FUZZY MOTION CONTROLLERS AND HYBRIDS

A thesis presented in partial
fulfilment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY
IN COMPUTER SCIENCE

Massey University,
Albany, New Zealand.

Anton Gerdelan

2011

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

This work would not have been possible without my supervisors at Massey University; Professor Ken Hawick, who insisted upon the highest scientific standards, and Doctor Napoleon Reyes, who inspired me to start moving things with fuzzy logic. I am in a great debt of gratitude to Professor Carol O'Sullivan, who provided support and guidance for 2 years of study in Ireland with the Graphics, Vision, and Visualisation (GV2) group at Trinity College Dublin.

I would like to acknowledge my talented peers, from whom I have gained a great deal of knowledge and experience during the last 3 years; Prof. Henry Rice, Dr Martin Johnson, Dr Chris Messom, Dr Sébastien Paris, Dr Simon Dobbyn, Dr Rachel McDonnell, Dr Yann Morvin, Dr Veronica Sundstedt, Dr Guy Kloss, Dr Ljiljana Skrba, Dr Sophie Jörg, Dr Darren Caulfield, Dr Cathy Ennis, Dr Steve Collins, Dr Rozenn Dahyot, Cormac O'Brien, Nithin Tharakan, Andrew Corcoran, Dr Michéal Larkin, Dr Arno Leist, Dr Daniel Playne, Fintan McGee, Martin Pražák, Teo Susnjak, Brian Cullen, Eric Risser, Jonathan Ruttle, Robert Smyth, Paul McDonald, and Tom Van Eyck.

I would also like to thank Craig Reynolds for his invaluable advice regarding motion control.

Abstract

This thesis describes implementations of motion control systems that are based on fuzzy logic; fuzzy motion controllers. The controllers are used by to drive a variety of simulated vehicles and computer-animated characters. The problem of heading towards a destination whilst simultaneously avoiding static and dynamic obstacles is addressed with fuzzy motion controllers. For situations where a level above reactive motion control is required, such as path-planning behaviour or traffic rule following, then hybrid algorithms are proposed; combining fuzzy motion controllers with other navigation algorithms. Consideration is given to behavioural level of detail models, with transition between behavioural models of different complexity based on the proximity, or visual importance of characters to the camera.

Fuzzy controllers have a set of fuzzy rules, or a "rule base" that defines the inference of the controller. There is no assurance that hand-calibrated rule bases are optimal, and indeed that calibration based on fixed test environment will apply well to a dynamic environment. Special consideration is given to genetic-fuzzy systems, which use a genetic algorithm to automatically calibrate a rule base. Various architectures for genetic-fuzzy system are proposed and evaluated including dynamic systems, which have the ability to learn "on the fly", rather than in fixed experiment scenarios. A relationship between genetic algorithm parameters and time-efficient fitness improvement is found.The time requirements of training more complex "cascading" fuzzy systems are discussed. Distributed and parallel training models are also considered.

A new, modular agent middleware is proposed, which is the underpinning software that perceives the complex environment, feeds inputs into the fuzzy motion controllers, and effects output actions for each character and vehicle. The middleware model is successfully used to drive a range of vehicles and characters used in experiments.

The problem of evaluating motion controllers within a scientific framework is discussed. Several candidate solutions are used, and a system for objectively evaluating mechanically simulated vehicle motion is defined and evaluated. A complete tool-chain for designing complex simulations and doing scientific experiments with them is is developed and discussed in detail, including simulation software design methods, libraries, visualisation tools, and useful algorithms, a well-defined mechanical simulation system, and practices for collecting data from simulations, and quantifying uncertainty.

Chapter 1

Overview and Aim of This Thesis

This thesis investigates new motion controllers for road and all-terrain vehicles (cars, trucks, caterpillar-tracked vehicles, etc.). The motion controllers handle steering, acceleration, braking, and gear changing behaviour, based on perceptual information from the environment; information about destinations, obstacles, and the current state of the vehicle. The controllers are designed to fill the rôle of a human driver and can therefore be described as a kind of artificial intelligence (AI). The environments studied in this work are all real-time 3D complex simulations, and are largely reproductions of environments from real-world data.

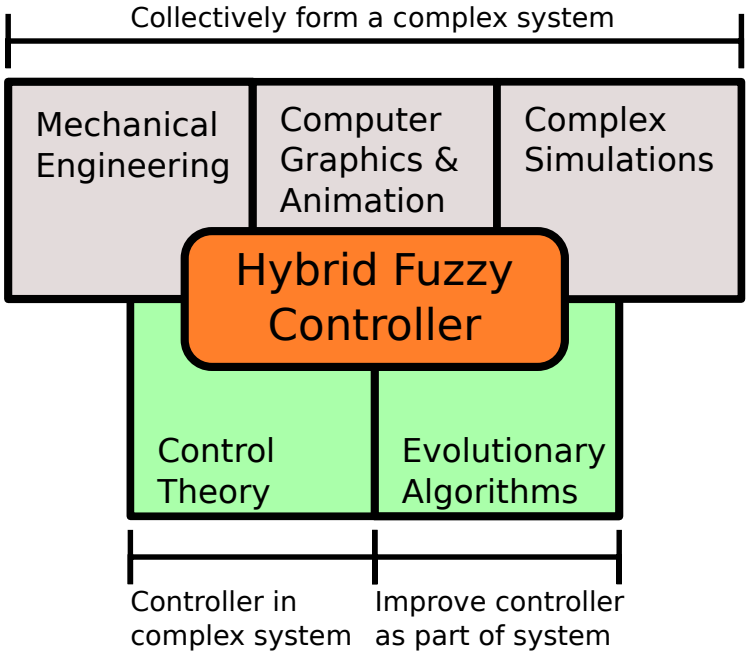

Figure 1.1: Areas contributing to Hybrid Fuzzy Controllers; with complex system theory shaded grey, and control theory shaded green. The "hybrid" title implies either the inclusion of multiple control theory areas or evolutionary algorithms into the controller.

The specific aim of this work is to:

- Design, implement, and evaluate new fuzzy-logic based controllers (using fuzzy control theory) for these complex simulated environments.

- Where controller design is extended to incorporate some non-fuzzy control theory to solve a particular complex problem (hybrid controllers).

Figure 1.1 illustrates the bodies of theory contributing to this work. The resulting controllers are part of a complex system based on a real-time simulation, rendered with 3D graphics, and incorporating mechanical models of the vehicles, and the controllers themselves are built from various control theory (including fuzzy control theory), with some controllers using an evolutionary algorithm to self-improve control performance as part of the complex system.

1.1 Method of Experimentation

The scientific method is applied to the design-evaluation process for each controller studied in this thesis. The basic process here is as follows:

1. A simulation is constructed with mathematical models representing a real-world environment and vehicles
2. Real-world problems emerge (driving from A to B whilst avoiding static and moving obstacles)
3. Some rules and constraints are added (road rules, for example)
4. A conjecture is devised for solving the problem with a control system
5. The control system is implemented and evaluated as a solution to the problem
6. And the results presented with appropriate measurements of error.

Each chapter in this thesis introduces a new controller. The conjecture, design, and evaluation criteria used differ from controller to controller, and are specified at the beginning of each chapter. The later chapters in this thesis, Part IV, introduce fuzzy controllers that have an evolutionary, self-training, component.

The main contribution of this work is the design and evaluation of fuzzy controllers that self-train in real-time, which means that they have the potential to adapt to an environment as it changes, a behaviour that has not been studied extensively in this type of controller. In particular, a system for evaluating *adaptivity* has developed for this thesis; until now such a technique has not been available for this type of behaviour.

Chapter 2

A Brief History of Motion Control in Animation

2.1 Introduction

Autonomous computer-controlled entities can make excellent extras in film scenes as large animated crowds, entertaining opponents in computer games, and to some extent useful autopilots or robotic drivers in real vehicles. The problem domain in each of these applications is similar; an actor or vehicle has to be moved through a noisy or complex environment. A noisy environment is one with a large amounts of unimportant information that needs to be selectively filtered out or ignored. When our actors and vehicles have a degree of autonomy, the software controlling each one can be called an *agent*, a term that will be used throughout this document. The type of motion desired from agents operating in these environments is some combination of:[1]

1. realistic; a simulation of real movement patterns
2. convincing; what an audience perceives to be realistic
3. efficient; minimising energy or reducing collisions
4. entertaining; motion may aim for cartoon-like motion

The distinction and balance between these four aims is often unclear in the literature; the aim is not always well understood or clarified by researchers. It is very easy to make false assumptions that efficient motion is realistic motion, or that realistic motion is convincing or desirable in games or film. From this complicated mixture of motion and steering aims several distinct schools of thought have emerged:

- The principle of least effort (PLE). Aim: a combination of realistic/convincing/efficient motion
- Data-driven crowds. Aim: realistic motion
- Simulation based on perceptual studies. Aim: convincing motion
- Level of detail (LOD) crowd behaviour. Aim: a balance of efficient/convincing motion

[1] Material in this chapter expands original research first published as A. Gerdelan, "A Brief History of AI in Entertainment," Tech. Rep. CSTN-105, CSSG, Institute of Information and Mathematical Sciences, Massey University, Albany, New Zealand, November 2009. [1]

A popular approach to motion control for crowd simulation is based on the principle that pedestrians will always try to minimise effort in going from A to B; often referred to as the *principle of least effort* (PLE) [2], and that introducing individual variation or detailed autonomy into the control algorithm is less important because the overall or aggregate motion flow will be realistic and produce emergent effects that are consistent with collected real-world data.

Controllers designed for realism tend to use a data-driven approach to evaluate the effectiveness of the controllers, and are used for architectural design planning applications such as the Sydney 2000 and London 2012 Olympic Games [3]. Modern data-driven studies show that people in real crowds in fact do not behave like the established efficiency-driven automaton models [4, 5], and that there is also a high degree of variation of movement of individuals within a crowd based on age, gender, disability, emotional state, and other factors that are visible in aggregate motion [5]. This makes data-driven motion controllers very difficult to validate, and renders simulations driven by data from a single region non-generalisable [6] to other populations.

Motion control and animation that aims for perceived realism can be evaluated by psycho-visual tools and user-studies. Such works aim to evaluate the perceptual importance of new motion techniques and to find thresholds of perceptual importance in order to optimise techniques. Recent works have studied the importance of group formations in believable crowds [7], the amount of variety required within a crowd for it to be believable [8], and compared pedestrian motions using perception-based metrics [9].

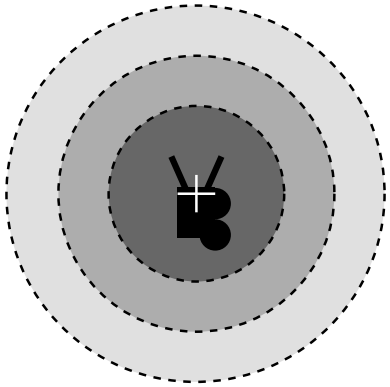

Figure 2.1: A spatial representation of classification of discrete levels of detail based on a camera position and orientation. The depicted method is using both distance, and yaw angle from camera orientation to determine each level. There are 9 levels of detail here; green 0, 1, 2, yellow 0, 1, 2, and red 0, 1, and 2. The idea here is that detail can be rolled off at up to 9 different discrete levels as objects are further away from the camera, and off either side of the screen. Information from outside the frustum (coloured area) is discarded completely.

The level of detail (LOD) principle [10] is used in 3D graphics to reduce the rendered detail of objects that are deemed perceptually less important [10–12]; which usually means further from the camera (see Figure 2.1). The idea is that the simulation is then more efficient and thus more detailed or more numerous objects can be simulated. This principle has also been applied to behavioural detail [12, 13]. The computational time slice allowed for steering vehicles or actors in perceptually important zones is then larger than those off-screen or in the background, making this a technique that aims to balance computational efficiency with perceived realism. An ongoing area of study is at what angle and distance from the scene viewpoint that these levels of detail change can occur without a perceived loss to realism [14, 15]

Controllers that aim for entertaining motion are evaluated by artists, film directors, and preview audiences [16,17]. Entertainment-driven motion is very difficult to design for as realistic motion may

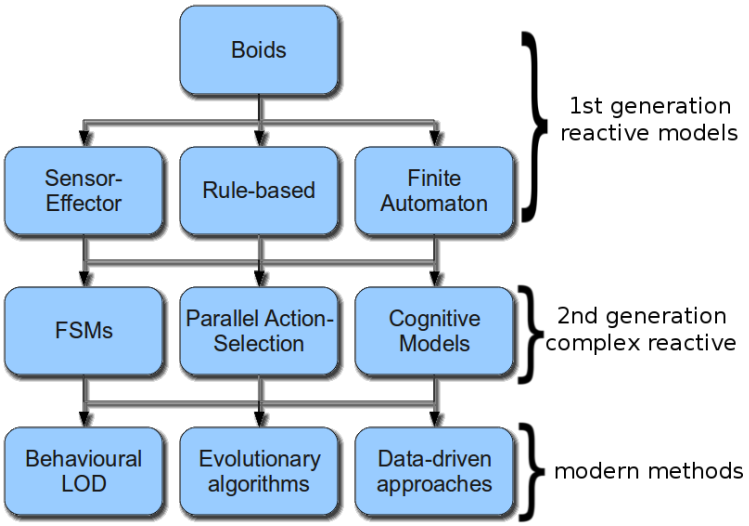

Figure 2.2: This diagram gives us an overview of the "evolution" of behavioural control theory for agents in computer animation. This model has been adapted from Donikian (2009) [19]. The topics in the model are not arranged in the order of their introduction in the literature broadly speaking (finite state machine (FSM)-type models have been in existence for hundreds of years), but rather specific to their adaptation to motion control in animation. The model is divided into 3 "generations", starting in the 1980s with the introduction of boids, and the adaptation of basic automaton models to software agents. The 2nd generation moves onto more complex models. FSMs are included here because this coincides with the introduction of FSMs as ubiquitous game "AI" in the 1980s and 1990s. The latest generation introduces some topics that will be explored in this thesis, and the emergence of data-driven approaches, where motion-control algorithms are designed to reproduce realistic behaviours by correlating with real recorded data.

be not actually desirable in entertainment. Designers of such controllers take great care in providing suitable "knobs and dials" that allow the motion to be customised during preview screenings [16]. A special cartoon-like motion was developed for a character in the Wall-E film, using a customised spring equation [16]; a technique that could not have been based on a realistic model. MASSIVE [18] software was used in both the Wall-E film and the Narnia film [17] for motion of large groups of characters. Amongst the many techniques used by MASSIVE, the reactive motion uses physics-based objects that encompass a space larger than the characters, which when colliding give the impression that the characters are avoiding each other, when computationally their physics bodies are actually colliding and sliding around each other [18].

2.2 Overview of Key Algorithms

Approaches to solving motion control and steering problems for animation have evolved in several distinct generations. Donikian (2009) [19] classifies reactive behavioural algorithms (of which motion control is one) into two previous generations, with proposed cognitive science models considered part of a new generation. The first generation in this classification is divided into three parallel streams of research; sensor-effector models, behaviour rules models, and finite automaton models. This classification system can be extended by considering fuzzy, evolutionary, data-driven and LOD algorithms to be part of the latest generation of research. Figure 2.2 illustrates this extension of Donikian's model.

2.2.1 Boids and Flocking

Reynolds' 1987 SIGGRAPH article [20] introduces an animation system for bird-oid (bird-like) actors called "boids". The article was accompanied by a short film featuring boids. A still is shown in Figure 2.3. The key idea of boids is to approximate the real world motion of groups of animals (flocks, herds, and schools) by simulating each individual within the group, and having the overall (aggregate) motion of the group emerge as a result. Each individual is moved using a balance of attractive and repellent forces. The actors in boid system are also able to work within 3D physics constraints. The flight dynamics required for realistic bird animation and motion are detailed in the article.

Figure 2.3: A screen capture from the SIGGRAPH Electronic Theatre short film Stanley and Stella in: Breaking the Ice (1987), featuring boid-driven birds and fish. A group of boids move in a group by a careful balancing of steering towards the average heading of the near neighbours, and basic avoidance of neighbours. Convincing flocks emerge from these basic rules.

Boids extends particle systems [21] by adding orientation, collision avoidance, and semi-autonomous motion. Each *actor* (a concept now synonymous with an *agent* [22]) is given a limited perception of the world with which to make steering decisions. Reynolds finds that if the actors are given complete information (the locations of all of the actors in the flock) then the aggregate motion of the flock is unrealistic or produces undesired centrifugal formations [20]. For this reason each actor is given only the location of its immediate neighbours in the flock, and a limited field of view as depicted in Figure 2.4.

Reynolds' *flocking* mechanism has become a standard in computer animation. Flocking has three components:

1. Separation: actors steer to avoid neighbours
2. Alignment: actors steer toward the average heading of neighbours

3. Cohesion: actors steer toward the average position of neighbours

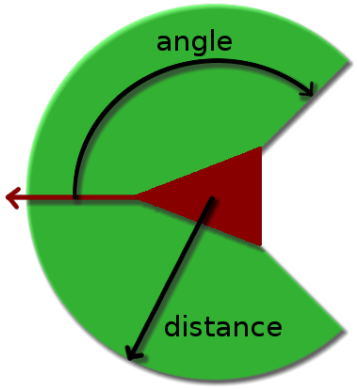

Figure 2.4: A boid's neighbourhood is limited to a distance, and an angle from its heading orientation, such that it only considers immediate neighbours in its steering behaviour. Interestingly, if the boid is allowed to consider all of its neighbours then flocking behaviour does not emerge, but rather flock formations depend on having only a limited neighbourhood perception.

Reynolds does not detail how the various forces are balanced other than to suggest the use of a dampening function [21]. This is no doubt a constraint satisfaction problem that requires quite some tedious manual trial and error (a problem common to many motion controllers).

The flocking mechanism is easily confused when presented with geometric obstacle problems. Reynolds suggests in the original article that this is due to the lack of a detailed perception model of virtual computer vision [21], but it is now known that the problem is more complex than this, and requires inclusion of a path-planning algorithm at a higher level. Finding a good balance between reactive steering methods such as the flocking mechanism, and path-planning navigation methods which guide them is an active research topic [13, 23–26].

An adapted copy of Reynolds' original boids software was used to animate swarms of bats and flocks of penguins in the 1992 action movie *Batman Returns*. The theory also contributed to the emerging artificial life (AL) [27, 28] field, with the aggregate group behaviour an example of the AL principle of emergence.

Boids has also been notably extended to use fuzzy controllers in place of the original mathematical decision-making model [29]. The authors found that fuzzy logic controllers were a solution to dealing with environment noise (many variable inputs to the decision process) and enabled more human-like or animal-like decision making.

2.2.2 Helbing's Crowds

Another notable work is Helbing's 1995 article on crowd simulation "Social force model for pedestrian dynamics" [30]. The article appears in the Physical Review E journal, which is home to key works in the area of traffic microsimulation, that is, traffic simulations based on fluid dynamics models which model gas and liquid flows but simulate the individual vehicles within the flow, rather than just the overall pattern of movement [31–33]. Helbing adapted the microsimulation model and applied it to the movement of pedestrians in crowds. The model is a basic attractive/repellent force model as seen in earlier the Boids [20] flocking mechanism. Interestingly Helbing has not drawn on Reynolds' attractive/repellent force model for computer animated animal groups, despite its similarities.

In the article Helbing gives us basic simulation layouts but no experiment result data which would indicate the effectiveness of the system in terms of crashes or expediency. Like Reynolds'

Boids article it is then really a proposal rather than a scientific study, but along with Boids it has since been adopted as a benchmark system for evaluating the effectiveness of new crowd simulation systems [13, 19, 24]. At this stage the emerging crowd animation field had still not crossed domains or levered the considerable body of similar research that had been done in psychology or robotics where the limitations of such approaches had already been analysed in detail [34]. Helbing's agent architecture; the "process leading to behavioural changes" [30] is almost identical to the concept of the Belief-Desire-Intention (BDI) agent model introduced by Bratman in 1987 [22], but without providing any of the concrete implementation details that BDI-based hierarchical architectures had developed for robot applications [35]. The concept of planning (of paths) or higher-level behaviour has not been given any special consideration; thus the system is completely reactive and therefore has similar limitations to Reynolds' flocks, as discussed in the previous section. It is not until Reynolds' later works [36] that forward-planning ideas are given serious consideration within the crowd simulation field.

The article makes much mention of its relevance to empirical data, and is built on the assumption that the emergent flow movement of real crowds is predictable, and gas or fluid-like. This assumption has been carried by the following generation of crowd simulations, but evidence of this correlation is not provided other than to cite a 1970 comparison of pedestrian flow data to the Navier-Stokes equation, and two of eight references to Helbing's own works which introduce a similar study with Boltzmann-like models. Modern data-driven models [5] contradict many of these claims; which find that the movement of pedestrians in crowds is indeed chaotic, containing many "curious" or otherwise inefficient patterns of movement which are not predictable by gas and fluid-dynamic equations.

Some emergent behaviour arrives from Helbing's crowd simulations; lane-following in open spaces, which is now widely regarded as unrealistic and undesirable [13], and oscillating direction changes at intersections which Helbing points out is a limitation of the approach.

For large-scale crowd simulations a Helbing-like particle or fluid model focusing on aggregate motion is still by far the most computationally efficient approach for producing massive crowds of animated pedestrians [37].

2.3 Evolving Motion Controllers

There are two basic problems with agent controllers that act in complex environments;

- Control systems have a lot of functions, variables, and rules that need optimising
- Behaviours may need to adapt to change in the environment

Therefore designing motion controllers by hand is a very difficult task, and in a stochastic environment there is usually no guarantee that they are optimal. Hand-designed controllers are also not able to adapt to unexpected change in conditions. Evolutionary algorithms have the potential to solve either of these problems. These are algorithms that are based on biological "survival of the fittest" type selection. The mechanism behind this is that the agents can try a range of behaviours, evaluate their effectiveness, keep the best behaviours, and use them to generate a new set. This should lead to a kind of self-tuning controller.

There are two categories of evolutionary motion controller;

1. Static tuning: Continually tries to solve a fixed (static) problem in batches.
2. Dynamic learning: Learns "on the fly"; adapts its behaviour continuously as it moves.

The static approach tunes a controller against a fixed problem. This approach is unable to solve the problem of adapting to change. Static tuning is characterised by batches of evaluation runs that are done prior to "final" implementation. A good example of this approach is Reynolds' 1994 algorithm for evolving corridor-following motion [38], where it was initially found that learning

a fixed problem in a strongly-objective evaluation forced the agent to learn the quirks of that exact problem such that the resulting behaviour did not generalise to similar problems [38] (over-training). Artificial noise ("jitter") was injected into the training to improve the generalisation of the result.

The corridor-following approach was extended to automatically discover agents that would learn how to play the game of "tag" [39], which moves more into the dynamic learning area. Reynolds' discusses the potential of using physically simulated vehicles (with mass, momentum properties etc.) as being a future work of merit as it requires the optimisation of a more complicated control programme [39].

There are very few dynamic learning algorithms for motion control, and self-adapting motion is regarded as a "holy grail" problem due to the complexity of controller design required, however, special mention in this category must be given to the real-time NeuroEvolution of Augmenting Topologies (rtNEAT) algorithm.

2.3.1 rtNEAT

A significant recent advance made possible by more powerful computer hardware is the discovery that machine learning techniques can be adapted to run in real-time, evolving vehicle controllers in a matter of minutes. A recent academic work of note in this area is the rtNEAT algorithm [40] which uses an adapted genetic neural network (GNN) (the NEAT algorithm) to perform very fast learning by augmenting a neural network (NN). rtNEATwas implemented in a research-orientated video game called NeuroEvolving Robotic Operatives (NERO) [41] which used a user-in-the-loop approach to learning. In NERO the user (a player) directs the training of a team of GNN-enabled characters by changing the training environment itself in real-time, and by adjusting a punishment-and-reward scheme (reinforcement learning). The characters are able to learn reactive movement, basic path planning, and some basic military-type tactics in less than an hour [42, 43], with the created behaviour unique to the user's input [44].

The most noteworthy feature of rtNEAT as implemented in NERO is the treatment of fitness evaluation. Defining a good fitness function is the paramount problem of all genetic systems. A fitness function is responsible for defining a problem to be solved by a genetic algorithm (GA) in clear, mathematical terms. A fitness function that is too simple or too complex will prohibit or confuse the learning process. The weighting given to different inputs to the fitness function will affect the priority of tasks; the ideal weighting is usually not known prior to testing. NERO addresses these problems by simply passing them onto the lap of the human user who acts as a training supervisor. The advantage of this approach is that the fitness function itself is dynamic; the supervisor can adjust the reinforcement learning weightings as the learning happens to learn additional complex behaviours [45]. The disadvantage is that the user needs to be present during the whole training process; therefore this approach is only suitable for a selected sub-set of training applications.

Because rtNEAT is built on GA and artificial neural network (ANN) technology it has all of the advantages and disadvantages of these algorithms; NNs are a black box; it is very difficult to dissect or manually tweak the resulting solution (the network itself) after training. Networks will almost certainly not solve a given problem as well as a specific hand made solution taking all problem features into account. The black box nature of neural networks can become a major frustration to designers seeking to improve the problem-solving efficiency of the network. With these limitations taken into account it can be said that the designers of rtNEAT have found an ideal application in the NERO game; a user-driven fitness evaluation scheme to address the fitness design problem, a stochastic and hard-to-specify problem domain which is ideal for neural-network learning, where the outcome is the effectiveness of a competitive team of characters versus another GNN team with its own training, so even if the neural network is not an ideal solution to a problem, it is at least on an even playing field.

Chapter 3

Fuzzy Logic Controllers

3.1 Overview

Previous works, contributing to this thesis, have made extensive use of fuzzy controllers for controlling soccer robots, both with remote-controlled vehicles, and with simulated robots [46–48]. This thesis focuses on fuzzy controllers for motion control in simulations and games, but it is worth noting that the control theory generalises across domains, and can apply equally well to motor-based motion control as it does to software-only motion control problems.[1].

Fuzzy sets were introduced by Lofti Zadeh as a mathematical theory in 1965 [49], extending classical sets. Fuzzy sets have their own set operators, equivalent to the union, complement, etc. operators of classical set theory. Fuzzy sets allow us to represent a partial truth, or imprecise values between completely true and completely false. Extending this classification tool to a fuzzy logic gives us a mechanism for discriminating imprecise or changing data into a small group of overlapping fuzzy sets. From this foundation a very simple judgement-based reasoning can be created; fuzzy inference [50], which can deal with complex real-world data; mimicking human decision-making. Equipping fuzzy inference with sensors and implementing them in a system with a control theory-based feedback loop enables the building of *fuzzy controllers*. Fuzzy controllers require very little computational overhead, and can also produce smooth transitional outputs. Fuzzy logic is therefore an ideal candidate for modelling human, animal, or vehicle behaviour in large-scale real-time simulation.

A good example for how fuzzy sets might be used to classify a real, continuous value into sets is colours. In his *Opticks*, Sir Isaac Newton observes;

> "the Spectrum formed by the separated Rays...with this Series of Colours, violet, indigo, blue, green, yellow, orange, red, together with all their intermediate Degrees in a continual Succession perpetually varying. So that there appeared as many Degrees of Colours, as there were sorts of Rays differing in Refrangibility". [51]

Newton has made several classifications of bands of colour, which is very convenient for human perception, but also notes the intermediate degrees between each colour, which are very hard to represent using the same kind of classical sets. The strength of fuzzy sets is that they are able to represent both definite classifications, and intermediate or overlapping classifications, with one model. Each fuzzy set is defined by a mathematical function, in the same way that a classical set is defined, but it is also possible to overlap these functions, and afterwards perform all of the *if-then-else*-type logical operations with the same ease as with Boolean logic values. An example of this can be built from Newton's observation. Taking the frequency of light in teraherz (THz) as

[1] Videos of various simulations and games using the fuzzy logic controllers developed in the work are available at `http://antongerdelan.net/videos.html` under the heading *Fuzzy controllers*

the continuous real variable, this can be classified into distinct colours. Fuzzy values made from 3 membership functions or fuzzy sets are used in the standard pattern fuzzy controllers in this thesis, so for this example we can consider the part of the spectrum that contains the 3 colours red, green, and blue. Figure 3.1 illustrates the process of classifying a real value into fuzzy sets. In the literature this process is referred to as *fuzzification*; or where crisp values are *fuzzified* into fuzzy sets.

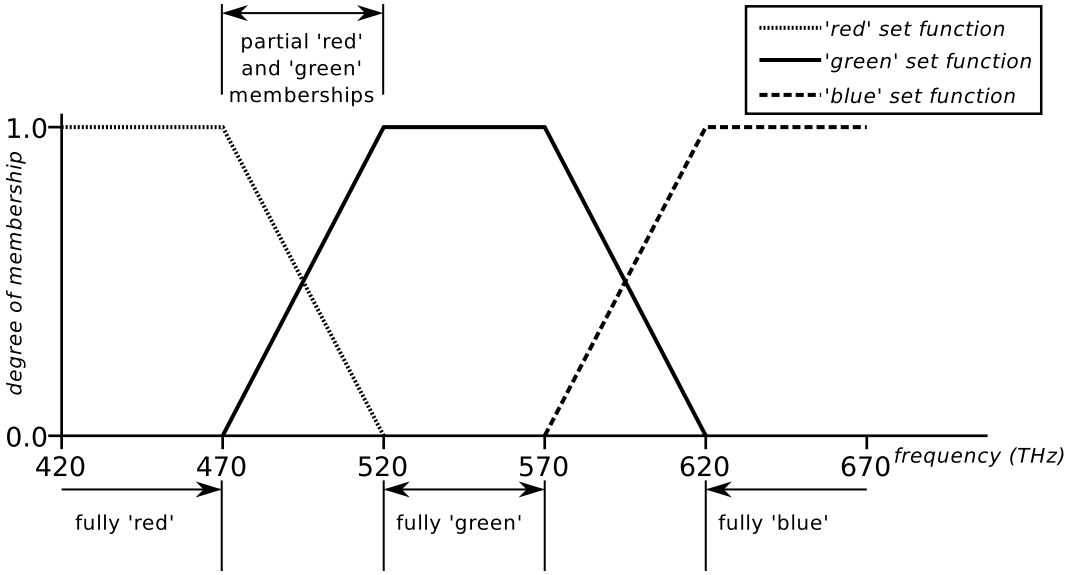

Figure 3.1: Here we have an example of fuzzy set classification of colours from frequencies of light. This example classifies into 3 fuzzy sets; one set each for the colours red, green, and blue. Set functions overlap, which allows us to classify intermediate colours as partial members of 2 sets.

In Figure 3.1 each colour has a mathematical function which defines its limits, but also the degree of membership in that set for each value of frequency. As we would find with classical sets, these fuzzy sets have definite frequency bands where we can consider a frequency as wholly one colour, and none of the others, but we also have intermediate bands; oranges, yellows, cyans, etc. where we can consider a colour to be a partial member of two colours. These partial members are awarded a set membership value of between 0 and 1 in each of two sets. A frequency of 650 THz would give us the set memberships of 0.0 red, 0.0 green, and 1.0 blue. A frequency of 482.5 THz (one quarter of the way between full red and full green), would give us set membership values of 0.75 red, 0.25 green, and 0.0 blue. All three of these membership values make up a single fuzzy value. Of course it is possible to represent fuzzy values with more than 3 sets; memberships for more colours for example, however, the more sets that make up a fuzzy value, the larger the set of rules that are required for fuzzy inference. The goal, therefore, is the represent a value with a minimum number of sets, whilst still retaining enough continuous detail. The motion controllers in this thesis will not be classifying colours, of course, but distances, angles, speeds, and other data relevant to motion control.

3.2 Standard Pattern 2-Input Fuzzy Controller

Fuzzy logic has been employed by other agent simulations for its flexible nature - easily expanding to accommodate more complex input variables [52]. The fuzzy control systems that in this thesis are based on a two-system pattern. This model comprises a route-following system, and a dynamic obstacle avoidance system; these systems are counter-balanced and roughly follow a route of way-points whilst simultaneously avoiding static and dynamic obstacles. These control systems therefore

handle *reactive* motion control. Discussed in the following chapters, this model has been used in the works described in this thesis, with varying degrees of success, to drive crowds of pedestrians with staggered levels of detail, individual cars in simulated road traffic, physically simulated vehicles, all-terrain vehicles, herds of animals, and self-training vehicles. The motion control model has been adapted somewhat for each implementation, the details of which are described in relevant chapters, but the general arrangement is described in this chapter.

Figure 3.2 gives us a breakdown of the basic fuzzy controller architecture used throughout the works in this thesis. In segment (a) in the figure, we have a representation of two fuzzy classifiers as described in the previous section. Each one of these classifiers takes a single continuous input (usually a distance or an angle). These are then "fuzzified" into partial memberships of 3 fuzzy sets, based on 3 fuzzy set functions, as in Figure 3.1.

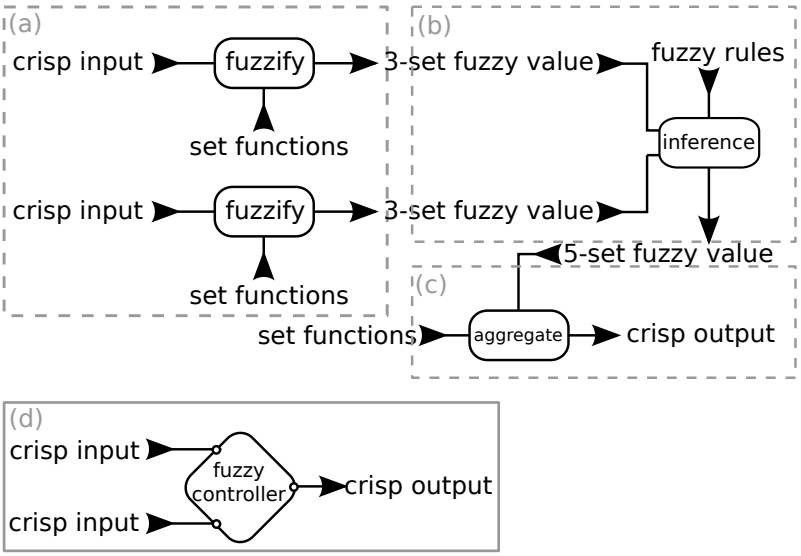

Figure 3.2: The 2-input 1-output fuzzy controller design. The controller uses fuzzy classification (a), as explained in the previous section, inference or rule-lookup (b), and de-fuzzifies by aggregation (c). The full process can be condensed into a single controller, with symbol (d).

With inputs expressed as fuzzy truth values we are able to perform fuzzy logic operations by matching inputs with a fuzzy rule. This process is called fuzzy "inference", and is depicted in window (b) of Figure 3.2. The inference process requires a set of fuzzy rules, matching every possible combination of inputs together. For two fuzzy values of 3 sets each we therefore need 9 rules to match all possibilities. The first 3 of 9 rules from a fuzzy controller are:

```
If the obstacle is a near distance away,
and the obstacle is a narrow angle from the heading,
then change speed to zero.

If the obstacle is a medium distance away,
and the obstacle is a narrow angle from the heading,
then change speed to slow.

If the obstacle is a far distance away,
and the obstacle is a narrow angle from the heading,
then change speed to medium.

...
```

Using fuzzy inference, a designer can make a series of rules in the same way as conditional logic but the rules are applied to continuous values. These rules are taken from a controller used for obstacle avoidance. Note that the outputs of each rule are also a fuzzy value.

Fuzzy rules for inference can be expressed in an abbreviated form in a rule table, as in Table 3.1. To cover cases where inputs are a partial member of multiple input sets because the input values fall inside overlapping set membership functions all of the rules are evaluated, and using the fuzzy Zadeh union operator [50], the minimum of the input memberships for each rule are selected, and then the Zadeh complement operator is applied to select the maximum output value of each rule.

	narrow	mid	wide
near	zero	slow	medium
medium	slow	medium	fast
far	medium	fast	top

Table 3.1: This table is the complete set of example rules as in the natural language sample earlier in the section. The two input fuzzy values are an angle, which is split between 3 fuzzy sets (narrow, mid, and wide) and a distance, with sets (near, medium, and far). Outputs are fuzzy sets for a vehicle speed. Note that there are 5 output sets for one fuzzy speed value.

The fuzzy values used for output throughout this thesis are a 5-set model. Rather than define an overlapping function for each set, as with the 3-set values, a single mid-point value (sometimes referred to as a "singleton" value in the literature) is defined for each set. Figure 3.3 gives us the singleton set functions. The single-point values are used for the "defuzzification" process, that is, converting fuzzy set memberships back into crisp values.

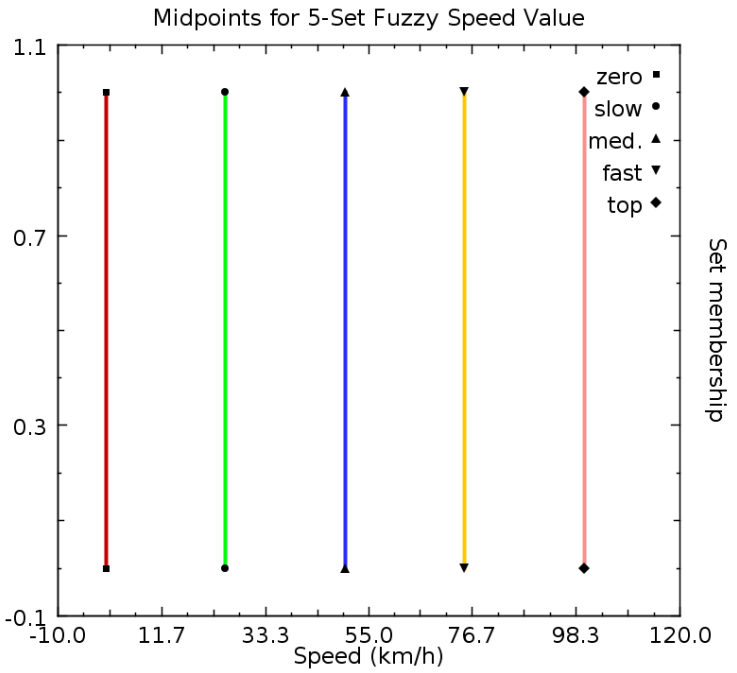

Figure 3.3: This graph is an example of a standard 5-set fuzzy output value. The plots give the membership set functions for each of the 5 fuzzy sets that make up the "speed" fuzzy value. Each set has only a single value defining it. This is because these sets are used to convert a partial fuzzy set membership values into a single crisp, or real output value.

To obtain a final crisp output value all of the fuzzy output sets are aggregated together using a centre of mass function, weighted by the degree of membership of each fuzzy input set. The aggregation process is modelled by Equation 3.1.

$$y = \frac{m_0 * w_0 + m_1 * w_1 + ... + m_n * w_n}{w_0 + w_1 + ... + w_n} \tag{3.1}$$

Where y is the crisp output, m is a mid value for a fuzzy output set, and where w is the weight value, between 0 and 1; the degree of membership of a particular fuzzy output set. In the vehicle controller example this crisp output will be a single speed adjustment value in km/h. Where most if-then-else decision making systems produce stepped outputs as cases change, this aggregation procedure allows us to smoothly transition outputs between cases. This means that as the values for input angles and distances move from full membership of one set to another; for example transitioning from 1.0 "far" distance to 1.0 "near" distance as a vehicle moves closer to an obstacle, the weighting assigned to each output rule in Table 3.1, and therefore matching fuzzy output set, gradually shifts "top" 100.0 km/h to "medium" 50.0 km/h, with reference to Figure 3.3.

3.3 As Part of a Hybrid Controller

There are several reasons why we might want to combine a fuzzy motion controller with another sort of motion controller; creating a kind of "hybrid". The most obvious cases are with algorithms that are very good at path-planning or use a search algorithm, which fuzzy controllers do not do, but lack the finely-grained or smoother reactive motion. The fuzzy controllers of the type used in this thesis only consider one obstacle at a time, so it is certainly advantageous to combine the controller with another that takes into account secondary obstacles, and is able to help the fuzzy controller plan a path that avoids local maxima traps (dead ends). A number of works were published during the course of this thesis combining path-planning algorithms with fuzzy model. A series of works concentrated on driving simulated road vehicles through traffic using fuzzy controllers, but combining custom search algorithms for ensuring that paths were given for the vehicles to follow that obeyed all of the relevant traffic rules [53, 54]. A hybrid was developed that used a fuzzy system that interpreted 3D distances and angles, and operated in a fully 3D all-terrain type environment. This controller was merged with a path-planning controller that was able to break down the environment into a topological map and compute a depth-limited A* Algorithm based path for the fuzzy controller through the environment [55]. And numerous works investigated combining a fuzzy controller with a genetic algorithm in order to give the controller a machine-learning capacity for optimising its own rules [56–58].

An interesting development was the utilisation of a fuzzy controller as one behavioural "level of detail" in a set of control systems used for controlling large crowds of animated pedestrians [13]. The fuzzy controller was less computationally expensive than the motion algorithms which took multiple moving obstacles and fixed geometry surfaces into account, so the idea was to use the geometric controller for crowd members that were closest to the camera viewpoint, the fuzzy controller for those in mid-view, and only a planned path for barely visible characters in the distance. In this way a much larger size of crowd was able to be simulated in real-time whilst giving progressively more realistic motion to characters occupying more space in the shot. Figure 3.4 depicts this level of detail arrangement in action. The behavioural levels of detail were tied to a "unified" level of detail that was based on perceptual experiments, and also governed animation levels of detail and graphical complexity of the models. Thus the characters in the extreme distance are rendered as a series of points, rather than as the convex meshes in close view.

Figure 3.4: Levels-of-detail behaviour. The characters closest to the bottom of the shot are being driven with a comprehensive, but computationally expensive "geometric" controller. There is a static group of 3 "conversationalists" centre shot, and behind them a very large group of characters using fuzzy controllers for reactive motion. In the extreme distance, visible only as dots, are characters with no reactive behaviour, following a pre-computed path.

Chapter 4

Assembling a Toolkit for Doing Science with Simulations

4.1 Overview

Measuring data with simulations, in particular game-like real-time simulations, is highly subjective and open to bias because the scientist typically develops the environment, the subject algorithms that are measured, and also the measurement tools themselves. It therefore follows that it is necessary to have a discussion of the apparatus used for creating the environments and measurement tools used in this thesis. It is also interesting to add this sort of discussion as this information is not usually found together in one place in the literature so is therefore also of value to other scientists developing similar experimentation frameworks.

The computer scientist therefore needs to be careful to design an environment that creates a suitably complex problem environment. This presents an unique problem to the computer scientist because it is difficult to design both the problem domain, and the algorithm(s) to solve the problem without introducing bias. There is no easy solution to this conundrum, it is very difficult to conduct a fully objective study in these conditions. The best approach that can be taken is to document, as much as is possible, the methods used and specifications of the created simulation, such that it can be recreated or at least equated to similar simulated environments in use elsewhere. The second step that can be taken is to introduce a level of data transparency into the simulation, that is, to integrate a series of tools that allow relevant observations to made both during, and after the simulation. It is often easier to appreciate the effectiveness of a particular motion controller once a video has been captured or series of visualisations presented. Altogether then, the computer scientist working with 3D simulations constructs a simulation from a variety of libraries and tools, creates a framework within the simulation for conducting repeatable experiments, and crafts a series of specialised visualisation and data collection tools. This chapter introduces the particular "toolkit" that has been developed during the course of this thesis for doing computer science with complex simulations.[1]

It would be most convenient if there were a constantly updated, standardised set of simulated environments for evaluating motion controllers. Using a known framework would increase the comparability of measured results. Unfortunately, given the expanding range of motion control problems it is difficult to find benchmarks to compare against. Research has been made towards establishing a standardised evaluation framework for motion-control and steering problems typical to crowd-simulation type problems [61, 62], however, the motion controllers evaluated in this thesis apply

[1] Material in this chapter draws on original works published as D. P. Playne, A. Gerdelan, A. Leist, C. J. Scogings, and K. A. Hawick, "Simulation Modelling and Visualisation: Toolkits for Building Simulated Worlds," Research Letters in the Information and Mathematical Sciences, Massey University, vol. 12, pp. 2550, 2008. [59], and [60]

to a range of characters with different types of constraints, and it does not make sense to evaluate lane-following traffic scenario-by-scenario against pedestrians in a crowd; we end up with "apples-to-pears" comparisons. The best we can do, therefore, is to develop a simulation that broadly represents the target problem domain for a particular controller, and develop some sensible metrics that measure the performance of the controller in terms that are easily grasped by the reader. For example; we might measure the performance of a race-car controller by measuring its lap times around a fixed racetrack, and an urban traffic vehicle by its number of crashes over a fixed period, or perhaps its overall fuel efficiency. In a stand-alone simulation this kind of observation is useful for investigating the improvement of a particular motion controller over a previous configuration, or observing change to performance as conditions in the environment change.

An absolutely essential task for showing that data taken from a simulation is robust, and that the evaluation of a controller is in any way objective, is to provide measurements of uncertainty or error. This is something that is done very poorly in the literature, thus it is imperative to mention here how tools have been developed to ensure that environments are sufficiently varied, and that large numbers of measurements have been repeated and how uncertainty is quantified for observations from simulations.

4.2 Constructing a 3D Simulation

Simulations that mimic real world environments tend to involve a large number of different simulation models, some of which interact with the others. A typical complex simulation will involve a 3D terrain or urban environment simulation, using algorithms optimised for rendering the environment efficiently whilst maintaining maximum graphical detail, a particle simulation engine for representing gases, dust, sprays of water, and other special effects, a method for animating human and animal characters, possibly based on motion-captured acting, a method for moving those characters around in the environment in a realistic manner, 3D audio models, mechanical simulations for representing vehicle motion, a pipeline for creating, animating, adding variations to, and loading scale models of physical objects into the simulation, and a physics or dynamics model for generating realistic interactions between objects in the environment. Figure 4.1 gives an example of a typical complex simulation combining several different models. In the figure a particle generation model is used for the dust effect; this is a purely cosmetic system that does not affect the rest of the simulation. There is also a 3D terrain mesh, which is optimised to display segments closer to the camera with a greater density of polygons, and those further away with less detail. This mesh, although primarily visual, has a fixed underlying shape which also interacts with the physics system; providing a platform for vehicles to move on, as well as being able to deform, reacting to the impact of physical elements. There are scale models of real vehicles, which have a physical-mechanical simulation which constrains their motion. There is an agent-based model for driving the vehicles, which uses a fuzzy logic controller paradigm. The agent architecture used in this thesis is described separately in Chapter 5. There are also trees represented on the landscape, which require a physical collision model as well, and act as obstacles to the motion-controlled vehicles.

The kind of 3D environments that we are typical asked to move characters and vehicles through are increasingly graphically realistic. Objects in 3D environments are often photo-realistic, and modelled to scaled dimensions. Visual immersion and realism is an increasingly important psycho-visual tool used in even scientific simulations, and especially those that lend themselves to the computer animation and game industries. Figure 4.2 illustrates a method used in this toolkit for creating a scale visual representation of a real vehicle. The entire process takes about 2 days of work per vehicle-sized model for a moderate level of mesh detail. The key to scale recreation of a real vehicle is to source either factory mechanical or model blueprints, and ideally a series of photographs covering most of the angles of a real vehicle. Using a variety of 3D modelling and image manipulation software it is possible to recreate a vehicle to scale. Despite the end result being mostly cosmetic or used for effective visualisation, the process is important to physical and mechanical simulation as

Figure 4.1: Real-world simulations often combine different visualisation models and simulated systems. Combined in the simulation pictured here are a deforming 3D field terrain model, scale models of vehicles developed from mechanical blueprints and photographs, and particle simulations used for the dust effects in the top-right of the image.

the less detailed collision shape is fitted to the visual model. Not only this, but unknown mechanical specifications such as strength of suspension springs can be adjusted until the physically simulated model visually matches the 3D mesh shape, which has been developed from reference images and diagrams. The process of creating a physical model for a vehicle in this manner is discussion in Chapter 11.

Another type of complex 3D simulation is pictured in Figure 4.3. The figure shows us a screen capture from the "Metropolis" project, which part of the work in this thesis went towards at Trinity College Dublin, in Ireland. The characters pictured are animated by motion-captured animation. The importance of this is to produce psycho-visually realistic motion. Figure 4.4 depicts the motion capture process, and lab, used to create animations for the characters in Figure 4.3. In motion capture animation, a real actor wears a dark-coloured costume covered with balls made of highly reflective material. These are positioned on joints and other key locations around the body. A ring of infra-red cameras surrounds the actor, recording the location of all of the reflective markers. This sort of motion-captured animation has been used to drive a body of perceptual research which quantifies limits to believability; up to what distance from the camera viewpoint that characters motions need to be motion-driven to remain perceptually convincing, the percentage of a crowd that needs to have unique actors' motions to retain a perception of variation, and if each animated character model can be driven be an actor of either sex [12, 63–67].

Most realistic terrain in 3D simulations is now generated by fractal modellers, which provide pseudo-realistic terrain but allow very little design control to scenario designers. The terrain illustrated in Figure 4.5, is however based on a novel method developed for several of the experiments in this thesis. The terrain is visually created in 3D by hand, in real time, from within the simulation itself, with a variety of WYSIWYG ("what you see is what you get") tools that are built into the simulator. This work is based on the ETM2 ("Editable Terrain Manager") for the Object-Oriented Graphics Rendering Engine (Ogre) graphics library [68], which adds the advantage of allowing simulated elements in event driven models to deform the landscape during simulation according to the physics model. The real-time optimally adapting mesh (ROAM) algorithm has been used with the terrain [69] to display the environment efficiently. A series of tools are integrated for quickly laying individual obstacle-type elements on the terrain, are distributing large "forests" of obstacles. Indi-

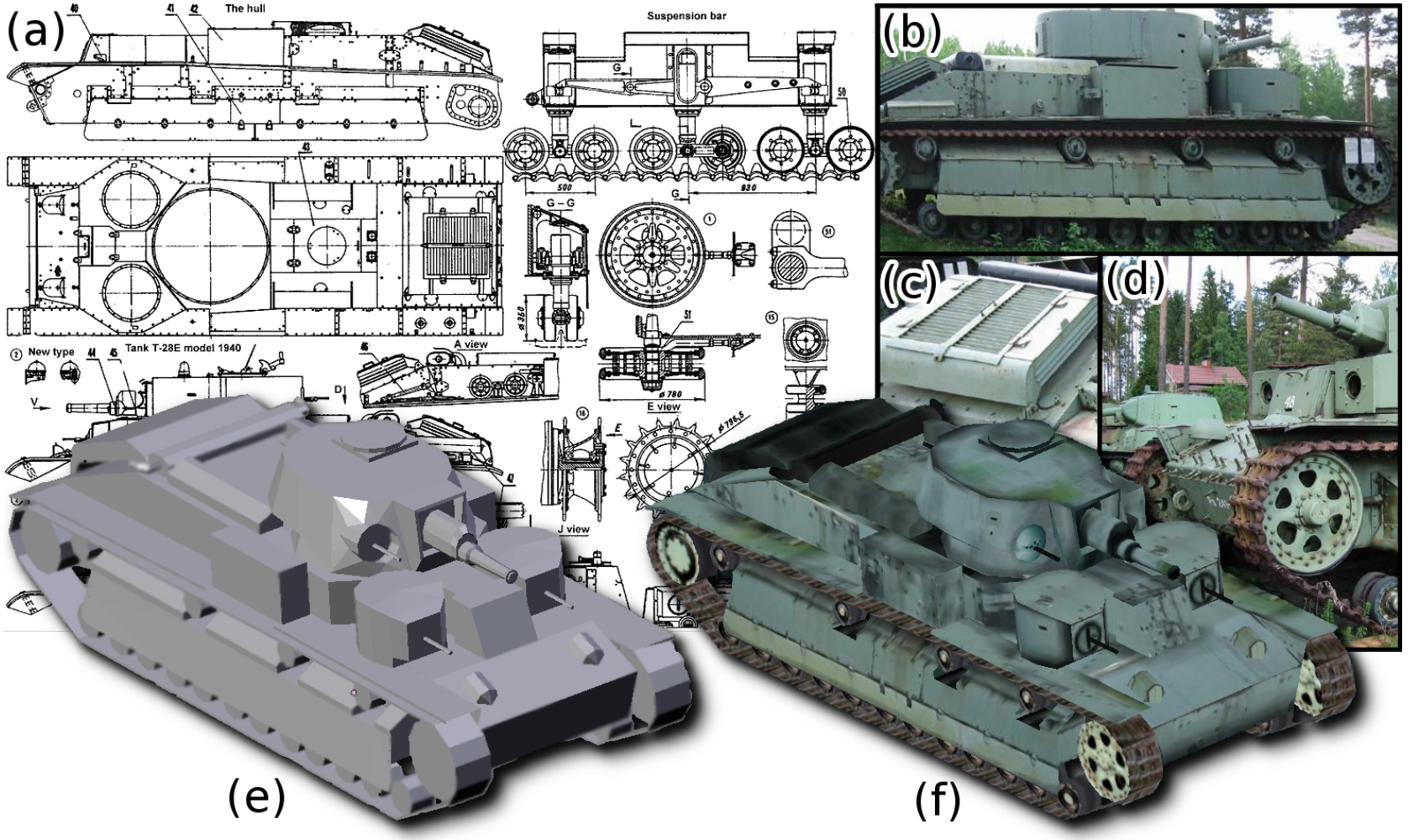

Figure 4.2: Part of the simulation toolkit is a method for reproducing scale models of real vehicles. Here reproduced factory blueprints are sourced (a), which were segmented into a variety of front-profile-rear-top views and adjusted to scale. These images then become the backgrounds for the same projections in a 3D modelling software. Wire-frames were then constructed over the top of the blueprints; producing a scale mesh. This mesh is populated with triangular polygons to create a solid object (e). Using known specifications for the vehicle, the dimensions of the 3D objected were adjusted (blueprints are sometimes inconsistent). Some artistic licence is necessary to fill in gaps. After taking or sourcing photographs, these are then skewed and cropped to produce flat surfaces (b)-(d). A texture bitmap is then created using as much photographic base as possible, and finally applied to the solid object by skewing various views of the object's polygons over the texture; a user tool-driven process that maps the texture to the polygon surfaces.

Figure 4.3: The characters in this crowd are being animated by motion-captured acting. The range of recorded motions can to some extent be interpolated to form a range of different intermediate walk speeds, but there are hard constraints applied that must be respected by movement controllers, which presents a considerable challenge to motion controllers. The number of different sets of motions (turning, walking at different speeds, etc.) from different actors required to maintain the perception of variation is determined by perceptual research.

