Parallel Motion Synthesis *

Martin Preston
Computer Graphics Unit, Manchester Computing Centre,
University of Manchester, M13 9PL, U.K.

Abstract

Increasingly more complicated models of virtual worlds have necessitated computer assistance for animators who wish to produce realistic motion. Such approaches are, however, reliant on computationally expensive algorithms, and are frequently viewed as too complicated for commercial use. This paper presents a number of ways of parallelising these algorithms for distributed computation, in order to illustrate how novel architectures may be employed in the synthesis of pleasing computer animation.

Keywords: *Computer Animation, Parallelism, Spacetime Control, Dynamics, Behavioural Animation.*

1 Introduction

Since its genesis in the early sixties computer animation has provided an elaborate and powerful tool to allow scientists and entertainers to visualise information in a virtual world. During its lifetime ever more elaborate ways of modelling this world have emerged, but fundamentally the creation of animation has remained a process of describing the behaviour of such an environment, which is then recorded via some medium.

While computer animation remained the preserve of a select few experts, the mechanisms of motion production remained essentially similar to those involved in the production of conventional cel animation. However, computer animation is a powerful visualisation tool, and such primitive control methods have hindered its more widespread use. Consequently the focus of research effort in computer animation over its lifetime has been the production of more pleasant mechanisms of control over the virtual world, which will enable less experienced users to profit from the use of animation.

To date the most promising products of this work have relied on allowing the computer to provide a model of the virtual world that is more advanced than the simple multi-dimensional drawing board employed by early systems [30]. By encapsulating some of the physical behaviour of the real world, it has been possible to assist the animator, for example by preventing implausible motion [18, 9] or providing higher level interfaces to the motion production

*Published in *International Workshop on High Performance Computing for Computer Graphics and Visualisation.* ©Springer-Verlag

process [34, 5, 26]. These techniques have advanced to such an extent that, in certain styles of animation, it is possible for the animator to function as a conventional director who provides indications of the desired film to a collection of semi-autonomous virtual actors [14, 16].

Such advanced techniques have yet to gain acceptance in commercial animation houses, largely as the computational expense precludes interactive use of such algorithms, and batch processing of animation is unpleasant. Consequently, the state of the commercial art in computer animation lags considerably behind current research.

Computer animation is currently undergoing a change in focus, as modern computer systems are capable of creating realistic animation interactively in response to the actions of an audience. In order to make such virtual environments respond pleasantly, it is important that the simulator is capable of creating motion without the intervention of a skilled computer animator in every motion [13]. Such *machine initiated* motion environments are heavily reliant on the efficient implementation of the realistic motion production algorithms devised for computer animation, and so are being hindered by their computational expense.

This paper advocates the application of parallel, or distributed, computer architectures to the use of high level computer animation algorithms. By employing either specialised tightly coupled parallel machines, or networks of workstations (a resource more likely to be available to a contemporary animation house), it is possible to produce motion using advanced high level control algorithms sufficiently quickly for use in commercial computer animation. Furthermore, such techniques are extremely well suited to application in the emerging machine initiated motion environments, where these complicated algorithms must be used in a time critical manner.

To support the thesis that motion synthesis is an important target for parallelisation, this paper describes some of the more significant animation generation tools, and highlights the features of each that may be exploited in a parallel environment.

Section 2 discusses the limited existing attempts to employ parallelism in the synthesis of computer animation. Following this section 3 introduces the concepts common to all advanced computer animation techniques, principally their reliance on an internally consistent, almost intelligent, world model. These techniques can be split into two principal classes for our purposes, physically based and behavioural animation. Section 4 relates how various physically based techniques can be parallelised and section 5 discusses how behavioural animation may be implemented profitably in a distributed setting.

It is important, at this stage, to differentiate between the twin goals of such effort, reduced synthesis time & real-time operation. The synthesis tools described in sections 4 and 5 are intended for use in the production of motion which forms *part* of the finished animation. Therefore there is no need to attain real-time synthesis speeds, though obviously it is desirable. However, in section 6, we outline how machine initiated motion environments may be implemented in parallel, and address how real-time movement is required.

Finally Section 7 reviews the advantages of parallel implementation and suggests areas for promising further work.

2 Related Work

Many early computer animation systems were intended primarily for use by conventional cel artists, who could employ the processing power of computers to handle many of the laborious jobs common to the animation industry, such as in-betweening (drawing hundreds of similar frames) and colouring of acetate sheets (or cels) [4]. These simple, though extremely useful, tasks are still employed in commercial animation production [22], and require limited processing power.

However, once it became possible to animate and render three dimensional spaces, the traditional animation techniques became cumbersome to use, and the focus of research effort moved to the creation of more convenient interfaces. The advantage of using computers in the production of animation has always been the perceived improvement in the speed at which films may be produced. Consequently the overriding concern in the creation of newer interfaces remained that the animator should be able to produce motion quickly on current hardware. Whilst many simple three dimensional animation tools have been developed with this in mind, this requirement has become a hindrance in the complexity of motion which can be synthesised. As a result of this many animation tools have been developed purely for research, and have yet to find acceptance in the commercial arena.

