




Fall 12-21-2011

Perceptually Driven Simulation

Ben Sunshine-Hill

University of Pennsylvania, bsunshin@seas.upenn.edu

Follow this and additional works at: <http://repository.upenn.edu/edissertations>

 Part of the [Artificial Intelligence and Robotics Commons](#), and the [Graphics and Human Computer Interfaces Commons](#)

Recommended Citation

Sunshine-Hill, Ben, "Perceptually Driven Simulation" (2011). *Publicly Accessible Penn Dissertations*. 435.
<http://repository.upenn.edu/edissertations/435>

This paper is posted at ScholarlyCommons. <http://repository.upenn.edu/edissertations/435>
For more information, please contact libraryrepository@pobox.upenn.edu.

Perceptually Driven Simulation

Abstract

This dissertation describes, implements and analyzes a comprehensive system for perceptually-driven virtual reality simulation, based on algorithms which dynamically adjust level of detail (LOD) for entity simulation in order to maximize simulation realism as perceived by the viewer. First we review related work in simulation LOD, and describe the weaknesses of the analogy that has traditionally been drawn between simulation LOD and graphical LOD. We describe the process of "perceptual criticality modeling" for quantitatively estimating the relative importance of different entities in maintaining perceived realism and predicting the consequences of LOD transitions on perceived realism. We present heuristic cognitive models of human perception, memory, and attention to perform this modeling. We then propose the "LOD Trader", a framework for perceptually driven LOD selection and an online approximation algorithm for efficiently identifying useful LOD transitions. We then describe "alibi generation", a method of retroactively elaborating a human agent's behavior to maintain its realism under prolonged scrutiny from the viewer, and discuss its integration into a heterogeneous perceptually driven simulation. We then present the "Marketplace" simulation system and describe how perceptually driven simulation techniques were used to maximize perceived realism, and evaluate their success in doing so. Finally, we summarize the dissertation work performed and its expected contributions to real-time modeling and simulation environments.

Degree Type

Dissertation

Degree Name

Doctor of Philosophy (PhD)

Graduate Group

Computer and Information Science

First Advisor

Norman I. Badler

Subject Categories

Artificial Intelligence and Robotics | Graphics and Human Computer Interfaces

PERCEPTUALLY DRIVEN SIMULATION

Ben Sunshine-Hill

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania in Partial
Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2011

Supervisor of Dissertation

Graduate Group Chair

Norman I. Badler, Professor,
Computer and Information Science

Jianbo Shi, Associate Professor,
Computer and Information Science

Dissertation Committee

Jan Allbeck, Adjunct Assistant Professor,
Computer and Information Science

Stephen H. Lane,
Associate Professor of Practice,
Computer and Information Science

Dave Mark, Founder, Intrinsic Algorithm
(External Committee Member)

Lyle Ungar, Associate Professor,
Computer and Information Science

Acknowledgements

I am first and foremost grateful to my wife, Jessamyn, for her love and support. I could not ask for a better friend, partner, confidante, listener, critic of ideas, recommender of books, guide to physics and mathematics, or maker of pies. I am likewise grateful to the rest of my family, and in particular to my parents (all of them), who kindled both the critic and the dreamer in me.

I have been privileged to belong to the most interesting and lively lab in the department, and am thankful to the colleagues who made it so: Jan Allbeck, Cory Boatright, Chris Czyzewicz, John Drake, Pengfei Huang, Joe Kider, Aline Normoyle, Alex Shoulson, Damian Slonegger, and Catherine Stocker. I couldn't ask for a better group to bounce ideas and paper drafts off of.

No man is an island, least so one who works in a graphics lab and can't draw. I am indebted to Grace Fong, Samantha Raja, Kaitlyn Reese, and Alice Yang for creating the art assets used in my research. Without them, it would all be white boxes. I am also grateful to the undergraduate and masters student programmers with whom I've worked most closely over the years: John Drake (again), Jon McCaffrey, Rob Mead, and Ian Perera.

I am indebted to my many, many, many teachers over the years. My advisor at UPenn, Dr. Norm Badler, has been an incredible source of wisdom and counsel, and his tireless work on behalf of me and the rest of his students has been a priceless gift. And I am likewise indebted to my masters advisor at UCLA, Dr. Petros Faloutsos, for teaching me the fundamentals

of academic research, and instilling in me the valuable lesson that when something seems impossible, the solution is usually to learn more math. And to Austin Ellis at EA, who taught me so much about the practice of development in just a few months, and to Kevin Dill and Dave Mark, who also guided my entry into the world of getting real stuff done. And to so many others, but in particular Alan Kay, Steve Lane, Michel Melkanoff, Benjamin Pierce, Alla Safonova, Stefano Soatto, and Lyle Ungar during my graduate education, and very much in particular Mrs. Henderson, Mrs. Paolini, and Mrs. Swovelin, the high school English teachers who gave me the skills that, by far, I use most.

It is staggering, and humbling, to look back upon how much these people have invested in me, and how much of what I am – to my very core – is a result of that investment. And perhaps ‘investment’ is the wrong word for it – no sane investor would stake so much on so unlikely a prospect. Perhaps ‘wild gamble’. Perhaps ‘love’. May I prove worthy of it.

ABSTRACT

PERCEPTUALLY DRIVEN SIMULATION

Ben Sunshine-Hill

Supervisor: Norman I. Badler

This dissertation describes, implements and analyzes a comprehensive system for perceptually-driven virtual reality simulation, based on algorithms which dynamically adjust level of detail (LOD) for entity simulation in order to maximize simulation realism as perceived by the viewer. First we review related work in simulation LOD, and describe the weaknesses of the analogy that has traditionally been drawn between simulation LOD and graphical LOD. We describe the process of “perceptual criticality modeling” for quantitatively estimating the relative importance of different entities in maintaining perceived realism and predicting the consequences of LOD transitions on perceived realism. We present heuristic cognitive models of human perception, memory, and attention to perform this modeling. We then propose the “LOD Trader”, a framework for perceptually driven LOD selection and an online approximation algorithm for efficiently identifying useful LOD transitions. We then describe “alibi generation”, a method of retroactively elaborating a human agent’s behavior to maintain its realism under prolonged scrutiny from the viewer, and discuss its integration into a heterogeneous perceptually driven simulation. We then present the “Marketplace” simulation system and describe how perceptually driven simulation techniques were used to maximize perceived realism, and evaluate their success in doing so. Finally, we summarize the dissertation work performed and its expected contributions to real-time modeling and simulation environments.

Contents

Acknowledgements	ii
Abstract	iv
1 Introduction	1
1.1 Of simulation and perception	1
1.2 Perceptually motivated simulation	2
1.3 The limits of realism discrimination	3
1.4 Rationing computational resources	4
1.5 Perceptually driven simulation	5
1.5.1 Perceptual criticality modeling	6
1.5.2 The LOD Trader	6
1.5.3 Alibi generation	6
1.6 Organization	6
2 Related work	8
2.1 Level of detail	8
2.2 Simulation level of detail	9
2.2.1 Perceptual validation	10
2.3 Behavioral level of detail	11
2.3.1 Non-comprehensive simulations	14

2.3.2	The LOD membrane	15
2.4	Presence	16
2.5	Breaks in Presence	17
3	Perceptual criticality modeling	19
3.1	The arithmetic of unrealism	20
3.1.1	Independence	20
3.1.2	Log-scaling	21
3.2	Categories of unrealism	22
3.2.1	Unrealistic state	22
3.2.2	Fundamental discontinuity	23
3.2.3	Unrealistic long-term behavior	23
3.3	Unrealism and linear functionals	24
3.4	Factor modeling	26
3.4.1	Observability	27
3.4.2	Attention	27
3.4.3	Memory	29
3.4.4	Duration	31
3.4.5	Return time	32
3.5	The criticality multiplier	34
4	The LOD Trader	35
4.1	Proactive and reactive LOD	35
4.2	The difficulties of reactive simulation LOD	37
4.2.1	Heterogeneous resources	37
4.2.2	Short- and long-term LOD selection	39
4.2.3	Levels and transitions	40

4.3	Optimal resource allocation as a knapsack problem	41
4.4	The trader analogy	44
4.4.1	Benefits and penalties	44
4.5	The environment of the LOD trader	45
4.5.1	Specification of resources	45
4.5.2	Specification of capabilities	45
4.5.3	Enumerating feature solutions	48
4.5.4	Costs, penalties, and transitions	49
4.5.5	The cost multiplier	51
4.5.6	Upgrades and downgrades	51
4.5.7	Evaluating usefulness	52
4.6	The design of the LOD trader	54
4.6.1	Efficiently eliminating un-promising transitions	55
4.6.2	Entity selection	57
4.6.3	Downgrades	59
4.6.4	Initial LOD assignments	60
5	Alibi generation	61
5.1	Introduction	61
5.2	Motivating example	64
5.2.1	World schema	64
5.2.2	Agent behavior	65
5.3	Components of perceptual simulation	66
5.3.1	Creating and deleting	66
5.3.2	Alibi generation	68
5.4	Deriving the probabilities	68

5.5	Alibi Sampling	72
5.6	Results	73
5.7	Drawbacks	75
5.8	Integration with the LOD trader	76
5.8.1	Cells as entities	76
5.8.2	Cells as triggers	77
5.8.3	Discussion	78
5.9	Beyond pedestrians	79
6	The Marketplace Project	82
6.1	Overview	82
6.2	The simulation	83
6.2.1	Architecture	83
6.2.2	Graphics	83
6.2.3	Behavior	83
6.2.4	Navigation	87
6.2.5	Animation	87
6.3	LOD control	87
6.3.1	EXISTENCE	89
6.3.2	BEHAVIOR	89
6.3.3	GOAL PICKING	91
6.3.4	INCIDENTAL ACTIONS	91
6.3.5	NAVIGATION	92
6.3.6	LOCOMOTION	93
6.3.7	ACTION ANIMATIONS	94
6.3.8	MESH RETENTION	95

6.4	Tuning	95
6.4.1	Measuring costs	95
6.4.2	Tuning the criticality model	96
6.4.3	Tuning audacity	98
6.5	Alternative implementation	99
6.6	Evaluation	99
6.6.1	Performance testing	99
6.6.2	Perceptual study	101
6.6.3	Subjective evaluation	103
6.6.4	Personal impressions	105
6.7	Discussion	105
6.7.1	Cascaded tuning	106
6.7.2	Transition costs and penalties	106
6.7.3	Costs and visibility	107
6.7.4	Feature graph design	107
6.7.5	Types of attention	108
6.7.6	RAM and memory	108
7	Conclusions	110
7.1	Future directions	112
7.1.1	Black-box alibi generation	112
7.1.2	Multi-horizon LOD planning	112
7.1.3	Better criticality modeling	113
7.1.4	Practical use	113
	Bibliography	114

List of Tables

3.1	The different categories of unrealism, and the factors which affect the likelihood that each will cause a BIR.	25
6.1	Perceptual study responses, on a five-point Likert scale, for the incidence and frequency of BIRs from low LODs at different population rates and with either the LOD trader (“tra”) or proactive selection (“pro”) controlling LODs. Lower numbers are better. Error bars represent 95% confidence intervals.	104

List of Figures

4.1	An example feature graph, with eight features. This feature graph has 252 feature solutions.	47
5.1	Kullback-Liebler divergence of alibi sampling versus ground-truth probabilities, under varying Metropolis-Hastings transition table size and number of iterations.	74
6.1	A birds-eye view of the Marketplace simulation environment.	84
6.2	Affordance sites, and the volume within which they are offered to agents, in the Marketplace simulation environment.	86
6.3	Cells used for visibility testing and agent creation in the Marketplace simulation.	88
6.4	Per-frame execution time during a 60-second flythrough of the Marketplace simulation, as a proportion of total frames rendered and as a proportion of total simulation time.	100
6.5	A frame from one of the videos shown to participants in the perceptual study of the Marketplace simulation.	102

List of Algorithms

4.5.1 FINDFEATURESOLUTIONS	48
4.6.1 RUNLODTRADER	54
4.6.2 MAKEEXPANSIONQUEUE	56
4.6.3 EXPANDEntity	57
4.6.4 FILLTRANSITIONHEAP	58
4.6.5 EXPANDREQUIREDENTITIES	58
4.6.6 SELECTTRANSITIONS	59

Chapter 1

Introduction

1.1 Of simulation and perception

Computer simulation has long proceeded along a viewer-agnostic trajectory.

That is, the role of a computer simulation has traditionally been to compute the complete state of a simulated system at a particular time, given information about the complete state of the system at some time in the past and rules for how the system must change over time. From there, the state of system is *visualized* as needed: The information is processed into graphics which can be perceived visually by a human participant. In scientific computing, these visualizations are generally a compromise between the quantity and precision of the information which can be presented, and the clarity and intuitiveness of the presentation. The purpose is to enable the viewer to quickly discard irrelevant information about the simulation state, while identifying and quantifying those bits of data which are of particular importance.

The standard approach has been to make simulation independent of visualization: The simulation produces, and the visualization consumes. Certainly this is true to reality: The world works in the same way whether it is being perceived or not, and is unaffected by what

particular bits of it are important to the viewer.¹ A user who desired realistic results from a simulation would be justifiably suspicious if it turned out those results changed whenever her back was turned.

1.2 Perceptually motivated simulation

However, for a certain class of simulations (of which video games are the largest and most visible example), “realistic” is a very different sort of thing. First of all, the “realism” in question may relate to how “realistically” the magic fireballs cast by an elf wizard affected the zombies surrounding him – a topic on which the medical literature is curiously silent. More to the point, however, the realism sought here is *perceptual*. The simulation must be realistic not because some rocket engine being designed won’t work right if the answers were wrong, but because a palpably unrealistic depiction of reality risks destroying the user’s suspension of disbelief. Visualization is no separate concern here. The rules about what the viewer sees when the simulated world is in a particular state are every bit as important as the rules about how that world works. It can be safely assumed that objective realism is a sufficient condition for perceptual realism.² But is it a necessary condition? Clearly not! As a simple example, it is a standard practice to break a video game world up into *levels*, with the user present in, and able to perceive, exactly one level at a time. The game doesn’t bother to simulate the other levels. They are frozen in time until awakened by the player’s arrival; awaiting their cues, like actors in a play, because it doesn’t matter how you act if you aren’t on stage. For video games, this laziness is a matter of necessity. Taken together, the demand for realism, and for large, open worlds, far outstrips the available computational

¹At quantum scales, observing a system may indeed change its state, although what constitutes an “observation” here is murky. Although quantum mechanics in some respects inform this work, the simulations considered here, as a rule, do not rely on the faithful simulation of quantum effects.

²That might be an oversimplification – stage magicians make a living by conjuring apparently unrealistic results, while working in the real world – but it’s a reasonable one for most situations.

power. Video games simply could not provide the necessary real-time experience, rendered at dozens of frames per second, without this sort of culling.

1.3 The limits of realism discrimination

This compromise of realism in the service of performance may not even be a “compromise” at all. Humans are not perfect, objective judges of realism. Recent behavioral studies have demonstrated that people can be surprisingly insensitive to major, unrealistic changes in their environment [Simons and Levin, 1998]. With that in mind, it is worth exploring two distinct forms of perceptual realism.

On the one hand is *ideal perceptual realism*, the quality of being indistinguishable from real life to a viewer with perfect perception of detail, memory of past events, and statistical analysis of the patterns of those events. This is distinguished from objective realism in that the data that can be gathered by the world is limited by conventional human rules of perception – at any given moment, the viewer is located at a particular point, looking in a particular direction, and cannot gather direct data about anything not visible from that perspective. This is an easy quality to pin down: The output of a perceptual simulation possesses ideal perceptual realism if and only if there exists an objectively realistic simulation which would produce identical output. For instance, consider a simulation of a parking garage which showed only the entrance and exiting of cars. As long as the series of entrances and exits were feasible (i.e. the garage was never over- or under-loaded with cars, the only cars exiting were those which had previously entered, etc.) the simulation would possess ideal perceptual realism, regardless of whether there were actually an assignment of cars to parking spaces being simulated.

On the other hand is *practical perceptual realism*, the quality of being indistinguishable from real life to a human viewer with human powers of cognition. For instance, suppose the

parking garage simulation no longer bothered to simulate the current inventory of parked cars, instead only generating random car exits, at plausible intervals. The simulation would no longer possess ideal perceptual realism, as before long, a car would exit which had never entered. It being beyond the capacity of mere mortals to keep track of the full inventory of cars based on entrances and exits, however, the simulation could be reasonably considered to possess practical perceptual realism.

Practical perceptual realism is a much more difficult quality to pin down comprehensively, because there's no recourse to objective equivalence. Our only models of human cognition are empirical. That is, the only way to verify that a simulation possesses practical perceptual realism is to run human experiments. And a lot of human experiments, at that – a simulation may be noticed as unrealistic only in a small percentage of runs (when the Wienermobile happened to exit twice in a row³), or only by a small percentage of users (ones with unusually apt memory), or only as a result of the user viewing the simulation from an unusual and unexpected perspective (if the simulation was only intended to be viewed for a few minutes, but the user decided to view it for three days and thereby was able to clearly discern that the number of exits far exceeded the number of entrances). For a wide range of simulations which do not possess ideal perceptual realism, it would not be possible to definitively prove that they possessed practical perceptual realism.

1.4 Rationing computational resources

The real utility of defining practical perceptual realism is not to create an absolute test, but to suggest a relative ordering. In any real-time simulation, computational resources must be rationed – if it is possible to build a computer with more computational power than anyone could possibly want, it has certainly not yet been done – and there are good and bad ways of

³Seven Wienermobiles are currently in operation, but as each is assigned to a separate region of the United States, more than one would not normally be observed in a single location.

rationing computational resources. For example, consider a crowd simulation which has run out of available memory, and must discard some people from the crowd in order to continue executing. Should the victims be chosen randomly? To do so might introduce a serious loss of realism, if one of the victims was nearby and fully visible to the viewer – people do not generally disappear like this. Better to choose a faraway agent, hidden behind something or out of the viewer’s frustum. The comparisons won’t always be apples-to-apples, though: we may instead choose to reclaim the memory from some other area of the simulation, say, by discarding the set of cars in a nearby parking garage. Better to lose this agent, or that car? In other words, what is the relative *perceptual criticality* of different entities? Again, defining “perceptual realism” does not by itself answer these questions, but it does provide an ideal framework within which to tackle them.

1.5 Perceptually driven simulation

This dissertation presents a framework for “perceptually driven simulation”. A perceptually driven simulation is a realtime simulation which rations computational resources in order to maximize perceptual realism.

A perceptually driven simulation must have the ability to perform the following tasks:

1. Simulate an individual entity through multiple strategies, and during the simulation, change which strategy is used to simulate an entity;
2. Estimate the importance of the individual entities for the viewer, and thereby the probable effect of changing their simulation strategies on the realism of the simulation as a whole; and
3. Coordinate the simulation strategies of all entities in order to maximize realism while respecting resource constraints.

The framework has the following components, which will be explored in subsequent chapters.

1.5.1 Perceptual criticality modeling

We have developed a system for estimating the expected impact of certain simulation decisions on perceptual realism, based on simple heuristic models of human perception.

1.5.2 The LOD Trader

The LOD Trader is a system for dynamically changing the way in which different entities in the simulation are simulated, in order to maintain an allocation of computational resources which provides near-optimal realism.

1.5.3 Alibi generation

Alibi generation is a technique we have developed for transparently upgrading the realism of agents in order to increase perceptual realism with a low computational cost. It provides two simulation strategies for virtual human agents, and the means to transition between them. We have integrated this technique into our perceptually driven simulation framework.

To evaluate this framework, we have developed the “Marketplace” simulation, which uses these components to maximize the perceived realism of a virtual-reality simulation of a crowded marketplace.

1.6 Organization

In the remainder of this dissertation, we will first review past work directly concerned with perceptually driven simulation, and in particular simulation level of detail; other

related work will be discussed in context. We will then describe the three components of the perceptually driven simulation framework listed above. We will then describe the “Marketplace” simulation, the process of integrating perceptually driven simulation into it, and tests performed to determine how effective these techniques proved to be. Finally, we will summarize the work performed, and its potential contributions to real-time modeling and simulation environments.

Chapter 2

Above all Siddhartha learned how to listen with a quiet heart, with a waiting, open soul, without passion, without desire, without judgment, without rebuke.

Herman Hesse, *Siddhartha*

Related work

This chapter reviews existing work in perceptual simulation and simulation level of detail (simulation LOD or SLOD).

2.1 Level of detail

It is difficult to pinpoint where work on simulation LOD began. From the earliest days of computer graphics, the limited computational power of rendering systems¹ required aggressive optimization techniques to perform interactive-rate rendering. Clark [1976] first introduced the concept of rendering different objects in a scene at different levels of detail, depending on their apparent size. This concept was embraced by many commercial rendering toolkits, and quickly became a standard element of interactive virtual reality rendering [Luebke, 2003]. Granieri et al. [1995], among others, extended this concept from still rendering to motion, using different articulation models to maximize the speed of animating faraway characters without compromising the realism of nearby characters. Because Granieri et al.’s characters used the output of their articulations to drive the motion

¹At least, by modern standards. In thirty years, researchers to come will no doubt pity us our own limited computational power.

of their root positions, this could technically be seen as simulation LOD, with more detailed articulations producing more nuanced paths. However, as all levels of articulation involved playback of prerecorded motion clips, this was still simulation LOD only in the loosest sense.

2.2 Simulation level of detail

Carlson and Hodgins [1997] first introduced real, rendering-decoupled simulation level of detail to computer graphics. Their simulation consisted of a crowd of one-legged creatures attempting to escape a “giant puck”. Each creature could be simulated dynamically (as a driven articulated body with four torque-controllable degrees of freedom) with collisions against rigid bodies in the world; via a hybrid model which displayed animations generated by interpolating prerecorded joint motions but directly accelerating a point-mass representation of the creature, or by a simple, unarticulated model which likewise accelerated the model as a point-mass but which did not synthesize joint motions and was therefore not appropriate for simulating on-screen creatures. Carlson and Hodgins defined conditions under which the LOD of a creature changed, based on both the immediate importance of the creature to the rendered scene and the capability of the different simulation levels to realistically react to the creature’s current state. For instance, because the point-mass models were not capable of reacting to inter-object collisions, fully dynamic simulation was used for creatures near a potentially colliding object. Other than that constraint, visibility and distance were used to select LOD: Off-screen creatures were simulated as pure point-masses, faraway visible creatures used the hybrid model, and nearby creatures used the fully dynamic model. The camera distance used to choose between the dynamic and hybrid models was chosen experimentally, but was not reactive to the simulation; the authors observed that different camera distance thresholds greatly affected the overall performance of the system,

but did not discuss the automatic and dynamic selection of an optimal threshold. Their results showed a great difference between the average and worst-case observed performance with far thresholds, although they did not comment on this result. This difference implies that with a far threshold distance, performance was determined largely by the population of the area near the camera. There was, therefore, a significant potential for experience improvements from dynamic threshold selection.

Much later work in real-time physical simulation likewise focused on modifying the type or parameters of an object's physical model by tracking the importance of the object and/or the capability of different models to realistically react to current state. For instance, Redon et al. [2005] adapted the detail in articulated dynamic characters in response to their distance from the camera (although they mention this criterion for level selection only in passing, concentrating instead on the mathematics of the joint simplification). Likewise, Debunne et al. [2001] performed real-time deformation of volumetric bodies by dynamically increasing the finite element resolution of areas of bodies undergoing more deformation and reducing the resolution of areas of bodies at rest. Although camera distance was not a factor in resolution adaptation, the increase in resolution in highly deformed areas led to realistic haptic feedback, arguably the most appropriate measure of perceptual quality in their application. Brogan and Hodgins [2002] built on the ideas introduced in [Carlson and Hodgins, 1997] with a bicycle race simulation which more accurately predicts the ability of a reduced-DOF controller to accurately model the outcome of the fully dynamic controller.

