Proxy Simulations For Efficient Dynamics

Stephen Chenney[†], Okan Arikan[‡] and D. A. Forsyth[§]

Abstract

Proxy simulations *reduce the cost of simulation in large virtual worlds, such as those used in training simulations or computer games. A proxy takes the place of an accurate simulation for objects that are out of view, while the accurate model continues to manage visible objects. A proxy must ensure that objects enter the view at reasonable times throughout the simulation and in states that reflect their time spent out of view. The quality of a proxy simulation is measured by how well it maintains reasonable behaviors, where the definition of reasonable depends on the environment and its application. We present two examples of proxy simulations based on discrete event models: one for city traffic simulation and another for multi-agent path planning and motion. For these examples, we demonstrate dynamics computation speedups of over two orders of magnitude as the environments grow in size and complexity.*

1. Introduction

Realistic dynamic behavior is a key ingredient in many real-time virtual environments, such as those for training or gaming. The cost of generating motion is significant, and has typically limited the complexity of the simulated environment by requiring fewer active objects and poor quality motion. For example, a city driving environment might impose restrictions on the number of vehicles in the city and use only approximate vehicle dynamics.

Ideally, only motion that is perceivable to the viewer should be computed – any other motion cannot possibly affect their experience of the environment. We therefore define the *efficiency*, η, of a simulation as the ratio of the work done computing perceivable motion to the total work done. An ideal simulator has an efficiency of one. With ideal simulation it is possible to simulate New York on a machine capable of inefficiently simulating only a small town, because the complexity of a given view of New York is only as great as the complexity of an entire small town.

In this paper we discuss techniques for producing efficient simulations that do not sacrifice the complexity of visible motion or the user's experience of the environment. Our approach is based on *proxy simulations*, those that take the place of the in-view simulator when maintaining dynamic state that is out of view. Dynamic state consists of all the parameters that describe an object's appearance and motion over time. We define *simulation* to include any means of generating dynamic state in a virtual environment.

A *proxy simulation* is one that approximates out-of-view dynamics to increase the overall simulation efficiency, while also minimizing any errors perceived by a user. An environment uses two simulators: the original, accurate simulator for motion that a viewer can perceive, and the proxy simulator for motion the viewer can't perceive. A proxy's primary task is feeding objects into the view at the right time in the right place. Some of those objects may have been seen before by the viewer, and the location and time that they reappear should reflect any events that would influence the objects while they were out of view. For example, the proxy is responsible for ensuring that when a car on a virtual race track leaves the view, it re-enters again at a place and time consistent with the influence of other cars on the track.

We illustrate the design of proxy simulations with two detailed examples: a basic city traffic simulation and a tile-based troop movement simulation typical of many computer games. In the first case, the proxy saves work by replacing the detailed dynamics of out-of-view traffic with a discrete event simulation that approximates the interactions between objects. The second case also uses a discrete event simulation, but statistical models are incorporated to account for interactions that are otherwise expensive to simulate. In both cases a proxy simulation results in large speedups while preserving the visible behavior of the simulation, allowing more

[†] University of Wisconsin at Madison, schenney@cs.wisc.edu
[‡] University of California at Berkeley, okan@cs.berkeley.edu
[§] University of California at Berkeley, daf@cs.berkeley.edu

than an order of magnitude increase in the number of simulated entities.

2. Related Work

In a variety of systems an accurate simulation model is replaced by a cheaper approximation. Setas, Gomes and Rebordão describe a virtual forest environment [11] that simulates trees as various levels of detail depending on their distance from the viewer. Carlson and Hodgins introduced *simulation levels of detail* [4] for hopping robots. An accurate dynamic model may be replaced with either a kinematic or point-based model depending on the robot's distance from the viewer. Trajectories are compared to assess the quality of the approximations. Carlson and Hodgins also question the broader impact of approximation on system behavior, such as possible changes induced in the outcome of a simulated game.

Grzeszczuk, Terzopoulos and Hinton describe the NeuroAnimator [8]: a system that replaces accurate simulations with neural network approximations. They do not discuss the long term stability of the approximations, nor their potential effect on the global behavior of an environment as the approximated objects interact. Yu and Terzopoulos discuss a virtual marine environment [14] in which *synthetic motion capture* is used to approximate the motion of creatures that are outside the view frustum. The authors note that the approximations introduce significant errors into the on-screen behavior of the fish — these are acceptable in their entertainment application.

All of the above systems perform work for every object on every frame, inherently limiting their efficiency. Chenney and Forsyth describe *dynamics culling* [5], in which systems that are outside the view frustum are ignored completely. When systems re-enter the view, their state is updated with deterministic and statistical approximations to reflect the expected change in state while the system was out of view. Chenney, Ichnowski and Forsyth [6] describe a method for automating the generation of approximations for culling. The algorithms in these papers are limited to objects that do not move very far (so that static bounding volumes suffice to determine visibility).

Sudarsky and Gotsman describe a dynamic occlusion detection algorithm [13] that is applicable when the motion of an object can be contained within a temporal bounding volume — a bounding box that is guaranteed to contain an object for some period of time. They attack the efficiency problem from a pure visibility perspective, ignoring the difficulties of constructing bounding volumes for non-trivial motion such as demonstrated in our examples. Many other visibility schemes support the occlusion of dynamic objects by static ones through the use of temporal bounding volumes (see Cohen-Or, Chrysanthou and Silva [7] for an overview), but only Zhang et. al. [15] discuss the use of dynamic occluders, and then only briefly, and all ignore the issue of constructing bounding volumes for complex motion.