One of the most popular forms this research has taken is the creation of motion as a result of rigid body dynamic simulation [33]. Here a Newtonian simulation process is performed, which enables the behaviour of physically defined solids to be animated, usually as a result of force impulses provided by an animator. Simulation of systems of any size has meant batch processing, and has reduced the ability of an animator to come to terms with this approach. One early proposed [1], though not implemented, solution to this involves the dynamic simulation of links of rigid chains, where the behaviour of the system can be found using simple parallel processing. Armstrong and Greens' [1] approach enabled some improvement in performance, but would not scale well as the number of processing elements employed increased, largely as the degree of parallelism was entirely dependent on the particular animation subject.

A similar approach was adopted by Friedmann & Pentland [8] where the behaviour of a system of unconnected bodies could be simulated in parallel, with efficient collision detection performed for a limited subset of object types. Whilst this technique indicated that some parallelism could be found, the application exhibited poor scalability.

The creation of motion as a result of application of forces is, whilst computationally convenient, an anathema to most animators. Consequently many attempts have been made to provide a high level interface to the motion production process, effectively shielding the animator from the underlying force-body

simulation process. One such technique [14] has involved the evolution of high level controllers, which are created to solve a particular problem (such as animal locomotion) as a result of applying forces to limbs. A high quality controller can be synthesised by evaluating a large collection of controllers, successful candidates are then evolved into better controllers, using genetic algorithms. A recent implementation of this technique, by Ngo & Marks [21], has proposed the application of parallelism in the evolution process. Here a Connection Machine 2 has been employed, and though unfortunately the limitations of the control afforded preclude its widespread use, it indicates the susceptibility of such animation techniques to parallelism.

Computer animation techniques, in their quest for higher level control, are requiring progressively more computational power and in doing so serving to alienate the commercial animation users. Whilst parallel rendering has become common industry practice, the application of parallelism to the production of *motion* for general (non-particle [25]) animation has yet to be adequately explored, but seems to provide a promising way of providing the ease of use promised by these techniques.

3 The Intelligent World Model

For a motion synthesis tool to be worth parallelising it is essential that it is computationally intensive enough to preclude widespread use in contemporary single processors. Those animation tools which fall into this category all share a common theme, they each employ a model of the virtual world more complicated than that required simply to encapsulate the geometry and appearance of its contents. The model must also be capable of retaining some of the *potential* for behaviour exhibited by the world, enabling the animation system to suggest suitable forms of motion. In this sense the world model is quasi-intelligent, as the model retains some internal consistency.

For example a conventional world model for a sphere might only require its location, radius and colour, whereas an intelligent world model might also need to know the spheres' mass, surface friction and properties of the substance of which it is constructed.

This information allows the animation tools built using this world model to respond realistically to the actions of the animator. The computational complexity of techniques using this model is due to the effort of maintaining consistency. This means that the expense of calculating the composition at any particular time is not predictable. Unfortunately this prevents the use of the parallelisation strategy which has proved so popular in distributed rendering, that of assigning different frames to different processors, and assembling the results to form the animation.

Broadly speaking the model of the world falls into one of two categories :

- A model of the world based on physical laws, which commonly employs some simulation process to synthesise the behaviour of bodies.

- A model of the world where the animator has strictly defined the potential behaviour of the world, often algorithmically. This body of work is commonly referred to as *behavioural animation*.

The potential for parallelism, and individual examples of it, exhibited by algorithms which employ each world model is the topic of the following two sections.

4 Distributed Dynamics For Animation

The simulation of dynamics, whether apparent to the animator or not, seems likely to be the most useful technique in the synthesis of pleasing motion. However, the degree to which an animator is aware of the simulation process is still a matter for debate among animation researchers, and so the potential for two different parallelism approaches are discussed in this section.

The most straightforward motion generation approach, at least from the researchers point of view, is that of pure forward dynamic simulation [6]. However, as previously mentioned, this is computationally expensive, and so is harder to use in animation synthesis than it should be. Consequently we first pay attention to parallelising this approach, for conventional rigid body dynamics, in section 4.1.

Such forward dynamics approaches though, provide control through forces, which most animators find unpleasant. Consequently higher level tools have been designed which create these forces for animators. Unfortunately these tools are not efficient, and so are not attracting the use they deserve. In section 4.2 we discuss how one of these approaches, spacetime control [34, 5], may be implemented in parallel.

4.1 Rigid Body Simulation

The production of motion from a description of bodies and forces is a straightforward process. The equations which describe a bodies response to the application of forces, at least for slightly simplified models of the world, are well known [6] and a wide variety of numerical techniques exist to enable computers to integrate through time [12], thereby obtaining movement.

Fundamentally all these techniques operate by discretising the continuous world model into a finite number of steps, at which point we determine the behaviour of the body during the next time step, account for collisions, determine the effects of gravity and so on. In common with most continuous processes though, this discretisation process requires considerable effort to ensure the simulation remains tolerably accurate, and so is computationally expensive. Whilst this is acceptable when we are running simulations to save the construction of models in engineering, it is problematic when we are using dynamics as a refinement process. The simulation of dynamics systems then, is an important target for parallelisation.

An initially appealing approach would be to assign the set of bodies we wish to simulate the behaviour of, to a collection of processors, where each body is

the responsibility of a single processor. Unfortunately the discretisation process which we employ to account for the continuous effects of the dynamic simulation causes problems when we wish to split the calculation across a number of processors. As the process is fundamentally a *synchronous* one, where the behaviour of bodies at each time step may be dependent on the behaviour of any portion of the remainder of the scene, a naïve implementation, in distributed memory machines, would require too much communication to be viable.

