a closer distance and we plan to test this hypothesis.

Low resolution meshes can be generated that are perceptually equivalent to the high resolution mesh at particular distances (Section 3). Since these models consist of fewer triangles, the rendering cost of these resolutions is substantially less when compared to that of the original high resolution mesh. These results suggest that the high resolution mesh should be replaced in our system with a simpler model (depending on the distance from the viewer), since the extra detail in the high resolution mesh is not perceived by the viewer, and therefore is unnecessary. The results also suggest that we are using a mesh that is too detailed (2,170 triangles) for the highest LOD at a distance less than 5 units.

In Section 5, we showed that a low resolution model was not perceptually equivalent to the high resolution model at conveying subtle variations in motion, whereas the impostor representation was. Therefore, if the application requires the motion of the models in the crowd to accurately replicate the motion of the high level geometry, designers should be aware that the lower the LOD of the character, the less likely it is to retain the motion of the high level geometry model. Impostors are better at replicating the motion of the high resolution model, but due to texture memory considerations, can only be used for a small set of pre-determined animation sequences. Table 8 summarizes the advantages and

Pre-Generated Impostor		Low Resolution Geometry	
Advantage	Disadvantage	Advantage	Disadvantage
Can appear visually equivalent to high resolution meshes	Limited to animation used in pre-generated sequences	Can appear visually equivalent to high resolution meshes	High rendering cost
Small rendering cost	Large texture memory consumption	Possible to have different animations	Doesn't replicate the motion of the high resolution geometry well
Replicates motion well		Texture memory consumption minimal	

Table 8: *Comparison of advantages and disadvantages of the 2 different low LOD representations.*

disadvantages of the two different representations, based on our results.

It was also found in Section 2, that impostors are perceptually equivalent to the high resolution model at a pixel to texel ratio of approximately 1.16, which corresponds to a distance of 12.416 virtual world units. However, low resolution meshes can be perceptually equivalent to their high resolution mesh at a closer distance. By using the results from Section 3, we can estimate the percentage of vertices at which to generate a low resolution mesh that is indistinguishable from the high resolution model at the same distance as the impostor. This corresponds to a low resolution mesh of approximately 27.5%. Due to the rendering cost of each model (see Table 9), we suggest that it would be advantageous to use the impostor instead of a low resolution mesh for virtual humans being displayed at a distance greater than the 1.16 ratio or acting as scene extras. The distances at which different LOD representations are perceptually equivalent to the highest resolution mesh is illustrated in Figure 12.

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

Table 9: *The distance at which $LOD_{Geometry}$ models are perceptually equivalent and their associated rendering cost.*

References

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

[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.

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

[HBF02] HARRISON J., BOOTH K., FISHER B.: Experimental investigation of linguistic and parametric descriptions of human motion for animation. *Computer Graphics International* (2002), 154–155.

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

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

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

[Joh73] JOHANSSON G.: Visual perception of biological motion and a model for its analysis. *Perception and Psychophysics 14* (1973), 201–211.