Proxies aim to avoid doing work for non-visible objects. The corresponding problem in distributed virtual environments is to avoid *transmitting* irrelevant dynamic state [12]. Dead-reckoning is the primary technique, in which state is transmitted infrequently, with the state of an object approximated locally on each machine based on its last transmitted state. The small errors due to inaccurate dead-reckoned state are considered acceptable given the major savings in bandwidth. One can also avoid transmitting the state of an object that is out of view — referred to as *area of interest management* [12]. Makbily, Gotsman and Bar-Yehuda [10] describe an algorithm for server-less distributed environments that predicts the time that must elapse before two objects are mutually interested. While their work is similar to that described here, in a distributed environment the precise state of each object is always known by some machine, whereas we avoid computing such precise state at all.

Our work advances the field of efficient dynamics in several ways. We introduce proxy simulations as a means of tracking hidden objects, we explore the relationship between visibility and simulation, we look at ways of measuring error, and we describe two case studies involving different simulation tasks.

3. Proxy Simulations

A proxy simulation takes the place of an accurate simulator for the hidden motion in an environment. We identify two other simulations: the *accurate* simulation for motion in view, and the *full* simulation referring to the accurate simulation model applied to the entire environment, with no proxy. A proxy is best evaluated on quality and cost: it must be of sufficient quality to support a viewer's experience of a dynamic environment while incurring minimal cost for hidden motion.

The simulation of out-of-view objects is dominated by *events* that happen at specific times and have specific outcomes. Chief among these events are the moments when a non-visible object enters the visible region, such that the viewer sees the object. We will refer to these as *view entry* events. It is essential to capture view entry events to ensure that a viewer sees the right thing at the right time. Other important events may be a car crash, or your enemy launching an attack from their remote base. However, the viewer can never perceive these events directly, so they are only of significance because they influence the time when view entry events should occur. This provides us with our first requirement for proxy simulation:

> **Proxy Requirement 1:** The proxy must provide a *reasonable* stream of view entry events.

Chenney and Forsyth [5] refer to this as the *completeness* problem, but provide no method for meeting the requirement.

The idea of reasonable behavior is central to the above requirement. *Reasonable behavior is defined as a part of the environment's model.* For example, a computer game might require that with reasonable behavior monsters should stay dead once killed, that they have a maximum travel speed, that they do not pass through walls, and so on. These conditions on behavior will influence the stream of view entry events. For example, a dead monster should never move itself into the view. A monster that is known to be on the other side of the maze should not be able to rapidly reach a visible region.

It is possible to consider the stream of events produced by an accurate simulation model as the only reasonable event stream. However, this is an overly restrictive requirement. The aim of a virtual environment is, after all, to provide a particular experience for the viewer. This experience is almost never precisely defined down to the timing of individual events — rather it is focused on ensuring that an appropriate set of things happen at appropriate times. In a game, for instance, it is not necessary for the enemy for appear at the 27th second of the game, only that they appear sometime around the 27th second. In any case, most real world systems cannot supply very high degrees of accuracy. We assume that the designer of an environment can define a model of what is reasonable, and proxy simulations must be held accountable only to that model. In section 4.4 we demonstrate one way of validating that a proxy meets the requirements for reasonable behavior, although we note that any discussion of reasonable behavior is likely to be highly dependent on the nature of the particular environment.

It is not sufficient merely to know when an object re-enters the view, it is also necessary to know the object's state at that time. This state must be reasonable or a viewer will detect an error. This gives us our second requirement for a proxy simulation:

> **Proxy Requirement 2:** The proxy must provide reasonable state for objects when they re-enter the view.

Chenney and Forsyth [5] refer to this as the *consistency* problem, and provide a high level approach to solving it. We provide further examples of solutions to this requirement. Note once again the emphasis on reasonable motion: the proxy is acceptable if it satisfies a designer's goals for quality.

Finally, a proxy simulation is intended to save work:

> **Proxy Requirement 3:** The proxy must be significantly cheaper to compute than the accurate simulation.

Ideally the proxy should cost nothing, in which case we would have an ideal simulator as defined in section 1. This is only possible if no work is done to identify view entry events and no work is done to generate reasonable state for re-entering objects. While this can be achieved under extremely limiting conditions, such as no viewer motion and perfectly predictable object trajectories, most interesting cases require that some work be done to compute reasonable view entry events. To see the problem, consider the specific case of a car on a circular race track where the viewer can see only the start-finish line. Say the viewer sees a car cross the line and subsequently leave the view. To achieve perfect efficiency we must be able to predict, with no effort, the precise time that that car will re-enter the view. But that time requires knowledge of the location of all the other cars on the track and how they interact with the specific car in question, so it is highly unlikely we can immediately compute an accurate re-entry time without doing something to evaluate the interactions.