2.2.1 Perceptual validation

Dingliana and O'Sullivan [2000] developed an anytime collision detection algorithm which could produce approximate results in a guaranteed time limit, and a collision reaction system which could act on this approximate data. In their later work (O'Sullivan and Dingliana, 2001, O'Sullivan et al., 2003), they performed a series of psychophysical experiments to

determine the perceptual consequences of different levels of collision approximation, thereby qualifying the types of inaccuracies which were most likely to be perceived by participants; this data could be used to tune the degree of approximation during simulation. O’Sullivan and Dingliana were nearly unique in actually performing controlled testing on the perceptual effects of their approximations; most authors stopped short of perceptual validation, concentrating instead on showing real-time performance at “good enough” accuracy levels.

2.3 Behavioral level of detail

Other work focused on behavioral rather than physical simulation. O’Sullivan et al. [2002] developed the ALOHA system for real-time simulation of groups of human agents through LOD adaptation. The ALOHA system encompasses rendering, physical, and behavioral LOD. For instance, there are three levels of detail for conversations between agents: One in which prerecorded sequences of dialogue and associated nonverbal behavior (e.g. gestures) are performed, one in which nonverbal behavior is randomly generated while respecting basic conversational rules (for situations in which agents can be seen but not heard), and one in which nonverbal behavior is randomly generated and does not correspond to actual communication. In [MacNamee et al., 2002], the technique of “role-passing” was integrated into the ALOHA system. In role passing, a character’s (initially simplistic) behavior is externally augmented in certain situations in order for them to realistically accomplish the activities expected of them in certain roles. For instance, a character behind a bar could assume the “bartender” role, with the behaviors specific to that role, and later doff the role and be able to walk home without accidentally mixing anyone a drink. While not precisely LOD in the classical sense, the authors made the case that externalizing roles in this manner reduces the computational expense of characters on average by freeing them from even considering the set of behaviors specific to any given role.

Musse et al. [1999] enumerated discrete “levels of autonomy” to describe the ability of a virtual human agent to successfully complete a goal with varying levels of input: “guided”, indicating that the agent must be given explicit and exhaustive action instructions specific to the situation; “programmed”, indicating that the agent is given an explicit and exhaustive description of the goal but is capable of determining a set of actions to accomplish that goal; and “autonomous”, indicating that the agent need only perceive the environment and will determine goals and actions for itself. Although the authors did not approach these levels as potentially on-the-fly substitutable, as conventional “levels of detail” would be, they did discuss the types of situations in which different levels of autonomy would be necessary or sufficient, and provided a formalism for externalizing intelligence from the agent to its environment, a common component of high-performance human simulation [e.g. Brom et al., 2007, Stocker et al., 2010, Kistler et al., 2010].

Niederberger and Gross [2005], drawing on the levels of autonomy described by Musse et al., described a system for level of detail in human behavior. Each agent is assigned an LOD depending on their distance and visibility, and this level determines how often the agent’s simulator is invoked, how much time it has to complete, and whether it may plan autonomously or must be managed by a higher level planner. From there, the system determines what type of behavior is used. High-level agents use proactive planning and are allowed to spend a long time searching for optimal plans, while low-level agents only behave reactively, with their higher-level functioning driven by a grouped planner. This system actively manages the total time spent on agent simulation, apportioning the total time-slice based on LOD rather than independently assigning time-slices with no regard for their sum. Although the selection of LOD is similar to that of other approaches, considering only distance and visibility, the system does use a “nearly visible” zone which allows for higher levels of detail for agents just outside the viewing frustum. The detection of nearly visible objects arises from their broadphase visibility determination algorithm, and thus has

no independent cost. The authors motivated higher levels of detail for nearly visible objects “just to make sure that they really behave correctly.”

Kistler et al. [2010] and Wißner et al. [2010] described a city simulation and a beer garden simulation, apparently branches of the same underlying project, which uses multiple dimensions of behavioral LOD. At lower levels of detail², the optimality of planned paths is reduced, inter-agent avoidance is removed, update rates are reduced, and “some behavior is dropped”. (There is additionally LOD applied to rendering, animation, and sound generation, but these do not affect the ongoing state of the simulation.) Similar to [Chenney, 2001], large and infrequent simulation steps are taken for out-of-view agents. A set of integrated levels of detail is defined in the application, with each integrated level specifying the LOD along all dimensions. The choice of integrated LOD is entirely determined by camera distance and visibility.

Brockington [2002] described the behavioral LOD used in the computer role-playing game *Neverwinter Nights*. In this multiplayer game (unlike earlier games by the same developers), players can wander around the world independently rather than remaining grouped tightly into the same area. Brockington developed five levels of behavioral LOD. While distance from the (closest) player character is used to choose between two levels, and a semantic view of game areas is used to choose within two others, levels of detail are also chosen based on the current activity of agents: Those fighting or otherwise interacting with a player character are given a higher LOD than those merely nearby a player character. At lower LODs, characters move coarsely from tile to tile or directly to their destination, do not execute random walks, and are allowed a lower percentage of the total computational budget (with their simulation steps not invoked on every frame).

²The authors actually used “higher LOD” to refer to *reduced* levels of detail, denoting full simulation as level 0 and giving simplifications progressively higher numbers. Brockington [2002] used a similar scheme. To avoid ambiguity, we defer to the majority and use “higher LOD” in all cases to indicate more detail.

Brom et al. [2007] developed a village simulation, noting in particular the simulation of miners drinking in a pub. Agents draw their behavior from an “and-or tree” [Thornton, 2007], with atomic actions as leaves, and composite actions as and-nodes. Agent behavior can be simplified by executing composite actions atomically. Additionally, a semantic hierarchy is defined over the simulation area, allowing areas of the world to be collapsed at low levels of detail; although agents can still perform actions, they do so in an abstract manner. However, the agents themselves are never removed from the simulation as a result of collapsing an area. Brom et al. made the unique observation that the viewer might not be the only entity around which LOD should be high, suggesting that “important places” and characters should create a halo of high LOD around them.

2.3.1 Non-comprehensive simulations

Some work focused on LOD in particular types of behavior, rather than human behavior in general. Osborne and Dickinson [2010] demonstrated the hierarchical grouping of flocking agents to minimize computational costs for faraway agents: Nearby agents are individually simulated, while faraway agents are simulated in progressively coarser group nodes. Although this technique drastically reduces the cost of simulation, the authors did not explore the effect of different grouping thresholds on perceptual or objective realism, and in fact did not so much as mention the consequences of grouping on realism. As such, their results are not terribly useful.

Jan and Traum [2005] described an algorithm for simulating conversation between background characters; this simple algorithm was intended for use as a low behavioral LOD, as an alternative to earlier conversational models developed by the authors, but only evaluated believability in a limited manner, and did not evaluate performance.

Narain et al. [2009] developed a crowd simulation which used an incompressible fluid approximation for densely populated areas to maximize performance and prevent unrealistic

deadlocks, but suggest that in sparser areas a technique such as reciprocal velocity obstacles [Van den Berg et al., 2008] could be used for more realistic individual agent paths, and show that the two techniques can interact realistically in boundary areas. The choice of levels here would be motivated by capabilities rather than perceptual importance. Going further in this direction, Singh et al. [2011] developed a modular architecture for crowd simulation which featured several steering algorithms, dynamically changing the steering algorithm on a per-agent basis to minimize computational cost without compromising the quality of the steering. Although distance from the viewer and other perceptual considerations were not considered for algorithm selection in these projects, the basic architectures would support that use.

2.3.2 The LOD membrane

The work of Brom et al. [2007] and that of Osborne and Dickinson [2010] share the concept of an “LOD membrane”, a formalism for guaranteeing completeness when simplifying a hierarchical simulation. An LOD membrane is a partitioning over the nodes of a tree into simulated and non-simulated objects, such that:

1. The root is simulated.
2. Either all of a node’s children are simulated, or none of its children are simulated.
3. If a node is not simulated, none of its descendants are simulated.

This partitioning implicitly defines a set of “membrane” nodes, which are simulated but whose children are not simulated. If these nodes are leaves, they simulate as normal for a fully-detailed simulation. If they are not leaves, they are responsible for mimicking the activity of their descendant nodes in an approximated fashion. Nodes which are simulated but which are not membrane nodes (that is, their children are also simulated) may execute

certain bookkeeping operations but their simulation is primarily handled by their descendants. This formalism guarantees that every leaf node has its activity handled by exactly one membrane node, and membrane nodes can assume that none of their descendants are acting autonomously. In [Osborne and Dickinson, 2010] the membrane nodes are the flocking groups which act as a single agent; in [Brom et al., 2007] the membrane nodes are the lowest non-collapsed spatial areas, within which agents act.

2.4 Presence

While many researchers have studied the perception of realism in the context of rendering simplification [Luebke, 2003, chap. 9], few have studied it in the context of simulation (but see subsection 2.2.1). However, a related concept, that of *presence*, has received considerable attention. Lombard and Ditton [1997] describes presence as “an illusion that a mediated experience is not mediated.” From a behaviorist standpoint, presence is a human state in which the human’s reactions to mediated (that is, simulated under controlled circumstances) stimuli are the same as that human’s reactions to the same stimuli would be in an unmediated setting. The concept is applicable to virtual reality, telepresence, movies, books – any medium in which stimuli are intended to be evocative, rather than merely descriptive, of a specific unmediated experience.

Presence is a larger, more holistic concept than perceptual realism as described here. A person viewing a simulation on a 9-inch monochrome monitor would feel less presence than if she were to view it on a stereoscopic 60-inch full color display. However, her perception of the realism of *the simulation itself* would be the same, or might even be higher for the monochrome monitor (if the low fidelity covered flaws in the rendering). Perceptual realism as described here is more analytic: The viewer, provided with sensory input, determines a “state of the world” corresponding to that input, and judges whether that state of the world is

realistic, rather than judging whether the sensory input itself is realistic. Nevertheless, there is obvious overlap between the two concepts.

The association of presence with telepresence and media studies (both of which involve recorded and reproduced stimuli) has led some presence researchers to invent catalogues of the determinants of presence which are not readily applicable to simulation; the determinants focus on how accurately and vividly stimuli are reproduced, and how plausible recorded actors' behaviors are, but do not consider the situation where the world is entirely computer-simulated. The models are not fundamentally incompatible, however. *Social realism*, the feeling that other people encountered in a virtual environment are behaving in a reasonable and consistent manner, is commonly mentioned as a determinant of presence. If one extends the concept of social realism beyond social behavior to encompass the simulation of all entities, the result is very much like perceptual realism.

2.5 Breaks in Presence

Slater and Steed [2000] introduced the concept of a *Break in Presence* (BIP), a discrete event in which a viewer transitions from feeling present in a virtual environment, to feeling present in the real world.³ Many things can cause a BIP; the viewer noticing a particular unrealistic event in the simulation is a sufficient but not a necessary condition. Though discrete events in themselves, a particular BIP may not have a discrete external cause: they can occur at unpredictable intervals, the products of unobservable mental processes, with their frequency being partially determined by ambient external factors.

The subset of BIPs which have proximal external causes – and, further, causes relating to the perception that the simulation is behaving in a realistic manner, are of importance to perceptually driven simulation. We term these *Breaks in Realism* (BIR), and discuss them

³The reverse event, in which a viewer becomes present in the virtual environment, is also considered a BIP.

further in chapter 3.

As mentioned earlier, vividness of presentation can have opposed effects on presence and perceived realism; and it seems clear that perceived realism has an effect on presence.⁴ It is unclear whether there is any causal force in the other direction – whether an increase in a viewer’s sense of presence, from whatever source, affects the likelihood of BIRs.

⁴This effect may not be consistent or monotonic. Vinayagamoorthy et al. [2004], studying the effect of texture repetition and character modeling fidelity, claim that high-fidelity character models lowered reported presence levels. Their confidence intervals, however, do not appear to support this conclusion. Garau et al. [2003] provided better support for this claim. Both studies, however, focused on the low end of the realism scale: a choice between the cartoonish and the freakish. Later work, such as that of Slater et al. [2009], supports a positive correlation between realism and presence at higher levels of rendering quality.

Chapter 3

The brain has its own language for testing the structure and consistency of the world. But we never see the machinery of logical analysis, only the conclusions.

Carl Sagan, *Cosmos*

Perceptual criticality modeling

In this chapter we discuss the design of heuristic models to measure the relative potential impact of per-object LOD decisions on the viewer’s overall perception of realism. The basic question to be answered is, given an entity in the world and a potential event affecting how the entity is simulated, “what is the probability that, if this event is made to take place, the viewer would perceive the entity as unrealistic”? The potential consequence of the viewer perceiving the entity – and, by extension, the simulation as a whole – to be unrealistic, is a *break in realism* (BIR).¹ An unrealistic event is one which has a nonzero probability of causing a BIR event; an event which has a greater probability of causing a BIR than another is more unrealistic than the other.

As we shall argue, estimating the impact of an entity’s LOD decision on the perceived realism of a simulation involves evaluating both the *a priori* unrealism of the LOD itself, and the degree to which the viewer’s relationship with that particular entity can exacerbate or mitigate that unrealism. We refer to this latter factor as *perceptual criticality*, and will describe the space in which it exists and how it may be estimated.

¹To be contrasted with a “break in presence” as defined by Slater and Steed [2000]. A BIR is a sufficient condition for a BIP, but not a necessary one.

3.1 The arithmetic of unrealism

3.1.1 Independence

We make the important simplifying assumption that BIRs caused by different entities are independent random variables. That is, it assumes that if, in the event, the viewer perceives entity A as unrealistic, the posterior likelihood of perceiving B as unrealistic does not change. This assumption may be criticized on several fronts.

Sensitization. If a viewer experiences a BIR, this may cause her to be more alert to other unrealistic entities, particularly similar ones. Noticing that a broken mailbox has been mysteriously repaired, for instance, may sensitize her to notice other entities which likewise are unexpectedly restored to their pristine state.

Location. Certain BIRs are dependent on the uncertain event of the viewer being able to perceive them in the future, so BIRs of entities near each other in the world become positively correlated. If and when a viewer experiences a BIR from a given entity, she must be located near that entity, and therefore BIRs from entities nearby it become more likely; likewise, BIRs from faraway entities become less likely.

Correlation. Two or more entities may be perceived as unrealistic due to the relationship between them. For instance, if two blue mailboxes next to each other on the street change to grey while the viewer is not watching them, the result may be less likely to be noticed than if only one of them changes to grey and the other remains blue.

Of these potential dependencies, the first may be disregarded as long as the viewer does not notice sources of unrealism to begin with. As the viewer noticing even a single unrealism should be regarded as a failure of the system, the subsequent greater or lesser likelihood of noticing other sources of unrealism is academic. The second dependency will be discussed

in section 3.2.2. It is unclear in what, if any, situations the third dependency would arise as a result of LOD simplification, and in our opinion is unlikely to be encountered in the design of a simulation LOD system.

3.1.2 Log-scaling

In our system, rather than working directly with BIR probabilities, we look at their negated log inverse probability – that is, $-\log(1 - p)$, the negated logarithm of the probability that a source of unrealism is *not* noticed. This quantity (which we will refer to as the “log-scale”) has the following properties:

1. 0 is the minimum possible value in the log-scale, equating to $p=0$ in the linear scale.
2. The sum of log-scale values for two unrealistic events is equal to the log-scale of the probability that *neither* event causes a BIR, assuming the independence criterion from section 3.1.1.
3. Multiplying a log-scale value by a nonnegative real number n produces the log-scale of the probability that none of n independent copies of the event cause a BIR.

The first two properties make it possible to determine the “total unrealism” of a simulation through a simple summation. The third allows us to assign mathematical meaning to a phrase such as “twice as unrealistic”; one event is n times as unrealistic as another if getting away with the first event is as likely as getting away with n copies of the second event. (n need not be an integer.) This interpretation is better behaved than operating directly on p : it means that an event which is “twice as unrealistic” as some event which would be noticed 50% of the time, need not be noticed *all* the time, and it does not restrict us from describing an event as, say, “three times as unrealistic” as some event which would be noticed 50% of the time.

3.2 Categories of unrealism

In order to determine the likelihood of a BIR, one must first decide what situations can give rise to a BIR. Under ideal circumstances, these situations are undesirable but not unexpected: they are the known consequences for LODs which have lower computational resource requirements relative to higher-fidelity LODs. We have identified the following categories of unrealism, which are not intended to be exhaustive but are intended to be nearly exhaustive of realism that might be created in the service of increased performance:

3.2.1 Unrealistic state

In this category, the immediately and directly observable state of an entity, or the directly observed propagation of that state over a short period of time, would be unlikely or impossible in real life. Two pedestrians occupying overlapping positions in space, or a vehicle traveling without its wheels turning, could cause a BIR of this sort. This category has direct analogues in graphical LOD: If an object is rendered at a detail level which is not perfectly indistinguishable from highest detail at its current draw distance, its state is unrealistic. Likewise, if a character is animated with a low-quality skeleton at a distance which allows the discernment of its joints, its short-term behavior is unrealistic.

The probability of a BIR from unrealistic state is affected by the audacity of the unrealism (the degree to which it is distinct from any likely or possible state), the observability of that portion of the state which exhibits the unrealism, and the attention which the viewer is paying to that exhibited state. In comparing BIR probabilities between heterogeneous entities, it is likely that the first factor can only be determined empirically.²

²Attentional factors also play a part in visual fidelity metrics, as an entity which the viewer is watching directly is observed with macular rather than peripheral vision and thereby observable in more detail. Organizationally, we delegate this overlap of concerns to attentional modeling, as without gaze tracking it is more difficult to extricate gaze from attention than from vision.

3.2.2 Fundamental discontinuity

In this category, an entity's current state is perceived as being incompatible with its observed state in the past. For example, a mailbox which the player had earlier crashed into being later observed as undamaged could cause a BIR of this sort. As another example, a pedestrian who, after temporarily being out of view, had traveled a distance which was patently incompatible with their walking speed. This category does not focus on situations where the discontinuity occurs while the entity is being continuously observed; those situations are adequately covered by the previous category. For this sort of BIR to occur, the viewer must draw on her recollection of past observations. The probability of a BIR from fundamental discontinuity is proportional to the audacity of the discontinuity, the degree of recollection of prior observations, and the expected attenuation of memory upon return to the entity (which is, in turn, dependent on the amount of time before that return).

Because this type of BIR is predicated on an uncertain future return to an entity, it is susceptible to compromising the assumption of independence described in section 3.1.1, by introducing positive covariance for memory attenuation for entities near each other, and negative covariance for entities far from each other. Our model for expected return time does not take physical location as input, and so sidesteps this issue. A more elaborate system could estimate fundamental discontinuity on a spatial basis instead of a per-entity basis, but this would not be appropriate for moving entities and, in our opinion, would not yield significant marginal benefits.

3.2.3 Unrealistic long-term behavior

In this category, an entity's behavior is perceived as realistic in the short term, but observing it for a sufficient quantity of time demonstrates unrealism. A pedestrian who is forever circling a single block can cause a BIR if the viewer observes it for long enough, but

cannot cause a BIR of this category based on only a short period of observation. Likewise, a car which never runs out of fuel becomes unrealistic only if the viewer follows it for several hours. This type of BIR has no analogue in graphical LOD, though weak parallels may be drawn with impostor-based rendering. This category is distinct from fundamental discontinuity in that it can (though does not necessarily) result from continuous observation of an entity.

The factors of the likelihood of a BIR from unrealistic long-term behavior are similar to those of fundamental discontinuity, but include the duration of continued observation of the entity.

The different categories of unrealism are summarized in table 3.1.

3.3 Unrealism and linear functionals

The various factors of BIR probability listed in table 3.1 may be broadly separated into two categories. The first, consisting of the “audacity” factor, contains properties of the event itself, rather than the viewer; the second, consisting of “observability”, “attention”, “memory”, and “duration”, are products of the viewer’s situation and his relationship to the entity associated with the event – the *criticality* of that entity. If the viewer’s observability of an entity increased, we would expect the BIR probability for all BIRs associated with that entity to increase in equal proportions. Conversely, if a viewer had equal observability and attention factors for two different entities, we would expect two unrealistic state events with the same audacity, one for each entity, to result in equal BIR probabilities.

This leads to an elegant mathematical model of BIR probability. If the audacity factors for the three categories of unrealism from a single event are gathered into a column vector, and the criticality factors for the categories of unrealism as induced by the current state of the viewer and his relation to the entity are gathered into a row vector, the product of the

Type of unrealism	Examples	Factors relevant to BIR likelihood				
		Audacity	Observability	Attention	Memory	Return time
Unrealistic state	Foot skate Interpenetration	Y	Y	Y		
Fundamental discontinuity	Disappearing damage out-of-view “teleportation”	Y			Y	Y
Unrealistic long-term behavior	Random wandering Inexhaustible fuel supply	Y		Y	Y	Y

Table 3.1: The different categories of unrealism, and the factors which affect the likelihood that each will cause a BIR.

two vectors is the total BIR probability of the event. In this model, the audacity vector of an event is fixed for all instantiations of that event and independent of the specific viewer, and a particular criticality induces a positive linear functional over the audacity vector space which is independent of the particular events taking place. We refer to this functional as a “criticality multiplier”.

For instance, consider a human entity who has just approached the viewer and asked her for directions, before being momentarily obscured by a passing door. Attention and return time factors are high, and memory and observability factors are low. This leads to a high probability of BIRs from unrealistic state, and a low probability of BIRs from fundamental discontinuity or unrealistic long-term behavior. An event which is audacious as to unrealistic state only, such as the entity’s animation being temporarily frozen, will lead to a low BIR probability overall, while an event which is audacious as to both unrealistic state and fundamental discontinuity, such as the entity’s face changing to someone else’s face, has a somewhat higher BIR probability (though still low, because of the low memory factor). If, on the other hand, the viewer has been conversing with the human entity for several minutes before the events occur, the frozen animation event will still lead to a low BIR probability but the face changing event will lead to a high BIR probability, without any need to change the audacity vectors for the two events.

3.4 Factor modeling

In this section, simple computational models for the various non-audacity factors involved in criticality modeling are derived. These models are not intended to be precise; are based on simple, informal reasoning about the underlying processes, with only occasional recourse to settled science; and have only informal empirical support. They are intended to be simple to understand, efficient to implement, and transparent enough in operation to be easily tuned.

More detailed models may be substituted, e.g. the inattentional blindness model of Gu et al. [2005] or the ACT-R declarative memory system [Douglass et al., 2009].

3.4.1 Observability

Observability is the property of being capable of being observed, as opposed to the property of actually being observed. As a result, it may be confidently estimated based purely on output from the renderer. An entity which is not on-screen, either because it is outside the viewing frustum or because it is hidden by an occluding object, has an observability of 0. An entity which is on-screen and subtends a large enough proportion of the screen to have its details clearly visible, has an observability of 1. Visible but faraway entities present the only real difficulty for quantifying observability. We take the simple expedient of measuring the solid angle subtended by unoccluded portions of an entity, a quantity which is roughly proportional to the number of pixels in which the entity can be found. Above a certain solid angle (the “saturation angle”), the entity is considered “visible enough” to be fully observable; beyond that distance the observability is proportional to the solid angle:

$$O_i = \begin{cases} \frac{\Omega_i}{\Omega_{sat}} & \text{if } \Omega_i < \Omega_{sat}, \\ 1 & \text{if } \Omega_i \geq \Omega_{sat}. \end{cases} \quad (3.1)$$

3.4.2 Attention

Modeling attention is an important aspect of criticality modeling for two reasons. First, scrutiny of an entity’s realism increases with attention. Simons and Chabris [1999] found that subjects who were shown a video of several people playing with balls, and a gorilla unexpectedly walking across the scene, were less likely to notice the gorilla if they were concentrating on a challenging cognitive task related to the people playing with balls. Second, rate of memorization increases with attention [Pessin, 1933], so a good model of

memorization should take expected attention into account to the extent that it depends on initial encoding strength.