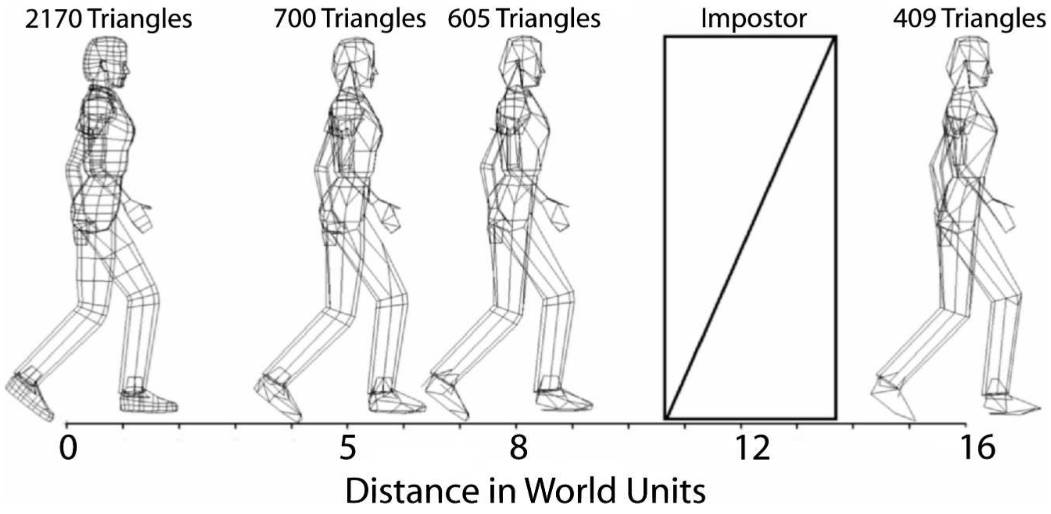


Figure 12: *Distances at which different LOD representations are perceptually equivalent to the highest resolution mesh.*

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

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

[Lev71] LEVITT H.: Transformed up-down methods in psychoacoustics. *Journal of the Acoustical Society of America 49*, 2 (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* (2001), 1330–1347.

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

[MDO05] MCDONNELL R., DOBBYN S., O'SULLIVAN C.: Lod human representations: A comparative study. *Proceedings of the First International Workshop on Crowd Simulation (V-CROWDS '05)* (2005).

[ODGK03] O'SULLIVAN C., DINGLIANA J., GIANG T., KAISER M. K.: Evaluating the visual fidelity of physically based animations. *ACM Transactions on Graphics (TOG) 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 (TOG) 22*, 3 (2003), 537–542.

[Tre95] TREUTWEIN B.: 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. *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/EUROGRAPHICS Symposium on Computer Animation* (2003), 232–238.

Populating Virtual Environments with Crowds: Navigational Strategies

S. Dobbyn and C. O'Sullivan

Interaction, Simulation and Graphics (ISG) Lab, Trinity College Dublin, Ireland

Abstract
Urban environments are far from being obstacle-free. Therefore, virtual humans wandering around this type of environment without obstacle avoidance would result in them walking through objects such as buildings and the overall realism of the simulation would be greatly reduced. This part of the tutorial will describe techniques implemented to simulate basic navigational strategies for characters.

1. Introduction

Typically, obstacle avoidance involves the virtual human checking whether its steering vector intersects an object and, if so, it steers in some way to avoid it. While the performance of this method can be improved by checking only those objects within the same node as the virtual human, this method would become computationally demanding as the number of humans and objects in each node increases. Given that we want to populate an environment full of static obstacles with a large number of virtual humans, the method of obstacle avoidance must be efficient enough so as not to unduly impact on performance.

2. Background

2.1. Path Finding

Path finding is necessary for humans to navigate the environment they inhabit in a successful and realistic manner. In order to do this, the environment needs to store pathfinding information across which a search can be performed. Typically, this is achieved by adding an invisible layer of nodes for the environment's terrain, where each node stores all accessible neighbouring nodes. Using this information, a virtual human can perform a search across these nodes for the shortest walkable path between its current position and goal position. While various search algorithms exists, such as simple breadth or depth first searches, A^* has become the standard in modern game development as it provides good, predictable performance without compromising optimality. To improve the realism of the path chosen, other important information can be stored in the nodes so that the search takes into account other heuristics in addition to distance, such as the danger element, or the terrain difficulty of a path. For a detailed discussion of the path-finding problem and the A^* algorithm, see [Sto96, HS02].

2.2. Obstacle Avoidance

To prevent the human colliding with the environment and other agents, Tecchia et al. utilize a space discretization approach [TC00]. Using a height-map to store the height of the environment at each point, the human performs collision avoidance if the difference in height between its current and next position is above a certain threshold. Otherwise, the human moves to its new position, updating his height above the ground based on the height-map. To avoid the humans getting too close to each other, a collision-map is used to store which positions in the map are occupied. The human's direction is adjusted depending on whether there are other humans in a 3×3 neighbourhood.