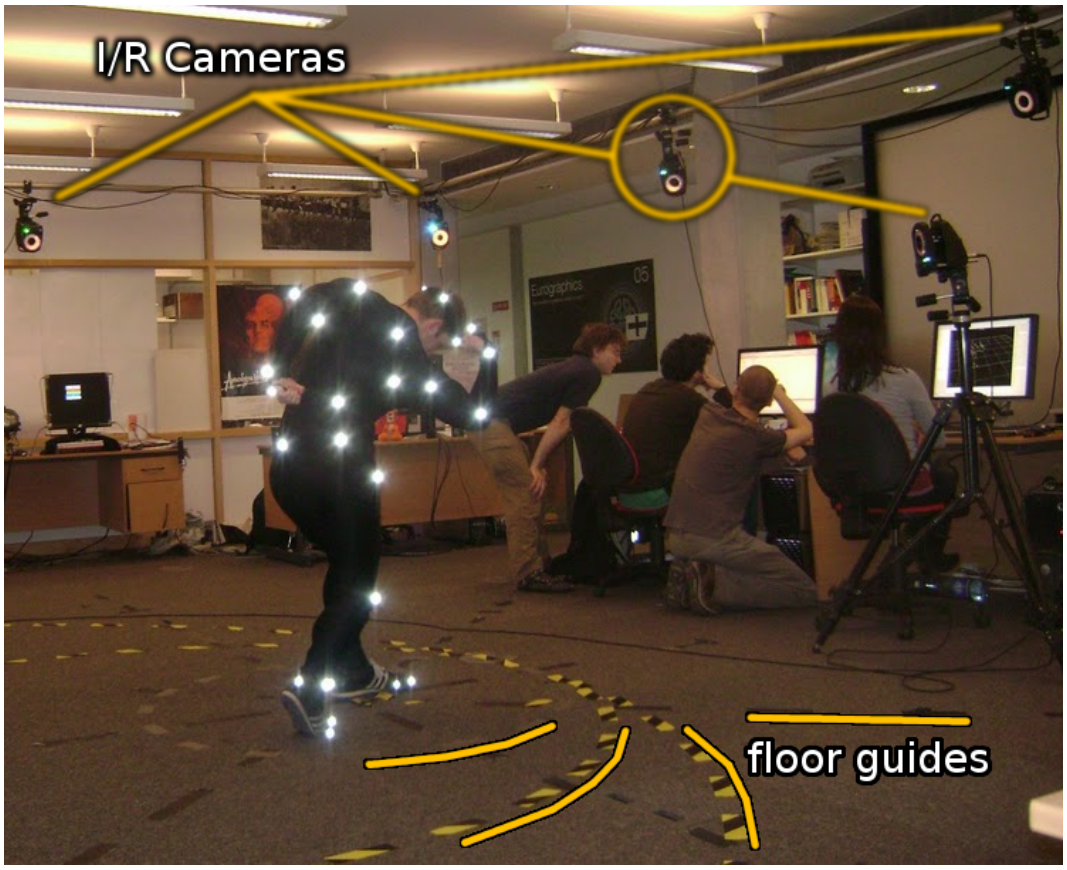

Figure 4.4: An actor (the author), covered in reflective markers, is being captured by a ring of infra-red cameras. The cameras shine highly intense infra-red light onto the actor from a variety of angles. Five cameras are visible in the figure. A 3D positioning algorithm is used to map the relative positions of the markers from each camera into a consolidated 3D skeleton; effectively recording the 3D motion of the actor. This can then be mapped onto joint positions, at equivalent locations to the markers, to animate a variety of 3D models. The floor has lines of tape to guide the actor through pre-set walks and motions. The researchers using the terminals in the background are visualising the mapping of markers onto a 3D skeleton, and are recording each animation sequence.This motion capture session was used to record some of the motions for the characters in Figure 4.4.

vidual elements can be selected and manually repositioned using keyboard and 3D mouse picking. This tool allows rapid construction of a large variety of different 3D environments that can be used for running experiments. For simulations that are used for experiments it is exceedingly useful for the computer scientist to have either a rapid scenario design tool at hand, such as the terrain creation system in the figure which can create a new simulation scenario in minutes, or to work off a larger simulation project as a research platform, such as the Metropolis simulation.

Figure 4.5: User-assisted dynamic landscape generation. Pictured is a real-time tool for deforming a 3D landscape into a desired experiment scenario by using the mouse to "drag" hills up and valleys down. The diameter and shape of depressions and hills is adjusted via the user-interface buttons. The tool can also place, rotate, and adjust environment elements and obstacles onto the landscape, and generate large randomised arrangements of obstacles to quickly create scenarios for motion-control experiments.

Combined visualisations can be quickly constructed by assembling an effective toolkit of different libraries and modules. There are now a huge variety of freely available open-source tools, libraries and high-level wrappers that offer very sophisticated visualisation technology. It is no longer the task of simulation builders to develop visualisation techniques for their simulations from the ground up, but rather to assemble a collection of these libraries and other resources that best suit the nature of their particular simulation project.

4.3 A Delta-State Video Capture Tool

One of the difficulties with research using 3D graphical simulations is presenting results to peers. It is always useful to be able to show recorded videos of selected results as recorded video clips. This considerably helps an audience appreciate both the problem at hand and understand how quantified performance results relate to actual paths of motion. A number of software tools exist for recording video and audio streams by reading either output of OpenGL, Direct3D, or final information written

to screen. Unfortunately these tools, if set to record at any sort of reasonable quality, draw a large amount of system resources (both memory and CPU) as they tend to save entire rasterised screen dumps (whole images) into memory during the recording process, which effectively reduces the speed at which the simulation can run. Real-time simulations that are slowed down start processing with larger time steps. If all the underlying simulation models do not use fixed-sized chunks of time for processing then the simulation itself is affected; most often this means that calculations for movement create different results for curved paths of motion. Slowing the simulation down also means that, even if the time steps are divided robustly, it may no longer be user active; meaning that the camera can not be manually panned around to capture the intended video. A much better solution is to build a video recorder into the simulation itself. Most rendering libraries have a "render to texture" function that captures the final screen projection after all 3D matrix manipulations and transformations have been calculated on hardware, and stores the image in memory in a bitmap format. Of course, doing this in real time, we run into the same problem. The solution devised for the work in this thesis was to build a "delta state" recorder into the simulation.

The delta state recorder, when activated, sits in-between the graphics library itself, and the interface (graphics wrapper code) in the simulation. Every time that a graphics function is called by the simulation to the wrapper code i.e. moving a scene node, rotating a graphical element, showing or hiding objects, or changing on-screen text then the delta recorder stores the unique ID of the element changed, the new values (parameters) given to the graphics library function, and the type of change or name of the function. This state information is all stored in a memory-optimised data structure. The data structure is added to the end of a queue of recorded delta states, and stamped with the simulation time that the change of state occurred. Only graphical changes to simulation state are recorded. The simulation proceeds as normal, with negligible resources used to record a list of graphical changes. When the recording process is told to stop the simulation is completely stopped (physics and audio are stopped etc.). The delta state queue is played back in reverse, and a render-to-texture image is saved at a regular time interval (any frequency can be used). A video assembly tool such as "mencoder" is then used to transform all of the output images into an MPEG video file. The advantages of this tool are that it is easy to code, can record as little or as much graphical information as is desired (for example it could be created to not record displayed user interface text or mouse cursors), does not disrupt the simulation during run time, and can output to any level of image quality and frequency.

4.4 Real-Time Data Plots and Time Tools

Another tool of use to the computer science is a real-time data visualisation tool that can overlay plots or graphs onto the display during simulation. It is trivial to display text output to the screen, which is useful up to a point, and to some extent collected data can be piped out to external programmes during run-time or analysed after a simulation has finished. There are, however, elements in motion control that are notoriously hard to debug after a simulation has finished, or by using a pre-built tool. These occasions call for creation of a custom visualisation tool. Fuzzy motion controllers use fuzzy set classification to determine the input to decision-making "fuzzy inference" tables. If this classification can be displayed in real time then we have some insight into why unusual motion behaviours might be occurring, rather than treating the system as a "black box". It was incredibly helpful to display a whole panel of custom-made graphs to the screen during experiments. Much of this decision making process happens too quickly to see what is happening, so it was useful to build in a "pause" button that was able to interrupt the time model so that the graphs could be analysed. This meant building a second simulation timer that kept the graphics running whilst the rest of the simulation stopped. Figure 4.6 shows us the graph overlay panel. A variety of different graphs are used here, including scrolling "CPU monitor" type graphs that are watching the vehicle's intended output values for velocity and steering, and a scaling-to-fit graph that is recording a figure representing the number and severity of collisions so far. But most interesting are the fuzzy set

classification graphs, which allow us to see inside the decision making process of the controller. The horizontal position of the green vertical needle on each graph shows the value of the input to the controller. This also shows which set or sets that it is being classified into. The intersection with the set function lines shows us the membership value. In real-time, armed with this overlay panel, we can now observe how quickly the transition between sets occurs, and if indeed the shape and scale of the input sets effectively represents the range of the input variable. If we pause the simulation at a problem point (e.g. when the vehicle starts to crash), then we can follow the decision-making process through and work out why the vehicle is not avoiding the collision at that point in time. The fuzzy classification process will be explained in detail in the next chapter.

4.5 Measurements of Uncertainty

After building a simulation for running experiments, and developing some tools for recording and visualising results, then the next task is to set up a framework for plotting relevant results with measurements of error or uncertainty. Results can be collected in a fairly straight-forward manner by appending performance data to a text file at regular intervals (i.e. after a simulation run has finished). Throughout this thesis we have used standard error to quantify uncertainty in results, although a standard deviation measurement could also have been used. This process can only be done after a simulation has completed a number of experiment runs.

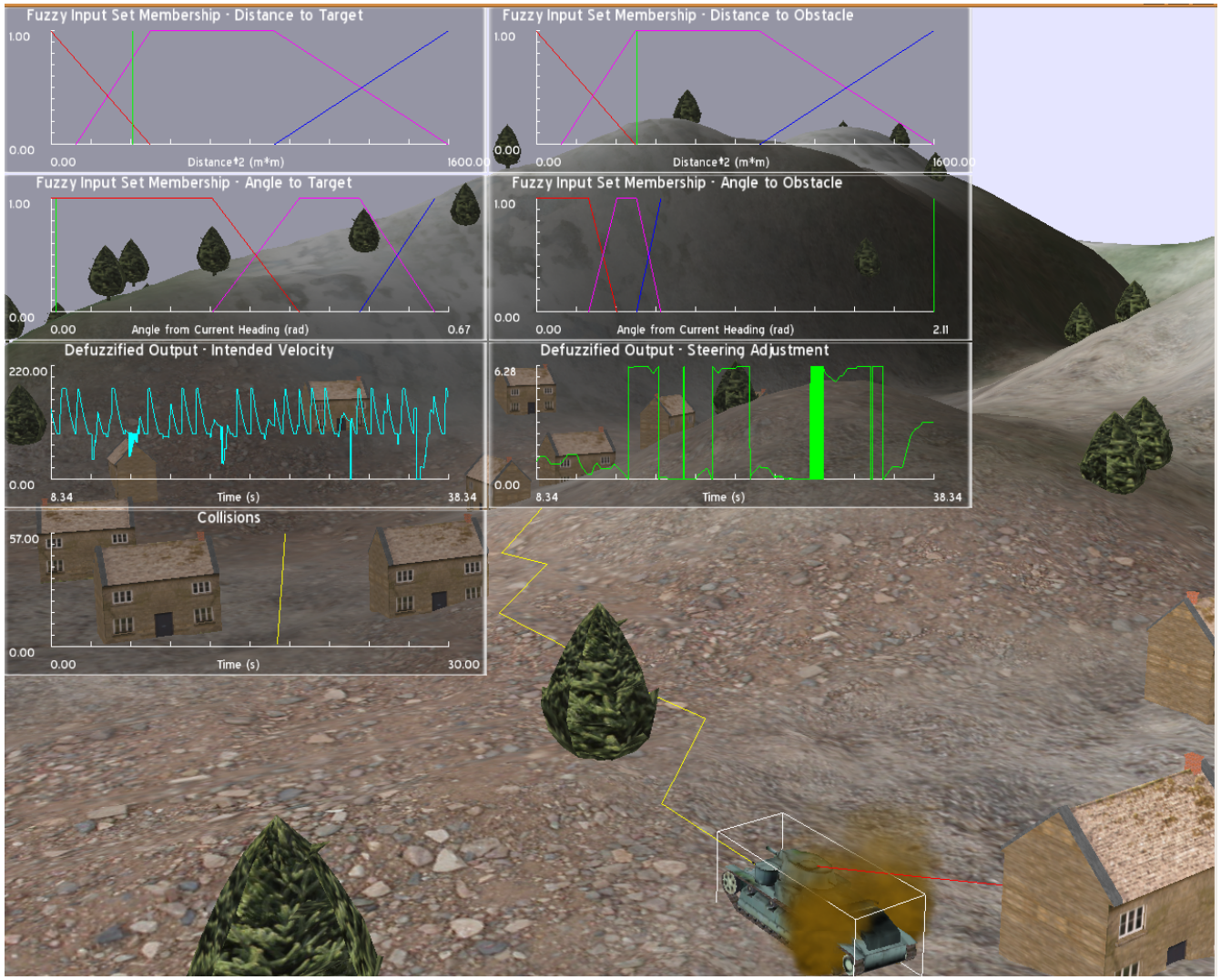

Figure 4.6: The graph overlay is custom built for fuzzy motion controller visualisation in real-time. The top 4 windows in the panel are classifying 4 crisp input values (distances and angles) of a vehicle into fuzzy sets. The green vertical needle on each graph gives the crisp value on the x-axis, and its set membership values on the y-axis. The next 2 panels are watching the desired velocity and steering output of the fuzzy controllers, and the bottom panel is collecting collision heuristics.

Chapter 5

A Modular Agent Middleware

5.1 Introduction

Many modern artificial intelligence (AI) systems, including both real physical robots and animat-based models are structured using intelligent agents. A key challenge for building large and complex AI systems is to manage the agent interactions in an appropriate architecture that supports complexity in a scalable and hierarchical manner. In this chapter various agent architectures for both physical and simulated robot systems are reviewed, and show how appropriate agent communications protocols can be developed to support artificially intelligent systems based on communities of interacting agents.

As well as reviewing some of the architectures and supporting software tools and technologies, ideas for a software architecture for managing intelligent agents are introduced. The importance of being able to incrementally augment the set of agents as new ideas are developed is emphasised. This chapter describes how key activities such as agent navigation in physical and simulated spaces; agent communication; world state management and sensory integration all need to be managed in an appropriate framework to support individual agents that will take responsibility for tasks and goals.[1]

Middleware is a software layer that sits between top-level applications and the operating system in distributed computing systems. Although the use of intelligent agents has expanded to operation in a huge variety of environments, many of the difficult but interesting problems of agent behaviour remain the same, and it becomes convenient to locate the programmed model in a smart middleware.

Whilst there are an increasing number of highly specialised jobs or scenarios for which cooperating intelligent agents are being designed, emerging from these specialised applications it can be observes that the processes and core software architecture required by these agents is often very similar. Indeed it has been found from previous works that these have often repeated nearly identically the foundation work behind agents developed for a variety of different environments; both simulated and robotic [13, 46, 53, 58, 71]. Figure 5.1 illustrates some of these different agent cases from works related to this thesis. In segment *(a)* an agent that has been developed for this drives a mechanically simulated vehicle. This agent has to operate brakes, accelerator, gear transitions, and steering to move itself optimally around a physically simulated evaluation course. Segment *(b)* in the figure gives us an example from a video game environment; here a typical computer game enemy is agent-driven and must pursue the game player's character and attack it. In this case the intelligence capacity is very limited but the agent must know how to move the zombie character in a cartoon-like manner that suits the game style, as well as presenting a predetermined level of challenge to the player and deciding when to give up the chase behaviour state when the player is too far away. Segment *(c)* shows an interesting scenario. Here each vehicle has been imbued with

[1] Material in this chapter expands original research first published as K. A. Hawick and A. Gerdelan, "Software Integration Architectures for Agents," Tech. Rep. CSTN-054, Complexity., Institute of Information and Mathematical Sciences, Massey University, Albany, New Zealand, May 2008 [70].

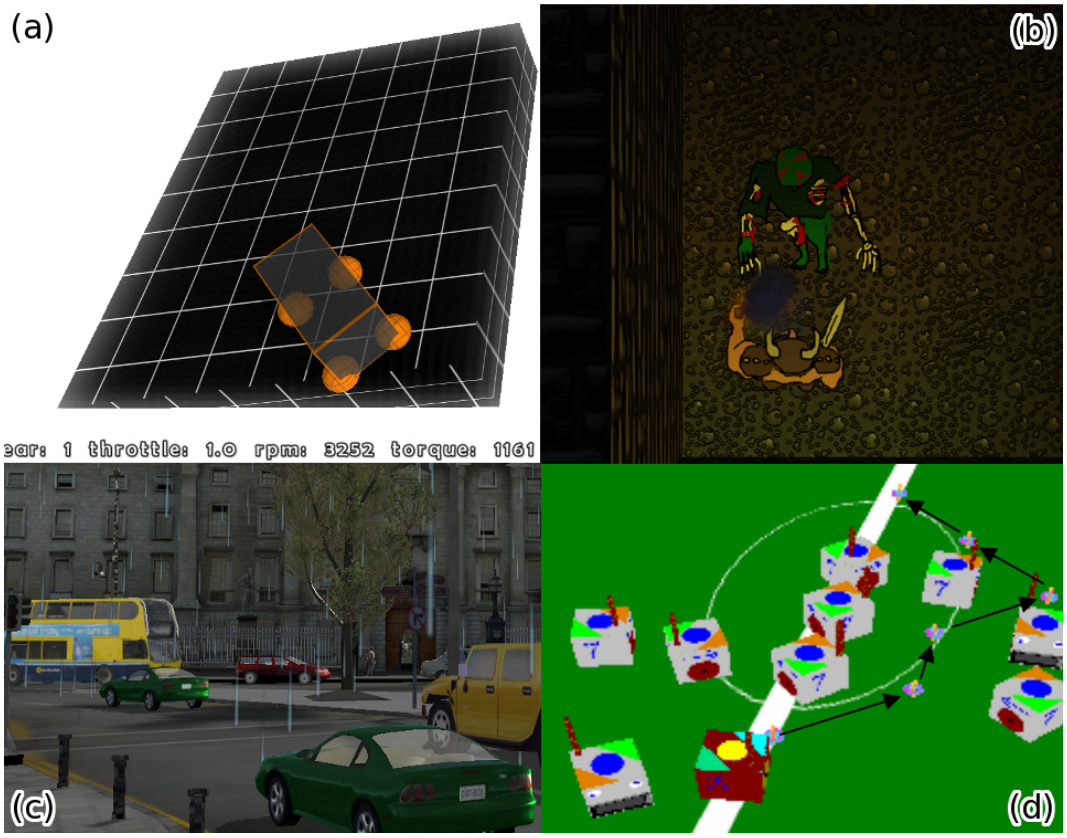

Figure 5.1: Agents being used to (a) drive a mechanically simulated vehicle with physical constraints, (b) "personify" a video game zombie enemy that hunts a human player, (c) simulate road traffic with multiple autonomous agent-driven vehicles that obey road rules and traffic signals, and (d) determine dynamic path-finding for real soccer robots' agents through a simulation. Although the tasks differ in each case, the architectural requirements of each agent are almost identical.

an agent that is similar to the car-driver from the first segment, but these agents have to cooperate as they must obey the rules of the road, which requires a different decision-making behaviour that affects navigation. As in other traffic simulations, these car agents focus on a *following distance*-type behaviour (a distance to maintain behind the vehicle ahead of it), which requires a unique behaviour. In this case the perceptual inputs to the agent are different; in addition to obstacle and destination information the agents are also given some information about the state of their current road lane (for maintaining following distances etc.), and of the intersections ahead such as the current state of red-amber-green signals. Segment *(d)* shows us a still from a robot soccer simulation. In this case an agent that combined a fuzzy-logic-based reactive motion controller with a dynamic path-planner was being developed. Here the red-shaded robot is reconfiguring its path through a large number of robots from the opposing team. The points along the path are designed to avoid the front-sides or most likely trajectories of the opposing robots. The robots themselves were real-world remote controlled cars, but the figure displays a visualisation from a real-time sand-box simulation environment in which the algorithms and agent architectures were developed before applying them to the real robots. Much of this work was published [46–48] and has been a background foundation of work for this chapter, as it was discovered there that the simulated game-like agents were mutually compatible with the complete real-world robot architecture, despite the real machines having additional requirements such as radio communication frameworks and computer vision inputs. In fact, it was advantageous to have a common architecture in different applications, as the new algorithms that were developed in the simulation could be simply plugged-in to the real robots. This is a huge time-saver in terms of development speed as it reduces the size of jobs between experiment iterations. This advantage can of course be extended to the next higher level; a common agent architecture as a middleware package that can be reused for various different types of agents allows us to simply build plug-in modules that represent more specialised agent behaviours, rather than to re-invent the wheel and spend this development time on development of infrastructure or porting of code between different architectures.

Experiments with agents in simulations, robot soccer, games, and other applications make use of agents; the requirements of which converge towards a similar model; large numbers of relatively simple agents that operate autonomously within their environment. More complex functionality is then either emergent (such as traffic flow patterns) or is directed by higher-level agents or humans (as in the case of games and robot soccer). Some of these low-level agents are intended to cooperate in a swarm arrangement, or compete with hostile or predatory agents, but all have a number of elements in common:

- A short-sighted perception of the surrounding environment
- An evaluation system to decide which action to take next
- Navigation functionality (a system that decides where to move)
- An ability to record a simple history of past actions
- A framework for interacting with other agents

Other types of agents such as natural language processors may have a radically different *modus operandi*, and require a specialised architecture, but for systems that employ agents in an autonomous or mobile context a new architecture is proposed; with an agent middleware to handle integration or communication between the common agent elements as listed above. This middleware is to be constructed in a modular fashion; where the elements of agent functionality can be swapped in or out, redefined, or extended to better suit the agent. This architectural approach should then be flexible enough to cope with a variety of heterogeneous agents within a common framework, and as will be shown this chapter, even lends itself to more complex or distributed agents.

5.2 Agent Society Model

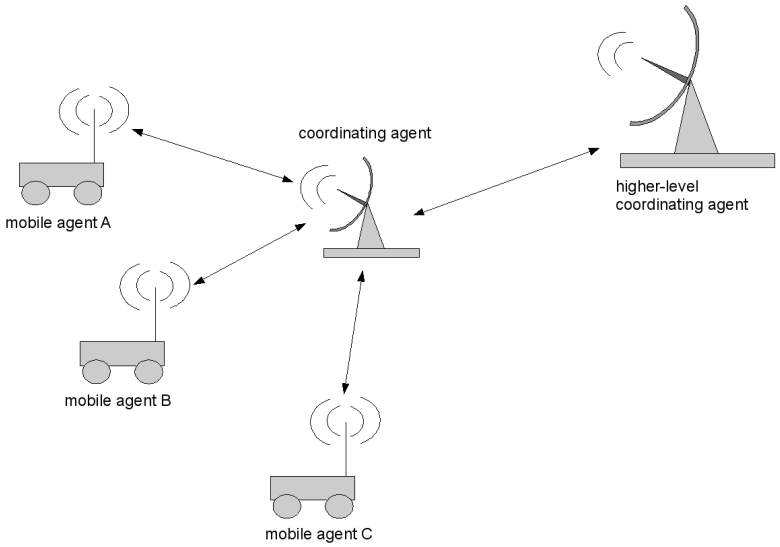

Figure 5.2: In this society of agents there are multiple levels of coordinating meta-agents. Each level of agent works at a different order of task planning; mobile agents simply move reactively from A to B as dictated by their coordinating agent, and send back locations of environment features to the coordinator so that it can populate its planning map. The higher-level coordinating agent is present to indicate the scalable nature of the hierarchy; it can have several medium-level coordinating agents to which it can issue movement instructions to whole groups.

Figure 5.2 illustrates the model for inter-agent cooperation and communication on which the agent architecture is based. Whilst recent research has looked at the possibility of peer-to-peer type communication models for agents [72,73], it has been discussed that a society of cooperating agents based around a military-style hierarchy, or *team captain* in the case of robot soccer, is convenient for adversarial games [74]. The bottom-level mobile agents are relatively non-complex, with mostly reactive behavioural intelligence capability, but act as eyes and ears for a coordinating agent to which they send asynchronous reports. This coordinating or higher-level agent processes environment information reported from its agents, and from a more complex model of the environment makes more complex forward-thinking plans relating to a common goal, and then sends *missions* to the agents under its umbrella in order to achieve best results through cooperative actions; goals which otherwise might not serve the highest interests of the individual agents. Because many multi-agent systems are constrained to an individual machine with limited cycles available for inter-agent communication and cooperation, the hierarchical model of communication also makes much more efficient use of resources for these systems. This kind of organisation is suitable for most multi-agent applications, including of course adversarial computer games, robot exploration, and robot soccer.

A less obvious example of where this particular arrangement can be advantageous is to traffic simulations. With this hierarchy we can give vehicle-driving agents only a very simple agent system that moves the cars towards their next free point along the road lane. A coordinating agent was embedded in each road lane, which is then able to take ownership of all of the cars entering the lane. The road lane agent knows the rules of that particular road and intersection set, including the state of the traffic signals, and if other systems such as pedestrians have walked onto or otherwise interacted with the road. The coordinating agent also collects the positions and other relevant information from each car on its road. With this information the road itself is able to build a small map, and

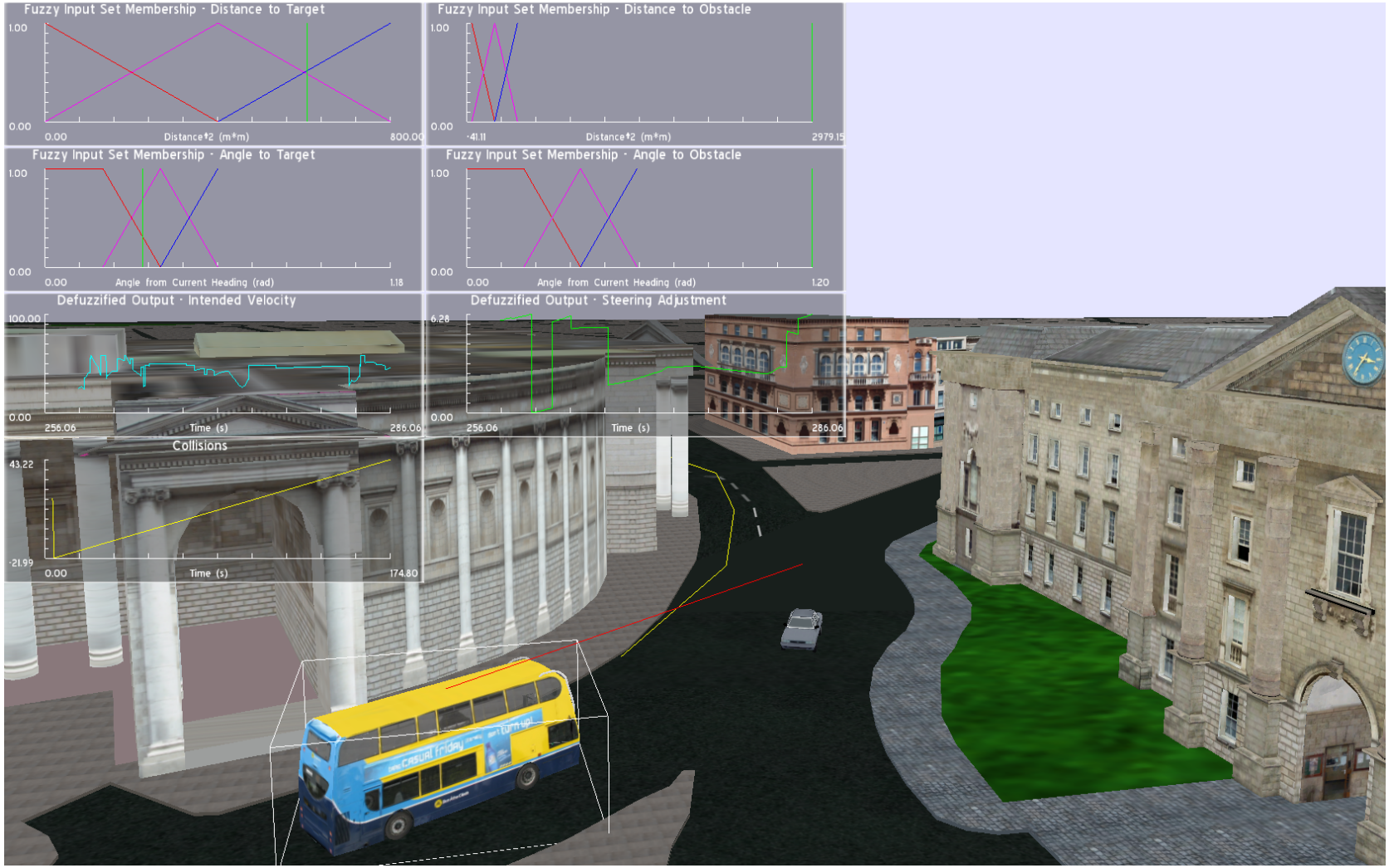

Figure 5.3: A coordinating agent embedded in the road has planned a path (yellow line) for the bus based on its abstract representation of the road; a linked list of segment along the road and the occupancy and state of each segment. The overlay shows the bus' agent handling smooth steering and that it is watching the nearest moving obstacle (red line pointing to the car) should it need to react to avoid colliding with it.

plan an effective path for each car, including following distances, stopping targets, lane changes, and intersection instructions. Most of the more complex tasks can be taken away from the moving agents and moved to a higher level. This allows us to build very simple car-driving agents that concentrate only on the controls of the car itself in respect to its planned course as given by the coordinating agent. The coordinating agents, in turn, are very simple functionaries, and treat each subordinate agent as a simple actuator and sensor set. Planning at the level of each agent can be very abstract and simplified - in the case of the road-embedded agents a linked list of nodes was used, representing regular intervals along the span of each road.

5.3 Modular Architecture

Most agent architectures are based on the traditional "stack" model, as in Figure 5.4, where input arrives at the top of the stack, and modules are arranged in order of abstraction from top to bottom; with higher-level planning behaviours preceding lower-level reactive behaviours and finally the actions, which in this case are instructions sent to the motors.

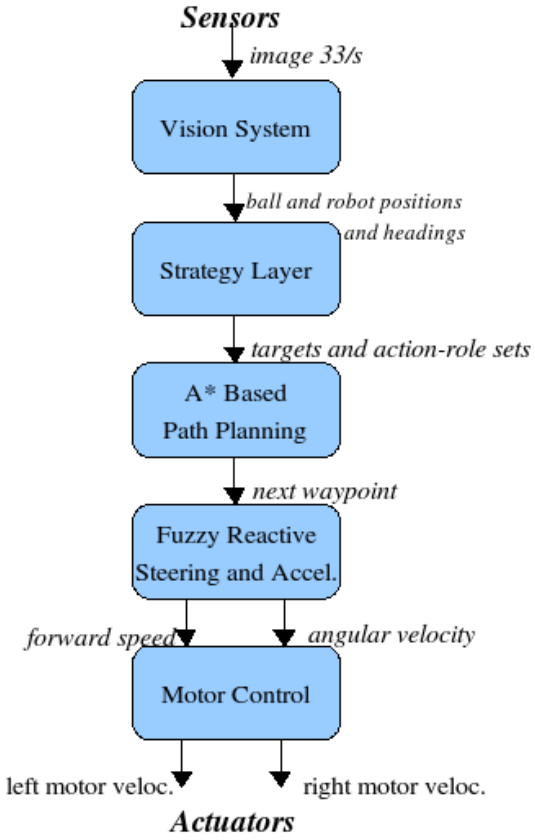

Figure 5.4: This is a typical "stack" architecture with processing in a linear order from input to output with higher-level behaviours preceding lower level actions. This particular model was first used in previous robot soccer work [46], and has to some extend inspired the modular middleware used in this thesis.

Our concept for the role of an agent middleware for the agents is illustrated in Figure 5.5. Some of the key tasks common to the agents has been identified, which have been included as tasks that will be handled by agent middleware. Each agent that is created will always require agent-specific

modules for interfacing with the unique actuators and sensors of that agent; software or hardware. It is possible for these calibration and control modules to be separated from the core of the agent, and communicate to an agent middleware *via* a common interface and communication protocol. It is not necessary to build a completely new agent architecture from scratch simply because one agent must operate physical motors as actuators, and another that employs the same algorithms exists only in a simulated environment.

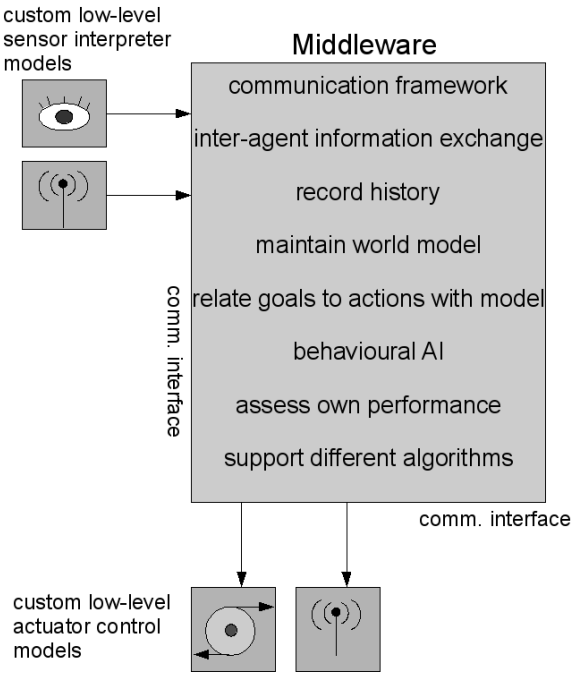

Figure 5.5: The role of middleware for agents. The actuators (output) will almost always be customised control modules, specific to each agent type, and similarly, sensor input modules will be specialised, but the large box contains a list of agent functions that are common across many types of agent. The idea is to group these functions together in a middleware layer that communicates via a common interface with decoupled sensor and actuator modules.

A progression of the agent middleware concept in this architecture is a move to modular middleware. Separating the functional components of this agent middleware into modules, as illustrated in Figure 5.6 allows us a much greater degree of flexibility in design, which is particularly useful for developing agents middleware that can be applied to various types of agent with differing functional requirements. With reference to the figure, we can consider a bus-driving agent in this traffic simulation. In this case we would write software components to determine the most likely collisions from a list of nearby vehicles and static obstacles, which replaces the LASER and camera modules and hardware in the figure. The information that we would collect would be a list of the most important 3D positions, shapes and sizes; sent to the middleware's "Environment Processing" module. We would also need functionality for receiving path-planning instructions from the road coordinating agent which would be sent to the "Mission Control" middleware module. After this perceptive stage we would need to use the action-selection "Decision Making Matrix" to determine which behaviour to use next; steering for collision avoidance or to head for the next point on the planned road route. Some of the intelligent agents are designed to learn or record a performance result as part of a genetic algorithm; these would also use the "Self Assessment" module, but this traffic simulation is unlikely to need this in normal operation so we can unplug this module from the

middleware. There is a "reporting" module, which in this case is going to tell the road coordinating agent which position on its road that is currently occupied so that it can update its planning maps. This will be a class-to-class software message in this simulation so we will write a message passing function to replace the output "Wireless Module" in the figure. The figure has a full navigation stack of modules. In the case of the bus the top level module is done by the road agent, so the coordinating agent will run this module for all its car agents, so that the bus can unplug this module and rather use the next one - the low level navigation module to compute finely-detailed steering and acceleration controls to either loosely follow, or to override its planned path and react to an immediate obstacle. The "Actuator Mapping" module is simply a method for converting or whatever symbolic output is given by the steering module, which we expect to be abstracted in terms of "steer left 60% and slow down 100%", into sensible instructions to send to specific actuators. In the case of the bus-driving agent this would mean decomposing "slow down 100%" into "set accelerator pedal to 0", and "set brake pedal to 1".

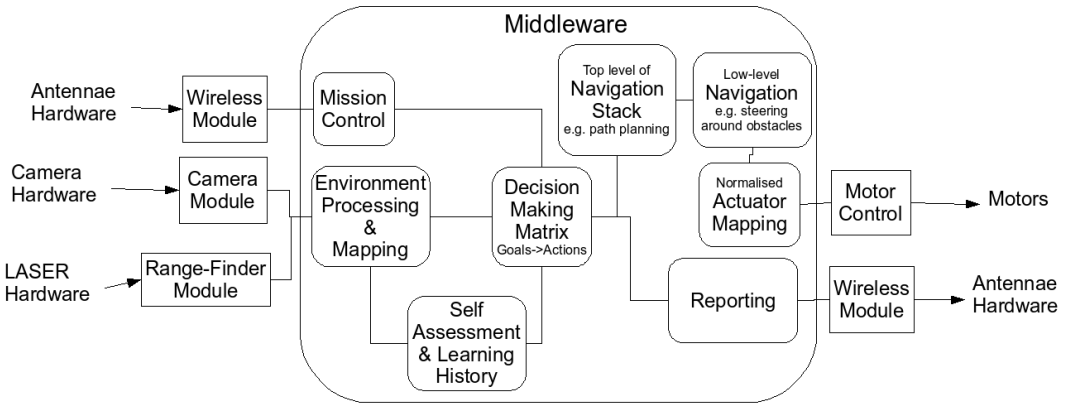

Figure 5.6: This is a fully extended model of the main modular components of this architecture as applied to a physical robot. The sensors and actuators (antennae, LASER, motors, etc.) have custom controllers and drivers, which of course would be completely different for a computer game character. Inside the middleware layer there are a variety of processing, action-selection, navigation, and communication modules, which can be reduced or added to depending on the type of agent.

Within *main-loop* style agent control programmes the tradition has been for all the different functional components of an agent to be computed once per frame. This model is not an efficient use of resources, as some of the typical agent modules such as long-term path planning do not need to be re-computed in every processing frame. If the modules of the agent middleware are created in an asynchronous, event-driven fashion then they can re-calculate on independent terms, and update their linked modules only when new input data has arrived, or according to a different throttling rate for each module. An asynchronous module design with common interfaces allows us to spread calculation load over many processing cycles.

A modular architecture also allows the designer of an agent to replace any of the modules with some of the designer's own code, or with a module from another agent. With an object-oriented implementation the designer should even be able to extend or override the core functionality of a module to add some more specialised functionality to the agent.

This semi-independent modular approach also gives us the freedom to distribute the agent middleware over several threads or multiple CPU cores, or by logical extension to distribute the middleware for a single agent over geographically separated machines as agents become more resource-hungry or require greater internal redundancy. This, of course depends in the implementation. In this thesis most of the applications contain large numbers of agents being simulated on one machine,

so the benefits of distributed computing diminish.

The architecture that has been designed here is scalable in two senses; firstly, the modular nature of the agents allows upgrade or downgrade in complexity. Because the modules themselves can be constructed to communicate asynchronously, through a common interface, this architecture allows entirely new modules to be included within the architecture. An example of where this approach is most useful is the navigation stack of autonomous agents; which can typically comprise several different systems for long distance map-based planning, short distance path-finding, reactive obstacle avoidance and target seeking, and several levels of interface with actuators. The number of systems in the stack is largely up to the designer of the system, and it is definitely beneficial to offer this kind of flexibility to designers of agents, or to allow agents with different levels of navigation complexity to operate under the same architecture.

Secondly, the society of these agents itself is scalable; if there are large numbers of agents then coordinating agents can be stacked in a hierarchy - coordinating agents control a group of agents as actuators, and report to a higher-level coordinator as its sensors (see Figure 5.2). Each level of coordinating agent deals with a model of the environment at a different resolution and sends and receives information at a different level of abstraction. Each agents knows only about the agents directly above and below it in the hierarchy. This model minimises conflict between cooperating agents as it scales in size as agents at each level can be given missions that avoid competition (or collision) by the level above.

5.4 Modelling the Environment

The perceived world is plotted on a 2D map which is divided into map references or cells. All elements of the environment in the agent's word model are then said to occupy a discrete map reference, rather than an in-exact position expressed in floating point or real numbers. There are several advantages to this approach:

- The world model can be stored in 2D arrays.
- Reduced error when communicating positions.
- Ease of computation for common algorithms.
- Ease of pattern identification.

A 2D map is a very simple tool for communication - it is usually not necessary for agents to communicate very accurate world information; high-precision information is usually only required for very low-level planning. Using integers to express positions as discrete map cell values is also very convenient for communication over a broad range of network implementations, as they are guaranteed to be exact.

The resolution of the map is obviously important; lower resolution maps are easier for planning, but are at risk of losing important or subtle data. Maps of higher resolution contain more information but are at risk of bogging down algorithms with unnecessary information, over-sensitive grid positions for moving objects and exponentially expanded search domains. Previous experiments have shown that an ideal map cell size was roughly 150% of agent size for Robot Soccer agents requiring very accurate navigation [46].

A key advantage of using a world model as simple as this is that the thoroughly tested and proven algorithms of computer science can be applied to good effect. The A* Algorithm or other real-time adaptations and hybrids can be used for very effective path planning. Common pattern identification and image recognition algorithms can be applied to the map for use in higher-level AI. At any time the map can be quickly transmitted as a human-recognisable image. Lossless compression algorithms can even be applied to the world models in the same way as they would an image, for more efficient transfer of very large or detailed world maps.

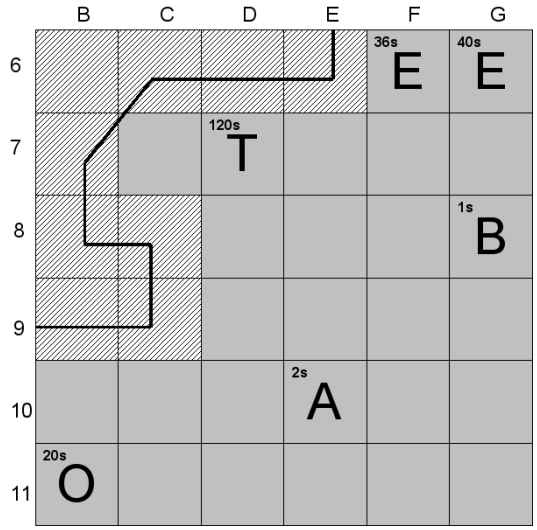

Figure 5.7: A typical Wumpus-style world model. The 3D environment can be abstracted into a 2D map as here, which allows powerful search algorithms to solve complex problems such as path-finding in a simplified environment. These kind of maps can be kept in 3D arrays, where detected environments elements can be stored, represented here by different characters, and time-stamped so to allow for information decay in stochastic environments, as depicted in the corner of those cells. Various cells have been shaded out of the image as a wall (black line) has been discovered. We can see that normally this kind of obstacle shape, which has local maxima traps, would be a problem for motion controllers, but degraded into squares it becomes a trivial path-finding problem for an algorithm such as A*.

Figure 5.7, which will be explained in detail shortly, shows the similarity between this sort of world model and the famous Wumpus [75] problem. The aim is to take what may be very complex 3D world information from the agent's sensor modules and convert this into a very simple view of the world - in much the same way as humans would - the complex problem is simplified or abstracted, then the simple (abstract) problem is solved, and applied back to the complex domain. Agent planning is done using this simplified and abstract model of the world, making high level plans such as "from our position move around this map cell to get to destination map cell" which are then interpreted by an intermediate control module into low-level complex outputs which can be fed to actuator control modules.

The main requirement in order to harvest this sort of high-level planning ability is a system for converting to and from complex and map coordinates. This system depends on the type of environment that the agent is operating in; systems have already been developed for robots operating in the physical world for exactly this sort of mapping and planning. Carnegie Mellon University's Lunar Rover gives us a very good example of this sort of system; and provides details for a stereo-vision camera system to identify positional information of terrain features, and plot them on a similar type of map for later use as a navigational aid [76]. Simulated agents may already operate in this sort of cell-based environment, but those that operate in a more complex 3D space simply require a system for agreeing on map cell placement, which was developed in earlier work [46].

In reference to Figure 5.7, we can consider the agent marked 'A' at map index (E, 10). The superscript in the top left corner of the cell indicates that this information was last updated 2 seconds ago, which shows us that we can actually store several layers of useful information in each cell, and also indicates that this sort of planning is useful in a dynamic and uncertain or stochastic environment; world model information can be date stamped, which can assist high-level planning modules in the assessment of certainty or out-of-date information. This type of information is useful in identifying a pattern of nearby hostile agents as being actually only one or two moving agents that have been reported multiple times, or indicating if a segment of the environment containing dynamic obstacles will no longer be reliable for path planning. Again in the figure, the agent has been informed of, or spotted first-hand, various other environment elements with their own date stamp. Each of these elements has been categorised with a different letter; this kind of very simple discrimination is useful for fast-paced decision making such as real-time path planning, assessing danger levels, or roughly categorising the possible speed of dynamic obstacles. A large segment of the map is also shaded out completely - a wall has been discovered and all of the cells that it covers or mostly covers have been removed from the navigation search domain. This is a blanket decision, but will reduce the load on long-range path finding. Lower-level navigation behaviour may actually move the agent through the parts of these map cells that are not impeded if necessary. We can see that in this way the complexity of the simulated world has been drastically reduced into a model not dissimilar from the world of the Wumpus, and for good reason - the powerful algorithms that have been developed for Wumpus-like problems can then be applied to complex domains.

In the all-terrain simulation pictured in Figure 5.8 the coordinating agents built a 2D grid-based map, represented as an array of characters, which was filled in with characters representing relevant obstacle positions as the mobile agents discovered them. Despite the coordinating agent treating the system as a tile-based environment, the lowest-level agents are able to steer smoothly through 360 degrees with freedom. An advantage of this approach is that vehicles of a larger size and shape were able to be coordinated in tandem with the smaller characters by utilising coordinating agent maps of different configurations. Another advantage used in this implementation is that several layers of 2D planning map can be used; a map of obstacle locations, a map of potential goal locations, and a map of different heights of terrain as weighted numbers to use in planning movement; all of which can be built as the mobile agents gather more information.

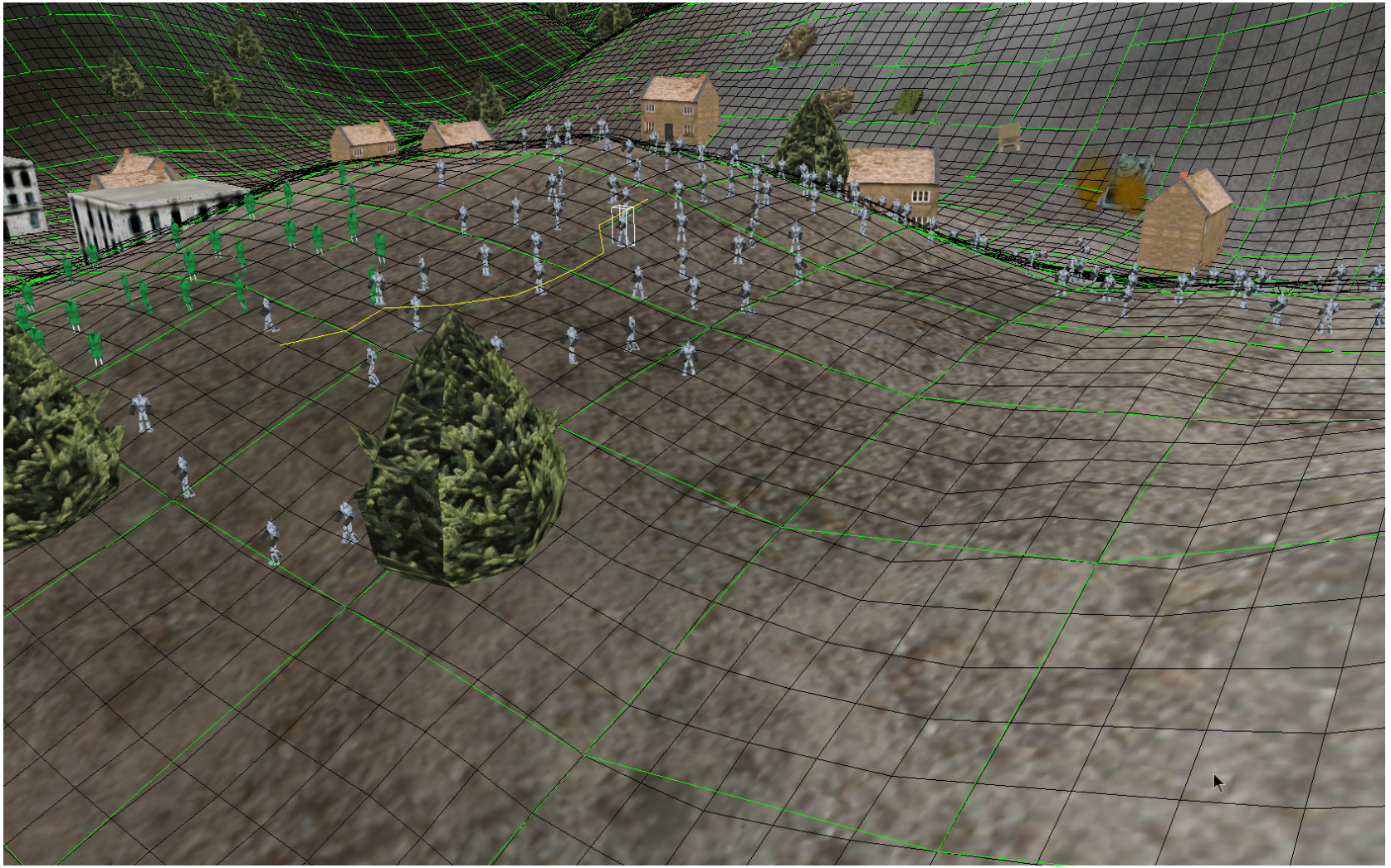

Figure 5.8: Here we can see large groups of characters which are controlled by a 3-level hierarchy. Mobile agents steer reactively, but loosely follow a path (shown as a yellow line) planned by a team-coordinating agent. The coordinating agent uses a 2D cell-based overhead map to plan this path, shown as high-resolution black grid overlay. Larger groups of agents are directed by issuing more abstract general paths to coordinators with the lower-resolution green grid.

5.5 Agent Behaviour

An agent navigation system within this architecture typically starts operation on the agent's 2D world map. High-level algorithms such as A* Algorithm or modifications such as Hybrid Evasive Fuzzy A* [47], Dynamic A* [77] (D* Algorithm) or other search algorithms can compute useful long-distance or mid-range real-time planning information here, based on combined heuristics assembled from various layers of the Wumpus-style environment maps. The Fuzzy A* algorithm can go so far as to generate forward-thinking probabilistic regions of *undesirability*, and factor these into a balanced path-finding algorithm, so that mobile agents can take a shortest path, except avoiding areas that are quite likely to be undesirable, for example a tear-drop or wave-crest shaped region surrounding an approaching heavy vehicle. An example of this algorithm in simulation is depicted in Figure 5.9, which gives us a way of visualising this extra dimension of information in each cell. In the figure the navigating robot will choose the most overall downward-sloping path through the graph, which is effectively a balance between the optimal path according to terrain, with a weighted consideration given to other environment features such as the trajectories of other vehicles.

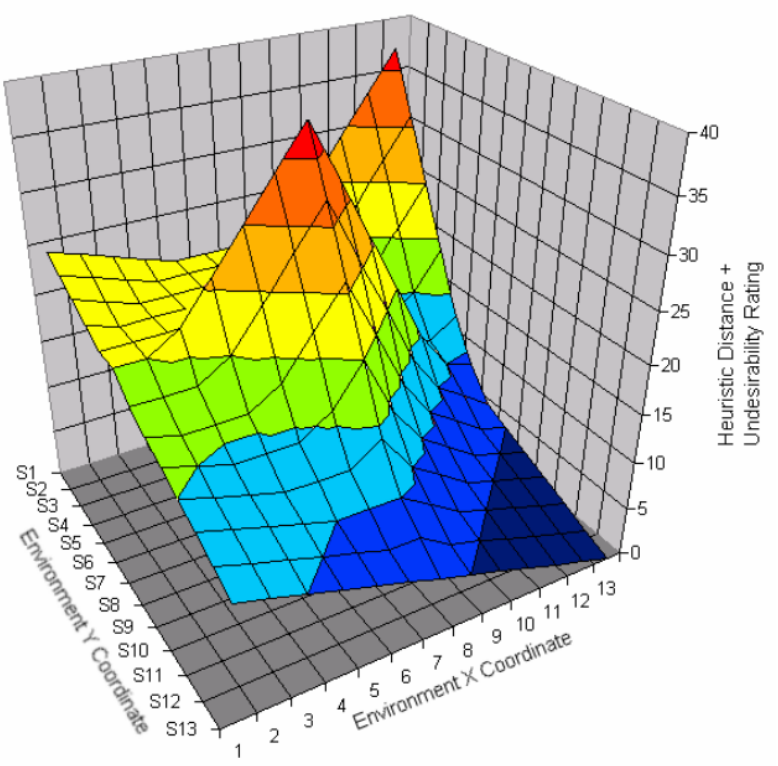

Figure 5.9: This graph visualises an example of the the Fuzzy-A* algorithm in action. The height of the graph points represents the cost of movement in that area - which is a weighted balance of distance to final destination, terrain movement cost, and trajectories of other vehicles. The algorithm will then choose the most overall downward-sloping path to the destination.

Because the navigation systems that are examined in this thesis generally operate dynamically in real-time, generally only the *next* way-point in the planned path is fed down from the planning layer to the next layer in a navigation stack. That next layer takes care of more subtle details, but attempts to direct the agent's motion as efficiently as possible, towards the next way-point. In the case of the Fuzzy A* algorithm, this next layer is more complex; and actually computes a Fuzzy Logic-based

reactive obstacle avoidance steering and acceleration system combined with a Fuzzy Logic-based target seeking system. This kind of control gives the agent a reactive navigation layer, and can quite sensibly control analogue mechanical actuator systems. Subsequent modules in the navigation stack would then be required to translate the generalised steering and acceleration outputs into specific motor instructions.

The key function of an intelligent agent is the decision-making or action-selection process; how the agent maps its goals to actions. Non-complex agents such as animats [28] are typically provided with a list of possible actions, and either a procedural or heuristic method for ranking these actions in order of priority based on information held about the current state of the environment, and any other information held by the agent. This model should be acceptable for simple agents, such as the mobile or bottom-level agents in the society hierarchy.

Higher-level coordinating agents deal with a large database of different types of information and can manage a large number of actuators - their subordinate agents - simultaneously. In addition, agents of this type need to make decisions based on a large number of variables; positions and states of agents, emergent patterns drawn from environment information, a history of past actions, results of past actions, quantified uncertainty of environment information, and goals received from higher-level agents.

The decision-making task of these agents is significantly more complex, and if based on the traditional heuristic and procedural methods is extremely difficult to develop as it results in staggered logic-gates (or very lengthy if-then-else conditions). Decision-making in complex environments means choosing a single definite task based on thresholds taken from non-discrete variables. This becomes very difficult to balance, and it is not ideally suited to real-time operation as decisions tend to flicker when variables are close to threshold levels. A more suitable decision-making model for complex agents has therefore been designed.

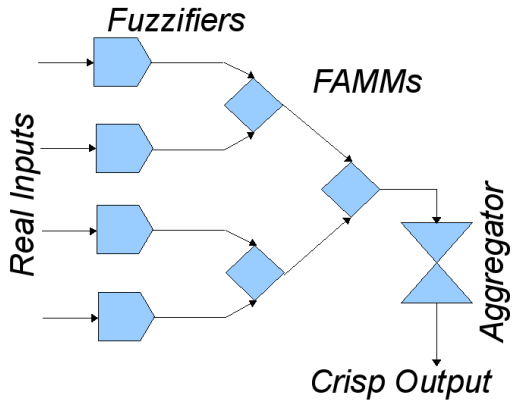

Figure 5.10: This staggered arrangement of inputs and rule tables uses fuzzy controllers to choose actions. Fuzzifiers convert real input data into fuzzy logic partial truth values. Each decision-making gate (FAMM) requires a small rule table to be written by the designer. The output actions are blended smoothly between transitions using an aggregation process.