Human attention is a limited resource. A viewer watching a crowd of one hundred people will be less likely to notice a single person's momentary unrealistic behavior than if she was watching a single person. It is tempting, therefore, to model attention as a constant which is divided among the entities in a simulation. At the same time, however, human attention is not a fixed quantity: A person performing an unchallenging cognitive task, such as watching a colored dot and waiting for it to change color, will not have their performance drop to half if a second dot is added. A simple model to account for these phenomena is the addition of a constant *ambient attentional load* before normalizing. This quantity represents that portion of the viewer's intellect which is not devoted to any entity in the simulation. For some per-entity indicator of attempted attention \hat{A}_i , the *effective attention* A_i given to an entity is then

$$A_i = \frac{\hat{A}_i}{L}, \text{ where total attentional load } L \text{ is} \quad (3.2)$$

$$L = \hat{A}_{amb} + \sum_{j=1}^n \hat{A}_j. \quad (3.3)$$

This behaves reasonably: For a small number of low-attentional-load entities, total attentional load is largely taken up by ambient attentional load. As the number of entities increases, per-entity effective attention decreases, but the total proportion of attentional load taken up by entities in the simulation increases.

It remains to model the input to this equation. Our model of attempted attention primarily draws on an exponential moving average of observability as an input. Certain behaviors, however, may be taken as indicative of increased attention:

- *Focusing*, in which the viewer manipulates the camera angle in order to roughly center the entity on the screen, and

- *Following*, in which the viewer manipulates the viewing position in order to limit the distance to the entity.

Time-averaging is likewise used to typify these behaviors. The degree of focusing is determined as an exponential moving average of the cosine of the angle between the viewer's forward-vector and a ray connecting the viewer to the centroid of the entity, modulated by observability. Similarly, the degree of following is determined as an exponential moving average of the distance between the viewer and the entity (saturated to a particular maximum range), also modulated by observability. The total attempted attention for an entity is the sum of these three factors, linearly weighted by empirically determined constants and with decay factors for the exponential moving averages also empirically determined.

3.4.3 Memory

Although no precise, comprehensive description of the structure and operation of human memory has yet gained wide acceptance, many memory researchers have proposed mathematical models to predict and explain the performance of experimental participants in memory tests. Of the various memory tasks commonly studied, one of the most relevant for this area is that of *associative recognition*. Associative memory tasks call for participants to memorize associations between pairs of items in a study list. The most common associative task is known as the *cued recall* or *paired associate* task, in which participants are shown one item from a pair of items from the study list and asked to respond with the other item from the pair. In the associative recognition task, in contrast, participants are shown either a pair of items drawn directly from the study list (an *intact pair*, or *target*) or a pair of items created by combining items from two pairs from the study list, but which did not occur together in the study list (a *rearranged pair*, or *lure*), and asked to respond whether the pair was on the study list. (In the recognition literature, a correctly recognized target item is a

hit, and a lure item incorrectly recognized as a target item is a *false alarm*.)

Item recognition tasks are well-described by a *dual-process model*, which describes two strategies by which a participant can successfully recall a target item: *familiarity*, in which the participant feels that they “know” that the item was on the study list, and *recollection*, in which the participant “remembers” the experience of studying the item [Yonelinas, 1994]. Yonelinas [1997] found that although both processes are required to account for observed performance on item recognition tasks, associative recognition draws almost exclusively on recollection. Moreover, in the case of associative recognition tasks, recollection has a second effect which is not found in item recognition tasks: A participant may recognize a test pair as a lure by recalling the original intact pair which was modified to produce the lure.

A well-known effect in human memory performance is *retroactive interference* (RI), whereby a participant’s ability to remember studied material is reduced if they study other material in the interim between the original material and the test of that original material [Britt, 1935].³ RI is maximized if the interpolated material is similar to the original study material. Many studies have demonstrated RI in cued recall tasks, but have generally failed to demonstrate a significant RI effect in associative recognition. Verde [2004], however, has proposed that this is due to RI causing opposed effects for recognition, reducing recollection but increasing familiarity. This hypothesis predicts that RI increases false-alarm rates, which was borne out by studies performed by Verde.

In the case of fundamental discontinuities, associative recognition provides a reasonable model for the cognitive task undertaken by viewers. An association between an entity’s identity (“the mailbox at that location”) and its state (“dented”) is memorized. Later, the participant is shown either the same identity/state pair, or one with the same identity but

³*Proactive interference*, whereby a participant’s ability to remember studied material is adversely affected by memory tasks performed *prior* to that material, has also been studied but is of less interest here.

modified state (“not dented”), and asked to determine if the pair is intact. The goal of criticality modeling here is to determine the false-alarm rate – that is, to quantify the probability that the participant will “accept” the rearranged pair. RI will increase this probability, particularly RI from similar items.

We have developed a simple model for memory estimation, based on exponential decay of the estimated memory of the viewer for an entity towards the current estimated attention for that entity. Two decay rates are used: α for increasing estimated memory, and β for decreasing estimated memory. The latter rate is multiplied by the current total attentional load, representing retroactive interference.

$$\frac{dM_i}{dt} = \begin{cases} \alpha(A_i - M_i) & \text{if } A_i > M_i \\ \beta L(A_i - M_i) & \text{otherwise.} \end{cases} \quad (3.4)$$

α controls the rate of memorization; β controls the rate of forgetting. The parameters are chosen such that $\alpha \gg \beta L$ for reasonable values of L , such that estimated memory tends towards the maximum recent attentional estimate for the entity.

An important consequence of this model is that the fractional fall-off of memory of an entity over time, assuming zero ongoing attention for that entity and a constant total attentional load, is given by

$$\frac{M_i(t + \Delta t)}{M_i(t)} = e^{-\beta L \Delta t}. \quad (3.5)$$

3.4.4 Duration

The duration factor represents the quantity of separate observations which, taken together, reveal unrealistic behavior. It may be simply represented as the sum of attention, modulated by observability, over time:

$$\frac{dD_i}{dt} = A_i O_i \quad (3.6)$$

3.4.5 Return time

Fundamental discontinuity events are unique in that the unrealism they create is not experienced immediately, but is deferred until the viewer next perceives the entity. As such, in order to quantify the probability of a BIR from fundamental discontinuity, it is necessary to estimate the expected memory of the viewer for the entity not at the time of the event but at that later meeting. We assume that only the next time the viewer returns to the entity is significant for an LOD decision. Assuming no BIR occurs, the entity's new state will thereafter be the most recently observed state for that entity, "removing" the discontinuity by bringing the viewer's expected expectation of the entity's state back into agreement with the entity's actual state.⁴

Analyzing the expected amount of time ("lifetime") which passes between the beginning of a period and the occurrence of a particular event is known as *survival analysis*. (The occurrence of that event is known as a "failure".) A lifetime probability distribution induces a particular *hazard function*, the momentary event rate at a particular age, conditioned on survival up to that age.

The lifetime distribution can be used along with a function relating the lifetime to the subsequent expected "reward" (resultant BIR probability). The distance between the viewer and the entity places a lower bound on the time to return to a point from which the entity may be observed. For a lifetime distribution function $F(x)$ and an expected reward function $R(x)$, if it has been t_0 units since the viewer last encountered the entity, the expected reward is

$$E[R|t > t_0] = \int_{t_0}^{\infty} \frac{F'(t)R(t)dt}{1 - F(t_0)}.$$

Although many probability distributions can be used to describe the lifetime of a survival

⁴This assumption is supported by the Comprehensive Holographic Associative Recall Model (CHARM) proposed by Eich [1982], wherein earlier stored associations are subject to strong interference from later stored associations with similar or identical cues.

process, one of the most common is the Weibull distribution. With the shape value set as $0 < k < 1$, the failure rate (per-unit-time probability of return) modeled by the Weibull distribution decreases over time, a desirable property: A viewer who had been absent from a given entity for one second would reasonably have a greater probability of returning to that entity in the next second, than a viewer who had been absent from the entity for several hours would have of returning to the entity over the following second. Because the return process is objectively measurable, the lifetime distribution may be automatically learned from user performance data.

The “reward” (in this case, a penalty) is directly related to the likelihood of successful recall upon return. Inserting the falloff rate of memory (evaluated at time t_0), assuming constant total attentional load,⁵ this makes the expected reward given time t_0 since the entity was last visited

$$\begin{aligned} E[R|t > t_0] &= \int_{t_0}^{\infty} \frac{\frac{k}{\lambda} \left(\frac{t}{\lambda}\right)^{k-1} e^{-(t/\lambda)^k} M_i e^{-L(t-t_0)} dt}{e^{-(t/\lambda)^k}} \\ &= \int_{t_0}^{\infty} \frac{k}{\lambda} \left(\frac{t}{\lambda}\right)^{k-1} M_i e^{-L(t-t_0)} dt \\ &= k(\lambda L)^{-k} M_i e^{-L t_0} \Gamma(k, L t_0) \end{aligned}$$

which can be separated into the current memory factor M_i and the return time factor

$$R_i = k(\lambda L)^{-k} e^{L t_0} \Gamma(k, L t_0). \quad (3.7)$$

⁵A smoothed L may be used to avoid estimating large memory losses during momentary spikes in attentional load.

3.5 The criticality multiplier

The various factor models in section 3.4 may be combined as described in section 3.3 to produce the criticality multiplier for an entity,

$$B_i = [O_i A_i c_{\text{US}}, M_i R_i c_{\text{FD}}, A_i M_i D_i c_{\text{ULTB}}], \quad (3.8)$$

where c_{US} , c_{FD} , and c_{ULTB} are scaling factors chosen to give the three components similar magnitude. For a given event, the dot product of this functional and a column vector indicating the audacity of the unrealism of the event with respect to the three types of unrealism will produce the total unrealism of the event, as measured in log-probability space.

Chapter 4

Greed clarifies, cuts through, and captures the essence of the evolutionary spirit.

Stanley Weiser, *Wall Street*

The LOD Trader

In this chapter we present the “LOD Trader”, a simulation LOD system which uses a cost-benefit formalism to optimize the perceptual realism of a simulation. We motivate the need for the system by describing the practical differences between graphical and simulation LOD, and show how careless co-opting of common graphical LOD techniques for simulation LOD can lead to sub-optimal realism and poor performance.

4.1 Proactive and reactive LOD

A graphical level-of-detail rendering system can be thought of in terms of two concerns: The process by which an object or effect is rendered at a particular quality level, and how that level is chosen. There are many solutions for the first concern, from simply having different versions of a mesh or shader, to continuous, real-time geometry simplification, to impostor rendering. The second concern, in contrast, has a single solution form for most systems: A monotonic function mapping distance to quality level, such that objects and effects which are further from the viewer are rendered at lower qualities.

While this approach, properly tuned, guarantees a certain per-frame graphical quality

for a scene, it does not guarantee the framerate at which the scene can be rendered. That is, while an LOD system can greatly improve graphical performance, it is still left to the scene author to produce a scene which can be rendered at the desired framerate. If unforeseen circumstances occur which result in an unusually large number of objects needing to be rendered at relatively high quality, framerate can suffer quite badly. The loss in realism from the low framerate, in such a situation, can far outstrip the loss in realism that would occur from simply rendering all objects at a lower level of detail [Luebke, 2003]. The heuristic function used to determine level of detail, in other words, suffers from not taking into account the total rendering load and the resultant framerate; it is *proactive* in reducing detail in faraway objects (even when framerate is not a problem), but not *reactive* to heavy rendering loads.

Some reactive graphical LOD systems do exist [Luebke, 2003]. These are based either on adjusting level of detail in response to measured framerate, as in the Viper system [Holloway, 1991], or on heuristic prediction of rendering costs of different LOD decisions, as in the visualization system developed by Funkhouser and Séquin [1993]. This latter system is particularly interesting, as it uses a cost-benefit formalism to optimize the realism of a scene while staying under a computational budget.

In contrast, all simulation LOD systems described in the extant literature are proactive: They change levels in response to manually predefined rule-sets. For instance, Brockington [2002] switched between five distinct per-agent levels of detail based on predefined boundary distances from the viewer; Brom et al. [2007] defined a hierarchical system of LOD nodes, with predefined distance and visibility rules used to define a “membrane” over the LOD tree to determine what was simulated and what was not; and Osborne and Dickinson [2010] used the same “membrane” approach but applied it to groups of agents, but again used predefined boundary distances to maintain the membrane.

As with graphical LOD systems, proactive simulation LOD systems cannot provide

performance guarantees, and do not react gracefully or effectively to especially crowded situations. Unlike graphical LOD systems, however, proactive simulation LOD systems also suffer in a second way: Graphical LOD systems are often tuned so that LOD transitions happen at far enough distances that the quality degradation can be held to a minimal objective standard. But because degraded behavior, in contrast to degraded graphical quality, is difficult to quantify objectively, the fact that LOD reduction takes place even in sparse situations which do not require it may lead to an unnecessary loss of realism. That is, while a sufficiently distant object can be rendered at a low graphical LOD without any loss of quality at all, it is usually not possible to show that lowering the simulation LOD for an entity cannot lead to a loss of realism, no matter how distant.

Simulation level of detail, in short, is badly in need of reactive design. So why has this approach been ignored?

4.2 The difficulties of reactive simulation LOD

Although simulation LOD was inspired by, and many aspects of it are taken from, graphical LOD, the two have significant practical differences. In this section we describe three of the most pressing issues.

4.2.1 Heterogeneous resources

Real-time rendering is not memory-limited. That is, arbitrarily complex scenes can be rendered with a fixed amount of graphical memory resources. The representation of those scenes which is presented to the graphics card, of course, must be capable of holding the entire scene, but can be streamed from larger memory stores on a per-frame basis.¹ There is,

¹Limiting the need for per-frame data streaming can dramatically improve graphical performance, but this is generally treated as a separate problem from graphical LOD, as most rendering systems do not provide the application explicit control over the caching of data in graphics memory. Additionally, the use of “texture

as far as graphical LOD is concerned, exactly one resource: time.

In contrast, simulation can become impractical by taking up an immense amount of memory, rather than by taking too long to complete. For instance, large-step simulation of out-of-view objects, such as that described by Cheney [2001], can be used to amortize simulation costs over many frames, minimizing the simulation time with large amounts of objects; but these objects' state must still be held in memory at all times, and the "potential state-space" which is maintained for each object can consume a large amount of memory. As an extreme example, even simply maintaining the full name of each person in a simulation of the population of New York City would take over 100 megabytes of memory, but the per-frame computational cost of simulating the change of these names would be quite low (usually occurring, after all, only upon birth or marriage).

A more commonly occurring illustration of this problem is found in the maintenance of changes to the world. Modern video games which include firearm combat as a central game mechanic generally show holes or dents in objects which are hit by bullets. Over the course of a game, millions of shots may be fired. The simulation cost of these bullet holes is essentially zero, as once created they are never changed. However, to maintain them throughout the entire course of the game would take a large amount of memory. Likewise, moved furniture, dented mailboxes, etc. must be reset eventually in order to keep memory usage under control.²

A simulation LOD system must consider at least these two independent costs of detail. Past approaches to reactive graphical LOD have relied on there being only one constrained resource, either by using algorithms which do not support multiple resources

atlases" to minimize rendering pipeline stalls means that the set of textures held in graphics memory changes infrequently, usually only when moving between levels or zones.

²A piece of furniture, of course, requires the same amount of information to identify its position whether or not the player has moved it. However, by resetting an area of the world to its initial state, this information can be drawn from the high-capacity read-only storage used to distribute the game, rather than from per-saved-game storage.

[Funkhouser and Séquin, 1993], or by adjusting a single global level of detail modifier to maintain the framerate near a specified level [Hitchner and McGreevy, 1993], and are therefore inadequate for use in simulation LOD.

4.2.2 Short- and long-term LOD selection

In graphical LOD, the cost of LOD changes is minimal. Consider a simple proactive graphical LOD system which switched an object between two levels of detail based on its distance from the viewer, fading the change over several frames to reduce the “popping” effect. If the viewer were to hover on the threshold of the distance and pace slightly, causing the object’s level of detail to vacillate, the only runtime cost would be that of processing the fade. (There may be a penalty to realism caused by the frequent fading, but this is easily managed by applying hysteresis to the transition distance.)

In contrast, the computational cost and the cost to realism from switching LODs may be severe in a simulation LOD system. For instance, alibi generation (a technique for upgrading the realism of a single agent, to be described in chapter 5) takes a significant amount of time for each upgraded agent; a single frame of alibi-ful simulation is vastly more expensive than a single frame of alibi-less simulation. These ramping-up (or ramping-down) costs are justified by amortizing over many frames: once the alibi has been generated, alibi-ful simulation is only slightly more expensive than alibi-less simulation, and the alibi must only be generated once.

Simulation LOD should ideally be considered on a long-term basis, as opposed to the frame-by-frame allocation of a graphical LOD system. But reactive graphical LOD depends, to a large extent, on frame-by-frame allocation. Reactive graphical LOD which relies on measured framerates must have the ability to “experiment” with different levels of detail in order to maintain the accuracy of its predictions, by perturbing the level of detail of an object for a single frame and measuring the results; but these experiments are not feasible for

simulation LOD. Heuristic reactive LOD does not suffer from this issue, but in the context of simulation LOD it raises difficult questions of optimal resource allocation on a long-term basis.

4.2.3 Levels and transitions

Because graphical detail is drawn on a frame-to-frame basis, there are no constraints on transitioning between levels – the levels of detail available for rendering an object during a particular frame are independent of which levels of detail were chosen for the object in the past. In contrast, transitioning from one particular simulation strategy to another may not only be expensive, as explained in the previous section, but may not even be possible at all.

As an example, consider a vehicle simulation, which can simulate cars either as simple kinetic objects moving along predetermined lane lines, or as physically realistic bodies capable of changing lanes or even going off-road. Transitioning from the first level is straightforward: The initial position, orientation, and velocity of the physical body are copied from the simple kinetic object, and other state parameters such as steering angle are calculated to continue the vehicle’s immediate trajectory. But transitioning in the other direction – from physical body to kinetic object – may not be feasible. If a vehicle is off-road, changing lanes, or otherwise off the predetermined lane lines, it is impossible to transition in a way which preserves the object’s identity and state.

Another example serves to demonstrate both the usefulness of transitions and the dangers of executing them in a short-sighted manner. Consider a simulation – of any sort, really, but the vehicle one will do – when there are an additional two levels of detail: “frozen”, in which the entity is immobilized and not simulated until it comes back into view, as in Cheney [2001], and “removed”, in which the entity is removed from the world entirely.³ In the

³The latter level is conceptually murky – lack of existence as a state – but its implementation is straightforward enough.

short term, the two levels have equal benefit, since neither would be chosen for a vehicle which was on-screen. And clearly the “removed” level has lower cost: “frozen” requires a per-frame visibility test to unfreeze at the appropriate time, and allocated memory to store its state, while “removed” of course requires nothing. By this algebra, objects would never be frozen, only removed, a clearly naïve strategy.

Because transitions may be one-way, or otherwise sparsely connect levels of detail, the decision to perform one must take into account the future transitions which would be available, and the likelihood that they would be necessary. As in the example above, it is sometimes worthwhile to spend computational resources not for immediate gain, but as maintenance on an option to later transition to a level which would otherwise be unavailable. It may not be possible to do this in an optimal manner – the problem is one of planning under uncertainty, a notoriously computationally intensive task [LaValle, 2006].

A final factor complicating the idea of a “transition” is that a single entity may have several simulation components, which may be partially or completely independent. For instance, a vehicle entity may vary in the fidelity of its driving physics, as well as in the quality of the AI used to steer it. If each feature has three different levels of detail, the vehicle as a whole has nine possible levels of detail. It is tempting to avoid this combinatorial explosion by disassociating the components from each other, but this may not be possible: If the lowest AI level requires simple, skid-free dynamics which are not provided by the highest physics level, only eight levels for the vehicle should be considered. (As we shall see later, there are other reasons to consider only a small subset of the possible combinations.)

4.3 Optimal resource allocation as a knapsack problem

Funkhouser and Séquin [1993], in their predictive LOD system, used a cost-benefit formalism to determine what level of detail should be used to render each object. For each level of

detail of each object, there is a rendering time cost to rendering that object at that level, and a quantified benefit to doing so (based on how well it will improve the perceived realism of the scene as a whole). Because each object has a finite set of levels of detail to choose from, and the total cost must be below the time budget for the frame, the problem of choosing optimal LODs for all objects is the *Multiple-Choice Knapsack Problem* (MCKP).⁴ This problem is NP-complete [Garey and Johnson, 1990]; their approximation algorithm was a greedy iterative one, modifying whatever level of detail gave the greatest cost/benefit ratio until the solution became stable. By reusing the previous frame’s LOD choices as the initial state for objects in the next frame, the algorithm benefited from temporal coherence.

Likewise, Garvey and Lesser [1993] introduced the concept of “design-to-time” real-time scheduling, an approach to solving a problem consisting of a discrete set of subtasks, each with one or more possible methods of approximation, given constraints on the total computation time available. The authors used a similar heuristic to that of Funkhouser and Séquin [1993], picking downgrades which maximized cost/benefit. Unlike Funkhouser and Séquin, however, Garvey and Lesser did not approach the problem as one of optimization: A fixed lower bound to solution quality was pre-specified, and schedules which met this bound were considered equally acceptable. The system never upgraded subtasks to more time-consuming and accurate methods.

Lee and Fishwick [1998] approached this problem as one of choosing process abstractions from an ideal model. Their approach was to bound the minimum number of abstractions necessary to meet the time constraint, and then test all combinations of abstractions to find the one which provides the best accuracy while respecting the time constraint. This approach does not leverage temporal coherence, and despite not guaranteeing an optimal solution

⁴Funkhouser and Séquin described the problem in one place as the “*Continuous* Multiple-Choice Knapsack problem”, but this seems to be a typo: The continuous version of the MCKP allows a continuous-valued weight to be given to each chosen item, but no such provision can be made in LOD selection, and the authors’ proposed heuristic did not address the continuous problem.

(which might involve more than the minimum number of abstractions) its expected time complexity is exponential.

As described in section 4.2.1, simulation LOD must consider both processing and memory resources. Because both constrain the detail of the world, the resultant problem cannot be effectively modeled as MCKP. Instead, it must be modeled as what is known as a “Multidimensional Multiple-Choice Knapsack Problem” (MMKP) [Akbar et al., 2001].⁵

The approximation algorithm given in [Funkhouser and Séquin, 1993] is not applicable to the MMKP, as it considers only a single cost-benefit ratio. Toyoda [1975] developed the concept of “aggregate resource consumption” for the Multidimensional Knapsack Problem (MDKP) (which, like the MMKP, supports multiple resource constraints, but uses 0-1 allocation rather than multiple-choice allocation). For an item i with resource requirement vector \mathbf{r}_i and value v_i , and current total resource consumption vector \mathbf{C} , aggregate resource consumption for the item is given by $a_i = \frac{\mathbf{r}_i \cdot \mathbf{C}}{|\mathbf{C}|}$. Toyoda described an approximation algorithm for the MDKP with good empirical performance based on this metric, which starts with no items selected and greedily picks items with maximal v_i/a_i , recalculating aggregate resource consumption each time in order to guide the search toward items which draw primarily from slack resources. This approach was applied to the MMKP by Khan [1998], and later improved by Akbar et al. [2001] and Khan et al. [2002] with an iterative refinement very similar in operation to that of Funkhouser and Séquin: After the initial solution is found, a candidate solution is produced by picking the upgrade which maximizes value per unit resource consumption, followed by one or more downgrades to pay for the upgrade. The iterative improvement is repeated as long as candidate solutions have a higher total value.⁶

⁵The same problem is referred to as “d-MCKP” by, e.g., Kellerer et al. [2004].