In [TLC01], Tecchia et al. extended their previous research on collision avoidance to use a platform that segments the virtual world into a 2-D grid in order to accelerate the development of agent behaviours. The 2-D grid is composed of 4 layers, where the grid cells in each layer contain specific data to govern the behaviour of individuals. The four types of layers are: inter-collision detection layer, collision detection layer, behaviour layer, and callback layer. The first two layers are used to compute collision detection between an agent and its environment or with other agents, while the other two layers provide more complex and individual behaviours. The behaviour layer encodes specific behaviour in each grid cell as a colour. Depending on the cell that the agent is inhabiting, the colour-encoded behaviour instructs the agent to perform simple actions such as wait, or turn left. The callback layer provides for more complicated agent-environment behaviour using an event-driven approach. This allows an agent to perform actions, such as waiting at a bus stop and getting onto the bus, by activating the callback when the bus arrives. The main advantage of this layer is that, even though the associated behaviours are quite complicated, they are only executed when needed.

2.3. Steering Behaviours

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

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

Since the pioneering work of Reynolds [Rey87], methodologies from many fields have been employed to address the problem of motion control for virtual humans [Vin97,Rey99, Pot99]. Although AI techniques [HP88, BY95, FTT99] have shown very promising results, such methods do not scale well with the number of virtual humans or obstacles and therefore are not suited for real-time applications. Alternatively, learning, perception, and dynamics-based techniques are easily adaptable to dynamically changing environments [TT94, NRTT95, HP97, BMH98]. The idea of using force fields around virtual humans in order to guide them originated from work on path planning in robotics [GLM98,GLM99]. Egbert and Winkler proposed a force-field technique which used a vector field around objects to prevent collisions between them [EW96].

3. Path Finding

With a fixed-sized grid, each node needs to store only the node accessible in the North, South, East and West direction. The time taken for the A* search to find the shortest path can be reduced by constructing a grid of larger rectangular nodes of either walkable or blocked values (see Figure 1 (c)). Since the nodes are of variable size, several nodes can be accessible from a node in a particular direction. To take this into account, each node stores four separate lists of nodes, where each list contains the nodes accessible in one of the directions. For example, Figure 1 (b) and (d), shows the nodes returned by the A* search (shown in green), between its current position (shown in red) and its destination (shown in blue).

For virtual humans far from the viewer, the path stored in the *walk* task can be constructed as a series of straight lines connecting the position of each node in the path. Since the nodes are of different sizes, a node's position with respect to a neighbouring node is the mid-point of their connecting edge (shown in yellow in Figure 1 (e) and (f)). The virtual human is orientated towards the position of the next node, and the human is translated along this direction. Once the virtual human reaches a node in the path, its direction is oriented towards the next node and this continues until it reaches its final destination node. The problem with using a straight-line path is that it can result in the virtual human performing sudden large and unrealistic changes in the direction it is walking (see Figure 1 (e)). Therefore, it should only be used when these artefacts are imperceptible to the viewer. For a higher level of detail, a B-spline curve is plotted, using the path's nodes as control points, to provide smoother directional changes (see Figure 1 (f)). This requires additional computations to translate the virtual human along the curve, but results in a more realistic behaviour and therefore should be employed for important characters.