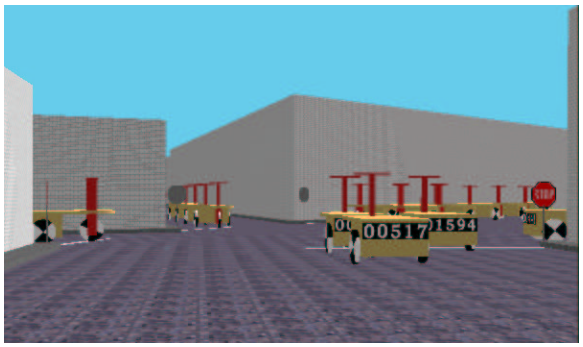

Figure 1: *The traffic simulation, in which cars move through a maze-like network of streets, stopping at intersections as shown here. Each car is uniquely numbered so a viewer may track its progress.*

Having outlined the basic requirements for a proxy simulation, in the following sections we work through two case studies in proxy design: a traffic simulation and a dynamic path planning simulation typical of many gaming environments.

4. Traffic Simulation

The traffic model simulates tricycle cars moving through a maze-like network of roads (figures 1 and 2). Each road is a visibility cell, open only at the ends. We use a cell and portal algorithm [9] to identify at run-time roads that are at least partially visible from the current viewpoint. We now describe reasonable behavior that we wish to maintain, and a proxy that reproduces such behavior at a significantly reduced cost as indicated by experimental results.

4.1. Reasonable Traffic Behavior

The accurate model takes time-steps of fixed length 0.01s. On each step it sets an acceleration for each car to be used throughout the step, and then numerically integrates the motion of each car. Numerical integration is necessary because

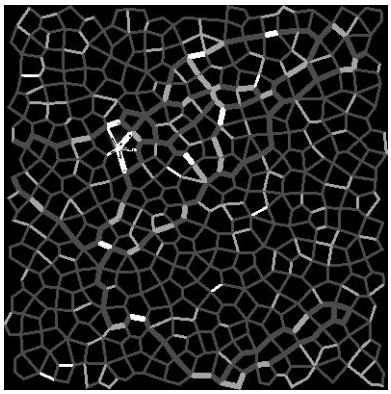

Figure 2: *A map view of the city while a proxy is in use. The cluster of white cells to the upper left of center are the roads potentially visible from the view shown in figure 1. The intensity of other cells corresponds to the number of events processed by the proxy for that road during the previous second of simulation time. Note that most of the roads are grey, indicating that no events were processed and hence that no work was done for those roads.*

the orientation and wheel rotations for each car are specified via a differential equation with no closed form solution. The time-step is kept short to reduce errors in our Euler integration scheme. Other integration schemes do not offer significant speedups in this case, and make it significantly more difficult to manage the behavior of cars at intersections. In any case, the longest useful time-step for any integration scheme for 20fps rendering is 0.05s.

Cars moving under the accurate model exhibit the following traits that together define reasonable behavior for the traffic:

- When a car reaches an intersection, it makes a random choice of which lane to follow next, selecting among the lanes that are not heavily congested.
- The cars obey stop-sign rules at intersections: they come to a stop, wait their turn to move into the intersection, and then pass through, allowing the next car in.
- Between intersections, each car follows a pre-defined acceleration profile, except that cars slow to avoid collisions with the car in front. If the car in front stops, so will the follower, causing queues to form at intersections as cars line up waiting to enter.

Together, all of these properties lead to one property that is visible to an adversarial user and sensitive to all of the others: car travel times. A viewer can measure travel times by noting the location of a car at two distinct points in time. If in a proxy simulation the cars don't make random choices, then the density of cars will be wrong which will impact travel times. If cars don't stop, or jump the queues at intersections, they will get places too fast. Cars following the wrong acceleration profile will move too fast or slow.

We use travel times as our primary measurement of the proxy's plausibility. Individually, the other properties should be satisfied within the visible regions, but in hidden regions the viewer cannot directly determine the density, or detect whether cars are stopping at intersections, so the proxy need not produce measurements for such things. In addition to travel times, we also visually assess the quality of the proxy by watching cars as they enter the view.

Note in particular that a viewer may be aware of the existence of traffic jams and other such phenomenon, having seen them in the recent past. The viewer is therefore aware of the factors influencing travel times and we must take such knowledge into account in the proxy model. For instance, it is not possible for the proxy to assume independence between cars — the travel time must be conditioned in some way on the location of the other cars in the simulation.

4.2. A Discrete Event Traffic Proxy

Recall the proxy's tasks: to ensure that cars enter the view at the right place and time, and to provide state for cars that re-enter the view, all with a minimum of computation.

4.2.1. Visibility and the Proxy

Rather than identify precisely when cars become visible, we instead choose to identify when cars enter the potentially visible set of roads. The accurate simulation then takes over, and ensures that a car is in the correct place when the viewer actually sees it. Cars enter the visible set of roads in two ways: they can drive onto a road that is already visible, or they can be on a road that becomes visible as the viewer moves. Note that the model we use places no restrictions on viewer motion.

The proxy simulation keeps track of all the cars on every road in the city, and the time when each car entered its current road. This ensures that we know which cars become visible if the viewer sees a road, and also provides enough information to account for traffic jams and other interactions between cars. On each frame, the visible roads are checked for cars that weren't previously visible. Such cars are put in an active list for the accurate simulator to simulate, while all other cars are managed by the proxy. Also on each frame, the locations of cars on the active list are checked against the set of visible roads, and cars that are no longer visible are taken of the list and handed back to the proxy.