⁶Khan approached the MMKP as a tool for adapting quality of service in a resource-limited multi-user multimedia system. This application is similar in many respects to simulation LOD control: it seeks to maximize perceptual quality in the presence of multiple resource constraints. However, resource constraints and usage are constant throughout a session, and quality rankings are explicitly provided rather than being heuristically determined. As a result, that application has little need for on-line updates of the allocation

4.4 The trader analogy

We propose an approximation algorithm for the MMKP, designed specifically for managing simulation level of detail, which combines a weighted value heuristic similar to that of Khan et al. [2002] with the temporal coherence exploitation of Funkhouser and Séquin. This algorithm selects levels of detail for all managed objects in a simulation on an on-going basis, adapting to changes in object importance, computational costs, resource availability, and the addition and removal of objects from the system.

The inspiration for the LOD Trader is economic: The system has a “budget” for simulation (the computational resources available), a set of “holdings” (the current LOD selections for all objects), and the ability to “trade” these holdings by changing an object’s level of detail. Each holding and each available trade has a “cost” (the computational resources consumed by that LOD) and a “benefit” (the ability of that LOD to maintain the perceptual realism of the simulation, as determined by *perceptual criticality modeling*, discussed in chapter 3). During each update, holdings and available trades may change their value, and under certain circumstances, their cost. The system then iteratively “buys” detail for objects where higher levels of detail provide advantageous cost-benefit ratios, and “sells” detail for objects with lower cost-benefit ratios in order to pay for them, ensuring that the final holdings can be afforded by the available resources.

4.4.1 Benefits and penalties

In the remainder of this chapter, we will describe both “benefits” for realism and “penalties” for realism. Although penalties more accurately describe the motivation behind LOD selection, the notion of “costs” and “benefits” is more straightforward to describe and reason about, and so the algorithm will be described in those terms.

solution.

4.5 The environment of the LOD trader

4.5.1 Specification of resources

The LOD trader supports an arbitrary number of heterogeneous resource types. These resources can each be expressed in any desired units, and it is not necessary to limit their relative magnitudes. Similarly, the preprocessing phase does not constrain the resource limits which can be imposed at runtime, and these limits may be adjusted from frame to frame. However, specifying constraints on the ratios between total available resources where practical may allow a slight increase in runtime performance by allowing the system to evaluate fewer transitions, as described in section 4.6.1.

For the case of memory and computation time as resources, we found it practical to specify resource usage in kilobytes and microseconds, respectively.

4.5.2 Specification of capabilities

As noted in section 4.2.3, an entity may have several “features”, each of which may be simulated through several methods. Certain features may be partially or wholly interdependent (as in the case of one of the AI simulation levels being incompatible with one of the physics simulation levels.) In fact, the LOD choice for one feature may remove the need to simulate another feature entirely. For instance, an entity which is temporarily frozen offscreen has no need to choose any LOD for its skeletal animation.

We represent the LOD-varying features inherent to an entity, and the relationships between them, via a *feature graph*, a variant of the *and-or tree*.⁷ This is a singly-rooted, directed acyclic graph, where nodes represent either features, or LODs for those features. The graph obeys the following rules:

⁷Kang et al. [1990] use “feature models”, a similar but more complex graph type, for representing features in commercial products.

1. The root node is a feature.
2. A feature node has one or more LOD node children, corresponding to possible LODs for that feature. Each of those LOD nodes has only the feature node as its parent.
3. An LOD node has zero or more feature node children, corresponding to features which must all be assigned LODs if the LOD is selected for the feature. Multiple LOD nodes can have a single feature node as a child.
4. An LOD node also has directed edges to zero or more LOD node children which share the same parent, corresponding to possible transitions between LODs.
5. An LOD node also has directed edges to zero or more LOD node children which do not share the same parent, corresponding to inter-LOD constraints which require that if the source LOD is selected, the destination LOD must also be selected.

An example tree is given in figure 4.1. This precise definition of feature graphs, and the resultant constraint capabilities (which are not comprehensive), is not crucial to their functionality; in particular, rules 3–5 may be replaced or augmented with any representation reducible to a set of logical constraints.

It is important to note that not all entities under the control of the LOD trader need to use the same feature graph. A separate feature graph can be used per entity type, each with its own features and LODs, and resources will be redistributed among the full set of entities, regardless of type. There is no need to artificially impose per-entity-type resource limits, only the actual global resource limits.

The goal of the LOD trader, then, is to find a “feature solution” for each entity. A feature solution is a mapping from feature nodes to LOD nodes, such that:

1. The root node is in the mapping.

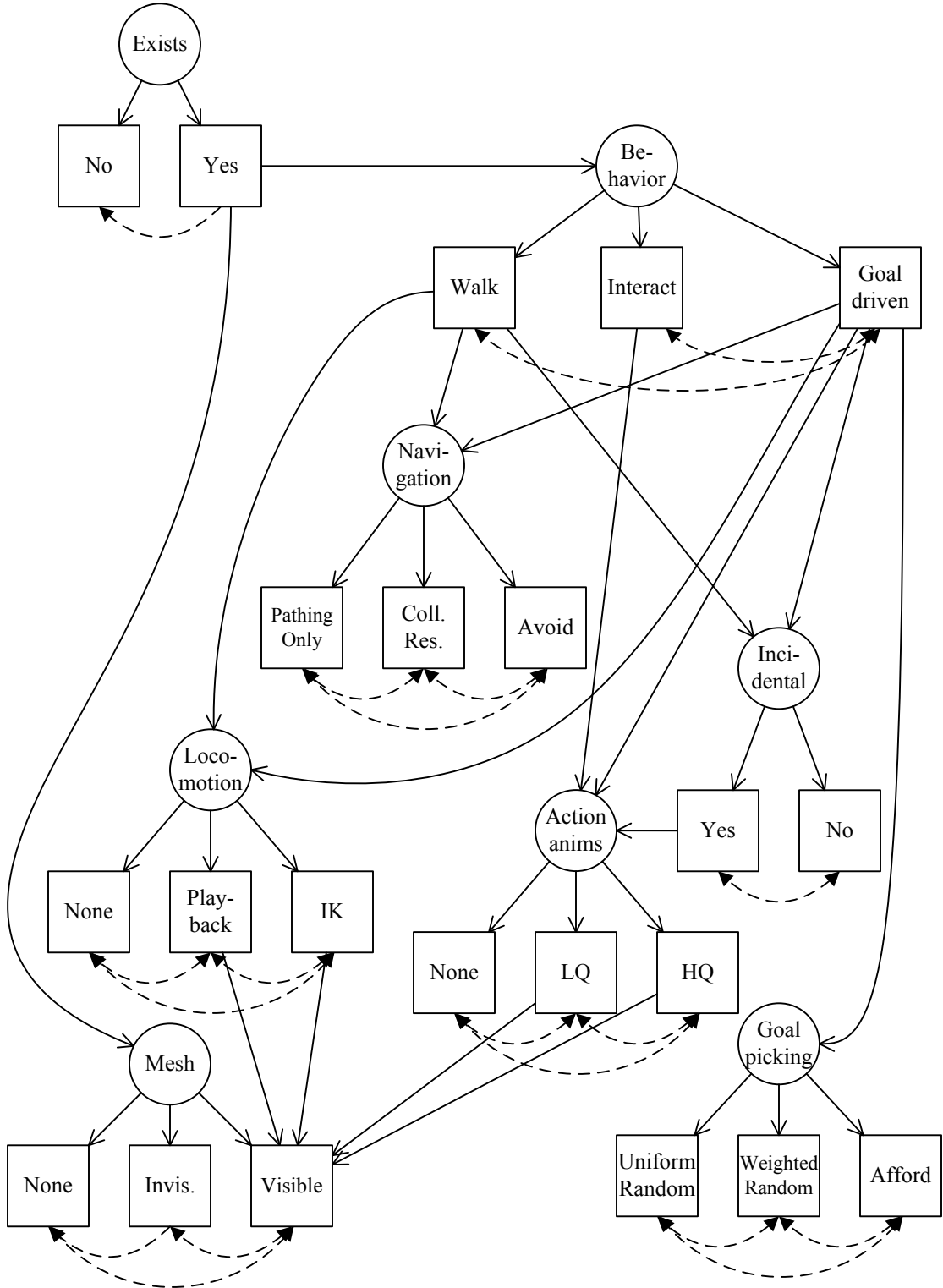


Figure 4.1: An example feature graph, with eight features. This feature graph has 252 feature solutions.

2. All feature nodes map to their own children.
3. Other than the root, a feature node is in the mapping iff one of its parent LOD nodes is in the mapping.

For example, one possible feature solution for the tree in figure 4.1 is $\{\langle \text{Exists}, \text{Yes} \rangle, \langle \text{Behavior}, \text{Interact} \rangle, \langle \text{Action anims}, \text{HQ} \rangle\}$; another is $\{\langle \text{Exists}, \text{No} \rangle\}$.

4.5.3 Enumerating feature solutions

The full set of possible feature solutions for a feature graph may be found by a simple recursive algorithm which expands incomplete solutions by making all possible choices for one of their required but unchosen features. If a different representation for the feature graph were used, the brute force method of enumerating all LOD combinations and filtering those that violated any constraint could be used instead. The time complexity of such a method would be exponential in the number of features.

Algorithm 4.5.1: FINDFEATURESOLUTIONS(*rootFeature*)

```
completeSolutions  $\leftarrow \emptyset$ 
incompleteSolutions  $\leftarrow \{\text{EMPTYSOLUTION}()\}$ 
while incompleteSolutions  $\neq \emptyset$ 
| solution  $\leftarrow \text{POP}(\text{incompleteSolutions})$ 
| requiredFeatures  $\leftarrow \{\text{rootFeature}\}$ 
| for each lod  $\in \text{GETLODS}(\text{featureSolution})$ 
| | requiredFeatures  $\leftarrow \text{requiredFeatures} \cup \text{FEATURECHILDREN}(\text{lod})$ 
| end
| chosenFeatures  $\leftarrow \text{GETFEATURES}(\text{solution})$ 
| if requiredFeatures = chosenFeatures
| | PUSH(completeSolutions, solution)
| else
| | feature  $\leftarrow$  some element of (requiredFeatures  $\setminus$  chosenFeatures)
| | for each lod  $\in \text{FEATURELODS}(\text{feature})$ 
| | | solution'  $\leftarrow$  solution
| | | ADDCHOICE(solution', feature, lod)
| | | if solution' does not violate any inter-LOD constraint
| | | | PUSH(incompleteSolutions, solution')
| | | end
| | end
| end
end
return (completeSolutions)
```

4.5.4 Costs, penalties, and transitions

An important result from chapter 3 is that the total estimated unrealism from a particular LOD choice can be computed as the product of a nonnegative row vector which is specific to that entity but independent of the entity's LOD, with a nonnegative column vector which is specific to that LOD but shared among all entities using that LOD. We refer to the former vector (really a linear functional) as a “criticality multiplier” or (equivalently) a “benefit multiplier”, and to the latter vector as an “audacity vector”, the negation of which can be considered a “benefit vector”.

Accordingly, each LOD node has a nonnegative vector-valued resource cost, and a

nonnegative vector-valued audacity vector, representing the computational cost and the penalty to realism which accrue from ongoing simulation of the feature at that level. The highest quality LOD for a feature will generally have a audacity vector of zero.

In addition, transitions between the different LODs of a feature are assigned a cost vector and an audacity vector, corresponding to the computational cost and the penalty to realism which accrue specifically from the act of transitioning between the two levels. Although it is not required for the proper operation of the LOD trader, common sense dictates that LOD transitions are transitive, and that their costs and penalties obey the triangle inequality – that is, that if it is possible to transition from LOD A to LOD B , and from LOD B to LOD C , then it is possible to transition from LOD A directly to LOD C , and the cost of doing so is not greater than the sum of the two other costs.

Finally, an LOD may have a transition to or from the “null LOD”, corresponding to the situation where another feature transition adds or removes that LOD’s feature’s need for simulation. These transitions have their own costs and benefits. (The null LOD itself has cost zero and benefit zero.) Most LODs will have transitions to the null LOD, though not necessarily from the null LOD.

The full benefit vector of a feature solution is equal to the sum of the benefit vectors (the negated audacity vectors) of all its feature LODs, and the full cost vector of a feature solution is likewise equal to the sum of the cost vectors of all its feature LODs. For a feature solution α the total benefit vector is referred to as b_α and the total cost vector is referred to as c_α .

The relative cost vector of a transition from one feature LOD to another is equal to the difference between the cost vectors of the two LODs, plus the cost of the transition itself. The relative benefit vector is likewise equal to the difference between the benefit vectors of the two LODs, plus the cost of the transition itself.

The relative benefit vector of a transition from one feature *solution* to another is equal to

the sum of all relative benefit vectors for features whose LODs differ between the solutions, and the relative cost vector is summed likewise. We denote transition benefits and costs from feature solution α to feature solution β as $b_{\alpha,\beta}$ and $c_{\alpha,\beta}$.

4.5.5 The cost multiplier

Just as an entity's criticality induces a linear functional on the space of audacity vectors which determines the overall realism penalty for that particular entity/transition combination, the resource constraints of the system induce a linear functional on the space of cost vectors which determines the overall resource effect of taking that transition. This cost multiplier is nonnegative, as it is never useful, all else being equal, to consume more resources. Without loss of generality, the L1 norm of the cost multiplier is constrained to 1, to streamline the entity elimination process described in section 4.6.1.

During the upgrade phase, the cost multiplier is proportional to available resources. During the downgrade phase, the cost multiplier of overspent resources is proportional to the amount of overspending; the cost multiplier of all other resources is 0. The cost multiplier is recalculated before each upgrade phase and each downgrade phase, but is not recalculated during the phase when a transition is chosen, as transitions could already have been discarded which would have been chosen given the new multiplier.

4.5.6 Upgrades and downgrades

It is useful to make a distinction between “upgrade” and “downgrade” transitions. An upgrade transition is one which could potentially result in a net benefit, for some valid benefit multiplier; this is a transition that should be considered during the upgrade phase of the algorithm. A downgrade transition is one with a nonpositive cost vector; this is a transition that reclaims resources without spending other resources, and should be considered

during the downgrade phase of the algorithm. Note that transitions with mixed cost vector signs are not considered as downgrades, despite the potential for them to reclaim a scarce resource by spending from an abundant resource; because the cost multiplier is zero for underspent resources during the downgrade phase, the aggregate resource heuristic does not accurately score these types of transitions as downgrades. A transition may be an upgrade, a downgrade, both, or neither.

4.5.7 Evaluating usefulness

Although the number of feature solutions for a feature graph may be quite high (in the hundreds or thousands), many of them have no practical use. For example, it would be a bad idea to simulate a person’s grasping animations with high-quality IK, and her locomotion with avoidance-free straight-line paths interpenetrating other people. Any character important enough for the IK would be important enough for at least basic avoidance behaviors. Put differently, there is no cost multiplier and benefit multiplier combination which would give this feature solution a better benefit/cost ratio than some other feature solution. The set of potentially useful feature solutions can be described as $\left\{ \arg \max_{\alpha} \frac{Bb_{\alpha}}{Cc_{\alpha}} \mid B \in \mathbf{B}, C \in \mathbf{C} \right\}$.

The potential usefulness of a feature solution may be evaluated through constraint satisfaction. Denoting the cost of a feature solution α as c_{α} and the benefit of the feature solution as b_{α} , the potential for a feature solution α to be better than a feature solution β in some circumstance is equivalent to the existence of some cost multiplier C and some benefit

multiplier B such that

$$\frac{Bb_\alpha}{Cc_\alpha} > \frac{Bb_\beta}{Cc_\beta} \quad (4.1)$$

$$Bb_\alpha \cdot Cc_\beta > Bb_\beta \cdot Cc_\alpha \quad (4.2)$$

$$Bb_\alpha \cdot c_\beta^\top C^\top > Bb_\beta \cdot c_\alpha^\top C^\top$$

$$B(b_\alpha c_\beta^\top)C^\top > B(b_\beta c_\alpha^\top)C^\top$$

$$B(b_\alpha c_\beta^\top - b_\beta c_\alpha^\top)C^\top > 0$$

Therefore, a feature solution is potentially useful iff

$$\exists B \in \mathbf{B}^*, C \in \mathbf{C}^* \text{ s.t. } \forall_{\chi \neq \alpha}, B(b_\alpha c_\chi^\top - b_\chi c_\alpha^\top)C^\top \geq 0. \quad (4.3)$$

This is a generalized bilinear program [Al-Khayyal, 1992], which may be formulated as a quadratically constrained quadratic programming problem [Al-Khayyal, 1992, Boyd and Vandenberghe, 2004] by solving for $x = [B \ C] \in \mathbf{B}^* \times \mathbf{C}^*$, representing each inequality in the quadratic form

$$[B \ C] \begin{bmatrix} 0 & M_{\alpha,\chi} \\ M_{\alpha,\chi}^\top & 0 \end{bmatrix} [B \ C]^\top \geq 0, \text{ where } M_{\alpha,\chi} = b_\alpha c_\chi^\top - b_\chi c_\alpha^\top. \quad (4.4)$$

Alternatively, one may simply randomly generate a large number of benefit/cost scenarios, evaluate which feature solution is optimal for each, and consider a feature solution potentially useful iff it is optimal for at least one benefit/cost scenario.

A related concept is the usefulness of transitions. This is a similar but distinct property: High transition costs may cause a particular transition to not be useful, even if the destination feature solution is useful as described above. (The converse, where a destination feature solution is not useful, yet a transition to it is useful, appears to be possible in contrived

circumstances but is unlikely in practice.) The random generation approach as described above may be used to evaluate the usefulness of a transition; the quadratic programming approach may not, as the denominators involved in equation (4.1) may be negative, making the step to equation (4.2) invalid. Upgrade and downgrade transitions should be considered separately when evaluating usefulness.

In our tests, roughly 10% of upgrade transitions were potentially useful, of which 50–80% were accepted. The culling process was less effective for downgrades, with 75% of transitions being potentially useful and about 12% accepted. However, these ratios are highly dependent on the feature graph and on relative resource constraints.

4.6 The design of the LOD trader

The LOD trader proceeds by hypothesizing a set of trades consisting of both upgrades and downgrades. After each set is hypothesized, the net benefit to realism that would result is calculated. If the net benefit is positive, the trades are made, changing the manner in which entities are simulated. If it is not, the set is discarded and no change is made.

Algorithm 4.6.1: RUNLODTRADER(*entities*, *availableResources*)

```
for each entity  $\in$  entities
|  $b[e] \leftarrow \text{CALCBENEFITMULTIPLIER}(\textit{entity})$ 
end
repeat
|  $\textit{savedAvailableResources} \leftarrow \textit{availableResources}$ 
|  $C \leftarrow \text{CALCCOSTMULTIPLIER}(\textit{availableResources})$ 
|  $\textit{upgrades} \leftarrow \text{SELECTTRANSITIONS}(\textit{entities}, C, \textit{upgrade}, \textit{availableResources}, C)$ 
|  $C \leftarrow \text{CALCCOSTMULTIPLIER}(\textit{availableResources})$ 
|  $\textit{downgrades} \leftarrow \text{SELECTTRANSITIONS}(\textit{entities}, C, \textit{downgrade},$ 
|  $\textit{availableResources})$ 
|  $\textit{transitions} \leftarrow \textit{upgrades} \cup \textit{downgrades}$ 
|  $\textit{netBenefit} \leftarrow \text{CALCNETBENEFIT}(\textit{transitions})$ 
| if  $\textit{netBenefit} > 0$ 
| |  $\text{APPLYTRANSITIONS}(\textit{transitions})$ 
| end
until  $\textit{netBenefit} \leq 0$ 
 $\textit{availableResources} \leftarrow \textit{savedAvailableResources}$ 
```

4.6.1 Efficiently eliminating un-promising transitions

Even with useless transitions culled, finding the downgrade transition (if any) which offers the best benefit/cost ratio for a particular entity is a non-trivial task, involving several multiply/adds and a floating-point division for each transition. For simulations with hundreds or thousands of entities, most of which are far away and undeserving of much computational resources, it is important to efficiently eliminate entities which are clearly not in need of upgrades, without checking all potentially useful transitions for each.

Consider the benefit/cost ratio for a particular upgrade transition. Clearly, this ratio will be maximized when the (scalar) cost is minimized, and the (scalar) benefit is maximized. The cost multiplier is constrained to be nonnegative, with an L1 norm of 1, and may have the ratios of its elements further constrained, as described in section 4.5.1. For a given transition, then, it is possible via simple constrained linear optimization to find the legal cost multiplier

which minimizes the scalar cost of this transition. The scalar cost thus realized may then be used to divide the benefit vector for the transition, without first taking the dot product of the benefit vector with the benefit multiplier for the entity; later, the dot product of this vector and the entity's benefit multiplier places an upper bound on the possible benefit/cost ratio of that transition. Restating, the resultant vector, which we denote $w_{\alpha,\beta}$ for a transition from feature solution α to feature solution β , is found as

$$w_{\alpha,\beta} = \frac{b_{\alpha,\beta}}{\min_{C \in \mathbf{C}^*} C c_{\alpha,\beta}}$$

and guarantees that

$$\forall_{B \in \mathbf{B}^*, C \in \mathbf{C}^*}, \frac{B b_{\alpha,\beta}}{C c_{\alpha,\beta}} \leq B w_{\alpha,\beta}.$$

For a particular starting feature solution, we can gather the w vectors for all valid and potentially useful transitions to other solutions into a matrix W_α , such that

$$\forall_{B \in \mathbf{B}^*, C \in \mathbf{C}^*, \chi}, \frac{B b_{\alpha,\chi}}{C c_{\alpha,\chi}} \leq \|B W_\alpha\|_\infty. \quad (4.5)$$

Not all vectors need be present in W , as some are dominated by others (even among potentially useful transitions); the reduced set may be found as the vertices of the intersection of the upper convex hull of all w with the space of valid benefit multipliers.

For all entities in the simulation, once their benefit multipliers are known, equation 4.5 may be used to derive upper bounds on the benefit/cost ratios for upgrades to those entities (an operation consisting of only multiply/adds, compares, and conditional moves, which can be implemented without conditional branches), and a max-heap of entities may be built, keyed on those upper bounds. This heap, the “expansion queue”, is used to pick an entity to expand into its set of potentially useful upgrades.

Algorithm 4.6.2: MAKEEXPANSIONQUEUE(*entities*, *transitionType*)

```
expansionQueue  $\leftarrow$  EMPTYHEAP()
for each e  $\in$  entities
    | ratioBound  $\leftarrow$   $W[\text{featureSolution}[e]] \cdot b[e]$ 
    | ADDTOHEAP(expansionQueue, ratioBound, e)
end
if transitionType = upgrade
    | MAXHEAPIFY(expansionQueue)
else
    | MINHEAPIFY(expansionQueue)
end
return (expansionQueue)
```

Expanding an entity from the expansion queue is performed by examining all available transitions for the entity, and picking the transition which gives the best benefit/cost ratio.