4. Steering Behaviour: Obstacle Avoidance

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

In an obstacle-free world, the new position of a walking virtual human (P') at each time-step can easily be calculated using its current position (P), its steering vector ($\overrightarrow{S}$) and the distance travelled (d) between its current keyframe and the last keyframe of its walk animation (see Equation 1).

$$P' = P + \overrightarrow{S} * d \qquad (1)$$

To avoid a virtual human walking in the same direction, we can rotate its steering vector at every time-step by certain

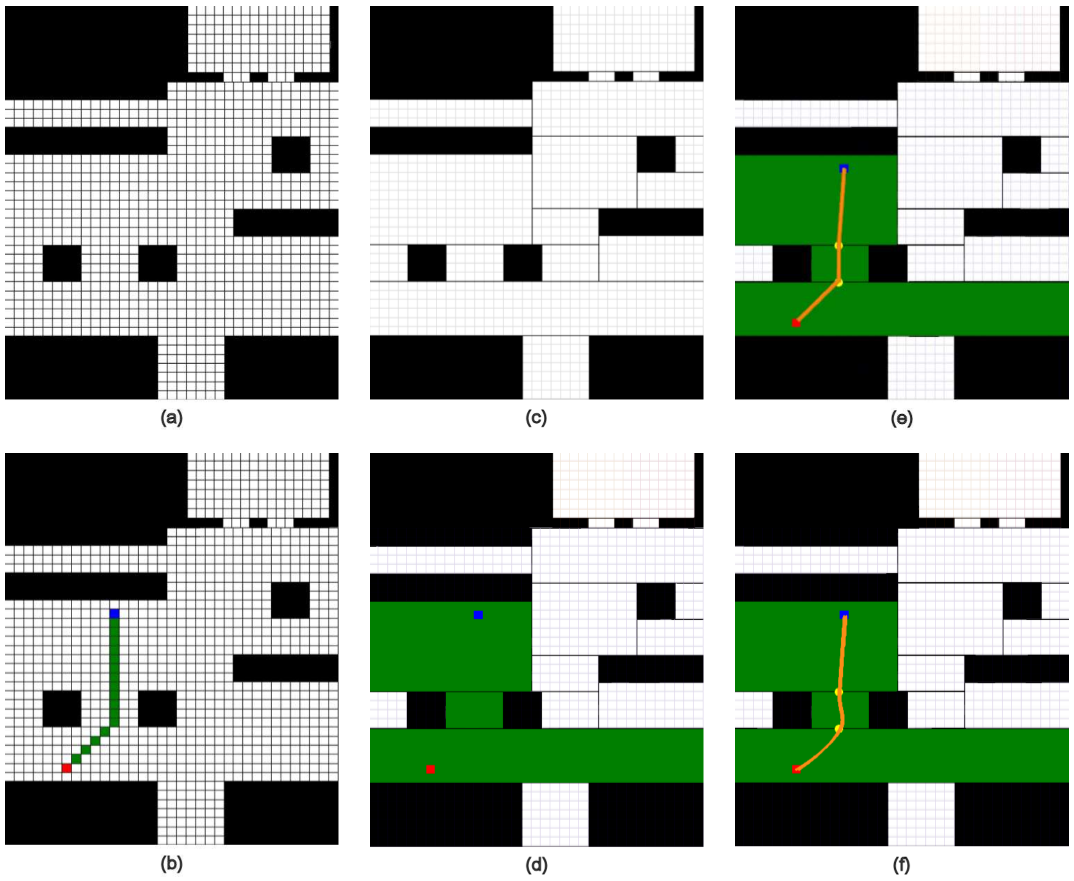


Figure 1: *(a) Grid consisting of fixed-size nodes, (b) Path returned by A* search for fixed-sized nodes, (c) Grid consisting of variable-sized nodes, (d) Path returned by A* search for variable-sized nodes, (e) Straight Line path (Low LOD) and (f) B-spline path (High LOD).*

amount, thus generating a random steering or *wandering* behaviour. However, this results in the directional vector oscillating and the virtual human appears to be unable to decide on where it is going. In order to make the wandering behaviour more believable, the directional vector is not changed every time-step but only every 200 milliseconds and this results in the virtual human wandering around its environment in a more coherent manner. Additionally, by constraining the maximum amount the steering vector can be rotated to $\pm$ 2.8125 degrees, the virtual human does not perform large unrealistic directional changes while it walks.

