Toward Virtual Actors

Maciej Kalisiak
Department of Computer Science
University of Toronto

1 Introduction

Computer graphics (CG) and animation are reaching previously unparalleled levels. They are now virtually ubiquitous in most visual media; so much so that the general audience is starting to develop a blasé attitude towards them. For example, nowadays only the most extensive computer graphics and animation efforts in movies manage to draw any admiration from the viewer. Computer animation in TV ads and commercials is completely taken for granted. The saturation has reached such levels that even computer games cannot rely anymore on technical advances in graphics and animation to sell the product, something that they have been notorious for in the past. Rather than mark the end of interest in the domain, this ubiquity only serves to highlight the underlying immense demand for graphics and animation.

Over time, a plethora of animation techniques have been proposed, each suited to some particular task. Some of the general approaches taken include global and local motion planning, global and local behavioral modeling, physical simulation, optimal control, motion capture, and puppetry. This abundance of method stems from each application imposing different requirements and constraints on the animation algorithm, and being open to different trade-offs. A case in point is the predominant area for computer animation, the entertainment industry. Movies often make use of CG for animating various digital stuntmen or chimerical beasts, and ever more frequently, for the entire productions themselves. Here, great importance is placed on the most minute details and realism of motion and appearance, while rendering time is of little consequence; CG-rendered cartoon features often further trade-in realism for "cartoon physics" and stylized over-exaggerated motions. Video games, on the other hand, usually require real-time animation at the expense of detail. Simulators, such as military trainers or sports games, present an even greater challenge as both, real-time animation and realism are a must. More specialized applications, like sport prototyping tools (motion feasibility and analysis for gymnastics, diving, etc) and choreography tools, pose yet another set of demands.

One of the key objects being animated are human or human-like characters. Whether it is the narcissist in us, or that we find human interaction very compelling and fertile ground for storytelling and drama, natural-looking human animation has been a long-sought goal. Although our present animation methods can indeed produce some impressive results, they require an immense amount of time and effort, and very often have the animator working at very low levels of abstraction, such as directly specifying limb position and orientation. This is dictated by the animator's tools. A much more natural approach would have the animator *direct* his characters, much like a movie director directs his cast of real life actors. Ideally, one would like to create animations by working at task-level: "pickup skull, slowly; look at it; wander to the chair and sit down heavily; gaze thoughtfully at skull, and recite 'To be, or not to be...' " What we need, in short, are *virtual actors*. Although it's not the panacea of computer animation, there is reason to believe that there would be a very large demand for this concept. The advent of virtual actors would make computer animation accessible to a much larger user base, bringing it to ordinary people, ones without any particular animation skills. Veteran animators would also derive benefit, as they could then concentrate on expressing their vision rather than having to contend with technical issues and limitations. Conventional movie directors could even start conducting these virtual actors directly, without any intermediation of animators. The development of such highly autonomous character animation methods thus seems like a useful research direction.

Any implementation of a virtual actor must confront three key areas: the directing of the character, motion planning, and animation. The character direction problem addresses the question of how to communicate

the animator's intentions to the virtual character. Motion planning, often also called path planning, concerns itself with working out the body logistics in the global context, such as planning a path between trees in a forest, and limb logistics in a more local context, such as planning hand motion to grab an awkwardly placed object. Animation then attempts to implement these plans in a natural looking way. More often than not, there are no sharp boundaries between these areas; a lot of character animation algorithms incorporate all three areas to various degrees.

Our current understanding and ability in these areas is insufficient to be able to implement such general virtual actors. Motion planning is arguably the area that requires the least of new research towards that goal. We can plan paths for simple robots in most environments, although full human-like objects are still unmanageable due to their complexity and the "curse of dimensionality". Fortunately, most of the time one can approximate the human model with a simpler object, one for which path planning solutions are easily obtainable. Similarly, in the animation field we have a large variety of methods to animate simple characters, but again, more complex characters are outside our grasp, at least in the context of virtual actors. Animation remains very time consuming, requiring a lot of work from highly skilled animators.

Current methods for directing a character are very much dictated by the animation method used. These methods often provide only a particular specialized interface for specifying required motion, and it is up to the user to translate his or her intent to this representation. This is often difficult, time-consuming, and frequently requires a certain amount of skill or intimate knowledge of the system being used. The situation is further exacerbated by a prevailing trade-off of *control vs. automation*. That is, most animation techniques either allow great flexibility in motion specification at the cost of requiring much user input, or vice versa, are highly autonomous but offer little control over the generated motion. Either alternative is a thorn in the user's side. This suggests that animation techniques are not mature enough yet to make discussing the directing of virtual actors practical, and hence will not be covered in any great length in this paper.

1.1 Scope & overview

This paper will concern itself mostly with providing an exposition of the current state of animation techniques, and to a lesser extent, of motion planning. Furthermore, it will only look at the more interesting case of *active* objects, ones which exert internal forces and torques in an attempt to control their motion, objects usually seen as having sentience or a will of their own. The focus will be on humanoid characters. The paper will also explore some open problems in these two fields and potential research directions.

The layout of the paper is as follows. In the remainder of this section some essential fundamental concepts and notation are introduced. This is followed by a section reviewing motion planning, discussing first the more classic deterministic methods, and then the more recent stochastic and kinodynamic variants. Similarly, animation methods are reviewed next, using the traditional categorizations. We then discuss "motion graphs", a recent animation technique that shows much potential. The paper ends with a look at some open problems in all these areas.

1.2 Fundamental concepts & notation

This section reviews some fundamental concepts essential to motion planning and animation, as well as the notation used throughout this paper. Alternate notations found in related literature are also mentioned, if common enough.

We adopt the usual variable notation:

x	scalar
$\mathbf{x}$	vector
$\dot{x}$	first time-derivative
$\ddot{x}$	second time-derivative

At the center of our attention is the *character*, or *model*, to be animated. It is modeled as an articulated tree, or *skeleton*, of rigid links that captures the overall dimensions and mobility of the subject. The links are attached together using various types of joints (prismatic, hinge, etc.), each usually with constraints on the range of motion. The links usually bear resemblance to the body part or segment which they approximate, although often they just serve as a skeletal structure to which further geometry can be attached (e.g., geometry representing muscles), or over which skin and cloth may be draped. Figure 1 illustrates a simple humanoid model.

Although real life motion is continuous with respect to time, its rendering on display devices, such as a computer or movie screen, is achieved by uniformly sampling the motion and displaying the sampled snapshots in rapid succession. In each such snapshot the character is in a certain *configuration*. This configuration, usually denoted by $\mathbf{q}$, is a vector of scalars, each one the current value of one of the character's *degrees of freedom* (DOF). For a human character the configuration usually consists of his or her coordinates in the world, the overall body orientation, and all the joint angles.

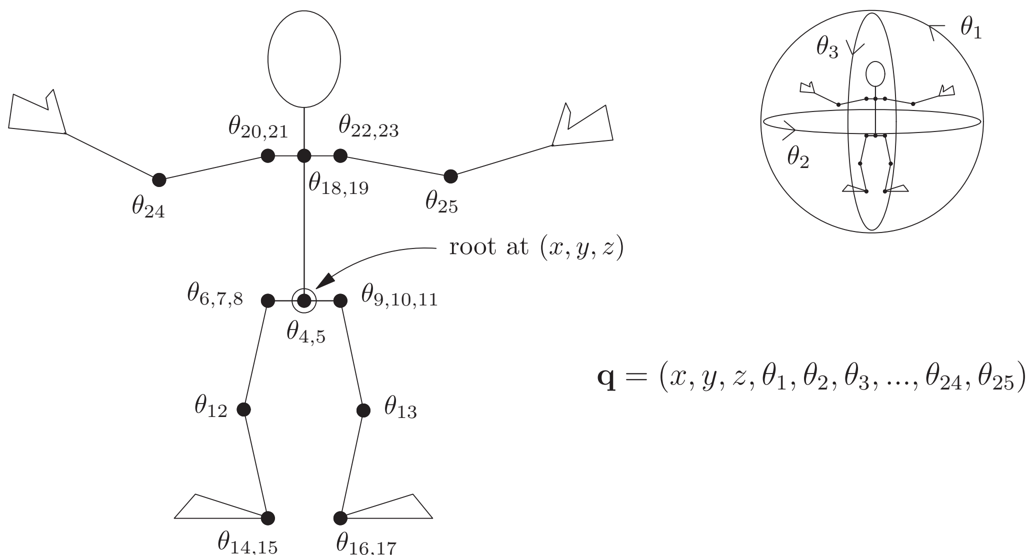


Figure 1: A basic humanoid character model, and one particular parametrization of its configuration $\mathbf{q}$.

All the possible values of $\mathbf{q}$ for a given character form an n dimensional space, where n is the length of the $\mathbf{q}$ vector, or alternatively the number of DOFs of the figure. This is the character's *configuration space*, denoted by $\mathcal{C}$, where $\mathcal{C} \subseteq \mathbb{R}^n$. A particular configuration of the character is a single point in this space, one whose coordinates are given by the values within the vector. The character's motion, on the other hand, traces out a curve in $\mathcal{C}$. The subset of configuration space in which the character is free of collision with any obstacles is termed the *freespace*, and indicated by $\mathcal{C}_{free}$. One can also talk of the configuration space obstacles, or *C-obstacles*. In general, a C-obstacle is a fattened version of its real world counterpart, obtained by convoluting the shape of the obstacle with that of the character (Minkowski sum). The union of all the C-obstacles is referred to as $\mathcal{CB}$. Thus $\mathcal{C} = \mathcal{C}_{free} + \mathcal{CB}$.

A brief example might serve well here to clarify and cement the concepts so far. Consider a character that needs to cross a warehouse full of crates scattered on the floor. Here the problem can be simplified by representing the character by a bounding box or oval, and planning in 2D using the bird's eye view. Note that any curve in $\mathcal{C}_{free}$ that connects $\mathbf{q}_{init}$ to $\mathbf{q}_{goal}$ is a motion that brings the character from the initial to the final configuration *while avoiding obstacles*. The general idea behind most motion planning algorithms is finding such a curve. Figure 2 on page 5 further visually illustrates these concepts, using a different simple example.

Although a configuration completely specifies the character's pose, it lacks any temporal information. Much like in a photograph, there is no way to tell if the character is moving, and if so, how and with what velocities. For applications that require such information one usually works with the character's *state*. The state augments the character's configuration with the first time-derivatives of all the DOFs. $\mathbf{x}$ or X are sometimes used to represent state variables. There is also a space analogous to $\mathcal{C}$ for states, which is understandably called the *state-space*, or less frequently the *phase-space*. In various work it is denoted by $\mathcal{X}$, $\mathbf{TC}$, or the abbreviation SS. Again, the character's current state is a point in this space, while a motion forms a curve through it. Curves in $\mathcal{C}$ are termed *paths*, while curves in $\mathcal{X}$ are termed *trajectories*. Path variables tend to use $\mathbf{p}$ while trajectories often use τ. This concept of state is very important, and is used heavily in animation.

2 Path planning

Path and motion planning problems were originally first addressed in the robotics field. The problems studied can be roughly divided into two types: that of freely moving bodies, and that of tethered or fixed robots. The free motion problem is best characterized by the "piano mover's problem": given a piano that has been unloaded at the curb, how best to move and maneuver it to its final destination inside the house? The problem is non-trivial as often a particular sequence of rotations is required to clear tight corners and narrow doorways. The fixed robot problem, on the other hand, usually deals with moving a link chain that is anchored at the base. The problem was motived by the large demand for motion planning of robotic manipulator arms in industry. This heritage gives rise to a robotics-biased terminology, such as "workspace", the environment in which planning is to occur, presumably derived from the workspace of a manipulator arm. This paper will discuss motion planning mostly in the original robotics context, but it should be understood that the concepts apply equally well to animation. In particular, whenever reference is made to the robot, the animated character can be envisioned instead.

Motion planning in computer animation is generally used to solve the same types of problems as in robotics. High-level, global context path planning is used for working out collision-free paths for characters, while low-level, local context motion planning is used for limb movements, especially in object manipulation scenarios (e.g., oil filter extraction from a car's innards under the hood). It has also been applied to character motion planning and animation in highly-constrained environments [Kal99, KvdP00, KvdP01]. This work addresses a problem that differs from the traditional piano mover's problem in that the character needs to be in constant contact with the environment (viz., terrain under foot or overhead), while freespace planning usually tries to keep as large an obstacle clearance as possible.

As mentioned in the introduction, the motion planner's task is to find a connecting path in $\mathcal{C}_{free}$ between q_{init} and q_{goal}. More formally: given a representation $\mathcal{A}$ of the robot, its workspace $\mathcal{W}$, as well as its starting and goal configuration, denoted by $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$, find a connecting path $\mathbf{p}$ through the freespace $\mathcal{C}_{free}$, which corresponds to a collision-free motion for the robot in $\mathcal{W}$. The path optionally is required to respect or optimize a number of hard or soft constraints, respectively. Some frequent constraints are limits on speed, acceleration, and applied torque. Soft constraints often include minimization of expended energy or time elapsed. A trivial path planning example is illustrated in figure 2.

It should be noted that $\mathcal{A}$ also often refers to an object being manipulated by a robot, instead of the robot itself. For example, consider again the piano mover's problem: the piano is a passive object, yet it is more convenient to plot the piano's motion and only then work out what movements the mover's themselves should perform.