It is theoretically possible to use spatial coherence to reduce these effects, i.e., attempt to place adjacent bodies on the same processor, but the implementation of such a tactic would be critical, and it is difficult to see it scaling well with arbitrary scenes.

A more promising approach was advocated by Friedmann and Pentland [8], who noticed that the restriction on entirely synchronous operation would be relaxed to a certain degree, provided processors could guarantee to receive updates from other cells at an acceptable rate. Essentially they approximate the simulation process. For the simplified object representation they employ [23], this seems initially appealing, but the poor results they obtained (achieving only 50% efficiency on 5 processors) indicates this isn't a satisfactory solution on general platforms.

The idea, however, of relaxing the synchronous nature of the process is still likely to lead to better performance. In order to extract the maximum amount of concurrency from the problem we wish to limit the synchronisation process to the absolute minimum. Therefore we would like to determine, in advance, the synchronisation points.

Dworkin and Zeltzer [7] describe an interesting approach to this problem. For each body in the process, they determine (analytically) the collection of events at which collisions *may* occur. Using either simplified descriptions of dynamic movement, or regions of possible movement and hence uncertainty, they can reduce the need for collision detection. They describe this approach in a serial fashion, intending it for use in virtual reality simulations but, with some modification, it may be used profitably in a parallel environment.

A promising parallel approach operates as follows. Each body is assigned to a particular processor, where, if the number of bodies is too large, each processor takes more than one body. Each processor then determines, analytically, the events queue for each body, as shown in figure 2. This queue contains the sequence of possible collisions with either the environment or other bodies, and the path which the body would take if *none* of these collisions would occur. Essentially the events mark points of uncertainty, and we will process these to determine what course the animation takes. The location of these events may be obtained relatively easily by placing a sufficiently large, spherical bounding volume around the object, and only testing for volume/volume intersections [7]. Between these points we can perform a conventional local dynamic simulation process, without any synchronisation with other bodies in the scene, and in doing so build up the paths which will later be used in the animation playback.

Each processor then begins following the algorithm shown in figure 1. When the processor finds the first event on the queue it then tries to determine what

```
calculate events list for each body;
for each body
  find first event in queue;
  determine nature of event;
  switch(nature of event)
  {
    possible_collision:
      if(collision_occurs)
        determine response of colliding bodies;
        recalculate events list for the two bodies;
      remove this event from the list;
    stale_event:
      remove this event from the list;
    undetermined:
      do nothing;
  }
```

Figure 1: The parallel dynamic evolution process

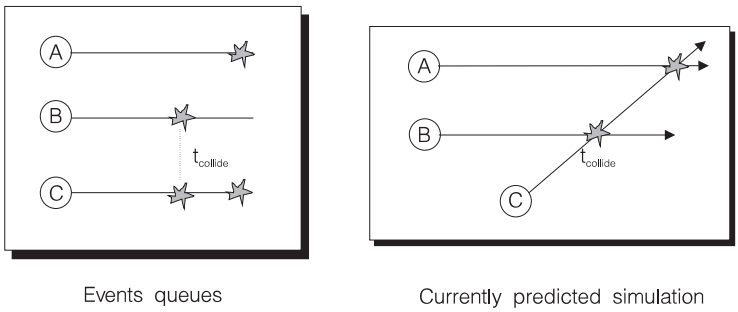

Figure 2: The events queue is shown in the left, with an indication of the possible movement on the right. Note that if C misses B, it may hit A.

further work needs to be performed. Much of this can be found by examining the events in each concerned (colliding) bodies' queue. There are 3 possible answers to this:

Unknown The event in one of the two event queues is not the first in that queue, and so we cannot determine whether the collision actually will occur. Therefore one of the processors must wait, or, more likely, continue processing with another body it is responsible for. In the example shown in Figure 2 this would occur with the processor responsible for body A at some point, as it is unable to determine whether the collision with C occurs until C's collision with B has been handled.

Both events exist In this case we must determine whether a collision actually occurs. How this is achieved varies depending on the object representation. If it does occur then we must re-calculate the events queue for each body, possibly finding new collisions, otherwise we simply remove the event from both queues.

One event exists This means either that the target has moved from its earlier path (and so this event can be labelled 'stale' and removed from the queue), *or* that the other object was unaware that a collision was going to occur, and so needs to be told.

This last point is worthy of further note. It is possible for events not directly concerned with one body to affect another, i.e., our analytic prediction may fail. In the example shown in Figure 2, body B may be deflected, after a collision with C at $t_{collide}$, into the path of A. In this event the colliding bodies (B) processor may need to update the targets event queue (A), i.e., add an event at the relevant point on the timeline. In order to avoid doing this unnecessarily it is sensible to actually perform an accurate collision detection. If the collision does not occur then A need not be informed, and B can continue to iterate down its event queue. However if the collision *does* occur then the path being synthesised for A must be modified.

This will not cause problems as the algorithm outlined is always looking for the *first* event in the queue. If a new event is added before events it had previously processed, it handles this without problems, simply deleting any work it may have done processing the path after that point. The algorithm, then, *evolves motion*, letting processors calculate as much movement as possible, and only requiring synchronisation at certain points. The process stops when *all* events queues are empty.