Due to the nature of the environment (a city filled with vertically rising structures), we perform collision detection in 2D using a top-down map of walkable areas similar to that described in [TLC02] and [LMM03]. We pre-generate a set of maps of walkable areas or *walk-maps* by capturing an orthographic top-down view of the city model in the OpenGL depth buffer. We store the city's pavements (including tunnels through buildings and pavements occluded by over-passing bridges) as walkable, and buildings, roads, and other obstacles as blocked, by selecting appropriate near and far planes when rendering the city model into the depth buffer. Each map is 800×600 pixels, where a pixel is white if it is walkable otherwise black if it is blocked (Figure 3) and corresponds to a physical area of 33×33cm. These 1 bit data (walkable or blocked) maps require 60KB of memory and sixty-three walk-maps for the 4.5km^2 area covered by the virtual city were pre-generated, requiring a total of 3.6MB of memory for the city.

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

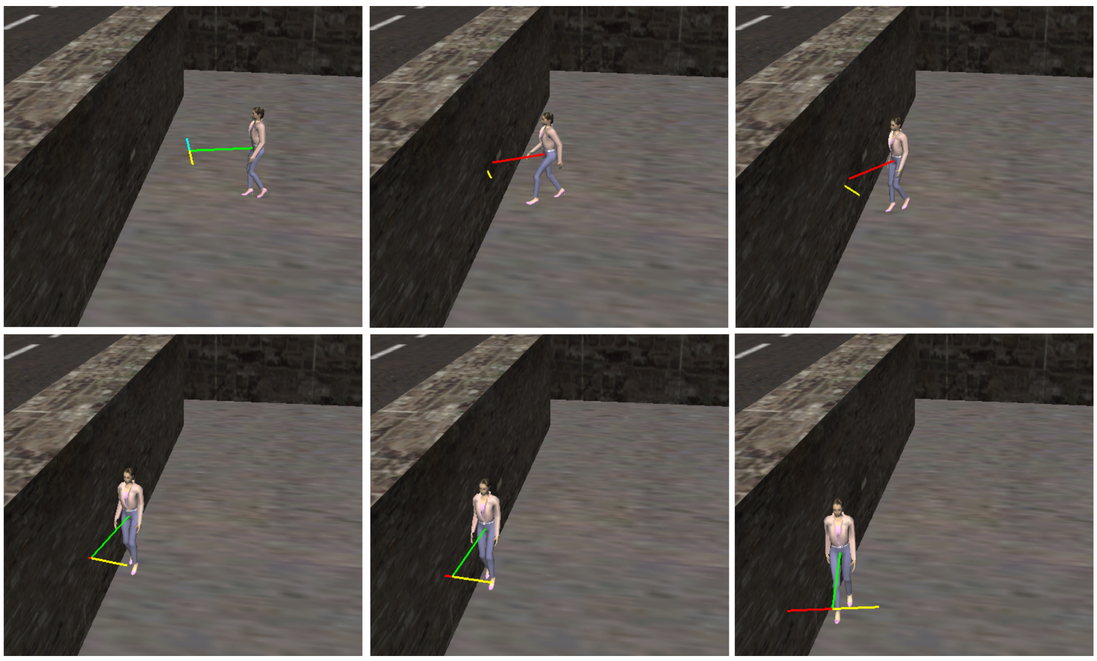


Figure 2: *Virtual human using its steering vector to avoid a wall.*

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

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

While this provides a more realistic collision avoidance behaviour (Figure 2), where the human seldomly needs to perform the unrealistic about turn, the problem with this approach is that it only takes into account what is free ahead of the virtual human, which can result in the human walking very close to and sometimes through obstacles that are to the side of it. Since the edges of buildings and roads are linear in nature, we can improve the realism of the obstacle avoidance by making sure the humans walk along these edges without getting too close to them. We implement this by looking up the positions to the left and right of its current position every second. If one of these positions is blocked, the human steers away from this area by rotating its steering vector in the opposite direction.