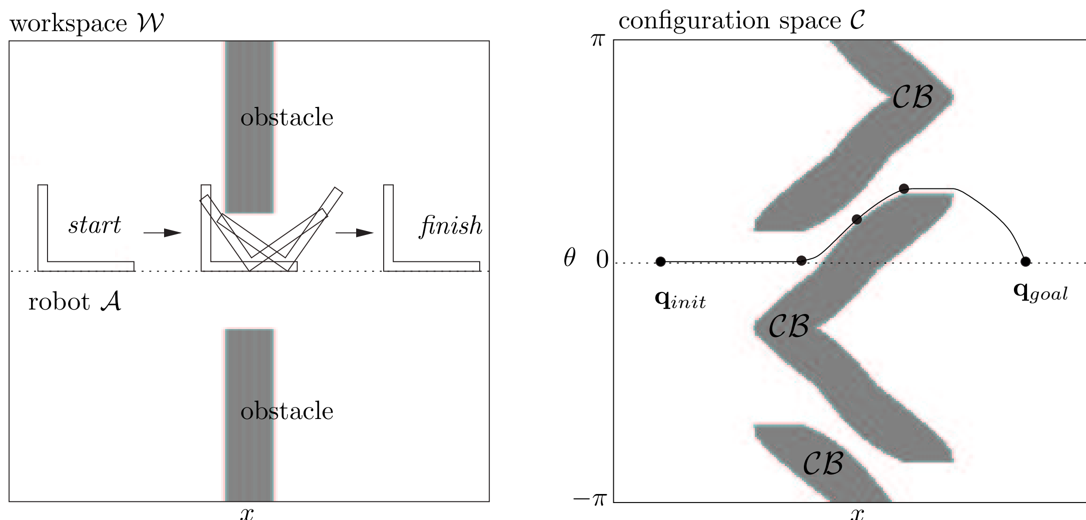


Figure 2: trivial path planning example; consider the L-shaped robot which is only capable of motion along the x axis and rotation. The workspace on the left illustrates one particular solution; on the right, the same solution is shown in the corresponding configuration space. Three intermediate configurations of the solution are marked in either view.

2.1 Types of planners

Much like in animation, the variety of motion planning problems has given rise to a collection of solution methods. Some common distinctions between planning problems are: fixed vs. mobile robots (as we've already seen), static vs. dynamic environments, single- vs. multi-query planners, and omniscient vs. exploring robots. Dynamic environments contain moving obstacles, which substantially complicates planning by implicating time in the calculations. Static environments, with all obstacles immobile, often lend themselves to multi-query planners, where some potentially costly precomputation can be performed for the sake of answering subsequent queries quicker. Dynamic environments, in contrast, usually lead to single-query planners, since a change in obstacle positions often invalidates any precomputations. Robotic arm manipulators are usually assumed to be omniscient about their workspace, while mobile robots in the real world do not have such well controlled environments and must discover them on their own. This severely limits the potential for use of global methods in planning.

A more recent distinction is that of *kinematic* vs. *kinodynamic* planners. Originally, the motion planning problem was viewed in a purely geometric manner. In finding a connecting path, and the attendant rotations, the planner was concerned only about not intersecting the geometry of $\mathcal{A}$ with that of the obstacles. In the context of real world scenarios, this corresponds to a robot or object that is capable of accelerating in any direction, regardless of its current orientation, and that is not under the influence of external forces, such as gravity. Many applications can be made to fit this pigeonhole by a number of simplifications (e.g., by having the robot perform the motion slowly, we can negate the effect of inertia; by being confined to planar motion on the floor, we can ignore gravity). For example, a holonomic robot moving through a warehouse full of crates can be thus approximated. Most planners proposed so far fall in this *kinematic* category.

Although such kinematic planners are capable of handling a large class of motion planning problems, other methods must be sought for cases where the robot's *dynamics*, the physics of their motion, must be taken into account. This includes, for example, cases where the robot's inertia plays a large role in its motion, or where the drive mechanism is non-holonomic, such as in most wheeled vehicles. To this end there has been some research recently, starting with the seminal paper by Donald *et al.*[DXCR93], on *kinodynamic* planners, ones which incorporate both, the kinematic and dynamic aspects of the motion problem. Since dynamics are often important in computer animation, we will pay particular attention to these planners.

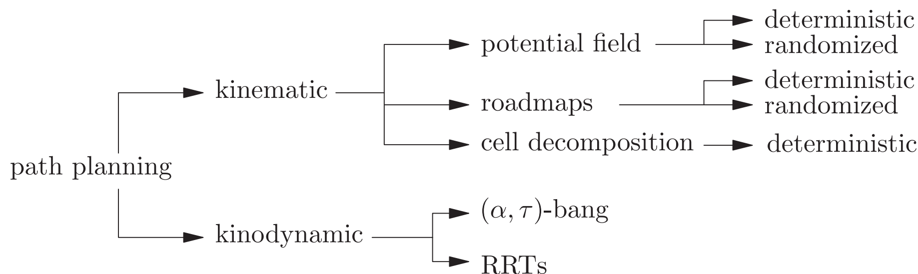


Figure 3: Taxonomy of path planning methods.

In the following subsections we outline some of the key techniques and results of motion planning.

2.2 Kinematic planners

This section briefly describes the three major types of kinematic planners. For an excellent and thorough review of these methods the reader is directed to [Lat91]. Much of what is covered in this subsection is based on material from that book.

roadmaps

The first planners, such as [Nil69], used a notion of a *roadmap*. The purpose of a roadmap is to capture the connectivity of the "roomy", open parts[1] of $\mathcal{C}_{free}$, establishing a canonical set of paths between them. Since these roadmaps span all the reaches of the freespace, every point in $\mathcal{C}_{free}$ is within sight of some part of this network; the path planning problem thus simplifies to finding the two $\mathcal{C}_{free}$ paths that connect $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$ to their corresponding closest points in the roadmap, as well as the connecting path through the graph. If we carry the suggested roadway analogy, given a (very dense) network of highways, the task of "getting from here to there" reduces to finding the on- and off- ramps, and figuring out which highways to take between the ramps.

A number of methods exist for generating roadmaps. The most popular, the visibility graph method, uses a *visibility graph* of $\mathcal{C}$ as a roadmap. This graph is constructed by considering all the vertices of $\mathcal{CB}$, the set of C-obstacles, and adding an edge between two such vertices if they can "see" each other; that is, they lie adjacent on a C-obstacle boundary, or a straight line connecting them lies completely in $\mathcal{C}_{free}$. The planning problem is solved by first treating $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$ as $\mathcal{CB}$ vertices, creating the visibility graph, and then using basic graph theory to find a connecting path. The left diagram of figure 4 shows a solved problem using this method.

cell decomposition

Cell decomposition is closely related to the previous method, as it also, in effect, generates a roadmap. With this method $\mathcal{C}_{free}$ is decomposed into a set of "simple" fragments, termed *cells*. The connectivity of the cells is then captured using a connectivity graph very similar to a roadmap. To plan a motion one simply finds a sequence of cells, termed a *channel*, which connects the cell containing $\mathbf{q}_{init}$ to the one containing $\mathbf{q}_{goal}$.

[1] a useful way to visualize this is to consider $\mathcal{C}_{free}$ as an interconnected system of underground caverns, where the roadmap encodes their connectivity.

The solution path is then obtained by stringing together the midpoints of all the cell boundaries within the channel, and connecting them to $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$. The middle diagram in figure 4 demonstrates the approach.

Various shapes and structures can be used for the cells. They must, however, make it easy to:

- check for adjacency between two cells
- find the portion of the cell boundary which two neighboring cells share
- find a connecting path between any two points in a cell

These conditions respectively ensure the ease of: constructing a connectivity graph of the cells, finding the midpoints, and constructing a continuous solution from the sequence of midpoints.

There are two types of cell decomposition: "exact" and "approximate". The difference is that the sum of cells in an exact decomposition equals the free space $\mathcal{C}_{free}$ exactly, while in the latter approach it is only approximated. The approximation is generally done with regularly shaped, and often (hyper-)rectangular cells which leads to simpler and faster calculations. This approach however can miss potential solutions if the resolution of the cells is not high enough to detect an essential narrow passage. Although one can always increase the resolution, this is mired by an increase in calculation time and memory storage space, which grow exponentially. This though can be mitigated by using hierarchical methods, such as a quad- or oct-trees, to refine the resolution only where needed. The diagram on the right in figure 4 shows approximate decomposition using a quadtree.

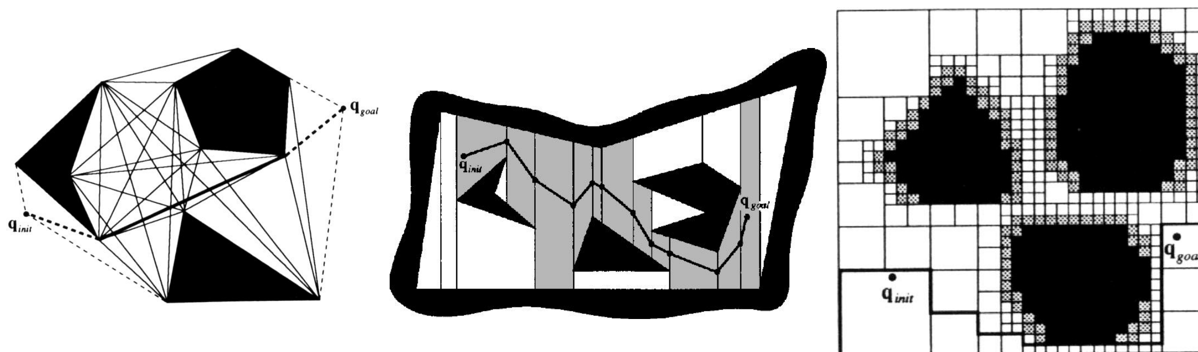


Figure 4: **left**: a visibility roadmap; **middle**: trapezoidal (exact) cell decomposition; **right**: approximate cell decomposition using a quadtree; the *channel* is outlined with a heavier line (*source:[Lat91]; middle diagram modified to show channel*)

potential field

Instead of focusing on the connectivity of the freespace, the last main approach constructs a *potential field* over $\mathcal{C}_{free}$. This field acts as a guiding force, providing local hints as to which way the goal lies, driving the planner towards $\mathbf{q}_{goal}$. The overall potential is often a composite of an overall *attractive potential*, one that provides the driving force, and a *repulsive potential* that repels the planner from the set of C-obstacles in $\mathcal{CB}$. Figure 5 shows an example potential field and solution for a simple problem.

The motion planning problem is solved by performing gradient descent on this potential field. As with all such methods, local minima are a major problem. One solution is to modify the field construction method in a way that minimizes or even eliminates local minima; see *navigation functions* in [Lat91]. The latter proposes a simple way to calculate such a potential field numerically: 1) the freespace is first discretized with a regular grid, 2) the $\mathbf{q}_{goal}$ grid point is assigned a 0 potential, and 3) this potential is then propagated outward, incrementing it by one at each iteration step. [Lat91] dubs this the "wavefront expansion" algorithm.

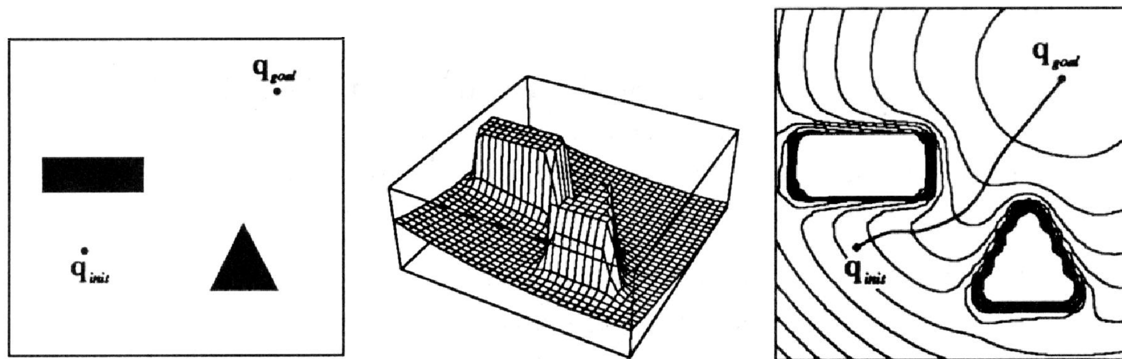


Figure 5: potential field method; **left**: configuration space $\mathcal{C}$ with C-obstacles; **middle**: potential field over $\mathcal{C}$; **right**: isolines and a solution (*source:[Lat91]; right diagram modified to correct* $\mathbf{q}_{init}$ *label*)

randomized methods

The above motion planning approaches are only feasible for low-dimensional configuration spaces since they suffer from the *curse of dimensionality*, having time complexity that is exponential in the number of dimensions. Starting with [BL91], research has started to look to random methods, such as Monte Carlo sampling and Brownian motion, to address motion planning in the more complex cases. By stochastically looking only at a small portion of these exponentially increasing hyper-volumes, large time gains can be made, as compared to the exhaustive brute force search of the previous algorithms. The bulk of motion planning research now focuses on these randomized methods.

[BL91] proposes the *Randomized Path Planner* (RPP) algorithm, a straight-forward extension of potential field planners. The main difference is the use of stochastic sampling to approximate the direction of the downward slope during gradient descent, as searching for it exhaustively is extremely costly. A further stochastic addition, *random walks* are used for escaping local minima.

Roadmap approaches have also benefited from randomization. For example, [KŠLO96] introduces the *Probabilistic Roadmap* (PRM) method. In this method the nodes of the roadmap, here called *milestones*, are sampled stochastically from $\mathcal{C}_{free}$, and connected with an edge only if a simple, collision-free path can be found connecting them. Straight line segments are usually used, much like in the visibility graphs above.

[SL01] has noted that PRM planners spend more than 90% of their time performing collision checking. This work goes on to suggest a new variant of the planner, named *SBL* (Single-query, Bi-directional, Lazy), which uses *lazy collision checking*. All milestones are initially assumed to be connected. When a possible solution has been found, the edges are checked for collisions using a recursive divide-and-conquer algorithm; if a collision is encountered, the offending edge is deleted, the partial collision calculations for the other edges are cached, and the roadmap creation process continues until another candidate solution presents itself.

One of the key problems with PRMs is that they have a hard time discovering narrow passages, often leading to disconnected roadmaps which do not accurately reflect the connectivity of the freespace. This problem has received a lot of attention. [HLM99] explores this topic analytically and defines *expansive* configuration spaces, ones that are likely to result in good probabilistic roadmaps. It further proposes a new planner, one that attempts to only construct the part of the roadmap which is directly relevant to the query. It proceeds by growing two roadmap trees, one from $\mathbf{q}_{init}$ and one from $\mathbf{q}_{goal}$. The trees are grown by picking an existent milestone from the less populated part of the tree, stochastically choosing a nearby point within in $\mathcal{C}_{free}$, and attempting to connect the two without any collision. If successful, the new point is added as a milestone to the roadmap tree. When the two trees overlap each other at a node, the solution can be found using basic graph shortest path algorithms.