This parallel dynamics simulation, then, is reliant for its scalability, on

- The limited cost of two processors, each responsible for different bodies, determining whether their bodies collide. How this is achieved is dependent on the ratio of cost of finding the collision, against the cost of transferring the object descriptions between processors. If, for example, the objects are parametrically defined, the cost of communication is low, and so it may be worthwhile for only one processor to perform collision

detection, while the other begins processing the path which it would follow if the collision didn't take place, all the while waiting to be told if it did happen by its colleague.

- The sparse nature of the simulation, i.e., a limited number of collisions each of which is momentary, as we are reliant on a limited number of interactions in comparison to the length of simulation. Dworkin and Zeltzer [7] believe this to be suitable for most animation requirements. It is not necessarily a suitable parallelisation technique for *general* dynamics simulation for this reason.

It is difficult, then, to present theoretical speedup figures without implementation on a *particular* platform, but the approach does seem to promise substantial scalability, and demonstrates the potential for concurrency in this common animation technique.

4.2 Spacetime Control

As previously described, force based control is confusing and counter-intuitive for all but the most trivial of problems. Consequently animation researchers have turned their efforts to producing higher level interfaces to the dynamic simulation process, which creates the forces from indications of desired motion supplied by the animator.

A variety of interfaces have been proposed, and one of the most popular themes which has emerged is that the animation system should suggest forms of movement, which are plausible in the context of the world, to the animator. Although it is possible to apply AI techniques to this problem [17], a more successful tactic has been to use the minimisation of energy expenditure as a means of choosing desirable movement [34, 2, 5, 11].

The justification for this is the belief that much animal motion is motivated by reducing the effort required. If a goal can be reached easily one way, with little effort, why do things a hard way? If this is applied to animation the motion synthesis process can be likened to a multi-dimensional energy minimisation process.

This approach can be described by the application of a statespace paradigm. Here the information required to represent the virtual world is collected into a space, with each dimension accounting for a single variable in the world. For example a single rigid body could be minimally described with a 6 dimensional statespace, with 3 dimensions for its Cartesian position and a further 3 for an Euler representation of rotation.

$$(x, y, z, rx, ry, rz) \tag{1}$$

If we add an extra dimension to account for time, t, then the animation production process is the generation of a trajectory through this statespace. If we can associate energy with a particular trajectory, then the spacetime minimisation process is one of finding the cheapest, or lowest energy, trajectory which satisfies our goals.

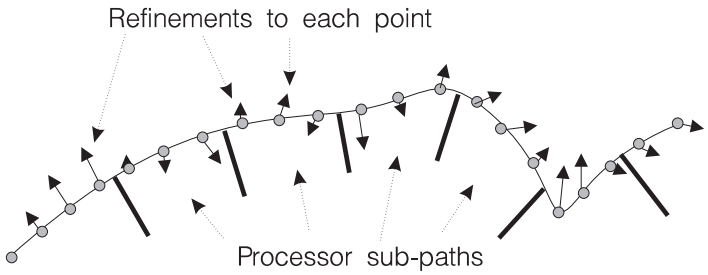

Figure 3: The path is refined in statespace, with each sub-path the responsibility of a single processor.

Brotman and Netravali [2] used a simplified form of this minimisation, which was forced to interpolate certain key points, with the intervening trajectories free to minimisation. Witkin & Kass [34] described how this can be generalised by letting the animator describe an initial path, as a number of points, which is the focus for perturbational minimisation, using existing numerical techniques [10], subject to a number of animator described functions.

This implementation of the technique suffers from several problems, and effort has been expended on improving the ease of specifying the functions [15] and optimising the minimisation process by advanced spline representations [5, 19]. However the minimisation process is still slow for even simplified 2D cases, so has yet to attract commercial use. Consequently the spacetime minimisation process is an important target for parallelisation, and due to its potential, is likely to attract widespread use if it can be achieved successfully.

A potential strategy for parallelisation operates as follows. The path is split into a number of sub-paths, as shown in figure 3, with each being assigned to a processor. Each processor is sent information detailing the goals which we are minimising the path by. Then, as described by Witkin & Kass [34], the path is split into a number of points, which we can interpolate a spline through to regenerate the path.

Each point on each processor is then iteratively minimised by one step, with it moving to minimise the energy expended at that step, limited by the necessity to maintain the constraints. The energy 'at each point' is the energy required to reach that position in statespace from the previous point on the path. This means that at boundary points, where the previous point is being minimised on another processor, we must take care to ensure we have an up-to-date record of where that position is.

Fortunately the technique is fundamentally a *local minimisation* process, where the positions of points within a sub-path is only marginally effected by points in other sub-paths, the global properties of the minimisation only emerging after many iterations. Therefore we can minimise each sub-path concurrently for some time, only performing a global synchronisation at certain points in the process.

These synchronisations can be performed by the first processor initiating

an update on its next colleague after a certain number of iterations. Each subsequent colleague must receive this new end-point, and re-process its own sub-path to determine how this effects the energy required to traverse it. After this the processor passes on the new information to *its* colleague. The updates to the path travel down the processor line in waves. This parallel computation stops when the magnitude of modifications made to each sub-path fall below a certain limit, i.e., we have converged to a minima.