The fuzzy controller approach to decision making was originally developed to aid robot soccer strategy [74], to take advantage of its very high degree of flexibility. It is possible to incorporate a multitude of input variables into a fuzzy strategy layer with no added complexity to the decision making process. Multi-dimensional decision arrays no longer have to be considered, and can sensibly interpolate unknowns by aggregation of fuzzy outputs. If pairs of real inputs are analysed using fuzzy associative memory matrix (FAMM)s, fuzzy outputs can be passed (without defuzzification) as inputs to subsequent FAMMs. Figure 5.10 illustrates this approach.

It is possible to analyse a very large number of input variables in this manner, and that due to the nature of the fuzzy process, that this would consume very little computational time because it only considers one 2D FAMM at any one time. The inputs converge to a final 2D FAMM, and a single, crisp output value is produced through the aggregation procedure.

The critical task of the agent architecture is matching goals to actions, and a fuzzy controller module is used for this so that it can be left up to the designer how to arrange the different rules. On the surface the structure is very similar to large if-then-else blocks, but at the kernel of this approach is the fuzzy controller which smoothly blends between output states, rather than flickering or stepping between output states as decision thresholds are arrived at. The design process itself is open to introduction of new rules or input variables with this arrangement. New fuzzifiers and FAMMs can be clipped onto an existing arrangement. When more than two outputs meet the final FAMM then a new level of FAMM must be added onto the end of the arrangement. The action-selection method can therefore be classified as a *cascading* arrangement of fuzzy controllers.

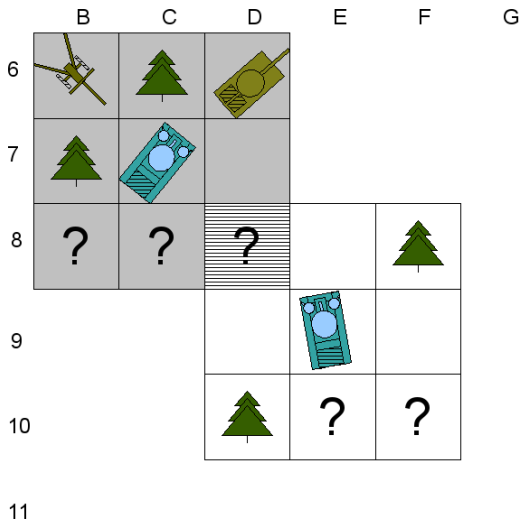

Figure 5.11: Two cooperating tank agents in a game-like environment are at C7 and E9. Under a peer-to-peer society the agents would be unable to make a tactical manoeuvre that might put one agent at risk, even if it were to the benefit of the group. Under the hierarchical society a third, coordinating agent is able to order each tank to surround the enemy tank at D7, despite making one tank vulnerable to the artillery at B6.

Figure 5.11 illustrates a typical agent coordination problem. In this example, we are dealing with a battle-field game or simulation, where two tanks are controlled by cooperating agents. The first agent is at map index (C, 7), and the second agent at index (E, 9). The agents each have a limited model of their environment, covering 9 map cells, but they are in close enough proximity to each other that there is some overlap of their neighbourhood models, which is indicated with a hatching pattern. In a robot system one of the agents would decide that it is the leader based on the order of a unique identifier, and then initiate the creation of a coordinating agent on its own internal hardware, which might then be further distributed between agents to increase redundancy. In the illustrated example, however, we are dealing with a simulated battle-field, so a fixed chunk of system resources can be designated to a new coordinating agent. If each agent's system knows how to generate a new coordinator, then no new hardware would be required, and a coordinator could be quickly rebuilt if the hardware supporting it is destroyed or disabled.

Let us assume that the agent at position (E, 9) in figure 5.11 has created a coordinating agent. The agent at index (C, 7) has spotted two trees, an artillery piece, and a hostile tank, so it communicates

the map index and type for each of these features in a report to the coordinating agent. The first agent does not know anything about the terrain behind it, which is indicated on its map by question mark characters. The agent at position (E, 9) has spotted two trees, and sends this information to the coordinating agent. It has also observed position (D, 8), which was unknown to the first agent, and so it communicates this information the coordinating agent also. We can assume that index (D, 8) has been confirmed to be featureless terrain, and so this then sent to the first agent, which can then reduce its "wariness" or threat rating on this side of the vehicle, and concentrate more of its defences in the direction of the spotted threats.

By this stage the coordinating agent has built a map modelling the environment, with features covering much of the area between (B, 6) and (F, 10). The coordinating agent has time-stamped the information as it is received from the tank-controlling agents, which it considers to be its sensors. In a common peer-to-peer type approach, where cooperation is usually defined loosely as a sum of actions based on individual interests or "bids" [72, 73], both tank agents could have swapped environment information between each other and then made independent decisions; the first agent would probably have retreated or attacked the artillery piece, which was more threatening to it than the tank, and the second agent may have moved to engage the hostile tank. The outcome of these individual decisions would have meant that the hostile tank had more time to move into a better defensive position, and move its' heavier frontal armour to face the second tank and minimise its vulnerability, essentially evening the odds between forces. Under this architecture, however, the coordinating agent has identified that the key target is the hostile tank, and directs both of the cooperating agents to attack it from two sides - maximising the chances of destroying it, although putting the first tank at greater risk. The overall outcome, however would then move the advantage further to the side of the cooperating agents; a tactical decision that could never have been made by true peer-to-peer cooperation, which are not able to make decisions which are bad for one individual, but good for the group, or indeed any sort of higher-level tactical decision.

In other sorts of environments, cooperating agents can be used to exchange a large variety of environment information in the same manner; map cells can be used to store obstacles, terrain heights, terrain types or conditions, vehicle locations, agent locations, stages of planned routes of other agents (to help avoid conflicting agent paths), and a variety of rôle-specific information that lends itself to be discretised at the same resolution as the environment map. Retaining the simplicity of the environment maps is the key to their usefulness, as they can then be easily and quickly looked-up and communicated between cooperating agents.

The agent architecture has been developed with a lot of redundancy in mind. This is particularly useful for a cooperating society of agents with unreliable communication, robots prone to battery or equipment failure, for agents distributed over a number of physical machines, or for agents that can literally be destroyed. Distributed agents should have some redundancy mechanism for regenerating lost modules on new hardware. The bottom-level mobile agents should be able to operate autonomously, even if no coordinating agent is present. If a group of cooperating agents loses contact with their coordinating agent, or if the mobile agent hosting the coordinating agent is destroyed, then the mobile agent with the highest unique identifier should generate a new coordinating agent, with only short term loss to environment data, but a possible loss of history information. This system of redundancy remains true for higher level agents as well; if a group of level 1 coordinating agents can make contact then the coordinating agent with the highest unique identifier value (inherited from its host) can designate a bottom level agent to host a level 2 coordinator.

5.6 Summary

This chapter moves towards a complete architecture for autonomous agents that communicate, cooperate, and can be coordinated by a hierarchy of special coordinator agents. Because of the modular architecture based on independent but communicating components these new agents can be distributed over multiple cores and even geographically separated machines, which means they can be

built with commodity hardware, or make use of existing idle resources. This new architecture introduces a more effective model for multi-agent cooperation, and allows not only tactical decisions to be made in a coordinated manner, but enables cooperative actions that benefit the objectives of the whole group, rather than simply serving a large number agents' individual goals. This architecture is modular, easy to customise, applies to a large range of agent societies, and introduces distributed agents and redundancy to agent components.

5.7 Possible Extensions to this Architecture

Theoretically, *factory* coordinator agents could be created like a virus (in the sense that it is not dependent on any physical machine itself but can exist in a transient state within a communication framework) that have the role of generating agents from pieces, using existing resources that it discovers on the fly - true virtual machines. Grid or cloud-enabled agents of this type would then provide massive redundancy to agents, so that even if its actuators and sensors were disabled the agent could resume operation by communicating with new resources. For example; an automobile-controlling agent that for some reason can no longer communicate with its cameras and range finders might then be connected (by the factory agent) to the cameras of nearby vehicles and road-side sensors and continue operation with all of it's existing modules for steering and obstacle avoidance intact, continuing navigation.

As agents become more resource-intensive the need to distribute agents grows, particularly with real-time agents that require large chunks of CPU time and process several tasks. A modular approach lends itself to a distributed system, and to some degree can be designed to work asynchronously. In this way the agent can exist within the Internet or cloud network, as long as it knows which modules (or resources) to communicate between. Distributed intelligent agents may be the next evolution in this architecture, seamlessly taking advantage of any resources that are made available over complex networks whilst retaining the functionality of a cohesive agent machine.

Chapter 6

On Design of Automatic Calibration Systems

6.1 Introduction

Intelligent navigation and path-finding for computer-animated characters in graphical 3D environments is a major design challenge facing programmers of simulations, games, and cinematic productions. Designing agents for computer-animated characters that are required to both move intelligently around obstacles in the environment, and do so in a psycho-visually realistic way with smooth motion is often a too-difficult challenge - designers generally sacrifice intelligent navigation for realistic movement or vice versa. Presented in this chapter is a specially adapted hybrid algorithm as a viable solution to meet both of these challenges simultaneously. The application of this algorithm to animated characters and outline our proposed architecture for automatic tuning of this system is discussed.[1]

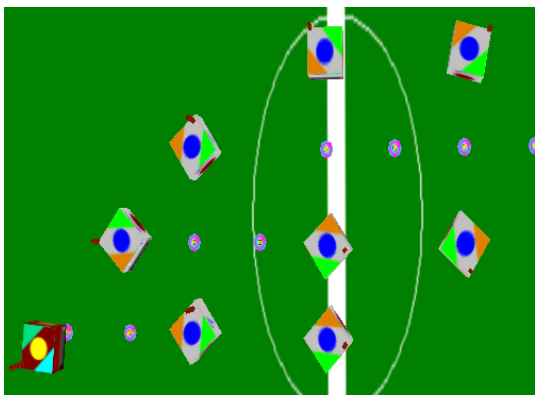

Figure 6.1: The 3D robot soccer simulator with robots controlled by the Fuzzy A* algorithm. The darker coloured robot at the top of the figure plans a path through moving robots. Blobs indicate the points along the current plan for the robot's path. The path itself is dynamic and will change depending on the positions of moving obstacles. The actual motion along this path is performed by the smooth-driving and reactive Fuzzy navigation system.

[1] Material in this chapter is based on original research published as A. Gerdelan and N. H. Reyes, "Towards A Generalised Hybrid Path-Planning and Motion Control System with Auto-Calibration for Animated Characters in 3D Environments," Advances in Neuro-Information Processing, Springer Verlag LNCS, vol. 5507, pp. 2528, November 2008. [55]

Previous works have focused on the development of next generation navigation algorithms for soccer robots by combining existing navigation and motion control algorithms into hybrids which exploit the benefits of the individual algorithms but also negate their limitations. To this end previous works developed the Hybrid Fuzzy-A* [46] algorithm which can perform both efficient and dynamic route planning in environments with static and moving obstacles, and also move using smooth-path motion and avoid obstacles at very high speeds, requiring only a very small allocation of CPU resources as robot soccer teams are typically controlled by one commodity desktop computer which also has to process computer vision and direct 3-10 other robots every 1/30th of a second. Several variants of this hybrid were created which enabled us to incorporate other properties into robot behaviour [48, 74] - in particular, an ability for the ball-carrying robot to *evade* hostile robots on its path to the opposition goal and pre-empt their path of movement as well [47]. The success of these works, and also the impressive appearance of this algorithm operating in 3D graphical simulation (see Figure 6.1), lead us to speculate whether the algorithm could be applied to a range of much more demanding applications beyond the mathematically simple 2D realm of soccer robots.

For this chapter a real-time simulation engine has been constructed using the Ogre graphics library [68], which has been used successfully for military visualisation and simulation projects [78] with success, to mimic a typical computer game-type environment as a proving ground for the application of hybrid robot soccer control algorithms to much more complex 3D environments. This type of environment also typically has limited CPU available for artificial intelligence computation as 3D graphics processing tends to occupy the bulk of resources [79] so the need for efficient algorithms is paramount, and can process sensory information with large amounts of environment noise or complexity [80] - the kind of environments where fuzzy logic is ideal.

The Hybrid Fuzzy A* algorithm has been adapted so that it is scalable and can be applied to a range of animated characters with their own specific motion characteristics. The initial implementation has been successful with vehicle-type animated characters - see Figure 6.2.

Due to the complexity of manually tuning the many algorithm parameters for optimal results an architecture for automatic parameter calibration has been designed so that the system can self-train for application to a range of new characters and environments by operating on a series of obstacle courses - typical operating environments created by a designer for training - and self-evaluating. Also detailed here are some new visualisation techniques, which were found necessary for visibility of complex multi-level artificial intelligence systems in real-time applications; as post-run analysis of animated character motion is an almost impossible task.

6.2 Hybrid Algorithm

The hybrid fuzzy A* algorithm is arranged in a cascade of systems. At the top level, the agent is given some information about the obstacles and it builds an environment obstacle map. The environments that are being considered so far can be simplified and expressed as a 2-dimensional map, and are stored as a 2D array of values. More complex path-planning might require a 3D array of values, but none of the simulations looked at in this thesis have been sufficiently complex movement domains to warrant investigation of this possibility. The simulation is using an array of character symbols to store the obstacle map, as this allows us to express a range of obstacle types with single letters ('*s*' for shrubs, '*b*' for buildings). A requirement of the system is that the agent has some ability to divide the environment into a graph or searchable area. The agent then translates known obstacle positions into graph indexes and marks the locations. In an environment with many dynamic (moving) obstacles this map is recomputed with every frame of calculation. Larger obstacles can occupy multiple graph cells.

The character will ultimately make use of several types of *environment map* as weighted search domains for a *depth-limited dynamic A* Algorithm* (depth limited to scale to available CPU window per application) to compute a a near-optimal, partially complete path to its target destination. Graph nodes marked as containing obstacles will be either excluded from the search domain (impassable

Figure 6.2: An animated character - in this case a to-scale model of a Soviet T-28 circa 1932 with simulated real physical operating characteristics - travels through a 3D-environment in the simulation engine complete with hills and a range of obstacles. The vehicle has planned a prospective path around obstacles using the A* component of the Hybrid Fuzzy A* algorithm (as indicated by the segmented line visible ahead of the vehicle in the figure) and has detected and is driving around the *nearest obstacle* on its right hand side (as indicated by the second line visible in the Figure) - in this case a house.

obstacles) or given a weight based on their resistance to the character's movement (for example shrubs may be given a moderate weight modifier for heavy vehicles in a military simulation or game, as they can be driven over, but buildings may be impassable and excluded from the search domain). This path is broken into *way-points* and the first or an early-stage way-point (if the graph resolution is very high) is used as the *current way-point* and given to the Fuzzy navigation system as *target location crisp input*.

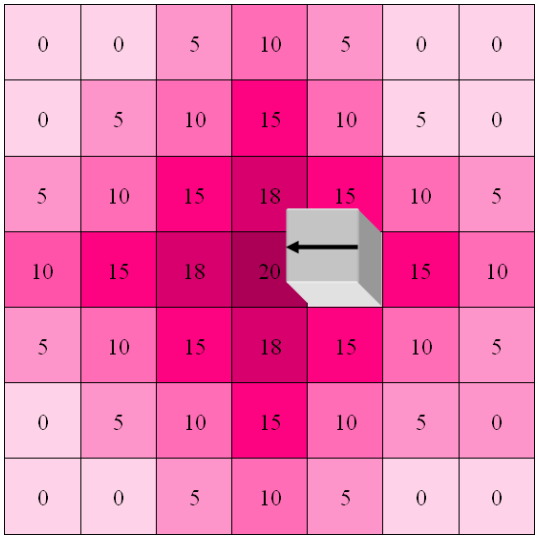

Figure 6.3: Representing combined heuristic weights of cells in the environment graph. The A* Algorithm will choose a path between grid cells minimising overall distance cost + any other balanced heuristic.

Experience with robot soccer path-planning [47] has shown us that the A* Algorithm's heuristic can be modified to incorporate many different weighted inputs into the path-planning decision process. A "heat" environment map is used to indicate areas of undesirability in the path of hostile robots, so that the robot would be less likely to favour those locations even if the distance would be slightly longer - inculcating an evasive property into the robot path planning process. Figure 6.3 gives a graphical representation of this heuristic balance used for evasive behaviour in the robot soccer system [47].

Factors such as slope of terrain, and known condition of ground surface can be weighted as proportions of the heuristic. Prudent designer decisions are required here to choose heuristics that are meaningful in a particular 3D environment, although the best ratio of weights of each input could be determined by self-training simulation runs. These heuristics may be tailored to a particular vehicle or re-evaluated periodically to adapt to changing conditions; reflecting the value of a true dynamic A*, as opposed to a less flexible variant such as D* [81].

The Fuzzy navigation layer is a rapid calculation system that takes the *current way-point* on the path created by the A* layer as a rough guide. The systems can disagree - for example if the A*-generated path dictates to move left around a tree, but the vehicle is actually already moving around it to the right - then the path will be recalculated in the next frame of calculation. The fuzzy system consider two groups of crisp input - the *distance and angle to the nearest obstacle* and the *distance and angle to the current way-point* see Figure 6.4 for an illustration of this classification. These two groups of inputs are fed into two distinct fuzzy set membership functions and fuzzy associative memory matrices. The fuzzified outputs for obstacle avoidance and target seeking behaviour are then be blended together using a centre of gravity function to produce smooth transitions between the two behaviours.

Navigation outputs are, in the adapted hybrid algorithm, expressed as ratios of character prop-

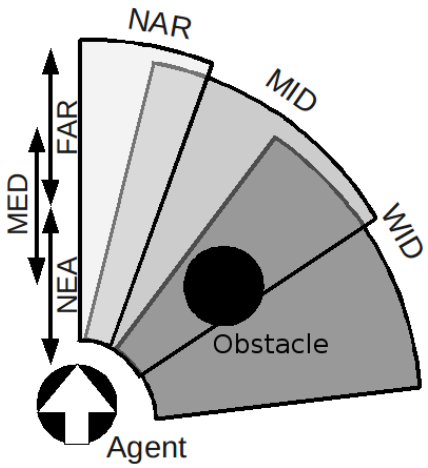

Figure 6.4: Classifying environment elements into overlapping fuzzy sets representing angles and distances.

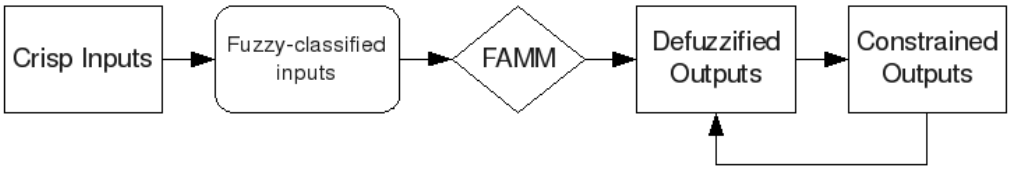

Figure 6.5: Scalable Fuzzy Navigation cascade with feedback loop to adjust instructions for a character's known physical limitations.

erties. Steering is also expressed as ratio of the speed control outputs. In this way the algorithm should produce motion that is both scalable to a character's motion limitations and applicable to a range of characters. The Fuzzy A* algorithm has been extended with the additional of a feedback loop (see Figure 6.5). This allows quick re-scaling of defuzzified output ratios should the intended output ratios exceed a character's physical limitations; for example a vehicle can not steer as sharply as desired at its current speed. The feedback loop in the figure allows us to incorporate a simplified mechanical simulation into the system, simply by adding a function that can limit the range of outputs to within the vehicle's allowed parameters (e.g. not steering at an angle beyond a vehicle's maximum mechanical limit).

6.3 Visualisation

Figure 6.6 is a screen shot from the 3D graphical simulation that created as a proving ground for the algorithms introduced in this chapter. Obstacles of various sizes are strewn about the 3D landscape (houses and trees) and several monitoring panels overlay the display. Because of the level of complexity and number of overlapping systems involved in real-time hybrid algorithms it is a great aid to have a system in place for visualising these different processes in real-time so that the designer can monitor the system as it happens with a full range of information. When an interesting case does seem to occur in the behaviour of the character - for instance, it was observed that an agent-driven vehicle was speeding up and slowing down even when travelling in a straight line - as illustrated by the sawtooth pattern in the 3rd graph down on the left in Figure 6.6. When this behaviour was observed it was able to pause the simulation time so that the vehicle stopped moving mid-sequence. The screen capture in the figure shows the observed saw-tooth velocity graph. A system was also in place whereby it was possible to export the data plots for all of these graphs to a range of formats for closer inspection in external programmes, where it could be seen that the current way-point being assigned to the fuzzy control system was always too close to the vehicle due to the high resolution of the path-finding system.

6.4 Proposed Self-Training Architecture

Given the large number of parameters that need to be calibrated for animated character control, and in particular for hybrid algorithms, it has been proposed that various automatic training systems be employed for this task; GAs have been explored for robots soccer agent training [82], and evolving neural networks have been used to improve and even generate entire behaviours for animated characters [43, 44] with some success. It is intended to adopt a subset of these ideas to self-train our characters. This chapter proposes an obstacle-course based *survival of the fittest* paradigm.

A range of obstacle courses can be quickly created with scenario designers, such as the real-time designer that is built into the simulation engine used in this chapter. These courses can be quickly designed as test cases for the sort of 3D environments that the characters would be expected to operate when finally deployed. For example, we might design a course or scenario consisting of flat landscape and large streets of house-like obstacles, and another course consisting of hills and valleys with sporadic trees to avoid for training an animated soldier character for a war scene in a film production.

The advantages of this kind of automatic training system would be:

- Initial characters start from some adequate basis, rather than from a zero-skill base
- Can train a large range of simulated characters or even simulations for real vehicles.
- Can adapt itself to new environments - e.g. we can hand-craft a new scenario that the character should have to be able to cope with and it can improve its system to deal with this new environment as well.

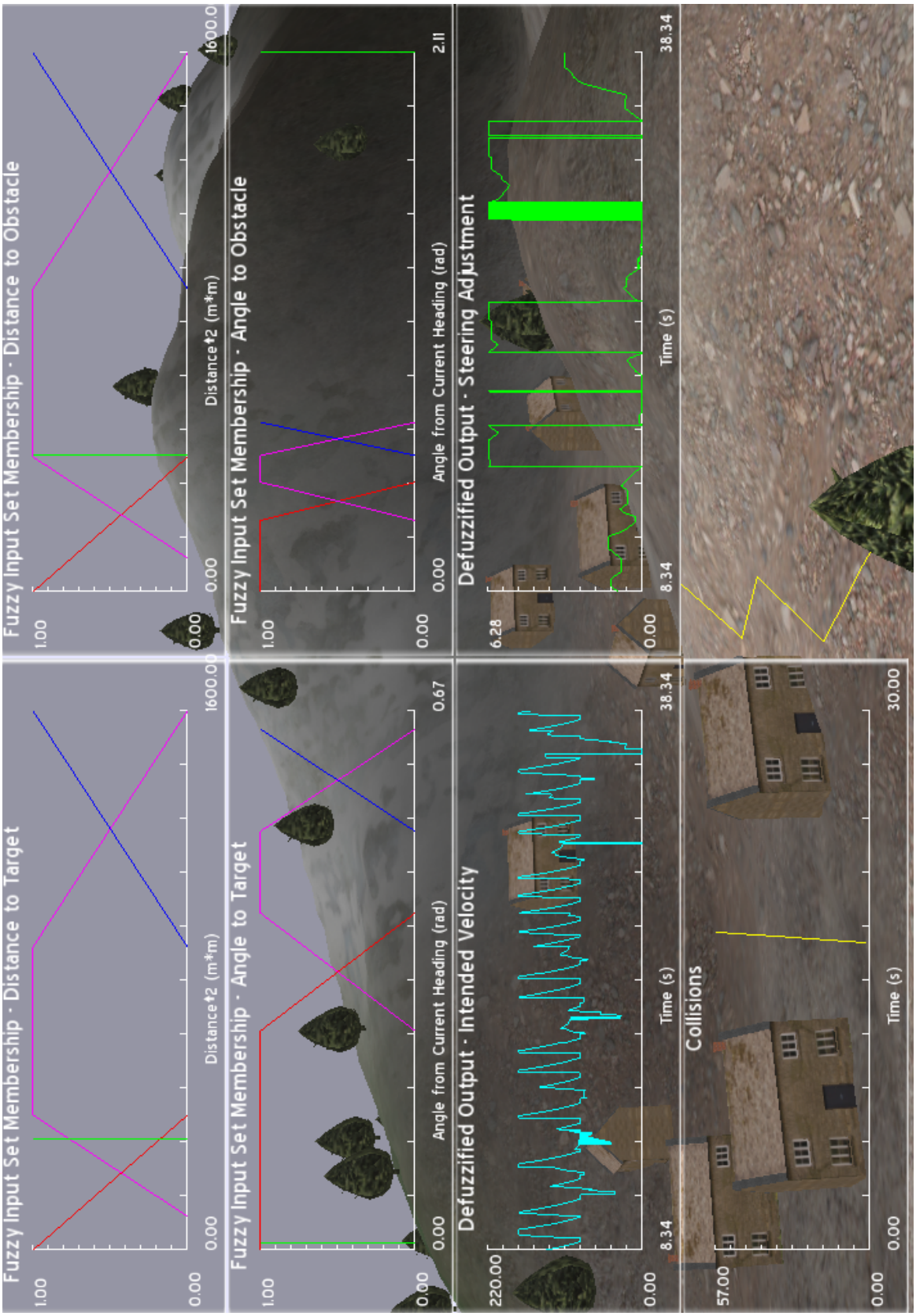

Figure 6.6: Graphing navigation data in real time.

- Could be set to self-improve in real-time during real execution over a time-slice basis, rather than a per-individual character basis.

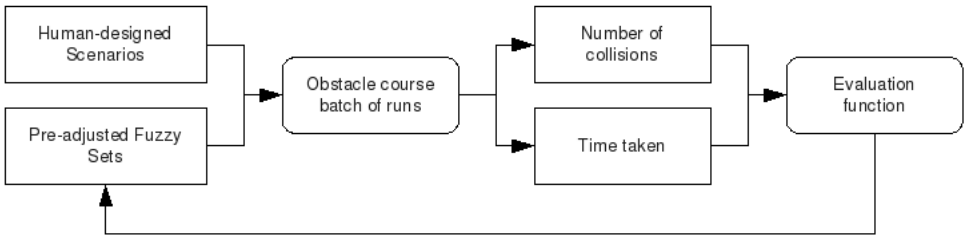

Figure 6.7: Proposed algorithm for self-improving animated character navigation. Obstacle courses are created manually; these represent, or are subsections of, the target operating environment. A number of training runs are taken in this environment; and a fitness evaluation and selection function tweaks navigation behaviour-controlling variables after each batch of runs in an attempt to gradually improve the behaviour.

To implement a genetic algorithm as a self-tuning mechanism for motion control the following steps are proposed:

1. Randomly adjust the parameters of a motion controller.
2. Randomly locate and orientate a character/vehicle that will use the controller.
3. Give the character's agent a goal position at a fixed distance.
4. Start a simulation run where the character moves to the goal.
5. Collect mesh interpenetration distances (measure the crashes of the character with scenery).
6. Collect time to move from start location to goal.
7. Generate a fitness score for the simulation run based on the collected data (time * cumulative distance of mesh interpenetration).
8. Repeat simulation run (above steps) a number of times to measure error in collected fitness score, and stop when error measurements are small enough.

As can be seen in Figure 6.6, these parameters are already being extracted per character evaluation. Using a "generation" of the character, each using slightly different modifications of parameters, it is possible to "select" (as in selective breeding) the best two members of the generation - those with the most minimal fitness scores to create the base parameters for the next generation. This process is illustrated in 6.7.

It is also a possibility to distribute this system over a network of commodity-level machines for very rapid mass training. There is infrastructure in the simulation engine in place for this sort of distributed computing and a full investigation of this training architecture will be be investigated in works in the near future.

6.5 Conclusions

This chapter has presented a snapshot of progress at adapting hybrid robot navigation algorithms to animated characters in 3D environments, introduced adaptations that have had to make to the system for application to more complex 3D environments, work towards completing a generalised version of this system that can be applied to a range of autonomous animated characters, and illustrated some visualisation techniques that were found to be particularly relevant to developing these sorts of real-time systems. Future works will seek to expand on this work by automating the calibration procedures for the hybrid algorithms dictating path-planning and motion control for these characters, and have in this chapter introduced the proposed architecture for this system.

Chapter 7

Adding Agent-Based Road Networks To Simulations

7.1 Introduction

A trend is emerging among architects, urban planners, and others traditionally in demand of scale-models and 3D visualisations away from *"What will this look like?"* models to *"What will this look like, and how will it work?"* simulations. A major demand of these simulations is the addition of realistic urban traffic in a non-intrusive, scalable way. This kind of feature can be used to demonstrate how intersections and pedestrian areas in proposed new building cope with traffic flow. It allows better immersion in a proposed design or reconstruction, and is also a highly valuable resource for increasing the realism of computer games and animations. To address this challenge a system for rapidly laying a working road-network within an existing 3D simulation has been designed, coupled with an agent-based approach to traffic simulation such that every vehicle has its own intelligence, goals, and decision-making ability.[1].

Traffic simulations can be generalised into two distinct categories; a top-down approach where traffic flow is based on mathematical flow formulae, and a bottom-up approach where each vehicle is given some unique properties and traffic flow is then an emergent behaviour; the collective result of each individual vehicle's actions. The latter approach is commonly referred to as microsimulation. As this work concerns 3D visualisations that produce convincing vehicle behaviour at the pedestrian level upwards, this chapter focus only on agent-based microsimulation models in this chapter.[2]

Most modern agent-based simulations provide agents involved with a simple set of traffic behaviour rules. The main governing behaviour in these simulations is a formula which determines the distance between each car on the road and the car that it is *following*.

$$d = l\frac{v}{k} \tag{7.1}$$

In 1953 Louis Pipes made a mathematical analysis of traffic patterns which he called an idealised *Law of Separation* [83]. In this simple model of traffic the focus is entirely on the *following distance* behind the car in front. Equation 7.1 gives us a following distance to maintain d behind a vehicle travelling at velocity v with a constant, so that for every ten miles per hour k there is the space of another car length l added to the following distance. Pipes considered *"the use of an electrical*

[1] A video of the road deployment system as described in this thesis with some prototype fuzzy-controlled vehicles is available at `http://antongerdelan.net/videos.html` under the heading *Early traffic simulation*

[2] Material in this chapter expands original research first published as A. Gerdelan, "A solution for streamlining intelligent agent-based traffic into 3D simulations and games," Tech. Rep. CSTN-072, CSSG, Institute of Information and Mathematical Sciences, Massey University, Albany, New Zealand, January 2009. [53]

analogue computer for studying the dynamical equations of the system", and this model has set the precedent for most agent-based traffic simulation.

$$a^f_{t+T} = \frac{\alpha(v^f_{t+T})^m(v^l_t - v^f_t)}{(x^l_t - x^f_t)^l} \tag{7.2}$$

Successive generations of this type of stimulus-reaction model extend Pipes' model by adding further factors of consideration. The General Motors (GM) models [84,85] (Equation 7.2) determine a vehicle's acceleration *a* response rather than the distance of separation directly. In Equation 7.2 a driver response delay *T* is considered for the response after an instance *t*. Speed, sensitivity, and headway coefficients are also modelled; *m*, α, *l* respectively. Accelerations *a*, velocities *v*, and positions *x* are considered for leading and following vehicles, denoted by superscripts *l* and *f*.

$$a^f_{t+T} = \frac{(v^l_t - v^f_t)^2}{2[S - (x^l_t - x^f_t)]} + a^l_t \tag{7.3}$$

Equation 7.3 - the Leutzbach Wiedemann model [86] - uses the leading vehicle's acceleration as stimulus for following vehicle. Whilst this formula may produce a pleasing traffic flow it is certainly not a realistic behavioural microsimulation as human-modelled agents would certainly not be capable of measuring a leading vehicle's acceleration with even remote accuracy.

$$v^f_{t+T} = \min\left[v^a_{t+T}, v^b_{t+T}\right] \tag{7.4}$$

$$v^a_{t+T} = v^f_T + 2.5aT\left[1 - \frac{v^f_t}{V_n}\right]\sqrt{0.025 + \frac{v^f_t}{V_n}} \tag{7.5}$$

$$v^b_{t+T} = b * T + \sqrt{b^2T^2 - b\left\{2\left[x^l_t - x^f_t - S_{jam}\right] - v^f_t T - \frac{(v^l_t)^2}{b^*}\right\}} \tag{7.6}$$

Gipps [32], Krauß, Wagner, and Gawron (KWG) [33, 87] and their derivatives provide further (again similar) formulae. Gipps' 1981 model is overly computationally complex as we can see from Equations 7.4-7.6 and therefore unsuitable for real-time simulation, but attempts to add realistic limits to vehicle behaviour. This gives us a velocity for a following vehicle to maintain *v* (Equation 7.4) based on an acceleration rate (Equation 7.5) and a braking rate (Equation 7.6). Krauß et al provide a generalisation of cellular automaton formulae, comparable to Gipps' model in form and function as they are both formulae for ensuring safe distances between vehicles at a range of accelerations, albeit sans expensive square root operations. The KWG model also adds some distribution of performance over vehicles based on a pseudo-random factor.

$$d = ax + bx \tag{7.7}$$

$$bx = (bx_{add} + bx_{multi}Z)\sqrt{v^l_t} \tag{7.8}$$

Wiedemann's 1974 model [31] for a *safe* following distance (Equation 7.7) is based on an average stand-still distance *ax* and a psychologically and statistically distributed safety distance *bx* (Equation 7.8). A bell curve distribution of driver car-spacing is created by setting *Z* between 0 and 1 with a mean of 0.5 and standard deviation of 0.15. Although we note the presence of the square root operator (undesirable for real-time simulations as it is time-consuming to calculate), the model has been used successfully in modern 3D graphical traffic simulation VISSIM [88] for the urban traffic component of that simulation framework.

It must be noted, that whilst formulae of this type may be used to control very simple agents having only an orientation, a position and a visual representation, these agents are then forced to conform to a flow-type mathematical model and exhibit no rational intelligence - emergence of traffic flow here is forced, and produces unconvincing motion when applied to graphical real-time microsimulation. Pipes described his law as a *rule of thumb* - and it has merit as a prelude to the sort of decision-making problems that vehicle-controlling agents will have to make, but these formulae are of limited value to the reproduction of psycho-visually realistic driver behaviour at the individual vehicle level - a property of paramount importance for creation of demonstrations and entertainment applications. The aim of this and the following chapter is to produce convincing and road-competent individual vehicles through well designed intelligent agents and have a working traffic flow emerge as a true side effect.

Fuzzy Logic allows us to represent a *partial truth*, or imprecise values between *completely true* and *completely false*. This gives us a mechanism for discriminating imprecise or changing data into a small group of overlapping *fuzzy sets*. From this foundation we can create very simple judgement-based reasoning, or *fuzzy inference*, to deal with complex real-world data; mimicking human decision-making. Fuzzy Logic systems require very little computational overhead, and can also produce smooth transitional outputs. Fuzzy Logic is therefore an ideal candidate for modelling human driver behaviour in large-scale, real time simulations. Although Fuzzy Logic has been largely overlooked for application to traffic simulation, research has been done to suggest that fuzzy logic might be used in two different modes within microsimulation [52]; firstly for traffic flow data analysis, and secondly as a controller module for a vehicle-driving agent. This second mode of operation is of interest to us, and has been used in other works as a substitute for the car following model, where a preliminary study has shown that a fuzzy model can more accurately reproduce real driver behaviour than the long-standing GM model [89]. These fuzzy systems that model driver behaviour also take advantage of the simplicity of cascading fuzzy classifiers such that weather and visibility factors have been added into the simulation recipes with relative ease.

7.2 Representing Complex Road Networks in 3D Simulations

Figure 7.1: An example road network to model: empty streets around a virtual Trinity College Dublin. Screen shot from Virtual Dublin [90]

Figure 7.1 gives us a good test-case on which to base the system. Pictured is part of the Dublin city model developed for the Virtual Dublin project [90, 91]. That project involved creation of a

scale 3D graphical model, or virtual world, constituting a large section of Dublin - complete with masses of pedestrians, detailed shop-fronts, accurately re-created buildings from the city itself, and extensive urban roads. In essence an ideal problem, containing a large number of 3D elements common to architectural simulations and modern game environments:

- To-scale models of urban landscapes already exist
- Geometrically large urban areas simulated
- Must run in real-time
- Interaction with existing 3D systems (world models, pedestrians)
- CPU and GPU constraints with existing systems
- Must be realistic at a range of camera angles and vantage points (car-level, pedestrian-level, birds-eye, etc.)

Figure 7.2: A street map of the area around the Trinity College Dublin campus; many urban traffic problem elements are present here.

If we refer to the map in Figure 7.2, which considers just a small subsection of streets surrounding those pictured in Figure 7.1, we can can see that a large number of complex and realistic problem elements for urban traffic are present:

- A dense network of one-way streets surrounds the campus
- Large volumes of pedestrian traffic crossing roads
- Large amount of buses with different routes through this area
- Complex, multi-feed intersections (see D'Olier Street - College Street intersection)
- Pedestrian-only streets (Grafton Street)
- Heavily congested areas (College Green - College Street).
- Variation of street sizes: tight one-way alleys and multi-lane roads.

To tackle all of these problems a simplified digital representation of the road network needs to be built, free of noise and complexity, such that the vehicle-controlling agents can use it as a guide in the same way that one might refer to a street map when driving through a new city. Because the 3D environment already exists, a digital road map needs to be created as a post-process, and married to the existing environment in such a manner so that it does not interfere with the existing work.

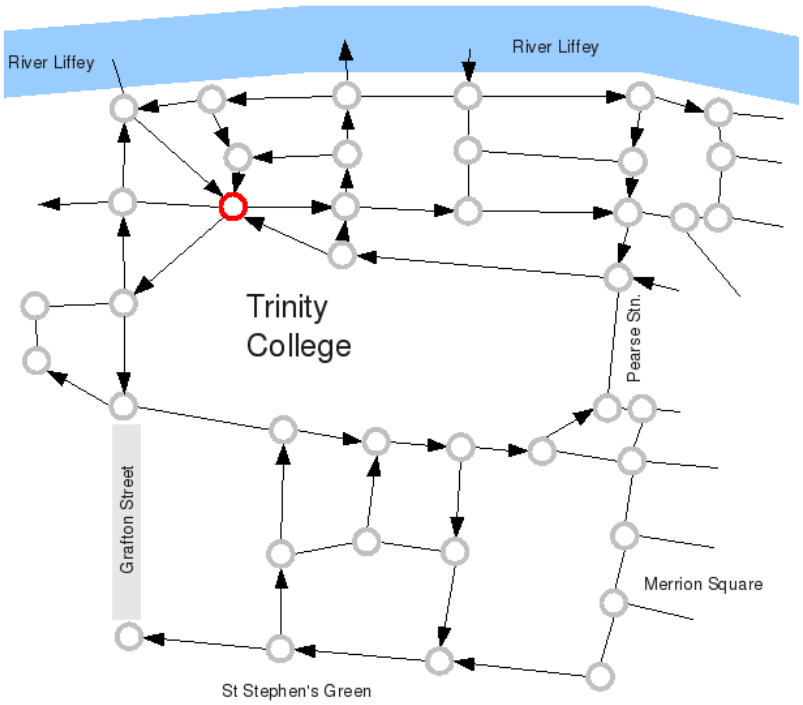

Figure 7.3: Representation of an automatically generated intersection and road network. Only a very small number of nodes are required for the search domain of a city route plan.

At the most basic level a searchable graph can be created to represent the traffic network, as illustrated in Figure 7.3. In the figure, circles are used to indicate the nodes which link the roads (and which are later used as intersection controllers), and straight lines to represent road connections. Uni-directional edges, or one-way streets, are indicated by the presence of an arrow head. Pedestrian-only areas (such as Grafton Street) are excluded from the graph, and therefore also from the traffic

route-search domain. We can see that if some weighting and heuristic information is added to the graph (such as estimated time to travel down each street, Manhattan distances between intersections, or congestion delay estimates), and an actual *cost* then a very effective low-complexity weighted graph can be quickly constructed, spanning even the entire city, that is searchable by either the A* AlgorithmA* [92] or D* Algorithm [77], with very low cost overheads in terms of CPU. Search routines can then be depth-limited to a desired threshold, such that a massive amount of dynamic route-planning agents can operate not just pre-trip, but even be allowed dynamically change course to reflect changing congestion conditions without demanding more processing time than is available.

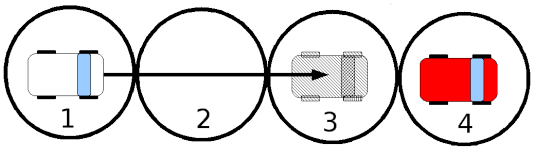

Figure 7.4: Lane occupancy and following distance model; the white car maintains following distance behind a leading car on a section of continuous road. Circles represent the nodular break-down of the road lane. In inner-city operation following cars aim to come to a complete halt at the last unoccupied node along their route. In high-speed operation vehicles store time-stamp each node as they exit it to emulate human following time estimation.

In practical application, bi-directional roads are treated as two separate streets. In order to monitor traffic congestion, and to allow merging, lane-changing, and overtaking behaviour, each road is split up into a smaller searchable graph of adjacent nodes; each node is large enough to accommodate a standard-sized vehicle. These sub-graphs allow us to monitor congestion per-lane. They also allow us to compute a car-following and lane-change model at a forward-thinking planning level, rather than in a non-planning formulaic fashion as used by all of the existing traffic microsimulations that were examined in this and the previous chapter. Thus the agents' route planning mechanism will comprise two layers:

1. A long-distance planning layer that creates a list of directions to follow to reach a destination most expediently. In the case of buses, this list of directions is fixed, but other vehicles may be allowed to dynamically recompute the directions to reflect changing congestion conditions.

2. A short-distance, depth-limited planning layer that plots the vehicle's next few movements up a street, and intelligently decides whether to change lanes or wait behind a slow leading vehicle.

In essence then there are two weighted graphs; one linking all of the streets for the whole city, and one higher resolution graph per-street. The D* Algorithm is almost certainly the ideal candidate for the short-distance planning layer as it has an in-built mechanism for choosing whether or not to *wait* for dynamic obstacles rather than simply moving around them as in other dynamic A* Algorithm variants [46, 47]. This is a substantially different approach to traffic simulation, and one which inculcates a thinking ability where over-taking behaviour is the result of cognitive, pre-meditated intention, rather than simply as the pseudo-random bi-product of a clockwork system. The long-distance graph could either be searched dynamically using a similar algorithm, or for a very large city where the graph might be distributed over several machines all possible paths could be pre-computed using a network routing algorithm. In this way it is known that the shortest path from every road to every other road prior to simulation run-time, and then weight-changes due to congestion could be propagated dynamically and update all of the routing tables in the system.

The advantage of sub-dividing road lanes into graphs (or chains, for single-lane roads) is illustrated in Figure 7.4. Here, a section of a continuous road is depicted that has been divided between 4 nodes. The vehicle occupying Node 1 is following the vehicle in Node 4. As the leading vehicle

drove into Node 4 it flagged the node as "occupied". This then impacts the short-term path planning of the following vehicle, which then aims to come to a complete halt by the time that it reaches the centre of Node 3, which is currently unoccupied. Should the leading vehicle move on a subsequent next node, then the following vehicle will instead aim to stop at Node 4, and so on. In this way a safe following distance is maintained, not by gearing the entire vehicle system around an attempt to reproduce a formula derived from countless hours of motorway watching, but by crudely emulating human following behaviour - the "two second rule" - or driving at a speed so that if the leading vehicle were to come to a sudden stop then the following vehicle has enough distance to stop without collision.

Should this system be extended to higher speed zones, then the two-second rule could quite literally be followed, and and add a time-stamp to the meta-data of each node as a vehicle exits it, so that the following vehicle can attempt to ensure that it is two seconds behind the leading vehicle, and inculcate this into its fuzzy acceleration and braking governor. This system is novel as the following vehicle does not need to know the leading vehicle's exact acceleration or velocity, as in other models, but rather emulates human decision making. This car following system would be suitable for stop-start traffic in congested inner-city areas, queueing at intersections, driving on empty streets, and maintaining a safe distance on other roads. The realistic reproduction of driver behaviour then largely hinges on the membership functions of each agent's own Fuzzy Logic controllers.

7.3 Architecture for Non-Intrusive Data Structures

To avoid interference with existing simulation architecture, and so that the entire traffic and vehicle-controlling system can be re-prioritised (or *reniced*, depending on the implementation) in terms of its CPU allowance (allowed to calculate at less or more frequent intervals) the entire system has been designed to operate as a separate, but encapsulated module with minimal communication links to existing simulation architecture (see Figure 7.5).

The communication links between the Traffic Module and the existing simulation engine in Figure 7.5 make two connections. The first link provides the *time-step length* and any changes of location to potential road obstacles controlled by other parts of the simulation (such as pedestrians moving onto the road). A second link communicates with a *Real-Time Editor Module*, which is introduced in Section 7.4. This link provides the editor with *3D mouse picking* information, or simply where a designer has clicked the mouse in 3D space to indicate that a road should be laid.

Experiments have not yet been done to find how time-step resolution changes might affect traffic flow, and thus what a minimum realistic (as compared to observed traffic flow data) frequency to maintain might be for the simulation, but common-sense suggests that real-time simulations and games targeting realism to human-perception should not fall too far outside of the normal rendering cycle (a minimum of approximately 30Hz, or the frequency of human visual perception), although this is a subject of ongoing study [93].

The navigation data itself is rather simple to organise; the smallest unit is the roughly car-sized *street node*, a string of which are encapsulated into a *street lane*. Several lanes combine to form a *street* unit, and these are linked together by *intersection nodes*, which link the start and end nodes of streets and control the wait/go semaphores of these connecting nodes. Navigation by agents is then a two-step process. The agent firstly queries the city map (street and intersection links only, as in Figure 7.3), and secondly queries the street that it is currently driving on for more precise navigation based on its nodes. As discussed in Section 7.4 the designer need only lay the street lanes; the streets and city map are formed automatically.

7.4 Rapid Construction of Virtual Road Networks

The Ogre graphics library [68] has been used to construct a real-time simulation engine as a research platform for this chapter. Into this engine a real-time editor module has been built, and an environ-

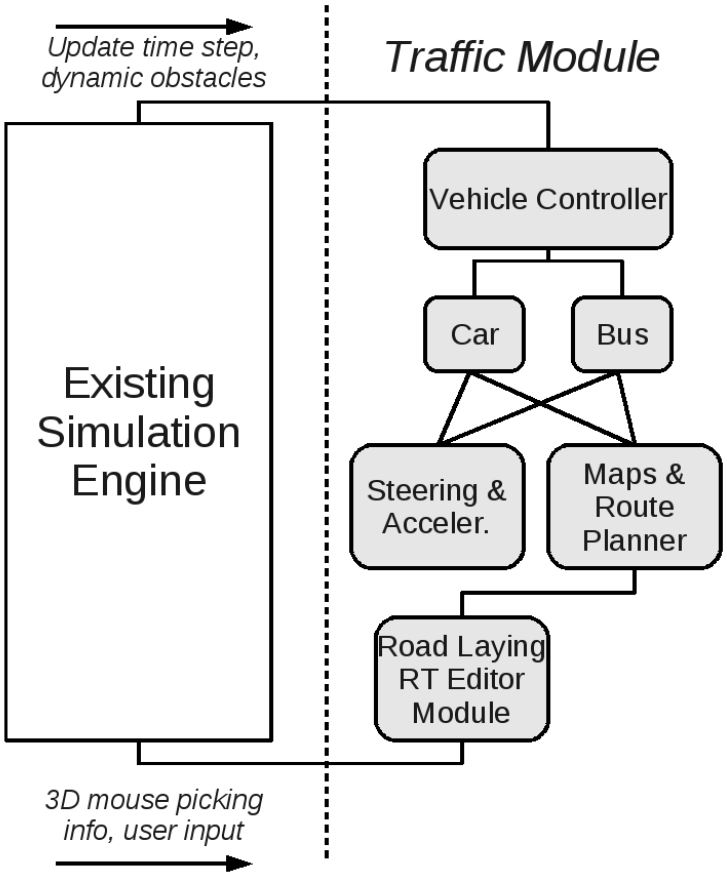

Figure 7.5: Non-intrusive module design. The entire traffic simulation sits externally to the main simulation and model, acting as an external library, and is thus unintrusive; the original model and simulation to do not need to be modified. The traffic module does need to be updated by the main simulation with a few parameters, however, to make sure that the time-step is consistent. The main vehicle controller module handles a collection of cars and buses which use fuzzy controllers for steering and acceleration, but follow a route planned by the lane that they occupy in the road map. A real-time editor module is also included in the model, which is used to manually match the road map with the existing 3D environment model. The editor requires some additional input (mouse instructions etc.).

ment has been created where the existing model of of Dublin city around the Trinity College Dublin campus has been loaded-in. Previous works have extolled the construction-time benefits of using point-and-click editing tools within the 3D simulations themselves to good effect [59], and these have been extended to build a rapid point-and-click design mechanism for the road-laying module with a 3D graphical overlay in this chapter. Conversely, it would have been feasible to have loaded the graphical editing module into the existing virtual Dublin simulation engine.

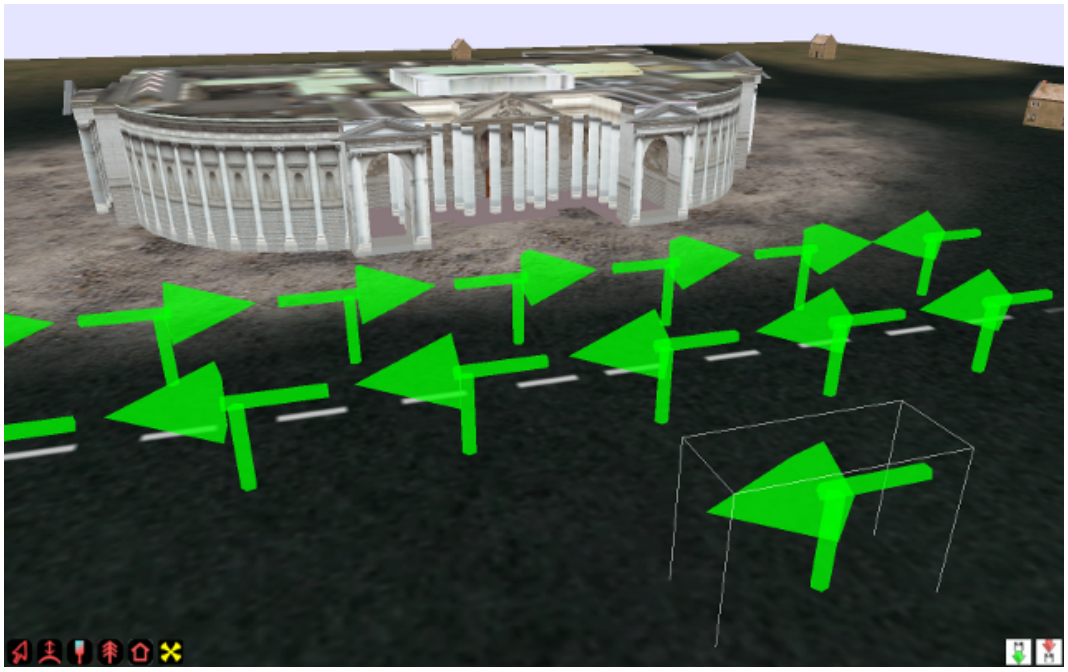

Figure 7.6: Laying roads within an existing 3D simulation model. Here translucent arrows are displayed to represent road lanes, and arrow heads indicate lane direction. The lanes are created by manually clicking and dragging in 3D with the mouse, which means that a designer can ensure that lanes follow the road according to the 3D model, and take into account as much or as little detail (such as intersections) as desired for the simulation.

Figures 7.6 and 7.7 shows us an example of a designer using the editing module *within* a running simulation. The key tool available is the road-layer, which allows the designer to choose to start laying a new road lane. The designer then *clicks and drags* a smooth line over the 3D environment, and a string of translucent arrows overlay the environment. These arrows indicate the direction of the street lane, and the location of each street node making up the street. The designer can then manipulate the individual nodes to alter the shape of the street's map representation or simply save the street lane. Should the designer link the end node of a street to another street's node, then an intersection is automatically created. In this way a designer can very rapidly lay a representation of an entire city's streets without interfering with the existing simulation. Future work will allow designers to customise intersections with different signals and controls.

Figure 7.9 shows us the system in operation; various automated and agent-driven vehicles employ a hybrid fuzzy algorithm to navigate through roads past the Bank of Ireland model from the Virtual Dublin model.

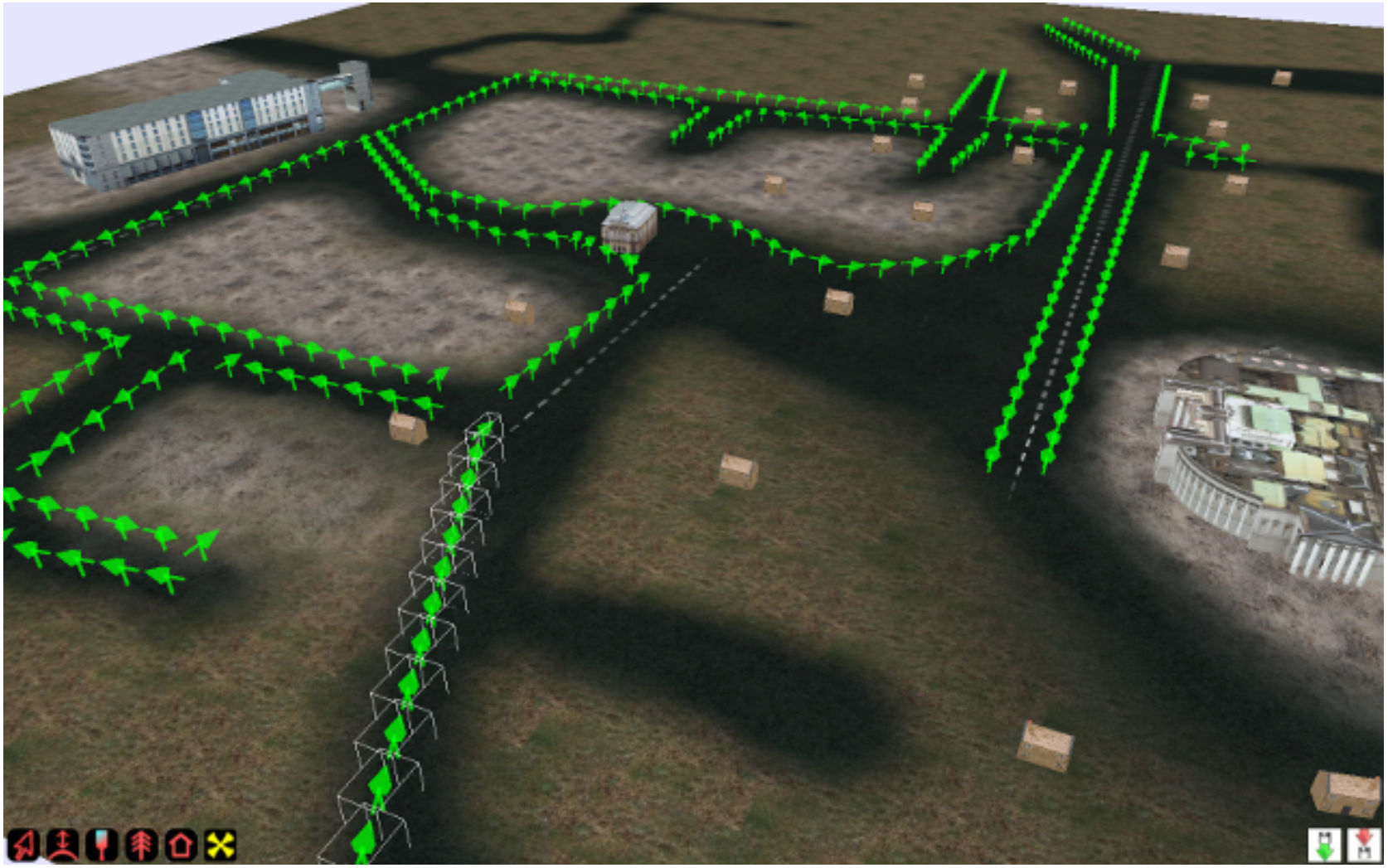

Figure 7.7: Laying roads in an existing simulation. The designer only needs to click at the ends of each road, and at any significant corners. This means that a massive road network can be crafted over an existing 3D landscape in minutes. The road lane at the bottom of the figure has been selected by the designer, as indicated by boxes around the nodes. A selected lane can be tweaked, branched off, or the individual nodes moved as required.