[HKL+98], on the other hand, proposes a solution to the narrow passage problem which retains the original PRM algorithm. Here, $\mathcal{C}_{free}$ is initially inflated to widen all the passages. After the roadmap is created using this "dilated" version, $\mathcal{C}_{free}$ is deflated back to its original size and shape, and milestones are adjusted by resampling around the problem areas to ensure that they all, as well as any edges, lie within the original freespace.

It is interesting to note that, of the three original planning approaches, a randomized approach has not yet been put forth for the cell decomposition method, possibly because it is not obvious how to do this. This is perhaps an interesting potential future research direction, seeing how hugely successful the other adaptations were.

2.3 Kinodynamic planners

[DXCR93] was the first work in motion planning to additionally consider the dynamics of the character's motion, in addition to the usual kinematic constraints. The proposed planner attempts to find time-optimal solutions by iteratively exploring the state-space using short fixed interval forward simulations.[2] Starting with $\mathbf{x}_{init}$, all possible control vectors are applied and simulated for a short, fixed duration τ, where each components of the control vector takes on values from $\{-a, 0, a\}$, a being some maximal control value.[3] Each such short simulated trajectory is called an (a,τ)-*bang* motion. This expansion process is repeated on each (a,τ)-bang motion end point from the previous iteration. Any motion segment that runs into an obstacle, or is otherwise undesirable, is discarded. In effect, this method samples the state-space in a regular grid-like pattern, forming a graph. The time-optimal trajectory can be then obtained by finding the shortest path connecting $\mathbf{x}_{init}$ and $\mathbf{x}_{goal}$. Since the method does an exhaustive exploration of the state-space, it is only applicable to low-dimensional configuration spaces.

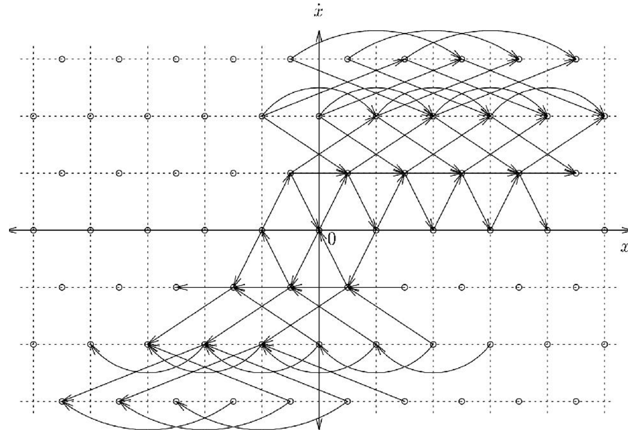


Figure 6: Donald *et al.* (a,τ)-bang exploration approach results in a reachability graph with vertices arrayed in a regular grid over the state-space; a one-dimensional case is shown above; edges have been drawn for clarity, and do not reflect actual trajectories in state-space (*source: [DX95]*)

[LK99] introduces *Rapidly-expanding Random Trees* (RRTs). This approach is similar to that of the expansive space planner of [HLM99], in that it is bidirectional, growing a tree from the initial and goal states. The key difference is that, instead of kinematically connecting the milestones using straight lines, short, physically-simulated curves are used. The other salient point, the reason for the "rapidly-expanding" qualifier, is that the stochastic choice of states for expansion is biased to give a uniform distribution over the

[2]This method bears much resemblance to the animation method of State-space Controllers [vdPFV90], discussed later.
[3]Since *bang-bang* control gives time-optimal motions, no other values need to be considered.

state-space, thus leading to rapid discovery of $\mathcal{C}_{free}$. This is achieved at each iteration using the following procedure for expanding a tree:

- choose a random state $\mathbf{x}_{rand}$ with a uniform distribution the state-space
- find the nearest tree milestone, $\mathbf{x}_{near}$
- forward simulate using all possible control values from $\mathbf{x}_{near}$ for some fixed duration Δt
- pick the control value $\mathbf{u}$ that brings the character the closest towards $\mathbf{x}_{rand}$
- insert the resulting state, $\mathbf{x}_{new}$, into the tree as a new milestone

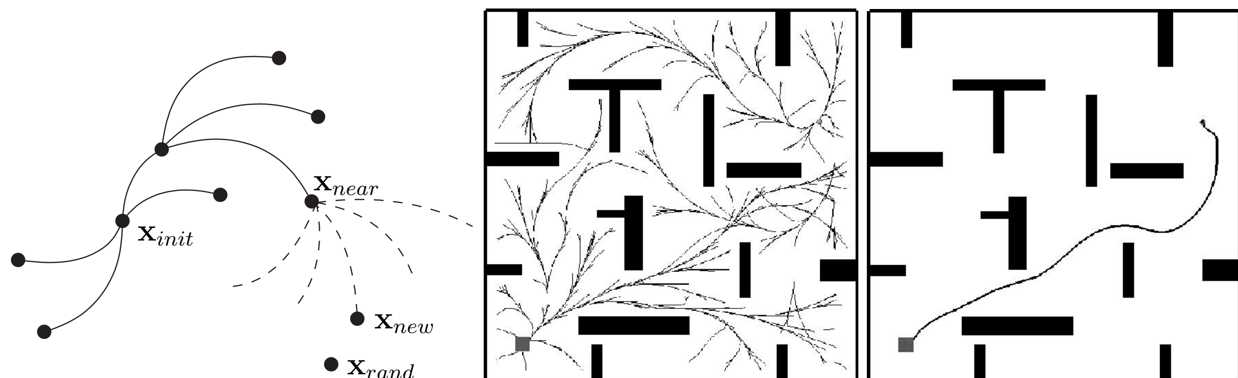


Figure 7: Rapidly-expanding random trees: **left**: the tree expansion step illustrated; **middle**: the two trees for a simple problem involving a non-holonomic robot; **right**: the solution found (*source: middle and right images are from [LK99]*)

[KL00] goes on to apply the RRT concept to purely kinematic problems and further proposes the "RRT-Connect" algorithm to improve the planner's efficiency. The limitation to kinematic problems again allows for the use of straight line segments in connecting the milestones, which drastically simplifies the finding of $\mathbf{x}_{new}$. Further speedup is obtained by allowing the planner to take multiple time steps towards $\mathbf{x}_{rand}$, stopping only when the random state is reached or an interposing obstacle is encountered. The authors speculate that adapting this approach back to the kinodynamic domain should still produce an improvement, although it is not yet quite clear how to do this.

A more difficult problem is that of kinodynamic motion planning within a dynamic environment. [KHLR00] and [HKLR00] use an approach similar to probabilistic-roadmaps over the state×time space. This space is just the regular state-space augmented with the dimension of time (i.e., state×time $\subseteq \mathbb{R}^{2n+1}$, where n is the number of DOFs of the model). The C-obstacles from the state-space get extruded along the time dimension into hyper-tubular shapes; if stationary, the C-obstacles extrude along a straight line, else along a curve representing their motion. This space is explored using a tree of short simulations, much like in RRTs in [LK99], using randomly chosen control inputs. [HKLR00] also applies the planner to non-holonomic vehicles.

For a wider overview of kinodynamic motion planning methods the interested reader is directed to [Kuf98].

3 Animation methods

Animation methods have the potential to play a number of roles in the context of virtual actors. For example, path-tracking algorithms can be used to implement the global motion path found by the planning stage. More local aspects of the motion, on the other hand, such as reaching or waving, will require other suitably-specialized methods. Furthermore, recent approaches do away with the two-stage framework by combining planning and animation into a single simultaneous activity. In this section we thus review a wide spectrum of animation methods. Figures 8 and 9 summarize known methods using two different classification systems.

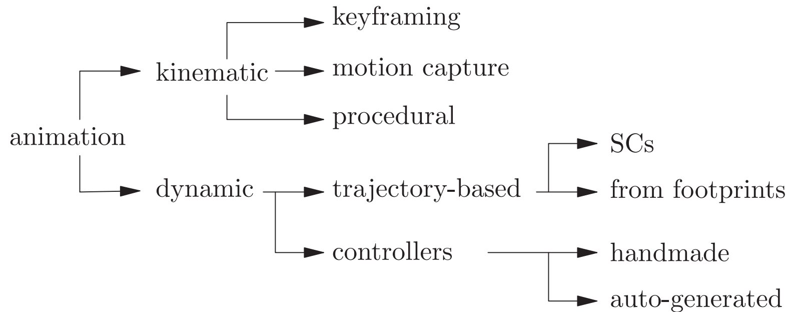


Figure 8: Traditional taxonomy of animation methods.

A key distinction between animation methods, much like in motion planning, is whether they incorporate the character's physics. *Dynamic* animation methods work directly with the forces and torques at the character's muscles to effect changes in the character's state, while *kinematic* methods simply treat the motion as a geometrical transformation in time. Historically, the first animation methods were all kinematic as these tend to be significantly simpler. On the other hand, they usually require a lot of effort to produce natural looking motion for non-trivial articulated models. Dynamic approaches were developed in the hope that this elusive "naturalness" would be derived automatically by restricting solutions to physically correct motions. Unfortunately, this turns out to be insufficient; although much better solutions are produced, only a very small subset appears natural. It is interesting though to note that, unlike robotics and related areas which deal with real-world physics-abiding objects, computer animation does not need to strictly follow

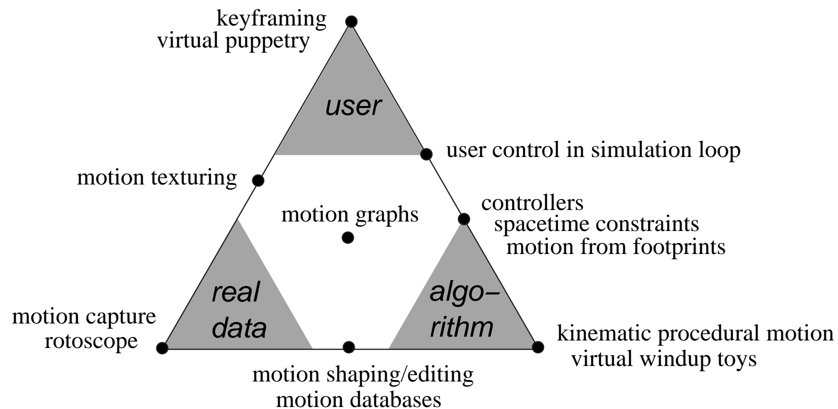


Figure 9: An alternate classification of animation methods; here we classify animation methods based on where the final motion comes from. The three extremities are: "from user input", "from external repository of motion data", and "from algorithm". The location of a few key methods are marked in this scheme.

those laws; physics is merely a heuristic which roughly points in the direction of natural motions. Dynamic methods therefore garner a fair amount current research effort.

3.1 Kinematic methods

keyframing

Keyframing has been the predominant animation method since the inception of computer graphics, and is still the principal method for animating non-humanoid characters and objects. In keyframing the animator provides *keyframes*, the defining character configurations at various moments in the motion, which are then interpolated with a smooth curve, such as a B-spline. The allure of this method is that it poses a relatively gentle learning curve for traditional media animators and cartoonists, allowing them to transfer their long acquired skills and talent, and be able to start producing computer animations with remarkably little additional learning effort.

One of the key advantages much appreciated by animators is that keyframing allows for arbitrary level of control of the final motion, through the addition of further keyframes that refine the motion. This same feature though is also a large disadvantage, in that complex motions require significant animator effort (i.e., large number of keyframes) to implement. Successful use of keyframing also requires a certain level of skill and talent, which severely limits its potential user base.

Being the most ubiquitous, keyframing is the benchmark to which all other animation methods are compared. Research generally focuses on new methods which require either less time or skill to use.

motion capture

One of the most popular methods for the animation of virtual humans and human-like characters is *motion capture* (often abbreviated to *mocap*). Here, a human actor in a motion capture studio acts out the desired motions which are captured using special recording instrumentation, and then mapped onto a virtual character. The motion can be recorded using a number of methods, such as magnetic sensors on the actor, or by affixing reflective markers on the subject and then reconstructing the actor's pose from a video of the motion.

One of the key benefits of motion capture is that motions of all levels of complexity can be captured with equal ease, and the results are completely life-like and natural. A comparable level of detail can be achieved with keyframing only with inordinate amount of effort and animator talent. The approach does have a number of disadvantages though. One downside is the financial expense of setting up such a studio and requisite hardware. Also, the procedure requires a significant amount of pre- and post-processing, from the calibration of recording equipment, to cleaning up of the results, which tend to have a significant noise component, a side effect of the various recording technologies. Furthermore, the method is inherently limited to recordable motions; clips of unacceptably dangerous stunts or physically impossible motions must be obtained by other means, as must motions for characters for which no equivalent real-world actor can be found, such as fictitious triple-legged creatures.

A shortcoming of motion capture data which has received much attention is its limited possibility of reuse. Once a motion is captured it is generally only useful for mapping onto a virtual character with the same dimensions as the actor. Also, should the animator desire a slight variation of a motion, the variant must generally be recaptured from scratch.

Starting in 1995, a flurry of papers appeared addressing this problem of *motion editing*. The most generic approach was put forward by Bruderlin and Williams [BW95], which applies signal processing techniques to motion curves. In particular they discuss multiresolution motion filtering, multitarget motion interpolation with dynamic time warping, waveshaping, and motion displacement mapping. Of these, displacement mapping has become very popular. It constructs the desired motion by calculating an error curve from the user constraints and then adding it onto the original. This popularity is due to the wide applicability of the method, as well as its beneficial property that, unlike the more obvious way of directly editing the original curve's parameters, the error curve can be arbitrarily (and thus differently) parametrized, in a way that is more useful to the particular task at hand.

Others have at the same time proposed similar methods, but with more focus on particular applications. Witkin and Popović [WP95] introduce "motion warping", essentially a mix of displacement mapping and time warping, and use motion blending to effect transitions between motion segments. Unuma *et al.*[UAT95], on the other hand, further explore multiresolution motion filtering and manipulation by performing Fourier expansion on cyclical motion captured data. Motion interpolation, extrapolation, and transitions are achieved by blending the Fourier coefficients. Furthermore, this paper shows how distinct styles or traits of motion, such as "briskness", can be characterized by taking the difference between the coefficients of a motion which displays the desired quality and those of an average, unstylized, equivalent motion. Other motions can then be imbued with the characteristic by adding on this difference.