This skeleton algorithm demonstrates the potential for concurrency in the spacetime minimisation process, and indicates that parallelism may be useful in the commercialisation of the technique. Note that, unlike the dynamics techniques described previously in this section, spacetime techniques are a highly active research topic (as we will see in the following section). Consequently the parallelisation of this particular technique does not necessarily constitute the final form that spacetime techniques will be used in [24], but indicates that it can form a part in future spacetime tools.

5 Parallelising Behavioural Animation

The production of motion through a higher level interface should allow an animator to let the animation system help him or her. At the highest level of control the animator would like to function as a director, ordering autonomous entities to perform motion, while leaving much of the detail of *how* the motion is achieved to the actor itself.

Behavioural animation systems achieve this effect by letting the animator construct algorithms which describe how the actor achieves certain goals, for example an animator may instruct a mannequin model in how it balances on a unicycle, perhaps using a programming language to express these instructions. Once an appropriate set of goal-achievement behaviours have been described, the animator constructs motion by activating the relevant set of behaviours at the appropriate time to cause movement. Providing the behaviours are expressed correctly, and our higher level sequence of goals is suitable, we should achieve the desired animation.

It is important to emphasize that behavioural animation often employs the physically based world models we discussed in the previous section. For example, our unicycle riding mannequin may employ a mass description of its component bodies, a friction model for how the cycles wheel moves across the ground, and generate forces which it applies to its bodies to cause balancing. The reason behavioural animation is seen as distinct from physics models is that the behavioural model of the world is said to be *active*.

Hodgins [14] described these models as active because they introduce forces into the simulation process. Pure physical world models, in contrast, only have forces added to them as a direct result of an animators interference, so they are said to be passive.

From the animators point of view the use of behavioural animation is a two step process; he or she must

- Develop behaviours which describe an actors response to a collection of stimuli. For example, our unicycle balance behaviour may react to motion away from a vertical 'stable' position, and so its stimuli may be the movement of the centre of mass. These behaviours, however we choose to express them, are fundamentally algorithms. Then, given these behaviours they,

- Perform a simulation process, where at each time step the behaviours are activated to see how they respond to the world.

Behavioural animation is an immensely powerful animation tool, as, in theory, if used with a sufficiently rich collection of behaviours, it provides animators with an extremely high level interface to the virtual world. However, both its principal steps are often regarded as computationally intensive, and are therefore targets for parallelisation. As the simulation of such worlds is closely connected with the efficient implementation of motion environments we will discuss this area further in section 6, and concentrate now on how parallelisation may be used to speed up the behaviour *production* process.

5.1 Constructing Behaviours

The construction of behaviours, or as they are sometimes called, *controllers*, is a time consuming and involved process. If the controller is poor, either because it is dependent on the wrong stimuli or produces unsuitable effects, it can never be used to generate acceptable animation. As a result of this controller construction is regarded as an extremely skilled task.

The relationship between cause and effect, which the controller must embody, is extremely difficult to ascertain in any serious example. For example, why does a cat stalk its prey?, is it responding to the stimuli of its prey noticing it on previous occasions, or is there some more involved reason? As a result of this most controllers concentrate on *extremely* simple systems, where there is a clear relationship between cause and result.

Miller [20], for example, describes a control system designed to move a snake across unpredictable terrain. More complicated examples, such as human locomotion, must simplify the problem, and conventionally function as higher level entities, such as Bruderlin & Calverts [3] gait controller, leaving lower level motion to more simple motion control techniques. Some research has been conducted in simplifying this hand-coding process [29, 28, 27], using generic forms of controllers, which the user can customise to produce the right effect.

In recognition of this, there is a growing body of work which attempts to *automate* the controller synthesis process. We have already discussed (section 2) Ngo and Marks genetic approach to this problem [21], using a parallel Connection Machine 2 to evaluate a large number of controllers, the desirable components of which are combined to produce ever more advanced behaviours. Addressing the same problem Van De Panne and Fiume [31] express their controllers in a manner reminiscent of Sun and Green [29], using a network of actuators, hidden nodes and sensors. The creation of these controllers involves

finding the optimal network, and ideal weightings, so for example, an actuator takes the correct amount of notice of each sensor.

They propose the use of a program to evaluate the effects of large numbers of different types of networks and weightings to produce the correct results. Each network/weighting combination is evaluated against a specified metric, for example a balancing controller might be measured against the amount of energy expended, and how close it remains to the optimal position. This is, of course, an ideal candidate for parallelisation, and, unlike Ngo and Marks approach [21], we are not reliant on large quantities of communication between generation of controllers to synthesise a desirable result.

Such an approach would split the space of controllers/weightings which the user wishes to evaluate, and evaluate different portions of these in parallel. Each slave processor need only return to its master the local optimal solution found, and need provide no indications of the form of controller it is considering to its slave colleagues, thereby reducing costly communication.

This provides an extremely promising motion controller synthesis technique, as it ought to be scalable in most parallel environments, and seems to provide a useful tool. However, the significant remaining problem with such automatically produced controllers is the mechanism of control an animator has over the result.

The techniques presented so far in this section can only choose between controllers using an arbitrary metric, such as how far an actor will move in a time period. It is impossible to express, in algorithmic terms, that the motion should look 'nice'. Consequently such techniques have an inbuilt disadvantage.