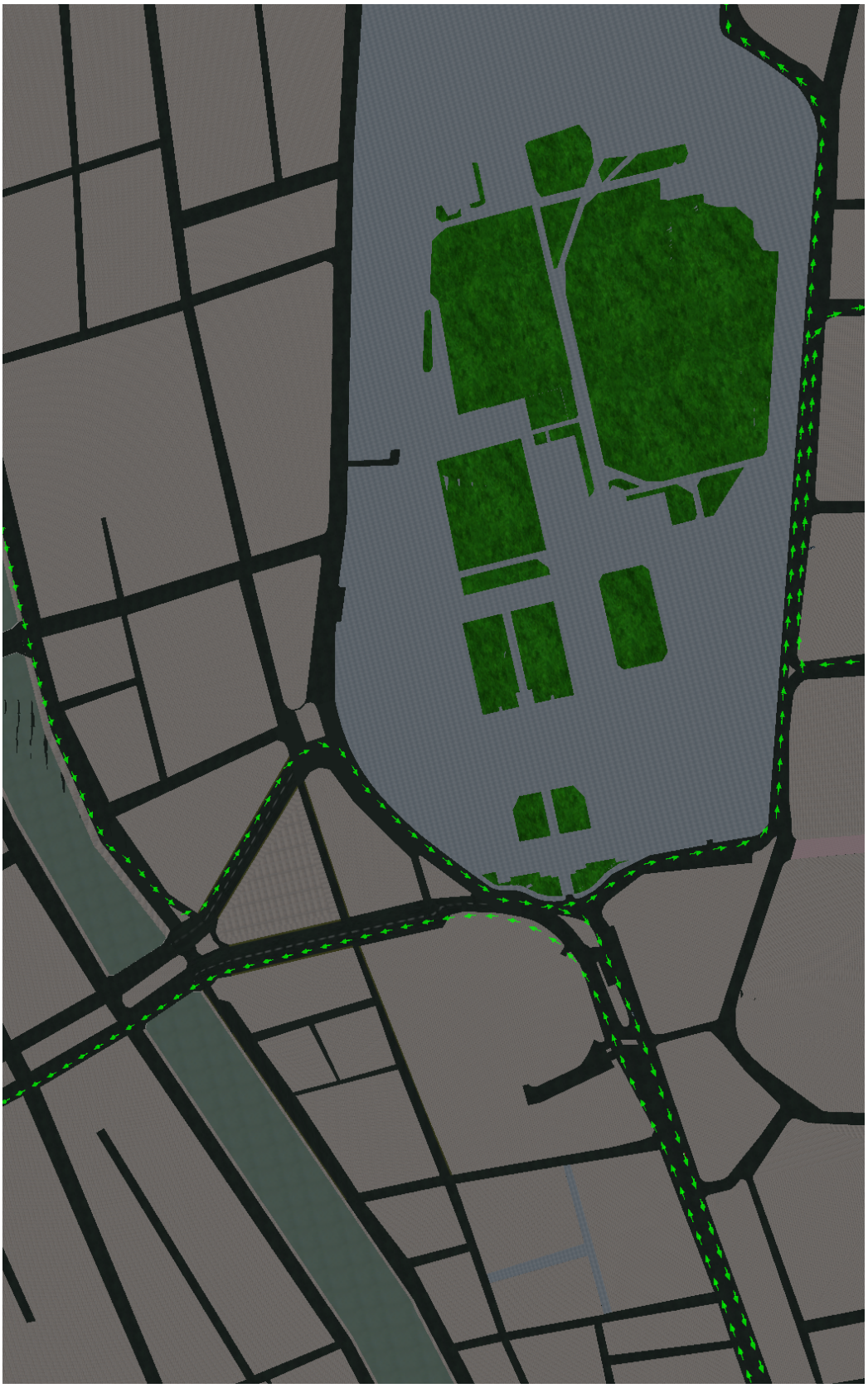

Figure 7.8: This figure shows us a birds-eye view of the original street area from Figure 7.1. A subset of the road lanes taken from the map in Figures 7.2 and 7.3 have been recreated, with the one-way directionality of lanes preserved. We can see that some of the nodes will need to be slightly manually adjusted to follow the modelled roads more closely as many of the points have been automatically interpolated and have not quite matched the model in this case.

Figure 7.9: Simulated traffic automatically navigate the previous empty streets of the Dublin model.

7.5 Future Works

A mechanism for handling intersections is required. Extensive study to draw upon has already been done in this area [94–96], as well as the handling of pedestrian-vehicle interactions at intersections [97–99]. In the simulation, intersections controllers need to handle controlled and some uncontrolled intersections; raising or lowering a meta-data semaphore (setting it to a *wait* or *go* state) for each of our street-level nodes entering an intersection. This means that even at intersections without traffic lights vehicles are still compelled *by the intersection itself* to comply with traffic rules, rather than each agent having an implicit knowledge of how to deal with stop signs and right-of-way rules. Turning out of *give-way* or *yield*-type intersections may be handled automatically by the car-following planning mechanism described above, but further experimentation is required to ascertain if this is indeed the case.

The design requirements of vehicle-controlling agents for the traffic system have been outlined in this chapter. Previous experience with agent architectures for both robotic and simulated vehicles indicate that a multi-layered navigation stack architecture using a fuzzy steering and acceleration controller may be a superior model in terms of smooth-motion output, expediency, and CPU utilisation [46–48, 70, 74, 100]. The chapter following this one will detail the agent architecture that has been designed for this road system, the specific functions and information passing requirements of the agent's navigation stack, and detail the set membership functions and fuzzy associative memory matrices used for agent reasoning.

Development of a special module for converting agent vehicle instructions into realistic animations will be investigated, with particular regard to steering behaviour, such that the path of each wheel obeys vehicle motion physics. This work may derive from the previous experience developing motor-control modules for two-wheeled robots using a similar fuzzy navigation module [101].

Due to the complex nature of optimising fuzzy set membership functions, the development of an automatic calibration or training system for agents controlling each vehicle with different performance and size specifications will be investigated. Research done for similar applications suggests that a GA might be employed here [82]. Experimentation here would extend previous work in this area [55].

Any realistic urban simulation involving both buses and pedestrians should also demonstrate interaction between both elements; a combination of animation and AI algorithms will be investigated, which may allow us to recreate realistic human-machine interaction scenarios. The effects that this interaction would have on each of the two systems involved (pedestrian crowds and urban traffic) would also be a subject of future study.

Distribution of the traffic system over multiple servers may become an issue for future consideration. We have already touched upon the possibility of using routing algorithms for traffic navigation. An architectural and communication design for a massive on-line distributed traffic system is a possibility for future research.

Also worthy of investigation is the scalability of the system; to conduct experiments to demonstrate the magnitude of computational complexity with increase in the number of road vehicles in simulation.

7.6 Discussion and Conclusions

An architecture for a streamlining traffic simulation into existing virtual worlds has been introduced, and a method for quickly designating streets and traffic lanes within existing 3D simulation or model has been detailed. The complete blueprint for the agent control system for navigating vehicles is documented in the next chapter. The system is successfully in operation as part of the Metropolis project at Trinity College Dublin, in Ireland.

The system introduces a novel level of realism to traffic simulation by utilising reasoning-based logic models. The models imitate human driver decision making and planning for individual agents.

This contrasts starkly with contemporary models which only approximate realistic human behaviour by injecting pseudo-random elements into mathematical formulae. Existing models can reproduce realistic overall flow patterns, but do not yet provide the level of behavioural detail that modern simulations and games require at the individual-vehicle level.

A large number of future works and extensions to the system have been planned, and particular focus will be put on development of an intelligent agent model to a level where buses can additionally cope with loading and unloading passengers in a complex 3D environment.

Chapter 8

Agents and Motion Controllers for Road Vehicles

8.1 Introduction

The new wave of computer-driven entertainment technology throws audiences and game players into massive virtual worlds where entire cities are rendered in real time. Computer animated characters run through inner-city streets teeming with pedestrians, get into and out of cars, and drive through rush-hour traffic; all fully rendered with 3D graphics, animations, particle effects and linked to 3D sound effects to produce more realistic and immersive computer-hosted entertainment experiences. Figure 8.1 illustrates our 3D traffic simulation (the final product). No modern urban environment is complete without realistic simulated road traffic, however traffic simulation has not kept pace with the entertainment industry - modern traffic simulations simply do not reproduce convincing individual driver intelligence to the level required by interactive entertainment. This chapter presents a new paradigm for agent-driven traffic simulation, specifically designed for such applications. The agent architecture utilises a new hybrid algorithm for dynamic road navigation, implicitly facilitates rational overtaking behaviour, and produces realistic smooth-motion vehicle control.[1]

The previous chapter introduced a rapid system for manually laying road networks within existing simulations. The roads created in this system had to support vehicle-driving agents that could integrate into the complex environments of modern visualisations and simulations. To tackle all of these problems a simplified digital representation of the road network is built; free of noise and complexity, such that our vehicle-controlling agents can use it as a guide in the same way that one might refer to a street map when driving through a new city. Because the 3D environment already exists, it is necessary to create the digital road map as a post-process, and the technique is to marry it to the existing environment in such a manner so that it does not interfere with the existing work.

In the previous chapter major traffic simulation models were analysed [31–33, 83–88] and it was found that they are based on a car-following formula as the primary system input to each vehicle-driving agent. Equations 7.4, 7.5, and 7.6 give us Gipps' [32] traffic model - which dictates the velocity of a following vehicle v^f as a combination of an acceleration rate v^a and a braking rate v^b based on a distance of separation behind the lead vehicle $x_t^l - x_t^f$.

The method discussed in this chapter takes a different approach, where the agents conduct a graph search of the road network; primarily using a list of way-points as input. Other traffic are taken into account as dynamic obstacles - car following in this model is a secondary, **emergent** property. An advantage of this approach is that lane-changing behaviour is dictated by a rational

[1] Material in this chapter expands original research first published as A. Gerdelan, "Driving Intelligence: A New Architecture and Novel Hybrid Algorithm for Next-Generation Urban Traffic Simulation," Tech. Rep. CSTN-079, CSSG, Institute of Information and Mathematical Sciences, Massey University, Albany, New Zealand, February 2009. [54]

Figure 8.1: The system in operation: real-time congested traffic simulation for Dame Street, Dublin, Ireland.

Figure 8.2: A test vehicle, created for the traffic simulation - an Enviro 400 Dublin Bus.

agent decision, rather than as the product of a pseudo-random formula, as it is in the major models. As this simulation is to be viewed at a range of levels of detail it aims to produce a more convincing behaviour model for individual vehicles first, and a traffic flow simulation second.

8.2 Agent Paradigm

The agents that have been designed operate with a Belief-Desire-Intention (BDI) [22] intelligent agent model, and are based on a simplified version of the stack architecture for robot-controlling agents in Chapter 5. It must be pointed out that the particular agents designed for this implementation do not use a drive-train simulation, but rather a simplified model where the agents' controllers directly affected the speed, and orientation angle of the vehicles.

Whilst other simulation developers have endeavoured to produce realistic models of human audio-visual perception, for the sake of simplicity this work "cheats"; providing the agents' *belief* or perceptual input with hard, generalised data. The agents are aware of all of the other vehicles and static and dynamic obstacles in their proximity. This simulation also broadly assumes that the agents have knowledge of local road congestion heuristics; representing knowledge of which roads to avoid at peak hour. The road itself simply provides all of the car-driving agents on it with the same network-weight information. A driver route-planning model that incorporates a psycho-visual model has not been investigated. Adding error to this information, or utilising real congestion and driver-behaviour data from city planners may be a subject for focus in future works.

The following specific information is regularly sent to each simulated agent, constituting its crude perception of the world:

- The relative location of the nearest moving or stationary obstacle (whichever is closer)
- The relative location of the next point along the path that the agent has planned for itself
- Local road network linkage information (a city map)
- Local road congestion heuristics (an estimate of congestion on each road)
- Occupancy of nearby road lanes

All of the agents are given a generalised goal, or list of goals upon entry into the simulation. The first case to consider is that of city buses operating set routes through the city. The bus-driving agents are given a list of way-points, stops, and time-table information, and then the agent is left with the task of driving through traffic between these points, making the appropriate stops. Private cars have even more generalised goals - they simply have a destination - the agent must contrive its own plan for navigation through traffic in a dynamic fashion.

The *intention* component of the agent model comprises several key functions;

1. Using "belief" information to form a multi-stage plan to achieve *desired* goals.
2. Working out specific actions to take immediately to move to the next stage of the plan.
3. Attempting to affect those actions

In the case of the car-driving agents, a specific example of this process is:

1. Using road-network information to find a short path to the destination.
2. Decide if it is necessary to change steering and acceleration such that the vehicle follows the road lane and avoids crashing into anything.
3. Adjust steering and braking/acceleration.

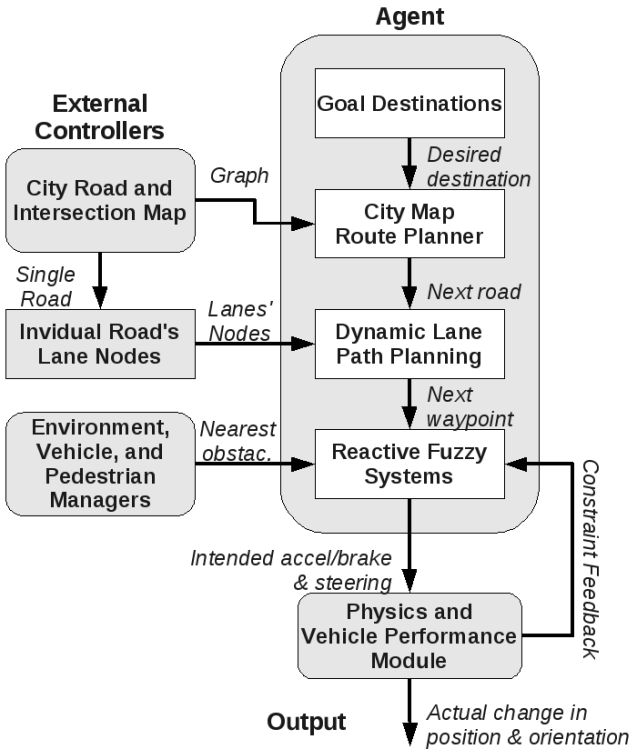

Figure 8.3: Architecture of the agents used for driving vehicles. The main modules represent belief, desire, and intention - input from external controllers, “goal destinations”, and output from fuzzy systems.

The architecture stack is illustrated in Figure 8.3. The architecture is broken into several layers of processing. This is a typical stack BDI agent architecture, of the type developed in Chapter 5, which is both CPU-efficient and successful for driving real-time agents using hybrid architectures, as demonstrated in previous works [46–48, 55]. Important to note about the design, is that the "desire" component of the agent, in this case the reactive fuzzy systems' output does not necessarily translate $1:1$ into the vehicle's actual movement in the world. Because it is intended to use this architecture as a generalised agent middleware, supporting a large range of vehicles with different performance characteristics, an additional output module is provided, which governs the actual output based on the physics of the environment, and the performance characteristics of a particular vehicle.

Any intelligent agent worthy of its name has the ability to evaluate the output of its own actions [75]. In order to facilitate this behaviour, a feedback loop is included - this lets the fuzzy controller module quantify how much its intended outputs have been constrained by physical limitations, and it can scale back its instructions to suit. The entire stack is generally updated with every frame, however to support a very large number of agents operating on one CPU simultaneously, the initial testing indicates that throttling back the dynamic path-planning module to $6Hz$, and the reactive module to no less than $30Hz$ provides an effective minimum threshold of updates. Any less frequent updates and approximated curved path-following behaviour becomes highly erratic. A study to confirm the effects of time-slice update throttling on AI and movement by vehicle-controlling agents is a an ongoing work [93].

8.3 Path Planning

The two layers of path-planning in the system both conduct fast searches of roads represented as nodular networks. The higher-level module takes an agent's desired destination and determines a path through a city, making use of a pre-computed graph that is generated from the rapid road-laying tool introduced in the previous chapter. This graph is purely made up of road intersection points, and as such is a very small graph to search (see Figure 7.3). There are several search method options for this graph:

1. a static route look-up, using pre-computed paths
2. pre-defining routes to every destination, updated dynamically *by the Road Manager* using a routing algorithm
3. a dynamic heuristic-based search (during driving) to reflect changing conditions such as congestion.

In extremely large simulations, or those where the time period simulated does not allow for changing conditions, then a pre-computed route model may suffice. Propagating congestion information around a city map using a routing algorithm would require a powerful central server, but may make for an interesting case study for urban planning. It was decided to use a standard approach and do a dynamic heuristic search of the road map at any time that an agent's goal destination changes. To reflect changing congestion conditions it is also possible to recompute this route during travel, but progressive traffic pattern change simulations have not been investigated at this stage, although a possible subject of future work may be to demonstrate changing patterns of traffic at peak hours - where cars take alternative routes. The ubiquitous A* Algorithm [92] has been employed for this task.

The lower-layer path finding is dynamic, and operates within individual road lanes. The road lane laying system automatically breaks lanes into vehicle-sized nodes. These are both searchable, and also aid car following separation and traffic queueing. Single lanes are simply list-traversal operations, where nodes that are occupied temporarily halt the generated path (see Figure 7.4 for illustration). Multi-lane roads facilitate emergent lane-changing behaviour. Given a small heuristic cost for changing lanes, it is possible to stimulate the agents to spread evenly over multiple road

Real Term	Rough Set Value Range	Fuzzy Term
Wide Arc	> 0.40 rad	WID
Mid-range Arc	$0.20 - 0.60$ rad	MID
Narrow Arc	$0 - 0.40$ rad	NAR
Far Distance	$> 20m$	FAR
Medium Distance	$0 - 28.28m$	MED
Near Distance	$0 - 20m$	NEA

Table 8.1: Fuzzy Input Term Definitions (Route-Following). A 3x3 input set model is used as based on previous works [46,47]. This table shows what range of input values are accepted as part of each set. The third column gives us the fuzzy shorthand name for each set. Note that the ranges for the sets overlap; this helps us smoothly transition between rules rather than hard-step as in traditional logic.

lanes, or for faster vehicles to decide to overtake slower leading vehicles. In this way it is possible to completely avoid designing or employing a complex, and clockwork mathematical formula for representing lane-changing behaviour as is widely employed for traffic simulation models.

8.4 Reactive Vehicle Control

Initial studies [89] have shown that fuzzy-logic based models can represent real traffic flow more accurately than traditional GM traffic simulation models, and take more detailed simulation variables into account - such as representing behaviour based on different levels of driver experience, and the more accurately modelling the affect that varying weather conditions have on driver behaviour across different experience levels.

The current driver agent model divides into two fuzzy control systems. Both operate in real-time, and both are blended together to form a combined result for steering and acceleration behaviour. These two complimentary systems are:

- reactive obstacle avoidance
- route following behaviour

The route following system takes as input the next way-point, fed down from the path-planning layer. Specific inputs used are the change in heading angle and distance to this way-point. Conversely, the obstacle avoidance system considers the change in heading angle, and distance to the nearest obstacle. Both of these systems classify the real inputs into overlapping rough sets, where they can be classified in human terms, for a quick look-up-table type decision. The mapping of these inputs is designed so that for a vehicle moving up a lane, oncoming traffic can be ignored unless it crosses into the path of the subject vehicle, and if so - a series of rules will progressively move the subject vehicle away from its route, avoid the obstacle, then finally return to its route. Tables 8.1, and 8.2 provide the design for fuzzy input set mappings. Fuzzy sets for classifying (or *fuzzifying*) real inputs into rough sets are illustrated in Figures 8.4-8.7.

The vehicles' output performance specifications are mapped to fuzzy output sets (Tables 8.3 and 8.4). Because it is convenient to represent a range of different vehicles with the same fuzzy system pattern, outputs are expressed in terms of a vehicle's maximum velocity (v_{max}). The steering adjustment outputs are also expressed relative to the vehicle's current speed; such that resultant movement vector of the vehicle is to some extent scalable, and so expect similar rates of vehicle turn at different speeds. A further feedback system is then used to scale back the output if current physical performances limits are reached.

Fuzzy Output set mapping functions are illustrated in Figures 8.8-8.10.

Real Term	Rough Set Value Range	Fuzzy Term
Wide Arc	> 0.80 rad	WID
Mid-range Arc	$0.40 - 1.20$ rad	MID
Narrow Arc	$0 - 0.80$ rad	NAR
Far Distance	$> 14.14m$	FAR
Medium Distance	$0 - 28.28m$	MED
Near Distance	$0 - 14.14m$	NEA

Table 8.2: Fuzzy Input Term Definitions (Obstacle Avoidance). An interesting feature of the obstacle avoidance input sets is that, whilst route-following behaviour can operate alone (when there are no obstacles), obstacle-avoidance is actually a modifier to route-following. The input angles here are double the range for route following so that the opposite extreme turn can be applied to any route-following turn if so required.

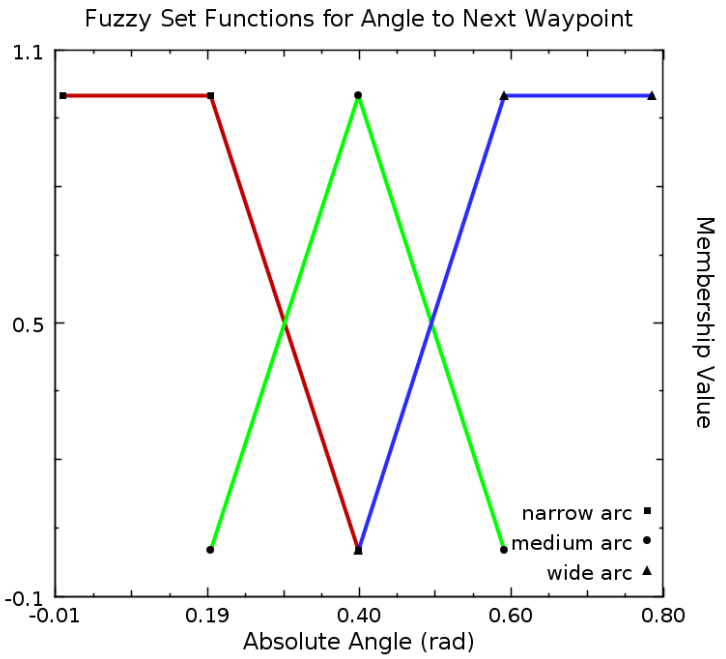

Figure 8.4: Fuzzy set membership functions for classifying the angle to the nearest next way-point in fuzzy terms. Angles here are absolute radians to the left or right of the current heading of a vehicle, so that way-points on the left hand side of a vehicle are treated the same as way-points to the right.

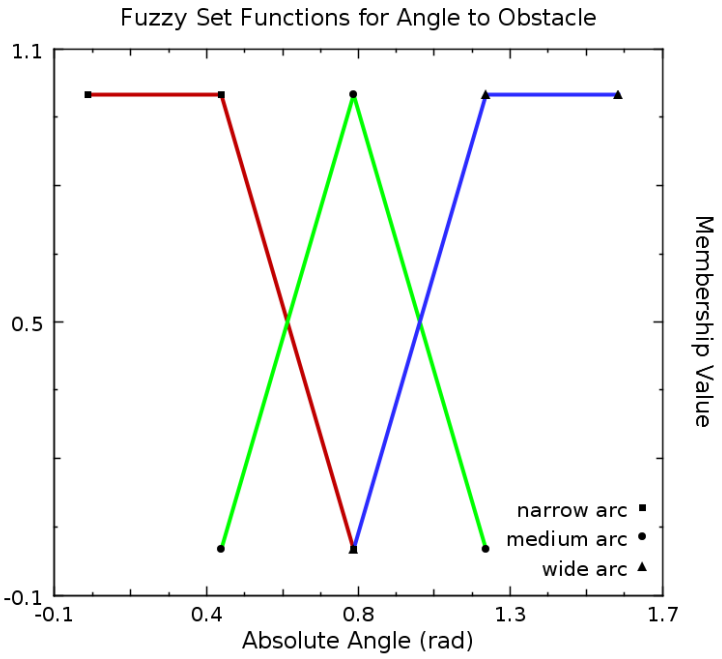

Figure 8.5: Fuzzy set Membership functions for classifying the angle to the nearest obstacle in fuzzy terms. Angles here are absolute radians to the left or right of the current heading of a vehicle, so that obstacles on the left hand side of a vehicle are treated the same as obstacles to the right.

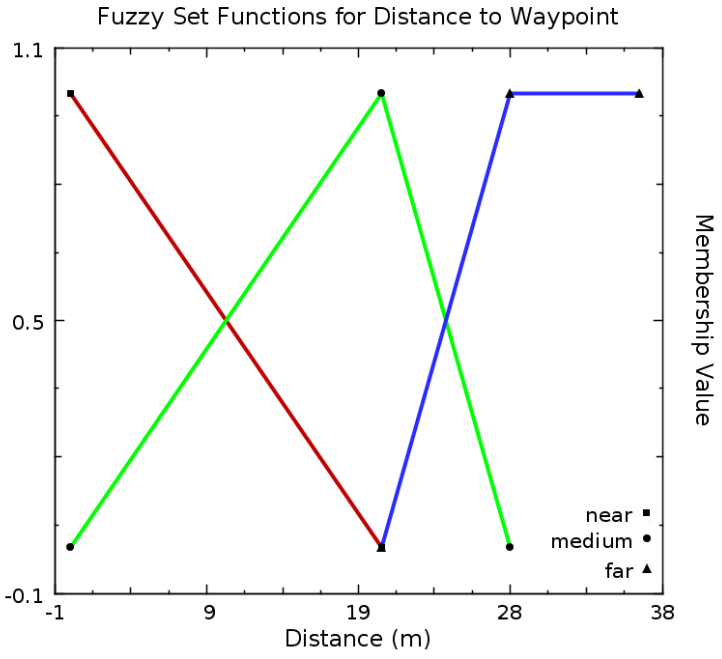

Figure 8.6: Fuzzy Input Set Membership Functions for classifying the distance to the next way-point in fuzzy terms.

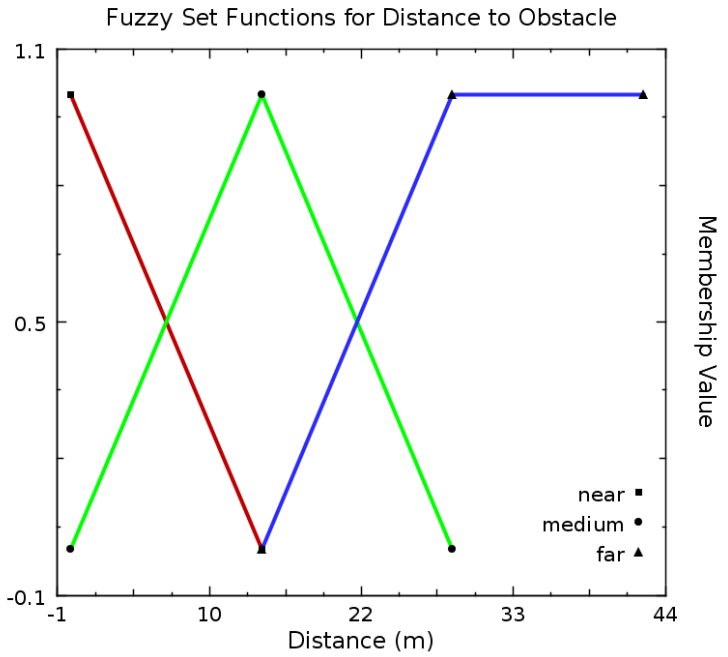

Figure 8.7: Fuzzy Input Set Membership Functions for classifying the distance to the nearest obstacle in fuzzy terms.

Real Term	Centre Value	Fuzzy Term
Top Speed	$1.0 \cdot v_{max}$	TOP
Fast Speed	$0.8 \cdot v_{max}$	FAS
Medium Speed	$0.6 \cdot v_{max}$	MED
Slow Speed	$0.4 \cdot v_{max}$	SLO
Very Slow Speed	$0.2 \cdot v_{max}$	VSL
Stop	$0.0 \cdot v_{max}$	ZER
Full turn	$v_{defuzz} \cdot 0.08 rad \cdot s^{-1}$	FUL
Very Sharp Turn	$v_{defuzz} \cdot 0.06 rad \cdot s^{-1}$	VSH
Sharp Turn	$v_{defuzz} \cdot 0.04 rad \cdot s^{-1}$	SHA
Medium Turn	$v_{defuzz} \cdot 0.02 rad \cdot s^{-1}$	MED
Light turn	$v_{defuzz} \cdot 0.01 rad \cdot s^{-1}$	LIG
Very Light turn	$v_{defuzz} \cdot 0.005 rad \cdot s^{-1}$	VLI
No turn	$v_{defuzz} \cdot 0.0 rad \cdot s^{-1}$	ZER

Table 8.3: Fuzzy output term definitions for the route following component. These fuzzy sets are quite different to fuzzy input sets, as their purpose is to define centre-points for a centre-of-gravity function. The process of combining fuzzy outputs into a final combined crisp output is called “defuzzification”. A large range of output sets are defined to give the rules more flexibility to switch between outputs. Note that not all outputs are used in rules. They have been left in the system as it is intended to develop an auto-training system in future works that can experiment with a range of outputs.

Real Term	Centre Value	Fuzzy Term
Top Speed	$1.0 \cdot v_{max}$	TOP
Fast Speed	$0.8 \cdot v_{max}$	FAS
Medium Speed	$0.6 \cdot v_{max}$	MED
Slow Speed	$0.4 \cdot v_{max}$	SLO
Very Slow Speed	$0.2 \cdot v_{max}$	VSL
Stop	$0.0 \cdot v_{max}$	ZER
Full turn	$v_{defuzz} \cdot 0.16rad \cdot s^{-1}$	FUL
Very Sharp Turn	$v_{defuzz} \cdot 0.12rad \cdot s^{-1}$	VSH
Sharp Turn	$v_{defuzz} \cdot 0.08rad \cdot s^{-1}$	SHA
Medium Turn	$v_{defuzz} \cdot 0.04rad \cdot s^{-1}$	MED
Light turn	$v_{defuzz} \cdot 0.02rad \cdot s^{-1}$	LIG
Very Light turn	$v_{defuzz} \cdot 0.01rad \cdot s^{-1}$	VLI
No turn	$v_{defuzz} \cdot 0.0rad \cdot s^{-1}$	ZER

Table 8.4: Fuzzy Output Term Definitions (Obstacle Avoidance Component). The values expressed in output sets are not actual velocities, as one would expect, but rather a portion of a vehicle's maximum allowable speed. This allows us to use the same output set definitions for a range of different vehicles, and also for vehicles in different speed limit zones. The output steering is relative to a vehicle's output speed - this helps us to maintain similar steering rates (velocity vectors) at a range of different speeds.

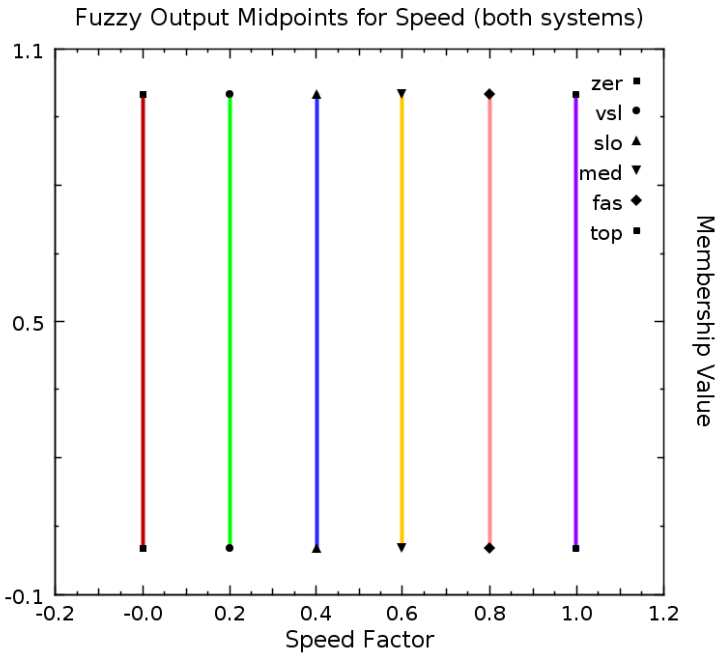

Figure 8.8: Fuzzy output value midpoints for desired speed factor. The defuzzified output speed factor will be multiplied with the vehicle's top allowable speed to produce a desired speed as a real number.

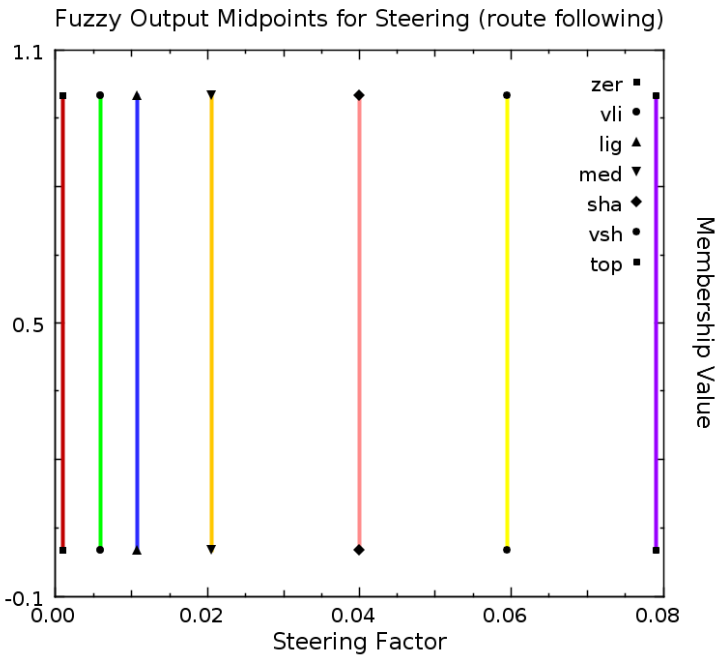

Figure 8.9: Fuzzy Output Value centres for base route-following desired *steering adjustment* factor. This factor is directly proportional to the vehicle's base route-following speed factor so that the vehicle will steer remain én route over a range of speeds.

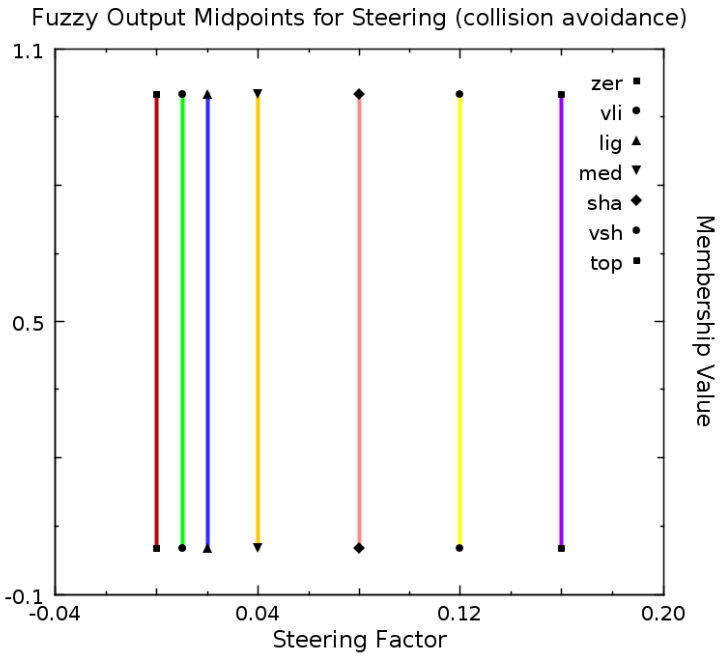

Figure 8.10: Fuzzy Output Value centres for obstacle avoidance desired *steering adjustment* modifier. This factor is subtracted from the route-following steering factor. To enable the same range of steering adjustment these values are exactly double those used for route-following.

	NEA	MED	FAR
NAR	ZER	VSL	ZER
MID	VSL	SLO	MED
WID	SLO	MED	FAS

Table 8.5: 3×3 FAMM for Desired Speed (Obstacle Avoidance Component). Output fuzzy speeds are given for all fuzzy input distances and angles.

	NEA	MED	FAR
NAR	SHA	MED	VLI
MID	MED	VLI	ZER
WID	VLI	ZER	ZER

Table 8.6: 3×3 FAMM for Desired Steering (Obstacle Avoidance Component). Output fuzzy steering adjustments are given for all fuzzy input distances and angles.

The labour-intensive task is then to design fuzzy rules. These map all of the different combinations of fuzzy input values to fuzzy outputs. A sample of the rule design is given below. Whilst rules can be hand-adjusted to produce a satisfying output for one particular vehicle, the fuzzy systems need to be recalibrated for every vehicle with different performance characteristics or physical dimensions. To simulate a large variety of urban vehicles this task becomes a complex constraint-based problem; and it is a problem that will be tackled in the next chapters by way of an automatic-calibration and self-training system. Initial rules are provided in Tables 8.5-8.8. Note that not all of the output sets have been used in these rules, and that there is certainly scope for designing more balanced rule-sets.

```
If the obstacle is a near distance away,
and the obstacle is a narrow angle from the heading,
then change speed to zero,
and change steering to a sharp turn.

If the obstacle is a medium distance away,
and the obstacle is a narrow} angle from the heading,
then change speed to very slow,
and change steering to a medium turn.

If the obstacle is a far distance away,
and the obstacle is a narrow angle from the heading,
then change speed to zero,
and change steering to a very light turn.

...
```

Above is a subset of the fuzzy rule design for obstacle avoidance. Rules map each possible combination of fuzzy inputs to valid fuzzy outputs. After designing the fuzzy inference engine in human terms like this it is possible to create look-up tables for quick operation as expressed in Tables 8.5 and Tables 8.6.

Once all of the rules have been evaluated an aggregation "centre-of-gravity" function merges the outputs for each system. The outputs of both systems are then blended together; the steering of the obstacle-avoidance component is exactly double that of the route-following system in order to reflect this. The rules, therefore, need to be designed to reflect the operation of the system in combination;

note: not all outputs used - scope for using these for adjustment of rules

	NEA	MED	FAR
NAR	VSL	FAS	TOP
MID	VSL	MED	FAS
WID	VSL	SLO	MED

Table 8.7: 3×3 FAMM for Desired Speed (Route-Following Component). Output fuzzy speeds are given for all fuzzy input distances and angles.

	NEA	MED	FAR
NAR	ZER	VLI	LIG
MID	VLI	LIG	MED
WID	LIG	MED	SHA

Table 8.8: 3×3 FAMM for Desired Steering (Route-Following Component). Output fuzzy steering adjustments are given for all fuzzy input distances and angles.

a difficult task to accomplish manually, and which further underlines the advantage that would be gained from an auto-calibrating system.

8.5 Discussion and Conclusions

An early version of the complete system (agent control with road map integration in a 3D city) was demonstrated at the Metropolis exhibition at the Science Gallery, Trinity College Dublin throughout May 2009. Figure 8.11 demonstrates the complete system in action; a range of vehicles have been *spawned* to replicate high-congestion traffic around the Trinity College Dublin campus. A range of vehicles, road lanes, and mergers are represented.

Figure 8.12 presents this same scene, but with road lane demarcations visible. Each one of the arrows in Figure 8.12 corresponds to one of the road lane *nodes* - a roughly vehicle-sized tag used by the agents to facilitate traffic queueing and vehicle-following higher-level behaviours.

Using vehicle-sized nodes is an effective method for calculating the spacing between vehicles when vehicles are all of roughly similar sizes and speeds. However, the traffic simulation described in this chapter also included long buses. This meant that nodes had to be expanded to bus-sized nodes as minimum, which means bigger spacing between queueing vehicles. This is not noticeable in low-congestion scenarios, and indeed in our simulation there was not enough available computational resources to simulate very large amounts of vehicles at once. However, larger-scale simulations would need to switch to a different mechanism for maintaining spacing when traffic is stopped and queueing.

In a large city scene, with over 2000 vehicles simulated in real-time simultaneously, it was possible to maintain processing of over $200Hz$ (frames rendered per second) using a single thread programme architecture on an Intel Core(TM)2 Quad $2.4GHz$ CPU with a GeForce 8800 GTX card, which more than satisfied the requirement that the traffic was unintrusive in terms of the overall programme. The main intention is to extend this work with a self-training module that operates by running a new vehicle through a range of obstacle courses representing typical simulation operating environments and key problems (tight corners, intersections, merging traffic, static and dynamic obstacles etc). Batches of these runs will generate a large amount of comparative performance data, which gives us grounds for full agent improvement, and should provide us with enough information to show how effective this approach is in tuning fuzzy sets and rules for best effect. The best *selection* method for self-improvement is a matter of study in future work, with options ranging from

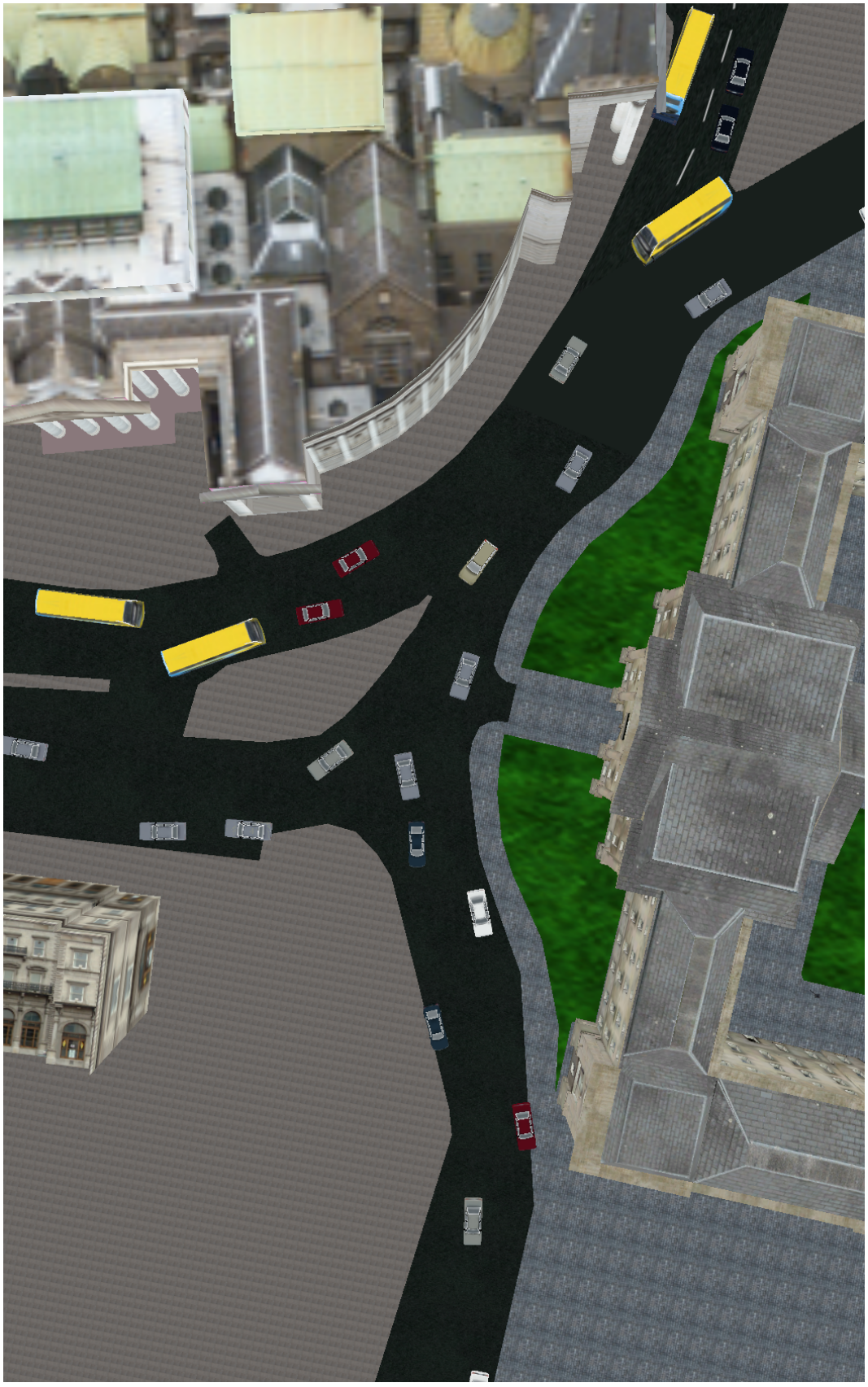

Figure 8.11: Prototype system in operation: real-time congested traffic simulation for College Street, Dublin City.

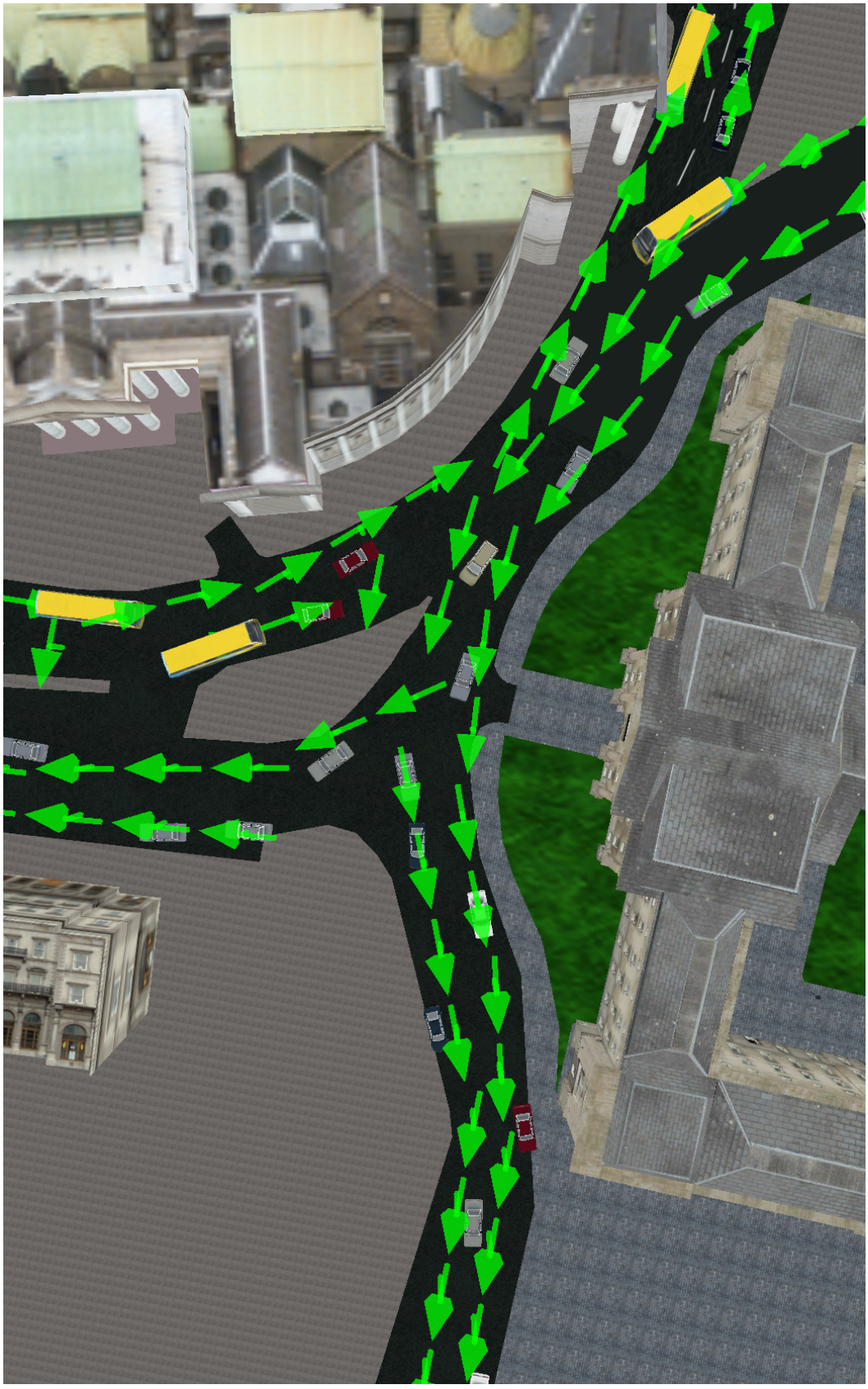

Figure 8.12: Prototype system in operation: lane demarcations for College Street, Dublin City.

brute-force methods, evolutionary selection, and genetic algorithms (GAs). GAs have been designed for the similar problem of soccer robots [82], and as this hybrid system is an extension of robot navigation algorithms, then GAs for simulated vehicle control warrants investigation also. All of these concepts are explored in the remaining chapters of this thesis. Figure 8.13 illustrates the final 3D traffic driving autonomously, using the generated road map, and fuzzy steering mechanism.

Figure 8.13: Street-level scene of system in operation: real-time congested traffic simulation for Dame Street, Dublin City.

Chapter 9

On Simulation Frameworks for Automatic Calibration Systems

9.1 Introduction

Enabling vehicle-driving intelligent agents to improve their own driving behaviour requires a specialised simulation framework. The design of such a framework is no trivial task, and a number of critical design questions are raised. Numerous training machine-learning paradigms are available; brute-force methods, evolutionary selection techniques and genetic algorithms - what are the training/performance trade-offs of each? How do we choose from the wide variety of variables that can be used to punish or reward the agents? There is also a completely open design choice when it comes to choosing a training arena - should the agents learn during application or in pre-built training courses? This chapter examines a range of different methods for application to particular agents, evaluates the use of different levels of complexity for fitness evaluation functions, and explores differences in framework design to optimise either navigation performance or human-perceived realism of simulated motion through machine learning techniques. In the course of this work new tools were created to aid design, and these are also discussed.

Given the large number of parameters that need to be calibrated for autonomous vehicle control, and in particular for hybrid algorithms [46–48, 74], various automatic training systems that might be employed for this task have been proposed [55]; genetic algorithms have been explored for robots soccer agent training [82], and evolving neural networks have been used to improve and even generate entire behaviours for animated characters [43, 44] with some success. This work intends to adopt a subset of these ideas to self-train our agents. This chapter introduces a *survival of the fittest* paradigm.

A range of obstacle courses can be quickly created with scenario design toolkits, such as the collection of tools which have already been discussed in Chapter 4 of this thesis for simulation frameworks; 3D training arena designers, rapid road-network-laying tools, and terrain formation tools [53, 54, 59]. These training arenas can be quickly designed as test cases for the sort of 3D environments that the vehicles would be expected to operate within when finally deployed. For example; we might design a scenario consisting of flat landscape and large streets of house-like obstacles, and another course consisting of hills and valleys with sporadic trees to train a simulated military vehicle for a war scene in a film production.

The advantages of this kind of automatic training system would be:

- Initial agents start from some adequate basis, rather than from a zero-skill base as in contemporary works.
- Can train a large range of simulated agents or even simulations for real vehicles, (this work

has developed from robotics applications [46]).

- Can adapt itself to new environments - e.g. we can hand-craft a new scenario that the agent should have to be able to cope with and it can improve its system to deal with this new environment as well.
- Could be set to self-improve in real-time during real execution over a time-slice basis, rather than a per-individual agent basis.

From a generation of agents, each with slightly different modifications of fuzzy rules, we can select and merge the best members of the generation (with the most minimal fitness scores) to create the base parameters for the next generation. This process is illustrated in Figure 6.7.

It is also possible to look at distribution of a training system over a network of commodity-level machines for very rapid mass training. This infrastructure sort of distributed computing is in place in the simulation engine developed for this chapter, although early experiences have found that this approach is not ideal for graphical simulations as most of the benefits of parallelised training are outweighed by the costs of GPU hardware, and driver-related unreliability for this type of application. Of course, one of the great strengths of the visualisation-based work that has been done throughout this thesis is the ability to monitor and comprehend the simulations as they happen. The option of interactively visualising, and interrupting simulations when desired, is possibly a more powerful tool than parallel processing power which introduces a black-box element to training.

9.2 Works of Note

Although the possibility of hybrid systems for animated character animation has been largely untouched, there is some interesting research being done in the field of Neural Networks relating to animated characters. Of particular importance is the development of the NERO game [43, 44] by researchers at the University of Texas at Austin. Researchers working on the NERO system have built a 3D real-time graphical simulation on 3D graphics engine software popular with games developers, and show that characters for computer games can be trained to *evolve* human-guided autonomous intelligent behaviours using genetic algorithms through a punishment and reward scheme [40, 42] which they have coined *neuroevolution* [102]. Interestingly this happens in real-time [103].

The designers claim that a team (or perhaps more appropriately a *species*) of these agents can be trained from scratch to a level of competitive behaviour in only a few minutes [41]. There are some research gems within this work can be used in training and automatic calibration systems:

- The characters are trained before final use on a range of human-designed obstacle courses that represent the sort of environment conditions that they will encounter when in actual use.
- The characters are trained in real-time; a time limit is taken to *kill off* agents controlling old surviving robots so that evolution can continue.

It is certainly possible to adapt the proposed genetic algorithm (GA) for training so that over a certain time interval the agents controlling animated characters are replaced by agents from the next generation - this would allow characters to adapt to changing or new conditions during a simulation's execution. There would certainly be value in evaluating whether or not this approach would improve results.