Rose *et al.*[RGBC96] looks at generating transitions between motions for humanoids in particular. The approach uses kinematic methods to compute the root position, special inverse kinematics for the support limbs, and spacetime constraints (SCs), a dynamic method we discuss later, for the free limbs. Gleicher[Gle97, GL98] uses SCs instead for motion editing through interactive user constraints. Here SC's constrained optimization methodology is used for finding the optimal parameters of a displacement map, one that will result in the final motion best meeting the user constraints. Interactive rates are obtained by simplifying the traditional SC approach: constraints capturing physical laws of motion are discarded, and a much simpler, purely geometric objective function is used. This approach is further used in [Gle98] to addresses the problem of retargeting a motion to a different character, although it is limited to differences only in link lengths, not link skeleton structure. Most of the retargeting constraints can be obtained using various automatic methods, such as a footplant finder, but in the end some have to be provided by the user. Nonetheless, these need to be provided only once for a given source motion, and amount to little effort when compared to generating a new motion from scratch.

Finally, Lee and Shin [LS99] propose a few improvements to interactive motion editing in [Gle97, GL98]. First, instead of using a single B-spline curve, the displacement map is represented by a hierarchy of B-splines. Each one cumulatively improves the resultant motion, and allows for matching the requested constraints with arbitrary, user-specified precision. Second, a much quicker way of calculating the displacement map is introduced, thanks to the new representation. Instead of the constrained optimization of spline parameters, a customized inverse kinematics solver is used to translate the user constraints (footplants, other end effector positions) directly to character configurations, which are then interpolated with the hierarchical spline through the application of multilevel B-spline fitting techniques.

A particularly interesting, recent development is the use of motion capture data for *texturing* or enhancing and enriching simplistic sketches of the desired motion [PB02], instead of being used for the final motion wholesale. The sketched motion usually only describes a handful of the most important DOFs. It is broken up into segments, which are then compared to similar chunks in the mocap database, and their most appropriate counterparts are identified. Each sketch segment is then replaced by said counterpart. This has the effect of "filling out" the missing DOFs' data, as well as enriching the excessively smooth splines of the remaining DOFs.

procedural

Another approach to animation is through the use of relatively simple algorithms or procedures to directly drive the character's degrees of freedom. A trivial example of such a *procedural* approach is using a sinusoidal function to drive the angular displacement of a pendulum. The procedures are often parametrized to allow for variation in a certain aspect of the motion, such as direction or speed.

These *procedural* methods were the first animation methods due to the ease with which they are implemented. Although still in use today for animating simple objects, or parts of more complex characters, their deployment is very limited due to the difficulty and effort required to procedurally create more complex motions. Nonetheless there have been attempts to employ this technique for human character animation, such as [Per95]. Here, each actuated joint angle is driven by a combination of sinusoidal sources, with some stochastic noise added in to give a less "computer-generated" look. The process of constructing and tuning the expressions for the various DOFs is done by hand. Furthermore, motion transitions must be well timed and constrained to occur only when the two motions are in compatible alignment to prevent utterly unrealistic results.

3.2 Dynamic methods

Dynamic methods enlist the aid of physics in producing realistic motions. The character is modeled as a dynamical system

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, t)$$

where $\mathbf{x}$ is the state vector (see section 1.2) and $\mathbf{u}$ is the *control vector*, the set of inputs to all the actuators of the character.

These methods can be roughly split into two types: trajectory methods and simulation-based approaches. The trajectory-based methods work with the whole motion in the form of a curve through the character's state-space, usually employing optimization to iteratively improve the motion's physical correctness and naturalness. Simulation-based methods, on the other hand, employ either hand- or auto-generated controllers that directly drive the torques at the joints, and employ forward physical simulation to generate the actual motion.

trajectory methods

The most influential trajectory-based method was proposed in 1988 by Witkin and Kass[WK88]. They proposed formulating character animation as a constrained optimization problem. The constraints are obtained from a number of sources: the character's structural limitations, such as allowable joint motion ranges or acceleration bounds; its initial, goal, and possibly intermediate configurations; and most importantly, the laws of physics. These constraints serve to circumscribe the set of acceptable trajectories through the state-space, thus narrowing the choice of potential solution motions. As this still leaves a rather large selection, optimization is used to pick out a solution that is optimal in some sense, as defined by the *objective function*. It is common for these objective functions to measure motion smoothness or energy-optimality, and hence they often contain terms such as

$$J = \int_{t_0}^{t_1} |\dot{\mathbf{x}}(t)|^2 \, dt \qquad \text{or} \qquad J = \int_{t_0}^{t_1} |\mathbf{u}(t)|^2 \, dt$$

These Spacetime Constraints (SC) can be used to produce a wide variety of interesting motions. Since this method works with the whole trajectory at once, it inherently produces appropriate anticipation as well as follow-through for jumps and bounding motions. Even if the problem is over-constrained by the user

(e.g., physically impossible sequence of configurations to interpolate, or impossible time intervals between them) SC can return the solution that is the closest to being physically correct, while satisfy the remaining constraints.

The original paper [WK88] animates Luxo, a simple energetic lamp, through a number of relatively simple motions. Unfortunately the SC method doesn't scale very well to more complex characters, or more elaborate motions. Cohen [Coh92] attempts to speed up the optimization by allowing the user to guide the process. This is done by interactively varying the importance of the various constraints. Furthermore, an interactive framework is proposed which also allows for the addition and modification of SCs. The user need not work on the whole motion at once, but can focus and frame parts of it as individual SC problems through the use of *Spacetime Windows*. These windows can comprise any or all DOFs of the character, and can extend over any time-contiguous subset of the motion. Further speedup is achieved by representing the time trajectories for the various DOFs as B-splines instead of a sequence of discrete values. This significantly reduces the number of parameters to optimize: in the original approach each DOF contributes a parameter for every frame of motion, while the latter often exposes only a handful of spline control points for each DOF's curve.

The number of B-spline control points to use is a difficult choice. Picking too few may cause the curve to be incapable of representing the desired motion, while picking too many causes unnecessary, expensive computation. This has led to [LGC94], which proposes the use of wavelets to model the DOF motion; this allows the use of the simplest possible curve for the basic motion and adaptively adding detail only where necessary.

The SC approach requires an initial guess of the motion before optimization can begin, since these tend to be implemented as local searches, and thus require some starting point. A bad guess for this motion can lead to slow convergence. Furthermore, if the specified constraints allow for multiple visibly distinct motions (e.g., an underhand vs. overhead throw), the initial guess indirectly determines which solution is found. Ngo and Marks thus propose a method [NM93] for generating these initial motions using a sensor-actuator controller (this work is discussed later, in the physical controller section). A more recent and advanced approach is that by Liu and Popović [LP02]. Here the user provides a rough and simplistic keyframed motion of the desired result. The keyframes are usually incomplete, giving values only for DOFs which are particularly significant to the motion. For example, keyframing just the character's location (x, y, z) is enough for obtaining realistic jumps. The system analyzes this simple motion, extracts various spacetime constraints, and then feeds them to a SC optimizer. This adds the missing details to the motion in a physically correct way.

An altogether different trajectory-based approach was proposed by van de Panne in [vdP97], where the animation is driven by the placement, timing, and duration of footprints. These can be either user-specified or obtained from external sources, and, along with rudimentary dynamics and optimization on a simplified model, are used to first establish a reasonable and physically plausible path for the character's center of mass (COM). The objective function in the optimization contains a term to encourage a comfortable distance between the COM and the footprints. The final step in the process uses inverse kinematics to place the character's limbs at the footprints. [TvdP98] extends the work to the more difficult case of a quadruped by modeling the character as a pair of mass points, one over the front and another over the back set of legs, which are connected by a spring.

simulation-based approaches

A more direct approach to physics-based animation is to manipulate the character by explicitly controlling the actuating forces and torques at the character's muscles, and using physical simulation to obtain the character's motion. The manipulation is performed by a *controller* which embodies a set of rules, the *control laws*, that aim to bring about a certain behavior, or set of behaviors, such as walking or balancing.

The use of a controller holds many advantages over trajectory based methods. Firstly, given a pre-fabricated controller, the generation of motion is relatively fast: instead of iterative optimization, all that is needed is a

single simulation using the control inputs calculated by the controller at each step of the way. Secondly, unlike motion capture and trajectory-based methods which return an unparametrized and generally unmodifiable, single instance of motion, a controller is applicable to a significant subset of the character's state-space (i.e., capable of handling a large variety of initial states of the character) and is thus able to generate a whole family of motions. The controller's *robustness* is a qualitative measure of the region of state-space over which it is applicable.

Although easy and fast to use, creating a controller, especially a robust one, is difficult and requires a fair amount of physical knowledge and "feel". With time it is possible to accumulate a large palette of controllers, but unavoidably, at some point the animation desired by the user will fall outside its scope. In this case the animator either has to forgo the motion, or design a new capable controller from scratch, something that requires a large time investment, especially if the animator has to first develop the necessary skills. Methods for controller design that do not require specialized skills or knowledge are still largely an open problem which has attracted a fair amount of research; we will discuss these methods after looking at hand designed controllers.

hand designed controllers

Initial work focused on hand design of controllers. This is a difficult and time-intensive task since making a controller stable and capable of a desired motion requires much tweaking and careful fine tuning. A common strategy for mitigating the difficulty of their design is to "divide-and-conquer" using *functional* and *temporal decomposition*: designing a set of specialized controllers, each crafted for a singular task or motion type. Such specialization not only makes the design process easier, but also tends to results in more robust controllers. Character animation is then performed by employing the most appropriate controller for the given situation.

Early controller design efforts concentrated on statically stable models, such as insects. [MZ90] dynamically animates a virtual cockroach using a two level approach. A *gait controller* synchronizes the timing of the legs and the overall motion by executing low-level *motor programs*, routines which do the actual limb manipulation by varying the rest positions of exponential springs. Inspired by biomechanical data, each leg is equipped with an oscillator that triggers the leg's transition between two alternating stages: *step* and *stance*. In the step phase the motor program brings the leg up and over to a new foothold, while in the stance stage the leg serves to keep the body supported. A further refinement is the use of *reflexes* to trigger a step early if the foot is overextended or to delay the step if the leg still bears too much of the body's weight. The results correlate well with actual cockroach motion, and contain important features such as the emergence of *wave* and *tripod* gaits as the oscillator frequency, and thus implicitly travel speed, is changed.

[RH91] tackles dynamic gaits of creatures. In particular, the paper describes controllers for a biped, quadruped, and a single-legged kangaroo-like creature. The biped and quadruped controllers are a computer graphics simulation of real hopping robots at the MIT Leg Laboratory. The use of hopping as the basis for locomotion was motivated by the observed behavior of real hopping animals, ones that have efficient gaits due to elastic ligaments that effectively capture some of the ground impact energy and release it on the subsequent hop. This is modeled by using telescoping legs. Posture control is effected through applying torques during leg contact with the ground, while speed control is achieved by selecting a touchdown point behind or ahead of the *neutral point*[4] (to speed up or slow down, respectively). The overall motion is driven by a finite state machine (FSM). Biped motion is treated as if it was single-legged, with the *idle* leg kept short and out of the way. All gaits of the quadruped, other than the gallop, pair up the legs, with each pair moving in unison, and thus reducing the control problem to that of the biped. The method of pairing determines the gait: diagonal pairing gives trotting while pairing front and back legs separately gives bounding. Galloping is treated with an extended FSM, with extra states for the various leg-ground contact combinations.

[4]neutral point: contact point that results in symmetric body motion, or alternatively where the leg angle with respect to ground is symmetrical between touchdown and liftoff

Hand designed physical controllers for human-like characters first appeared in [HSL92]. Crude and planar human models are made to pump a swing, ride a seesaw, juggle, and pedal a unicycle. Again finite state machines are used for the overall motion, while spring-and-damper actuators provide the low-level control. The swinging and seesaw scenarios associate a desired target pose of the character with each state of the FSM, which the underlying spring-and-damper actuators attempt to match. State transitions are triggered based on the character's state. The FSMs for juggling and unicycle use less generic, hand-tweaked control laws.

By 1995 human controllers have advanced significantly. [HWBO95] demonstrates controllers capable of running, bicycling and vaulting for fully 3D characters. As in previous work, a cyclical state machine representing the various stages of the motion selects which control laws to apply at a given point in time, while inverse kinematics are used to select active limb placement, and spring-and-damper control is used to achieve the resultant limb configurations. Limbs not directly involved in the motion are used for absorbing disturbances and balance.

One serious drawback of hand designed controllers is that they are often very sensitive to the character model being used. Varying body dimensions or mass distribution will frequently result in controller malfunction. [HP97] introduces a method for automatically adapting controllers to such model variations. In a two stage approach, a number of the controller's parameters are first scaled and then optimized (only a small subset) using simulated annealing. The technique is applied to running and bicycling controllers by adapting them from an adult male model to one of a female and a child.

The problem of combining multiple specialized controllers into a "mega-controller" is addressed by [FvdPT01]. Faloutsos *et al.* describe a framework where a *supervising controller* delegates character control to a collection of *individual controllers* based on the current state of the character as well as the controllers' "pre-conditions", the regions of state-space which form acceptable entry points of each particular controller. The framework is applied to a virtual stuntman, demonstrating recovery from shoves and other external stimuli, as well as getting up after knockdowns, from various prone and supine configurations.

automatically generated controllers

One way to make creation of controllers more accessible to the general user is to offload most of the work onto the computer. Progress in research in this area so far is mostly limited to simple characters.

An early approach introduced the concept of *state-space controllers* [vdPFV90]. Each such controller is associated with a particular goal state for the character, and encodes the set of all time-optimal trajectories leading to it from every conceivable starting state. This formulation leads to some interesting properties: a controller with a goal state with nil velocities will bring the character to a stable stop at the corresponding goal configuration, while one with non-zero goal velocities will result in a cyclical motion, as the character time after time keeps overshooting the goal configuration. Interesting and useful motion can be produced by creating a number of such controllers and allowing character control to switch between them progressively. The controllers are generated with the help of dynamic programming and local optimization methods. Three applications are presented: a pendulum, a car being parked with nearby obstacles, and Luxo, the lamp, doing flips.