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

For each walkable pixel, the total force at that pixel ($\sum\overrightarrow{F}$) is calculated by summing the repulsion forces of any blocked pixels in a 7×7 neighbourhood affecting the walkable pixel. The summed force's normalised x and z component $\overrightarrow{F}_x$ and $\overrightarrow{F}_z$ are mapped to a value between 0 and 255 and stored in the red and green component, respectively, of the potential field map (using Equation 2). To store the magnitude of the summed forces $|\sum\overrightarrow{F}|$, the maximum magnitude of a force $|\overrightarrow{F}_{MAX}|$ exerted on a pixel needs to be calculated by summing all of the forces exerted on a walkable pixel inhabiting a completely blocked neighbourhood. Using Equation 2, $|\sum\overrightarrow{F}|$ is mapped between 0 and 255 and finally stored in the blue component of the potential field map.

$$\begin{aligned} R &= ((\overrightarrow{F}_x * 0.5) + 0.5) * 255 \\ G &= ((\overrightarrow{F}_z * 0.5) + 0.5) * 255 \\ B &= \frac{|\sum\overrightarrow{F}|}{|\overrightarrow{F}_{MAX}|} * 255 \end{aligned} \tag{2}$$

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

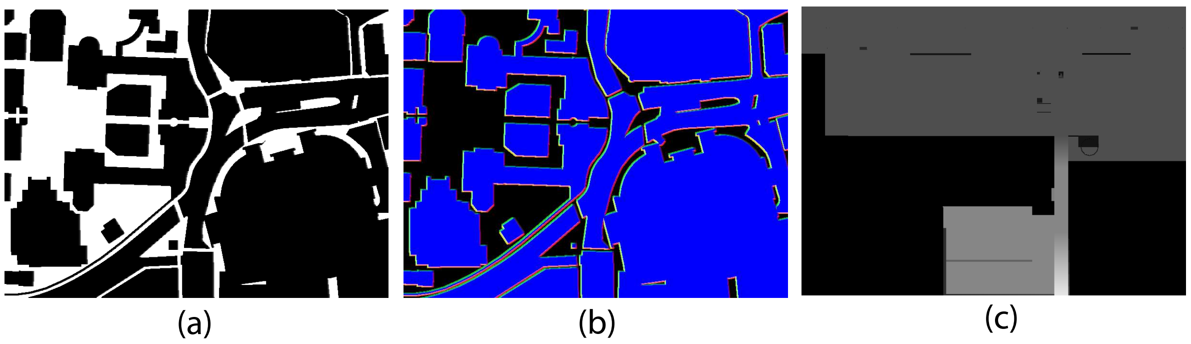


Figure 3: *(a) Walk map. (b) Potential field map. (c) Height map.*

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

Finally, in order to position the virtual humans so that they do not intersect with the ground, we use a *height-field map*. We pre-generate these images in a similar manner to the walk-maps, but the near and far planes are appropriately selected so that the height of the walkable areas are mapped to a value between 0 and 255 in the depth buffer, thus requiring 180KB of memory per map. By looking up this height value at its new position, the virtual human can use this to adjust its height above the ground plane. It should be noted that the ground is at the same height throughout the city and therefore height-field maps are not necessary. However, these maps have been successfully implemented with another virtual environment where the ground consisted of several planes at different heights connected by ramps (Figure 4).

5. Steering Behaviour: Group Formation

In the real world, while a large number of humans walk by themselves as they carry on with their everyday lives, it is also common for humans to form and walk in groups while conversing. To enhance the realism of a crowd populating a virtual city, virtual humans need to form groups and interact with each other. In our system, we allow virtual humans to form groups of two (Figure 5) and we base our approach on the research by Reynolds [Rey99] on three steering behaviours related to groups of characters: cohesion, separation and alignment. We use these behaviours to determine how a couple should react with each other when moving through the environment, while still avoiding static obstacles.