9.3 Initial Approach

In a standard-pattern fuzzy-based navigation system there are a very large number of variables than can be tweaked and would make good candidates for automatic training:

- 12 fuzzy input set membership functions (categorising distance and angle to targets and obstacles).
- cut-off thresholds for each of the input angles and distances
- 12 fuzzy output set mid-point values
- 36 adjustable fuzzy rules (six or seven discrete values possible per rule).
- resolution of grid-cells (size of the cells in meters)
- depth-limits of dynamic path-finding component of navigation system

The **knowledge-base** (*KB*) of a learning fuzzy system can be classified into two main component areas for optimisation [104];

- A **rule-base** (*RB*) consisting of fuzzy *inference engine rules*
- A **data-base** (*DB*) consisting of fuzzy *set membership functions* and set *scales*, and in the case of this system *case grid-cell resolution* and other *threshold levels* as well

The standard pattern fuzzy controllers have 3 inputs for distance and 3 inputs for angle for a total of 9 rules defining a controller. There are 4 controllers in the simplest motion controller that we look at in this thesis; obstacle avoidance steering, obstacle avoidance speed, target seeking steering, and target seeking speed. So one of these motion control systems has a full rule-base of 36 rules in total. For an example of rules in the rule-base; the first 3 rules might be expressed in natural language as:

1. If the obstacle is a **near** distance away and the obstacle is a **narrow** angle from the heading then change speed to **zero**, and change steering to a **sharp turn**.
2. If the obstacle is a **medium** distance away and the obstacle is a **narrow** angle from the heading then change speed to **very slow**, and change steering to a **medium turn**.
3. If the obstacle is a **far** distance away and the obstacle is a **narrow** angle from the heading then change speed to **zero**, and change steering to a **very light turn**.

To limit the number of variables to optimise, and to concentrate on the most manually-intensive of the variables, the system concentrates only on automatic training of the fuzzy inference engine rule-base. In the initial evolutionary algorithm (EA) the entire rule set is treated as a genetic code - this is the "Pittsburgh" approach [104, 105]. A "population" of candidate rule sets is maintained. The evolutionary system is very rudimentary at this stage; the process of mutation for the initial approach is a only very basic sequential tuning operation (only one step above a brute force method) to evaluate the effectiveness of genetic algorithms for auto-tuning motion control problems.

Considering a very simple control system that has 36 different rules, each rule having only 5 possible output options, then we have 5^{36} possible combinations of rules. If we repeat each measurement a moderate 150 times then we have to make $2.1828 \cdot 10^{27}$ measurements. Using only 30 seconds per measurement we can see that this is going to take 1.25 hours to evaluate one combination or more than $2.9 \cdot 10^{23}$ years of computational time to evaluate the entire set. It might be possible to reduce the number of measurements by rejecting all rule combinations that are not in a sensible order, i.e. when steering from a moderately close object the output should not be smaller than when steering from a far away object. Even if we do a factorial reduction thus, we still have to make at least $4.3052 \cdot 10^{16}$ measurements, which makes it infeasible to solve the problem using brute-force algorithms.

There are a large number of possible combinations of rules, but it makes sense for most of these rules to remain in sequence - therefore the approach can start with a default "reasonably good" set of hand-made fuzzy rules, and then to attempt a tweaking evaluation on these - a "tuning" approach.

The hypothesis is that this method, although not using a heuristic guide, might improve results relatively quickly compared to trying all alternatives sequentially. If each rule has 5 possible outputs; i.e. "zero" "very light", "medium", "sharp", and "full" in the case of steering outputs, then we can enumerate these as values from 0-4, and simply "tweak" a value by shifting it ± 1, i.e. from a "sharp" to "medium" turn.

A preliminary evolution process was proposed in Chapter 6 and in previous work [55]. This stage-one algorithm is illustrated again in Figure 6.7; where the method in the figure is using the time taken and a collision heuristic as a fitness (evaluation) function. Initially, the method is only focused on the most basic form of rule-set tuning; note that no pre or post scaling of fuzzy set functions takes place. We have also not yet touched upon scaling grid resolution - a much more technically involved process that would alter the balance of the whole system - current thinking is that if the cell resolution is changed, then all of the rules and sets would need to be recalibrated to suit this new resolution; thus cell size changes would need to happen in-between runs of the EA.

A test-cycle of the new simulation infrastructure produced some results for the first two rule variations by tweaking the first rule by ± 1 and running a number of prototype simulations. The goal was to build a simulation infrastructure that could generate all 70 additional ± 1 rule variations for each individual, and rank them. The best could be given a subsequent round of ± 1 tweaking. This is analogous to a depth-first search on a tree of depth 6 or 7 with $<= 72$ children per node, and where evaluation of each node takes 2-3 minutes, and a large number of runs are required. The test cycle put a character with an established set of specifications (top speeds etc.) at a random location, a set distance from a fixed target - placed in the centre of a 3D obstacle course test environment. The idea here was to bring the simulation testing framework up to working order and shake out any flaws in the system.

Numerous problems were discovered as a result of this test; the largest problem being that Nvidia 180.11 video drivers for the GeForce 8800 GTX on the Ubuntu Intrepid Linux-x86_64 platform were unstable with the graphics library; Ogre [68] would bring the entire Operating System down every 100th run or so; making the entire experiment platform very unreliable, and slow to process. An interruptible single-programme-execution experiment platform (where one execution of a programme could conduct the entire experiment) would be ideal for this kind of testing. The range and pseudo-randomness of the experiment conditions were later improved by the addition of random initial orientation to the learning agents (in addition to the random starting position).

The previous experiment with time-step size in this training simulation framework has shown that the collision avoidance system scales linearly with time compression up to the 2.0x range [93]. Therefore it may be a possibility to run the training at up to double the time compression factor to speed up the training process. Without experimenting with the concept it is hard to be confident that training at 2.0x time compression will produce navigation rules that are equally well suited to 1.0x operation as if they had been trained at that level; further experimentation is necessary to determine this, but it might be advantageous if it were possible to speed up results by a factor of 2 at only a small loss of training quality.

9.4 Preliminary Experiment

How many runs are required to obtain a reliable fitness or evaluation heuristic for a particular rule set within the evolutionary fuzzy framework? Without this information it is not possible to proceed sensibly with an automatic learning process. The easiest way to obtain a guideline for this vital parameter was to run an experiment and observe the stability of the score heuristic as the number of simulation runs increased.

This experiment was also a good trial-by-fire of the entire simulation/learning tool pipeline. To begin with a test environment was quickly constructed using the 3D design tool. The concept was for each vehicle to head from some point around a 300m radius in the environment towards a central fixed navigation target. A number of obstacles of various shapes and sizes were put into this area,

Figure 9.1: The test environment for agents to navigate through. Each run of the simulation starts the T-28E vehicle at a pseudo-random orientation and location 300 meters from the centre of the image (around the periphery of the outer buildings), and the goal is to head toward the centre of the environment (in the middle of the three large buildings). The landscape is littered with obstacles; trees, and buildings of various sizes. The vehicles must surmount hills and drive through valleys with approximated physics affecting ascent/descent speed.

and some light hills and valleys (which affect vehicle speed, but are not able to be driven around intelligently at this stage) created. A view of the test environment is presented in Figure 9.1.

Figure 9.2: The to-scale 3D model of the T-28E vehicle with continuous tracks that was constructed for use in the experiment. This is a good example of an animated vehicle that would be reconstructed for a movie production or modern computer game or simulation. All of the historical performance data is available for this vehicle, and speed, weight, tracked steering movement and other available data is simulated.

Each run of the experiment consisted of one programme execution where one agent would use our default (hand-crafted) knowledge-base and drive course. Post-processing would evaluate the agent, store the result, and start another programme execution with the next agent. We used the following parameters for all simulation runs, which were conducted on an Intel Quad-core 2.40GHz machine with a NVIDIA GeForce 8800 GTX card:

- 800x600 graphical resolution at 32 bits-per-pixel
- 3 minute time limit
- pseudo-random initial orientation (heading adjusted within a full circle)
- pseudo-random initial position - around a 300m radius from target
- 1.0x time compression factor
- using specifications for the T-28E vehicle (see Figure 9.2 for the 3D reproduction of the vehicle).
- scoring function for self-evaluation (maximising fitness heuristic)
- default, hand-crafted rule-base.

In order to evaluate the effectiveness of navigation through an environment, each agent is awarded an heuristic score upon completion of a simulation course "run". This score is the same as the fitness evaluation, except that the aim is to maximise the value, rather than to approach zero. The heuristic is a weighted combination of *collision severity* evaluation and the *time taken* to complete the obstacle course; Equation 9.1. The entire evolutionary process hinges on the effectiveness of this score as a fitness measure.

$$h = (200 - c) * w_c + (60.0 - T) * w_T \tag{9.1}$$

Where h is the heuristic score obtained, c is the cumulative mesh-interpenetration of the vehicle with boundaries of obstacles in meters of overlap per second, based on a radii from the centre of the vehicle and the object, and T is the total time period taken to complete the run in seconds. w_c and w_T are optional weights to shift the balance of the function towards either expediency or safety, but they were both set to 1 for the experiment.

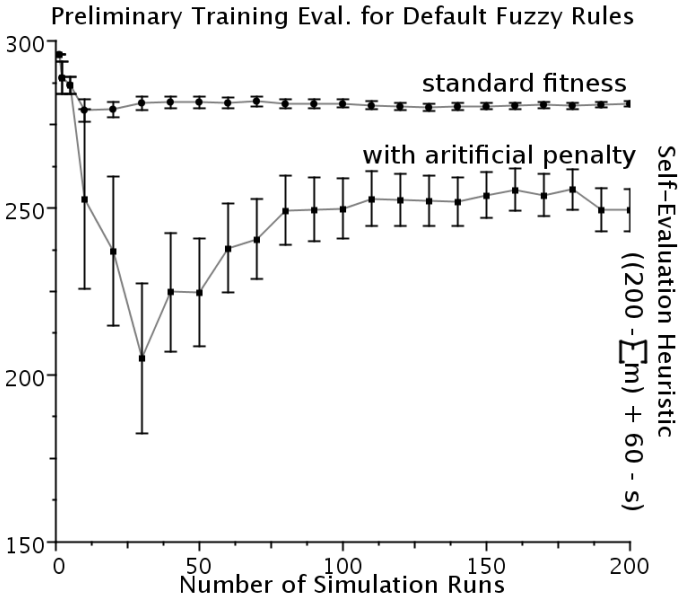

Figure 9.3: The upper plot line is the raw fitness result, and has very little error. The lower plot is the same result, but those runs that did not complete the entire course have been artificially penalised with a -1 fitness score. This second plot might be a more useful optimisation process for implementations where the vehicle absolutely must reach the destination, and the first plot where obstacle avoidance is relatively more important. Having a smaller error margin, however, will let us make an accurate fitness assessment with fewer measurements, so we can see that it is desirable to use as few exceptions to the fitness function as possible.

9.5 Discussion and Conclusions

It became quickly obvious that for the tuning problem brute force methods are too slow to be usable (tweaking 5 rules is acceptable, but with 36 rules the turn-around for complete rule trial would last 1 week per character). Note that it took 48 hours to tweak each rule only ± 1 for one agent (the full compliment is ± 3 or 4). Even basic guided search algorithms are too slow for this problem. It was found that batch-driven methods, while easy to parallelise across machines, are fraught with technical hang-ups in a 3D graphical environment, as supporting libraries occasionally hang or crash the system when fully restarted.

In this work established a rule-of-thumb number of runs to use for evaluation in this type of obstacle course has been found. Because this is only 20-30 runs (assuming that current problems are fixed) it might be possible able to train continuously in real-time; i.e. training runs occur dynamically in real-time, not as individual programme executions.

For these reasons the next iteration of this research is driven towards a new approach; single-execution simulation architectures that can evaluate multiple, if not all 72, rule-base variations simultaneously (one per-agent) within one test environment. This is illustrated in Figure 9.4 with a

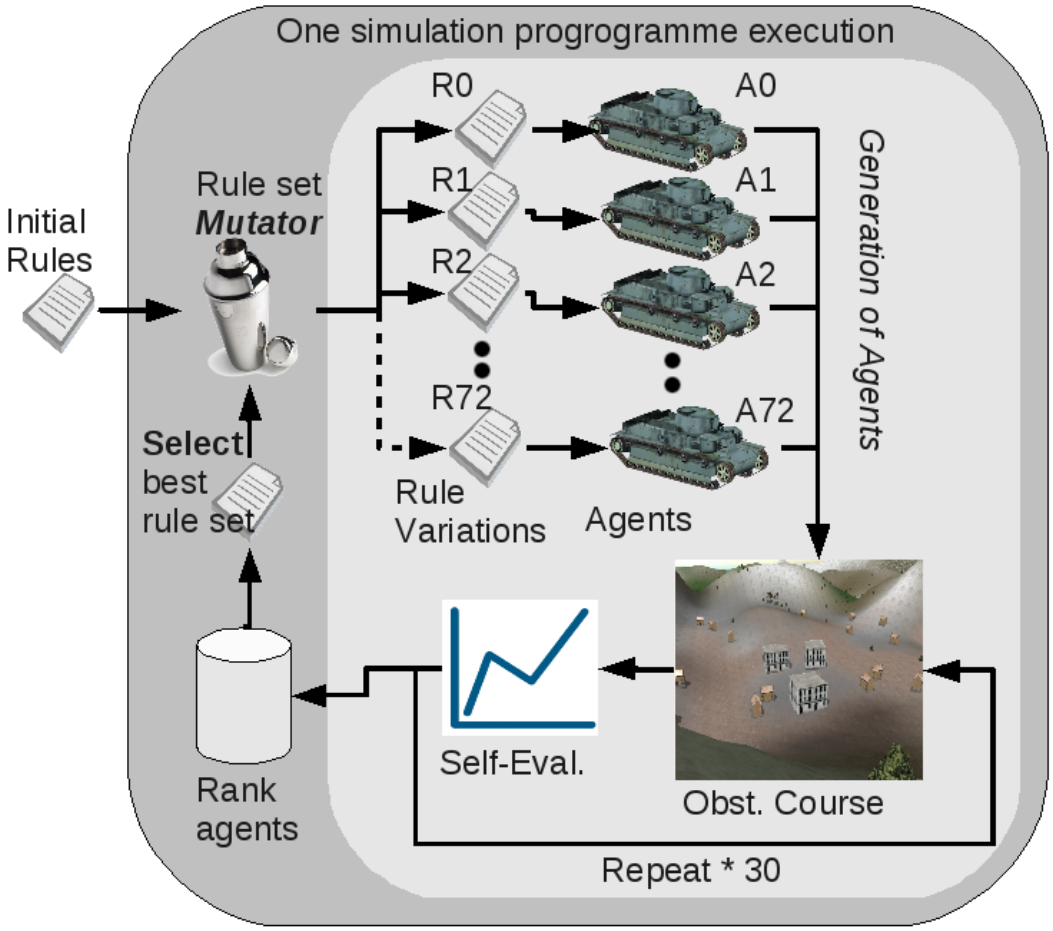

Figure 9.4: A simple architecture for a dynamic real-time training system. The advantage of this approach is that a large number of rule set variations are training simultaneously within one programme execution - with no need for training to be distributed across multiple machines. The reliability issues of training are reduced, as the programme does not need to be restarted. Agents self-evaluate after a 2-3 minute interval. An entire generation is then processed within the 30 run range which is established - just over an hour per generation. The next question that we have to ask is - "How many generations are required before we optimise training?" This architecture is then an intermediate step between traditional off-line training, and intelligent agents that learn during normal execution. At this stage only the most simple selection method is used to breed the next generation - the best agent from a generation is used and mutated; all others are removed from the gene-pool.

more sophisticated version of the EA diagram. This is a much more efficient approach (as used to some extent by the rtNEAT algorithm in the NERO game). There is then a basis to time-limit individuals and create generations with breeding of results, and pruning of the weakest members. This leads to comparison of various evolutionary methods, and it is possible to explore genetic operators - mutation, selection, and crossover - for a more effective tuning process.

This new approach asks new questions however;

- When do we stop the training process?
- How many agents can we train at once before frame-rate performance upsets learning? Or rather
- To what extent does simulation frame-rate affect learning?
- Can we train effectively at compressed time-steps to further increase learning efficiency?
- What is the best way to incorporate data-base tuning into this process?
- How vulnerable are we to being stuck in evolutionary minima and maxima traps?

All of these questions will have to be explored. Training with different grid sizes, and of tweaking the fuzzy sets will be considered (to suit the trained rules). Would it even be possible for us to develop the *holy grail* machine learning algorithm in this area - vehicles that to self-modify their own behaviour *during* execution? Our future works will examine the application of genetic-fuzzy logic hybrid algorithms to the specific areas of computer games with hostile agents, traffic simulations with road structures, and crowd simulations with very high-density groups of virtual human agents whilst moving towards a generalised auto-calibration algorithm for animated characters in 3D environments.

Chapter 10

A Genetic-Fuzzy System for Optimising Motion

> "It is not the strongest species that survive, nor the most intelligent, but the ones most responsive to change." - Charles Darwin [106]

This chapter introduces a new genetic-fuzzy system (GFS) that automates the tuning process of rules for animated character steering. An advantage of the GFS is that it is able to adapt the rules for steering behaviour during run time, which means that it is suitable for games and real time simulations where the environment can change. This chapter will explore the parameter space of the new GFS and discuss how the GFS can be implemented to run as a background process during normal execution of a simulation.[1].

10.1 Introduction

An outstanding problem with the use of fuzzy controllers for steering and moving animated characters is that the controllers need to be tuned to suit each new type of agent's combination of rôle, physical and performance characteristics, and operating environment. This means that, while the essential kernel of the system - the fuzzy decision making process - applies broadly, all the parameters of the fuzzy systems need to be tailored to suit the peculiarities of each new type of agent; be it a lumbering troll, a lane changing city bus, a tank surmounting a rubble strewn battlefield, or a crowd of pedestrians [52, 89]. Building and calibrating fuzzy controllers consists of the following steps, as according to Passino (1998) [107]:

1. Fuzzification, and defuzzification as defined earlier in this text.
2. Designing a complete set of rules that match all combinations of fuzzy input sets to fuzzy output sets.
3. Adjusting either fuzzy sets or fuzzy rules to improve the performance of the system.

This process is very time consuming for the designer. Modifying a system by trial and error based on test cases has taken 5-10 hours in applications. If there are various types of agent involved in a simulation, or an agent has to cope with a large variety of different cases then the required tuning time multiplies. There is no guarantee that hand tuned rules and sets are optimal. The fuzzy controllers are also not able to adapt to any change to the environment after tuning. Specific aims of this chapter are:

[1] A video of part of the dynamic genetic-fuzzy system experiment described in this chapter is available at `http://antongerdelan.net/videos.html` under the heading *Dynamic Genetic-Fuzzy System experiment*

1. To automate the tuning process of fuzzy controllers used for steering and moving agents in a 3D simulation, saving developer time and improving manually calibrated controllers.

2. To design an architecture that allows this tuning process to happen in real time, giving the agents an adaptive quality.

3. To enable the use of available multi core hardware during the calibration process.

The novel contribution of this work is that it provides an automatic calibration mechanism for fuzzy controllers specific to 3D animated agents. This gives fuzzy controllers some of the advantages of the other 3D game evolutionary systems, but does not share the disadvantage of being a closed system "black box", meaning that the rules evolved by the system are visible and easily manipulated by a user during the process. Future work will analyse the performance of this approach within the broad domain of agent steering and movement control.

10.2 Related Work

Optimisation of intelligent agents' behaviour using genetic or other EAs is usually performed in staged iterations; one iteration per generation. This has been used to evolve bipedal animated character motion with promising results [108] Each generation will be evaluated by running the full generation's complement of individuals through a contrived environment. These iterations will continue until a desired fitness or arbitrary generation evaluation limit has been reached. Complete training is usually performed prior to use of the agent in its intended environment, and then no further optimisation take place. This commonly used training paradigm provides a solution to the time intensive design problem, but it does not address the problem of dealing with varied or changing environments.

An alternative approach is employed in the 3D graphical NERO game [43,45]. A key innovation of rtNEAT as used in NERO is that the evolutionary training process happens continuously in real time. Rather than running the agents through staged, repeatable experiments of fixed length, the characters are given a time limit of about 1 minute, during which they are continuously evaluated, after which they are destroyed and replaced by an agent of the next generation. This speed up of the evolutionary process allows the GA to run in real time during simulation execution.

To further speed up the evolutionary process, NERO distributes training up to a large group (50) of the same type of character simultaneously (each one representing an individual of the generation). Although this work focuses on fuzzy systems rather than NNs, it is possible to build on the continuous evaluation paradigm, which addresses the problem of adapting training to a changing environment. The parallel approach to reduce training time is also incorporated. Whilst NERO's "human in the loop" is suitable for directors or game designers that want to interactively direct the evolving behaviour, this work aims to optimise existing steering behaviour whilst minimising human effort, and so aims to fully automate the evaluation process.

GFS hybrids combine a GA with fuzzy controllers, and have long been used to solve optimisation problems inherent in fuzzy systems by evolving the fuzzy set functions or through tuning of fuzzy rules [104]. However, each new GFS requires a unique, problem dependent architecture and fitness function. Genetic-Fuzzy algorithms have been used for training mobile robots to avoid obstacles. This application domain is inherently similar to animated agent steering, and the GFS approach has been shown to generate reliable fuzzy rules [109]. Although a GFS has been recently proposed for autonomous agent motion in very basic stochastic applications [110], using it for automatic training of 3D animated character navigation is a new approach which requires a new GFS framework specific to the domain.

Table 10.1 provides a comparative overview of features of different agent steering approaches. The basic fuzzy system is manually optimised, the GNN is trained in staged batches prior to use, and the final two systems are trained dynamically in real-time. An important aspect of differentiation

	Fuzzy	GNN	rtNEAT	GFS
Training	manual	batch	online	online
Black box	no	yes	yes	no
Adaptive	no	no	yes	yes
Interactive	no	no	yes	no

Table 10.1: Comparison of features of agent control systems. A fuzzy controller is compared to a genetic-neural network, the rtNEAT algorithm, and a genetic-fuzzy system.

between approaches using EAs (offline GNN, rtNEAT and GFS) is that those based on NNs are "black box" systems, i.e. the rules evolved as ANNs are not easily visible or manipulated outside the training process by a human designer. The rtNEAT algorithm as implemented in NERO counters this problem by allowing the designer to interact during the training process, but this decreases the automation.

10.3 Background: Fuzzy Controllers in Agent Steering

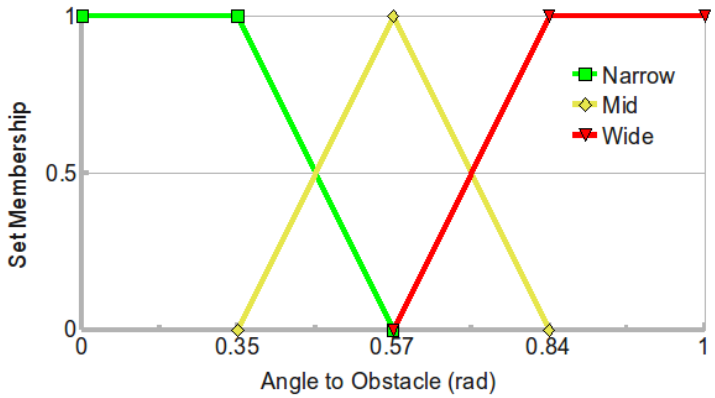

Figure 10.1: These fuzzy input set functions take a real angle to an obstacle in radians, and fuzzify this into either full or partial membership of the *narrow* (0 to 0.57 rad), *mid* (0.35 to 0.84 rad) and *wide* (> 0.84 rad) fuzzy angle values.

The primary use of fuzzy controllers is to simplify an agent's understanding of its environment. Instead of classifying distances and angles in terms of meters and degrees, for example, classifying angles in human-like terms as being members of fuzzy sets [107]. For example, we use *narrow, mid* or *wide* and distance sets *near, medium* or *far* to define a car's fuzzy sets. An example of this procedure is illustrated in Figure 10.1. For a spatial representation of this classification see Figure 10.2, where in this case the agent at the bottom of the image is classifying the angle to an obstacle from its heading direction in fuzzy terms; *narrow*, *mid*, and *wide*. Objects covered by more than one set are considered to be a partial member of both; thus the obstacle in the image is at a partially *mid* and partially *wide* fuzzy angle.

Once perceptions have been simplified into discrete forms thus, it is possible to perform some human-like reasoning by matching inputs with a fuzzy rule. The result of each rule is also a fuzzy set, but each one of these values is assigned a single midpoint value (also known as a singleton value). For example, a *sharp* turn might have a mid value of $2rad \cdot s^{-1}$, and a *very light* turn might have a mid value of $0.5rad \cdot s^{-1}$. One such fuzzy rule is:

"If a car is **near** and the angle to it is **narrow** then steer **sharply** away."

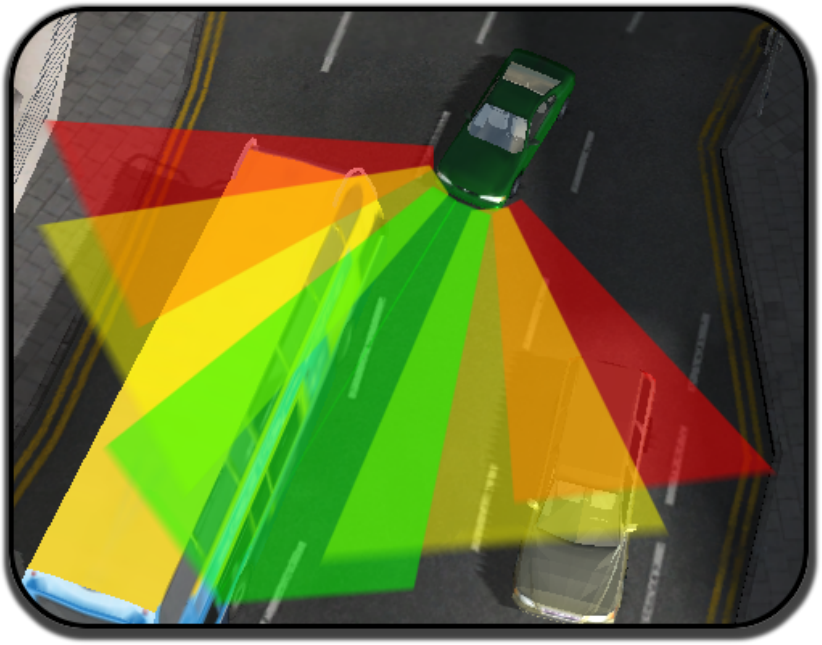

Figure 10.2: Overlapping fuzzy input sets in a spatial example. The agent has classified an obstacle as a partial member of both *mid* and *wide* angles from its heading direction.

	narrow	mid	wide
near	sharp	medium	very light
medium	medium	very light	zero
far	very light	zero	zero

Table 10.2: Rules for change to steering in the obstacle avoidance component of a simulated car.

Most intelligent agent systems would use a mathematical function to match inputs to outputs, but fuzzy systems can use a table to look up our rules very quickly. This process, which is known as a Fuzzy Inference Engine or Fuzzy Associative Memory Matrix matches each fuzzy distance and angle to fuzzy output values. As an example, the inference engine that being used for change to steering in the route following component of a simulated car in the traffic simulation is given in Table 10.2, where output fuzzy steering adjustments are given for each fuzzy input distances and angle combination.

The basic fuzzy decision making architecture used is illustrated in Figure 10.3. Complex fuzzy logic architectures for movie characters might contain trees of thousands of cascading fuzzy decision making nodes of this type, but the systems considered in this work are considerably simpler case and comprise only two of these fuzzy decision making nodes:

- A reactive obstacle avoidance fuzzy controller
- A target seeking or route following fuzzy controller

Using both of the decision making modules, environment elements (obstacles and destinations) are fuzzified into input values for angle and distance. Once these have been obtained, the fuzzy distances near (N), medium (M), and far (F) are matched to the fuzzy angles narrow (N), mid (M), and wide (W) in a rule table called a FAMM. 3 sets for each input is sufficient (which gives us a 3×3 rule table), and that the larger, more detailed FAMMs are superfluous in this kind of application,

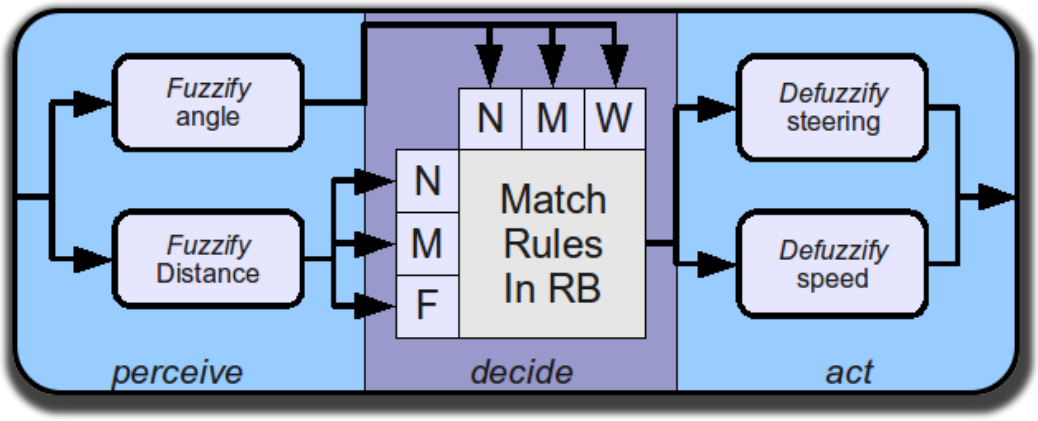

Figure 10.3: The component architecture has two fuzzy perception inputs (angle and distance), a 3×3 rule table which matches input fuzzy sets to output fuzzy sets, and two fuzzy motion defuzzifiers which convert fuzzy outputs into real speed and steering instructions.

but system is using a 5-set fuzzy output value for greater rule flexibility. The FAMM contains the complete matching for every input, and provides an output fuzzy set for each rule, for both change in steering, and for desired speed. Therefore, with 2 fuzzy decision making modules that have 3×3 fuzzy inputs each with 18 combinations of inputs, and as each of these combinations is used for 2 fuzzy outputs then in total the fuzzy system requires 36 fuzzy rules.

10.4 Architecture of the GFS

In a fuzzy system all of the problem specific variables and parameters are grouped into what is called a fuzzy Knowledge Base (KB), which consists of two distinct parts:

- the Data Base (DB), which determines the size and shape of fuzzy set functions (used for fuzzification and defuzzification).
- the Rule Base (RB), which contains a list matching every possible combination of fuzzy inputs to a valid fuzzy output.

A GFS is designed to optimise either the scale and shape of set functions in the DB or to tune the rules in the RB, as simultaneous optimisation is conflicting [104]. This architecture only optimises the RB, and not the fuzzy DB as output fuzzy values can be treated as discrete numbers, and are therefore easy to increment or decrement during mutation. As DB optimisation is a more complex operation, scaling and modification to fuzzy set functions will a the subject of future works.

One drawback of existing GA models is that they require extensive changes to simulation infrastructures when implemented directly in code. Therefore this chapter's GA is designed with a view to making minimal intrusions on the target simulation. This work separates the GA from the simulation architecture with only a small simulation plug-in used to interface with an external GA. This approach separates the evaluation and breeding functions into separate programmes, leaving only the performance logging and rule distribution functions running as part of the simulation. This means that the whole architecture has only a very small dependency on the target simulation's programming language and implementation.

Our simulation plug-in is illustrated in Figure 10.4. Referring to the figure, a "Run Manager" (RM in the figure) component loads all available chromosomes into the simulation on programme execution. During run time it distributes these to the agents, and once all runs are completed it looks

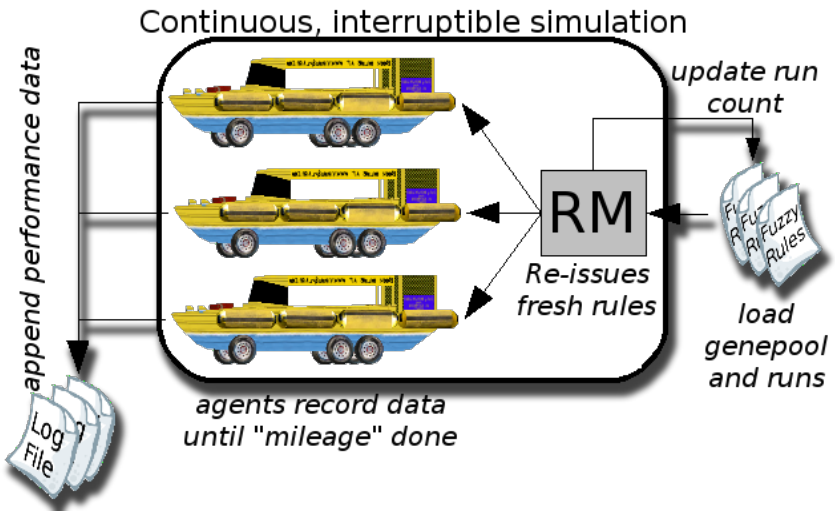

Figure 10.4: Simulation plug-in architecture. The "Run Manager" (RM) intermittently loads new rules from the gene pool. The trucks represent the agents, which append their performance to logs at small mileage intervals.

for a new generation of rule sets to load and distribute on the fly. The count of runs completed for each individual are also recorded so that if the game or simulation is stopped evaluation can resume from the same point. As indicated in the figure, agents log their performance (fitness function components) at regular mileage intervals. These can then be compiled at a later stage for fitness evaluation. This method allows continuous evaluation that is interruptible, which makes it ideal for computer games where a game might not last for an entire evaluation cycle. Because the runs are logged incrementally the training can be halted and resumed at a later stage with very little loss of training time. Using flat files as input and output makes negligible impact on simulation performance. A network loop could easily be used in place of file I/O to reduce processing time, or to distribute training across multiple machines.

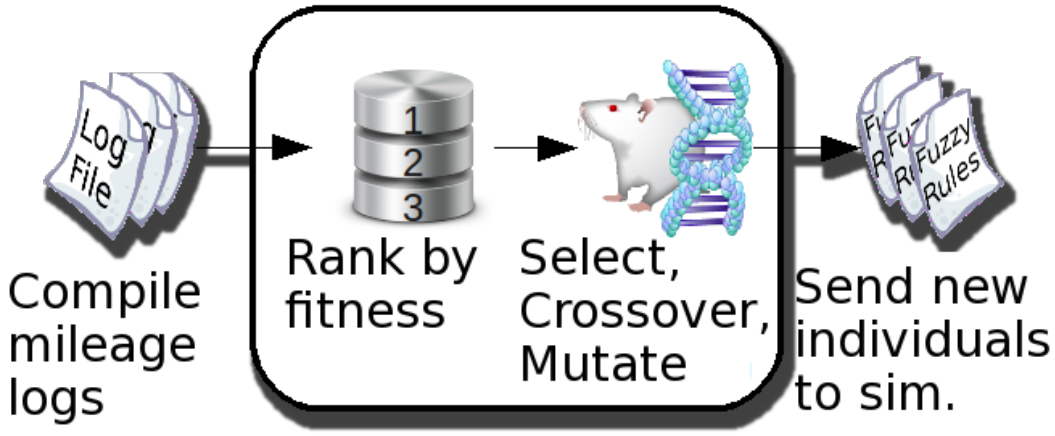

Figure 10.5: Operation of the breeding tool chain. Firstly fitness scores are created from agents' "mileage" logs. Individuals are ranked in order of fitness. Genetic operators are applied to create a new generation. This is sent to the simulation.

The actual breeding of new generations based on genetic operators is handled in an external pipeline, as illustrated in Figure 10.5. This pipeline is independent of the implementation of the target simulation. The CPU demands of the whole system are therefore very low, as the breeding pipeline does not need to remain in-synch with the updates of the simulation. It also readily takes advantage of any available multi core hardware as it occupies a separate operating system process.

This separate tool chain reads in all of the logged evaluation results output by the simulation and compiles them into complete runs. The runs are then evaluated with a fitness function, and when all of the runs for an individual are compiled, then that individual is ranked according to its fitness score. The gene pool in the architecture retains the best individuals from earlier generations, and new individuals are ranked against these.

As for genetic operators, the selection operator takes the best p parent individuals from the top of the fitness rank and these are used for breeding the new generation. For the sake of broad applicability (so that the algorithm functions consistently even when very small population sizes are experimented with) only a value of $p = 2$ has been used in experiments. The parent chromosomes are then crossed over to produce a population size of n children. The crossover mechanism moves along the chromosome one rule at a time and has a set $\frac{1}{p}$ probability of choosing either parent's output fuzzy set for each rule. Each rule has an r ("radiation level") probability of being mutated by 0 to m fuzzy output levels. This work attempts to exhaustively explore the complete range each of these GA variables.

The fitness function that used for initial experiments has been designed to be as simple as possible. It combines two heuristics; firstly a "crash rating" - the mean penetration or intersection of the character with obstacles during the run in meters, and secondly a heuristic representing the mean speed of the vehicle out of its maximum desired speed given current conditions. Each of the equation components are multiplied by a weighting factor which can be used to reward obstacle avoidance behaviour or expediency to a higher or lower degree. For initial experiments these weights have been set to 1. A perfect fitness is a score of 0 so the aim is to minimise the fitness. The fitness function is given in Equation 10.1 as the fitness awarded to an individual i:

$$fitness_i = \bar{c}^2 * w_c + (1 - \frac{\bar{v}}{v_{max}}) * w_v \tag{10.1}$$

Where $\bar{c}$ represents the crash rating, and v represents the speed of the vehicle. The weights for the crash rating heuristic and speed heuristic are represented by w_c and w_v, respectively. The equation is representing the crash heuristic in a squared form, as distance comparisons in most simulations are usually squared to avoid use of the CPU expensive square root operator. The speed heuristic takes the mean speed of the vehicle over the maximum desirable speed as given by the environment limitations or vehicle's top speed, i.e., the ideal speed, v_{max}. For an example fitness evaluation, if we have a crash rating of $0.4m^2$ and an average speed of 20 out of an ideal $30k \cdot h^{-1}$, then we award the individual a fitness of 0.7333.

The mapping of fuzzy controllers to a genetic representation (chromosome) string of 36 values is given in Figure 10.6. There are 4 rule tables in the most system used in this chapter; target seeking steering, target seeking throttle, obstacle avoidance steering, obstacle avoidance throttle. We need to have a fuzzy output value for every possible combination of inputs (c); these are represented as rules r0 to r35. If we enumerate fuzzy output values from fuzzy values, "zero", "light turn", "medium turn", "sharp turn", "full turn" in the case of steering, into corresponding integers 0, 1, 2, 3, and 4, then we can express our chromosome as an array of digits (d), which is very convenient for use by genetic operators. Thus the chromosome that we are using in this chapter has a length of 36 values. Each gene's value is in the range 0-5. Each gene represents the output of a fuzzy rule, and we know which rule it is because the chromosome is arranged in a fixed order. The figure shows how the first 9 values (r0 to r8) are taken in order from the first controller's rule table.

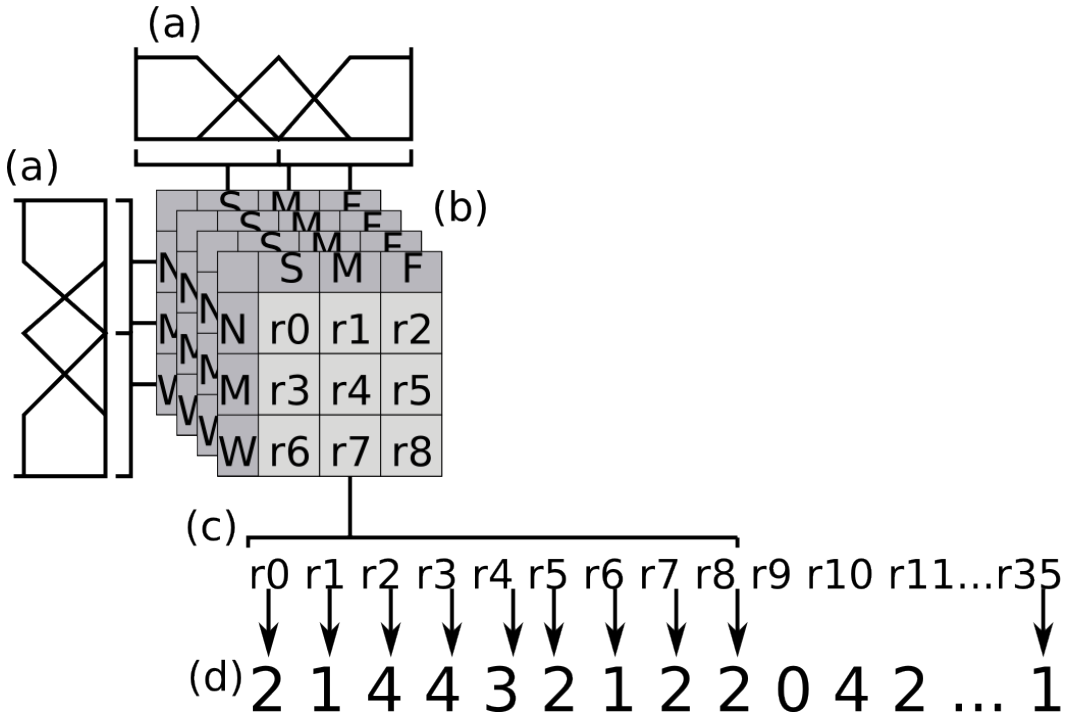

Figure 10.6: The illustration shows how fuzzy input values (a) are mapped to a 3x3 fuzzy rule tables (b) and finally to a chromosome as a string of digits (d).

10.5 Benchmarking the Genetic Algorithm Component

To help to understand the complexity of the problem, and also illustrate the evolutionary process, we first can benchmark the genetic algorithm in a controlled simulation by keeping the fuzzy processing constant. The best way to visualise if the chromosome is actually converging towards an optimum or not is to arbitrarily define a known optimal chromosome. We can say that our optimal chromosome is:

```
000000000000000000000000000000000000  fitness: 0
```

The chromosome above is 36 genes, or rules, in length. Each digit defines the output of a fuzzy rule. The order of the genes is fixed as we are only storing the output values of each. We can measure the fitness of each individual as the sum of differences of each gene from the optimal. Therefore the optimal chromosome has a fitness score of 0. For example, a chromosome that is the same as the optimal, except the first two digits are 2 and 1, respectively, would have a fitness value of 3.

To start the evolutionary process we can seed the gene-pool with 2 randomly generated parents. We will repeat the process 10 times to produce a measurement of error, but we will print just selected chromosomes from the first run of the process rather than the whole set. The seeding chromosomes from the first set are:

```
302040404224144203011100213022344032  fitness: 68
210030022302343040042212001000144142  fitness: 56
```

Using some default parameters; a population size of 10, 2 clones per generation, 2 parents selected from each generation, a gene mutation probability of 0.3, and a mutation range of ± 5 then we can run an initial benchmarking experiment to observe the genetic algorithm in action. We can print the best chromosome of the first run for selected generations to help to illustrate the process. As we

Generation	Chromosome
5	00324012204410141020100400020402030
6	00302000203201322020003010000103030
7	00203101303100401111013310000321 1030
8	01201100004200401010002230000211 0210
9	01201100004200401010002230000211 0210
10	01411000024200401040000231000400 0230
11	20410010014101400101101000100031 3200
12	20400010004200201010100100002201 1101
13	11402000004210221010100010122201 2101
14	01400000003200200013000310001220 0001
15	01400000003200200013000310001220 0001
25	10201102014000102001100010000210 3000
50	20000000001000300002300103002020 0000
75	02101100000000000000000000103100 1000
100	01200000010000100000000000101000 1001
200	00000010000000011000010002100000 0000
500	00000000000100010000001001100000 0000
999	00000100000000010000000000000100 0010

Table 10.3: The most-fit chromosomes from selected generations in an evolutionary run. The chromosomes are converging towards an optimal individual of all zeros.

will illustrate in a graph shortly, the evolutionary process follows a law of diminishing returns, so the following generations are printed at increasing intervals:

As we can see in Table 10.3, by 1000 generations we have arrived at an individual with a fitness of 4, very close to optimal. The improvement to fitness of the whole process is illustrated in Figure 10.7. So we can see that the evolutionary does work quite effectively, especially in the first few generations. The question now is; can we make the learning process faster by using different parameters (population size etc.)?

10.6 Experiments and Results

In order to establish a range of guideline parameters for implementation of the GFS, a range of experiments to exhaustively explore the larger part of the parameter space of the GA has been designed.

All of the following experiments were conducted in a 3D graphical simulation, where vehicles traversed an obstacle strewn environment similar to that found in many modern computer games. The vehicles were driven by a simple physics simulation of braking, acceleration, turning friction, hill climbing and descent. Thus, the fuzzy controlled agents had to perceive and operate in a relatively complex environment using a very simple fuzzy system. It was found that it was possible to quite comfortably distribute training over 40 agents simultaneously on an Intel Core2 Quad CPU 2.40GHz desktop machine with a Nvidia GeForce 8800 GTX card without adversely affecting the frame rate of the simulation or overly cluttering the test environment with moving vehicles.

At the start of each experiment agent-controlled vehicles were randomly scattered around the landscape, and continuously given pseudo-random way points to move to. To evaluate each agent the fitness function as introduced in the previous section was used. As depicted in Figure 10.8 the vehicles had to avoid a large variety of shapes and sizes of static and dynamic obstacle, including long walls, and other moving vehicles, whilst being forced to move through steep craters, hills, and flat areas.

The simulation was not restarted between evaluation runs, but rather the new runs were awarded randomly to available characters in a continuous fashion. All of the experiments started all of the agents with the same default hand crafted fuzzy RB, which was capable of motion but could be

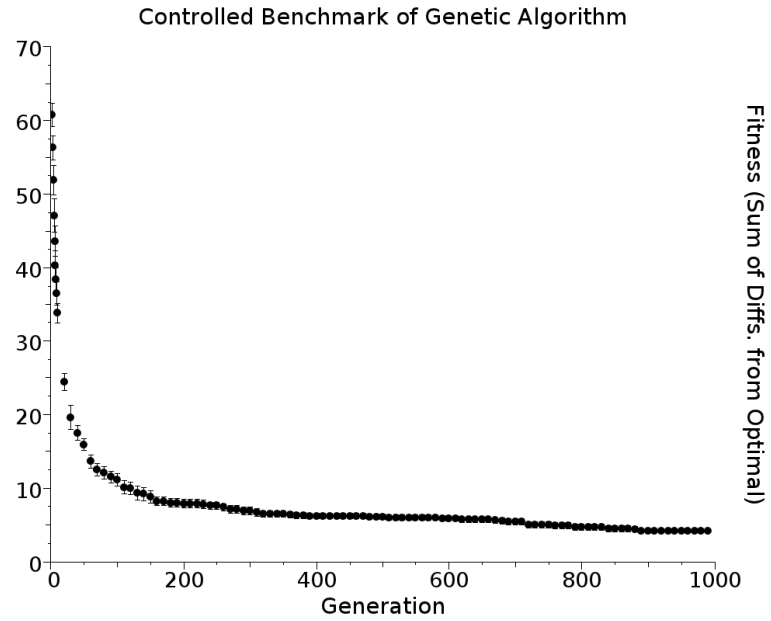

Figure 10.7: The graph shows us the learning curve, or improvement to fitness, of the genetic algorithm component in a controlled benchmarking experiment where all other factors have been held constant, and fitness is the sum of differences of a chromosome from a known optimal chromosome.

improved, i.e., were using a "rule base tuning" approach. The first experiment was designed to establish how many runs were required to evaluate each individual reliably, a baseline which would use in further experiments. Subsequent experiments evaluated the GA's variables: mutation range (*m*), where each fuzzy output value to be mutated would be adjusted $\pm m$ levels, probability of gene mutation (*r*), and population size (*n*).

The first experiment was designed to find the ideal probability of mutation *r* for each gene in the chromosome for each new individual created. All of the other GA variables were kept constant, with mutation level $m = 3$, and population size $n = 4$. As this experiment was evolving the rule base during a continuous simulation, a control case of $r = 0$ was included to observe how the changing condition of the simulation affected the fitness evaluation itself.

The experiment was run at several different levels of mutation probability, and the results of this are presented in Figure 10.9. The control case showed that the simulation stabilised after 3 generations as the characters tended to spread themselves further apart. At this point a base fitness level of 0.86 is found. The extreme cases $r = 10$ and $r = 80$ were slowest learners, with all results tending to a local minimum around a fitness of 0.3. Only $r = 20$ passed this minimum by 45 generations. Overall the results indicate that the GA significantly improved fitness over the control case. The difference in the rate of improvement to fitness made by probability of mutation is marginal, with best results at $r = 20$.

The next experiment was designed to explore the parameter space of the mutation level *m*. It was assumed that changing the m variable would make a bigger difference to fitness than other parameters, and that higher mutation levels would pass minima encountered at lower levels. The experiment conditions were the same as in the earlier experiment, but with r held constant at 20%. The results of are experiment are presented in Figure 10.10. Contrary to the assumption, it was found that lower mutation levels of 1-2 were adequate for rapidly tuning the RB. All of the different levels were trapped in the familiar 0.3 local minimum of the previous experiment, with only $m = 3$ passing this within 50 generations.

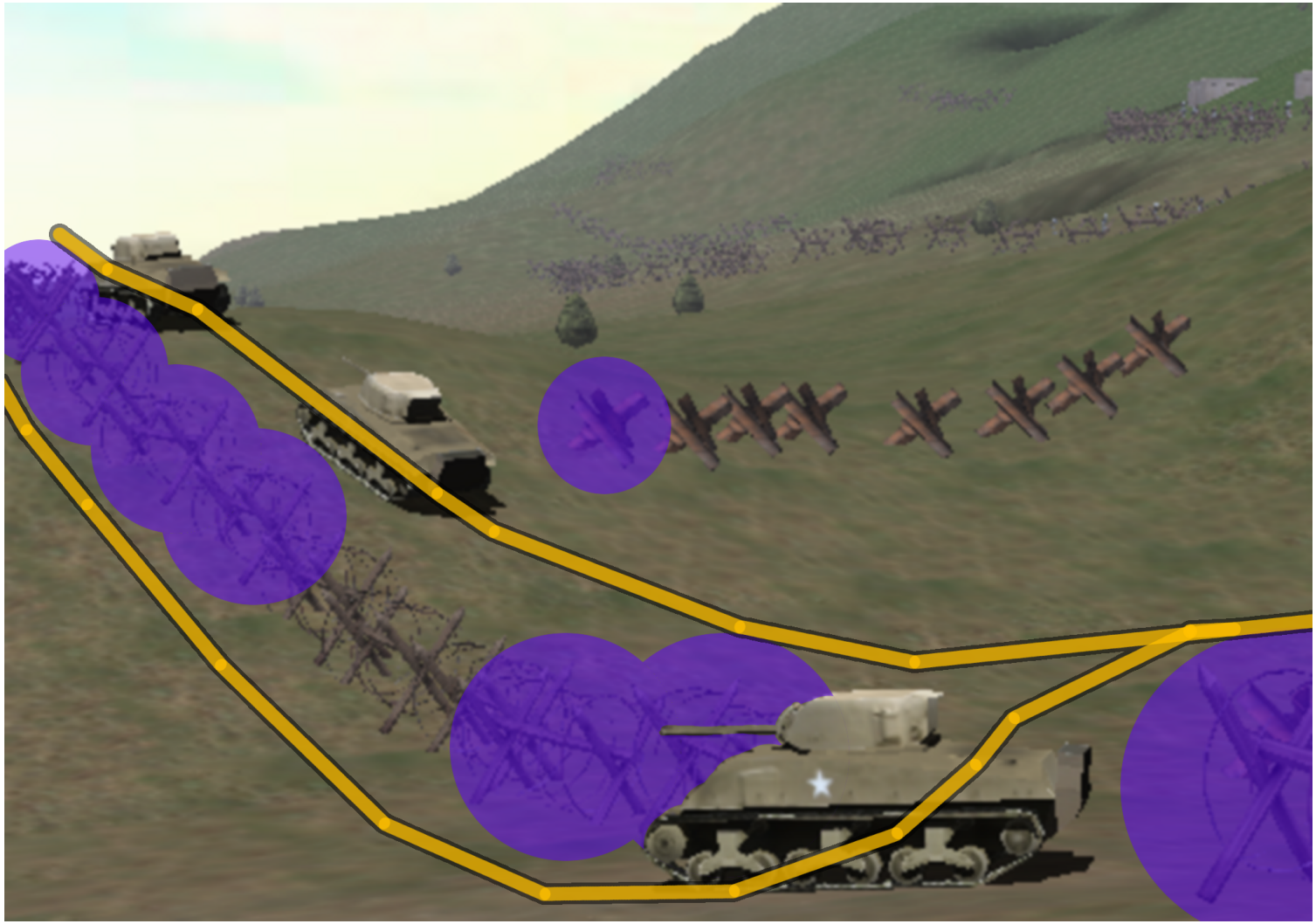

Figure 10.8: Vehicles under the control of the GFS move through the obstacle-strewn test environment. Each vehicle is being driven by a different individual from the gene-pool which means some vehicles may have a different rule-base, thus the vehicles pictured all have a common destination but have split into two streams of motion (orange paths) to move through the two obstacles fields. As each obstacle is approached and becomes the "nearest obstacle" the vehicles motion is repulsed around it; as shown by the spherical shading around selected obstacles. The environment features long strings of obstacles in different configurations.

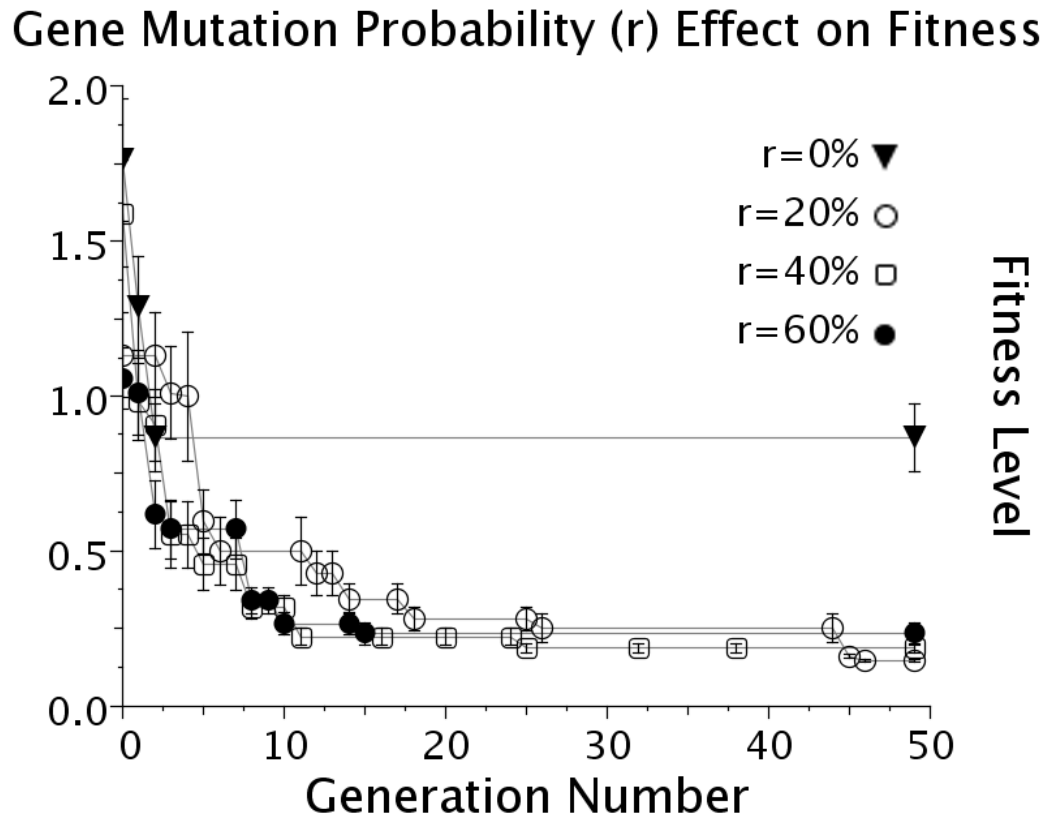

Figure 10.9: The graph compares the effect different probabilities of each gene being mutated (r) have on fitness. A control plot of $r = 0$ shows the change to fitness evaluation with no evolution. The various probability plots are clustered together, indicating that r has little effect.