The proxy must compute the appropriate times for cars to move from one road to the next, and hence keep the associations between roads and cars correct. To determine when updates are required, we use a discrete event model for the motion of the cars. This model only considers a car when it might be moving from one road to the next, and does no

processing for that car at other times. We thus achieve significant cost savings with the proxy, because the overhead of managing an event queue for many objects is much less than the cost of an accurate simulation for those objects.

The discrete event model uses two events, and there may be zero or one event in the queue for each car. The events are:

Enter events occur when a car *might* reach the end of a road and enter an intersection. We schedule these events optimistically, and test whether the car is really entering the intersection only when the event is processed.

Exit events occur when a car moves out of an intersection and onto a road proper.

Each lane maintains a list recording which cars are currently on that lane, and in which order. This allows us to identify visible cars when we identify visible roads. Each road intersection maintains a list of the cars that are at stop signs waiting to enter the intersection, and in what order they arrived. Cars store pointers back into any intersection and lane lists that they are currently on.

When an Enter event is processed, the car may be free to enter the intersection, it may be stopped waiting for another car in the intersection, or it may be queued behind cars that are already stopped. Depending on the case, we shift the car among the lists and possibly schedule a new event for it. Exit events are processed by scheduling the car's next Enter for the soonest time that it could reach the end of the road, which is computed in a preprocessing step based on the length of the lane and the acceleration profile of the car. If necessary, we also wake other cars that were stopped at the intersection this car is leaving and schedule new events for them.

Sometimes Enter event processing moves a car into a visible road, in which case it is removed from the proxy and passed over to the accurate simulation. Because the car was previously stopped at an intersection, it is simple to compute other parts of its state as described below. Furthermore, we may have to insert events when a car exits the visible region and is taken over by the proxy.

By processing events in the correct order, the proxy maintains the correct associations between cars and roads, taking into account all the important dependencies between cars. Work is only performed when an event is processed, resulting in highly efficient simulation. The assumptions made in scheduling future events introduce some errors, but we claim that they are small, and our experiments described below support that. It remains to discuss how state is set when an object re-enters the view.

4.3. Generating Complete State

The proxy must supply complete state for cars that re-enter the view. This can happen in two ways:

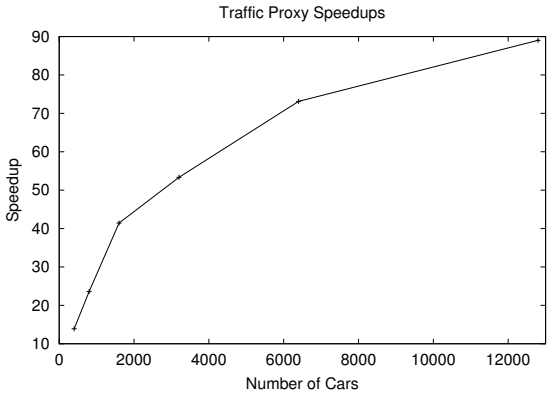

Figure 3: *The speedups achieved by the proxy simulation. We see almost two orders of magnitude speedup for 12800 cars in the simulation.*

Arrival: the car may drive onto a visible road when it enters an intersection. But cars always enter an intersection from a standing start and aligned along the road, so we know its correct position (the end of the road), speed (zero) and orientation. We do not know the correct rotation angle for each wheel, which is determined by the length of the path traveled by the wheel. But we can safely assume that a viewer has no idea exactly which path a wheel has taken, so we set the rotations randomly.

Exposure: the viewer may turn a corner and reveal an entire road and all the cars on it. We isolate the road, shifting it back in time to the entry time for the first car on the road, and simulate just the road forward from there to the current time, adding cars at the time they entered according to the proxy, and stopping all the cars in a queue at the end of the road (where they must be stopped, or they would not be on the road). This process generates highly accurate state for cars re-entering the view, but significant lag may result when a busy road comes into view. This lag could be reduced with a more aggressive approximation strategy, such as simply placing all the cars on the road stationary in a queue, at the expense of greater errors.

4.4. Results

The city traffic environment using the proxy described above generates visually reasonable behavior when compared to a full model. It also achieves significant speedups, allowing the simulation of cities larger than could be handled with an accurate simulation alone. We performed a set of experiments to determine the proxy's efficiency, and another to measure its quality.

We determined the dynamics computation speedup achieved as the number of cars in the simulation is increased, by comparing the time taken to simulate one frame of dynamics at 20fps (0.05s of simulated time per frame) with

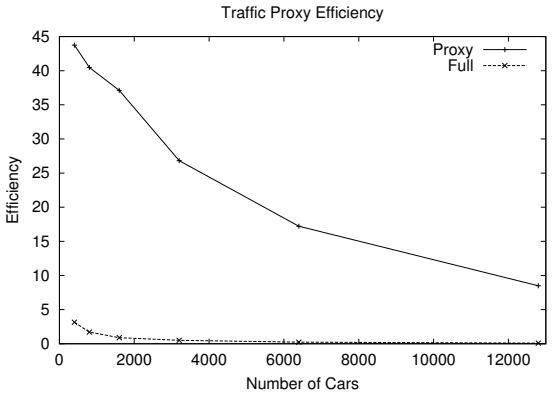

Figure 4: *The efficiency achieved with the proxy and full simulations, computed according to the definition in section 3. The efficiency drops as the number of objects increases. This is expected, because the traffic density is kept constant as the total number of objects grows. Hence the number of objects in view remains roughly the same, as does the amount of work done simulating those objects. Yet the number of out of view objects is increasing, so more total work is required and the efficiency declines. We get proxy efficiencies ranging from 45% for the smallest city with 400 cars down to 8% for the largest city with 12800 cars, while the full simulation efficiency ranges from only 3.1% down to 0.1% for the largest city, reflecting more than an order of magnitude gain in efficiency.*