Algorithm 4.6.3: EXPANDEntity(*entity*, *C*, *transitionType*)

```
bestTransition, bestRatio  $\leftarrow$   $\langle \emptyset, \emptyset \rangle$ 
transitions  $\leftarrow$  AVAILABLETRANSITIONS(featureSolution[entity],
                                           transitionType)

for each transition  $\in$  transitions
    | ratio  $\leftarrow$   $(B[\text{entity}] \cdot b[\text{transition}]) / (C \cdot c[\text{transition}])$ 
    | if ratio is better than bestRatio
    | | bestTransition, bestRatio  $\leftarrow$   $\langle \text{transition}, \text{ratio} \rangle$ 
    | end
end
return (bestTransition, bestRatio)
```

4.6.2 Entity selection

The goal of the LOD trader is to choose upgrade transitions which maximize benefit/cost, and then to pay for them by making downgrades which minimize benefit/cost. To that end, throughout the selection of upgrade transitions, a set of upgrades is maintained with the invariant that it *slightly* overspends the resources available: if the lowest-ratio transition were

discarded from this set, it would no longer overspend any resource. This set is maintained as a min-heap. To initialize this transition heap, transitions are added by expanding entities popped from the expansion queue until the resources are overspent, and then the set is heapified.

Algorithm 4.6.4: $\text{FILLTRANSITIONHEAP}(\text{expansionQueue}, C, \text{transitionType}, \text{availableResources})$

```

transitionHeap  $\leftarrow$   $\text{EMPTYHEAP}()$ 
while availableResources  $\geq 0$ 
    entity  $\leftarrow$   $\text{POPVALUE}(\text{expansionQueue})$ 
    transition, ratio  $\leftarrow$   $\text{EXPANDEntity}(\text{entity}, \text{transitionType})$ 
     $\text{ADDTOHEAP}(\text{transitionHeap}, \text{ratio}, \text{transition})$ 
    availableResources  $\leftarrow$  availableResources  $- c[\text{transition}]$ 
end
if transitionType = upgrade
     $\text{MINHEAPIFY}(\text{transitionHeap})$ 
else
     $\text{MAXHEAPIFY}(\text{transitionHeap})$ 
end
return (transitionHeap)

```

More entities are then popped from the expansion queue, expanded to upgrades, and pushed into the transition heap. After each push, transitions are popped from the transition heap to maintain the “slight overspending” invariant. Once the maximum key of the expansion queue is lower than the lowest ratio actually chosen for the upgrade set, the remainder of unexpanded entities may be disregarded without further computation.

Algorithm 4.6.5: EXPANDREQUIREDENTITIES(*expansionQueue*, *C*, *transitionHeap*, *availableResources*)

```

while PEEKHEAP(expansionQueue) is better than PEEKHEAP(transitionHeap)
    entity  $\leftarrow$  POPHEAP(expansionQueue)
    transition, ratio  $\leftarrow$  EXPANDENTITY(entity, C, transitionType)
    PUSHHEAP(transitionHeap, ratio, transition)
    availableResources  $\leftarrow$  availableResources - c[transition]
    while (availableResources + costs[PEEKVALUE(transitionHeap))]  $\nless$  0
        transition  $\leftarrow$  POPVALUE(transitionHeap)
        availableResource  $\leftarrow$  availableResources + c[transition]
    end
end

```

The result of upgrade transition selection is the values remaining in the transition queue after all required entities have been expanded, either through the initial heap fill or further expansion.

Algorithm 4.6.6: SELECTTRANSITIONS(*entities*, *C*, *transitionType*, *availableResources*)

```

expansionQueue  $\leftarrow$  MAKEEXPANSIONQUEUE(entities, transitionType)
transitionHeap  $\leftarrow$  FILLTRANSITIONHEAP(expansionQueue, C,
                                         transitionType, availableResources)
EXPANDREQUIREDENTITIES(expansionQueue, transitionHeap, C,
                       availableResources)
return (VALUES(transitionHeap))

```

4.6.3 Downgrades

The selection of downgrades is congruent to the selection of upgrades; the system attempts to pick downgrades which minimize the benefit/cost ratio (with a small negative benefit and a large negative cost), and the transition set invariant is that no resource is overspent but if the highest-ratio downgrade were removed then at least one resource would be overspent.

4.6.4 Initial LOD assignments

When the simulation begins, initial LOD assignments are needed to seed the algorithm. Khan et al. [2002] start with the lowest-benefit assignment for each object, with an additional backup plan if this violates any resource limits. Finding this initial solution is non-trivial in the generalized MMKP (in fact, it is itself NP-complete), but in the case of simulation LOD, most objects will have a simulation level with extremely low utilization of all resources, meaning that a clear path exists from any non-feasible solution to a feasible solution. The choice of initial LOD assignments is therefore of less concern from this point of view. As we shall discuss in chapter 6, the choice of initial LOD assignments is largely dictated by the circumstances under which entities are created.

Chapter 5

I found myself halfway between the perception of the concept ‘horse’ and the knowledge of an individual horse... I could expect all horses, but not because of the vastness of my intellect, but because of the paucity of my deduction.

Umberto Eco, *The Name of the Rose*

Alibi generation

In this chapter, we describe the technique of alibi generation. We originally created alibi generation as a standalone perceptually-driven real-time simulation system for pedestrian agents within a large open-world video game. In this form, it does not interface with a coordinating system such as the LOD trader, and does not require the quantification of unrealism. In sections 5.8 and 5.9, we discuss how alibi generation may be embedded into a larger framework of perceptually driven simulation, and how its techniques can be extended beyond pedestrian goal generation into other domains.

5.1 Introduction

In recent years, the “open world” style of video game, in which players wander freely through a large populated area, has become very popular. Most agents in open world games are not enemies, but residents, going about their life as the backdrop of the main gameplay. At times they are drawn into the gameplay, and must react to the player’s behavior and to the indirect results of that behavior. In order to maintain suspension of disbelief in these games, it is important that agents behave realistically in all situations, giving the impression

of a living city. Patterns of unrealistic or unlikely behavior negatively impact the realism of the game.

The increased demands for agent realism and large, open worlds present practical difficulties for game developers. Memory and processing power is limited on video game hardware, and much of it is devoted to graphics, sound, and other resource-intensive tasks. As the size and complexity of the virtual population grows, it quickly becomes intractable to simulate the virtual population in real-time.

Luckily, real-time simulation of the entire virtual population is not required for these games. Because the player can only see a small portion of the world at a time, she cannot evaluate the overall realism of the world, only that portion which is visible to her. As far as the player is concerned the world is realistic as long as the visible portions of it at any given time are realistic.

These twin demands — thrifty use of computational resources, and the appearance of realistic simulation — have driven the development of a variety of clever methods (tricks) which maintain the *perception* of realism while reducing the actual realism. For instance, out-of-view portions of the world will have their agents deleted entirely, with a set number randomly created when the player returns to the vicinity. Even agents which are invisible because they are behind the character may be frozen in place in order to free up simulation resources. Agents will often have only a veneer of realistic behavior, appearing at first glance to be moving towards some desired destination while actually turning randomly at each intersection.

Such tricks are difficult to get just right. User testing is required to determine whether they compromise realism in unanticipated ways. Parameter tweaking is often necessary to arrive at a realistic configuration, and the parameters may not map cleanly to quantities which designers wish to directly control. For example, if the rate at which agents enter an area does not match the initial population, the agents' speed of motion, and the geometry

of the area, agents may exit the area at a greater rate, and the player will perceive that an area is always well-populated when she arrives, only to mysteriously empty out soon afterwards. In contrast, if the rates of exit and entry are directly tied to each other, the player may perceive that a small area always happens to have three people in it, with one entering every time another leaves. Agents with only random turning will quickly reveal their unrealistic behavior if followed by the player. These tricks may produce realistic results early in development, only to become unrealistic later due to designers' varying demands on the world.

We have developed an alternative approach, whereby designers build a highly realistic whole-world pedestrian simulation, from which a perceptually identical "perceptual simulation" is generated. Although the perceptual simulation simulates only a small portion of the world at a time, and does so with inexpensive approximations, it can be statistically guaranteed that the results are perceptually indistinguishable from those of the original simulation.

The basis of the simulation is the probabilistic modeling of observable information about an agent. Agents are initially created with only a minimal set of information, and any information which is later required for a higher level of detail is created on demand in a manner which ensures consistency with previous observations of the agent. We refer to such an incremental addition of information about an agent as an **alibi**, which fills in details of the agent's goals and back-story to give the impression that its behavior is driven by concrete needs and goals. The act of giving an agent an alibi causes no immediately visible changes, but allows the realism of the agent's actions to hold up under closer scrutiny from the player.

5.2 Motivating example

Here we present a simple but detailed pedestrian behavior model, which describes where agents go, when they go there, and why they go there. This example is intended to showcase various features that are difficult to model with perceptual simulation. By itself this model is unsuitable for actual use in a large, heavily populated world, as it requires full simulation of out-of-view agents at all times, thereby motivating the need for a perceptually identical real-time simulation. We do not intend this model to be perfectly realistic: it ignores certain important dimensions, like time of day, in favor of a straightforward presentation.

5.2.1 World schema

The world is partitioned into walkable region “cells”, which are connected to each other by “portals”. Agents walk from cell to cell through the portals. In a city model, a length of sidewalk or a crosswalk will be represented as a cell.

In addition to walking between portals, agents may exit and enter buildings. The entryways of these buildings are known as “targets”. Targets are categorized into a dozen or so “types”, representing what the building is used for. Examples of target types in our model are “house”, “office building”, “restaurant”, and “grocery store”. (A building with multiple uses will be represented as several co-located targets.) The number of targets in the world is much greater than the number of target types. Targets are additionally organized by which cell they are in.

All portals and targets in a given cell can be directly reached from all other portals and targets in the same cell, without passing through any other portals. In our navigation graph, portals and targets form the nodes of our world, and any two nodes which are in the same cell have an edge between them. Each portal is present in two cells, and is connected to all other nodes in both. We refer to a directed edge in the graph as a “segment”, a linear-shaped

area along which pedestrians may be found, walking in a particular direction away from one node and towards another. We denote by n the number of targets in the world, by m the number of target types, and by s the number of segments in the world.

5.2.2 Agent behavior

An agent, at any given time, will either be inside a target, or will be travelling from its previous target to its next one. Agents always travel along the shortest path through the world to their next target. (For distance calculations, portal locations are measured as the midpoint between their edges; for pedestrian simulation along sidewalks, this is not a significant oversimplification.) Upon arriving at a target, an agent spends a randomly determined amount of time there, which we model using a normal distribution. Afterwards, it chooses a “goal” randomly, from a distribution conditioned on the type of the target it is currently at. A goal consists of a type of a target to choose as a destination, as well as instructions on how that target is determined; the agent may have a particular preferred target for that goal which is determined when the agent is created (for instance, the agent’s own home will always be chosen for the “go home” goal), may choose the closest target of that type (as would be appropriate for grocery stores and other such fungible destinations), or may uniformly randomly choose a target of that type (for, say, a courier delivery to an office building). Certain goals may be round-trip; after leaving a target reached by a round-trip goal, the agent will return to the target from which it had previously departed. Other than returning from round-trips, the agent’s choice of goals is independent of its previous goals and the goals of other agents; a target type therefore has an associated goal distribution, consisting of the probability of choosing each goal next.

We simulate low-level pedestrian motion control using a simple social forces model. The method is compatible with other crowd motion controllers, such as the many cited by Pelechano et al. [2008].

5.3 Components of perceptual simulation

Maintaining the perceptual realism of our real-time simulation requires two main tasks: Adding and removing agents as necessary to ensure the realism of aggregate behavior without overtaxing simulation resources, and generating alibis to upgrade agents to keep their behavior realistic under scrutiny.

5.3.1 Creating and deleting

Agents are created in two circumstances: *in media res* when a new area of the world comes into view, and entering at random intervals from targets or areas of the world which are out of view.

For the first circumstance, whenever a new cell comes partially into view, agents are generated along each of its segments. An individual agent from among the entire population of the world will have a particular steady-state probability of being on a given segment when that segment's tile comes into view, and (without knowledge of agents' bound targets) each agent in the world will have the same such probability. The number of agents present on a segment when it comes into view, therefore, can be modeled as a binomial process, with n the population of the world not currently visible and p the probability that a given agent is on the segment at a given time. Simple and efficient algorithms to generate binomial variates are well-known; for several examples refer to [Devroye, 1986]. Because of the high population of the world and the high number of segments in the world, this distribution can also be approximated by a Poisson distribution if desired [Devroye, 1986].

For the second circumstance, each segment beginning in a target, and each segment whose starting portal is out of view, has a given probability during each small time-step of having an agent appear, and in the steady state of the simulation this probability is independent of previous appearances (with the exception of social forces, discussed later).

As a result, the random variable representing the time before the next agent appears is exponentially distributed, with λ the inverse of the expected time between arrivals on that segment [Devroye, 1986], and can be sampled in constant time. This arrival process is not applicable to segments whose starting node is a portal in view of the player — that is, whose companion cell is visible — as these segments will have already-generated agents arriving from the companion cell.

At the moment when a new tile comes partially into view after being completely out of view, therefore, three tasks must be performed: The number of agents along each segment within the tile must be sampled and the resultant number of agents generated, and each segment whose starting node is a target or a portal whose other tile is still out of view must have a “next arrival time” sampled and remembered. Arrival times for all segments of this type are placed in a priority queue. Whenever a scheduled arrival occurs, a new agent is generated at the beginning of that segment, and a new exponential variate is sampled for the segment and placed back in the priority queue. Finally, any segments currently in the arrival priority queue whose starting node was a portal on that cell are removed from the queue, as their agents will henceforth arrive in already-generated form.

Cells which pass entirely out of view must continue to be simulated for a short time, or a return to the cell may produce obvious discrepancies. The cell must continue to be simulated for at least as long as it takes for the set of agents in the cell to change entirely. Following [Sung et al., 2004], this burden may be alleviated by freezing simulation of the agents, and advancing them at once only if and when the cell comes back into view. Likewise, agents which pass from a visible cell into an invisible cell must continue to be simulated for some time, but can also be simulated in a frozen manner.

5.3.2 Alibi generation

While the previous section suffices to distribute momentarily plausible agents over the visible portion of the world, it does not by itself give them plausible long-term behavior. A short-term solution is to store the conditional probability table for transitioning to a given segment given the agent's last few segments; this table can be stored in a compressed form [Gagie, 2006] but the memory requirements quickly become intractable as the segments grow longer. If the player follows an agent for any significant period of time, interacts with the agent in a nontrivial manner, or inspects its behavior in any other way (for instance, by blocking the agent's segment and forcing it to replan) an actual goal destination is required.

The agent's destination, and any other information about the agent generated some time after its creation, forms the agent's *alibi*. As far as the player is concerned, this information had always existed but was not known to her. An alibi, therefore, must be consistent with all previously observed information about the agent, and the geometry of the world and the distribution and habits of agents in the world in general. Additionally, over time there must be no observable patterns to alibis, even those generated for different agents in exactly the same situation. In short, any alibi must be an independent, fair sample from the conditional distribution of possible alibis given observed behavior. In section 5.5, we show how this may be done without explicitly storing the conditional probability table.

5.4 Deriving the probabilities

We now return to the motivating example in section 5.2, and show how the distribution parameters mentioned above can be analytically determined. The general approach is to determine the different circumstances under which an agent can have a particular alibi (source and destination), and then to weight and sum these circumstances to determine the prior probability of having that alibi. We then filter to determine the posterior probability of

an alibi given the agent’s observed path.

First, some definitions. For a given pair of nodes i, j , We denote the distribution of the time required for an agent to travel from target i to target j by D_{ij} . (Here and for the remainder of this paper, we use i and j as target indices, and k and l as target type indices.) We denote the type of target i by $S(i)$, and the number of targets of type k in the world by $\|S_k\|$. We denote the conditional probability of choosing a goal with type l from target type k , assuming the previous goal was not round-trip, by p_{kl}^G ; we further split this into one-way and round-trip probabilities p_{kl}^O and p_{kl}^R , which are the normalized conditional probabilities of choosing a goal with type l given that the previous one-way goal was of type k and that the next goal is constrained to be, respectively, one-way or round trip. We denote the conditional probability of picking any one-way goal from a target of type k by p_k^R , giving the identity $\sum_{l=1}^m p_k^R p_{kl}^R + \sum_{l=1}^m (1 - p_k^R) p_{kl}^O = 1$. We denote the distribution of the waiting time at a target of type k by W_k . Finally, we denote by $closest(i, l)$ the closest target to i of type l .

Recall that goals may be round-trip, forcing a return to the original target after leaving the destination, or one-way, with future goals sampled independently of past goals. Consider the sequence of targets which an agent reaches as a result of one-way goals only, disregarding for the moment the sequence of round-trip goals which it may perform between each one-way goal. This continuous-time process of one-way goals may be modeled as a *semi-Markov process* [Ross, 1996]. A semi-Markov processes is a random process which, like a standard Markov chain, moves from state to state with transition probabilities independent of all but the most recent state, but which will stay in a particular state for a real-valued period of time whose distribution depends on the current and/or the next state. The limiting behavior of a semi-Markov process can be described by the tuple $\langle p_{ij}, F_{ij} \rangle$, where p_{ij} is the probability of transitioning to state j given current state i (the “jump process”), and F_{ij} is the distribution of the time for that transition to occur. We denote an agent’s current state in the semi-Markov

process as the last target the agent reached by a one-way goal, irrespective of whether they are still inside that target as well as of whether they have performed other round-trip goals since reaching it. For this process, $p_{ij} = p_{S(i)S(j)}^O / \|S(j)\|$ if $S(j)$ is not fungible, $p_{S(i)S(j)}^O$ if it is fungible and $\text{closest}(i, S(j)) = j$, and 0 otherwise. (Without prior information on bound targets, there is no need to differentiate between goals to bound targets and goals to uniformly chosen targets.) Note that the jump process is independent of all but the target types, because the agent chooses from between goals directly, but the transition time is dependent on the specific targets, because different targets may take more or less time to transition due to their positioning in the world.

While p_{ij} is simple to calculate as above, the calculation of F_{ij} is more complex. We consider the agent's actions upon arriving at a target i , of type k . First it waits for a time given by W_i . It will then carry out zero or more round trip goals; for each goal to some target j , it walks to that goal, taking time given by D_{ij} , waits at that goal for time W_j , walks back to the original target taking time D_{ji} , and then waits again at target i for time W_i . Finally, it walks to its next one-way destination j for time D_{ij} , at which point it is in the next state and the clock stops. We denote the round-trip travel time for a round-trip destination to a target of type l by D_{il}^R , remembering that this may be dependent on whether l is a fungible target type. The number of round-trip goals it chooses before leaving on a one-way goal is given by a negative binomial random variable [Ross, 1996] with $r = 1$ and success probability $p = p_k^R$, with expected value $\frac{p}{1-p}$, and the proportion p_{kl}^R of those are specifically round-trip goals to target type l ; we denote the number of those round-trips taken by N_{kl} . Organizing one-way goals by their target type, we thus have

$$E[F_{ij}] = E[W_k] + \sum_{l=1}^m E[N_{kl}] (E[D_{il}^R] + E[W_l] + E[W_k]) + E[D_{ij}]. \quad (5.1)$$

It remains to calculate these expected values. $E[W_k]$ is simply the mean of the nor-

mal distribution, and $E[N_{kl}] = p_{kl}^R (p_k^R / (1 - p_k^R))$, as above. $E[D_{il}^R]$ can be calculated as $\min_{j, S(j)=l} E[D_{ij}] + E[D_{ji}]$ if l is fungible, and $\|S_l\|^{-1} \sum_{j, S(j)=l} E[D_{ij}] + E[D_{ji}]$ otherwise.

This is sufficient to describe the steady-state probability of the semi-Markov process. First, the steady state of the jump process σ is given by the equation $\sigma_j = \sum_{i=1}^n \sigma_i p_{ij}$ (recall the definition of p_{ij} as above), and can be found as the eigenvector of $[p_{ij}]$ with eigenvalue 1. We can marginalize F_{ij} over the destination target as $F_i = \sum_{j=1}^n p_{ij} F_{ij}$, and then entirely as $F = \sum_{i=1}^n \sigma_i F_i$. The steady state of the semi-Markov process itself is then $\hat{\sigma}_i = \sigma_i \frac{E[F_i]}{E[F]}$. The probability $\hat{\sigma}_i$ of having last visited one-way target i at any given time can be further broken up; for instance, the probability of having just reached target i and being currently waiting inside it before leaving either for the first round-trip goal or for the next one-way goal is $\sigma_i \frac{E[W_i]}{E[F]}$. Additionally, $\hat{\sigma}_i$ can be used to determine the probability of being on one's way to, or coming back from, a particular goal. Since any agent who is not inside a target will be on a one-way goal, going to a round-trip goal, or coming back from a round-trip goal, we can find the summed joint probability of being on one's way from target i to target j for any reason as

$$P(i, j) = \alpha_{ij} + \beta_{ij} + \gamma_{ij}, \text{ where} \quad (5.2)$$

$$\alpha_{ij} = \hat{\sigma}_i p_{ij} \frac{E[D_{ij}]}{E[F_{ij}]}, \quad (5.2a)$$

$$\beta_{ij} = \hat{\sigma}_i \delta_{ij} \frac{E[N_{S(i)S(j)}] E[D_{ij}]}{E[F_i]}, \quad (5.2b)$$

$$\gamma_{ij} = \beta_{ji}, \text{ and} \quad (5.2c)$$

$$\delta_{ij} = p_{S(i)S(j)}^R \cdot \begin{cases} \|S(j)\|^{-1} & \text{if } S(j) \text{ is not fungible,} \\ 1 & \text{if } \textit{closest}(i, S(j)) = j, \text{ or} \\ 0 & \text{otherwise.} \end{cases} \quad (5.2d)$$

From here, one can easily find the joint probability of going from i to j and also being

on a segment $\{a, b\}$ as

$$P(i, j, \{a, b\}) = P(i, j) SP_{ij}(\{a, b\}) \frac{E[D_{ab}]}{E[D_{ij}]}, \quad (5.3)$$

with $SP_{ij}(\{a, b\})$ equal to 1 if $\{a, b\}$ is on the shortest path from i to j and 0 otherwise, and can sum over i and j as

$$P(\{a, b\}) = \sum_{i=1}^n \sum_{j=1}^n P(i, j, \{a, b\}) \quad (5.4)$$

which gives us the parameter for our binomial and exponential variates, to sample populations and arrival intervals. (This can be extended to observed paths consisting of multiple segments.)

5.5 Alibi Sampling

In this section we demonstrate how to use the above derivations for practical alibi sampling. While it is straightforward to find $P(i, j | \{a, b\})$ for an individual situation, sampling from this distribution would be more difficult. It would be possible to precompute all values and store their cumulative sums, and sample using a binary search, but this would require $O(n^2 s)$ space for single-segment observed routes and even more for longer observed routes, far from practical for worlds of large size. (It is important to sample both i and j , in case the resultant alibi is the first half of a round trip.) Instead, we express this probability with the equivalence $P(i, j | \{a, b\}) = P(i, j, \{a, b\}) / P(\{a, b\})$. Note from equations 1–3 that this quantity is entirely dependent on $E[D_{ij}]$, $SP_{ij}(\{a, b\})$, $E[D_{ab}]$, $E[D_{il}^R]$, $E[W_k]$, $E[F_i]$, $P(\{a, b\})$, $\hat{\sigma}_i$, $closest(i, l)$, $E[N_{kl}]$, p_{kl}^O , and $\|S_k\|$. Of these, the first two can be determined through shortest path searches, the third is taken directly from the world, and the remainder can be precomputed and stored with size $\Theta(s + m^2 + mn)$ in the number of segments,

target types, and targets. (Recall that $m \ll n$.)