In response to this the most promising control technique from the pure passive physical models, the spacetime control methodology [34, 15, 5], has been applied to controller construction. Van de Panne *et. al.* [32] describe the construction of a controller which achieves a particular orientation in statespace, in an optimal manner, for a particular actor. In this way the aesthetically pleasing qualities of the motion will be produced in response to the search for an energy-optimal controller. Such a controller requires still greater effort to construct, and so is of interest to us.

An animator constructs a spacetime controller by expressing a desired position, and a volume in statespace which the controller encompasses. When a controller is activated, for a body which falls in this volume, it will cause the body to reach the destination in an optimal manner, as shown in Figure 4. The controller achieves this by having previously determined, for a discretised form of the volume, the forces which drive the actor to the destination. The evaluation process, which we will discuss in more detail in the next section, is simply one of examining a table for the current position, which will tell us which forces the controller introduces.

The principal problem with this approach is the extreme expense of evaluating the search space to find the optimal forces. However, just like the sensor actuator approach [31], it is a straightforward process to efficiently parallelise. The controller synthesis process is performed as follows:

We begin by considering a small area around the destination, which is dis-

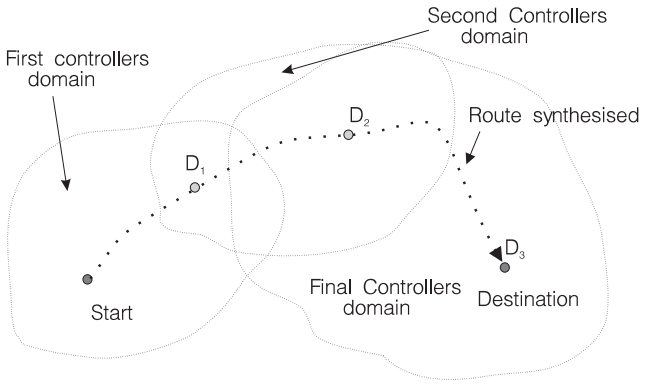

Figure 4: A slice in statespace, with controllers' regions marked. Provided the destination for one is within the bounds of another, they may be combined to cause motion.

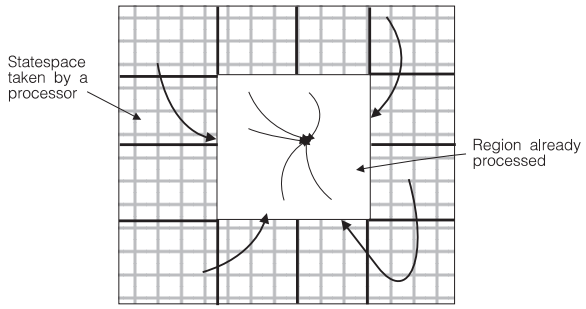

Figure 5: A larger portion of the space, encompassing the already processed region.

cretised into a finite number of points in each dimension. For each point we perform a local optimisation to determine the forces which bring a body *at that* point to the destination, D, which is found using

$$f_{opt}(P_{SD}) = t_{P_{SD}} + \int_0^{t_{P_{SD}}} KT(t)dt \tag{2}$$

where P_{SD} is the path from an initial state S to the destination D, and $t_{P_{SD}}$ is the time taken to traverse this path. K is a user defined control which specifies whether we wish to concentrate on fast or cheap movement, and $T(t)$ represents the control values. This can be performed by evaluating for different values of $T(t)$, which we may perform in parallel for high dimensional spaces.

Then, having produced an initial controller for a small volume, we increase the volume we consider, and continue to process each discretised point. However we now only need to produce forces which take the body to the inner volume,

as we already know the forces which will drive it to the destination once in there.

In order to parallelise this more expensive step, we split the larger volume into a number of convenient units, each of which is assigned to a particular processor, as shown in Figure 5. Each processor may evaluate each point which falls into its region, again performing the process shown in equation 2, which may be evaluated in parallel if we are operating on a machine which encourages such fine grain concurrency. The size of the volume is repeatedly increased until we have evaluated all the space which the controller will be used for.

This controller synthesis process will scale well, as, yet again, there is little need for inter-processor communication, and the space can be split into an arbitrary number of units, so we are not restricted to the degree of parallelism we can exploit.

The synthesis of motion controllers presents an ideal target for parallelisation, and, as the emphasis is on constructing *reusable* controllers animators do not require real-time operation. This means the ability to improve synthesis time by increasing the number of processors should be desirable.

6 Machine Initiated Motion

So far the focus of discussion has been on the efficient parallel implementation of existing animation techniques. However, as previously described, the definition of computer animation is being broadened to encapsulate the simulation of an *interactive* world. For this world to behave plausibly it is important that it responds quickly, and yet reacts in a realistic manner. Consequently it is an area well suited to the application of parallel implementations of animation techniques.

The previous section has described some promising techniques to improve the speed and ease by which high level behavioural controllers may be synthesised for later use. However, as previously noted, some of these controllers must be expressed algorithmically, and occasionally significant amounts of processing must be performed during the playback.

In a machine initiated motion environment we are primarily interested in using these controllers in a near-interactive setting, and so this delay is likely to restrict the uses to which animation may be put. Consequently the use, as well as the synthesis, of these controllers is an important target for parallelisation.