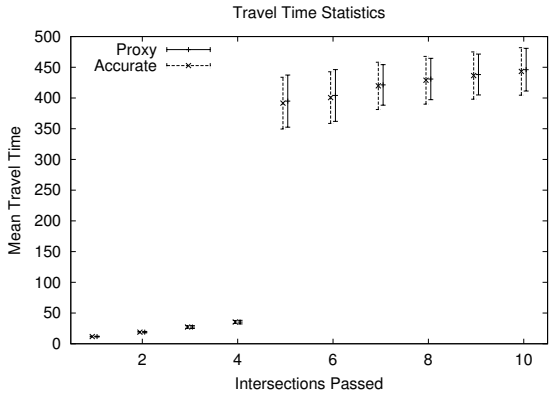

Figure 5: *The average time taken by a car to reach various way-points on a fixed route, as estimated by the full model and the proxy model. The way-points are numbered from 1 to 10 across the bottom. The error bars represent one standard deviation. The averages were computed by fixing the path of one car, then running multiple simulations with the other cars starting out in different configurations. The times estimated by the proxy are reasonably close to those estimated by the full model, with a roughly 1% difference in the estimates, and almost the same variance.*

the full model and the proxy. The city size was increased along with the number of cars, keeping the overall density constant. For each data point we averaged timing data from four runs each of ten minutes duration with different sets of initial conditions. A different animated viewpoint was used for each run to simulate a moving viewer. All computations were performed on a Pentium III 800MHz. The speedup is plotted in figure 3. From the same data we computed the efficiency of the proxy simulation, plotted in figure 4. Our data indicates that it is possible to accurately simulate only 2000 cars in real-time on this machine, while the proxy could simulate 12800 cars using only 0.07 seconds of real time for each second of virtual time.

A tough test for the quality of the proxy is to examine the estimates it produces for travel times. We do this by fixing the path of one car, and running multiple simulations with the other cars starting in different configurations. We then compare the average time it takes the car to follow its path when the proxy is employed and then the accurate model is employed. This experiment is the virtual equivalent to determining how long it takes to drive home from work, and checking how both the accurate and proxy models estimate how that time varies over different days. The results are plotted in figure 5, using data gathered from ten simulation runs in a city with 1600 cars.

These results indicate that the proxy achieves significant speedups while producing very accurate estimates of system behavior. Note, however, that the efficiency of the system as a whole goes down as more and more out of view objects are introduced. This is to be expected: we still do some work for every out of view object, so as more are added more time is spent working on them, reducing efficiency. The only way to avoid the efficiency drop is to ignore some objects completely, by adding and deleting them entirely at a boundary or by some other method.

5. Case Study: Path Planning and Collision Avoidance

Real time strategy games provide an important example of a very large virtual environment with complex object dynamics. Typically, a virtual battle is modeled by simulating the behaviors of individual objects that move on a 2D terrain or attack other objects on the user's orders. In such games path planning and collision avoidance between objects constitute the bulk of the computational workload of the simulator, due to both the large numbers of objects and the typically high complexity of path planning and collision detection algorithms.

Reasonable behavior for objects in this model has them following shortest paths between two points, always avoiding obstacles and each other. Hence the time taken to get

somewhere and the path taken depends on the nearby objects. The primary concern for gaming applications is that a player who knows about the proxy should not have an advantage over a player who does not. Thus, the proxy simulation should meet the following requirements for every individually identifiable object:

- Objects in any part of the battlefield should not pass through stationary obstacles (otherwise the informed player could "teleport" units).
- Inter-object interactions must be accounted for. For example, an object should not be able to pass through a bridge whose entrance has been blocked by other static objects.
- Objects should have speeds consistent with their environments. For instance, the average speed of an object should be low if it has interacted with lots of other dynamic units recently. This means that objects should (a) arrive at their destinations at times consistent with a full simulation and (b) have encounters along their paths that are also consistent. For example, objects should encounter ambushes at reasonable times.

We first describe how the accurate simulation achieves this behavior. This description is intended to give the flavor of our path planning implementation, rather than be complete. We then move on to our proxy model. See [2] for a detailed description of the path planning algorithm and its proxy.

5.1. Accurate Path Planning and Collision Avoidance

The path planning and collision avoidance is done in three steps. Firstly, an initial path that does not pass through any fixed obstacles (represented as simple polygons) is constructed. This is accomplished via the precomputed visibility graph of the obstacle polygons (figure 6). In this context, if two points are mutually visible then an object can move on the line connecting them without colliding with a fixed obstacle. The visibility graph has a node for each vertex of each fixed object and two nodes are connected with an edge if the vertices can see each other. Free space is also partitioned into regions such that points in the same region have the same visible obstacle vertices (the same *horizon*). Note that the shortest path between any two points that are not directly visible to each other must involve going to a vertex on the horizon of the beginning point, traversing zero or more edges on the visibility graph to a vertex in the horizon of the ending point and finally going to the ending point (figure 6).