This approach does not, however, allow for a binary search-based sampling. Instead we use the Metropolis-Hastings algorithm [Hastings, 1970] to sample from this distribution based only on the ability to calculate the ratio of individual probabilities, and a function to iteratively perturb i and j randomly in space. This algorithm uses a Markov chain whose steady state distribution is the desired distribution, sampling from it after a burn-in period which allows the distribution to converge. For the perturbation function, we store at each target a table of the nearest q targets, with one-way transitions pruned to avoid a division by zero in the transition probability ratio. The probability ratio is calculated from the two i, j pairs as in equation 5.3, noting that $P(\{a, b\})$ and $E[F]$ cancel out. $E[D_{ij}]$ and $SP_{ij}(\{a, b\})$ are iteratively maintained using A* searches from a to i , from b to j , and from i to j ; the first two change only their goal nodes, and thus computations can be reused by keeping the closed list and lazily recomputing the heuristic for all nodes on the open list. For the third search, MT-Adaptive A* search [Koenig et al., 2007] shows promise for speeding up computation.

5.6 Results

We have tested the perceptual simulation described using a world with approximately 12,000 targets and 1,000,000 agents (approximately 80% of whom were inside at any given time), on an Intel Core i7-based computer. The persistent memory overhead of the precomputed data, not including data about the geometry of the world, was approximately 52 kB, scaling roughly linearly with the number of targets. Sampling the population of all segments in a cell took less than 0.1 ms. Using the Kullback-Liebler divergence between the the distribution of generated alibis and the ground-truth probabilities to monitor the convergence of the Metropolis-Hastings chain to the equilibrium, we found satisfactory convergence for alibi

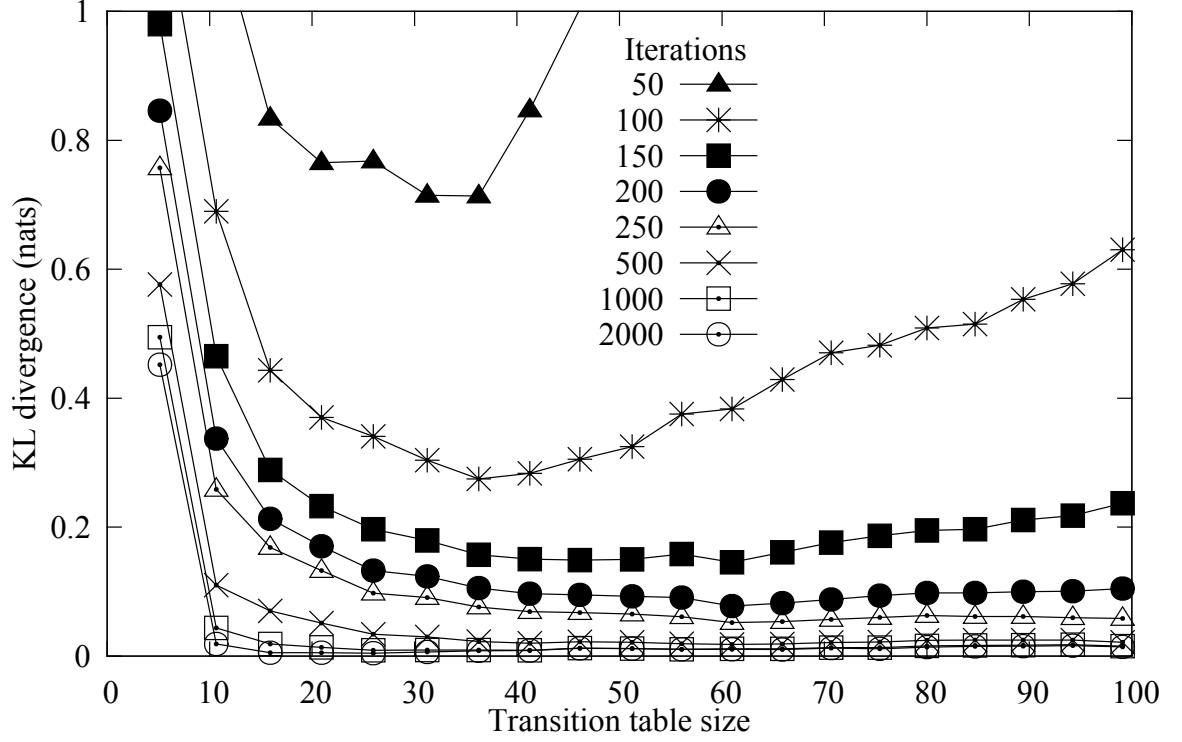


Figure 5.1: Kullback-Liebler divergence of alibi sampling versus ground-truth probabilities, under varying Metropolis-Hastings transition table size and number of iterations.

sampling with 500 iterations and transition tables of size $q = 50$. The optimum transition table size was dependent on the number of iterations (Figure 1). Alibi sampling took approximately 15 ms; this computational load can easily be spread over hundreds of frames if desired, making the additional computational burden affordable for even the stingiest of AI time-slices, and can be used as an anytime algorithm in especially processing-intensive situations.

Alibi sampling, therefore, produces agents whose observed behavior is virtually indistinguishable from those in the original simulation model. Players would have to analyze the behavior of hundreds or thousands of agents in order to determine whether the simulation they were viewing was the original simulation or the perceptual simulation, an activity which is not usual for video game players.

5.7 Drawbacks

There is a subtle but meaningful caveat to the guarantee of realism provided by alibi generation. Because agents which pass into an out-of-view cell are destroyed, it is possible for a viewer to realize she is observing unrealistic behavior by observing someone nearing a portal, stepping out of view so that the destination cell becomes invisible, then returning. Since the agent would have reached the portal while the destination was invisible, no alibi would be generated for it, and the agent would have vanished. A possible solution would be to generate alibis for agents passing into cells which were *recently* visible, but this could greatly increase the number of alibis required, lowering performance.

Likewise, the treatment of alibi-ful agents which pass out of view becomes a compromise between realism and performance. If alibi-ful agents are destroyed after a short period of invisibility, a viewer following one from afar can notice them vanishing after they are temporarily obscured (a situation which could easily arise in real gameplay, instead of as a “gotcha”). If they are allowed to linger for an unlimited amount of time, over time the simulation can tend towards most agents having an alibi, removing the benefits of alibi generation.

Finally, unnecessary alibi generation can often occur, and in a problematic fashion. Newly visible cells create agents at random positions along segments, so some agents will inevitably be very close to the end of their segments. This causes these agents to invoke alibi generation almost immediately, despite the fact that the player may not have even noticed them yet. Moreover, if the processing burden of alibi generation is being spread over many frames, there will not be time to finish the generation process for these agents. These two problems can actually be used to solve each other, to some extent: if alibi generation has not completed for an agent, the most recently generated alibi may be used solely to compute the next segment for the agent, but that alibi may then be discarded and the agent remain in

alibi-less simulation until the next segment. This puts a de-facto lower limit on the amount of time visible before an agent is considered to “deserve” an alibi, although its precise value is more an artifact of processing than a well-supported compromise.

Solutions to these problems are discussed in the following section.

5.8 Integration with the LOD trader

The original alibi generation system cannot be precisely implemented as a subordinate of the LOD trader. With minor modifications, however, it can be, and gains desirable properties which were not available without the LOD trader. There are two potential forms for this integration, which will be described in turn.

5.8.1 Cells as entities

In this form, alibi generation as described in this chapter uses two separate feature graphs: One for cells, and one for agents.

The “cell” feature graph is the simpler of the two, and exists primarily to drive the rules for agent creation. It consists of one feature, with two LODs: visible and invisible. The “invisible” LOD, of course, has an extremely high audacity coefficient for unrealistic state. The cost of simulation at the “visible” LOD is based on the expected cost of simulation of newly created agents. The transition from invisible to visible also has significant resource costs, as it entails the creation of agents; however, the high audacity coefficient for unrealistic state means that, under all but the most extreme circumstances, this transition will occur whenever the cell region comes into view.

The “agent” feature graph is slightly more complex. It has one feature, with three LODs: alibi-less, alibi-ful, and nonexistent. As envisioned in the original design, transitions are only available from the first to the second, from the second to the third, and from the first to

the third LOD. in the alibi-less LOD, agents reaching portals whose other cell is visible do not immediately generate alibis, but transition to a new segment randomly based on a simple per-segment probability table. The alibi-less LOD has a small but significant audacity for unrealistic long-term behavior, causing its unrealism to increase over time while the agent is visible. Transitioning from the alibi-less to the alibi-ful LOD, of course, has a high CPU cost; the alibi-ful LOD has a slightly higher CPU and memory cost than the alibi-less LOD. The alibi-ful LOD has a zero audacity vector, reflecting its status as the “realistic” LOD. The transition from alibi-less or alibi-ful to nonexistent LOD has a high audacity coefficient for unrealistic state and a medium coefficient for fundamental discontinuity; coupled with the zero resource cost for the nonexistent LOD, this will cause un-memorable agents to be destroyed when out of view for a little while, but will prolong the lives of agents which have been the source of attention recently.

In addition, a transition may be added from the alibi-ful to the alibi-less LOD, with zero transition cost or penalty (the additional penalty coming from the destination level). This allows agents which were given alibis and subsequently unattended to discard those alibis for a small gain in resources.

5.8.2 Cells as triggers

In this form, two crowd simulations are used: the normal discrete agent simulation, and an overlaid flow-based simulation. Each cell will have a set of currently present discrete agents, and a real-valued continuous population.

At the beginning of the simulation, all cells have zero discrete population, and each has a continuous population equal to its average population. When a cell becomes visible following a period of invisibility (or is visible at the beginning of the simulation), as many agents are generated as the integral portion of the continuous population, with that population being correspondingly reduced. This transition only occurs at the moment when the cell

becomes visible.

Portals operate in a complementary manner. When both cells of a portal are invisible, it transfers continuous population in each direction with a flux proportional to the average flux, and to the incoming cell's continuous population. When one cell of a portal is visible and the other is invisible, continuous population flows bidirectionally, but flux from the invisible cell to the visible cell does not increase the visible cell's continuous population. Instead, a Poisson process generates discrete agents at the portal heading into the visible cell at the corresponding average rate.

Agents' feature graph is similar to the previous form, with the added caveat that agents are not allowed to become nonexistent while in a visible cell (even if they themselves are not visible). When an agent does become nonexistent, its current cell's continuous population is incremented by 1. Sufficiently memorable agents are thus allowed to continue operating in invisible cells, while less memorable agents are absorbed into the continuum.

5.8.3 Discussion

Both forms are reasonably straightforward to implement. The first form has a definite performance edge, allowing the instant destruction of swaths of agents when cells are sufficiently out of view. The second form, however, allows for arbitrary degrees of compromise between resource usage and realistic persistence. Moreover, it guarantees conservation of population, even in circumstances where the viewer's actions change the normal flow of traffic.

Implementing the decision to create, destroy, and alibi-ize agents in terms of these benefit/cost tradeoffs removes some of the deterministic crispness of the original design, and compromises the elegant guarantee that alibi-less behavior can never be noticed. In particular, the benefit of giving an agent an alibi is calculated based on the duration of observation, rather than the number of inter-segment turns made. If desired, this latter criterion can be substituted, by simply removing the transition from alibi-less to alibi-ful

from the feature graph, and instead manually forcing it when the agent nears the end of the segment.

The benefit, however, is an effective solution to the concerns described in section 5.7. Agents are no longer destroyed needlessly or carelessly: the decision to destroy one is made only when the resources are needed, and the agents chosen for destruction are those the viewer is least likely to miss. Nor are alibis generated for agents almost immediately after creation, since the penalty for alibi-less simulation only becomes significant after a period of actual visibility (of the entity, not the cell). Finally, these LOD decisions may be coordinated with those of other simulation types, to provide better guarantees on total system performance.

5.9 Beyond pedestrians

In chapter 4, we discussed how transitions present considerably more of a challenge for simulation LOD systems than they do for graphical LOD systems. Rendering is an essentially stateless, side-effect-less process of transforming input into output; the LOD choices made on the previous frame do not affect the rendering process in the next frame. Moreover, different graphical LODs, while differing in operation, take as input the same data about a scene's contents. At a high level, a graphical LOD system may be viewed as a collection of functions which transform scene information into rasterized graphical data, all operating on the same domain.

A simulation LOD, in contrast, consists of both data processing and data description. All simulation LODs for a particular simulation need will generally output the same data – however crowd simulation is done, the result must be positions for agents – but additional internal state will differ between LODs. This state may be an implementation detail of the LOD, but more often it is a reflection of the particular rules being followed: A physically-

based locomotion LOD *must* keep track of individual limb velocities, and a playback-based locomotion LOD has no need to.

When transitioning between simulation LODs with different state spaces, a mapping process must take place. Consider the case of the locomotion LODs above. Transitioning from physics to playback is trivial: limb velocities are no longer needed, and are discarded. Transitioning from playback to physics is slightly more involved, as initial limb velocity state must be generated through finite differencing.

Note the potential for discontinuous limb velocities when transitioning from physics to playback. This occurs because the state mapping is overconstrained: The physics state space has much higher dimensionality than the animation state space, and so the closest animation state to a given physics state may induce significantly different output. The state mapping from playback to physics, in contrast, is not overconstrained, as any animation state may be transformed through finite differencing into an equivalent physics state. Neither is it underconstrained: There is exactly one “correct” set of limb velocities for a given animation state, which should always be generated when transitioning from that animation state.

So far, so good. But there are situations in which the state mapping involved in an LOD transition *is* underconstrained. Transitioning a vehicle from animation to physical simulation, for instance, might involve the generation of a fuel tank level. There is no single “correct” fuel level, as nothing about the previous simulation unambiguously implied a specific level. But there are “incorrect” fuel levels. If the vehicle was in the process of pulling into a gas station when the transition occurred, it would be silly for the fuel tank to be full; if it was pulling away from the gas station, it would be silly for the fuel tank to be empty.

Not *fatally* silly, of course. We could come up with plausible excuses for each of these: The first driver was pulling into the gas station for a snack, not a fill-up, for example. But a viewer’s suspension of disbelief has its limits. In the long term, they expect cars to generally

leave gas stations with more fuel than they came in with. But they also expect a certain amount of variety: if every car pulling into a gas station had a fuel level of precisely one eighth, realism would suffer as well. These underconstrained mappings, in other words, map from a determined entity state to a probability distribution of possible states. And this determined entity state is not just immediate simulation state, but encompasses the viewer's history of observations of that entity.

Alibi generation fills precisely this niche: the generation of plausible data to solve an underconstrained state mapping. Its applicability extends beyond goal generation for pedestrians. For the fuel tank example above, simple rules governing when people stop at gas stations and what happens to their fuel level when they do can be used to derive a conditional distribution over fuel level given observed activity, and this distribution can be sampled to generate fuel levels for vehicles being transitioned to simulation LODs which require them.

Please take a minute to tell us what happened in as much detail as possible.

The Unity3D crash reporter

Chapter 6

The Marketplace Project

6.1 Overview

The Marketplace project is a use case scenario for comprehensive human agent animation and modeling. As part of this project, we have developed a realtime perceptually-driven simulation of a large Middle Eastern public marketplace.¹ In the simulation, virtual humans travel around the marketplace, interacting with objects and other humans. The humans are intended to appear intelligent and rational in their behaviors, and the marketplace as a whole is intended to feel crowded and “alive”.

Perceptually-driven simulation techniques are used to maintain real-time performance on commodity computer hardware while maximizing realism. In particular, an implementation of the LOD Trader controls many features of the virtual humans’ simulation. In this chapter, we describe the design of the marketplace simulation. We then enumerate the features which are subject to LOD control and the various levels of each, including the computational costs and perceptual penalties involved. We describe the process of tuning the estimation of

¹While the influence of cultural factors on human behavior is an important component of the project as a whole, this simulation does not attempt to faithfully reproduce a particular culture; it draws indiscriminately on Arab, Afghan, and even Mediterranean cultural influences.

these costs and penalties, as well as the process of tuning the parameters of the perceptual criticality model. We describe informal tests of the simulation’s performance and realism, and their results. Finally, we discuss the consequences of the LOD trader design and operation on the simulation.

6.2 The simulation

6.2.1 Architecture

The simulation is implemented in the Unity3D game engine, and targets an Intel Core i7-based PC running Microsoft Windows 7. Most of the system is programmed in C#, with computationally intensive routines rewritten in C++ as necessary.

6.2.2 Graphics

Human models are generated from an off-the-shelf collection of body part and clothing meshes, with simple constraint rules to prevent incompatible clothing and visible interpenetration during animation. Face and clothing textures are randomly chosen from a small database to maximize the appearance of variety. In all, 115200 unique male and 2268 unique female meshes are possible. World geometry is made up of low-polygon objects, procedurally assembled into stalls and product displays through a set of simple rules. Static occlusion culling is used to reduce rendering costs.

A birds-eye view of the Marketplace environment is shown in figure 6.1.

6.2.3 Behavior

High-level behavior is based on “smart objects” similar to the AI system in *The Sims* [Viega, 2001], with dynamic utility weightings used to guide goal selection. A person will have a



Figure 6.1: A birds-eye view of the Marketplace simulation environment.

vector of utility coefficients for various events such as “Buy fruit”, “Have fun”, and “Spend money”. A utility coefficient may be positive to indicate the person’s desire for the event to occur, negative to indicate aversion to the event, or zero to indicate indifference. The simulation world is outfitted with “affordance sites”, each one with a description of an action a person can take and a corresponding vector of values describing the presence and degree of those events which would result from taking the action. For instance, an apple stall would have a “Buy apples” affordance with positive “Buy fruit” and “spend money” values. The affordance also describes a transformation on utility coefficients which occurs when the person performs the action, consisting of a scaling followed by an addition for each coefficient. The person herself also has a set of utility transformations which are performed after each action. A view of the Marketplace environment showing the available affordance sites and the extents of their offer volumes is shown in figure 6.2.

This simple system allows a wide range of behavioral effects. For instance, each person starts (assuming full quality simulation) with positive utility weightings for a subset of the “Buy X” events, which serves as a shopping list: the affordances which provide these events have transformations which scale the corresponding utility by 0, causing a person to not re-buy a product she has already bought. As another example, a person’s personal utility transformation subtracts a small amount from the person’s “spend time” utility. Most affordances have a small positive “spend time” value, with the exception of the “leave the marketplace” affordance, which has a zero “spend time” value and negative values for all “Buy X” events. Over time, this causes people to leave the marketplace after awhile, particularly once they have completed all of their errands.

The existence of people, and their state, is driven by the alibi generation approach discussed in section 5.8.2. Rather than symbolically derive the alibi distribution for affordance-driven agents, we simply ran the full-quality simulation until we had gathered several hundred alibi samples for each affordance site, and we only generate alibis when agents are

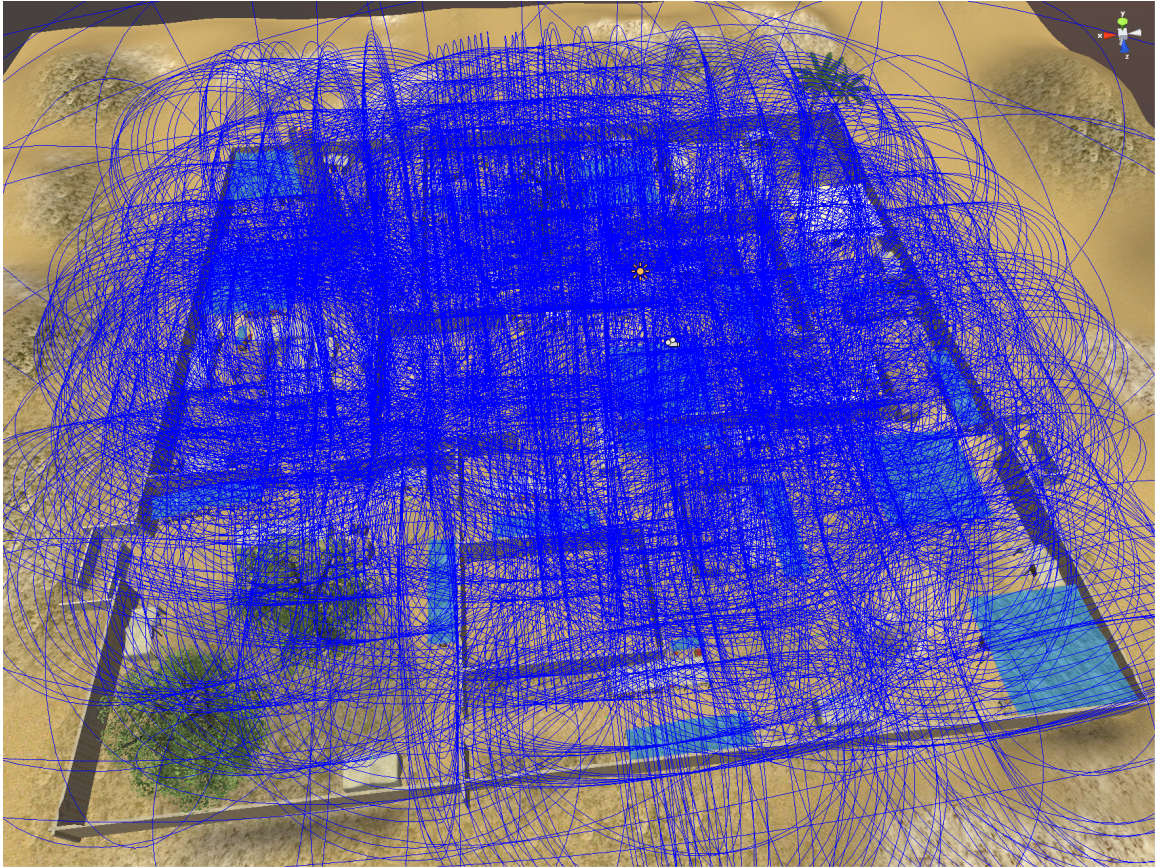


Figure 6.2: Affordance sites, and the volume within which they are offered to agents, in the Marketplace simulation environment.

at affordance sites. The cells used for world visibility are shown in figure 6.3.

6.2.4 Navigation

Pathfinding is performed by building a navmesh (a 2-dimensional convex polygonal tessellation of walkable areas), performing an A* search to identify a path between polygons, and refining the path with a visibility-based string-pulling algorithm. Navigation along the path is controlled by a variant of the Reciprocal Velocity Obstacles algorithm [Van den Berg et al., 2008]. The publicly available Recast and Detour libraries are responsible for these steps.

6.2.5 Animation

The animation of human characters is split into locomotion animation and action animation. Locomotion animation is performed either with an off-the-shelf data-driven IK system which blends different walk cycle animations to match the character's local velocity, or by simple time-scaled animation playback. Action animation is based on playback, and may include IK retargeting.

6.3 LOD control

As detailed in chapter 4, the LOD-varying features inherent to an entity are described by a *feature graph*. This graph consists of a set of features, a set of LOD choices for each feature, and a set of features which is required for each LOD choice. Virtual people in the Marketplace simulation have seven features, which are described in turn.



Figure 6.3: Cells used for visibility testing and agent creation in the Marketplace simulation.

6.3.1 EXISTENCE

This is the root feature of the graph, and is thus the only feature which must be present in a feature solution (it therefore has no *null* level). It has two levels: *Yes* and *No*. The only allowed transition is from *Yes* to *No*, which causes the person to be deleted from the simulation. The *No* level, of course, has no required features; the *Yes* level requires the BEHAVIOR feature.