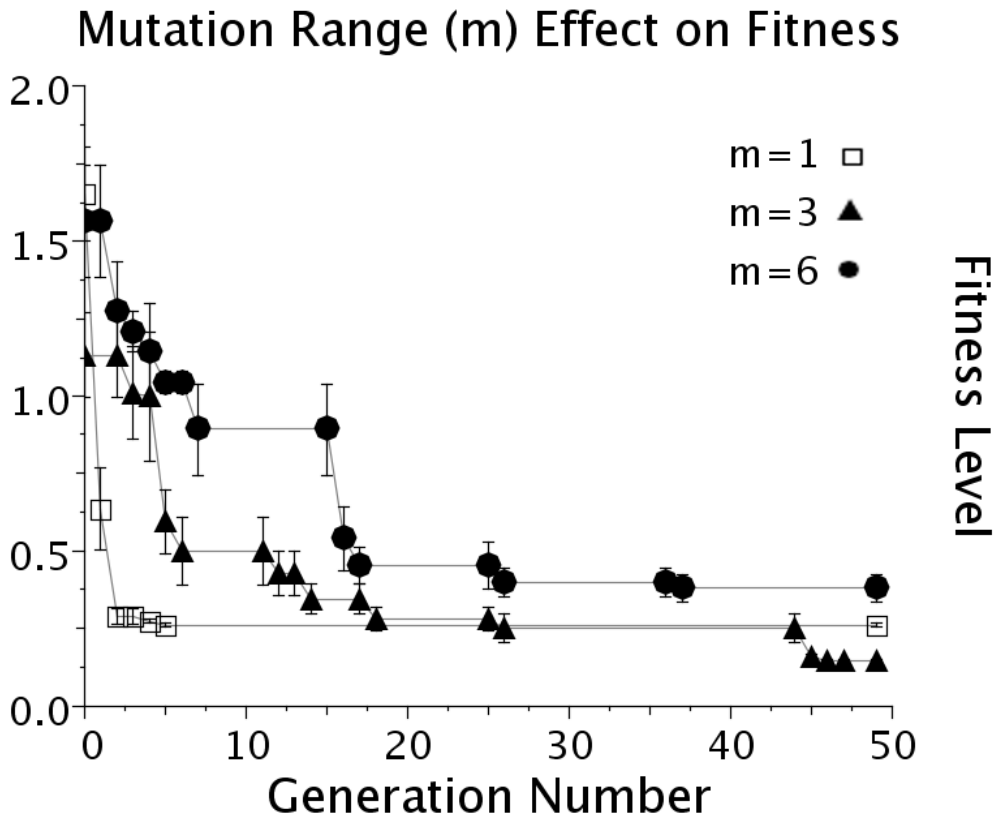

Figure 10.10: The three different plots here show points where fitness for three different mutation ranges (m) improved. When a gene is mutated, the output fuzzy set for a rule is modified between the range of $\pm m$ fuzzy set values.

An important observation of mutation levels is that, where m was set to higher levels of 5 or 6, a large proportion of the individuals produced were completely unsuitable and remained stationary or vibrating. The next step therefore, was to develop a special condition to quickly identify and eliminate these individuals in order to continue the experiment. Although $m = 6$ was capable of making very large jumps in fitness (generation 15 in Figure 10.10 for example), it can also produce a series of generations with no improvement and many unsuitable agents; it took 15 generations before any significant improvement was made to fitness. This behaviour would be highly undesirable in an interactive computer game, so it can be said that, whilst high mutation levels may allow pre trained RBs to break out of minima over a very large number of generations, for run time training where consistently good motion is desired, a RB tuning approach is better, with mutation level set relatively low (between $1 - 3$ in the GFS).

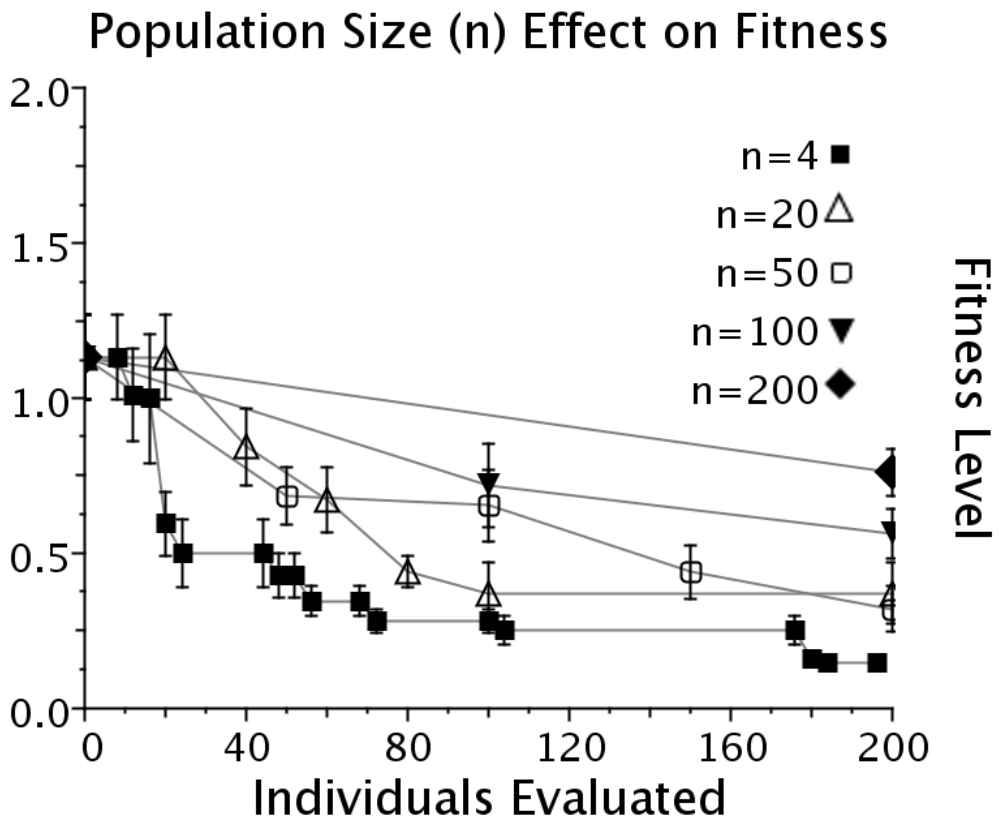

Figure 10.11: Shown here are rates of fitness improvement per evaluated individuals for different population sizes (n). We can see that the bigger n sizes evolve more slowly, with $n = 4$ being the fastest.

Our last experiment was designed to explore the population size (n) parameter space, specifically to find the population size per generation that allows us to improve the fitness score most rapidly. It was assumed that larger population sizes would improve the rate of fitness improvement beyond the very low default of $n = 4$. The results are presented in Figure 10.11, where we can see a clear indication that bigger population sizes evolve more slowly. Of course, where the number of agents being trained in parallel can train more than an entire generation of runs simultaneously then it is possible to increase the population size to scale.

This results of this experiment raise the question - should the number of parents scale with the

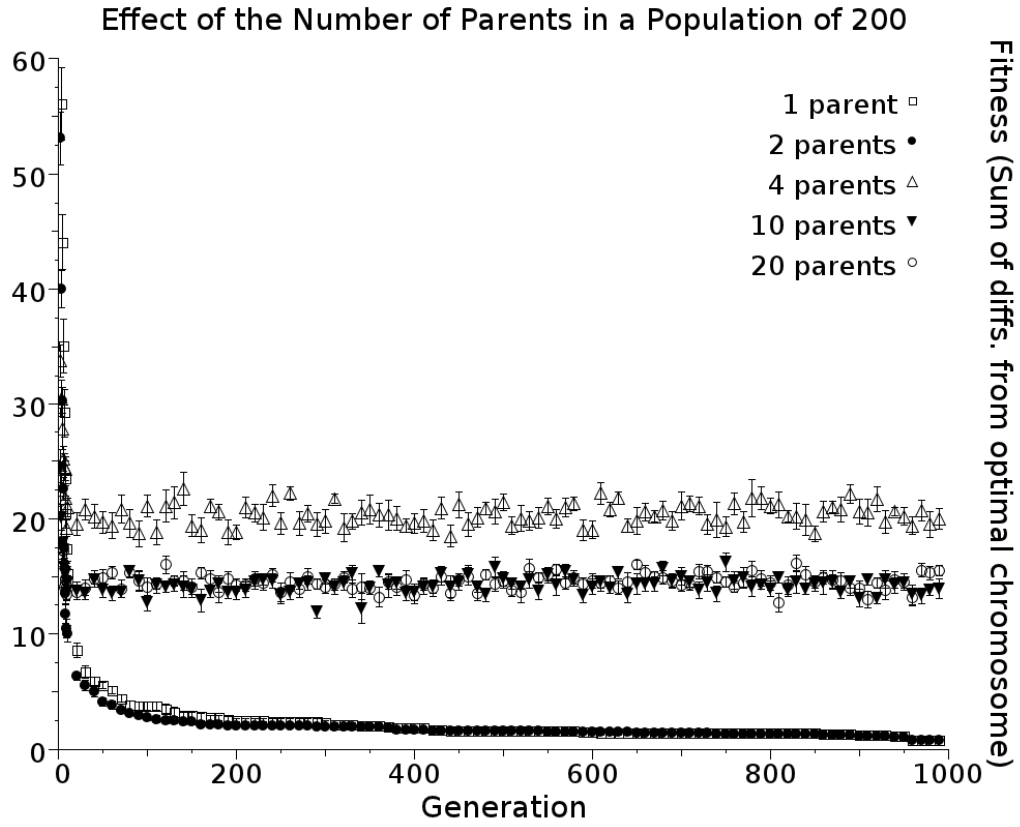

Figure 10.12: Shown here is the effect of increasing the number of parents to scale up to the largest population size from the previous experiment. Our default number of 2 parents bears resemblance to the learning curve from our initial benchmark, which we would expect. We can see that increasing the number of parents for this problem does not improve the learning curve, but in fact forces it to plateau too early. Interestingly, having only one parent (eliminating the cross-over mechanism altogether) did not adversely affect the learning curve. We can speculate that this might be because we are using a very highly selective evolutionary process for optimising to one specific solution, rather than to create a broad gene-pool for solving a range of problems, and resistant to environmental change, as we would expect in a biological evolutionary process.

Genetic Parameter	Guideline Value
Minimum Runs	30
Population Size n	4
Mutation Level m	± 3
Mutation Chance r	20%

Table 10.4: Baseline genetic parameters for GFS

size of the population? We can now benchmark if this is the case or not. Certainly, if it were the case it would skew our above results for population size. If we take the largest population size examined in the experiment (200), then we can process the genetic algorithm with various numbers of parents and compare the learning curves. Figure 10.12 gives us the results of this series of benchmarks. Firstly, our default value of 2 parents gives us a good learning curve, very similar to that obtained in the initial benchmark, Figure 10.7, which we would expect. Higher numbers of parents; 4, 10, and 20, all plateaued at worse fitness levels. We can see from the larger error bars on those plots that there was more distribution of results for these values. The most interesting information in the graph is that if we look at the plot for $p = 1$, we can see that it is almost identical to the 2-parent plot. At face value this suggests that the cross-over mechanism isn't useful at all for this particular problem space. We can conclude from the figure that scaling the number of parents with the size of the population is clearly not an advantage, in fact, the plots show us that increasing the number of parents is a disadvantage. However, it may be valuable to have multiple parents in a population where there is more than one optimal solution. In this case we might be able to use additional parents to converge to multiple local maxima simultaneously.

10.7 Conclusions and Future Works

In this chapter it was found that the GFS architecture can improve the reactive navigation behaviour of 3D animated characters over an initial hand trained set, assuming that the fitness function is a valid measure of performance. This can be seen in Figure 10.9, where all of the evolved results surpass the fitness of the control case. This suggests that the GFS is a useful tool for reducing the human calibration time requirements of agents using fuzzy controllers. It was also found that a GFS can operate dynamically, in a real time simulation, and evolve with a small population size. This offers an alternative to other EAs for games, and suggests that it might be capable of tuning intelligent agents that are based on much more complex fuzzy systems.

In addition to presenting the complete GFS architecture, a range of developer baseline parameters have been identified for use. These are presented in Table 10.4.

The next steps will be to benchmark performance of the GFS compared to other agent steering systems, and to investigate application of the GFS to a broader range of animated character types, specifically city road traffic and pedestrians, and to investigate the capability of the GFS to adapt to changing conditions in real-time. The feasibility of tune multiple complex fuzzy systems in real time will also be investigated.

Chapter 11

Mechani$x$: Vehicle Mechanical Simulation

This chapter describes a vehicle mechanical simulation system. The system has been built as a layer above the popular rigid body dynamics engine Open Dynamics Engine (ODE), which is used for collision handling and integration of classical mechanics. After reading this chapter, a reader familiar with ODE or other rigid body physics libraries should be able to recreate the mechanical simulation to the same specifications. The aim of the system is to provide a level of mechanical realism high enough to reproduce accurate acoustic and graphical representations of real vehicles in real-time, using measured performance data and known mechanical specifications. Rigid body dynamics does not represent all of the forces involved in vehicle physics, so additional physics models used are discussed. Methods for 3D visualisation of both physics primitives and high-quality graphical meshes are discussed, and details of algorithms for animating flexible body vehicles (such as "bendy" articulated buses, and Caterpillar tracks) using only rigid body movements are also provided. Special consideration has been given to reproducing realistic 3D audio accurately, and the mechanical requirements for accurate audio reproduction are also discussed here.[1].

11.1 Overview of Architecture

The simulation concept is a vehicle mechanical simulation and rigid-body physics system. The mechanical simulation model has been built from scratch but is wrapped over the ODE for integrating classical mechanics (basic physics) and collision handling.

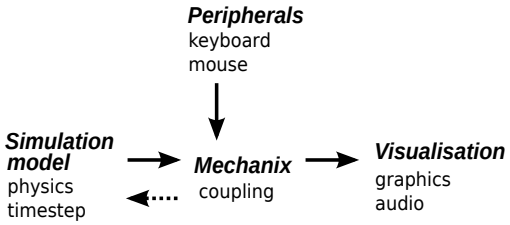

Figure 11.1: Loosely coupled simulation architecture. The layers do not communicate directly. The central "Mechani$x$" programme handles interaction.

[1]Selected videos of the mechanical simulation in use are available at `http://antongerdelan.net/videos.html` under the heading *"Mechanix" mechanical simulation*

The architecture used for simulation is a loosely coupled approach. The main programme, "Mechani$x$", performs the task of coupling together the different components (see Figure 11.1). There are three basic layers;

1. visualisation / auralisation
2. user input
3. mathematical models of the world

User input is limited to moving the viewpoint, initiating demo recording, and changing the time-step of the main simulation (represented by the dashed arrow in the figure). The layers do not directly interact with each other, and are thus modular. Each layer component has a somewhat abstracted interface wrapper, so that the back-end libraries can be interchanged but the common interface retained.

The Mechani$x$ programme itself is simply a main loop that makes application-programming interface (API) calls to each wrapper and swaps information between them. Supported back-end libraries for graphical visualisation are Ogre3D and Irrlicht. Ogre3D and Irrlicht are both high-level graphics libraries that provide an abstract interface to either Open Graphics Library (OpenGL) or Direct3D. They are both open-source and support multiple platforms. Additionally, a number of scripting functions are provided by both libraries so that various high-level graphical shading, texturing, and post-processing techniques can be quickly written as scripts. Bindings for various languages are available for each, but the model in this chapter is using the native C++.

The simulation supports and encapsulates both Fmod Ex and Open Audio Library (OpenAL). Both libraries provide a range of sound file loading and playback functionality based on the concept of a 3D "listener" which has a definite world position, orientation, and velocity. Sound sources can then be played with 3D effects such as left-right headphone panning, and the Doppler effect. OpenAL is the universally available open-source 3D audio library with support for multiple platforms. FMod Ex is a closed-source library but provides inbuilt support for more audio formats and 3D audio techniques, and wider platform support than Open AL. The audio module built for this simulation is abstracted at a high enough level to support both libraries, and provides some additional vehicle-specific 3D audio algorithms.

Again, with Input libraries, both Object Oriented Input System (OIS) and Irrlicht can be used for keyboard and mouse input. Input capture is somewhat tied to the graphics libraries. The Irrlicht engine has its own event-driven input system, and Ogre3D tends to use the OIS system. It is possible to replace these systems with another, perhaps more fully-featured, input library such as SDL, but it has been more convenient to simply wrap the functionality of both of these default choices.

ODE is used to represent solid bodies in the environment as a collection of jointed primitives (spheres, capsules, and boxes). It is a somewhat stable rigid body dynamics (covering the basic physics of classical mechanics) integrator. Performance is comparable to other physics libraries, but ODE is open-source and supports most platforms. In the simulation, the mechanical calculations (gears, engine torques etc.) are used to apply forces to the rigid bodies which are handled by ODE. The ODE library then calculates interactions between these bodies, including collisions (crashes), contact with road surface, basic static friction (rolling friction is calculated at the mechanical level), slipping, and bouncing. Every frame the current state (orientation and position) of each rigid body is read, and this is used to drive the more complex 3D graphical objects in the scene.

The mechanical operations of vehicles; torque curves, gear ratios, some spring calculations, mechanical frictions, etc. are all custom-built for this simulation. It is perhaps worth noting here that the mechanical system is also a detachable module (called *libMechanix*), that can be plugged into other projects with no dependencies. It is important to note the distinction between the rôle of ODE, which is used only for collisions, some spring calculations, and joints between bodies, and libMechanix, which tackles all of the higher-level workings of cars. The mechanical library is the critical component in the simulation (hence "Mechani$x$" as the name of the simulation), and most of this chapter is spent discussing the mechanical models used in it.

The normal operation of the simulation system, as interactions between library wrappers in one time-step can therefore be summarised as in Algorithm 1, below.

Algorithm 1 Interactions between the simulation and library wrappers (in bold font) in each time step.

loop
- **User input wrapper:** capture input.
- **Graphics wrapper:** move viewpoint based on captured input.
- **Audio wrapper:** move listener based on captured input.
- **Mechanical module:** change gear, brake, and accelerator controls based on user input or automatic system.
- **Mechanical module:** calculate engine torque, gear ratios, brakes, resistances, and final drive torque.
- **Physics wrapper:** apply forces to rigid bodies (i.e. wheels of vehicles) using drive torque etc.
- **Physics wrapper:** calculate contacts and collisions (calculate several steps of the physics integrator).
- **Physics wrapper:** get new state of each rigid body.
- **Audio wrapper:** update listener and sound sources with rigid body positions, orientations, and velocities.
- **Graphics wrapper:** update viewpoint and visible objects with rigid body positions, orientations, and velocities.
- **Audio wrapper:** playback audio.
- **Graphics wrapper:** render graphics.

end loop

If the user is driving the car then the camera can also be updated based on the state of the car's rigid body.

Additionally, the simulation can run with audio, video, and input disabled, but the physics functionality is still required. The main simulation model can run as a factor of wall-clock time (real-time) or in time steps of a fixed size for more accurate simulation or for smooth demo recording.

The appendices to this thesis tabulate the variables used in this model with corresponding values used in simulation. Table B.1 provides all of the mechanical-physical values required for creating a vehicle using this system.

11.2 Visualisation

Advances in computer graphics hardware allow us to use a number of advanced rendering techniques to increase the realism of rendering, with the expensive computation of these techniques done on the specialised graphics processing unit (GPU) rather than on the CPU. For realistic rendering both custom shaders (small scripted programmes that run on the GPU) and compositors (post-processing programmes that modify the final 2D screen image) are used. In addition to realistic rendering a method for visualising the underlying rigid body primitives, which are considerably simplified representations of vehicle geometry, has been devised to observe the computed dynamics of physics-based interactions, which subtly differ from the final rendered results.

Whilst graphics hardware has become very powerful, simulations tend towards rendering large number of low-complexity meshes, each one of which stacks up on the CPU before it reaches the graphics hardware. The major graphical design challenge is therefore to reduce the number of individual draw operations by combining meshes or using a level of detail (LOD) approach to simplify or hide objects out of camera focus.

Rigid-body interactions (contact computation) are computationally expensive. Physical representations of vehicles are therefore simplified as much as possible to reduce the number of inter-

Figure 11.2: Views of a simulated Willys MB jeep. The top row visualises the jeep's approximated geometric shape (outer grey cuboid) and mass distribution (inner orange cuboid). The wheels are approximated as geometric spheres with uniformly distributed mass. The bottom row is the same jeep's final appearance using a 3D mesh.

actions calculated. Simulated vehicles are typically represented as a single box-shaped body, and several spherical "wheels" (which generally only require one contact per wheel). Figure 11.2 gives us a comparison between a graphical model of a jeep, and its simplified rigid-body representation. The Mechani$x$ system has a set of pre-defined graphical primitives (spheres, boxes, etc.) which are scaled to the size of the rigid-body representation. The primitives can be semi-transparent to allow internal components, or bodies behind the vehicle to be observed. This gives us a method for directly debugging what interactions are happening in the dynamics simulation. We can see in the figure that the graphical shape of the jeep does not directly correspond to the shape of the rigid-body vehicle; particularly the width of the wheels and the height of the body. A scripted interface is provided to allow both graphical and physical models to be displayed simultaneously. This is useful where the differences between physical and graphical model need to be determined, or where the graphical model needs to be scaled to correct dimensions.

Using the Ogre3D library it is possible to define computer graphics shaders as material script templates [68]. The different car components can be split into different material scripts; allowing different surfaces of a vehicle to have their own shaders and rendering properties.

Figure 11.3: Before (above) and after (below) effect of the car paint shader. Original image Tom Van Eyck. Image used with permission.

The most important aspect of realistic car rendering is the modelling of car paint; which tends to be glossy and highly-reflective. Car paint has two layers:

1. An undercoat of matte (non-shiny) paint
2. A topcoat of highly specular polished paint

The visual properties of layering of real car paint are complex, so a computationally simplified model that approximates some of the effect is needed. The shader used is illustrated in Figure 11.3.

The diffuse component of the car paint is modelled with Half-Lambert lighting [111, 112]. This technique is not physically accurate, but it provides an approximation of sufficient quality that does not end up appearing too flat. The specular model of the car is made up of two components; a broad highlight, and a focused highlight. The broad highlight has colour slightly lighter than the diffuse surface paint and is rendered using the Cook-Torrance model [113]. Cook-Torrance provides a "roughness" parameter which allows us to manipulate the highlight's coverage area and intensity. The focused highlight is more intense, and is modelled with Phong [114] lighting. Finally, visible reflections on the car are approximated with a cube-shaped environment map, where a Fresnel [115] term is used to determine the level of reflectivity based on the angle to the viewpoint. The surface normals are perturbed slightly to prevent perfect mirror reflections.

The *bloom* compositor effect is used as a rendering post-process to create feathers of light around the very brightest or shiniest objects. The bloom algorithm works by passing a 3x3 pixel kernel (window) over the rendered image, and convolving the pixels with a Gaussian blur filter [116–118], simplified for real-time rendering. The end effect is a smudging of light colour over nearby colours, which produces a glare effect; the strength of which can be tweaked. Some examples of cars rendered using all of the above-mentioned techniques are provided in Figure 11.4. The bloom effect is especially pronounced around the edges of the cars in the figure because of the pure white background. Note that the angle of these images has been taken to maximise the shader-induced brightness on the car surfaces, and therefore the bloom effect as well; the rendering technique's effects vary dramatically as the angle between the surfaces, the light sources, and the camera changes.

The shader effects do not work as well on low-polygonal, and squarer-shaped vehicles, and are not appropriate for vehicles that would have a non-glossy surface. In these cases an alternative technique is to use a combination of specular highlights and bloom post-processing to create a similar effect albeit without the sky-reflective properties. Figures 11.5 and 11.6 illustrate these alternative rendering techniques. The most obvious difference between the pre-bloom rendering and the post-processed examples is the level of brightness added by the bloom process; the pre-bloom rendering has been deliberately darkened to compensate for bloom, thus the difference to overall light level can be disregarded. We can see in Figure 11.5 that the flat-shaded bus can be improved with the bloom post-processor, which effectively creates a colour gradient when rendering the larger panels on the roof. The vehicle in Figure 11.6 has some shiny surfaces, particularly on the wheels, and we can see that the specular effects are considerably enhanced by the bloom post-process (right hand side of the figure).

11.3 Drive-train Simulation

We can walk through the complete torque calculation process by stepping through the diagram in Figure 11.7. The architecture is based on two controllers (Motor and Transmission). Each loads its own properties and functions. The transmission has a set of gear ratios which factor the motor's output torque, and angular velocity inversely. A differential gear with its own ratio has been added to the end of the transmission to make simulation of real vehicle's easier; the real values can simply be plugged in.

Where possible real performance data can be used to drive the vehicles, as in Figure 11.8. The idea is to approximate the torque curve (the relationship between frequency and torque) for the simulated motor. A linear interpolation model is used (it will be a quadratic in future works) and simply read a series of defining 2D points from a script; the system will read as many points as is required to define the curve, for example:

```
addTorqueCurvePoint: 600 150
addTorqueCurvePoint: 1000 250
addTorqueCurvePoint: 2000 300
addTorqueCurvePoint: 4000 140
```

Figure 11.4: .
Views of several mechanically simulated vehicles rendered with the shader implementation and “bloom” post-processing effect. The shaders are not as effective on low-polygonal meshes, as some harsh edges are visible in the car in the top-left.

Figure 11.5: A vehicle rendered with no shaders and no specular highlights (left), and with the bloom effect but no highlights (right). Note colour gradient on roof.

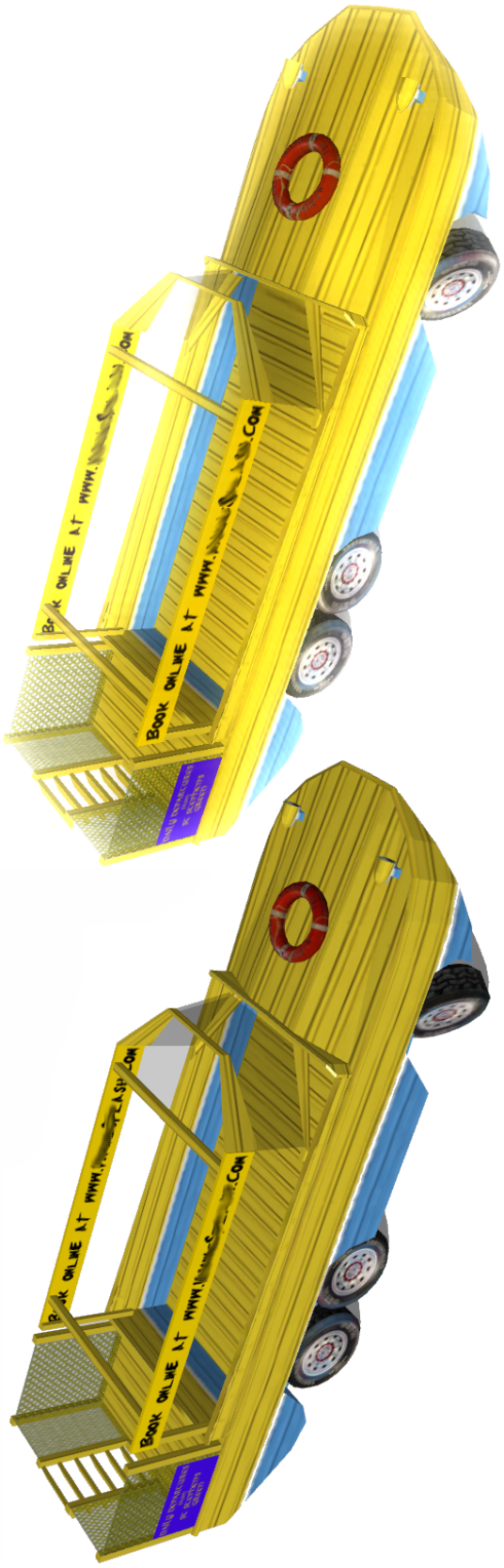

Figure 11.6: A vehicle rendered with no shaders but with specular highlights (left), and with the bloom effect after specular highlights (right). Note shiny appearance of wheels.

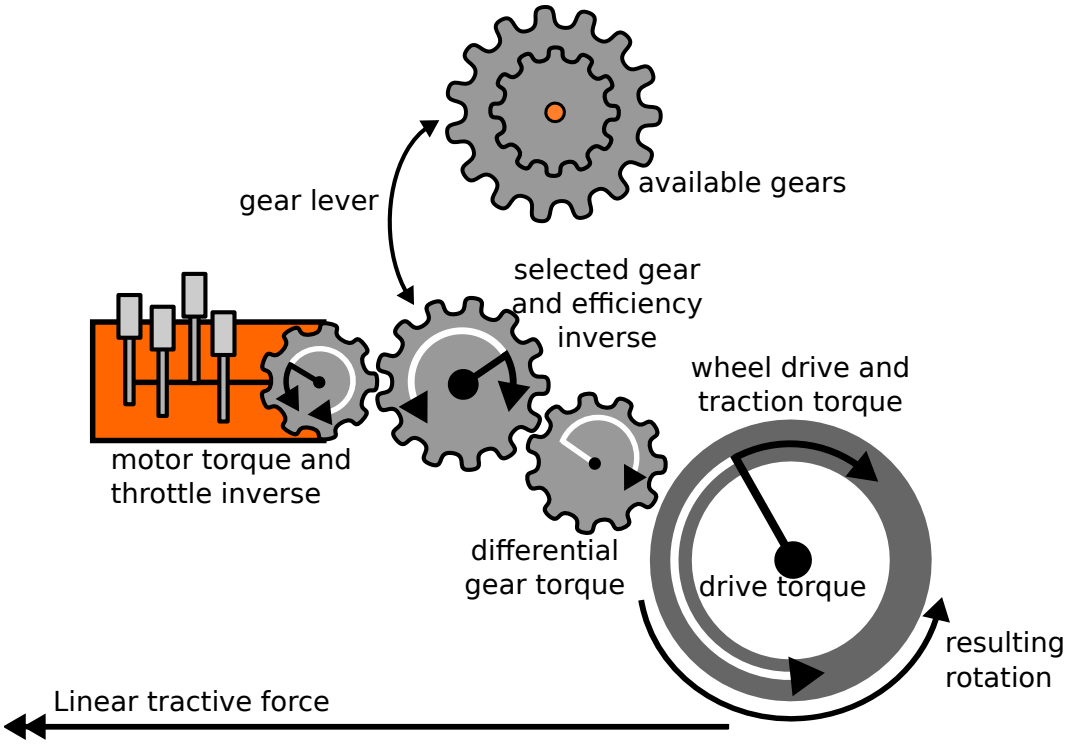

Figure 11.7: The complete drive-train assembly used with all torque-affecting forces and their opposing forces shown.

Where a series of RPM values have been defined in the first column, and their corresponding torque values in Newton-meters in the second. Note that these values have been converted from the foot-pound units of Torque used in the Figure. The linear interpolation function for fetching intermediate torques is:

$$\tau_{engine} = \tau_a + (f - f_a)\frac{\tau_b - \tau_a}{f_b - f_a} \tag{11.1}$$

Where τ is the output torque based on current frequency f. b and a refer to the higher and lower defined points, respectively, on the torque curve graph. The reason for using a linear model is simple; real recorded engine data is usually recorded as a set of 2D points, and these can then be used directly. The linear interpolation function is also computationally cheaper than solving a quadratic equation approximation of the curve.

The transmission multiplies the engine output torque by the selected gear ratio, and the differential ratio (an extra gear) to produce a range of effective output torques as in Figure 11.9. The gearbox for each vehicle can be scripted, as well as the final drive, and a blanket efficiency rating:

```
addReverseGear: torqueRatio: -3.489 wheelsEngaged: 4
addForwardGear: torqueRatio: 4 wheelsEngaged: 4
addForwardGear: torqueRatio: 1.551 wheelsEngaged: 4
addForwardGear: torqueRatio: 1 wheelsEngaged: 4
addDifferentialGear: 5.38
transmissionEfficiencyFactor: 0.7
```

Here in the script a collection of gears are added, with their ratios (factor of engine output torque), and the number of wheels engaged by the gear. This allows us to switch from 4-wheel drive to 2-wheel drive, for example. The final gear is the differential. Real vehicle data will often give you

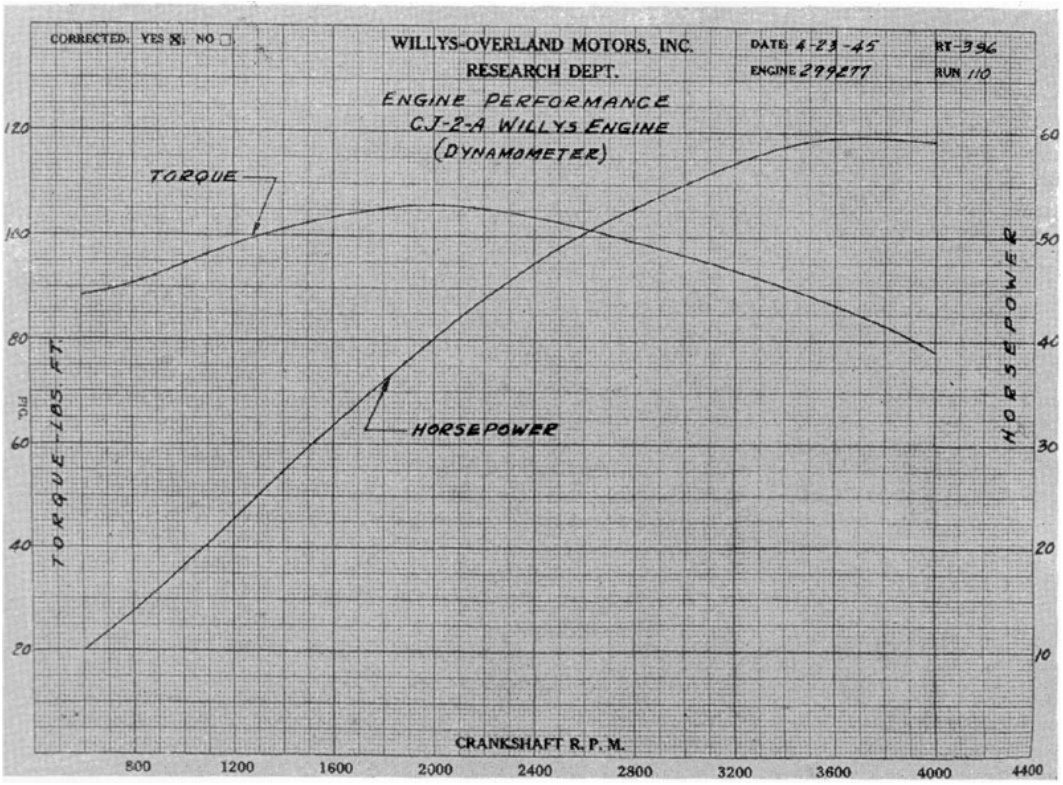

Figure 11.8: The RPM-torque curve for the Willys MB Jeep. The second plot is an RPM-power curve for the same engine. Source: Willys 1945 CJ2A Maintenance Manual.

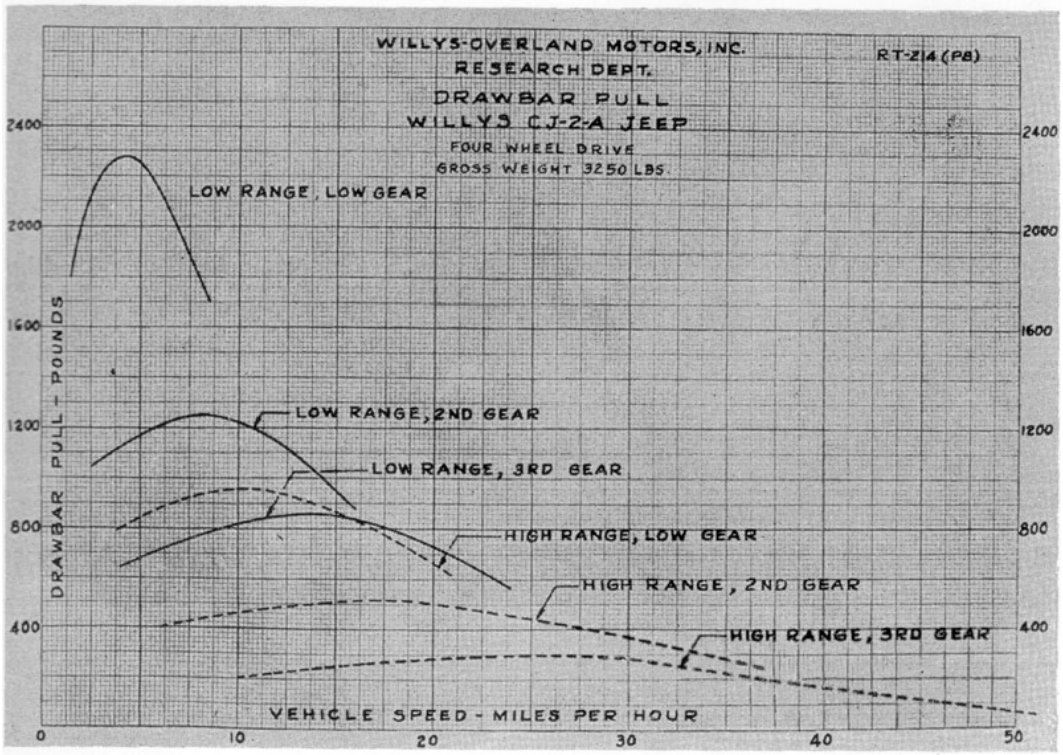

Figure 11.9: The RPM at final drive (proportional to vehicle speed) for each gear of the Willys MB Jeep.

each gear ratio, as well as the differential ratio. It's convenient to be able to just plug all this straight in to the script. The final line is a blanket efficiency rating. Real vehicles lose a lot of energy in the mechanical system - this represents that to dampen the output and keep it in realistic limits.

We can give the torque calculation for the whole drive-train, for each wheel independently, (Equation 11.2), again as a function of engine frequency f.

$$\tau_{drive} = \tau_a + (f - f_a)\frac{\tau_b - \tau_a}{f_b - f_a} \cdot x_g \cdot x_d \cdot e \tag{11.2}$$

Where the final drive torque τ_{drive} (sum of the torque on each wheel) is based on the engine's output from Equation 11.1, and multiplied by the ratio of the currently selected gear x_g, the ratio of the differential x_d, and a variable representing the blanket efficiency of the transmission e. The drive torque is then divided by the number of engaged wheels n (i.e. 4 in a 4x4 vehicle) and applied to each wheel.

11.4 Resistance Forces

Opposing the torque generated by the drive-train two main resistance forces are simulated:

- total rolling resistance (of entire drive-train + wheels)
- aerodynamic drag (wind resistance)

Apart from brakes, these are the only forces that gradually slow the vehicle when the throttle is disengaged. Without these forces it would also be impossible to maintain a constant speed when the throttle is held at a set position; the constant speed is arrived at when the resistance forces reach equilibrium with the generated torque force.

As we will see in the following subsections, drag force is proportional to the square of velocity, which means that at higher speeds drag becomes exponentially stronger. Rolling resistance is proportional to velocity. The two resistance forces converge at approximately $30m \cdot s^{-1}$ [119].

The standard fluid dynamics drag equation is used to model aerodynamic wind resistance in simulations, given in Equation 11.3, below.

$$F_d = \frac{1}{2}\rho \cdot v^2 \cdot C_d \cdot A \tag{11.3}$$

Where F_d is the total linear drag force, ρ is the mass density of the fluid; at an approximated $1.2kg/m^3$ for air, v is the forward speed of the vehicle, and A is the frontal area of the vehicle in m^2.

To simplify the maths used in this model it can be assumed that the vehicle will almost always be travelling forwards when at speeds where drag would have a significant effect ($> 40k/h$). We can therefore simply provide a constant frontal area A, and a fixed coefficient of drag for each model of car. This data can be easily obtained for most makes of production vehicles. It would be very difficult to accurately simulate the coefficient of drag for arbitrary angles of attack, as this coefficient of drag data is obtained from wind tunnel testing, and usually only the front-on data for well-known vehicles is published.

Additionally, it can be assumed that the air density ρ will be fixed throughout a simulation. This can however be changed between simulations to simulate the effect of different climates on both resistance forces and acoustic properties if that level of accuracy is desired.

The consistent component of the drag equation (independent of velocity) can then be defined as in Equation 11.4, below. This will be used in the next section for the rolling resistance estimations, which puts rolling resistance somewhat proportional to drag.

$$C_{drag} = \frac{1}{2} \cdot \rho \cdot C_d \cdot A \tag{11.4}$$

The rolling resistance model ties together all of the internal mechanical resistances of the vehicle (axles, gears, motor, etc.) as well as each wheel in contact with the ground. These resistances are all tied together into a single coefficient of rolling resistance C_{rr}, which is linearly proportional to the speed of the car as in Equation 11.5 below;

$$F_{rr} = C_{rr} \cdot v \tag{11.5}$$

A major assumption can be made that, as we know the forces converge at approximately $30m \cdot s^{-1}$ [119] then it can be said that C_{rr} is roughly 30 times greater than the pre-calculated, known value of C_{drag} from Equation 11.4. In practice this estimate for C_{rr} is too large, as the simulated Willys MB Jeep was not able to reach its known top cruising speed, but it provides a ballpark estimate within a factor of 10, which helps to reduce trial-and-error tweaking.

11.5 Effective Torque

An effective torque τ_f can now be provided, balancing engine output with resistance forces. See Equation 11.6 below;

$$\tau_f = \tau_{drive} - \tau_{rr} - \tau_d \tag{11.6}$$

Where τ_{drive} is the total drive-train output torque as described in Equation 11.2. τ_{rr} and τ_d are the rolling resistance force F_{rr} and drag force F_d respectively, expressed as angular forces.

This gives us a torque that can be delivered to the wheels. The effective torque is divided by the number of engaged wheels n and applied to each wheel as in Equation 11.7. Note that the unpowered wheels are not subject to separate rolling resistance; the total resistance has been divided only between engaged wheels in this force approximation, which is noticeable in rare circumstances when some wheels are not in contact with the ground. The full torque model can now be given as in Equation 11.8.

$$\tau_{wheel} = \frac{\tau_f}{n} \tag{11.7}$$

$$\tau_{wheel} = \frac{\tau_a + (f - f_a)\frac{\tau_b - \tau_a}{f_b - f_a} \cdot x_g \cdot x_d \cdot e - C_{rr} \cdot v - \frac{1}{2}\rho \cdot v^2 \cdot C_d \cdot A}{n} \tag{11.8}$$

Where, once again f is the current RPM at the engine, which interpolates between linear defined lesser and higher points a and b, respectively, to find a torque τ from the torque curve chart for a particular make of vehicle. Where x_d and x_g are the gear ratios, e is the blanket transmission efficiency, C_{rr} is the coefficient or rolling resistance (roughly 30 times C_{drag}), v is our current speed, ρ is the air density (about 1.2), n the number of engaged wheels (e.g. 2 or 4 for 4-wheel drive), and C_dA is the drag profile of front coefficient multiplied by front cross-sectional area A; a standard rating model for production vehicles (also commonly referred to as C_xA in the automotive industry).

Some care must be taken to ensure that resistance forces always oppose the current direction of travel; otherwise reversing, and movement in neutral gear (when drive torques are zero) or opposing current gear direction can be confused. We can simply investigate the current angular velocity ω on the x axis of one of the wheels, and negate the resistance forces whenever ω is found to be negative.

11.6 Suspension Simulation

All vehicles are simulated with all-wheel independent suspension (front and rear). This is simply the easiest model to implement as it can use ODE's spring constants per-wheel without requiring additional linkage calculations. It is also a fair approximation of most modern vehicles' suspensions.

Each wheel on each vehicle can be given a spring constant k in that vehicle's script. When this has been provided, springs are approximated by Hooke's Law [120] (see Equation 11.9 for the ideal model). Wheels that have no spring constant are simulated with no springs (rigid joints).

$$F_s = -k \cdot x = m\frac{\delta^2 x}{\delta t^2} \tag{11.9}$$

The equation above gives us the ideal simple harmonic oscillator, where F_s is the resulting oscillatory force acting on the system, k is the spring constant, and x is the displacement of the sprung object (the wheel) from the base. The differential equation relates this to the mass of the wheel m and change in time t.

A damping coefficient of velocity c can also be provided, and in this case the model is also subject to a damping force F_d (see Equation 11.10).

$$F_d = -c \cdot v = -c \cdot \frac{\delta x}{\delta t} \tag{11.10}$$

Where c is the ideal viscous damping coefficient given in Newton-seconds per meter ($N \cdot s \cdot m^{-1}$).

After parsing spring constants and coefficients from a vehicle's script, these are converted into ODE's native error reduction parameter erp and constraint-force mix cfm using Equations 11.11 and 11.12, and simulated by ODE.

$$erp = \frac{t \cdot k}{t \cdot k + c} \tag{11.11}$$

$$cfm = \frac{1}{t \cdot k + c} \tag{11.12}$$

Where t is the size of the time-step in seconds, and k and c are the spring constant and damping coefficient, respectively.

An anti-roll parameter, in Newtons per meter displacement, can be given to each opposing set of wheels. This simulates anti-sway bars, which in real vehicles are a kind of torsion-bar spring (twisting bar) that runs under the vehicle between wheels on each side, and restores balance when the vehicle starts to roll over. The wheels on the vehicles are actually spheres, so they are always trying to flip over. To counter this the vehicles can be given super-powerful anti-sway forces. The default value provides $20kN$ of restorative force to each wheel for every meter of difference between left and right side. ODE is very bad at handling large forces, so if this force gets much bigger then it will break the physics integrator.

A spherical model is used to represent both the shape and uniform distribution of mass for the wheels. Each simulated wheel combines the mechanical wheel and approximates a tyre, and has a mass m in kilograms, a softness approximation using ODE's constraint-force mix (cfm), a Coulomb dry friction coefficient property, and a side-slip Coulomb dry friction coefficient property.

Cylinders would be the most natural option to choose for simulating wheel geometry, but are also very expensive as multiple points are in contact with the ground at once. Tyres do not retain a consistent cylindrical shape during travel, and are therefore not always well represented as cylinders. ODE's integrator does not handle this sort of collision efficiently.

Another option for simulating wheels is a ray-casting method, where no wheel geometry is directly simulated but contact with the ground is manually derived from collision-detection rays projected from a normal to the underside of the vehicle chassis. This is a computationally fast method but does not accurately simulate wheel contacts, especially in multi-point contact situations (e.g. climbing stairs).

Limitations of the spherical wheel model are increased computational cost over ray-based models, the possibility of colliding with objects close to vehicles' sides due to protruding spheres, reduced vehicle tolerance to rolling over, and a restorative ability to recover from rolls that is only possible with spherical geometry.

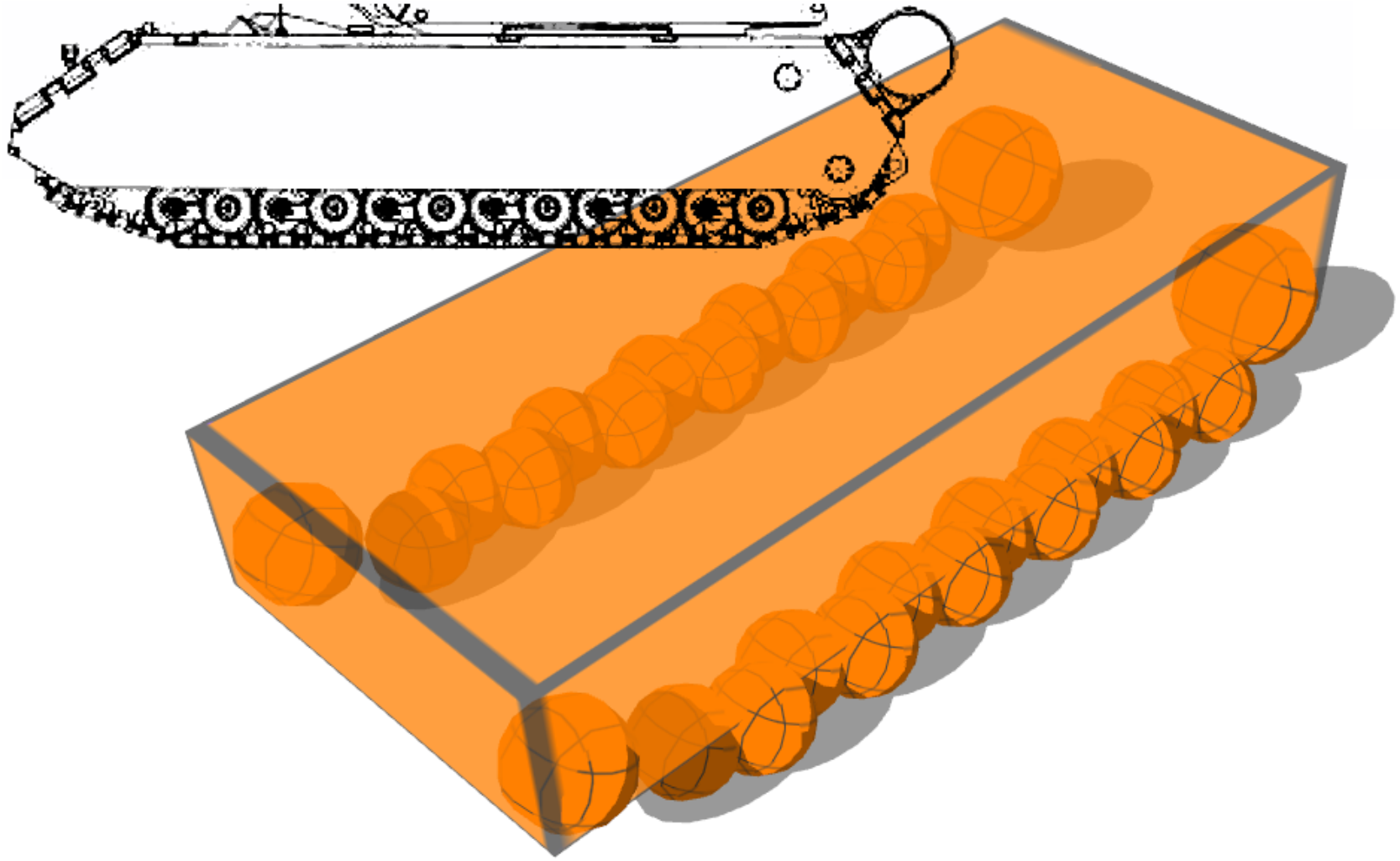

Figure 11.10: Overlaid is a mechanical blueprint, and depicted is its physically simulated shape; the tracks are approximated by spheres at road wheel positions but have slightly larger radii than the actual wheels to make up for the thickness of the tracks.

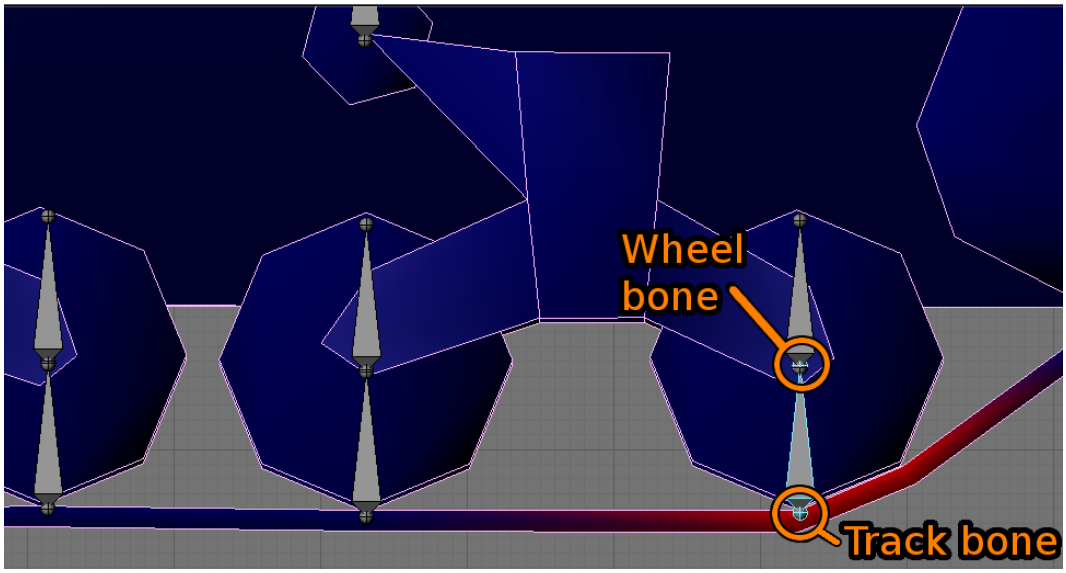

Figure 11.11: Associating segments of continuous track (highlighted red) with an armature bone. When the physically simulated wheel moves vertically then the nearby bone will move; animating the track deformation. The idler wheel at the top-middle of the figure has no physical function (the tracks have no weight), but has its own wheel bone that can is animated to turn at the correct angular velocity relative to the road wheel. The sprocket wheel to the right of the figure could be animated as with the idler, but here it has been simulated as a wheel so that it also has a collision shape and can simulate traction using the elevated rear of the track; tall obstacles can be climbed or drawn under the track.

The mechanical simulation can support a flexible-body approximation. This is part physical simulation and part graphical animation. The model is simplified physically for CPU efficiency and animated as a visualisation post-process. Efficiency and batch count, particularly respecting real-time constraints are very high for vehicles with many moving parts if each one is a separate mesh as per the physics visualisation system. Therefore in final implementation it is much more efficient to reduce batch count to one rendering operation per vehicle or as low as possible. Figure 11.10 displays a physical model *before* flexible bodies are animated. We see can in the figure that the road wheels are larger than the real mechanical wheels; their size also approximates the thickness of the track links and makes physical contact with the ground. The flexible belts are added later as a visualisation layer - thus the tracks are simulated at the lowest level by bigger spheres. The wheels will eventually be rendered at their to-scale size.