Using precomputed shortest paths between any two point of the visibility graph, the vertices in the horizons of the beginning point and the ending point can be enumerated pairwise to find the shortest path. Note that if the points are directly visible to each other, than the shortest path between them is the straight line which can be detected by ray-casting in 2D. Since the number of vertices in the horizon of any point is small, the shortest path enumeration can be done

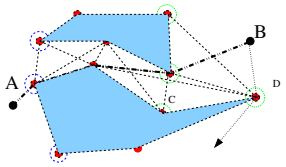

Figure 6: *Shaded polygons represent obstacles on a 2D terrain. The filled circles are the obstacle vertices and form nodes in the visibility graph. Dashed lines are the edges of the visibility graph for these obstacles. The horizon vertices for points A and B are circled in dashes and dots respectively. Note that a shortest path between any two points must first reach a vertex of the horizon of the starting point, traverse zero or more edges in the visibility graph and go from a vertex in the horizon of the destination point to the final destination point. The shortest path between A and B is shown in the figure.*

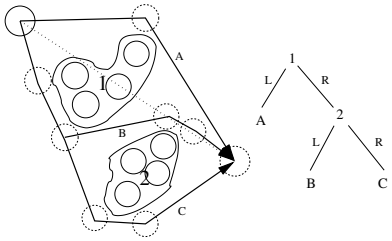

Figure 7: *There are static objects, groups 1 and 2, on a moving object's intended path (shown as a dashed line). The possible paths are enumerated by considering traversing the left and right sides of the obstructing group (1) and continuing the process recursively. Note that traversing one side of the obstructing group (1) may reveal other obstructions (2). The plans discovered this way can be represented as a binary tree whose leaves are the candidate paths.*

quite fast. Moreover, the complexity of this space decomposition is usually small, so the whole precomputation can be stored very efficiently. Finally, with the shortest paths between any two obstacle vertices available, the enumeration can be repeated to get the second or third best routes if the best one is blocked by other objects.

The piecewise linear path plan for objects *A* can still intersect other mobile objects along the way. Collisions with stopped objects (*static*) and moving objects (*dynamic*) are handled differently as dynamic objects can move out of the way whereas we may need to plan around the static objects (figure 7). The collisions are detected by checking the future locations of object *A* for some time (plan ahead time) for the presence of any other object, *B*. If a collision is found and *B* is static, then the right and left sides of *B* are evaluated for a detour. This process is continued recursively for any collisions found during the process until all alternatives are

evaluated or a good enough path is found. Note that the number of recursions (maximum recursion depth) done at this step controls the quality of the path. Moreover, the amount of time in the future that is checked for any collisions also provides a quality versus computation time tradeoff. If the colliding object B is dynamic, then the path plan for A is evaluated to decide whether to wait to let B clear the path or actively plan around it as if it was static.

In order to expedite the collision checking, we use a hierarchical decomposition of space and tune the parameters controlling the path planning and collision avoidance (plan ahead time and maximum recursion depth) for optimal performance. We also employ partial path planning where the path plan is constructed only for some time in the future is constructed and objects re-plan their paths if they reach the end of their partial plans before reaching their final destination.

5.2. Proxy Model Overview

In order to obtain the necessary visibility information, we subdivide the battlefield to smaller rectangles that act as our cells. The viewer has a rectangular overhead view of the battlefield, so can only see the cells that intersect with this rectangular region. Only the objects that are in these cells may be visible.

Like the traffic example, the proxy model for path planning and collision avoidance is a discrete event simulator that is used to manage the membership information of every object for every cell. More specifically, the proxy is used to make sure cells contain a list of objects that are time stamped with their entry and exit times to the corresponding cell. At every frame, dynamic state for objects entering any of the visible cells is obtained from the proxy simulator and the proxy simulation is turned off for those objects. Similarly, objects leaving the visible cells are switched to the proxy simulation.

The proxy simulator gathers the path plans for the objects that it needs to simulate and inserts each object into the cells that its path plan intersects in the future. The proxy simulator also marks each object with the corresponding entry and exit time stamps for every cell that it is scheduled to visit. The time stamps are computed by considering the object's speed and the other objects with which it is expected to interact along the path. These can be identified using a spatiotemporal data structure for each cell that reports space-time path intersections within that cell.

The outcomes of detected interactions are approximated by delays to the interacting parties' original path plans. These delays are sampled from a probability distribution over delays that is learned in a preprocessing phase that runs the full simulation and gathers statistics on delays. The distribution is represented as a mixture of gaussians and learned using an EM algorithm [3]. This is adequate for capturing reasonable speed relationships for the dynamic objects.

Introducing a delay in an object's intended plan may invalidate some of the interactions that have been computed previously, in particular all those objects that previously expected to interact with the delayed object. Thus, all the future interactions must be rechecked after the introduction of a delay. Since objects can have very long path plans into the future, this step may involve re-computation of interactions for lots of other objects already in the proxy simulator. We overcome this problem by computing interactions only for a particular amount of time in the future and rechecking. The amount of time that we will compute an object's interactions is controlled by a latest interaction time parameter.