Costs and benefits for the EXISTENCE levels are a little idiosyncratic. Costs for the *Yes* level are small, and represent primarily the bookkeeping overhead involved in maintaining the person in the simulation at all. This includes the expected cost of processing by the LOD trader itself. Costs for the *No* level, of course, are zero. It bears mentioning that although the *Yes* level has only small costs, a transition from a feature solution including the *Yes* level to one including the *No* level may represent a large total reduction in costs because the *No* level has no required features and thus any such transition will realize the cost reduction from moving all other features to the *Null* level.

As it is the highest level for the feature, the penalties for the *Yes* level are zero. The penalties for the *No* level, perhaps counterintuitively, are also zero, as a nonexistent person cannot be *a priori* unrealistic purely as a result of its ongoing unrealism. However, the transition from the *Yes* level to the *No* level incurs very high Unrealistic State (US) and Fundamental Discontinuity (FD) penalties, representing the patent unrealism of a person either winking out of existence in front of the viewer or mysteriously vanishing while off-screen.

6.3.2 BEHAVIOR

This feature controls the basic behavioral strategy of the person, and is required by all existent people. It has three levels: *Interact*, *Walk*, and *Goal Driven*. The *Goal Driven*

behavior is the highest level, at which the person repeatedly identifies a useful goal to perform, walks to a place where that goal may be accomplished, and performs an action which accomplishes the goal, which may affect its own state and the state of the world. It may interrupt its behavior cycle to perform incidental actions which become available (described later). *Walk* and *Interact* are simple, non-goal-driven behaviors which are used to provide background texture to the simulation. At the *Walk* level, the person continuously walks between randomly chosen sites in the marketplace, never performing any actions at those sites, and choosing a new site two meters before they get to their current destination. At the *Interact* level, the person stands in a single location and interacts with an object or person at that location, looping the interaction endlessly without affecting itself or the world.

The costs of all levels are small, as most costs are subsumed by their required features. As the highest level, the penalties for the *Goal Driven* level are zero. The *Interact* and *Walk* levels each incur a small Unrealistic Long-Term Behavior (ULTB) penalty, from the unrealism of continuously acting without accomplishing anything. Additionally, the *Walk* level incurs a small US penalty from the potential for the person to reach a random point and then walk towards a random point in a very different direction, in an apparently unmotivated U-turn.

Transitions are allowed from either *Interact* or *Walk* to *Goal Driven*, or from *Goal Driven* to *Interact* or *Walk*, but not between *Interact* and *Walk*. Transitioning from *Walk* to *Interact* is not allowed: it would require the person to be in a position where interaction could occur, but because the person is walking in a random place this cannot be guaranteed. Transitioning from *Interact* to *Walk* would be possible, but is not necessary to implement, as its functionality is replicated by a transition to the *Goal Driven* level with random goal-picking and no incidental actions (discussed in sections 6.3.3 and 6.3.4, respectively). The costs and penalties for all transitions are zero.

Four features are required by the *Goal Driven* level: NAVIGATION, INCIDENTAL AC-

TIONS, ACTION ANIMATIONS, and GOAL PICKING. The *Interact* level requires only ACTION ANIMATIONS, and the *Walk* level requires only NAVIGATION.

6.3.3 GOAL PICKING

This feature controls the algorithm which goal-driven people use to choose a site, and is required by people with *Goal Driven* behavior. It has three levels: *Uniform Random*, *Weighted Random*, and *Affordances*. At the *Uniform Random* level, a random site is uniformly chosen from the entire simulation. At the *Weighted Random* level, a person's next site is chosen from a distribution conditioned on the person's previous site. At the *Affordances* level, all sites within range are evaluated as described in section 6.2.3.

The first level has a small FD penalty representing the possibility for manifestly unrealistic successive sites, and both the first and the second level have small ULTB penalties. The costs of the *Uniform Random* level are essentially zero, and the costs of the *Weighted Random* level are low as well. The CPU cost of the *Affordances* level is higher, and there is additionally a small memory cost to maintain the utility vector. Transitions to the *Affordances* level require a utility vector to be generated through alibi generation, based on a pre-sampled distribution attached to the person's current affordance site, and so incur a significant CPU cost. If the person is not currently at a site, alibi generation is deferred until they reach their destination.

6.3.4 INCIDENTAL ACTIONS

This feature controls whether a goal-driven person monitors the area around her for, and responds to, opportunities for incidental actions; it is required by people with *Goal Driven* and *Walk* behaviors. These include actions like talking to a friend or giving money to a beggar. Like goals, these may be considered based on their affordances and the person's

current utility weightings; if the person does not have use *Affordances* for GOAL PICKING, they are instead invoked randomly. The two levels are *No* and *Yes*, and are self-explanatory. The *No* level has a small ULTB penalty representing the unrealism of a person *never* performing incidental actions (rather than passing up a single incidental action). The *Yes* level has a small CPU cost.

The INCIDENTAL ACTIONS feature is unique in that it is required by certain levels of two different features.

6.3.5 NAVIGATION

This feature controls both the process by which a person determines a shortest path from one point to another and the process by which it navigates along that path. It is required by people with a BEHAVIOR level which involves walking. It has three levels: *Pathing Only*, *Collision Resolution*, and *Avoidance*. At the *Pathing Only* level, a person finds a shortest path to the desired end point, then moves along that path without attempting to avoid other people or reacting to interpenetrations. At the *Collision Resolution* level, a person checks for and attempts to resolve interpenetrations with other people but does not proactively avoid them. At the *Avoidance* level, a person uses the Reciprocal Velocity Obstacles algorithm to predict and avoid interpenetrations with other people. Path finding requests are serviced at a maximum per-frame rate, with higher LOD requests prioritized over lower LOD requests; hence, higher LODs additionally reduce latency by a small amount.

The *Pathing Only* level has a small but nonzero CPU cost and a high US penalty (due to the potential for interpenetrations). The *Collision Resolution* level has a higher CPU cost and a small US penalty, due to its potential to collide unrealistically with other people. The *Avoidance* level has the highest CPU cost, and a zero penalty. All transitions are allowed, and there are no transition costs or penalties.

The NAVIGATION feature is nearly unique in that it governs interactions between people,

rather than individual behavior. The *Collision Resolution* and *Avoidance* levels cause the person to be registered within an occupancy grid; this grid is then used to determine potentially colliding neighbors. At the *Pathing Only* level, this registration does not occur. As a result, *Pathing Only* people are invisible to *Collision Resolution* and *Avoidance* people, causing their effective behavior (and unrealism) to degrade to that of *Pathing Only*. In practice, however, this is not an issue: Because the penalty multiplier for the lower levels emphasizes unrealistic state, the inputs to which are mostly spatially coherent, people which could potentially interact will tend to have the same NAVIGATION level.

Another unique aspect of the NAVIGATION feature is its sensitivity to population density. If all people in an area are simulated with *Pathing Only*, the frequency of interpenetrations experienced by a single person is proportional to the density of that area. As a result, the penalty multiplier for the NAVIGATION feature assumes a particular ambient population density, and becomes inaccurate as the density diverges from that estimate. It would be possible to address this issue by duplicating the US penalty factor into two factors, one including population density as a factor, but this would require the new factor to be considered in all penalty calculations.

6.3.6 LOCOMOTION

This feature controls the generation of walking animation for a person. It is required by people with a BEHAVIOR level which involves walking. It has three levels: *None*, *Playback*, and *IK*. At the *None* level, skinned animation is disabled entirely while walking; the character's limbs are frozen in place and it glides along its path. At the *Playback* level, a single forward walk animation is played in a loop, time-warped to match the person's speed. At the *IK* level, full IK-based locomotion is performed, giving realistic walking animations at any speed and along any trajectory.

The CPU cost of the *None* level is very low but nonzero, representing the computational

cost of batching the static geometry. The CPU cost of the *Playback* level is higher; the CPU cost of the *IK* level is higher still. Likewise, the *None* level has a very high US penalty, the *Playback* level has a medium US penalty, and the *IK* level has a zero penalty. Transitions are allowed between all levels, and there are no transition costs or penalties.

6.3.7 ACTION ANIMATIONS

This feature controls the playback of animations for people performing actions. It is required by people which perform actions, either due to a *Goal Driven* or *Interact* behavior, or a *Walk* behavior with INCIDENTAL ACTIONS enabled. It has three levels: *None*, *Low Quality*, and *High Quality*. These levels are roughly equivalent to the corresponding LOCOMOTION levels: *None* freezes the animation, *Low Quality* plays a fixed animation, and *High Quality* adapts the animation for style (driven by the person's mood) and uses IK to retarget end effector positions. Like the LOCOMOTION feature, these features differ in their CPU cost and their US penalty.

The ACTION ANIMATIONS feature and the LOCOMOTION feature are clearly similar: Both control the generation of skinned animation, and their levels roughly correspond to each other. It is tempting to combine them into a single feature to simplify the graph. However, leaving them separate allows their cost and penalties to be individually tweaked, the IK involved in navigation being considerably more expensive than that involved in action animations. Additionally, because the two features are required by different sets of LODs, leaving them separated allows the cost of doing any locomotion to be separated from the cost of doing any action animations, preventing the overstatement of the cost of feature solutions which only do one.

6.3.8 MESH RETENTION

This feature controls how a person's skinned mesh is retained in memory. There are three levels: *None*, *Invisible*, and *Visible*. At the *None* level, no mesh is assigned to the object; at the *Invisible* level a mesh is generated and assigned but not shown; and at the *Visible* level the mesh is shown. A transition to the *None* level, or to the null level, causes the mesh to be returned to a pool of available resources; thus a person who transitions from *Visible* to *None* then back to *Visible* will have a different appearance. The memory cost of the *Invisible* and *Visible* levels are high, and there is a large US penalty for the *Invisible* and *None* levels. All transitions are allowed; transitioning from *Visible* or *Invisible* to *None* or the null level introduces an FD penalty.

This feature is unique in that its level is directly constrained by other LODs. The feature as a whole is required for all existent people, but additionally LOCOMOTION and ACTION ANIMATIONS levels other than *None* require the *Visible* level specifically, as they operate on the mesh skeleton. This allows visible but unanimated people, but does not allow animated people to be invisible. If not for this constraint, the system could theoretically choose to spend resources on animating an invisible person due to the lower total realism penalty relative to that of an invisible unanimated person. This constraint sidesteps that weakness in the additive model of unrealism.

6.4 Tuning

6.4.1 Measuring costs

The process of quantifying the costs of different feature LODs was relatively straightforward. A large number of people were spawned in the simulation world at a particular feature solution, the simulation was run for a set period of time while the viewer was moved along

a pre-recorded path through the scene, and the average and peak frame times and memory heap usages were recorded. By re-running this benchmark with different feature solutions, the resource usages of individual LOD features could be accurately estimated.

The resource overhead of the world itself was also measured. Different viewpoints within the scene could have an order of magnitude more or fewer objects visible, resulting in a high range of overhead costs. To account for this, a second automated measurement procedure was used. The world was divided into a 2 dimensional grid with a resolution of 2 meters. For each grid point at each of 8 evenly spaced view directions, the average time to render a frame was measured and stored. At runtime, trilinear interpolation was used to estimate the overhead at the viewer's current position and view direction. This was subtracted from the total resource usage targets each time the LOD trader ran, to update the maximum resources apportioned among people by the LOD trader.

The ratios between available resources were also tracked. We observed extremal ratios of up to approximately 100 available kilobytes of RAM per available microsecond of CPU time, and up to approximately 0.15 available microseconds of CPU time per available kilobyte of RAM. These extrema were fed back to the transition culling process, slackened by a safety factor of two, to serve as constraints on subsequent runs.

6.4.2 Tuning the criticality model

The process of tuning the various factors of the perceptual realism criticality model was less straightforward, and required different techniques for each factor.

Observability

Observability was the second easiest factor to tune. At a distance of closer than four meters it was found to be easy to distinguish between people performing IK-based and pre-animated locomotion, the subtlest of the unrealistic state-based sources of unrealism. We therefore

chose the solid angle subtended by a person at a distance of four meters as the saturation angle. This is the only parameter of the observability model.

Attention

Attention was considerably more difficult to tune, because of its extreme subjectivity. For instance, the value for the ambient attentional load parameter has a strong effect on the relative behavior of the LOD trader in crowded versus sparse situations. At a high ambient attentional load, the attention estimate for individual people is mostly independent of crowding. At a low attentional load, crowded situations lead to a more precipitous lowering of per-person attention. When tuning this parameter, the knowledge that too high a value would lead to background people behaving unrealistically strongly primed us to be sensitive to this consequence. At one point, I greatly increased the ambient attentional load value to test the results, and anecdotally observed a sharp drop in realism for background people. It was only after the test that I realized I had not saved the new parameter settings: the drop in realism was entirely a creation of my subjective expectations.

Tuning the relative weighting of observability, focusing, and following factors revealed the third factor to be of little benefit. Unrealistic state BIRs did not seem more probable for entities which were being followed but not focused; other BIRs would naturally be subject to a higher criticality for followed entities because of the greater memory and duration factors. We found an observability weighting of 0.7 and a focusing weighting of 0.3 to predict attention well.

Memory

The two parameters of the memory model, α and β , respectively govern the rates of memorization and forgetting. The first was set to 0.6, equating to a rise in estimated memory from 0 to 95% of the estimated attention in approximately 5 seconds. The second was set

to 0.001, equating to an approximately 10-second half-life of memory under the highest observed attentional loads. These parameters prevented underestimation of memory in all but the most contrived scenarios, but did overestimate long-term retention of memory for most entities. This overestimation did not appear to lead to a significant waste of resources, as discussed in section 6.7.6.

Duration

The duration model had no parameters to tune. This made it the easiest model to tune.²

Return time

The parameters of the Weibull return model were directly estimated by recording actual return times over several hours of performing self-directed tasks within the simulation. A statistical analysis program was used to fit a Weibull distribution to this model, including censoring of data for entities which never returned to being visible. The best-fit shape parameter was $k \approx 0.8$, supporting the hypothesis that the hazard rate decreases over time. (The scale parameter of the distribution serves only as a constant factor in the expected reward model, and is thus unimportant to estimate independently of audacity factors.)

6.4.3 Tuning audacity

The process of tuning features' audacity vectors was ad-hoc and iterative, consisting primarily of adjusting audacities in response to consistently noticed BIRs. Rough quantitative reasoning was used to adjust relative audacities from different LODs for the same category of unrealism ("This one's about twice as unrealistic as that one"), but this was less useful for comparing different categories of unrealism because the criticality factors could not easily

²Zen koan: also the hardest.

be compared, particularly when the criticality model was re-tuned. This issue is discussed in section 6.7.1.

6.5 Alternative implementation

To evaluate the relative usefulness of the LOD trader, we implemented a simple proactive LOD control system based on the general approach taken by existing simulation LOD systems, such as that integrated into the game *Neverwinter Nights* [Brockington, 2002]. This system did not perform criticality modeling or respect resource constraints, instead picking LODs based only on viewer distance and visibility. The threshold distances for LOD transitions were tuned to maximize quality at a particular framerate quality, as described in section 6.6.1; separate thresholds were used for each feature.

6.6 Evaluation

The LOD trader was evaluated independently, and relative to proactive LOD control, in terms of several quality metrics. The ability of the two systems to respect resource constraints, and the resultant effect on performance, was objectively measured. The ambient visual realism of the simulation under the two LOD systems was also subjectively evaluated, with the viewer wandering the simulation and engaging in self-directed tasks.

6.6.1 Performance testing

Each of the two systems was benchmarked with a prerecorded 60-second flythrough of the scene. The time to simulate and render each frame was recorded. The LOD trader was set to a target framerate of 30 FPS (0.033 seconds per frame); the proactive LOD selector was tuned (by means of adjusting its threshold radii) to maintain an average framerate of 30.

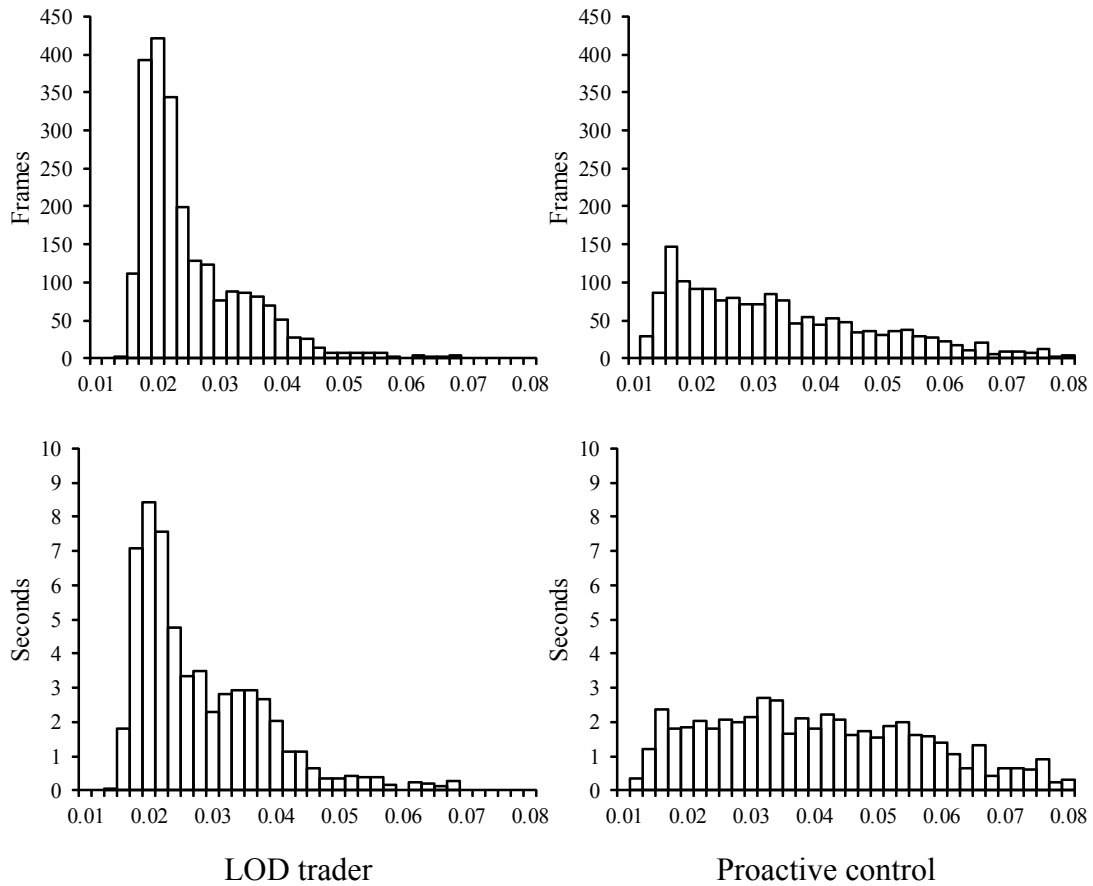


Figure 6.4: Per-frame execution time during a 60-second flythrough of the Marketplace simulation, as a proportion of total frames rendered and as a proportion of total simulation time.

Frames during which a garbage collection occurred were not considered. Figure 6.4 shows the distribution of frame times for each system, as a proportion of total frames rendered and as a proportion of total simulation time.

Although the mean frame time of the two systems was nearly identical, the standard deviation of proactive LOD selection was nearly double that of the LOD trader: 19 milliseconds versus 10 milliseconds. The LOD trader maintained a framerate of at least 25 FPS during 80% of the execution time; the proactive LOD selector maintained this framerate

during only 50% of the execution time.

The LOD trader itself had an execution time of 57 microseconds per frame, or 0.17% of the target frame time. Its memory usage was approximately 500 kilobytes for the feature graph and transition data, plus 48 bytes per entity; this usage could easily be reduced by half by picking narrower datatypes, with no reduction in functionality.

6.6.2 Perceptual study

A perceptual study was undertaken to determine whether the LOD trader produced lower BIR rates than proactive LOD selection. Thirty-second real-time video captures were made of the simulation from a fixed viewpoint, with LOD selection performed either by the proactive selector or by the LOD trader, and with a population of 1, 3, or 5 times the base rate, resulting in six videos. Each was encoded to FLV format at a resolution of 640 by 480 pixels at 30 frames per second (The simulation was capped to 30 frames per second but was allowed to skip frames.) A screenshot from one of these videos (LOD trader, population rate 5) is shown in figure 6.5.

Subjects were recruited through the Mechanical Turk web service. Each subject was shown a randomly chosen video once. Subjects were instructed not to pause or replay the video. Afterwards, the subject was given a list of potential problems with the video. Each problem described either an underlying problem with the simulation which was independent of LOD selection (e.g. “Unrealistic head motion”), a problem which was present at low LODs but not high LODs (e.g. “Feet sliding along the ground”, “Sudden change in direction”), or a problem which did not exist (e.g. “Sunglasses changed from black to white”). For each problem, subjects were asked to score on a five-point Likert scale whether the problem was perceived in the video (1 for “Definitely not”, 5 for “Definitely”), and how often the problem occurred (1 for “Never”, 5 for “Constantly”). Subjects were voluntarily allowed to view other videos, up to all six. Each video was viewed 20 times,



Figure 6.5: A frame from one of the videos shown to participants in the perceptual study of the Marketplace simulation.

resulting in 120 data points from 46 participants. In order to filter out participants who did not actively attend to the video, all responses from participants who answered “4” or “5” when asked whether the problems that did not exist in the video occurred were discarded (3 participants, representing 10 data points).

Results of the survey questions for problems present at low LODs but not high LODs are given in table 6.1. No significant difference in incidence of direction change perception between proactive selection and the LOD trader was observed at population rates of 1 or 3 ($p > 0.05$); Incidence of direction change perception was significantly lower ($p < 0.05$) with the LOD trader than with proactive selection at population rate 5. Incidence of foot skate perception, and frequency of foot skate and direction change were all significantly lower ($p < 0.05$) with the LOD trader than with proactive selection at all population rates. These results support the hypothesis that the LOD trader is effective at reducing the incidence and frequency of BIRs from low-LOD simulation.

6.6.3 Subjective evaluation

In addition to the frequent slowdowns caused by the proactive LOD selector – one area of the simulation was nearly unnavigable – its ability to maintain the perception of realism was middling at best. In order to maintain the required average framerate, it was necessary to decrease the threshold radius for IK-based locomotion to 5 meters, a distance at which foot skate was easily perceptible, particularly in sparsely populated areas. It was also necessary to threshold existence at 50 meters for out-of-view people; at this radius, it was quite common for memorable people to perceptibly disappear.