We could achieve this in one of two ways:

- We could encode the processing performed in each controller in a manner suitable for parallel implementation, or alternatively,
- we might execute each controller independently.

The first tactic is of course tempting, but is unlikely to be extremely helpful. Most of the behavioural controllers techniques have, as described previously, concentrated on very simple problems. Consequently the actual processing performed during the evaluation of each controller, at each time step, is not

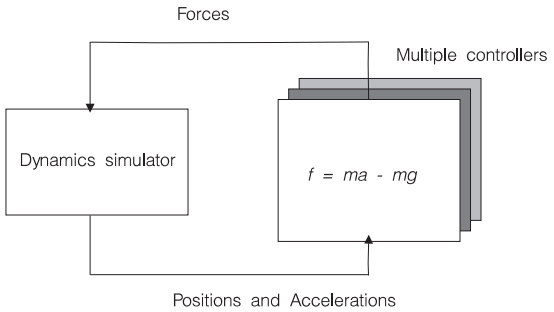

Figure 6: Simplified flow with loosely coupled controllers. This loop is traversed for each Δt

sufficiently large to warrant parallelisation. However, given the possible future adoption of parallelism in motion synthesis, it may be that this will become necessary. The more promising approach at the moment, then, is that we evaluate the response of multiple controllers independently, in parallel.

Kearney [16] identifies two distinct ways of viewing the controller evaluation process, loose or tight coupling. With loose coupling each controller produces forces, which are added to the Newtonian simulation process between time steps. Tightly coupled systems, in comparison, introduce constraints which the simulation process must take into account internally. Loosely coupled systems tend to encourage development of controllers as portable models, but have a tendency to have to perform greater processing internally, as they must maintain constraints themselves, and pass the constraint forces to the simulator explicitly.

From our point of view the loosely coupled system is likely to provide greater scope for parallelism, but both systems use controllers which may be evaluated separately, and so are capable of supporting distributed computation.

A parallel implementation of a behavioural simulator may operate as follows. The world is initially described as a force-body abstraction suitable for Newtonian simulation. The simulation is then started, possibly with some initial processing to initialise the state.

An initial time step is performed, which should produce, for each body, a new position, velocity and so on. Then, before the next simulation step, we determine which controller, out of the collection of those available, is applicable to each body. This cannot be statically determined, as most systems will require the ability to add and remove controllers from the cycle.

Once we have found the set of active controllers, we pass the position and accelerations of the relevant bodies to those controllers. However each controller is physically executing on remote processors, and so the speed at which we perform this communication is important. Each controller performs some local processing, and then returns a set of forces to the simulation process, again using message passing communication techniques. This cycle, as shown in figure 6, is performed for each timestep the simulator takes.

This form of parallelisation is essentially allowing us to use greater degrees of control in animation systems, and allows us to accommodate complicated scenes. Previously animators have been restricted to extremely small scenes, as otherwise the computational expense forced the use of batch processing.

Machine initiated motion environments are not restricted to using controller based animation production, and may integrate more advanced techniques, such as controllers which perform a continuous body dynamics simulation in order to find their response to the actions of a user. However, the fundamental approach of splitting such 'agents' onto different processors, which communicate to agree on the system state, is still likely to be helpful.

7 Conclusions

The field of computer animation is a large consumer of processing power, but has yet to adopt parallelism as a solution to its principal remaining problem : the ease of motion control for non-skilled animators. This paper has described some of the most significant families of motion generation tools, and highlighted areas of concurrency which may be exploited in a parallel implementation.

In particular it has described

- A framework for the parallelisation of *sparse* dynamic simulation for animation, which attempts to extract the maximum concurrency by prediction of points of likely synchronisation.
- A basic implementation of a parallel spacetime minimisation process.
- Techniques for parallelising leading controller generation procedures.
- A tactic for distributed simulation of interactive worlds, where the computation caused in the movement of each body is the responsibility of different processing elements.

Previously the use of parallelism in the synthesis of motion has not been widely explored, and it is hoped that the topics highlighted in this paper will lead to a further investigation of this area.

Acknowledgments

Thanks to the students and staff of Manchesters' Computer Science Department and MCC's Computer Graphics Unit for all their support & ideas. A special vote of thanks goes to Andy Grant for encouraging my work on this paper. Some of this work was undertaken while the author was an EPSRC research student so thanks also to my supervisors, Prof. F. Sumner and W.T. Hewitt.

References

[1] William W. Armstrong, Mark Green, and Robert Lake. Near Real-Time Control of Human Figure Models. *IEEE Computer Graphics & Applications*, 7:39–51, June 1987.

[2] Lynne Shapiro Brotman and Arun N. Netravali. Motion Interpolation by Optimal Control. *Computer Graphics*, 22(4):309–315, August 1988.

[3] Armin Bruderlin and Thomas W. Calvert. Goal-Directed Animation of Human Walking. *Computer Graphics*, 23(3):233–242, 1989.

[4] N. Burtnyk and M. Wein. Computer–Generated Key-Frame Animation. *Journal of the Society of Motion Picture and Television Engineers*, 80:149–153, 1971.

[5] Michael F. Cohen. Interactive Spacetime Control for animation. *Computer Graphics*, 26(2):293–302, July 1992.