[vdPF93] presents Sensor-Actuator Networks (SANs), a structure reminiscent of neural networks, for discovering a particular character's modes of locomotion. The user provides the character's structure, as well as the type and placement of actuators and various sensors. The system then finds feasible locomotion modes for the character using a *generate-and-test* method: at each iteration a random assignment is made to all of SAN's parameters, and the controller's usefulness gauged. The metric used is the distance traveled in a fixed amount of time. Only 1–5% of the tests result in useful controllers. These are then fine tuned in a *modify-and-test* fashion, by repetitively applying a single step of gradient ascent or simulated annealing on the SAN's parameters, and immediately checking the controller's new score. The approach allows for 4

types of sensors: tactile, angular, length, and eye sensors. The last is particularly interesting as it allows the creatures to "see" and track a *follow point*, which effectively allows the user to direct the character around.

The low incidence of useful controllers above, as well as the general observation that most interesting motions (and useful controllers above) are cyclic, leads van de Panne *et al.* to use an inherently cyclical structure in their "virtual wind-up toys" [vdPKF94]. The *cyclic pose control graph* is a finite state machine where each state carries associated target pose for the character, to be matched using spring-and-damper PD actuators, while each arc specifies the time delay before the corresponding state transition is executed. This structure is similar to that used in [HSL92], although here state changes are triggered purely by elapsed time rather than the character's state. Particular controllers are created using the generate-and-test and modify-and-test techniques from [vdPF93]. Interestingly, although this open-loop system lacks feedback, it can create interesting behavior such as period doubling, or even chaotic motion. The method is applied to a planar Luxo and a virtual cheetah. [vdP96] extends the approach to 3D, allows the optimization of state durations, linearly interpolates between a number of specific controllers to obtain parametrized ones (e.g., interpolating between "running" and "walking" controller parameters), and achieves aperiodic motion through the unwinding of cycles of the FSM.

A key problem with local optimization of controller parameters is that the search space is rife with sharp local minima (or maxima, as the case may be), which makes it very difficult to reward partial progress, and thus to find the extrema. Inspired by the real-life example of a baby learning to walk with a parent's helping hand, [vdPL95] attempts to aid the optimization process by providing motion guidance through the use of a "Hand of God" (HOG), a balancing external torque, which in effect reshapes the extrema to be more mild and easier to climb. This technique is applied to the synthesis of a walking controller in a three stage process: after a stable controller is found for the HOG-aided character, it is optimized to minimize the reliance on this external support, followed by complete withdrawal of the "hand". The last step is not always possible, which is ascribed to some models simply not being capable of the desired gait.

Another nature-inspired approach to controller synthesis is the use of genetic algorithms. Ngo and Marks [NM93] encode the character's behavior as a set of *stimulus-response* pairs, where the stimulus is the subset of sense space which triggers the corresponding response, a target configuration for the character. At each time step of the simulation a single stimulus-response pair is active, dictating the character pose to be matched by a kinematic method imitating a PD controller. New behavior variants are obtained through gene *crossover* and *mutation*, which are then ranked by the *fitness function*, a measure of their suitability (in this case, distance covered in a given amount of time). This global search approach has resulted in some very interesting motions for some five link characters of various topologies: a 5-link chain, a 5-link star, and a 5-link spine-on-legs humanoid-like creature. Sims' virtual creatures [Sim94b, Sim94a] go one step further by also allowing mutation of the character's structure. The character is represented by an arbitrary tree of rectangular solids. It is controlled by a neural-net-like structure, where the nodes of the middle, "hidden" layer perform an operation chosen from a rather large repertoire, ranging from simple summing and multiplication to integration and various types of oscillators. Some of the evolved motions include swimming, walking, jumping, and light source following. Although the method produces some bizarre and fascinating motions — and creatures! — the applicability of this approach is very limited since most animation tasks already have a specific, non-negotiable character in mind.

4 Motion graphs

A promising animation method that is gaining attention recently is the *motion graph*, an entity that encodes all available motions of the character, as well as the possible transitions between them, as a directed graph. The power and usefulness of the construct comes from its ability to synthesize new motions of arbitrary length and variety, simply by performing graph walks. Furthermore, many motion queries can be implemented using graph path search methods. This paper devotes a separate section to this topic since the subject in general has not been explored very much, and promises plenty of research opportunities.

At its heart, the motion graph approach is kinematic, but various parts of it might be enhanced with dynamics in actual implementations. There are two key areas to motion graphs: their creation, and their application. The graph creation process finds transitions between motion clips, usually by only considering the character's kinematics. The original motion clips themselves can potentially come from a dynamic process, such as simulation. The application is likewise mostly kinematic since selecting an appropriate graph path and then pasting together the constituent motion clips does not require considering dynamics. One of these two stages has to actually instantiate the transitions as motions between original motion clips. Here one might see some dynamics, depending on the motion blending method employed (e.g., Spacetime Constraint based techniques).

The general idea of the motion graph has already been in use in video games, under the guise of *move-trees* [MBC01]. These are constructed by working out all the motion segments that will be necessary (e.g., steps of varying stride, turns of various curvature, different range jumps, etc.), recording them, and then manually assembling a tree that captures possible motion successors for each segment. This is an extremely time consuming task due to the exact timing and positioning constraints required to ensure that there is no motion discontinuities ("popping") at the transitions. Recent work in motion graphs focuses on automating this tree/graph construction process.

Perhaps the most lucid implementation of motion graphs is given by Kovar *et al.*[KGP02]. In the graphs proposed here, edges are associated with motion segments while graph nodes correspond to motion frames, and thus character states, at which the character can choose from a number of subsequent motion paths. Any walk of this graph produces a valid and continuous motion for the character. Queries for motions specified with an initial and final character state can be easily solved by applying any "shortest path" algorithm. The graph is constructed from externally generated motion data; in this case a single 78.5s motion capture of an actor walking in various styles as well as performing karate moves. The key step to this process is identifying pairs of points in the motion data at which the character is in a nearly identical state, and creating *transitions* between them. Where the original motion data can be envisioned as a long string of frames, or a disjoint set of such strings, these transitions are effectively shortcuts and detours that transport the character backward or forward to a different part of the original motion string(s). This is illustrated in figure 10. The motion frames forming the endpoints of the transitions thus become the nodes of the graph, while the node-delimited segments of the original motion data, along with the transition motions, form the edges. [KGP02] goes on to show one method of using motion graphs for animation: the graph is used to create a character motion that best accommodates a user-specified path sketched on the ground. One distinguishing feature of this work from the other motion graph approaches is that here the original motion data can be labeled with a "style" (e.g., "walking", "sneaking", "karate moves"). This is used to give the user more control in getting a desired motion by allowing the user to constraint various segments of the solution to a particular style or set of styles. Unfortunately most third-party or older motion capture data is not labeled, and for large motion data sets this could very well be quite time consuming to do manually. Automatic labeling of motion styles is currently an open problem.

A far more extensive work on motion graphs appears in [LCR+02]. This paper applies motion graphs to four environments: a maze, a "discretized" rough terrain, a jungle-gym playground, and a step stool on the floor. Three different user-interfaces are provided for controlling motion generation: user-sketched path on the ground plane, a vision-based performance method where the user acts out the motion desired, and a "choice" interface in which the user is continuously presented with a preview of a handful of available motions at that point in time. It should be noted that the method by Kovar *et al.*[KGP02] from above is a subset of this work; it is equivalent to the "sketch" interface used in the "maze" environment. The underlying structure used here to model the motion is a first-order Markov Process: a matrix P_{ij} stores the probability of transitioning from frame i to frame j, where each entry is computed based on character state similarity. As this matrix is rather dense and requires $O(n^2)$ storage space, a number of filtering operations are applied to make it sparse, such as pruning of entries below a user-specified probability threshold or pruning when frames i and j have different ground contacts. This formulation has been inspired by work on *video textures*[SSSE00], where it made a lot of sense; unfortunately it is somewhat awkward and cumbersome as applied to character motion, and the approach would gain much in clarity if formulated as an equivalent

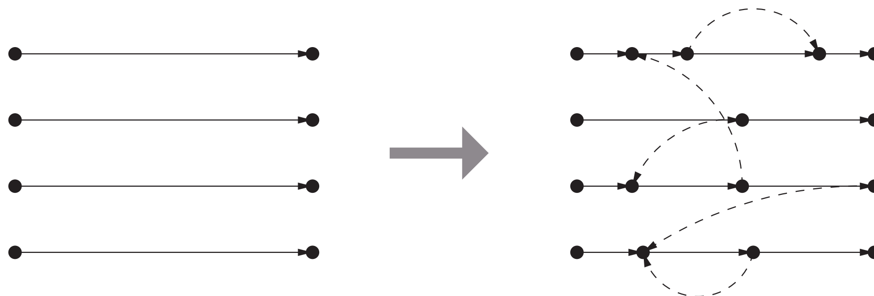


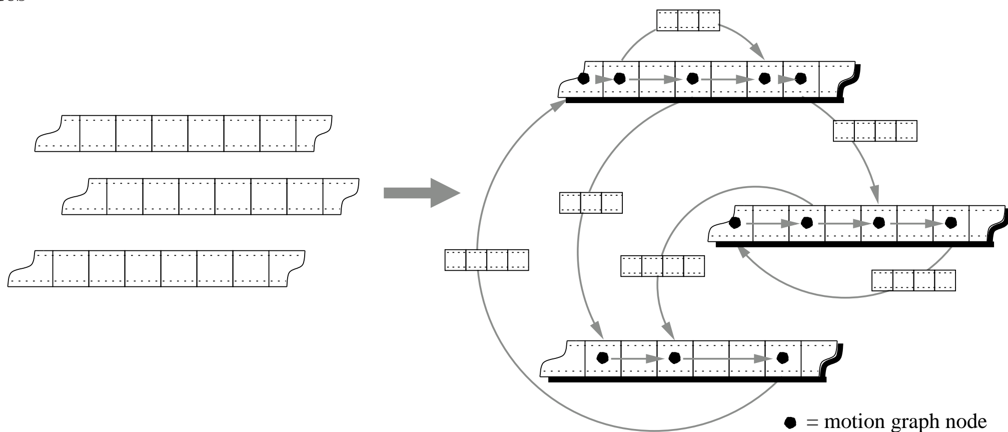
Figure 10: In [KGP02], *transitions* can transport the character to other motion clips, as well as forwards and backwards in the same clip. **left**: original motion clips; **right**: similar frames have been found and transitions between them inserted, thus forming a motion graph

directed graph. A rough isomorphism is not hard to see: non-zero entries in P_{ij} correspond to the transition-type edges of the graph, the frames i and j for such entries constitute the nodes, and the remaining motion data thus segmented makes up the remaining edges. A key difference with the Markov Process approach is that the transition motions are not computed at graph/matrix creation time, but rather on the fly during motion synthesis. Furthermore, the transition trajectories are not fixed: as transition points are encountered in the synthesis stage, a trajectory that smoothly eases into the target motion is computed and buffered as a sequence of frames. These are then used for the subsequent time steps, instead of immediately switching to the target motion. If another transition point is encountered before the buffer is exhausted, the second transition motion is computed from the current buffered frame. This inter-transition interaction can easily result in a large number of transition motions, many more than the number of non-zero entries in P_{ij}, which would be difficult and expensive to precompute.

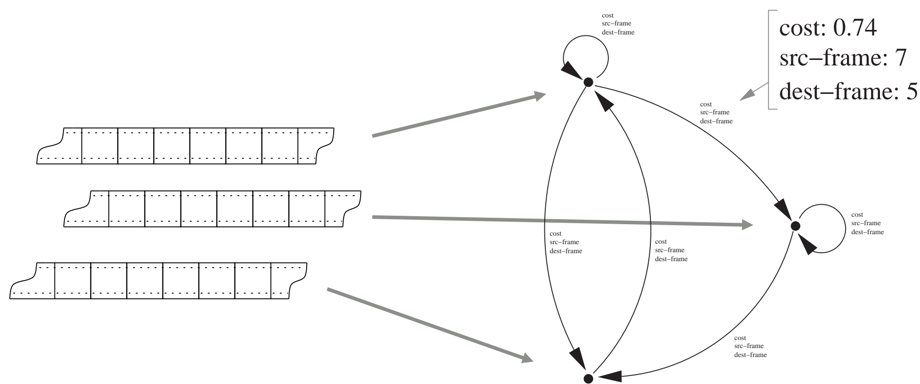
[LCR$^+$02] also employs a second, higher layer for summarizing available motions, using statistical models. All the frames are clustered into groups based on character state similarity, and each frame of motion is then annotated with a *cluster tree*, the hierarchy of clusters accessible from the frame within some fixed number of time steps. The purpose of these is to roughly capture the set of available distinct behaviors that can be executed at a given frame. The main benefit of clustering is the limiting of the number of motion choices presented to the user in the “choice” interface, as well as the improvement in efficiency in finding the best matching motion in the vision-based interface; the sketch-based interface stands to gain little from this extension, hence clustering is not used there.