The LOD trader maintained a good perception of realism under nearly all circumstances. Long sightlines in foreground-crowded areas seemed to pose a problem for the criticality model: background people could occasionally be seen to be simulating without animations while visible along these sightlines. In very crowded situations the weaknesses of the

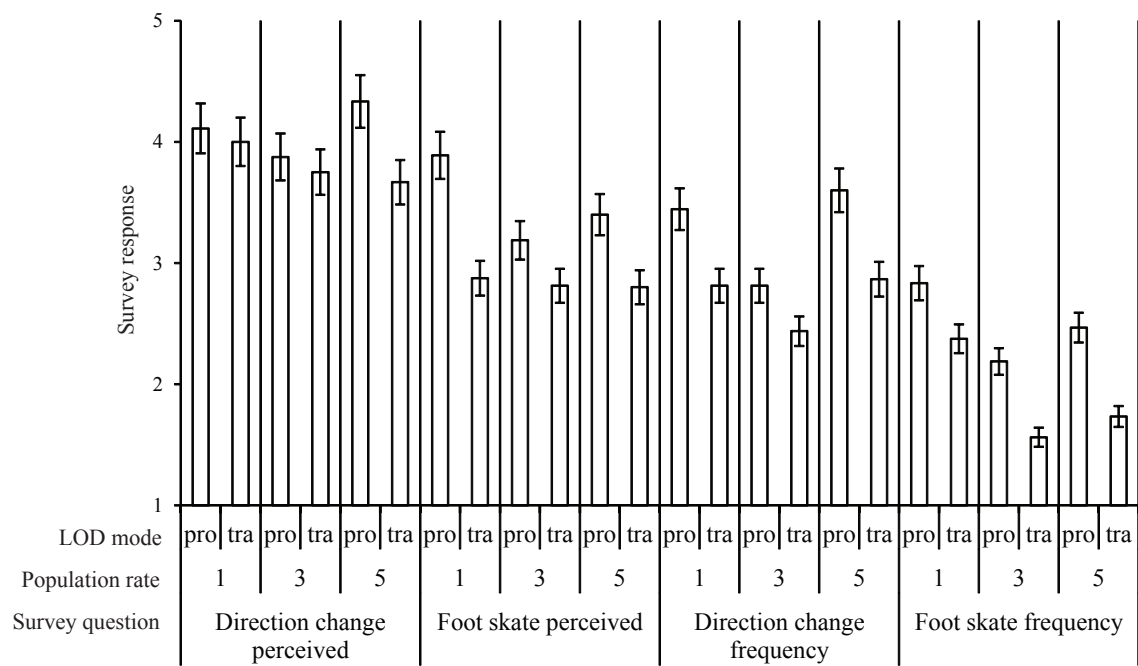


Table 6.1: Perceptual study responses, on a five-point Likert scale, for the incidence and frequency of BIRs from low LODs at different population rates and with either the LOD trader (“tra”) or proactive selection (“pro”) controlling LODs. Lower numbers are better. Error bars represent 95% confidence intervals.

Collision resolution navigation could also cause BIRs due to people vibrating in an attempt to push past each other. Other unrealistic situations were largely the result of weaknesses in the “ground truth” simulation.

6.6.4 Personal impressions

The LOD trader is by its nature deceptive to the viewer, and the surprising audacity of its decisions is matched by its surprising ability to get away with them. This made debugging difficult at times. For instance, it was nearly impossible to confirm that mesh pooling – which causes people to change their appearance, sometimes radically – was working properly. It was necessary to memorize people’s appearance during a short period of time, then look away for a few seconds, then look back and check that their appearance had changed. If a single person were close to the camera, or the same group of people was rendered for more than a few seconds, the memory model correctly ascertained that the people were now memorable to the viewer (i.e. the programmer), and refused to pool their meshes until they had been invisible for some time, at which point I really *had* forgotten what they looked like. It became necessary to give a small percentage of the people bright pink hats, merely to fool the criticality model. Of course, the LOD trader would have been more than capable of accounting for the additional audacity of changing bright pink hat state, had that been exposed in the feature graph.

6.7 Discussion

The implementation of the Marketplace simulation led to several lessons about the unrealism modeling methods chosen, and about the design of the LOD trader.

6.7.1 Cascaded tuning

As should be clear from section 6.4, many steps are involved in tuning the criticality model and realism penalties. This process is significantly complicated by the long chains of dependencies in tuning. Changing the saturation angle of the observability model, for instance, affects attention calculations, which in turn affects the memory and duration models, to different degrees. This has the effect of changing the relative weightings of FD and ULTB BIRs, invalidating any prior tuning of audacity factors in a feature graph. Any change to a parameter in the criticality model has this sort of a cascaded effect. A quiescent and dependable set of parameters for the criticality model could significantly reduce the effort needed to tune the system as a whole. In order for this to happen, the criticality models must be validated as providing acceptable accuracy over a range of types of simulations.

6.7.2 Transition costs and penalties

A weakness of the LOD trader lies in its inability to heuristically distinguish between temporary costs and penalties from transitions, and ongoing costs and penalties from LODs. For instance, consider a graph of one feature with two levels, the lower level having a cost of 0 and the higher level having a cost of 1. Suppose that the transition cost from the higher level to the lower level was 1.1. Under these circumstances, the total calculated cost of the downward feature solution transition would be positive, so this transition would never be taken. Put differently, the LOD trader is not able to trade short-term costs and penalties for long-term gain. A more insidious form of this relates to unrealism penalties: If there is a transition penalty for an upgrade transition which is greater than the realism benefit from the two levels in some coefficient, this transition should ideally be taken speculatively while criticality is low and so the transition would do little harm, but instead the transition is taken later, when criticality is high. This was the case with the initial transition penalties for the

MESH RETENTION feature from the *None* level to the *Visible* level: The transition itself had a large US penalty (due to the unrealism of “blinking into view”), while the *None* level had a small but significant ULTB penalty (related to an earlier design of the mesh pooling system). This resulted in people remaining invisible unless the viewer inadvertently followed them; after a short period of time the rising ULTB criticality would overcome the US penalty of the transition, and the person would pop into view exactly where they shouldn’t.

A potential solution to this issue is discussed in section 7.1.2.

6.7.3 Costs and visibility

Many features have ongoing costs which are not constant, but vary greatly depending on factors such as visibility and crowding. The LOD trader relies on constant costs in order to cull feature solution transitions, but could perform the culling based on a range of costs instead. This would require a model of costs for each such LOD to be created and tuned. In the Marketplace simulation, the MESH RETENTION feature was a prime candidate for this, as the cost of rendering a mesh was dependent in part on its size on the screen and needed to be estimated as an upper limit. While a non-constant cost model would have avoided this overestimation and thereby allowed upgrades elsewhere, it would not have affected the selection of the MESH RETENTION feature itself, as the observability and hence the US audacity vector would grow faster than the rendering cost.

6.7.4 Feature graph design

Cost accounting was a little tricky with the feature graph design: It was necessary to decide and remember which costs were handled by high-level features, and which were delegated to lower-level features, due to redundancy in the expressiveness of the graph. If a feature was required by exactly one LOD, for instance, increasing the cost of that LOD by a certain

amount was equivalent to increasing all the required feature's LODs by the same amount. While not a serious impediment, some care was necessary to avoid under- or over-counting costs, and diagnosing an inaccurate accounting was sometimes difficult.

6.7.5 Types of attention

The tuning of our attention model was in some ways a compromise between different types of attention. Certain unrealistic state BIRs, such as blinking in and out of existence, can easily happen in a viewer's peripheral vision; others, such as foot skating, are likely to go unnoticed unless the viewer is directly attending to them. This is not necessarily addressed sufficiently by tuning the audacity of these events, as the proper ratio may be different closer to the center of the viewer's vision. We felt that this did not present a serious enough problem in the Marketplace simulation to warrant complicating the attention model or bifurcating the US category of BIRs, but those options might be considered in other applications.

6.7.6 RAM and memory

The Marketplace simulation world was approximately the size of two football fields, with a population of up to 500. This makes it several orders of magnitude smaller than the environment in the Heisenburgh simulation. At the same time, it has considerably more variation than can be found in the entirety of Heisenburgh, with distinctly different-looking "rooms" and areas of activity. We felt that this was a worthwhile tradeoff, allowing us to concentrate on the minutia of human behavior rather than large-scale content creation. This had a major effect on LOD selection. With a population of only 500, RAM requirements were rarely taxed, even with wild overestimation of the viewer's memory for people. In a modern open-world video game, huge development team sizes allow for worlds which are both large and varied; under these circumstances, RAM would be a more valuable

commodity, and careful tuning of the memory model to avoid over-estimation would be more important.

Chapter 7

Conclusions

The prime advantage of current proactive simulation LOD control systems is simplicity – of implementation, of execution, and of tuning. This simplicity is not a quality to be undervalued. In problem areas such as LOD control, where the theoretical weaknesses of the “standard approach” obscure the soundness of the practical motivations underlying it, a more complex approach can appear self-evidently useful to its creators but not to its potential users. A technique such as reactive simulation LOD control faces high barriers to adoption into widespread practical use. It must convincingly defend itself on the following fronts:

- Ease of implementation. The system must be reasonably straightforward to implement, leveraging well-established data structures and algorithms where possible.
- Real-world efficiency. The system must be tested in conditions designed to mirror its intended real-world use, and be found to consume only a small proportion of system resources. While it would be possible to argue that a sufficiently effective LOD control system could use significant resources yet make up for it with more significant resource savings from the chosen LODs, ideally the resource utilization should be so low that it would not be seen as any sort of a tradeoff. Ideally, it should require under one percent of the processor time and memory space of its target hardware, and in

the case of NUMA architectures, be operable on a single processing element without heavy data transfer or synchronization penalties. Both average and peak resource utilization should be considered.

- Obvious practical advantages. The system must be compared to the standard approach, again in conditions designed to mirror its intended real-world use, and its performance and quality advantages must be unequivocal. It must provide far better quality when targeting similar performance, and far better performance when targeting similar quality.

We feel that the perceptually driven simulation system described in this dissertation succeeds on all these fronts.

For ease of implementation, the most conceptually complex element of the system is the mathematical model underlying alibi generation; but alibi generation, while a useful tool for simulation LOD, is not obligatory for such a system. The most complex data structure in the entire system is a priority queue, and the algorithms underlying the LOD trader are short and straightforward. As to real world efficiency, the processor and memory requirements of the system, even at peak, are well under one percent of those available in a commodity desktop PC or video game console. The only input to the LOD trader is visibility and distance information for updating the criticality model; the only output is LOD decisions. In the case of the Cell Broadband Engine processor used by the PlayStation 3, this makes the LOD trader a good candidate for offloading to an SPU, with entity visibility information sent by DMA from the graphics process and LOD decisions distributed to required systems by mailbox. Finally, the practical advantages of this reactive LOD control compared to standard proactive LOD control are clear: the lack of resource guarantees offered by proactive LOD control mean that the target framerate cannot be consistently maintained under any but the most austere quality settings. The LOD trader, in contrast, maintains an acceptable framerate

even when crowd size and rendering complexity swell, without compromising the average quality of the simulation.

7.1 Future directions

7.1.1 Black-box alibi generation

The complexity of alibi generation is largely in its analytic models of agent behavior, which even for the simplistic behavior model in Heisenburgh required the use of an exotic and little-known generalization of Markov processes, as well as several different probability distributions. For agents with somewhat more complex behavior, analytically based alibi generation will likely be intractable. Here, an empirically driven approach may be promising, with observations of the “full fidelity” simulation used to build approximate probability distributions for initial agent generation and alibi generation. This would be a Machine Learning approach to alibi generation, and the primary challenges would be to smooth the observed data to effectively cover the large sample space, and to compress the distributions into a reasonable space without losing important features.

7.1.2 Multi-horizon LOD planning

As discussed in section 6.7.2, a significant weakness of the LOD trader was the short-term nature of its decisions. A simple extension to the LOD trader which might mitigate this weakness would be to split each resource into multiple “futures” (again, in the financial sense), each one constraining that resource a particular number of frames hence. This would greatly complicate the calculation of costs and benefits, which would need to be estimated based on future criticality changes and Bellman’s principle of optimality. It is also unclear how the set of horizons would be chosen.

7.1.3 Better criticality modeling

Though motivated by experimentally supported knowledge of the underlying cognitive processes, the criticality models presented here are by no means the most accurate, precise, or broadly applicable models possible. A wealth of literature about human cognitive processes exists, nearly ready to be integrated into criticality models. (Admittedly, most of this knowledge is motivated by a desire to *increase* optimize human cognitive ability, rather than a desire to “lie to the viewer and get away with it”.) The categorization of BIRs and of their factors could also be reexamined.

7.1.4 Practical use

As discussed in section 6.7.6, the perceptually driven simulation techniques discussed in this dissertation have not really been thoroughly stress-tested yet. Doing so in an academic environment presents major practical problems, chiefly that of producing the huge amount of creative effort involved in the development of a simulation environment that is large, varied, interesting, memorable, and realistic. It is our hope that video game developers and others with a business need for large, realistic simulation environments will give these techniques the thorough vetting due them after the promise they have shown thus far.

Bibliography

- M. Akbar, E. Manning, G. Shoja, and S. Khan. Heuristic solutions for the multiple-choice multi-dimension knapsack problem. In V. Alexandrov, J. Dongarra, B. Juliano, R. Renner, and C. Tan, editors, *Computational Science - ICCS 2001*, volume 2074 of *Lecture Notes in Computer Science*, pages 659–668. Springer Berlin / Heidelberg, 2001.
- F. A. Al-Khayyal. Generalized bilinear programming: Part i. models, applications and linear programming relaxation. *European Journal of Operational Research*, 60(3):306–314, 1992.
- S. P. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
- S. Britt. Retroactive inhibition: a review of the literature. *Psychological Bulletin*, 32(6): 381–440, 1935. ISSN 0033-2909.
- M. Brockington. Level-of-detail AI for a large role-playing game. In *AI Game Programming Wisdom*, pages 419–425. Charles River Media, 2002.
- D. C. Brogan and J. K. Hodgins. Simulation level of detail for multiagent control. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, AAMAS '02, pages 199–206, New York, NY, USA, 2002. ACM. ISBN 1-58113-480-0. doi: <http://doi.acm.org/10.1145/544741.544789>.

- C. Brom, O. Šerý, and T. Poch. Simulation level of detail for virtual humans. In *Proc. IVA*, 2007.
- D. Carlson and J. Hodgins. Simulation levels of detail for real-time animation. In *Proc. Graphics Interface 1997*, 1997.
- S. Cheney. Simulation level-of-detail. In *GDC 2001*, 2001.
- J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Commun. ACM*, 19:547–554, October 1976. ISSN 0001-0782. doi: <http://doi.acm.org/10.1145/360349.360354>.
- G. DeBunne, M. Desbrun, M.-P. Cani, and A. H. Barr. Dynamic real-time deformations using space & time adaptive sampling. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 31–36, New York, NY, USA, 2001. ACM. ISBN 1-58113-374-X. doi: <http://doi.acm.org/10.1145/383259.383262>.
- L. Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, 1986.
- J. Dingliana and C. O'Sullivan. Graceful degradation of collision handling in physically based animation. In *Computer Graphics Forum*, volume 19, pages 239–248. Wiley Online Library, 2000.
- S. Douglass, J. Ball, and S. Rodgers. Large declarative memories in act-r. In *Proceedings of the 9th International Conference of Cognitive Modeling*, 2009.
- J. Eich. A composite holographic associative recall model. *Psychological Review*, 89(6): 627–661, 1982.
- T. A. Funkhouser and C. H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of the*

- 20th annual conference on Computer graphics and interactive techniques, SIGGRAPH '93*, pages 247–254, New York, NY, USA, 1993. ACM. ISBN 0-89791-601-8. doi: <http://doi.acm.org/10.1145/166117.166149>.
- T. Gagic. Compressing probability distributions. *Information Processing Letters*, 97(4): 133–137, 2006.
- M. Garau, M. Slater, V. Vinayagamoorthy, A. Brogni, A. Steed, and M. Sasse. The impact of avatar realism and eye gaze control on perceived quality of communication in a shared immersive virtual environment. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 529–536. ACM, 2003. ISBN 1581136307.
- M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. ISBN 0716710455.
- A. Garvey and V. Lesser. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, 23(6):1491 –1502, nov 1993. ISSN 0018-9472. doi: 10.1109/21.257749.
- J. Granieri, J. Crabtree, and N. Badler. Production and playback of human figure motion for 3D virtual environments. In *Virtual Reality Annual International Symposium*, page 127. Published by the IEEE Computer Society, 1995.
- E. Gu, C. Stocker, and N. Badler. Do you see what eyes see? implementing inattentional blindness. In *Intelligent Virtual Agents*, pages 178–190. Springer, 2005.
- W. Hastings. Monte Carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.
- L. E. Hitchner and M. W. McGreevy. Methods for user-based reduction of model com-

- plexity for virtual planetary exploration. In J. P. Allebach and B. E. Rogowitz, editors, *Proceedings of SPIE*, volume 1913, pages 622–636. SPIE, 1993. doi: 10.1117/12.152736.
- R. Holloway. Viper: A quasi-real-time virtual-worlds application. Technical report, University of North Carolina at Chapel Hill, 1991.
- D. Jan and D. R. Traum. Dialog simulation for background characters. In *Proc. IVA*, pages 65–74, 2005.
- K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Carnegie-Mellon University, 1990.
- H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.
- S. Khan. *Quality adaptation in a multisession multimedia system: model, algorithms, and architecture*. PhD thesis, University of Victoria, 1998.
- S. Khan, K. Li, E. Manning, and M. Akbar. Solving the knapsack problem for adaptive multimedia systems. *Stud. Inform. Univ.*, 2(1):157–178, 2002.
- F. Kistler, M. Wißner, and E. André. Level of detail based behavior control for virtual characters. In *Intelligent Virtual Agents*, pages 118–124. Springer, 2010.
- S. Koenig, M. Likhachev, and X. Sun. Speeding up moving-target search. In *Proc. AAMAS 2007*, 2007.
- S. LaValle. *Planning algorithms*. Cambridge University Press, 2006. ISBN 9780521862059.
- K. Lee and P. Fishwick. Generation of multimodels and selection of the optimal model for real-time simulation. *Enabling technology for simulation science II, Proc SPIE*, 3369: 164–175, 1998.

- M. Lombard and T. Ditton. At the heart of it all: The concept of presence. *Journal of Computer-Mediated Communication*, 3(2), 1997. ISSN 1083-6101.
- D. P. Luebke. *Level of detail for 3D graphics*. Mor, 2003.
- B. MacNamee, S. Dobbyn, P. Cunningham, and C. O’Sullivan. Men behaving appropriately: Integrating the role passing technique into the ALOHA system. In *Proc. AISB 2002*, pages 59–62, 2002.
- S. Musse, M. Kallmann, and D. Thalmann. Level of autonomy for virtual human agents. In D. Floreano, J.-D. Nicoud, and F. Mondada, editors, *Advances in Artificial Life*, volume 1674 of *Lecture Notes in Computer Science*, pages 345–349. Springer Berlin / Heidelberg, 1999.
- R. Narain, A. Golas, S. Curtis, and M. Lin. Aggregate dynamics for dense crowd simulation. In *Proc. SIGGRAPH Asia*, 2009.
- C. Niederberger and M. Gross. Level-of-detail for cognitive real-time characters. *The Visual Computer*, 21(3):188–202, 2005.
- D. Osborne and P. Dickinson. Improving games AI performance using grouped hierarchical level of detail. In *AISB Symposium on AI and Games*. SSAISB, 2010.
- C. O’Sullivan and J. Dingliana. Collisions and perception. *ACM Trans. Graph.*, 20:151–168, July 2001. ISSN 0730-0301. doi: <http://doi.acm.org/10.1145/501786.501788>.
- C. O’Sullivan, J. Cassell, H. Vilhjalmsón, J. Dingliana, S. Dobbyn, B. McNamee, C. Peters, and T. Giang. Levels of detail for crowds and groups. In *Computer Graphics Forum*, volume 21, pages 733–742, 2002.

- C. O’Sullivan, J. Dingliana, T. Giang, and M. K. Kaiser. Evaluating the visual fidelity of physically based animations. In *Proc. SIGGRAPH*, pages 527–536, 2003. ISBN 1-58113-709-5. doi: <http://doi.acm.org/10.1145/1201775.882303>.
- N. Pelechano, J. Allbeck, and N. Badler. *Virtual Crowds: Methods, Simulation, and Control*. Morgan & Claypool Publishers, 2008.
- J. Pessin. The comparative effects of social and mechanical stimulation on memorizing. *The American Journal of Psychology*, 45:263–270, 1933. ISSN 0002-9556.
- S. Redon, N. Galoppo, and M. C. Lin. Adaptive dynamics of articulated bodies. In *Proc. SIGGRAPH*, pages 936–945, 2005. doi: <http://doi.acm.org/10.1145/1186822.1073294>.
- S. Ross. *Stochastic Processes*, pages 213–218. Wiley and Sons, second edition, 1996.
- D. Simons and C. Chabris. Gorillas in our midst: sustained inattention blindness for dynamic events. *Perception*, 28:1059–1074, 1999.
- D. Simons and D. Levin. Failure to detect changes to people during a real-world interaction. *Psychonomic Bulletin and Review*, 5:644–649, 1998. ISSN 1069-9384.
- S. Singh, M. Kapadia, W. Hewlett, , G. Reinmann, and P. Faloutsos. A modular framework for adaptiv agent-based steering. In *Proceedings of the 2011 symposium on Interactive 3D graphics and games, I3D ’11*. ACM, 2011.
- M. Slater and A. Steed. A virtual presence counter. *Presence: Teleoperators & Virtual Environments*, 9(5):413–434, 2000. ISSN 1054-7460.
- M. Slater, P. Khanna, J. Mortensen, and I. Yu. Visual realism enhances realistic response in an immersive virtual environment. *IEEE computer graphics and applications*, pages 76–84, 2009. ISSN 0272-1716.

- C. Stocker, L. Sun, P. Huang, W. Qin, J. Allbeck, and N. Badler. Smart events and primed agents. In *Intelligent Virtual Agents*, pages 15–27. Springer, 2010.
- M. Sung, M. Gleicher, and S. Chenney. Scalable behaviors for crowd simulation. In *Computer Graphics Forum*, volume 23, pages 519–528, 2004.
- C. Thornton. *Artificial Intelligence*. New Age International Ltd., 2007. ISBN 9788122416619.
- Y. Toyoda. A simplified algorithm for obtaining approximate solutions to zero-one programming problems. *Management Science*, 21(12):pp. 1417–1427, 1975. ISSN 00251909.
- J. Van den Berg, M. Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1928–1935. IEEE, 2008.
- M. Verde. Associative interference in recognition memory: A dual-process account. *Memory & cognition*, 32(8):1273, 2004. ISSN 1532-5946.
- J. Viega. Those darned sims: What makes them tick? Lecture at the 2001 Game Developers Conference, 2001.
- V. Vinayagamoorthy, A. Brogni, M. Gillies, M. Slater, and A. Steed. An investigation of presence response across variations in visual realism. In *The 7th Annual International Presence Workshop*, pages 148–155, 2004.
- M. Wißner, F. Kistler, and E. André. Level of detail AI for virtual characters in games and simulation. In R. Boulic, Y. Chrysanthou, and T. Komura, editors, *Motion in Games*, volume 6459 of *Lecture Notes in Computer Science*, pages 206–217. Springer Berlin / Heidelberg, 2010.

- A. Yonelinas. Receiver-operating characteristics in recognition memory: evidence for a dual-process model. *Journal of experimental psychology: Learning, memory, and cognition*, 20(6):1341, 1994.
- A. P. Yonelinas. Recognition memory ROCs for item and associative information: The contribution of recollection and familiarity. *Memory and Cognition*, 25(6):747–763, 1997.

Index

- alibi, 68
- ambient attentional load, 28
- and-or tree, 45
- associative recognition, 29
- attention, 27
- Break in Realism, 17
- break in realism, 19
- criticality, 24
- cued recall, 29
- dual-process model, 30
- duration, 31
- effective attention, 28
- false alarm, 30
- familiarity, 30
- feature graph, 45, 87
- Focusing, 28
- Following, 29
- fundamental discontinuity, 23
- hazard function, 32
- hit, 30
- intact pair, *see* target
- level of detail
 - proactive, 36
 - reactive, 36
 - simulation, 8
- lure, 29
- memory, 29
- Multiple-Choice Knapsack Problem, 42
- observability, 27
- paired associate, *see* cued recall
- perceptual criticality, 5, 19
- perceptual criticality modeling, 44
- presence, 16
- realism
 - perceptual, 2
 - ideal, 3
 - practical, 3
- rearranged pair, *see* lure

recollection, 30

retroactive interference, 30

return time, 32

semi-Markov process, 69

survival analysis, 32

target, 29

unrealistic long-term behavior, 23

unrealistic state, 22

visualization, 1