For the flexible body simulation to work the vehicle must be one graphical mesh with one material so that it is not automatically split into sub-meshes (and hence also batches) by the rendering library. To visualise the movement of individual wheels and parts within a single mesh an animation armature system is used, and each part associated with a bone joint in the armature. Vertices for the part are manually assigned to the bone (see Figure 11.11) such that when the bone is moved automatically by the physics system, then the associated flexible body vertices are also moved. This process usually has a 50:1 batch reduction in our test cases and therefore increases the amount of similar entities able to be displayed on-screen by that factor.

Animation is skinned manually by forcing the bones to move to the positions and orientation offsets dictated by the physics model. In practice this means overriding the key-frame animation system of the rendering engine. The links between flexible body vertices and wheel joints for each type of vehicle are scripted. The results are presented in Figure 11.12.

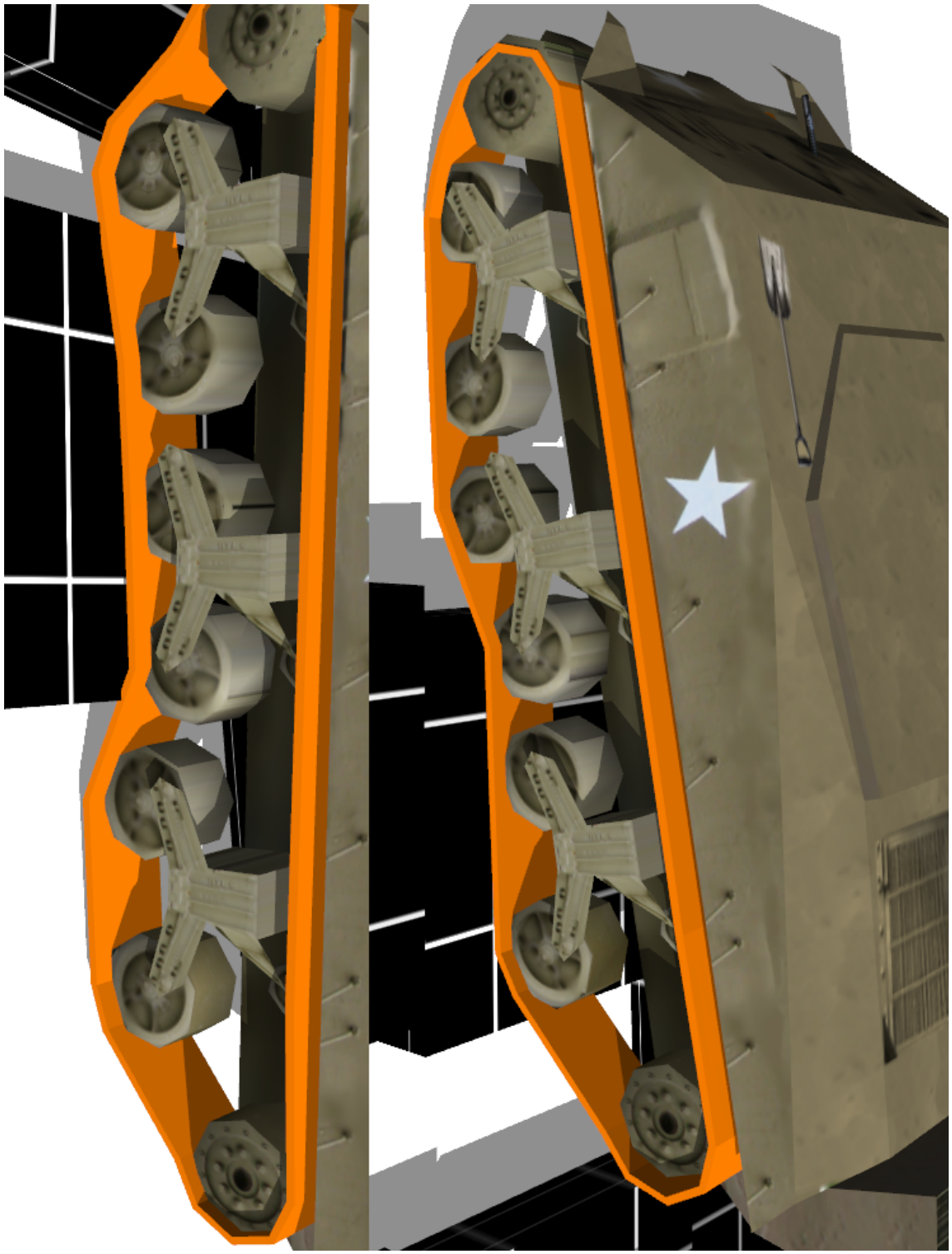

Figure 11.12: Views of a simulated 30 tonne vehicle's chassis with the animated track deformation. Although the vehicle's tracks are physically simulated as a group of spheres, the target animation approximates smaller road wheels and flexible belts.

11.7 Trailers, Articulated Vehicles, Trains & Trams

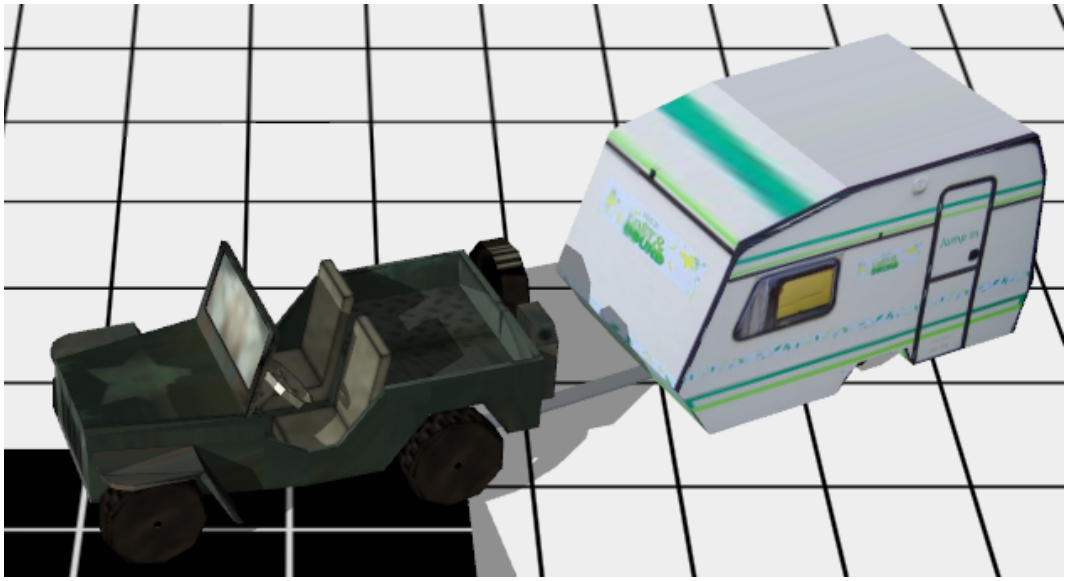

Figure 11.13: Articulation in Mechani$x$ with an unpowered trailer. The trailer is simply an unpowered vehicle limbered to the jeep with the coupling approximated by a universal joint.

The physics wrapper ($ODEWrapper$) supports creation of articulated vehicles at run time, including but not limited to towed trailers (see Figure 11.13). Trailers are treated simply vehicles that are attached to other vehicles. A whole string of vehicles can therefore be attached together i.e. train carriages.

The wrapper's function for attaching trailers is:

```
bool ODEWrapper::attachTrailerToVehicle(const int& tractorIndex,
const int& trailerIndex, const float& x, const float& y, const float& z);
```

Where $tractorIndex$ and $trailerIndex$ are the ID numbers of each vehicle, and where x, y, z are the location of the attachment point (the tow bar) as an offset from the origin of the leading vehicle.

The hinge itself is coded as a universal joint (see Figure 11.14) - this overcomes most of the roll limitations without having to build artificial springs or limits in the joint.

Some articulated vehicles need bendy/rubber/flexible attachments. You can achieve this as well by using the flexible body animation of the previous section.

11.8 Limitations

The most obvious limitation of the system is the computational cost of collision handling in real-time with a rigid-body dynamics engine. The collision handler is a recursive function, where every wheel has a contact with the ground. In practice this has limited us to about 50 simultaneously active vehicles in a graphical simulation on a modern quad-core desktop, which was able to support 200 vehicles of the same type without collision-handling.

The main limitation of the system, however, is that the underlying rigid-body library, ODE, becomes unstable when large forces or high speeds are involved. Specifically, the wheels will tend to break off their rigid joints, visibly buckling, when cornering or braking at full force whilst turning,

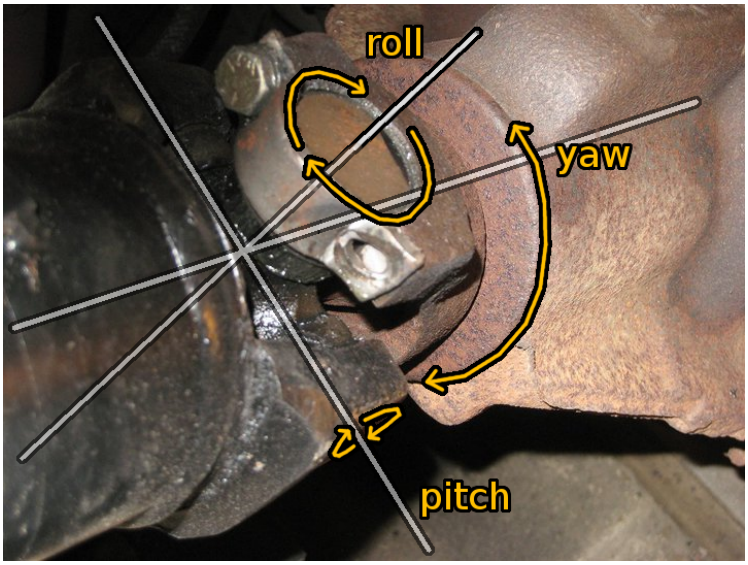

Figure 11.14: This photo of a universal joint has its axiis of rotation overlaid. We can see that it is possible to rotate almost freely around any axis, whilst still maintaining an attachment. It was more robust to simulate this kind of joint for articulation that attempting to simulate a ball-and-socket joint with limited degrees of freedom of rotation. Original, un-annotated photograph courtesy of Michael Gormack.

or turning sharply at high speed. This is a problem common to most ODE-based systems; a number of work-arounds are documented on the ODE website. The system is therefore suitable for low-speed urban traffic simulation or slow, heavy vehicles, but is simply not stable enough to simulate high-speed racing or car-chase type games.

There are a number of alternative solutions for increasing the performance or stability of the system; the choice of which would depend on the intended application and available resources;

- Increase the number of physics integrations per simulation time step.
- Replacing spherical wheel bodies and joints with ray-cast car-ground contacts.
- Reducing forces on wheels and joints by applying them to the car body instead.
- Replace ODE back-end with vehicle-specific library.
- Parallel physics computation on a GPU.

A small increase to car performance at high speed can be made by doubling the number of physics integrations per simulation step, but this also roughly doubles the computational cost per step, and thus reduces the number of cars that can be simulated in real-time by half. This is perhaps the easiest method for increasing stability if there is CPU to spare, but it will only increase the stable driving speed, not completely solve the buckling problem.

A commonly used method used to eliminate the wheel-buckling problem, and to decrease computational cost is to remove the rigid-body wheels altogether. The wheel-ground contacts are then manually created by casting a ray from the car axle positions to the ground. The wheels can of course still be graphically represented, but the rotations will have to be calculated manually. This approach will also eliminate the tendency of the car to roll due to the spherical wheels, which means that the anti-roll bar torsion forces can be reduced to realistic levels as well. The drawbacks to this approach is that the wheels will only make one contact with the environment, which would be acceptable for a flat race-track, but complex all-terrain climbing behaviour would not be possible.

Another solution might be to retain the 3D wheel contacts, but reduce the overall torque loads on the wheel joints by applying the larger resistance forces (especially brake and anti-sway forces) directly to the body of the vehicle, rather than to the wheel joints. Of course, ODE could also be replaced with a physics integrator optimised for vehicles.

The PhysX library provides the same rigid-body dynamics as ODE, but is able to utilise the parallel architecture of Nvidia hardware to perform parallelised physics integration on the GPU. PhysX is a proprietary library that requires Nvidia graphics hardware. This is probably the best option for improvement to performance, and to accuracy, if the hardware is available.

A number of blanket assumptions and approximations have been made to either reduce computational cost of design complexity. Further models could be added to the system to increase mechanical realism. Some of the simpler improvements that could be made:

- Linked suspensions (e.g. independent front and linked rear).
- A tyre shape model; tyre shape is currently represented by fixed-shape rigid spheres with a "softness" allowance.
- Clutching; there is currently no clutch in the model - gear transitions are seamless which produces unusual audio transitions.
- Non-linear torque curves; a curve could be fitted to the engine torque model, rather than using linear interpolation. This might improve engine-based audio reproduction.

In terms of rendering, the major modern limitation is no longer the count of triangles or polygons in the scene (modern GPUs can render millions of triangles in a scene in one pass), but the count of rendering *batches*, or passes; which are limited by the speed of the CPU. An in-depth explanation by Matthias Wloka of the relationship between triangle-count and batches can be found on the Nvidia website [121], where we see that a 1.5GHz CPU with a minimum of frame rate of 50 graphical frames per second (FPS) can process up to a maximum of 700 batches per frame. Every one of the Ogre3D materials applied to a sub-mesh requires its own draw call; one batch. This means that every car mesh in the camera frustum demands 7-10 draw calls; even if the car is one mesh it is split into separately rendered sub-meshes based on it's materials (the body, the windows, the interior, etc.). With an average of 10 vehicles in a traffic scene this is 100 batches per scene required by vehicles alone. Horizontal camera shots, where more vehicles are in the scene but obscured by building meshes, are typically not optimised, and an additional 100 redundant draw operations may be added. If additional resources are to be rendered, we can see that the vehicle rendering at this quality can quickly overload the CPU; decreasing the frame-rate. Some possible optimisations to batch count include:

- Use a level of detail (LOD) model to decrease complexity of vehicle further from camera focus.
- Combine parts with similar shaders (i.e. wheels) into one mesh, and use armatures to rotate and translate the wheels.
- Use an occlusion algorithm for large geometry (i.e. buildings) to remove hidden vehicles from drawing.
- Reduce number of vehicles in scene.

Chapter 12

Gremlin: A System for Benchmarking Mechanical Motion

12.1 Introduction

This chapter describes a method for evaluating machine-learning control systems for vehicles. Whilst motion control evaluation frameworks exist, there is not yet a standard test case for evaluating motion control of mechanically simulated vehicles within a physically simulated environment, where a large number of behavioural constraints exist. This chapter provides specifications for one such framework, including environment design, vehicle specifications, and evaluation criteria. The framework is used to evaluate an existing control system with results discussed. The framework has potential for comparison of control algorithms or for machine-learning in an objective self-evaluation rôle.[1].

Making an objective evaluation of steering behaviour is a complex task, however, evaluation is invaluable to us as it can be used to objectively measure:

- performance comparison at a given task of a new algorithm versus an existing one
- objective self-evaluation (fitness function) of a machine-learning algorithm

Comparative or objective analysis of steering and motion control algorithms remains a difficult problem. It is hard to construct a level playing field because the designers of motion control systems typically design the problem environment as well as the solution to it, so that any comparative performance analysis to existing algorithms, although perhaps based on objective observation, is highly biased in favour of the new algorithm. One framework has been proposed to tackle this task; SteerBench [61, 62]. SteerBench presents a large test set of typical motion-control problems as small independent, and well defined scenarios. Using a selection of these scenarios it is possible to compare the strengths of algorithms over a wide range of problems, which provides a much more level playing field. A good evaluation of a new algorithm would ideally use both a problem-specific evaluation (fitness for task) as well as a SteerBench type evaluation to give an overview of general application.

What existing steering evaluation frameworks do not yet do is provide a set of scenarios for evaluating control of vehicles with mechanical systems and constraints in environments with rigid body physics systems and constraints. The primary objective of this chapter is to extend the set of test scenarios offered by existing frameworks to the mechanical-physical vehicle domain by defining a test vehicle and a physically simulated environment. The secondary objective is to provide a

[1] Videos of selected simulation runs from the experiment in this chapter are available at `http://antongerdelan.net/videos.html` under the heading *Selected runs from "Gremlin" mechanical motion benchmarking experiments*

test environment or scenario that lends itself to self-adapting controller optimisation in an objective (not dynamic) learning environment where the "adaptive" potential of genetic-fuzzy systems can be clearly demonstrated and quantified.

12.2 Definition of Test-Course Environment

To determine if the benchmarking system can provide a robust fitness measure two mechanically simulated vehicles will be evaluated, each with quite distinct motion properties. The "Mechani$x$" system, as introduced in Chapter 11 is used to simulate each vehicle with real, known vehicle specifications. A large, heavy, 2-level Enviro 400 bus as defined in Table B.3 of Appendix B. The bus has slow acceleration, and a long body is used for the first test vehicle. The bus has very high momentum when driving at speed, so has the potential to knock over smaller obstacles that are in its path. The second test vehicle is the small, fast, light, Willys MB jeep as defined in Table B.3 of Appendix B. The jeep has a high centre of gravity that puts the vehicle at risk of flipping over when cornering at speed. The vehicles then have quite different motion control properties, and using the same controller for both should produce quite different paths of motion. A well balanced evaluation framework should be able to distinguish clearly between the motion of each vehicle. If this is the case then the benchmarking framework should lend itself to self-adapting controllers that can adaptively learn (optimise controller rules) to drive different vehicles.

The current evaluation scenarios that makes up the SteerBench test set are most relevant to pedestrian or crowd simulation; with several models of corridor problems and multiple agent scenarios. Rigid body dynamics properties of the world and kinematic/mechanical limitations (of agents) are not given special consideration. The scenarios are also static which is not suitable for evaluation of stochastic or dynamic motion controllers, and means that the test sets do not generalise well to target environments, which tend to be stochastic. Thus, if we aim to improve a motion controller based on its performance in selected test cases (perhaps automatically) then we run into the over-training problem where the algorithm learns the specific quirks of the problem and does not generalise well to slight variations of the problem. Other works have solved this issue by introducing "jitter"; random variations into the test scenario [38]. SteerBench is currently the most established evaluation framework for motion control, and therefore provides a solid base for comparative analysis. So, with this in mind the best approach is to define a scenario that extends the set offered by SteerBench, but incorporates a rigid-body physics and mechanical system, as well as a level of randomisation to increase the robust nature of training and introduce a level of uncertainty into evaluations.

A fairly generic scenario that represents the basic problem domain of obstacle avoidance-target seeking controllers as used throughout this thesis, but also introduces a reasonable level of problem complexity is the "forest of obstacles" scenario as used in Craig Reynolds' "OpenSteer" library of motion-control software. It seems reasonable then to recreate the "forest" problem space, with the same sort of randomised distribution of obstacles, but with rigid-body physics underpinnings, and use this to extend the standard benchmarking scenarios as offered by SteerBench. In this way the problem space is very similar to those found in well-known libraries.

Given a rigid body dynamics environment we have the option of making the "tree" obstacles in the scenario able to be knocked over, that is, not fully static obstacles. This gives us not only a more stochastic environment to navigate, but also further polarises the optimised machine-learning rules for different vehicles. The obstacles are encoded as $2 \times 2 \times 10$m poles, and weigh one tonne. This means that it is advantageous for a large vehicle, such as the bus, to knock the obstacles out of the way when travelling at a significant speed, rather than to drive fully around them. The jeep on the other hand will not be able to push over the obstacles, thus any learning would have to diverge rules to optimise each controller.

The test course or scenario itself is made up of a lattice of 210 of these 1-tonne posts, which are initially evenly distributed over a 200×100m rectangular area; one post every 10×10m intersection. The position of each post is randomly varied by up to 4 meters in each direction on the horizontal

plane, such that the posts can never overlap, but clusters and gaps in the field can form. The length of the obstacle field is 100m, but it is doubly wide so that the controllers can not learn to avoid all of the obstacles and circumnavigate the course; an unexpected behaviour that was emerging in the experiments of Chapter 10. Boundary walls could also have been used, which would constrain vehicles to the course, but this would not guarantee that controllers would not over-develop avoidance behaviour. The starting position is exactly half way across the course, and 10m behind the first obstacle. This ensures that longer vehicles start clear of any obstructions. The finishing point is positioned on the opposite side of the obstacle field, exactly 120m meters from the starting location. This means that, were there no obstacles, the vehicle would drive in a straight line, 120m to the destination. The stating orientation of the vehicle is not randomised, because this introduced too much variation into the starting conditions of the experiment in Chapter 9, relative to the length of the course (turning around took a significant amount of time and affected the fitness evaluation). The 3D-rendered scenario is depicted in Figures 12.1 and 12.2.

12.3 Evaluation Method

Genetic-fuzzy systems in previous chapters have used fitness functions of the type expressed in Equation 12.1 to analyse the performance of vehicle motion.

$$fitness_x = \frac{\sum_{i=0}^{n}(w_c\frac{\bar{c_i}}{d_i} + w_e(1 - \frac{t_i}{d_i v_{max}^{-1}})}{n} \tag{12.1}$$

The fitness function (Equation 12.1) gives us a fitness score for an individual x. An agent is evaluated for its performance on each path segment i along its route of navigation. These evaluations are accumulated until a distance comprising a complete run has been covered. The mean is then taken with respect to the total number of segments completed n. The core of the function comprises a collision component and an expediency component that are added together. The collision and expediency components have weighting factors, w_c and w_e respectively. These weights emphasise the relative importance of each component, and are therefore arbitrarily set by the designer. $\bar{c_i}$ represents the individual's mean mesh intersection in meters over a path segment. Intersection depth is recorded every frame of calculation and accumulated. The mean intersection is divided by the distance d_i of the segment. Expediency is calculated by taking the time to complete a segment t_i over the ideal time to complete the segment, where v_{max} is the ideal velocity of the agent. This is subtracted from 1 so that the range of expediency values increases from 0 (ideal).

The problem with this sort of evaluation method is that it is hard to know how to weight each component to justify "good" motion according to each rating. It may well be essential to use a method such as that in the equation when evaluating an individual dynamically, because we want to take as many environment features as are present into account and make a comprehensive fitness evaluation; there is no guarantee that a path segment will contain the obstacles at all, so it is convenient to use as much "rating" criteria as is available to *penalise* the motion evaluation for an individual.

However, the objective, repeated "forest" test scenario allows the luxury of using a much more simple evaluation criteria. In this chapter the fitness, or quantified score given to a motion controller moving through the forest environment can simply be $fitness_x = t$, or the time, in seconds, for an agent to complete a run of the course. The reason that we are able to exclude collisions from the evaluation is because a physics model is being used and will punish collisions realistically. To diminish this reward it is simply a case of increasing the mass of the obstacles. But what happens if a vehicle becomes stuck and can not complete the course? An infinite time evaluation is not useful. It is convenient to establish a set of conditions that detect if a vehicle has failed to complete the course, and award a penalty evaluation. But what if all of the motion controllers fail to complete the course; it is still desirable to rank them all comparatively (especially in the case of machine-learning

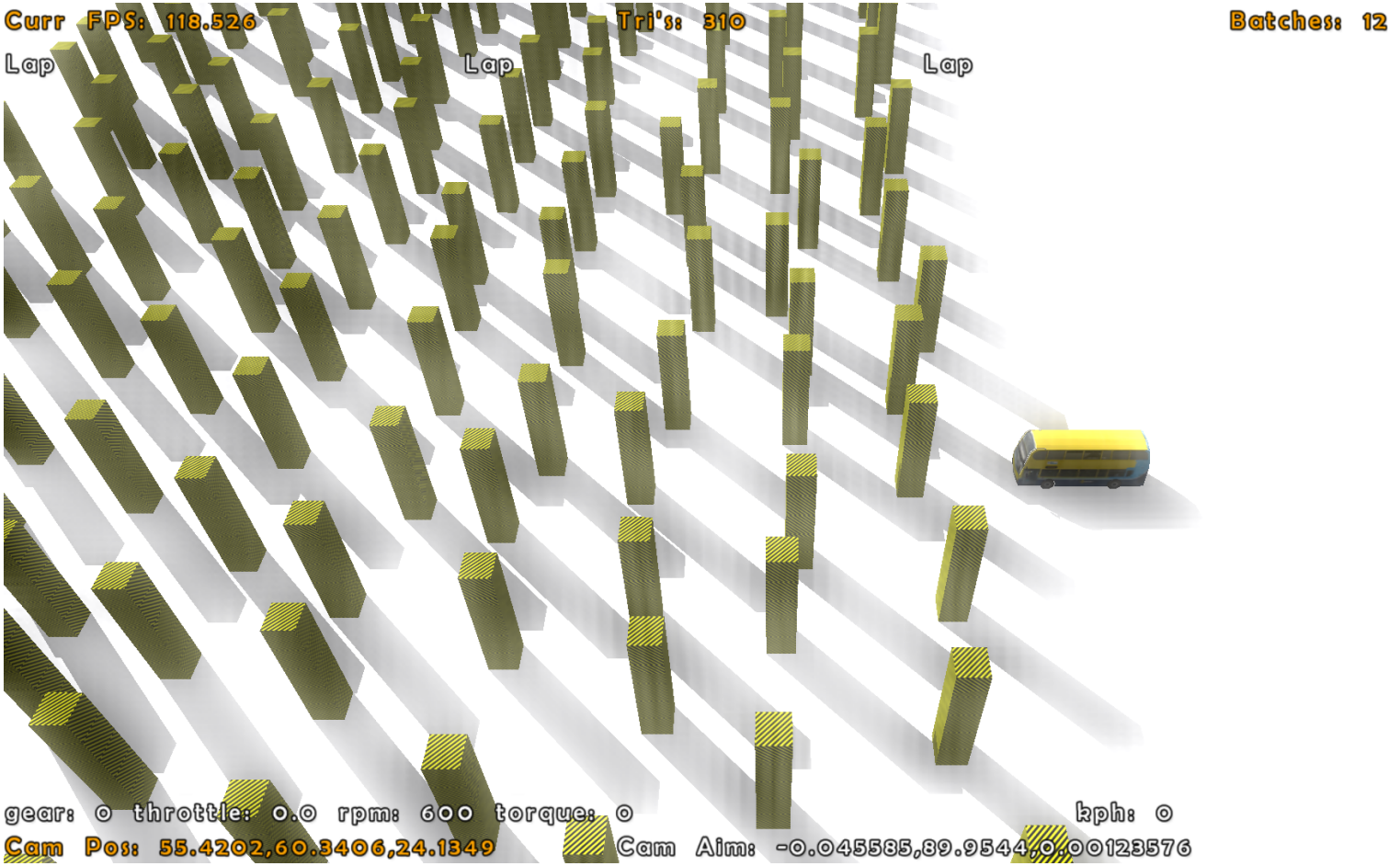

Figure 12.1: This obstacle course was designed to bear a resemblance to the "forest" obstacle course from the OpenSteer library. The vehicle, pictured at its starting position, must move 120m autonomously to the other side of a lattice field of randomly scattered cuboid "tree" obstacles. Each obstacle measures $2 \times 2 \times 10$m and weighs 1000kg. Physically simulating the obstacles means that heavier vehicles can learn to push through if they are slowed less by this than by moving around.

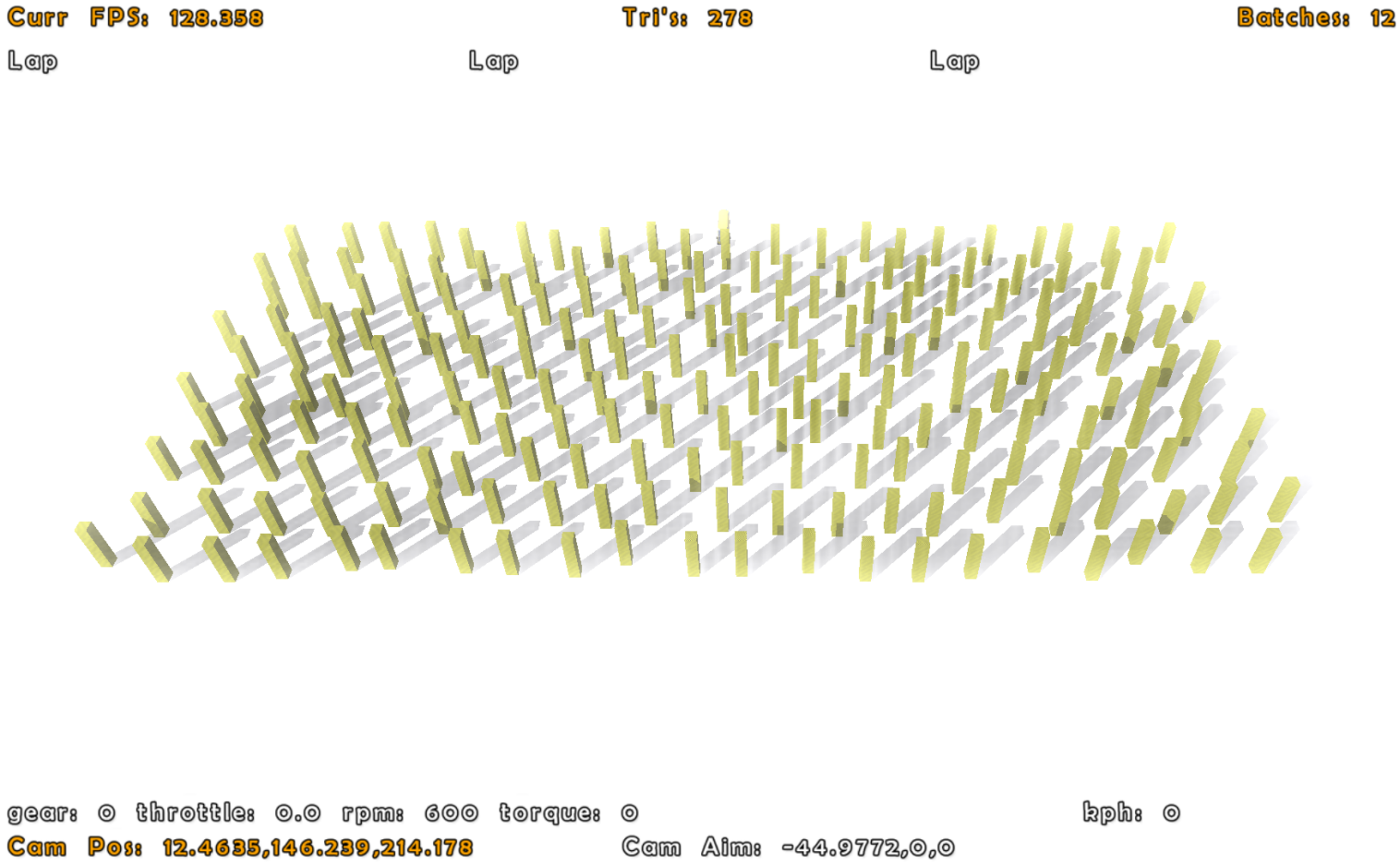

Figure 12.2: A wide-angle view of the "forest" scenario. At the beginning of a run, each "tree" is positioned on a 10×10m grid, and randomly moved ± 4m on the ground plane. This randomness introduces robustness into evaluations, and forces vehicles to cope with clusters and gaps. The cuboids extend 100 meters forward and 200 meters laterally to prevent the vehicles from learning to circumvent the entire course (an issue encountered in the previous chapters).

scenarios)? The physical nature of the "forest" test environment means that collisions are handled by the environment; either by stopping the vehicle or by obstacles being rammed over. We then do not need to include a collision cost in the fitness function and can represent it as simply:

$$fitness_x = t - d \tag{12.2}$$

Where the fitness score awarded to an individual, x, is the time taken to complete the course t, in seconds, or a penalty score of 60 for uncompleted runs, and the distance remaining to the destination d, in meters. For completed runs the distance will be 0.0 meters, but the uncompleted runs will have larger values for d (up to 120m). In this way the uncompleted runs are awarded larger, and less "fit" scores, but they can still be ranked comparatively based on the distance achieved towards the goal position.

A series of catch-all conditions are needed to determine if a run did or could not complete. Detecting these states early will reduce the overall time needed to make a benchmarking evaluation. The conditions are given in Algorithm 2.

Algorithm 2 End run condition detection in the "forest" scenario.

if Vehicle body roll exceeds 1.775 radians. **then**
  Vehicle has rolled over. End run. Penalty time 60 of seconds.
else if Vehicle has not moved more than 0.5 meters in last 12 seconds **then**
  Vehicle is stuck. End run. Penalty time 60 of seconds
else if Time for run exceeds 60 seconds **then**
  Vehicle is lost. Stop run.
else if Distance to destination is less than 6 meters. **then**
  Vehicle has reached destination zone. Run completed.
end if

This completes a definition of the test scenario. For a computer scientist familiar with this sort of simulation construction it should be possible to recreate the course and conditions, using the specifications given throughout Appendix B, and a mechanical simulation similar to that described in Chapter 11. The remainder of this chapter will describe a motion controller that is going to be evaluated, and a discussion of the results obtained from this evaluation.

12.4 Control System Design

A fuzzy motion controller will be analysed using the test scenario. A more sophisticated type of fuzzy motion control architecture has been developed based on experiences with earlier chapters in this thesis. The main innovation is the addition of switches to prevent controllers from having conflicting rules. A large amount of training time was wasted identifying and removing controllers that had been tested with conflicting rule bases in the experiments of Chapter 10, which can be a recurring problem for controllers that are optimised with a genetic algorithm as conflicting rules can be randomly generated as the result of mutation. Figure 12.3 gives us a three-stage control system for basic target-seeking behaviour. There are four fuzzy controllers in this motion control system. There are two controllers for braking behaviour. One takes the distance to destination "x" and the current speed of the vehicle as inputs. The controller is for stopping the vehicle as it approaches the destination. A second braking controller takes the current speed of the vehicle and the angle to the destination from the vehicle's current heading as inputs. The purpose of this braking controller is to reduce the speed of the vehicle if it needs to turn sharply. An algorithmic switch "S2" performs a simple logical operation and outputs only the greater of the two outputs. A throttle controller considers distance and angle to destination, as with controllers in previous chapters. Finally, a steering controller considers the angle to the destination, and the current speed of the vehicle to make steering adjustments. This gives the controller an ability to steer sharply, but not at speeds where

the vehicle is prone to flipping over. To prevent simultaneous braking and throttling behaviour from developing, an algorithmic switch "S1" allows only one behaviour to dominate, the other control output remains zero. The switch runs on a small set of logical instructions. In this case braking behaviour over a threshold factor of 0.05 (5% of maximum) will disable the throttle completely.

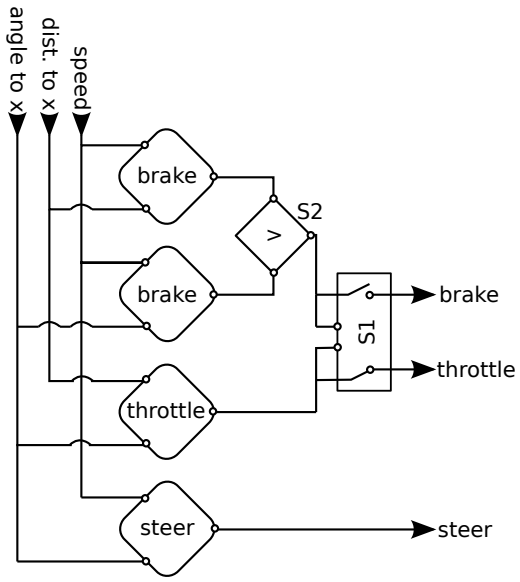

Figure 12.3: Control system used for destination-seeking behaviour. Inputs are angle and distance to destination x, and current vehicle speed. Outputs are a factor of brake, throttle, and steering controls. One braking controller is used for stopping at the destination, and the other slows down the vehicle when it needs to make a sharp turn. Switch $S2$ chooses the larger of the two braking outputs. Switch $S1$ prevents simultaneous braking and throttling. The steering controller takes the speed of the vehicle into account to prevent turning so sharply that the vehicle flips over.

The controller that will be used in the benchmarking experiment is one level of complexity higher; it has a collision-avoidance module added to it, and a more complicated algorithmic switch for switching between the outputs of both systems. The switching or blending of collision avoidance and destination seeking controllers was not effectively solved by earlier motion controllers in this thesis; either outputs could conflict (cancelling out steering controls for example) in blending systems, or sharp flickering back and forth between systems would occur as obstacles moved in and out of threshold angles which resulted in oscillating steering behaviour. Figure 12.4 illustrates the most sophisticated controller used in this thesis. The nearest obstacle o is considered. An estimated time to collision (distance to obstacle / speed of vehicle) input is used, which allows the use of only one layer of controllers rather than one layer for distance and one for speed. Switch "S3" is a more sophisticated solution to the system switching and blending problems of earlier controllers. The algorithm used is given in 3, and effectively enables blending of outputs where appropriate, as well as hard switching between outputs where necessary.

12.5 Experiments and Results

The experiment for this chapter was designed to investigate whether a stable fitness evaluation could be made with the parameters and evaluation criteria used, find out how many simulation runs it would take in this scenario before a reliable assessment or benchmark could be arrived at, and determine if there was enough sensitivity in the calibration of the benchmarking system to differentiate between the performance of different vehicles using the same controller. Gathering this information from

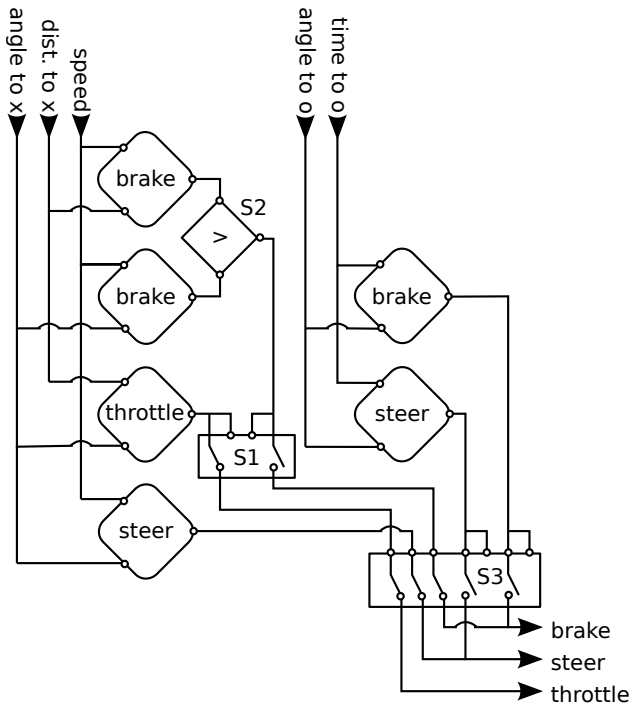

Figure 12.4: A mixed function motion control system with algorithmic switches. A reactive collision avoidance system has been added with two controllers that consider estimated time (distance / speed) to and angle to nearest obstacle o. Switch $S3$ flips between seeking and avoidance outputs and activates with any significant collision avoidance output.

Algorithm 3 Logic for mixed motion control system output switch "S3" in Figure 12.4. Both hard switching (one output on, one output off), and blending (both outputs enabled) are used.

if Avoidance braking crisp output > 0.05 **then**
  Set throttle output to 0.0, Set braking output = obstacle avoidance brake controller crisp output
end if
if Avoidance steering crisp output > 0.10 **then**
  Set steering output to avoidance steering crisp output. Ignore other steering controls.
end if

an experiment gives us a series of benchmarks which we can either use to base future evaluations of motion controllers on, or that we can use as parameters to a machine-learning system such as a genetic-fuzzy system, to allow it to effectively optimise its rule base for this test scenario.

The experiment was run and repeated indefinitely with both test vehicles. Each run would reset the test environment, using a new randomised environment configuration, and record the fitness assessment in an accumulating array of results. At the end of every run the total of current results was summed and the mean result found. The standard deviation and standard error were extracted for the data collected at each run, and then plotted on the graph in Figure 12.5, which displays results up to the 600th run. The idea is that we can watch the averaged evaluation stabilise, and the error margins decrease as more run results are collected. We can see that by the 150th run we have quite a reliable result, in that the results for both vehicles have separated into distinct streams of data, and the uncertainty calculations no longer cross over. If we use the data from the 150th to the 600th run (the earlier runs are shaded over in the figure), then we can fit straight lines to the plots (blue and red horizontal lines); indicating a fairly stable result. The error threshold does not decrease beyond about ± 5 fitness points, as there is a level of randomness in the simulation, which we can say is the maximum accuracy or granularity of the fitness evaluation.

It can be concluded that the benchmarking system presented in this chapter is a good evaluation scenario for motion controllers in a mechanical system, which is a new contribution to the range of benchmarking tools in the literature. A minimum number of runs required for a robust evaluation can be quantified at 150 runs. It can also be said that this benchmarking system can differentiate between motion control results, but has a margin of error at approximately ± 5 fitness points. This system would therefore also make a good candidate for machine learning systems to optimise motion control rules.

12.6 Future Works

Future works will expand the range of scenarios available for mechanically simulated tests. Some key examples scenarios to consider are the car pendulum-balance problem (a typical fuzzy logic demonstration problem), and the car-and-trailer reverse into a park problem. It is also an intention to train a genetic-fuzzy system with this simulation and to develop a metric for quantifying the "adaptability" of the system, for example, "How well can the system adapt when asked to drive with a new vehicle?".

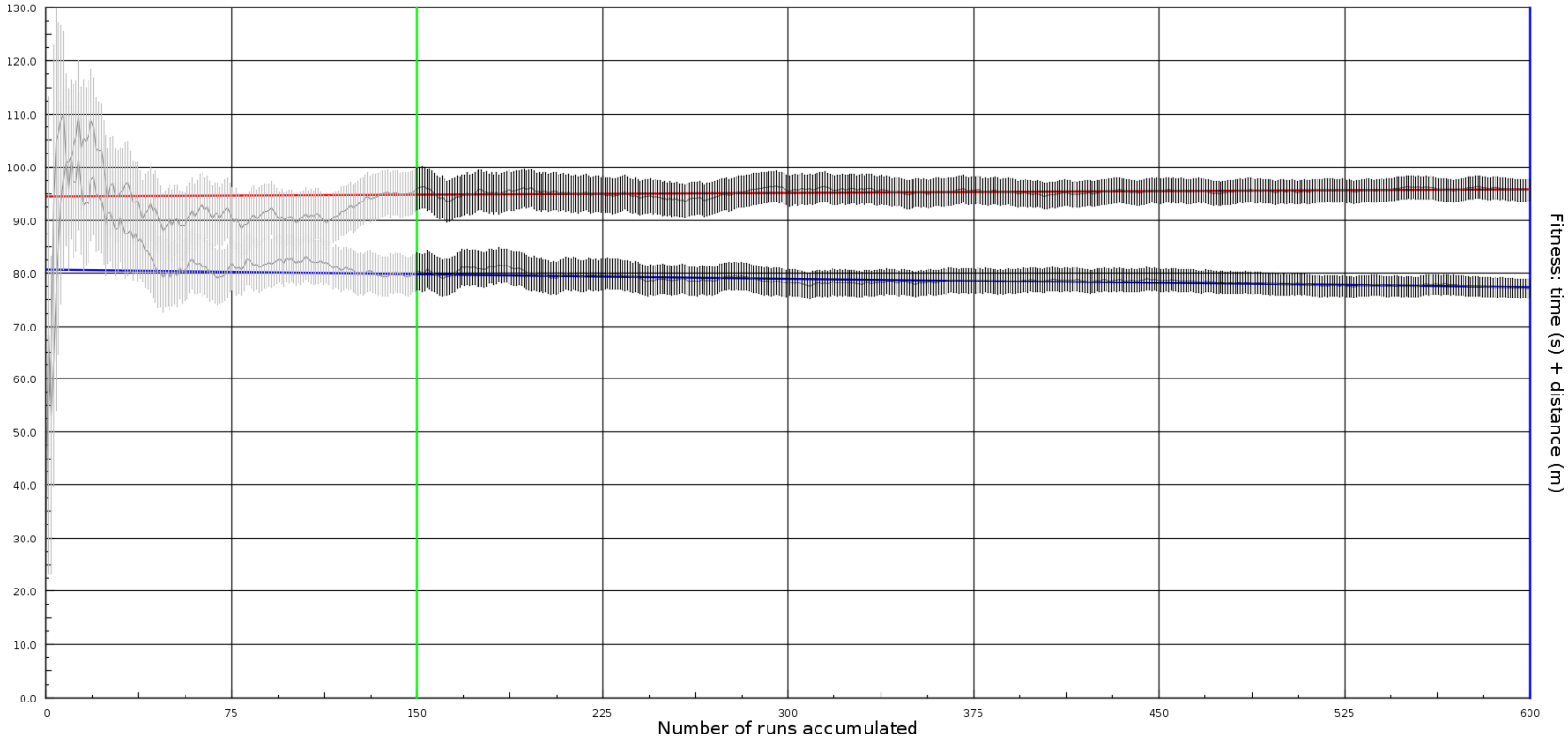

Figure 12.5: This graph gives us accumulated fitness scores and uncertainty measurements for two different vehicles being driven by the fuzzy motion controller from Figure 12.4. The first thing that the graph shows us is that the fitness measurement for both vehicles stabilises after about 120-150 runs have been collected. We can fit straight lines to data collected after the 150 run threshold (illustrated by the horizontal blue and red fits) so we can say for the simulation used that 150 runs is a safe threshold to stop evaluating with a reliable estimate of fitness. We can also see a clear distinction between the result from both vehicles (the error bars do not overlap), which indicates that the fitness function makes a course enough evaluation to distinguish between different motion control results. The simulation appears to have a fixed spread of results of ± 5 fitness points due to the robust (randomised) starting conditions.

Chapter 13

Conclusions and Discussion

13.1 Review of Fuzzy Motion Controllers

Fuzzy motion controllers are an elegant solution to motion control problems because they can be applied to almost any type of motion control. During the work on this thesis fuzzy controllers have been used to control the motion of simulated road traffic, tracked vehicles, pedestrian (and zombie) crowds, herds of cows and horses, boats, amphibious trucks (in and out of water, as well as transitional motion), heavy 2-level buses, and the original Willys MB jeeps that were prone to flipping over in reality, and still are in simulation. In some of these cases the motion controllers didn't work as well, and weren't an advantage over conditional logic (boats). But, in those cases where constraints exist to motion such as pedestrians with a limited range of animations, or vehicles with mechanical constraints, then fuzzy controllers are an effective tool for "filling in the blanks" or smoothing between states that are hard to transition between with a conditional logic model.

Experiences with the work for this thesis have brought a number of caveats to the fore. Firstly, that as the complexity of the environment (or the vehicle) increase, then the number of controllers required also increases. For example; in some of the works in this thesis the vehicles or pedestrians travelled at more of less constant speeds. The means that steering or turning behaviour produced circular paths of motion. Four controllers were sufficient in these cases to handle steering and acceleration for collision avoidance and destination following. With the later, fully mechanically simulated vehicles, a lot of braking, leaning over on suspension, and more rapid acceleration occurred, which meant that there were situations where steering was happening during braking and accelerating behaviours, which produce much more complex paths. One of the advantages of fuzzy controllers is that when they are not steering enough to avoid collision, then they are often robust enough to move onto the next level of steering behaviour and start to steer harder, despite inadequate rules. A problem arises, however, when we have physically simulated vehicles with a high centre of gravity, that tend to tip over if cornering at speed; a jeep for example. This added complexity of being able to tip over due to the physics involved necessitates the addition of another level of controllers that takes current speed into account when considering both braking and steering behaviours, otherwise it is simply impossible to create a rule-base that differentiates high-speed steering behaviour from low-speed steering behaviour. From a design point of view it is very hard to say exactly how many and what type of fuzzy controllers are required to fully accommodate for all of the conditions required for effective decision-making. Of course the motor controllers that are involved, the more complex the optimisation task as the number of rules increases.

13.2 Conclusions on Genetic Hybrid Systems

Navigation and motion control systems are notoriously difficult to calibrate, and introduce complex constraint-based problems that may shift during run-time (are stochastic). Genetic algorithms are an attractive tool for helping to solve these complex constraint-based problems automatically. A major conclusion of the work in thesis is that fuzzy control system architectures need to be specifically designed as being part of a genetic-fuzzy system. A genetic algorithm is a not a universal solution to the problem of optimising fuzzy controller rules. The arrangement or architecture of controllers needs to be carefully made so that it lends itself to optimisation by a guided search algorithm such as a genetic algorithm. Specifically, this means that designers of control systems need to include a number of special catches and conditions that prevent (perhaps unexpected) rule combinations from emerging that confuse the outputs of controllers such that they interfere with one another. In particular, if a genetic process is set to improve a controller's rule-base dynamically. A second outcome of this thesis was the evaluation and development of a range of fuzzy motion controller "hybrids". The designer needs to take special care in the arrangement of controllers such that there are no situations where the controllers can conflict or cancel each other out. The consequence of not designing fuzzy controllers specifically for tuning by genetic algorithms is that the training process will be enormously inefficient. The genetic algorithm has a degree of randomness, so there will always be rules generated that are undesirable mutations. The key is to make sure that it is not possible to breed rule-bases that are incapable of effective operation. A work-around built into research in this thesis was to give a number of "fail" detectors to each agent, after which training could be terminated prematurely, removing some excessively long training times:

- A detection mechanism to tell if the vehicle has rolled over and can no longer move effectively.
- A "stuck" timer. This is reset after the vehicle has moved a small distance (such as 0.5m), but has a very small threshold of 30s or less. This is a catch-all that penalises individuals that have conflicting motion rules, and also vehicles that have become wedged between obstacles.
- A total time-out. If a run is not completed in under a threshold time, somewhat longer than the stuck timer, then the training is terminated prematurely. This catches vehicles with rule-bases that are "too scared" of hitting obstacles, and continually circle, and vehicles that are driving far too slowly.

Another key design feature is the inclusion of "programmatic switches" in genetic-fuzzy controller architectures. This gives the designer the ability to imbue a set of logical operators into a switch that detects and prevents confused or badly mixed outputs from multiple controllers. These switches usually need to be no more complicated than deactivating some controllers when one controller's output reaches some threshold level.

A major consideration to make is the time it will take to train a set of controllers using a genetic-fuzzy system. The more complex a simulation, the more fuzzy controllers are needed for sensible motion (more conditions are taken into account), which mean more rules in the rule-base and hence longer chromosomes, and more runs required to evaluate an individual's fitness. Adding more complexity to simulations; mechanical simulations, more obstacles, dynamic obstacles, etc. also increases the number of runs required to assess an individual's fitness. So we can see that as environments become more complex, the time requirements of training increase exponentially. Therefore, a shrewd designer that wants to take advantage of a genetic-fuzzy system as a dynamic on-the-fly tuning process will design the entire simulation, as well as the controller architecture to deliberately take advantage of the genetic-algorithm tuning process. Introducing multiple agents training within one simulation can divide training time costs, but this is only going to suit some applications.

It can be concluded that real-time dynamic on-the-fly training, the special case where agents learn as they are used in real time, is really only feasible for very short chromosomes, that is genetic-fuzzy systems with no more than 2 or 3 controllers using a 3×3 FAMM. However, if large number

of these agents are able to be deployed at one time and trained in parallel (a game environment would be an ideal example of this) then we can afford to increase the complexity of the controllers or environment.

Genetic-fuzzy systems can be quite effective at relieving the time requirements of hand tuning a large number of fuzzy rules in a static training situation, and optimising fuzzy rule-bases over hand-optimised sets by using fixed runs through obstacle courses, but the effectiveness of training depends on a number of different training tools being calibrated correctly. Firstly, the environment itself needs to contain the same type of environment as expected in final application. The environment elements (obstacles, etc.) must be regularly distributed over the test area to ensure a more or less uniform training environment. The length of course runs needs to be long enough to collect enough fitness data to differentiate "good" runs from "bad" runs. This will only be clear after studying graphs of results from a control study. The environment needs to include some sort of randomised element to prevent overly-brittle training and the learning of quirks in the environment, as discussed in Chapter 2.

Summarising the work on genetic-fuzzy systems in this thesis, we can say that it introduces the following original contributions:

1. A GFS and architecture specific to vehicle motion control.
2. An adaptive, real-time variation of the above.
3. A generalised genetic architecture that can be linked to any motion controller.
4. Evaluations of all of the above.

Introducing genetic systems may help to solve more difficult stochastic calibration problems, however, a number of new constraint-based problems are also introduced. The genetic system itself has a large number of parameters which need to be optimised to ensure efficiency of the learning/optimisation process, particularly if it is to be run with real-time constraints. An exhaustive evaluation study of these parameters is provided in this work for one environment, with the optimal configuration presented. There is no guarantee, however, that this configuration will apply equally well to other environments, and as such only serves as a starting point for environment-specific studies. Future implementations of hybrid genetic systems should expect to conduct a similar meta-optimisation process (optimising the optimiser) before normal operation can proceed.

Considering the amount of work involved in manual tuning compared to setting up the training process, we can conclude that using a genetic hybrid it is only a clear advantage to fuzzy motion control systems that have to solve a difficult stochastic problem with a large number of constraints. There are two types of application where this is the case; either where the rules need to change in real time, or where the standard approach of optimising by test case will not guarantee a good balance across a dynamic environment. Examples of ideal applications are:

1. crowd-type simulations where the desired overall (group) motion is hard to arrive at by individual case scenarios
2. vehicle motion in unfamiliar, mixed, or changing environments where the rules need to adapt in real-time
3. motion calibration with "black box" constraints such as physics or some animation systems

13.3 Overarching Conclusions

Although it was not a stated aim of this thesis, one of the main contributions of this work was the development of a robust 3D simulation design framework for scientific work. The features of this being robust toolkits for rapid design of simulation environments for running experiments, a

software design pattern for partitioning different simulation components and libraries in a modular, flexible configuration, a series of data visualisation and transparency tools and graphing approaches, and benchmarking systems for running repeatable motion control experiments with detailed discussion of how performance data should be collected from these experiments with measurements of uncertainty. Perhaps most important in this particular outcome is the vehicle mechanical simulation framework of Chapter 11, which has not been published elsewhere in academic literature, and the benchmarking system of Chapter 12, which develops this into a well-defined scenario for making objective measurements of motion control performance, as mechanical and physical constraints have not been considered in benchmarking systems in the literature.

A second outcome of this thesis was the evaluation and development of a range of fuzzy motion controller "hybrids". The most interesting result of these works was the discovery of how difficult it is to put the evaluation of simulation-based motion controllers into a objective (scientific) evaluation framework. This is due to the inherent difficulties of comparing motion controllers that are custom-designed for solving motion control problems of a specific simulation. These are "apples-to-pears" comparisons. It is a nonsense to try to compare the performance of a mechanically simulated agent with an agent that does not respect the same physical constraints. The major conclusion drawn here is that the evaluation frameworks need to evolve alongside new simulations and controllers. Both for demonstration to peers, and also to provide an effective self-evaluation tool for machine-learning algorithms such as the GFS.

The importance of effectively demonstrating the performance of motion controllers remains, and the best practice discovered in this work is to use a "mixed arms" approach, or a varied toolkit of visualisation and data collection tools; recorded videos of simulation runs, performance data from well defined scenarios and simulations, real-time graphs, and if possible, replication of well-known problems and test-environments.

13.4 Future Works

One of the future works directly stemming from this research is the development of more standardised test environments utilising physical-mechanical simulation; designing the car-and-trailer reverse problem, and the pendulum balance problem as well-defined test cases with effective evaluation criteria would be particularly interesting works for this area.