Arikan and Forsyth [AF02] form the motion graph in yet another way: here the nodes represent whole input motion clips, while the edges denote potential transitions between them. Loops (i.e., transitions between frames in the same motion clip) are allowed and frequent. Each edge is annotated with the frame numbers of the endpoints of the transition, as well as an associated cost computed by a similarity metric, reflecting how much discontinuity would be introduced by the transition. Transitions with costs higher than a user-specified threshold are discarded. Furthermore, a hierarchy of summarizing graphs is created to provide various granularity levels. The original graph represents the finest granularity, while the coarsest level is obtained by clustering of similar edges, which often appear in clumps due to strong similarity between consecutive motion frames, and replacing each cluster with a single edge. The in-between graphs are obtained from the coarsest by recursively splitting the clusters into two. The main reason for the hierarchy is to speed up motion query evaluation which can first find a coarse solution using a smaller graph. Queries are specified in the form of hard and soft constraints by the user. Hard constraints, such as frames to pass through, are met exactly, while soft constraints are met as closely as possible by optimizing an objective function containing said constraints. Solutions are found by applying a random search algorithm, whereby a handful of initial, very coarse solutions are repeatedly mutated until the objective function reaches a local minimum. Multiple motions may be found as the various candidate solutions mutate to different minima.

method [KGP02]: transition clips connect similar frames in original data; frames at arc endpoints become graph nodes



method [AF02]: clips becomes nodes; edges describe possible transitions



method [LCR$^+$02]: a non-zero $P_{i,j}$ entry signifies an available transition from frame i to frame j (the clips have been serialized first)

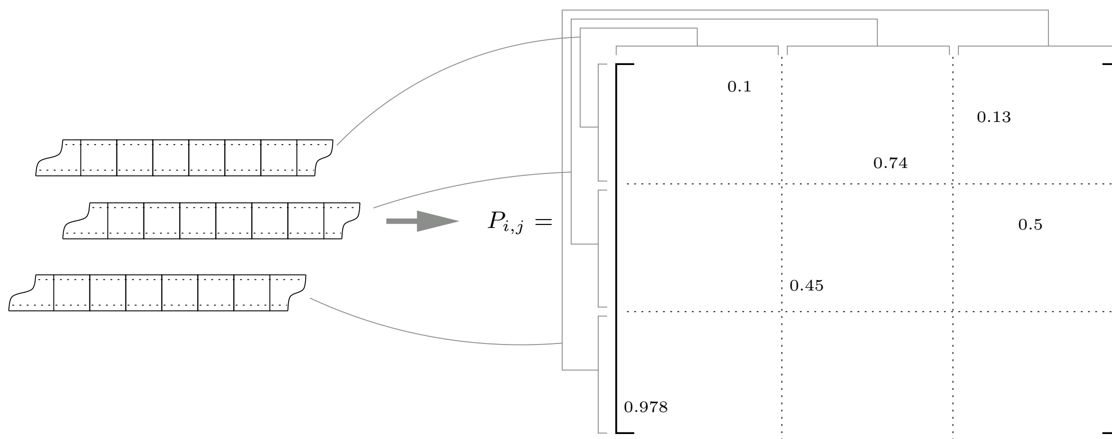


Figure 11: Comparison of the three implementations of motion graphs (equivalent graphs shown).

4.1 Related approaches

The earliest predecessor of motion graphs was the "motion database" approach of [LvdP96], which demonstrates Luxo, the lamp, traversing uneven terrain by drawing on a large collection of example motions, picking the ones most useful for a particular terrain segment, and splicing them all together. The high level planning is done using the decision-tree algorithm[HvdP96], by searching the tree of possible motion sequences, to some arbitrary maximum depth, for the most appropriate line of action. The tree is built recursively by appending as children all clips from the database whose initial frame approximately matches the last frame of the parent motion. The key difference between motion graphs and this database method is that the latter models elapsed time explicitly, which forms the depth axis of the tree; if that was to be eliminated one would have a motion graph equivalent.

Another closely related approach is the use of statistical methods to model motion. The similarity is so strong that it is often hard to judge which camp a given work belongs to. These statistical approaches often use a graph-like structure to capture the connectivity of human configurations, but synthesize motion based on the statistical properties of the input data rather than using the original motions themselves. This sometimes results in motions which do not look very natural, are excessively smooth, and lack any characteristic nuances from the original motion.

For example, [Bow00] describes a method whereby Principal Component Analysis (PCA) is applied to reduce the dimensionality of the motion data, while clustering, and a secondary PCA on each cluster model the data as a set of linear patches, similar in spirit to approximating arbitrary curves with a sequence of straight line segments. The principal components of each cluster can thus be used to efficiently span the space of possible character configurations within the locality of the cluster. Temporal behavior is modeled using Markovian analysis; a probability matrix, similar to the P_{ij} of the [LCR+02] approach, is computed based on the inter-cluster transition frequencies. Motion synthesis is achieved by working out the most probable cluster sequence using the probability matrix, and then interpolating between the *exemplars*, the mean configurations of the corresponding clusters. This work has limited use since motion synthesis cannot be controlled; one can only synthesize "the most likely motion" for the character, and even that is of limited use as it is heavily biased by the contents of the training data.

[TH00] describes a very similar approach, with the key difference that the final motion is spliced together from original motion clip segments. Once again PCA is used to reduce data dimensionality, a K-means classifier subdivides it into K regions, and a Markov chain is used to capture inter-region temporal behavior. The innovation comes in the second layer, a discrete output Hidden Markov Model, where the hidden states are the various training motion clips, while the observed symbols are the region labels. Motion queries are accepted in the form of an initial and goal keyframe. These are vector-quantized into their corresponding regions, and a most probable region-path between them is computed. The Viterbi algorithm is used to find out the most likely sequence of training motion "segments"[5] which are then joined together using linear interpolation over a small window at the segment junctions.

A related recent approach, that of *motion textures* [LWS02], chooses an interesting alternative: instead of manipulating segments of training data, it works with motion primitives called *motion textons*, basic building blocks of motion, which are implemented as linear dynamic systems (LDS). Each LDS attempts to reproduce a fragment of the character's motion; they are robust enough to be able to handle mild variation in initial conditions. The LDSs also contain a Gaussian noise term to give slightly different behavior on multiple invocations. A second layer again attempts to capture the global behavior of the character using a texton transition probability matrix. Motion queries are specified as a two-boundary problem, with texton endpoints. The authors claim that their LDS approach is superior to that of [TH00] as it allows for motion primitive editing, yet further on they note that this editing is not very robust: the primitives can be edited a limited amount before the texton becomes "contaminated".

[5] "segment": in the context of this paper, it is a portion of an input motion clip circumscribed by frames belonging to a different cluster

4.2 Key issues

A few factors and issues play a critical role in the creation of effective and useful motion graphs, and thus deserve special attention.

relativity

A key design decision for a motion graph is whether a fixed coordinate system will be used, or whether all motions will be defined relative to the character, at least in some DOFs. Figure 12 illustrates the difference between the two graph types. In [LCR+02], for example, the jungle-gym and stepping stool scenarios use a fixed coordinate system, while the maze and the rough terrain use a relative one[6]. As the authors point out, this was done because handling obstacles in relative coordinates is difficult. This is a very important open problem for motion graphs.

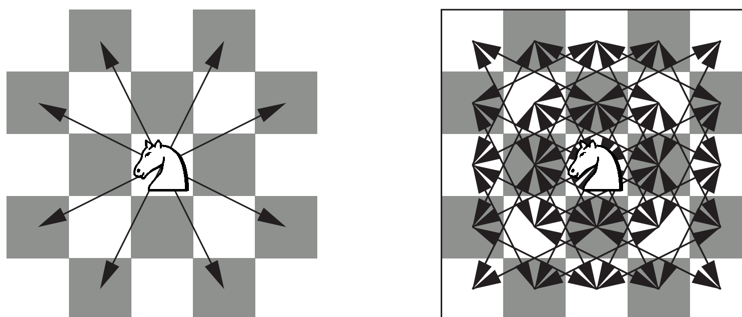


Figure 12: **left**: relative coordinate system motion graph; **right**: absolute coordinate system motion graph

The disadvantage of using fixed coordinate graphs is that, although flexible and simple, they contain a large amount of redundancy: they frequently contain a number of motion clips which, for most intents and purposes, are identical, other than the motion's location and orientation. Conversely, a particular type of motion recorded at one point of the workspace does not automatically allow the virtual character to perform it in other parts of the environment; this can only be achieved by having the actor perform the motion numerous times at different locations. A relative motion graph eliminates this redundancy, by always storing motions in the character's frame of reference.

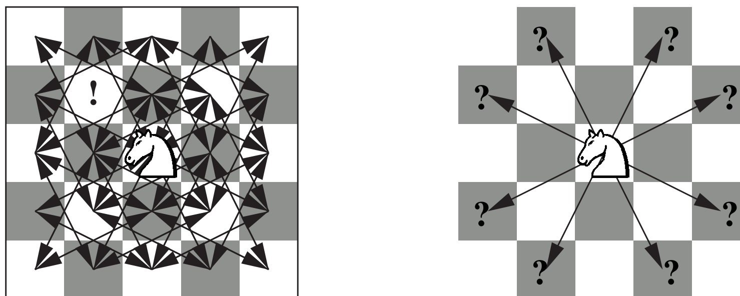


Figure 13: **left**: obstacle ('!') avoided trivially in absolute coordinate system by removing colliding motions; **right**: in relative system, there's no way to know *a priori* which motions are going to collide

[6]only the character's x and z coordinates are relative here (y is "up")

The difficulty of handling obstacles is illustrated with the simple example in figure 13. When working in a fixed coordinate system, avoiding an“obstacle square” is trivial: all motions which would cause a collision can be discarded at the outset, and thus any paths obtained from the reduced graph will automatically be collision-free. In the relative coordinate system, on the other hand, depending on the knight’s position, the obstacle square can appear at any of the available bearings. Because of this, there is no simple way of discarding such colliding paths *a priori*; the collision check must be made at each step of the planning process.

state distance metrics

The key activity of motion graph construction is the identification and implementation of transitions. This usually consists of three steps: using a similarity/distance metric to find pairs of “similar” states that are to become transition end points, computing the transition, and finally correcting any constraints that were broken, such as footholds or hand grasps. We begin by looking at the various metrics used.

Historically, the most common configuration similarity metric is the Euclidean, L_2 distance. When comparing states, temporal behavior (i.e., velocities) must also be considered. This has traditionally been achieved by computing the metric as the sum of L_2 distances over a time window: that of the two “similar” states, as well as a handful of subsequent corresponding state pairs following them. Unfortunately, this metric rapidly loses effectiveness as the dissimilarity of the two states grows. The main problem is that the L_2 metric treats all the state coordinates equally, which is rarely the case in reality: a variation of angle at a joint near the character’s root almost always produces a dramatically larger displacement than the same variation made to a joint near the extremities. A further complication is that the parametrization of the character’s state is not unique; by varying the locale of the character’s root alone, one can produce an infinite number of them. The L_2 metric behaves differently under each parametrization, favoring certain state pairs under one parametrization, different ones under another.

[KGP02] attempts to solve the latter problem by making the state parametrization irrelevant. The metric is computed by summing L_2 distances between corresponding pairs of points on the character’s body in the two states. Each such “point cloud” would ideally be some downsampling of the mesh representing the character’s body surface. Temporal behavior is captured using the same time window approach described above.

[LvdP96] takes a similar route with its *mass-distance metric*. Again L_2 distances between sets of body points are used, but here they serve as a discretization of the character’s mass. Furthermore, each distance is multiplied by the corresponding point’s mass before summing. This tends to give a more physically-biased result, in general tending to give better (lower) scores for state pairs that would result in lower energy consumption to physically perform the displacement.

The distance metric in [LCR$^+$02] draws inspiration from [SSSE00]. Here, P_{ij}, and thus the similarity of frames i and j, is computed as

$$P_{ij} \propto e^{-D_{i,j-1}/\sigma},$$

where

$$D_{ij} = d(p_i, p_j) + wd(v_i, v_j).$$

σ is a parameter that controls the mapping from distance measure to probability, w is a weight factor for the velocity distance term, $d(v_i, v_j)$, which itself is just the L_2 norm of the velocity vector difference, while $d(p_i, p_j)$ is the corresponding configuration difference norm that has special handling for the joint angle components.

motion blending techniques

Once a pair of nearby points have been identified on two trajectories, a transition is implemented, whether through the creation of an additional trajectory or the modification of an existent one. Overall, most of the motion graph work so far uses rather rudimentary methods for effecting transitions. [TH00] uses plain linear interpolation of joint angles. [KGP02] employs an ease-in/ease-out curve to blend root position and spherical linear interpolation ("slerp") on the quaternion joint angles. [AF02] and [LCR+02] both use forms of displacement mapping, which appears to be the most reasonable approach. Other motion editing approaches mentioned in section 3.1 ("motion capture" subsection) would probably do as well. These give satisfactory results for walking type motions, but there seems to be much room for improvement, especially when other motion types are considered, such as jumps and other airborne maneuvers.

constraint fixup techniques

An issue closely related to motion blending is that of constraint fixup. Most blending techniques concentrate only on the character's internal joint angles, which makes it easy to break various constraints, causing sliding footplants, ground penetration, broken hand grasps, etc. Although sometimes it is possible to ignore these artifacts if they are small enough [AF02], most of the time some correcting strategy must be implemented. [LCR+02] use constraint-based motion editing techniques [Gle97, GL98, LS99]. [KGP02], on the other hand, employs a novel humanoid-specific inverse-kinematics method [KGS02] to correct sliding ground contacts. The method is interesting in that it meets the constraints exactly through the lengthening or shortening of the limbs as the final step of the process. Apparently human perception is not very sensitive to such limb variations.

5 Open problems & potential avenues of research

In this section we explore open problems and potential research directions. The ultimate open problem that has been mentioned throughout this paper is that of the virtual actor. Alas, this is a rather complex topic that probably won't be solved for quite some time, as it requires pieces which themselves are still open problems. We describe them below, along with more generic problems that are particular to the various research domains that have been discussed above.

5.1 Motion planning

Much like computer animation, motion planning can be considered to be still in its infancy. Although both fields have yielded many results, these almost always solve a specific class of simplified problems. For motion planning the major stumbling block has been the sheer computational complexity of applying common sense algorithms to non-trivial characters. Randomized methods have to a certain degree mitigated the problem, and we now have a firm, although not yet complete grasp of motion planning for a single character in static environments.

A related problem which has received limited attention and yet is the natural next question is that of motion planning of a number of cooperating characters or agents. Applications for such planners are numerous: interacting or collision-avoiding robotic arms at assembly lines, maneuvering of parties of virtual characters in video games or virtual environments, plane management on decks of aircraft carriers, etc.

These multi-body path planning problems can be solved on many levels, depending on the demands of the particular application. For example, in the context of moving a cohesive group of objects, such as a party of

virtual characters, the simplest approach would be to move one object at a time. If a relatively time-optimal solution is desired, a very likely requirement for most real world applications, this approach is unsatisfactory; even though each object on its own might move in a time-optimal method, the collective group move is far from optimal. The desired solution should move all characters simultaneously, and somehow sequence their passage through narrow passages or evenly distribute the characters over a number of the bottlenecks. This leads to the consideration of character velocities, accelerations, and bounds thereupon. These key factors shape and constrain the character's motion, and thus their consequences must be taken into account in the planning process. In the simplest case, if objects are all traveling at equal speeds, one can just stagger character departures time-wise by sending each subsequent agent a small time interval after the previous one has departed, thus giving a near-optimal solution. The problem starts becoming interesting and non-trivial once the velocity bounds are allowed to vary between characters. The presence of velocity and acceleration as key factors in the problem naturally suggests a kinodynamic motion planning strategy. A likely solution to this, and the multi-body problem in general, is to use the spacetime roadmap method of [KHLR00], with each agent carving out its own extruded path through the space. This still leaves a difficult constrained optimization problem: the extruded paths must be geometrically fitted into $\mathcal{C}_{free}$, must not impinge on each other, and their length along the time axis should be minimized.