[6] John J. Craig. *Introduction to Robotics : Mechanics and Control.* Addison–Wesley, Second edition, 1989.

[7] Paul Dworkin and David Zeltzer. A New Model for Efficient Dynamic Simulation. In *4^{th} Eurographics workshop on Animation & Simulation*, 1993.

[8] Martin Friedmann and Alex Pentland. Distributed Physical Animation. In *3^{rd} Eurographics workshop on Animation & Simulation*, 1992.

[9] Marie-Paule Gascuel, Anne Verroust, and Claude Puech. Animation with collisions of deformable articulated bodies. In *1^{st} Eurographics workshop on Animation & Simulation*, 1990.

[10] Phillip Gill, Walter Murray, and Margret Wright. *Practical Optimisation.* Academic Press, 1981.

[11] Michael Girard. Constrained Optimization of Articulated Animal Movement in Computer Animation. In *Making Them Move : Mechanics, Control and Animation of Articulated Figures.* Morgan Kaufmann, 1991.

[12] Mark Green. Using Dynamics in Computer Animation: Control and Solution Issues. In *Making Them Move : Mechanics, Control and Animation of Articulated Figures.* Morgan Kaufmann, 1991.

[13] Mark Green and W.W. Armstrong. Interactive Animation Responding to the Real World. In *4^{th} Eurographics workshop on Animation & Simulation*, 1993.

[14] Jessica K. Hodgins, Paula K. Sweeney, and David G. Lawrence. Generating Natural-looking Motion for Computer Animation. In *Proceedings of Graphics Interface '92.* Canadian Human-Computer Communications Society, 1992.

[15] Michael Kass. Inverse Problems in Computer Graphics. In Nadia Magnenat Thalmann and Daniel Thalmann, editors, *Creating and Animating the Virtual World*, 1992.

[16] Joseph K. Kearney, Stuart Hansen, and James F. Cremer. Programming Mechanical Simulations. In *2^{nd} Eurographics workshop on Animation & Simulation*, 1991.

[17] Richard Lee. Artificial Intelligence in Computer Animation. Master's thesis, University of Manchester, In Preparation.

[18] Ming C. Lin and Dinesh Manocha. Interference Detection Between Curved Objects for Computer Animation. In Nadia Magnenat Thalmann and Daniel Thalmann, editors, *Models and Techniques in Computer Animation*. Springer–Verlag, 1993.

[19] Zicheng Liu, Steven J. Gortler, and Michael F. Cohen. Hierarchical Spacetime Control. In *Proceedings of SIGGRAPH '94*, pages 35–42, 1994. In *Computer Graphics* proceedings, Annual Conference Series.

[20] Gavin S. P. Miller. The Motion Dynamics of Snakes and Worms. *Computer Graphics*, 22(4):169–178, August 1988.

[21] J. Thomas Ngo and Joe Marks. Spacetime Constraints Revisited. In *Proceedings of SIGGRAPH '93*, pages 343–350, 1993. In *Computer Graphics* proceedings, Annual Conference Series.

[22] Mark Owen and Philip Willis. Modelling and Interpolating Cartoon Characters. In *Computer Animation '94*, 1994.

[23] Alex Pentland and John Williams. Good Vibrations : Modal Dynamics for Graphics and Animation. *Computer Graphics*, 23(3):215–222, 1989.

[24] Martin Preston. Parallel Spacetime Animation. Paper submitted for publication.

[25] Karl Sims. Particle Animation and Rendering Using Data Parallel Animation. *Computer Graphics*, 24(4):405–413, August 1990.

[26] Karl Sims. Evolving Virtual Creatures. In *Proceedings of SIGGRAPH '94*, pages 15–22, 1994. In Computer Graphics proceedings, Annual Conference Series.

[27] A. James Stewart and James F. Cremer. Animation of 3D Human Locomotion: Climbing Stairs and Descending Stairs. In *3^{rd} Eurographics workshop on Animation & Simulation*, 1992.

[28] A. James Stewart and James F. Cremer. Beyond Keyframing : An Algorithmic Approach to Animation. In *Proceedings of Graphics Interface '92*. Canadian Human-Computer Communications Society, 1992.

[29] Hanqiu Sun and Mark Green. The Use of Relations for Motion Control in an Environment With Multiple Moving Objects. In *Proceedings of Graphics Interface '93*. Canadian Human-Computer Communications Society, 1993.

[30] Nadia Magnenat Thalmann and Daniel Thalmann. *Computer Animation, Theory and Practice*. Springer-Verlag, Second edition, 1990.

[31] Michiel van de Panne and Eugene Fiume. Sensor-Actuator Networks. In *Proceedings of SIGGRAPH '93*, pages 335–342, 1993. In *Computer Graphics* proceedings, Annual Conference Series.

[32] Michiel van de Panne, Eugene Fiume, and Zvonko Vranesic. Reusable Motion Synthesis Using State-Space Controllers. *Computer Graphics*, 24(4):225–234, August 1990.

[33] Jane Wilhelms. Using Dynamic Analysis for Realistic Animation of Articulated Bodies. *IEEE Computer Graphics & Applications*, 7(6):12–27, June 1987.

[34] Andrew Witkin and Michael Kass. Spacetime Constraints. *Computer Graphics*, 22(4):159–168, August 1988.