Since only dynamic objects are considered by the proxy simulator and the path plans for every dynamic object already account for the stationary obstacles and static objects, the first two requirements for reasonable behavior from section 5 are automatically satisfied. The third is satisfied by the introduction of probabilistic delays learned form the full simulation.

The main source of error in the proxy simulator is the approximation of dynamic object interactions as delays. This usually a reasonable assumption considering the fact that dynamic objects can not block a path (they either move out, or we plan around them). These delays are sampled from delay statistics gathered from the full simulation as a preprocessing step, so on average they are consistent with the full simulation. One infrequent case where the proxy is seriously awry is when two objects deadlock in try to pass, but we claim this has a very little effect on the behavior of the simulation as a whole.

5.2.1. Events for the Path Planning Proxy

Our discrete event simulator has four different events:

Stop: An object in the proxy simulator has reached its destination. The proxy simulator computes the time to stop by adding the delays computed for the dynamic object interactions to the original estimate of the time to stop and inserts this time into the event queue. When this event happens, the proxy simulator switches the object from dynamic to static and removes it from all the cells that it has been associated with.

Replan: Since objects may also have partial path plans, the validity of a path plan may expire before the object reaches its final destination. In this case the proxy must generate a new plan from the last point in the current plan to the final target of the object.

Reinsert: Note that the proxy simulator only considers a finite time in the future for dynamic object interactions in order to avoid discovering interactions too far in the future that will be rolled back when one of the interacting parties collide with another object before the interaction.

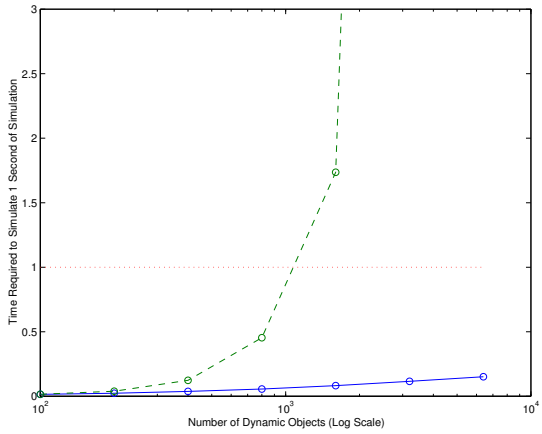

Figure 8: *The real time required to simulate 1 second of simulation time as a function of the log of the number of objects on a constant density terrain. The upper curve (dashes) represents the simulation without proxy and the lower curve (full) represents the simulation with proxy. The dotted horizontal line is the real-time cutoff. Since the object density is constant for all simulations, the user sees approximately the same number of objects in the view. Since the proxy simulator does very little work for the invisible objects and their interactions are handled using a discrete event simulator, the simulation cost increases much slower than the full simulation. The very fast increase in the full simulation cost comes from the fact that the path plan revisions needed to avoid collisions between dynamic objects tend to snowball, while the proxy approximates dynamic-dynamic object interactions with probabilistic models, thus avoiding local path plan revisions.*

Thus it needs to go back and compute new interactions periodically to capture all dynamic interactions for each object. This event is generated for every object when their latest interaction time is hit.

Entry: This event is scheduled for objects in the proxy simulator that will enter one of the visible cells. When this event happens, the proxy simulator deletes the object from its previously associated cells and samples a state for the object. Since the proxy simulator has the path plan for every object, it can sample a point on this path as a function of the entry time and all the interactions that the object carried out in the proxy simulator.

The full simulator checks if an object has moved out of the visible cells whenever the new position of the object is computed and sends these objects to the proxy server.

5.3. Results

We have tested our proxy simulator on different maps with different numbers of uniquely identifiable objects. Figure 8 shows the amount of real time required to simulate 1 second of virtual dynamics as a function of the number of objects at a constant density on a Pentium III 800MHz computer. For all the simulations, objects are distributed randomly on the map and pick random places to go. The object density on the terrain is kept constant by increasing the size of the world as the number of objects is increased.

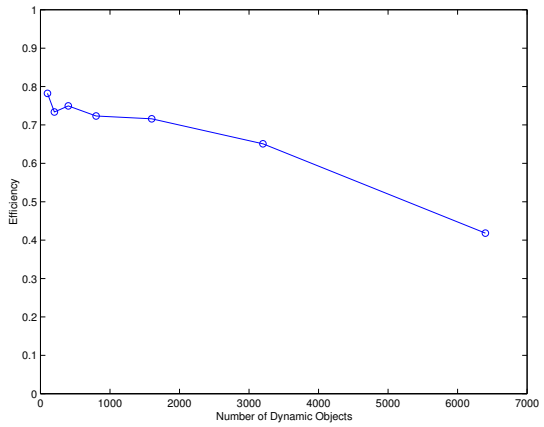

Figure 9: *The efficiency as a function of the number of objects on a constant density terrain, computed directly from the definition in section 3. The efficiency falls because as the number of out of view objects increases we must perform some, even if only a little, work for each one.*

Increasing the number of objects increases the number of possible interactions quadratically. Since the full simulator needs to re-plan in order to avoid collisions, it may have to deal with additional collisions that the revised plan may cause, suffering additional cost for every interaction that needs plan revision. On the other hand, the proxy simulation approximates these interactions with probabilistic delays and thus does not re-plan. Even though the proxy simulator still detects all the interactions between the dynamical objects, the cost of handling these interactions is considerable cheaper than the full simulation. Figure 9 shows the efficiency for the same tests.