Another sensible and straight-forward approach would be to reformulate the problem into a group of related single-character planning problems, with each agent guided by its own path planner, while treating the other agents simply as dynamic obstacles. This requires methods for dealing with unpredictable dynamic obstacles, which itself is an unsolved problem. Some research, such as [KHLR00] deals with dynamic environments but is often not general enough. The above, for example, is limited to linear motion obstacles whose future position and orientation can be easily predicted. A promising first stab at a viable approach might draw inspiration from real life and simply have the character wait a short interval when the planned path becomes blocked, coming to a temporary stop if necessary. If the path obstruction is cleared within some time window, the character can proceed with the old plan, else a new path should be planned based on the new freespace topology.

5.2 Character animation

While the animation of *passive objects*, ones that are controlled only by external forces, is well understood and considered solved, that for *active objects*, that employ internal forces and torques to direct their motion, still remains an open problem. Although we have a multitude of methods for animating such objects or characters, they are generally highly specialized and disparate. Achieving a large repertoire of motions thus requires significant breadth in skill and knowledge, as well as much time and effort in adapting the methods to interact well or combining their separate results. This is not only undesirable, but also puts animation out of reach of the general computer user. Although the specialized methods would still serve a useful purpose in particular applications, when working with animation in general one would ideally like a "Grand Unified Motion Model", a single control parametrization and framework capable of producing all potential motions.[7] Introspection suggests that we use something of the type in our daily lives: executing (and learning) an arbitrary motion generally has the same feel and seems to follow the same process, whether it is a dance step or a martial arts move. Finding such a parametrization is an open problem. DANCE[NF] is a step in the right direction, as it attempts to collect and combine numerous simulation-based controllers, but is nowhere near complete enough yet, and there is no single overall parametrization for generating motion.

Scrutinizing nature's solutions to problems always offers some hints; it's hard to beat millennia of evolution. In probing nature for such a convenient unified model, we can make one observation immediately: considering that all humans learn to walk, run, and perform similar tasks regardless of their mental acuity, it is likely that such a parametrization would be relatively simple. The success of the hopping robots at MIT labs[mit], even though for simpler characters, would seem to corroborate this. Another observation is that optimization

[7] This is intended mostly for whole-body motion; since the face and hands have very high degree of maneuverability, and since their actions are an important part of certain tasks and motions, it is advisable to separate out their control.

plays a key role in our control systems, something which a number of animation methods have already made heavy use of. For example, after a baby is taught the sequencing of limb motions, it then requires prolonged practice to become adept at walking; this is effectively optimization, at first for stability and robustness, and later for energy efficiency. Changes in body parameters, such as attempting to walk on stilts, wearing a heavy backpack, or fast growth during teenage years[8], all immediately cause loss of optimality in motion, which, after practice (i.e. again, re-optimization) is regained. Thus the general motion model might consist of a control parametrization that has parameters itself (i.e., "meta-parameters"), which need to be optimized for a specific character. Carrying the analogy a little further, employing teachers and trainers to accelerate the learning of a task or technique corresponds to the use of supervised learning or guided optimization.

Further observation of human motion suggests one possible model: considering humans spend a large amount of their waking hours on their feet, perhaps one could model their motion using an inverted pendulum. This idea has been explored already in robotics, but has yet to make an appearance in animation, which is odd, since the latter offers more possibilities. Unlike robotics, animation does not have to strictly stick to the laws of physics; rules can be bent, or broken, if that is what it takes to get robust and convincing motion. For example, while the torso is modeled with an inverted pendulum, the feet could be positioned kinematically in the environment to roughly provide the forces and torques requested by the model. By further constructing the pendulum from two telescoping links, the model could also express jumps and other airborne maneuvers. An alternate approach might use the current configuration of the inverted pendulum model as a lookup index into a database of motion capture data, and then using this to supply the motion of the remaining body parts. This is in effect just motion texturing, although instead of using a user-specified sketched motion as the base, one uses a pattern motion obtained from simulation on a very simple physical model. It should be noted thought that the general concept of using simplified models has been already explored to some degree for walking [RH91, PTDP96]. [vdP97, TvdP98] are particularly close in spirit to this idea, although the underlying physical model there is much simpler, and a more direct and intuitive method of control would be needed.

There are also countless open problems in specialized motions and controllers. When considering full-body motions, we for example lack methods to animate certain motions, such as figure skating and ballet. Here, the non-holonomic nature of skate motion makes the animation problem more constrained and complex than is usual. Both skating and ballet also require a certain amount of grace, something a lot of methods, especially controller-driven simulations, are not yet capable of reproducing. We also lack methods for animating various object manipulation motions that involve skill or technique, such as soccer (ball manipulation), handwriting (pen manipulation), sword fighting (sword/weapon manipulation), etc.

Another interesting problem is the animation of dressing and undressing. The difficulty lies in that the motion is highly dependent on garment and body dimensions, possibly requiring drastically different motions when either parameter changes. The problem can be formulated in an interesting way: the task of putting on a shirt, for example, can be seen as motion planning of the hands, head and torso, while the shirt acts as an obstacle, forming a dynamic environment. What's particularly novel about this problem is that here the obstacle (i.e., shirt) is indirectly under the planner's control, through the interaction with the character's body, and its motion must be coordinated with the motion of the character to achieve the end goal. Alternatively, both the character and the garment can be considered as two distinct agents, each under the control of a separate planner, that must cooperate.

Motion specification is yet another large open problem. Current methods use either very verbose or very limited interfaces. Some current methods use simple parameter selection, such as speed and direction. Others accept an example, possibly incomplete motion which they try to match (e.g., acting out a desired motion in [LCR+02], or simplistic keyframing of the motion in [LP02]). Neither seems appropriate for virtual actors, where one would like to direct at task level (e.g., "walk to table", "dribble the ball there and back", "read a book", "dress yourself", etc). The problem is twofold: how to translate such requirements to a form the motion model understands, and how to convey this information in the first place, from human to computer. It seems one should be able to do better than written or spoken English commands.

[8] The cliché of teenagers "tripping over their own feet" is telling.

As motion capture equipment becomes easier and cheaper to obtain, it is natural to expect it to become more widespread. A potential future application for this technology would be in video games in arcade shops as an input device. One computation task that is bound to come into demand is "motion recognition": inferring what the person is doing from their motion data, whether, for example, they are waving, sitting down, or playing air-guitar. By extracting the semantics of the player's motions the computer can interact with him or her on a much more abstract level. One particular difficulty of recognizing motions is that motion capture data is rarely accompanied by the specification of the environment in which it was recorded. This makes identifying some motions difficult. Consider a character sitting down: without any knowledge of whether a chair is present or not, it is hard to tell whether the character is trying to sit down or just squatting. Other uses for motion recognition might include auto-labeling of motion graphs. A motion graph so labeled, for example with annotations such as "sitting down", "waving", etc, might go a long way in facilitating intuitive direction of virtual actors, by allowing motion selection using such labels. Consider how much easier it would be to execute the command: "go to location marked X, wave, and then sit down".

Another interesting application of the above motion recognition would be in performing "behavior capture". By observing a subject's motion and translating it into a sequence of tasks being performed, one can start collecting statistical data on the overall activity. Again, this allows animation using primitives of a higher semantic level. For example, one could characterize the behavior of an office worker using primitives "shuffle papers", "write memo", "drink coffee", "go for chat at water cooler", etc. Behavior capture would also extract the relative frequencies of these actions, as well as the observed transitions. One could then construct a behavior graph, in the spirit of a motion graph, and thus animate such a virtual office worker on a much more general level. The key difference between the two graphs is that the former deals with more abstract primitives, whole actions (e.g., making a bed), whereas the motion graph will usually break up most such actions into short motion segments.

Finally, an interesting open problem that is seeing significant attention is vision-based human motion tracking. Here the character's 3D motion is extracted from 2D data. Since the popularity of motion capture has given birth to a large variety of motion editing methods, and in view of the cost and burden of recording new motion, the potential rewards of reusing the as yet untapped wealth of motion data in stock video footage is irresistible.

5.3 Motion graphs

Since motion graphs are a relatively new field of study, there is a large number of unsolved problems. One of the biggest problems is how to make the graphs more scalable. Current research uses relatively small datasets, and little work has been done on coping with the curse of dimensionality when motion graphs are applied to complex models, or when rich variety in available motions is required (i.e. many variations of each motion). First, a method is needed for removing redundancy in the original motion data that is fed to the graph creation process. More generally, one could develop a method to limit the density of motion data in arbitrary parts of state-space; this would be useful for motion graphs created from unscripted, arbitrary mocap data, where one is likely to encounter excessively frequent actions, such as walking. Further redundancy can be eliminated by decomposing complex motion graphs into a number of smaller ones, each dedicated to a particular, relatively independent body part. A likely decomposition would, for example, have motion graphs for the upper body, lower body, face, and hands. For sections, such as upper and lower body, which are not completely independent, the limited motion correlation could be later reintroduced using heuristic methods.

One could also improve upon the time complexity of the graph creation process; in current methods it suffers from being $O(n^2)$, since the motion transition points are found using a brute force approach, which will likely severely curtail creation of very large motion graphs. Some sort of hierarchical spatial segmentation would likely go a long way in reducing the number of comparisons that need to be made when performing this search.

Much work also remains that pertains to the use of motion graphs. General questions such as how to effectively specify a motion query, or likewise, how to efficiently answer such a query, remain unsolved. [LCR+02] proposes a few sensible query methods, each suited to a particular task, but further user evaluations are needed to assess the effectiveness of these methods. The most intuitive interface presented therein is the performance method, where the user acts out the desired motion. This approach is inherently unsuitable for realtime applications, such as video games, since a certain amount (3 seconds, in the implementation described) of the user's motion must be captured before any reasonable decision can be made about what motion to select from the graph. Other, more responsive, interactive methods should exist. A promising starting point in this search might be previous work on character control using reduced DOF input devices [LvdPF00] or virtual puppetry [OTH02].

A more challenging open problem is the handling of relative coordinate systems in motion graphs. The main benefit of such graphs is that they encode the motions the character is capable of in a very compact form, whereas the fixed coordinate system graphs contain much redundancy, in the form of repeated motions at different locations of the workspace. The problem is particularly interesting in environments containing obstacles. As has been observed in [LCR+02], the relativity makes a number of tasks more difficult, such as avoidance or interaction with objects in the environment. Extending the motion graph to handle variable slope terrain becomes even more complicated. Clustering of similar motions and other motion summary methods likewise become harder to achieve, since the character's relative context must now be taken into account. This problem of relativity is really an instance of a more general one: how does one incorporate and handle parametrized motions in a motion graph? Relative motions are merely ones with a variable parameter for the initial values of the relative DOFs. A simpler parametrized motion that one might want to include in the graphs is, for example, a jumping motion where the character's altitude at the apex is variable. Incorporating such motions is an important topic since the method could serve to further reduce the size of motion graphs.

Another interesting open problem that is bound to attract attention in the near future is how to use motion graphs in realtime, dynamic, and unpredictable environments, such as found in video games. Research so far has only looked at offline planning of motions, where a motion is planned and then executed to its natural end. Many games and simulations, on the other hand, have environments which change rapidly, causing any longer-term plans to quickly become invalid. Furthermore, such dynamic environment applications will frequently place the character in parts of state-space where there is no data. Handling this eventuality is yet another open problem. Some obvious strategies for dealing with this include resorting briefly to physical simulation until motion graph data is rejoined, or using on-the-fly motion editing to adjust existent motion trajectories to the new circumstances.

As has been noted by Kovar *et al.* in [KGP02], some sort of auto-labeling of motion graph edges would be useful. In their context, they were interested in stylistic labeling, with labels such as "walking", "tiptoeing", "prancing", etc. In fact, a more general auto-labeling scheme would be desirable. Allowing arbitrary labeling of edges would likely go a long way in improving user interfaces for query specification, by allowing users to constrain solutions to motions with particular labels. Some potential labelings that one might be interested in are: speed, effort, smoothness, environmental parameters, etc. Some of these qualities constitute open problems themselves; it is not obvious, for example, how to computationally measure effort or grace.

Most of the current motion blending and warping methods have been only applied in the context of human earth-bound motions. These algorithms do not work very well when the character is airborne.[9] Unless the two motions are very similar, the blended motion often does not describe a physically correct motion. They furthermore result in perceived actuation at joints that are not really actuated. How to compensate for these blending shortfalls is not quite clear. Some inspiration might be drawn from work on human perception, such as [OD01]. For instance, it sounds plausible that the blended motion could be sufficiently improved by simply adjusting the character's center of mass' trajectory, such that it follows the familiar parabola. In fact, an interesting future research direction is the design and execution of a series of experiments that help determine what visual artifacts humans are sensitive in motion blends, thus paving a way to better blending algorithms and related tasks.

[9] The only methods that should work relatively well are the SC ones as they directly incorporate physical laws of motion.

A number of interesting applications of motion graphs bear investigation. Motion graphs, for example, might be particularly suited to avalanche simulation. By encoding the dynamics of rock-like objects in a graph, and relying on graph searches being faster than direct simulation, one could obtain avalanche like motion and interaction at a fraction of the computation time. More generally, the topic of the interaction of multiple motion-graph-driven characters is also worth further study.

One application that would be particularly interesting is the use of motion graphs for kinodynamic motion planning. Some of the approaches in that domain, such as the seminal kinodynamic paper by Donald *et al.*[DXCR93], in fact resemble motion graphs that use an absolute coordinate systems. But posing the problem as the equivalent motion graph offers no advantages, since the graph likewise suffers from exponential growth as the state-space area that it must span increases. The potential benefit lies in applying relative coordinate system graphs to this problem, as the graph's size is then independent of the dimensions of the workspace. Unfortunately, trying to find graph walks that displace and rotate the character by particular values can be shown to reduce to the Traveling Salesman Problem[10]. Nonetheless, perhaps one could employ the various heuristic approximate methods that have been proposed for solving TSP. In fact, the recent paper by Arikan&Forsyth[AF02] appears to be doing just that: a randomized graph search method is used to find paths that interpolate the desired states or sets of states at the required time points. The application presented therein is rather rudimentary, and it would be interesting to see this method applied to more typical kinodynamic path planning problems, such as non-holonomic vehicle maneuvering, holonomic motion amid moving obstacles, etc.