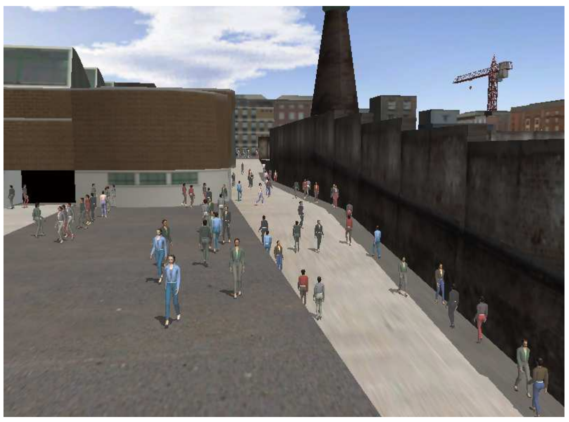


Figure 4: *Virtual human using a height map so that they do not intersect with the ground.*

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

References

[BMH98] Brogan D., Metoyer R., Hodgins J.: Dynamically simulated characters in virtual environments.

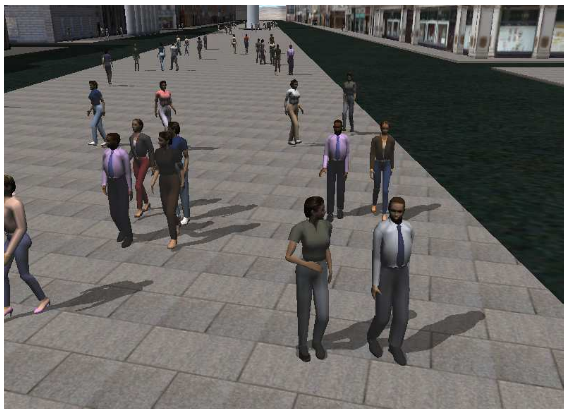


Figure 5: *Virtual humans walking in groups using cohesion, separation and alignment steering behaviours.*

IEEE Computer Graphics and Applications 18, 5 (1998), 59–69.

[BY95] BEARDON C., YE V.: Using behavioral rules in animation. *Computer Graphics: Development in Virtual Environments* (1995), 217–234.

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

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

[GLM98] GOLDSTEIN S., LARGE E., METAXAS D.: Dynamical autonomous agents: Game applications. *Proceedings of Computer Animation '98* (1998), 25–33.

[GLM99] GOLDSTEIN S., LARGE E., METAXAS D.: Non-linear dynamical system approach to behaviour modeling. *The Visual Computer 15* (1999), 349–369.

[HP88] HAMANN D., PARENT R.: The behavioral testbed: Obtaining complex behaviours from simple rules. *The Visual Computer 4*, 6 (1988), 332–347.

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

[HS02] HJELSTROM G., SMITH P.: Polygon soup for the programmer's soul: 3d pathfinding. *Proceedings of the Game Developers Conference* (2002).

[LMM03] LOSCOS C., MARCHAL D., MEYER A.: Intuitive crowd behaviour in dense urban environments using local laws. *Theory and Practice of Computer Graphics* (2003), 122.

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

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

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

[Rey99] REYNOLDS C.: Steering behaviors for autonomous characters. *Proceedings of the Game Developers Conference* (1999), 763–782.

[Sto96] STOUT B. W.: Smart moves: Intelligent pathfinding. *Game Developer Magazine* (October 1996), 28–35.

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

[TLC01] TECCHIA F., LOSCOS C., CHRYSANTHOU Y.: Agent Behaviour Simulator (ABS):a platform for urban behaviour development. *GTEC '01: The First International Game Technology Conference and Idea Expo* (2001).

[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.

[Vin97] VINCKE S.: Real-time pathfinding for multiple objects. *Game Developer Magazine* (June 1997).