6. Conclusion

We have defined proxy simulations for virtual environments and given the basic requirements of their implementation. The traffic and path planning case studies clearly demonstrate the effectiveness of proxy simulations for reducing the cost of motion generation, suggesting that a proxy simulation can manage at least two orders of magnitude more objects than a full simulation, without significantly sacrificing the quality of a user's experience. There are many possible future research directions.

There are various statistical aspects implicit in our discus-

sion of reasonable proxy behavior. The idea of reasonable behavior is best modeled in statistical terms, and in both our case studies we use simple statistics to either validate the performance of the model or as part of the proxy itself. Statistical techniques may provide a way to further quantify the behavior of proxy models. For instance, it is likely to be important to retain the average behavior of objects, but not so important to restrict the variance. It is also unlikely to matter if one car in a hundred has an outrageous travel time. Exploiting this may allow even greater efficiency gains.

The quality metrics we use are potentially more demanding than necessary. In the traffic simulation, for instance, not every car is likely to be important to a viewer. A large amount of work could be saved by relaxing the travel time constraint on non-essential vehicles, which could free us to completely ignore them if they are not influencing the important objects. This is essentially what current traffic related games do: the viewer and their immediate adversaries are important, but they are always close together around the viewer, allowing all other cars to be simulated only in a sphere about the viewer [1]. A proxy simulation could extend this idea to important objects that range widely, and would otherwise require a very large active sphere.

Finally, proxy simulations that include random components, like our tile-based path planning example, appear to offer very large efficiency gains. They also provide us with a handle on scenario management in virtual environments. The proxy has a choice, made randomly in our model, for how the object should behave. If instead this choice is made based on some other criteria, like its benefit to constructing a scenario, the proxy can guide the simulation in particular ways. For instance, if we wish to test a trainee soldier with an unexpected enemy encounter, the travel times for enemy objects could be manipulated to ensure that they intersect the trainee.

The ultimate aim is to construct tools that automatically generate proxies, given an accurate model and the desired quality measures. Such tools will require advances in many areas, including proxy design, statistical inference and event modeling techniques.

Acknowledgement: This work was supported by ONR award N00014-96-11200.

References

1. Joe Adzima. Using AI to bring open city racing to life. *Game Developer Magazine*, pages 26–31, December 2000. 10
2. Okan Arikan, Stephen Chenney, and D. A. Forsyth. Efficient multi-agent path planning. In *Proceedings of the 2001 Eurographics Workshop on Animation and Simulation*, 2001. To appear. 7
3. Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995. 8
4. Deborah A. Carlson and Jessica K. Hodgins. Simulation levels of detail for real-time animation. In *Graphics Interface '97*, pages 1–8, 1997. Kelowna, BC, Canada, 21-23 May 1997. 2
5. Stephen Chenney and David Forsyth. View-dependent culling of dynamic systems in virtual environments. In *Proceedings 1997 Symposium on Interactive 3D Graphics*, pages 55–58, April 1997. Providence, RI, April 27-30. 2, 3
6. Stephen Chenney, Jeffrey Ichnowski, and David Forsyth. Dynamics modeling and culling. *IEEE Computer Graphics and Applications*, 19(2):79–87, March/April 1999. 2
7. Daniel Cohen-Or, Yiorgos Chrysanthou, and Cláudio T. Silva. A survey of visibility for walkthrough applications, 2000. SIGGRAPH 2000 Course Notes 4. 2
8. Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Computer Graphics, Proceedings of SIGGRAPH 98*, pages 9–20. ACM SIGGRAPH, 1998. 2
9. David Luebke and Chris Georges. Portals and mirrors: Simple, fast evaluation of potentially visible sets. In *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, pages 105–106, April 1995. 3
10. Yohai Makbily, Craig Gotsman, and Reuven Bar-Yehuda. Geometric algorithms for message filtering in decentralized virtual environments. In *Proceedings of the 1999 Symposium on Interactive 3D Graphics*, pages 39–46, 1999. 2
11. M. N. Setas, M. R. Gomes, and J. M. Rebordão. Dynamic simulation of natural environments in virtual reality. In *SIVE95: The First Workshop on Simulation and Interaction in Virtual Environments*, pages 72–81, July 1995. University of Iowa, Iowa City, IA. 2
12. Sandeep Singal and Michael Zyda. *Networked Virtual Environments: Design and Implementation*. Adison Wesley, 1999. 2
13. Oded Sudarsky and Craig Gotsman. Dynamic scene occlusion culling. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):13–29, 1999. 2
14. Qinxin Yu and Demetri Terzopoulos. Synthetic motion capture: Implementing an interactive virtual marine environment. *The Visual Computer*, pages 377–394, 1999. 2
15. Hansong Zhang, Dinesh Manocha, Thomas Hudson, and Kenneth E. Hoff III. Visibility culling using hierarchical occlusion maps. In *Computer Graphics: Proceedings of SIGGRAPH 97*, pages 77–88, August 1997. Los Angeles, CA. 2