5.4 *Dense* motion graphs

The scant motion graph literature at this point deals exclusively with what amounts to *sparse* motion graphs. That is, the motions composing the graph span a small portion of the character's usable state-space[11]. Such graphs in general will only be capable of producing motions similar in character to the training set. This leads one to consider *dense* motion graphs (DMGs), ones which attempt to sample the character's whole usable state-space in a roughly uniform way, or at least a large part of it. How to construct effective dense motion graphs in an efficient manner is an open problem that has received no attention yet. Direct exploratory character simulation would seem to be the most natural way. It would likely be much faster to do this through simulation rather than using a live actor since one can always add hardware to speed up physical simulation while an actor is stuck in realtime.

Since the scalability issue is even more pronounced for dense motion graphs, additional methods of coping with it will be required. One promising avenue is the use of "pruning functions". In most DMG applications, one will usually only be interested in a particular subset of all physically possible motions of the character. For example, in a visualization package of a traffic simulator, one would generally not be interested in motions in which the car and driver are airborne or upside-down. It is the role of the pruning functions to identify and discard from the motion graph such unwanted states and motions leading to them. They ensure that the graph contains only relevant motion data. In a lot of cases, developing an effective pruning heuristics and criteria is a non-trivial problem [HvdP96].

A number of potential advantages of DMGs stem directly from the construction method, viz., exploration using simulation. First and foremost, DMGs are particularly useful for characters or actions which cannot be recorded using motion capture (e.g., fictitious three legged beasts, dangerous stunts, etc). Second, the stochastic sampling and exploration automatically discover the character's usable state-space. Furthermore, if combined with a pruning criteria that reject falling motions or fallen configurations, the graph automatically describes the part of state-space in which the character has dynamic balance. This alone could be useful, and leveraged by other algorithms that currently use static balance checking. In addition, if such an external algorithm found that its character is balanced dynamically but with the center of mass outside the support

[10]Private communication with Prof. Molloy, University of Toronto, 2002

[11]"usable state-space": the portions of state-space which the character is able to reach during normal operation

polygon, it could find out from the motion graph how to "rescue" the character, and bring it back to static balance.

The "dense" aspect also makes DMGs a reasonable physical simulator replacement. Since a given motion probably has more variations available in the dense graph, any interpolations are likely to be more correct. Also, since the dense graph models the character's motion capabilities more accurately in the areas of interest, one should be able to answer interesting questions by using graph searches, questions like "find the minimal energy walk cycle", "how high can the character jump", etc.

One of the critical areas of the DMG building process will be physical simulation. In order to get efficient and thorough coverage of the state-space, an effective exploration methodology is needed. [DXCR93, LK99] offer a few suggestions; generally this consists of alternating between picking random control inputs, and forward simulating to discover new reaches of state-space from which to explore again. The behavior of this exploration process highly depends on how the character's forces and torques are generated. A very popular method of actuating character joints is the use of *proportional derivative* (PD) controllers. A PD controller implements the character's muscles as spring and damper systems. When a goal angle ("setpoint") is set by a higher level control program, the controller exerts a torque that attempts to smoothly minimize, and eventually eliminate the angular difference between current angle and setpoint. Although very simple and easy to implement, the PD controller has some undesirable properties. The key problem is that, for a single setpoint, the controller's torque always follows an exponential decay curve to the new value. This limits the behaviors the controller can express. One could obtain arbitrary torque curves by rapidly varying the setpoint, but figuring out how to vary it such that a rich variety of torque curves is obtained is not at all clear. Since in the end one is only concerned about getting random torque behavior when exploring the state-space, a more direct method of generating torques would be appropriate. One promising approach is to have the controller model the torque function directly with a spline, thus intrinsically allowing an arbitrary, smooth shape. Random torque curves can be obtained simply by randomly picking the four spline curve parameter values.

Furthermore, regular PD actuators are unrealistic in that all their motions are actuated. In contrast, in the real world we often let gravity do as much of the work as possible, often using muscles only to control or stop a passive motion already in progress. An interesting problem would be to develop a "smart-" or "lazy-PD" controller that imitates this behavior, discouraging unnecessary energy expenditure. In this case, a higher level control program would specify a joint setpoint, as well as a time-window in which to achieve it; the controller would then remain passive until either the setpoint was in close proximity, or if it determined that attaining the setpoint with passive motion is not possible in the allotted time. Such controllers should produce more natural motion.

References

[AF02] Okan Arikan and D. A. Forsyth. Interactive motion generation from examples. *Proceedings of SIGGRAPH*, 2002.

[BL91] Jérôme Barraquand and Jean-Claude Latombe. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10(6):628–649, December 1991.

[Bow00] Richard Bowden. Learning statistical models of human motion. *IEEE Workshop on Human Modeling, Analysis & Synthesis*, July 2000.

[BW95] Armin Bruderlin and Lance Williams. Motion signal processing. *Computer Graphics Proceedings*, 1995.

[Coh92] Michael F. Cohen. Interactive spacetime control for animation. *Computer Graphics*, pages 293–302, 1992.

[DX95] Bruce Randall Donald and Patrick G. Xavier. Provably good approximation algorithms for optimal kinodynamic planning: Robots with decoupled dynamics bounds. In *Algorithmica*, pages 443–479, 1995.

[DXCR93] Bruce Randall Donald, Patrick G. Xavier, John F. Canny, and John H. Reif. Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066, 1993.

[FvdPT01] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Composable controllers for physics-based character animation. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 251–260. ACM Press / ACM SIGGRAPH, 2001.

[GL98] Michael Gleicher and Peter Litwinowicz. Constraint-based motion adaptation. *The Journal of Visualization and Computer Animation*, 9(2):65–94, 1998.

[Gle97] Michael Gleicher. Motion editing with spacetime constraints. *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, 1997.

[Gle98] Michael Gleicher. Retartgetting motion to new characters. *Computer Graphics Proceedings*, 1998.

[HKL+98] David Hsu, Lydia E. Kavraki, Jean-Claude Latombe, Rajeev Motwani, and Stephen Sorkin. On finding narrow passages with probabilistic roadmap planners. *Proceedings of the International Workshop on Algorithmic Foundations of Robotics*, 1998.

[HKLR00] David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Randomized kinodynamic motion planning with moving obstacles. *The Fourth International Workshop on Algorithmic Foundations of Robotics*, 2000.

[HLM99] David Hsu, Jean-Claude Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. *International Journal of Computational Geometry & Applications*, 1999.

[HP97] Jessica K. Hodgins and Nancy S. Pollard. Adapting simulated behaviors for new characters. *Computer Graphics Proceedings*, pages 153–162, 1997.

[HSL92] Jessica K. Hodgins, Paula K. Sweeney, and David G. Lawrence. Generating natural-looking motion for computer animation. *Proceedings of Graphics Interface*, 1992.

[HvdP96] Pedro S. Huang and Michiel van de Panne. A planning algorithm for dynamic motions. In *Computer Animation and Simulation '96*, pages 169–182, 1996.

[HWBO95] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating human athletics. *Computer Graphics Proceedings*, pages 71–78, 1995.

[Kal99] Maciej Kalisiak. A grasp-based motion planning algorithm for intelligent character animation. Master's thesis, University of Toronto, 1999. Available online at `http://www.dgp.utoronto.ca/~mac/thesis`.

[KGP02] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. *Proceedings of SIGGRAPH*, 2002.

[KGS02] Lucas Kovar, Michael Gleicher, and John Schreiner. Footskate cleanup for motion capture editing. *ACM SIGGRAPH Symposium on Computer Animation*, 2002.

[KHLR00] Robert Kindel, David Hsu, Jean-Claude Latombe, and Stephen Rock. Kinodynamic motion planning amidst moving obstacles. *Proceedings of IEEE International Conference on Robotics and Automation*, 2000.

[KL00] James J. Kuffner, Jr. and Steven M. LaValle. RRT-connect: An efficient approach to single-query path planning. *Proceedings of IEEE International Conference on Robotics and Automation*, 2000.

[KŠLO96] Lydia E. Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[Kuf98] James J. Kuffner Jr. Motion planning with dynamics, March 1998. Physiqual.

[KvdP00] Maciej Kalisiak and Michiel van de Panne. A grasp-based motion planning algorithm for character animation. *Computer Animation and Simulation 2000*, August 2000.

[KvdP01] Maciej Kalisiak and Michiel van de Panne. A grasp-based motion planning algorithm for character animation. *The Journal of Visualization and Computer Animation*, 12(3):117–129, 2001.

[Lat91] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[LCR+02] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive control of avatars animated with human motion data. *Proceedings of SIGGRAPH*, 2002.

[LGC94] Zicheng Liu, Steven J. Gortler, and Michael F. Cohen. Hierarchical spacetime control. *Computer Graphics Proceedings*, pages 35–42, 1994.

[LK99] Steven M. LaValle and James J. Kuffner, Jr. Randomized kinodynamic planning. *Proceedings of IEEE International Conference on Robotics and Automation*, 1999.

[LP02] C. Karen Liu and Zoran Popović. Synthesis of complex dynamic character motion from simple animations. *Proceedings of SIGGRAPH*, 2002.

[LS99] Jehee Lee and Sung Yong Shin. A hierarchical approach to interactive motion editing for human-likefigures. In Alyn Rockwood, editor, *Siggraph 1999, Computer Graphics Proceddings*, pages 39–48, Los Angeles, 1999. Addison Wesley Longman.

[LvdP96] Alexis Lamouret and Michiel van de Panne. Motion synthesis by example. In *Computer Animation and Simulation '96*, pages 199–212, 1996.

[LvdPF00] Joseph Laszlo, Michiel van de Panne, and Eugene Fiume. Interactive control for physically-based animation. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 201–208. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[LWS02] Yan Li, Tianshu Wang, and Heung-Yeung Shum. Motion texture: A two-level statistical model for character motion synthesis. *Proceedings of SIGGRAPH*, 2002.

[MBC01] Mark Mizuguchi, John Buchanan, and Tom Calvert. Data driven motion transitions for interactive games. *Eurographics 2001 Short Presentations*, 2001.

[mit] MIT leg lab's planar biped robot. `http://www.ai.mit.edu/projects/leglab/robots/2D_biped/2D_biped.html`.

[MZ90] Michael McKenna and David Zeltzer. Dynamic simulation of autonomous legged locomotion. *Computer Graphics*, 24(4):29–38, 1990.

[NF] Victor Ng and Petros Faloutsos. Dynamic animation and control environment (DANCE). `http://www.cs.ucla.edu/magix/projects/dance/`.

[Nil69] N. J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, pages 509–520, 1969.

[NM93] J. Thomas Ngo and Joe Marks. Spacetime constraints revisited. *Computer Graphics Proceedings*, pages 343–350, 1993.

[OD01] Carol O'Sullivan and John Dingliana. Collisions and perception. *ACM Transaction on Graphics*, 20(3), July 2001.

[OTH02] Sageev Oore, Demetri Terzopoulos, and Geoffrey Hinton. A desktop input device and interface for interactive 3D character animation. *Graphics Interface*, 2002.

[PB02] Katherine Pullen and Christoph Bregler. Motion capture assisted animation: Texturing and synthesis. *Proceedings of SIGGRAPH*, 2002.

[Per95] Ken Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1995.

[PTDP96] Jerry Pratt, Ann Torres, Peter Dilworth, and Gill Pratt. Virtual actuator control. *IEEE International Conference on Intelligent Robots and Systems*, 1996.

[RGBC96] Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael F. Cohen. Efficient generation of motion transitions using spacetime constraints. *Computer Graphics*, 30:147–154, 1996.

[RH91] Marc H. Raibert and Jessica K. Hodgins. Animation of dynamic legged locomotion. *Computer Graphics*, 25(4):349–358, 1991.

[Sim94a] Karl Sims. Evolving 3D morphology and behavior by competition. *Artificial Life IV Proceedings*, pages 28–39, 1994.

[Sim94b] Karl Sims. Evolving virtual creatures. *Computer Graphics Proceedings*, 1994.

[SL01] Gildardo Sánchez and Jean-Claude Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. *International Symposium on Robotics Research*, 2001.

[SSSE00] Arno Schödl, Richard Szeliski, David H. Salesin, and Irfan Essa. Video textures. *Proceedings of SIGGRAPH 2000*, pages 489–498, 2000.

[TH00] Luis Molina Tanco and Adrian Hilton. Realistic synthesis of novel human movements from a database of motion capture examples. *Proceedings of the IEEE Workshop on Human Motion*, 2000.

[TvdP98] Nick Torkos and Michiel van de Panne. Footprint-based quadruped motion synthesis. *Proceedings of Graphics Interface*, pages 151–160, June 1998.

[UAT95] Munetoshi Unuma, Ken Anjyo, and Ryozo Takeuchi. Fourier principles for emotion-based human figure animation. *Computer Graphics Proceedings*, 1995.

[vdP96] Michiel van de Panne. Parametrized gait synthesis. *IEEE Computer Graphics and Applications*, 16(2):40–49, March 1996.

[vdP97] Michiel van de Panne. From footprints to animation. *Computer Graphics Forum*, 16(4):211–224, 1997.

[vdPF93] Michiel van de Panne and Eugene Fiume. Sensor-actuator networks. *Computer Graphics Proceedings*, 1993.

[vdPFV90] Michiel van de Panne, Eugene Fiume, and Zvonko Vranesic. Reusable motion synthesis using state-space controllers. *Computer Graphics*, 1990.

[vdPKF94] Michiel van de Panne, Ryan Kim, and Eugene Fiume. Virtual wind-up toys for animation. *Proceedings of Graphics Interface*, pages 208–215, 1994.

[vdPL95] Michiel van de Panne and Alexis Lamouret. Guided optimization for balanced locomotion. *Computer Animation and Simulation '95 – Proceedings of the 6th Eurographics Workshop on Simulation and Animation*, pages 165–177, September 1995.

[WK88] Andrew Witkin and Michael Kass. Spacetime constraints. *Computer Graphics*, 22(4):159–168, 1988.

[WP95] Andrew Witkin and Zoran Popović. Motion warping. *Computer Graphics Proceedings*, 1995.