An ongoing work is the integration of a generic genetic-fuzzy system library into mechanically simulated vehicle simulation with objective benchmarking tools such as the scenario developed in Chapter 12. Given the larger number of constraints and rules to optimise for these systems it is of interest to discover if a GFS is capable of this more sophisticated level of optimisation, how long this would take in terms of training hours, and indeed how this might compare to a human driver's learning curve.

Given the range of simulations, including mechanical vehicle simulations, which fuzzy controllers have been successfully applied to it is reasonable to assume that larger numbers of fuzzy controllers will be used in real vehicles. We have already seen a commercialisation of embedded fuzzy control chips for consumer appliances such as washing machines. It is not unrealistic to expect that fuzzy controllers will be used in most light rail (tram) and that we will see much greater uptake by heavy rail applications by the end of this decade. Emerging technologies for autonomous control of heavy road traffic (lorries and truck-and-trailer units) will certainly be an area of study sponsored by industry before autonomous private vehicles are commercialised. The goal of this research is to reduce transport costs and to act as an emergency "auto pilot" to help reduce driver tiredness related collisions on long trips. Fuzzy controllers may well be investigated as a vehicle-driving technology in this area.

Simulation will continue to play a key rôle in the development of any fuzzy motion control technology, and the toolkits used for visualisation and physical simulation of real problem domains will have to evolve alongside the hardware technology.

The development of fuzzy controllers for computer animation will continue, based on the success of Massive's fuzzy logic for simulation of very large, interacting crowds and battle scenes. At some point within this decade the interactive media technology used in video games will converge with the level of technology currently used to render motion pictures. We will see true "interactive films", that is, computer games rendered on home-theatre type systems, using motion-sensing hardware and 3D projection technology to immerse players, but also featuring real actors, and using the same interactive, large-scale crowds and armies currently limited to film production. When this cross-over occurs we will have a surge of technologies used to animate and move plausible, interactive crowds, and a high demand for computational efficiency and realism.

Appendices

Appendix A

Summary of Publications

The following works were published during the course of this thesis.

A.1 Peer-Reviewed Articles

1. Cathy Ennis, Anton Gerdelan and Carol O'Sullivan, "Plausible Methods for Populating Virtual Scenes", Crowd Simulation Workshop, Computer Animation and Social Agents, June 2010, Sant-Malo, France.

2. Anton Gerdelan and Carol O'Sullivan, "A Genetic-Fuzzy System for Optimising Agent Steering", Computer Animation and Virtual Worlds, May 2010, v.21, pp.453-461. [58]

3. Sébastien Paris, Anton Gerdelan, and Carol O'Sullivan, "CA-LOD: Collision Avoidance Level of Detail for Scalable, Controllable Crowds", Motion in Games, Springer Berlin / Heidelberg, 2009, v.5884, pp.13-28. [13]

4. Anton Gerdelan and Napoleon H. Reyes, "Towards A Generalised Hybrid Path-Planning and Motion Control System with Auto-Calibration for Animated Characters in 3D Environments", Advances in Neuro-Information Processing, Springer Berlin / Heidelberg, 2009, v.5506, pp.1079-1086. [55]

5. Daniel P. Playne and Anton Gerdelan and Arno Leist and Chris J. Scogings and Ken A. Hawick, "Simulation Modelling and Visualisation: Toolkits for Building Simulated Worlds", Research Letters in the Information and Mathematical Sciences, Massey University, 2008, v.12, pp.25-50. [59]

A.2 Technical Reports

1. Anton Gerdelan, "Mechani$x$: Vehicle Mechanical Simulation", Tech. Rep., Institute of Information and Mathematical Sciences, Massey University, November 2010, CSTN-118, Albany, New Zealand. [122]

2. Anton Gerdelan, "A Brief History of AI in Entertainment", Tech. Rep., Institute of Information and Mathematical Sciences, Massey University, November 2009, CSTN-105, Albany, New Zealand. [1]

3. Anton Gerdelan, "Auto-Training Animated Character Motion: A Rule-Base Tuning Hybrid Fuzzy-Genetic Algorithm", Tech. Rep., Institute of Information and Mathematical Sciences, Massey University, June 2009, CSTN-098, Albany, New Zealand. [57]

4. Anton Gerdelan, “Architecture design for self-training intelligent vehicle-driving agents: paradigms and tools”, Tech. Rep., Institute of Information and Mathematical Sciences, Massey University, April 2009, CSTN-088, Albany, New Zealand. [56]

5. Anton Gerdelan, “Driving Intelligence: A New Architecture and Novel Hybrid Algorithm for Next-Generation Urban Traffic Simulation”, Tech. Rep., Institute of Information and Mathematical Sciences, Massey University, February 2009, CSTN-079, Albany, New Zealand. [54]

6. Anton Gerdelan, “A solution for streamlining intelligent agent-based traffic into 3D simulations and games”, Tech. Rep., Institute of Information and Mathematical Sciences, Massey University, January 2009, CSTN-072, Albany, New Zealand. [53]

7. Anton Gerdelan, “Grid-Ireland and e-Research Review for KAREN and BeSTGRID”, KAREN Community Reports, September 2008, Wellington, New Zealand. [123]

8. Ken A. Hawick and Anton Gerdelan, “Software Integration Architectures for Agents”, Tech. Rep., Institute of Information and Mathematical Sciences, May 2008, CSTN-054, Albany, New Zealand. [70]

Appendix B

Reference Cards: Simulation Models and Specifications

Phenomenon	**Model used**	**Units stored in memory**
Numerical precision	single-precision floating-point values	32-bit float
Integrator precision	20 iterations of ODE quick step	integrator iterations per time-step
Time Step T	accumulated fixed-steps of 0.01	seconds (s)
Change in time δt	seconds	seconds (s)
Mass m	Newton's Second Law [124, 125]	kilograms (kg)
Distance d	$1 \cdot ODE$ units and $1 \cdot Ogre$ units	meters (m)
Angle $\angle$	$1 \cdot ODE$ units and $1 \cdot Ogre$ units	radians (rad)
Force F	Newton's Second Law	Newtons (N)
Friction F	Coulomb friction [126, 127]	μ coefficient and force ($N \cdot m$)
Rigid body softness	ODE inter-penetration model	constraint-force mix (cfm)
Power P	$10^3 \cdot ODE$ ACR units	Watts (W)
Torque τ	$\tau = \frac{P}{\omega}$, $10^3 ODE$ units	Newton-meters ($N \cdot m$)
Frequency f	$f = \omega \cdot \frac{60}{2 \cdot \pi}$ (mechanical) and $f = \frac{1}{T}$ (acoustic)	Hertz (Hz) and rotations per minute (rpm)
Velocity v	$1 \cdot ODE$ units and Newton's Second Law	meters per second ($m \cdot s^{-1}$)
Angular Velocity ω	$1 \cdot ODE$ units	radians per second ($rad \cdot s^{-1}$)
Acceleration a	Newton's Second Law	meters per second per second ($m \cdot s^{-2}$)
Angular Acceleration α	Newton's Second Law	radians per second per second ($rad \cdot s^{-2}$)
Spring equation	ODE's approx. of Hooke's Law [120]	ODE error reduction parameter (erp) and
Gravity	-9.8 ODE y-axis gravity variable	meters per second per second ($m \cdot s^{-2}$)
Min. speed	rigid bodies disabled < 0.5	meters per second ($m \cdot s^{-1}$)
Min. angular speed	rigid bodies disabled < 0.0075	radians per second ($rad \cdot s^{-1}$)
Min. idle steps	inactive bodies disabled > 10	integrator iterations
Min. idle time	disabled this ODE parameter	seconds (s)

Table B.1: Classical mechanics simulation model details with parameter values used. Units given are the values and factor stored in one computational variable (which affects floating-point calculation errors).

Component	**Model used**	**Variables**
Vehicle velocity v	ODE's quickstep integrator	derived $m \cdot s^{-1}$
Engine power P	*Mechanix*	max. power (kW)
		and max. frequency (rpm)
		or max. torque (N)
Engine acceleration α	linear approx. model	max. $\frac{\delta P}{\delta t}$ $(\frac{W}{s})$
	or non-linear function	interpolated graph points
Transmission	*Mechanix*	forward and reverse gears
Gears	forward or reverse direction	
	gear ratio	factor of engine torque
	wheels engaged	count
Wheel torque τ	$\frac{\tau_{transmission}}{wheels(n)}$	ODE-derived (N)
Wheel angular velocity ω	transmission output	ODE-derived ($rad \cdot s^{-1}$)
Wheel steering	yaw limits	$\pm range$ (rad)
Wheel steering	and factor of steering control	$factor$
Inverted steering	as above with negative factor	$-factor$
Wheel differentials	wheels independently simulated	
Tyre friction	Coulomb-friction coefficient	μ
Tyre bounciness	ODE's bounce approx.	bounce factor
Tyre softness	ODE's rigid-body softness approx.	softness constant
Sprung suspension	all-wheel independent	spring equation (see Table B.1)
Bogie suspension	not simulated except where no springs	
Anti-sway bars	approx. to restorative force	max. force F in Newtons (N)
Articulation and trailers	universal joint between vehicles	3D anchor position
Continuous tracks	approx. by road wheels and torque factor	torque factor τ in Newtons (N)

Table B.2: Summary of mechanical simulation models used with parameters specific to vehicle model and values loaded from vehicle's script.

Component		Value	Units
Hull	geometric $\vec{shape}$	$[1.57,\ 1.37,\ 3.34]$	m
	mass distribution $\vec{m}$	$[1.57,\ 0.50,\ 3.34]$	m
	uniform mass m	1000.0	kg
	ground clearance d	0.22	m
	friction f	2.0000	μ
	softness	0.000010000	cfm
Wheel & Tyre	radius r	0.41	m
	mass m	10.000	kg
	friction f	0.50000	μ
	softness	0.10000	cfm
	$\vec{position}_0$	$[-0.68,\ -0.69,\ -1.10]$	m
	$\vec{position}_1$	$[+0.68,\ -0.69,\ -1.10]$	m
	$\vec{position}_2$	$[-0.68,\ -0.69,\ +1.35]$	m
	$\vec{position}_3$	$[+0.68,\ -0.69,\ +1.35]$	m
Suspension Springs	Type	4x4 independent	
	ODE model	1.0000	erp
	ODE model	0.00125	cfm
Anti-Sway Bars	Restorative Force F	20000	$N \cdot m$
Steering	Torque τ	2000.0	$N \cdot m$
	Wheels engaged i	2, 3	indices
	Angle limit	1.5000	rad
	Full steer time	12.000	s
Motor	Torque curve point	600.00, 119.31	rpm, $N \cdot m$
	Torque curve point	2000.0, 143.72	rpm, $N \cdot m$
	Torque curve point	4000.0, 105.75	rpm, $N \cdot m$
Transmission			
	gear_R	-3.4890, 4	torque ratio, wheels engaged
	gear_N	0.0000, 0	torque ratio, wheels engaged
	gear_1	2.5710, 4	torque ratio, wheels engaged
	gear_2	1.5510, 4	torque ratio, wheels engaged
	gear_3	1.0000, 2	torque ratio, wheels engaged
	differential	5.3800	torque ratio
	efficiency	0.70000	torque factor
Resistance Forces			
	foot brakes	1750.0	max. N
	drag	0.55000	coefficient
	drag area	0.80	m^2
	roll friction	18.000	coefficient

Table B.3: Specifications used for mechanical simulation of a Willys MB Jeep. Distances accurate to centimetre.

Component		Value	Units
Hull	geometric $\vec{shape}$	$[2.55,\ 3.6,\ 9.8]$	m
	mass distribution $\vec{m}$	$[2.55,\ 3.6,\ 9.8]$	m
	uniform mass m	15500.0	kg
	ground clearance d	0.45	m
	friction f	2.0000	μ
	softness	0.000010000	cfm
Wheel & Tyre	radius r	0.55	m
	mass m	10.000	kg
	friction f	0.50000	μ
	softness	0.10000	cfm
	$\vec{position}_0$	$[-1.1,\ -1.9,\ -2.7]$	m
	$\vec{position}_1$	$[+1.1,\ -1.9,\ -2.7]$	m
	$\vec{position}_2$	$[-1.1,\ -1.9,\ +2.5]$	m
	$\vec{position}_3$	$[+1.1,\ -1.9,\ +2.5]$	m
Suspension Springs	Type	4x4 independent	
	ODE model	5.0000	erp
	ODE model	0.00125	cfm
Anti-Sway Bars	Restorative Force F	100000.00	$N \cdot m$
Steering	Torque τ	20000.00	$N \cdot m$
	Wheels engaged i	2, 3	indices
	Angle limit	1.5000	rad
	Full steer time	30.000	s
Motor	Torque curve point	600.00, 500.00	rpm, $N \cdot m$
	Torque curve point	1300.00, 705.00	rpm, $N \cdot m$
	Torque curve point	1600.00, 895.00	rpm, $N \cdot m$
	Torque curve point	2600.00, 1000.00	rpm, $N \cdot m$
Transmission			
	gear_R	-3.4890, 2	torque ratio, wheels engaged
	gear_N	0.0000, 0	torque ratio, wheels engaged
	gear_2	1.5510, 2	torque ratio, wheels engaged
	gear_3	1.0000, 2	torque ratio, wheels engaged
	differential	5.3800	torque ratio
	efficiency	0.70000	torque factor
	automatic shift up at	2200	rpm
	automatic shift down at	850	rpm
Resistance Forces			
	foot brakes	125000.00	max. N
	drag	0.4	coefficient
	drag area	6.0	m^2
	roll friction	15.000	coefficient

Table B.4: Specifications used for mechanical simulation of a Enviro 400 bus. Distances accurate to centimetre.

Glossary

A* Algorithm An ubiquitous search algorithm that is commonly used for path-finding in computer games and robot navigation. Many variants have been created for specific applications. A* is a graph search algorithm that uses an admissible heuristic to guide itself to the goal. It is optimal for the heuristic used.

admissible heuristic A heuristic function is admissible if its output cost heuristic is no greater than (never under-estimates) the lowest-cost path to the goal.

artificial intelligence (AI) A field within computer science that studies artificially intelligent agents. Artificial intelligence attempts to create in machines decision-making behaviour that is either rational, or can create convincing human-like behaviour. AI raises philosophical issues about the nature of intelligence.

artificial life (AL) A field that studies biological processes or animal behaviour using complex computer simulations. Artificial life attempts to recreate biological phenomena based on logical rules.

algorithm A formalised sequence of instructions for describing a task or procedure. Commonly used in mathematics and computing. In computer science an algorithm is an abstract description of a procedure as a series of logical steps to be taken; this can then be implemented into a working form using a specific computer programming language.

angular velocity Speed in a circular direction. SI units are in radians per second ($rad \cdot s^{-1}$).

animat A contraction of anima-materials, animats are artificial animals. Animats can refer to robots, but more commonly animat refers to very simple agents in complex artificial life simulations, which are used, in particular, to study the nature of emergence.

artificial neural network (ANN) Often described as "the second best way to implement a solution", ANNs are a computational model that mimics the structure and functionality of a biological neural network, where the artificial equivalent of biological neurons are linked nodes. ANNs are usually used in machine-learning operations, where input information flows from one end of the network to the other. A statistical approach is usually used to add or remove neurons The resulting network is a computational model of the relationship between the inputs and the outputs. ANNs are most commonly found in complex pattern-matching problems such as automatically identifying car plate numbers from digital photographs.

application-programming interface (API) Acts as a gatekeeper between pieces of software. The API is a list of requests that the programmer is allowed to ask of the programme behind the interface.

articulated vehicle From the Latin *articulus*: small joint. A segmented vehicle joined by a pivot. Allows for sharper turning.

artificially intelligent agent A machine component that has the properties of an intelligent agent. In this thesis these agents are used to enable virtual characters and robots to operate autonomously.

Belief-Desire-Intention (BDI) BDI is a popular paradigm for design of intelligent agents, particularly those that are designed to mimic humans or fulfil a human-like rôle such as driving a vehicle. BDI agents form a perception of their environment based on information gathered from their sensors, maintain inherent "desires" or goals to accomplish, and map those goals into actions that affect the environment through an "intention" process.

bogie An individual chassis or framework carrying wheels. Trains or tracked vehicles have multiple bogies.

Caterpillar tracks *See continuous tracks*.

chassis A framework connecting only the wheels, suspension and basic automotive components of a vehicle.

classical mechanics A major field within physics that studies the physical laws which mathematically describe the motion of bodies and the affect of forces on bodies.

cloud computing An abstraction of several computer networked services into one layer with a common interface. Analogous to grid computing.

computational grid A computer network based on the operation of an electrical power grid. Grids are typically used for the computational component of international "big science" experiments such as the Large Hadron Collider (LHC) experiments at CERN. Resource sharing is performed via grid services, which are analogous to web services. Grid access is usually protected by federated trust networks.

continuous tracks Mechanical system using a loop of moving interlocked metal segments designed to spread ground pressure over a wider area than wheels and increase surface traction.

Coulomb's Law of Friction That kinetic friction is independent of sliding velocity; $F_f \leq \mu F_n$; a dry friction approximation proposed by Charles-Augustin de Coulomb.

D* Algorithm A dynamic variation of the A* Algorithm that has been used in various autonomous robot motion applications as it is able to deal with intermittent changes to an environment such as opening and closing doors.

deterministic A deterministic system is one where the future state is a product only of the current state, and it is therefore possible to determine any future state from any past or current state. A non-deterministic system is one where some unpredictable or unknowable element exists that affects the state of the system. Time step-based computer simulations are usually deterministic until a random function is introduced, as is the case in all of the simulations in this thesis, at which point they can be considered non-deterministic.

differential A device that allows opposing road wheels to rotate at different speeds.

differential gear A fixed gear that multiplies any output gear ratio from a gearbox by a set amount.

Direct3D A Windows library for visualising 2D and 3D graphics from simple primitives. Part of the DirectX toolkit.

distributed computing Distributed computer programmes are designed to be split up into parts that run simultaneously on multiple computers communicating via a network.

evolutionary algorithm (EA) A problem solving method emphasising a self-improvement system that is inspired by biology. The methods given particular study with in this work are genetic algorithms.

emergence Complex systems or patterns arising from a large number of simple interactions. In this thesis we are mainly concerned with emergent behaviours; complex group behavioural patterns that arise from interactions between individuals with only a small number of rules for interacting with one another. Examples discussed in this thesis include line-forming and dispersal behaviours in crowds or flocks and also queuing and over-taking behaviours in road traffic.

epoch Unix Time, or the Unix "epoch" is the number of seconds elapsed since midnight, 1 January 1970, Coordinated Universal Time. This event is roughly the time that the first Unix machine was started.

fuzzy associative memory matrix (FAMM) A rule table used to match fuzzy input values to fuzzy output values in the fuzzy inference process.

fitness function A function that rates the effectiveness of a chromosome in a genetic algorithm. The chromosomes from each generation can then be ranked in order, and the best chosen to reproduce for the next generation.

Fmod Ex A proprietary, multi-platform, 3D audio library made by Firelight Technologies. Plays a variety of different digital and analogue audio formats. Supports higher-level physics based audio simulation techniques such as the Doppler effect, and geometric occlusion.

frames per second (FPS) The frequency that graphical output to screen is redrawn (a "frame" is processed) by a programme. This can also be given in Hertz (Hz), but is usually given as "FPS" to avoid confusion with the monitor's hardware refresh rate. The frame rate produced is dependent on graphics hardware, computing hardware, complexity of underlying software, and the efficiency of the software pipeline. Designers of real-time visualisations generally aim to produce frame rights high enough to maintain the illusion of motion on their lowest-powered hardware target.

finite state machine (FSM) A behaviour model based on a finite number of possible states, and the transitions between those states. FSMs can be modelled in a graph similar to a flow chart. This model is used, amongst other applications, to drive automata (automatic machines), and represent a basic level of convincing behaviour in computer actors.

fuzzy logic Also known as "crisp logic", fuzzy logic is an extension of fuzzy set theory, and is a system of logical reasoning for values that are imprecise; between 0 and 1, or *true* and *false*. Classic predicate logic, or first-order logic only reason based on values that are either true or false.

fuzzy sets A fuzzy set is a class of objects with a continuum of grades of membership from 0 to 1 or completely false to completely true. Introduced by Lofti Zadeh in 1965 as an extension of mathematical sets [49]. The Fuzzy Sets article gave rise to the fields of fuzzy logic, and fuzzy control theory.

genetic algorithm (GA) A heuristic used to automatically improve a search method to solve a particular problem. Genetic algorithms are one of several evolutionary algorithms.

gear A wheel with teeth that interlock with larger or smaller gear wheels. Used in a gearbox.

gear ratio The torque to speed ratio of a gear. Larger gears transmit higher torque and lower angular velocity.

gearbox A device that allows a driver to select different gears from the transmission.

genetic-fuzzy system (GFS) Fuzzy control system hybrids that incorporate a genetic algorithm for determining optimal fuzzy system parameters. Genetic-fuzzy systems require an architecture and a fitness function specific to the problem that they solve.

genetic neural network (GNN) A hybrid algorithm that optimises the structure of a neural network by using a genetic algorithm.

graphics processing unit (GPU) A specialised microprocessor designed to compute graphics rendering operations. Modern graphics hardware cards contain processors with 32 or more cores, and gain a significant processing speed advantage because graphics processing jobs lend themselves to parallel execution. GPUs are now commonly used for processing non-graphical parallel algorithms via the CUDA programming architecture [128–130].

heuristic Heuristics are rule of thumb guides for helping to find a solution more quickly than by trying all possible alternatives indiscriminately. In path-finding a heuristic is usually expressed as a distance. A popular example is the "Manhattan distance", which judges a rough distance through a city by adding up the number of city blocks up and across.

hull Any part of the vehicle built on top of the chassis.

hybrid algorithm A new algorithm created by combining and also possibly modifying existing algorithms. Hybrids typically provide additional infrastructure or interfaces which allow the existing algorithms to communicate.

idler Smooth wheels that carry mechanical links and are not directly driven (pulleys). Supporting pulleys that do not change the direction of links are often simply referred to as "rollers" e.g. in a continuous track's top row of wheels, *the idler* is the front wheel, intermediate pulleys are rollers, and the powered, rear wheel is the sprocket.

intelligent agent Any entity that is capable of perceiving its environment and carrying out goal-directed action. In this thesis use of the term "intelligent agent" specifically implies an artificially intelligent agent.

Irrlicht A scene-graph library abstracting either Direct3D or OpenGL to produce 3D graphics. Irrlicht also add support for all of the major shader implementations.

level of detail (LOD) A series of techniques for reducing graphical or computational detail given to objects that are deemed to be less perceptually important, usually based on distance and angle from a viewpoint or camera. This decreases the computational cost of reproducing larger scenes. Usually this means reducing the number of polygons used to render an object's 3D mesh, but can be applied to physics, audio, or behavioural computation.

microsimulation Within the field of road traffic simulation, microsimulation models are those where individual vehicles are simulated moving through a defined road network. The advantage of simulating individual vehicles, rather than simply overall flow patterns, is that traffic queuing behaviour (at intersections) can be simulated. Commercial microsimulation models are typically coupled with a 3D graphics to visually demonstrate the effect of road infrastructure changes on traffic conditions.

middleware A software layer that sits between top-level applications and the operating system in distributed computing systems. Middleware encapsulates some core functionality (such as mail sorting), whilst only loosely coupling it to both the target platform and the high-level application. The middleware paradigm is also used to wrap common features of intelligent agent systems so that they can be shared between applications.

NeuroEvolving Robotic Operatives (NERO) A video-game based on the rtNEAT algorithm.

neural network (NN) *See ANN.*

non-deterministic *See deterministic.*

Open Dynamics Engine (ODE) A programming library for simulating rigid body dynamics systems (classical mechanics). Developed by Russell Smith

Object-Oriented Graphics Rendering Engine (Ogre) Ogre3D is a scene-graph-based object-oriented rendering library that abstracts both OpenGL and Direct3D, as well as adding some cross-functional higher level tools such as material scripting.

Object Oriented Input System (OIS) A library providing an abstract interface to various input hardware (keyboard, mouse, joystick, etc.).

Open Audio Library (OpenAL) An open-source, multi-platform, 3D acoustic library designed to inter-operate with OpenGL.

Open Graphics Library (OpenGL) A cross-platform library for visualising 2D and 3D graphics from simple primitives.

partial truth A logical truth value on a continuum between completely true, and completely false. This is usually represented as a numerical value between 0 and 1. Fuzzy set theory is based on the concept of partial truths.

path-finding Path-finding or path-planning is the use of a search algorithm to find a valid and unobstructed route of movement for a particular object through an environment from a start position to a goal position.

real-time simulation "*Real-time*" implies that the operation of a system has a time constraint that relates it to the real-world. In graphical applications this usually implies that the rate of processing needs be fast enough to appear realistic to the human viewer.

renice To change the priority of a process running on a Unix-like operating system using the Unix programme "nice". This maps to the priority in the kernel's scheduler. -20 is the highest priority and 20 is the lowest.

rigid body dynamics The study of the motion of rigid bodies. Rigid bodies are not able to be deformed, but have a geometric shape, and various physical properties such as mass. Rigid body dynamics is a part of classical mechanics, and is based largely on Newton's Second Law. Computers often avoid flexible and deformable calculations as these are more computationally expensive, thus physics libraries for real-time processing are still largely rigid body-based.

road wheel On a tracked vehicle road wheels are those that make contact with the lower part of the belt and support the vehicle's weight.

real-time optimally adapting mesh (ROAM) A level of detail algorithm that optimises large, sprawling terrain meshes by reducing the polygon count of sections of terrain that are further from the viewpoint.

robot Originally *robota*; characters from the Čapek brothers' play *R.U.R.*. For this thesis we use our own specific definition of a robot; "physical systems driven by an artificially intelligent agent". This then separates robots from automata (no intelligence) and pure-software agents.

robot soccer A competitive research platform, where the aim is to develop robots that can play soccer without human intervention, in order to further the study of various robot-related technologies. There are two major competition leagues; FIRA, and RoboCup.

routing algorithm A method used to find optimal paths for transmission through a network. Commonly used in electronic data transmission between computers.

real-time NeuroEvolution of Augmenting Topologies (rtNEAT) An algorithm that enables CPU-efficient machine learning with real-time constraints. Used in the NERO video game.

search algorithm A set of algorithms used in computer science that solve problems which require evaluation of a number of possible solutions before a valid solution is discovered. This may involve scanning a database or phone book for a specific entry, or searching a range of possible navigation options for a shortest path (path-finding). Search algorithms typically abstract items or options to be investigated as "nodes". Once a node has been investigated, the subsequent possible search options are represented by "edges" connecting to other nodes. The search domain can then be abstracted into a tree or graph structure.

sprocket A wheel with teeth that meshes with a chain or track. Usually the drive wheel.

stochastic From the Greek *stochastikos*, meaning "aiming at a target"; implying that decisions require some estimate or guesswork. Stochastic systems or processes are non-deterministic, meaning that subsequent states are the result of predictable actions as well as some random element. In computer programmes there are standard functions for introducing a random element. These are fixed patterns of numbers, but can usually be mathematically seeded with the post-epoch time at programme execution in order to make the system behave differently every time that it is executed; hence non-deterministic from the point of view of the system.

suspension Any components suspended beneath the hull or body of a vehicle on which it runs (wheels, bogies, etc). Suspensions are usually sprung for shock absorbing and cross-country performance.

torque Rotational force. SI units are in Newton-meters ($N \cdot m$).

transmission A complete series of gears and connecting mechanisms that allows a driver to deliver a range of different torques and speeds to the vehicle's wheels.

universal joint Connects two bodies and allows rotation around any axis. One axis can be constrained to a vehicle to allow a trailer to yaw and pitch but not roll independently.

Wumpus Hunt the Wumpus was an early (1972) computer game. In the game a player moved through a map of adjoining rooms, and could do one action per turn; move to an adjacent room, fire an arrow into an adjacent room, etc. The Wumpus game is a classic example used in game theory and artificial intelligence because an AI agent playing the game would have to deal with uncertainty. Because the environment is discretised into 'rooms' rather than continuous distances, predicate logic can be used to determine the next best move.

Bibliography

[1] A. Gerdelan, "A Brief History of AI in Entertainment," Tech. Rep. CSTN-105, CSSG, Institute of Information and Mathematical Sciences, Massey University, Albany, New Zealand, November 2009.

[2] S. Guy, J. Chhugani, S. Curtis, P. Dubey, M. Lin, and D. Manocha, "PLEdestrians: A Least-Effort Approach to Crowd Simulation," in *Symposium on Computer Animation (SCA'10)*, Eurographics/ACM SIGGRAPH, July 2010.

[3] S. Paris, J.-L. Berrou, and J. Amos, "Legion: genesis of a commercial crowd simulator," in *Computer Animation and Social Agents (CASA'10)*, (Saint-Malo, France), June 2010.

[4] L. C. Malone, D. Kaup, R. Oleson, M. Rosa, F. Jentsch, T. L. Clarke, J. Faulkner, and R. Jaggie, "Validation of crowd simulations," in *Summer Computer Simulation Conference (08SCSC)*, (Edinburgh, Scotland), pp. 363–370, June 2008.

[5] A. Lerner, Y. Chrysanthou, A. Shamir, and D. Cohen-Or, "Data Driven Evaluation of Crowds," in *Motion in Games (MIG2009)*, vol. 5884, (Zeist, the Netherlands), pp. 75–83, Springer-Verlag LNCS, 2009.

[6] D. J. Kaup, T. Clarke, R. Oleson, and L. C. Malone, "Crowd dynamics simulation research," in *16th Conference on Behavior Representation in Modeling and Simulation (BRIMS)*, pp. 173–180, 2007.

[7] C. Ennis, C. Peters, and C. O'Sullivan, "Perceptual evaluation of position and orientation context rules for pedestrian formations," in *SIGGRAPH Symposium on Applied Perception in Graphics and Visualization (APGV'08)*, pp. 75–82, 2008.

[8] R. McDonnell, M. Larkin, S. Dobbyn, S. Collins, and C. O'Sullivan, "Clone attack! perception of crowd variety," *ACM Transactions on Graphics (SIGGRAPH 2008)*, vol. 27, no. 3, pp. 26:1–26:8, 2008.

[9] M. Pražák, R. McDonnell, L. Kavan, and C. O'Sullivan, "A perception based metric for comparing human motion," in *Eurographics Ireland Workshop*, pp. 75–80, 2009.

[10] J. H. Clark, "Hierarchical Geometric Models for Visible Surface Algorithms," *Communications of the ACM*, vol. 19, pp. 547–554, October 1976.

[11] L. Williams, "Pyramidal Parametrics," *SIGGRAPH Comput. Graph.*, vol. 17, pp. 1–11, July 1983.

[12] M. Larkin, *Level of Detail Representations and Variation Methods for the Rendering of Large Animated Crowds*. PhD thesis, Trinity College Dublin, Dublin, Ireland., December 2010.

[13] S. Paris, A. Gerdelan, and C. O'Sullivan, "CA-LOD: Collision Avoidance Level of Detail for Scalable, Controllable Crowds," in *Motion in Games* (A. Egges, R. Geraerts, and M. Overmars, eds.), vol. 5884 of *Lecture Notes in Computer Science*, pp. 13–28, Springer Berlin / Heidelberg, 2009.

[14] R. McDonnell, S. Dobbyn, and C. O'Sullivan, "LOD Human Representations: A Comparative Study," in *International Workshop on Crowd Simulation (V-CROWDS'05)*, pp. 101–115, 2005.

[15] R. McDonnell, S. Dobbyn, S. Collins, and C. O'Sullivan, "Perceptual Evaluation of LOD Clothing for Virtual Humans," in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'06)*, pp. 117–126, 2006.

[16] P. Kanyuk, "Brain Springs: Fast Physics for Large Crowds in WALL-E," *IEEE Computer Graphics and Applications*, vol. 29, pp. 19–25, 2009.

[17] D. Ryu and P. Kanyuk, "Rivers of rodents: An animation-centric crowds pipeline for ratatouille," tech. rep., Pixar Technical Memo 07-02, May 2007.

[18] D. Thalmann, C. Hery, S. Lippman, H. Ono, S. Regelous, and D. Sutton, "Crowd and Group Animation," in *SIGGRAPH '04: ACM SIGGRAPH 2004 Course Notes*, (New York, NY, USA), p. 34, ACM, 2004.

[19] S. Donikian, "A Comparative Review of Reactive Behaviour Models as Proposed in Computer Graphics and Cognitive Sciences," in *Motion in Games (MIG2009)*, vol. 5884, (Zeist, the Netherlands), pp. 63–74, Springer-Verlag LNCS, 2009.

[20] C. W. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioural Model," in *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, (New York City, New York), pp. 25–34, ACM, 1987.

[21] W. T. Reeves, "Particle Systems—a Technique for Modeling a Class of Fuzzy Objects," *ACM Trans. Graph.*, vol. 2, no. 2, pp. 91–108, 1983.

[22] M. E. Bratman, *Intentions, Plans, and Practical Reason, Agents*. Cambridge, Massachusetts: Harvard University Press, 1987.

[23] J. V. den Berg, *Path Planning in Dynamic Environments*. PhD thesis, Utrecht University, The Netherlands, 2007.

[24] I. Karamouzas, P. Heil, P. van Beek, and M. H. Overmars, "A Predictive Collision Avoidance Model for Pedestrian Simulation," in *Motion in Games (MIG2009)*, vol. 5884, (Zeist, the Netherlands), pp. 40–52, Springer-Verlag LNCS, 2009.

[25] R. Geraerts, *Sampling-based Motion Planning: Analysis and Path Quality*. PhD thesis, Utrecht University, Utrecht, Netherlands, 2006.

[26] R. Geraerts and M. H. Overmars, "The Corridor Map Method: A General Framework for Real-Time High-Quality Path Planning," *Computer Animation and Virtual Worlds (CAVW)*, vol. 18, pp. 107–119, 2007.

[27] C. J. Scogings and K. A. Hawick, "Altruism Amongst Spatial Predator-Prey Animats," in *Artificial Life XI*, (Winchester, United Kingdom), August 2008.

[28] K. A. Hawick, C. J. Scogings, and H. A. James, "Defensive Spiral Emergence in a Predator-Prey Model," *Complexity International*, vol. 12, p. 37, 2008.

[29] I. Lebar-Bajec, N. Zimic, and M. Mraz, "Simulating Flocks on the Wing: The Fuzzy Approach," *Journal of Theoretical Biology*, vol. 233, no. 2, pp. 199–220, 2005.

[30] D. Helbing and P. Molnár, "Social force model for pedestrian dynamics," *Phys. Rev. E*, vol. 51, pp. 4282–4286, May 1995.

[31] R. Wiedemann, "Simulation des Straßenverkehrsflusses," tech. rep., Schriftenreihe des Instituts für Verkehrswesen, Heft 8, Universität (TH) Karlsruhe, Karlsruhe, Deutschland, 1974.

[32] P. G. Gipps, "A behavioural car following model for computer simulation," *Transportation Research Part B: Methodological*, vol. 15, no. 2, p. 105111, 1981.

[33] S. Krauß, P. Wagner, and C. Gawron, "Metastable states in a microscopic model of traffic flow," *Physical Review E*, vol. 55, pp. 5597 – 5602, May 1997.

[34] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *IEEE Conference on Robotics and Automation*, (Sacramento, California), pp. 1398–1404, April 1991.

[35] R. A. Brooks, "A robust layered control system for a mobile robot.," *IEEE Journal of Robotics and Automation*, vol. RA-2, no. 1, pp. 14–23, 1986.

[36] C. W. Reynolds, "Steering behaviors for autonomous characters," in *Game Developers Conference*, 1999.

[37] R. Narain, A. Golas, S. Curtis, and M. C. Lin, "Aggregate dynamics for dense crowd simulation," in *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, vol. 28, p. 122:1122:8, 2009.

[38] C. W. Reynolds, "Evolution of Corridor Following Behavior in a Noisy World," in *Third International Conference on Simulation of Adaptive Behavior (SAB94)* (D. Cliff, P. Husbands, J.-A. Meyer, and S. Wilson, eds.), pp. 402–410, 1994.

[39] C. W. Reynolds, "Competition, Coevolution and the Game of Tag," in *Artificial Life IV* (R. Brooks and P. Maes, eds.), pp. 59–69, 1994.

[40] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Evolving Neural Network Agents in the NERO Video Game," in *IEEE 2005 Symposium on Computational Intelligence and Games (CIG'05)*, (Piscataway, New Jersey), 2005. Best Paper Award.

[41] K. O. Stanley, R. Cornelius, R. Miikkulainen, T. D'Silva, and A. Gold, "Real-Time Learning in the NERO Video Game," in *Artificial Intelligence and Interactive Digital Entertainment Conference Demonstration Program (AIIDE 2005)*, 2005.

[42] R. Miikkulainen, B. D. Bryant, R. Cornelius, I. V. Karpov, K. O. Stanley, and C. H. Yong, *Computational Intelligence: Principles and Practice*, ch. Computational Intelligence in Games, pp. 155–191. IEEE Computational Intelligence Society, 2006.

[43] R. Miikkulainen, "Creating Intelligent Agents in Games," *The Bridge*, vol. 36, no. 4, pp. 5–13, 2006.

[44] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-Time Neuroevolution in the NERO Video Game," *IEEE Transactions on Evolutionary Computation*, vol. 9, pp. 653–668, December 2005.

[45] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-Time Evolution of Neural Networks in the NERO Video Game," in *Twenty-First National Conference on Artificial Intelligence (AAAI06)*, (Boston, Massachusetts), pp. 1671–1674, 2006.

[46] A. Gerdelan and N. H. Reyes, "A Novel Hybrid Fuzzy A* Robot Navigation System for Target Pursuit and Obstacle Avoidance," in *First Korean-New Zealand Joint Workshop on Advance of Computational Intelligence Methods and Applications*, vol. 1, (Auckland, New Zealand), pp. 75–79, 2006.

[47] A. Gerdelan and N. Reyes, "Synthesizing Adaptive Navigational Robot Behaviours Using a Hybrid Fuzzy A* Approach," in *Computational Intelligence, Theory and Applications* (B. Reusch, ed.), pp. 699–710, Springer Berlin Heidelberg, 2006.

[48] A. Gerdelan, D. Iskandar, A. F. Djohar, and N. H. Reyes, "Utilising the Hybrid Fuzzy A* Algorithm in a Cooperative Multi-Agent System," in *4th Conference on Neuro-Computing and Evolving Intelligence (NCEI'06) and 6th International Conference on Hybrid Intelligent Systems (HIS '06)*, 2006.

[49] L. A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338–353, June 1965.

[50] R. E. Bellman and L. A. Zadeh, "Decision-making in a fuzzy environment," *Management Science*, vol. 17, pp. B141–B164, December 1970.

[51] I. Newton, *Opticks*, ch. Prop. II, Theor. II. London, England: Royal Society, London, 1704.

[52] M. Dougherty, K. Fox, M. Cullip, and M. Boero, "Technological advances that impact on microsimulation modelling," *Transport Reviews*, vol. 20, no. 2, pp. 145–171, 2000.

[53] A. Gerdelan, "A solution for streamlining intelligent agent-based traffic into 3D simulations and games," Tech. Rep. CSTN-072, CSSG, Institute of Information and Mathematical Sciences, Massey University, Albany, New Zealand, January 2009.

[54] A. Gerdelan, "Driving Intelligence: A New Architecture and Novel Hybrid Algorithm for Next-Generation Urban Traffic Simulation," Tech. Rep. CSTN-079, CSSG, Institute of Information and Mathematical Sciences, Massey University, Albany, New Zealand, February 2009.

[55] A. Gerdelan and N. H. Reyes, "Towards A Generalised Hybrid Path-Planning and Motion Control System with Auto-Calibration for Animated Characters in 3D Environments," in *Advances in Neuro-Information Processing* (M. Köppen, N. Kasabov, and G. Coghill, eds.), vol. 5506 of *Lecture Notes in Computer Science*, pp. 1079–1086, Springer Berlin / Heidelberg, 2009.

[56] A. Gerdelan, "Architecture design for self-training intelligent vehicle-driving agents: paradigms and tools," Tech. Rep. CSTN-088, CSSG, Institute of Information and Mathematical Sciences, Massey University, Albany, New Zealand, April 2009.

[57] A. Gerdelan, "Auto-Training Animated Character Motion: A Rule-Base Tuning Hybrid Fuzzy-Genetic Algorithm," Tech. Rep. CSTN-098, CSSG, Institute of Information and Mathematical Sciences, Massey University, Albany, New Zealand, June 2009.

[58] A. Gerdelan and C. O'Sullivan, "A Genetic-Fuzzy System for Optimising Agent Steering," *Computer Animation and Virtual Worlds*, vol. 21, p. 453461, May 2010.

[59] D. P. Playne, A. Gerdelan, A. Leist, C. J. Scogings, and K. A. Hawick, "Simulation Modelling and Visualisation: Toolkits for Building Simulated Worlds," *Research Letters in the Information and Mathematical Sciences, Massey University*, vol. 12, pp. 25–50, 2008.

[60] D. P. Playne, A. Gerdelan, A. Leist, C. J. Scogings, and K. A. Hawick, "Simulation Modelling and Visualisation: Toolkits for Building Simulated Worlds," Tech. Rep. CSTN-052, CSSG, Institute of Information and Mathematical Sciences, Massey University, Albany, New Zealand, March 2008.

[61] S. Singh, M. Kapadia, P. Faloutsos, and G. Reinmann, "SteerBench: a benchmark suite for evaluating steering behaviors," *Computer Animation and Virtual Worlds*, vol. 1, pp. 1546–4261, 2009.

[62] S. Singh, M. Naik, M. Kapadia, P. Faloutsos, and G. Reinmann, "Watch Out! A Framework for Evaluating Steering Behaviors," *Lecture Notes in Computer Science: Motion in Games*, vol. 5277/2008, pp. 200–209, 2008.

[63] C. Ennis, R. McDonnell, and C. O'Sullivan, "Seeing is Believing: Body Motion Dominates in Multisensory Conversations," *ACM Transactions on Graphics (SIGGRAPH 2010)*, vol. 29, no. 4, p. Article 29, 2010.

[64] J. E. McHugh, R. McDonnell, C. O'Sullivan, and F. N. Newell, "Perceiving emotion in crowds: the role of dynamic body postures on the perception of emotion in crowded scenes," *Experimental Brain Research*, vol. 204, no. 3, pp. 361–372, 2010.

[65] J. K. Hodgins, S. Jörg, C. O'Sullivan, S.-I. Park, and M. Mahler, "The Saliency of Anomalies in Animated Human Characters," *ACM Transactions on Applied Perception*, vol. 7, p. Article 1, May 2010.

[66] S. Jörg, J. Hodgins, and C. O'Sullivan, "The perception of finger motions," in *Proceedings of the 7th Symposium on Applied Perception in Graphics and Visualization (APGV 2010)*, (New York, NY, USA), pp. 129–133, ACM, July 2010.

[67] C. Ennis, C. Peters, and C. O'Sullivan, "Perceptual Effects of Scene Context And Viewpoint for Virtual Pedestrian Crowds," *ACM Transactions on Applied Perception*, vol. 8, no. 2, p. To appear, 2011.

[68] G. Junker, *Pro OGRE 3D Programming*. APress, 2006.

[69] M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein, "ROAMing terrain: real-time optimally adapting meshes," in *VIS '97: Proceedings of the 8th conference on Visualization '97*, (Los Alamitos, CA), pp. 81–88, IEEE Computer Society Press, 1997.

[70] K. A. Hawick and A. Gerdelan, "Software Integration Architectures for Agents," Tech. Rep. CSTN-054, CSSG, Institute of Information and Mathematical Sciences, Massey University, Albany, New Zealand, May 2008.

[71] L. Skrba, L. Reveret, F. Htroy, M.-P. Cani, and C. O'Sullivan, "Animating Quadrupeds: Methods and Applications," *Computer Graphics Forum*, vol. 28, pp. 1541–1560, 2009.

[72] R. M. Zlot, A. T. Stentz, M. B. Dias, and S. Thayer, "Multi-robot exploration controlled by a market economy," in *IEEE International Conference on Robotics and Automation* (IEEE, ed.), vol. 3, pp. 3016–3023, May 2002.

[73] W. Sheng and Q. Yang, "Peer-to-peer multi-robot coordination algorithms: petri net based analysis and design," in *Advanced Intelligent Mechatronics. Proceedings, 2005 IEEE/ASME International Conference on*, pp. 1407–1412, July 2005.

[74] A. Gerdelan, "Artificial Intelligence in Robot Soccer," honours thesis, bachelor of engineering, Institute of Information and Mathematical Sciences, Massey University, Albany, New Zealand, October 2006.

[75] S. Russell and P. Norvig, *Artifcial Intelligence: A Modern Approach*. Englewood Cliffs, New Jersey: Prentice Hall, 2nd ed., 2003.

[76] R. Simmons, E. Krotkov, L. Chrisman, F. Cozman, R. Goodwin, M. Hebert, L. Katragadda, S. Koenig, G. Krishnaswamy, Y. Shinoda, W. Whittaker, and P. Klarer, "Experience with Rover Navigation for Lunar-Like Terrains," in *Intelligent Robots and Systems (IROS '95)*, vol. 1, (Pittsburgh, Pennsylvania), pp. 441–446, 1995.

[77] H. M. Choset, S. Hutchinson, K. M. Lynch, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementation*, ch. D* Algorithm, pp. 536–546. MIT Press, 2005.

[78] G. Lawes and M. Barlow, "Visual Realism and Decision Making: A Novel Approach to Real-Time Maritime Battlespace Visualisation," in *SimTecT 2007 Conference Proceedings*, Virtual Environment and Simulation Laboratory (VESL), School of Information Technology and Electrical Engineering University of New South Wales at the Australian Defence Force Academy, Simulation Industry Association of Australia, 2007.

[79] T. Barron, *Strategy Game Programming With DirectX 9.0*. Wordware, 2003.

[80] J. D. Funge, *Artifcial Intelligence for Computer Games*. Wellesley, Massachusetts: A K Peters, Ltd., 2004.

[81] H. M. Choset, S. Hutchinson, K. M. Lynch, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementation*. MIT Press, 2005.

[82] C. H. Messom, "Genetic Algorithms for Autotuning Mobile Robot Motion Control," *Research Letters in the Information and Mathematical Sciences, Massey University*, vol. 3, pp. 129–134, 2002.

[83] L. A. Pipes, "An Operational Analysis of Traffic Dynamics," *Journal of Applied Physics*, vol. 24, no. 3, pp. 274–281, 1953.

[84] S. R. Perkins and J. Harris, "Criteria for Traffic Conflict Characteristics," Tech. Rep. GMR632, General Motors Corporation, Warren, Michigan, 1967.

[85] S. R. Perkins and J. I. Harris, "Traffic Conflict Characteristics: Accident potential at intersections," *Highway Research Record, Highway Research Board, Washington DC*, vol. 225, pp. 45–143, 1968.

[86] W. Leutzbach and R. Wiedemann, "Development and applications of traffic simulation models at the Karlsruhe Institut für Verkehrswesen," *Traffic Eng. Control*, vol. 27, p. 270278, 1986.

[87] S. Krauß, *Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics*. PhD thesis, Mathematisches Institut, Universität zu Köln, 1998.

[88] L. Bloomberg and J. Dale, "Comparison of vissim and corsim traffic simulation models on a congested network," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1727/2000, pp. 52–60, January 2007.

[89] J. Won, S. Lee, S. Lee, and T. H. Kim, "Establishment of Car Following Theory Based on Fuzzy-Based Sensitivity Parameters," *Advances in Multimedia Modeling*, vol. 4352/2006, pp. 613–619, 2006.

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

[91] J. Hamill and C. O'Sullivan, "Virtual Dublin - a framework for real-time urban simulation," in *Winter Conference on Computer Graphics 11*, pp. 1–3, 2003.

[92] H. M. Choset, S. Hutchinson, K. M. Lynch, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementation*, ch. A* Algorithm, pp. 527–536. MIT Press, 2005.

[93] K. A. Hawick, D. P. Playne, A. Leist, A. Gerdelan, and C. J. Scogings, "Its About Time: the Role of Time in Simulations," tech. rep., CSSG, Institute of Information and Mathematical Sciences, Massey University, Albany, New Zealand, January 2009.

[94] S. A. Boxill, "An Evaluation of 3-D Traffic Simulation Modeling Capabilities," tech. rep., Center for Transportation Training and Research, Texas Southern University, Houston, Texas, June 2007.

[95] T. Wan and W. Tang, "An intelligent vehicle model for 3D visual traffic simulation," in *International Conference on Visual Information Engineering (VIE 2003)*, pp. 206– 209, July 2003.

[96] J. Ruttle, "Virtual Traffic Simulation," Master's thesis, Trinity College Dublin, Ireland, September 2008.

[97] R. Hughes and B. Schroeder, "3D Visualization and Micro-Simulation Applied to the Identification and Evaluation of Geometric and Operational 'Solutions' for Improving Visually Impaired Pedestrian Access to Roundabouts and Channelized Turn Lanes," in *5th International Visualization in Transportation Symposium and Workshop*, (Denver, Colorado), October 2006.

[98] R. G. Hughes and D. Harkey, "Evaluation and Application of Pedestrian Modeling Capabilities Using Computer Simulation," tech. rep., Highway Safety Research Center, University of North Carolina, June 2002.

[99] R. G. Hughes, S. Turner, and H. Landphair, "On the Integrated Application of Modeling, Simulation, and 3D/4D Visualization: the Concept of a 'Laboratory' for Non-Motorized Travel Research," in *9th ITS World Congress*, (Chicago, Illinois), October 2002.

[100] A. Gerdelan, "Practicum III: Hybrid Algorithms for Soccer Robots," tech. rep., School of Engineering and Advanced Technology, Massey University, Albany, New Zealand, March 2006.

[101] A. Gerdelan, "A Novel Motor Control Algorithm for Two-Wheeled and Caterpillar-Tracked Autonomous Vehicles Using a Fuzzy Navigation Abstraction," honours year engineering journal articles, School of Engineering and Advanced Technology, Massey University, Albany, New Zealand, 2006.

[102] B. D. Bryant and R. Miikkulainen, "Neuroevolution for adaptive teams," in *Congress on Evolutionary Computation CEC '03*, vol. 3, pp. 2194–2201, 8-12 December 2003.

[103] B. D. Bryant and R. Miikkulainen, "Exploiting Sensor Symmetries in Example-based Training for Intelligent Agents," in *IEEE Symposium on Computational Intelligence and Games*, pp. 90–97, May 2006.

[104] O. Córdon, F. Gomide, F. Herrera, F. Hoffmann, and L. Magdalena, "Ten years of genetic fuzzy systems: current framework and new trends," *Fuzzy Sets and Systems*, vol. 141, no. 1, pp. 5–31, 2004.

[105] S. F. Smith, *A learning system based on genetic adaptive algorithms*. PhD thesis, Department of Computer Science, University of Pittsburgh, Pittsburgh, Pennsylvania, 1980.

[106] C. Darwin, *On the Origin of Species*. Albemarle Street, London: John Murray, 1859.

[107] K. M. Passino and S. Yurkovich, *Fuzzy Control*. Menlo Park, California: Addison Wesley Longman, 1998.

[108] B. F. Allen and P. Faloutsos, "Evolved Controllers for Simulated Locomotion," in *Second International Workshop, Motion in Games*, vol. 5884, (Zeist, The Netherlands), Springer-Verlag LNCS, 2009.

[109] M. Mohammadian and R. J. Stonier, "Fuzzy Logic and Genetic Algorithms for Intelligent Control and Obstacle Avoidance," *Complexity International*, vol. 2, pp. 149–157, Apr. 1995.

[110] R. Neves and M. L. Netto, "Evolutionary Search for Optimization of Fuzzy Logic Controllers," in *1st International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, vol. 1 of *Hybrid Systems and Applications I*, (Singapore), pp. 202–206, Springer, Nov. 18-22 2002.

[111] Valve Software, "Half Lambert http://developer.valvesoftware.com/wiki/Half_Lambert." Public developer wiki, February 2010.

[112] J. H. Lambert, *Photometria sive de mensure de gratibus luminis, colorum et umbrae*. Augsburg, Bayern: Eberhard Klett, 1760.

[113] R. L. Cook and K. E. Torrance, "A Reflectance Model for Computer Graphics," *ACM Trans. Graph.*, vol. 1, pp. 7–24, January 1982.

[114] B. Phong, *Illumination for computer generated pictures*. PhD thesis, University of Utah, 1973.

[115] J. Mitchell, "Shading in Valve's Source Engine," in *Course 26: Advanced Real-Time Rendering in 3D Graphics and Games. SIGGRAPH*, (Boston, MA), 2006.

[116] R. Deriche, "Recursively implementating the Gaussian and its derivatives," tech. rep., INRIA, 1993.

[117] L. G. Shapiro and G. C. Stockman, *Computer Vision*. Upper Saddle River, NJ: Prentice-Hall, 2001.

[118] R. C. González and R. E. Woods, *Digital Image Processing*. Prentice Hall, 3rd ed., 2008.

[119] T. Zuvich, "Vehicle dynamics for racing games," in *Game Developers Conference*, 2000.

[120] P. G. Hewitt, *Conceptual Physics*, ch. Solids, pp. 202, 210, 325n. Addison Wesley Longman, 1998.

[121] M. Wloka, ""Batch, Batch, Batch:" What Does it Really Mean?" http://www.nvidia.de/docs/io/8230/batchbatchbatch.pdf." nVidia, October 2010.

[122] A. Gerdelan, "Mechani$x$: Vehicle Mechanical Simulation," Tech. Rep. CSTN-118, CSSG, Institute of Information and Mathematical Sciences, Massey University, Albany, New Zealand, November 2010.

[123] A. Gerdelan, "Grid-Ireland and e-Research Review for KAREN and BeSTGRID," tech. rep., KAREN, Wellington, New Zealand, September 2008.

[124] I. Newton, *Philosophiæ Naturalis Principia Mathematica*. London, England: apud Sa. Smith, 1687.

[125] R. A. Serway and R. J. Beichner, *Physics for Scientists and Engineers with Modern Physics, Volume II*, ch. 23.3, pp. 714, 1085. Brooks Cole, 2000.

[126] R. A. Serway and R. J. Beichner, *Physics for Scientists and Engineers with Modern Physics, Volume II*. Brooks Cole, fifth ed., 2000.

[127] P. G. Hewitt, *Conceptual Physics*. Addison Wesley Longman, 8th ed., 1998.

[128] A. Leist, D. Playne, and K. Hawick, "Exploiting Graphical Processing Units for Data-Parallel Scientific Applications," *Concurrency and Computation: Practice and Experience*, vol. 21, pp. 2400–2437, December 2009.

[129] K. A. Hawick, A. Leist, and D. P. Playne, "Parallel Graph Component Labelling with GPUs and CUDA," *Parallel Computing*, vol. 36, pp. 655–678, 2010.

[130] K. Hawick, A. Leist, and D. Playne, "Regular Lattice and Small-World Spin Model Simulations using CUDA and GPUs," tech. rep., Computer Science, Massey University, 2009. To appear in Int. J. Parallel Programming (2010).