Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2005)
K. Anjyo, P. Faloutsos (Editors)

Behavior Planning for Character Animation

Manfred Lau[1] and James J. Kuffner[1,2]

{mlau,kuffner}@cs.cmu.edu
[1] School of Computer Science, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA., 15213, USA.
[2] Digital Human Research Center, National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan.

Abstract

This paper explores a behavior planning approach to automatically generate realistic motions for animated characters. Motion clips are abstracted as high-level behaviors and associated with a behavior finite-state machine (FSM) that defines the movement capabilities of a virtual character. During runtime, motion is generated automatically by a planning algorithm that performs a global search of the FSM and computes a sequence of behaviors for the character to reach a user-designated goal position. Our technique can generate interesting animations using a relatively small amount of data, making it attractive for resource-limited game platforms. It also scales efficiently to large motion databases, because the search performance is primarily dependent on the complexity of the behavior FSM rather than on the amount of data. Heuristic cost functions that the planner uses to evaluate candidate motions provide a flexible framework from which an animator can control character preferences for certain types of behavior. We show results of synthesized animations involving up to one hundred human and animal characters planning simultaneously in both static and dynamic environments.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism:Animation

1. Introduction

Creating realistic motion for animated characters is an important problem with applications ranging from the special effects industry to interactive games and simulations. The use of motion capture data for animating virtual characters has become a popular technique in recent years. By capturing the movement of a real human and replaying this movement on a virtual character, the resulting motion exhibits a high degree of realism. However, it can be difficult to re-use existing motion capture data, and to adapt existing data to different environments and conditions.

This paper presents a behavior planning approach to automatically generate realistic motions for animated characters. We first organize motion clips into an FSM of behaviors. Each state of the FSM contains a collection of motions representing a high-level behavior. Given this behavior FSM and a pre-defined environment, our algorithm searches the FSM and plans for a sequence of behaviors that allows the character to reach a user-specified goal. The distinguishing features of our approach are the representation of motion as abstract high-level behaviors, and the application of a global planning technique that searches over these behaviors.

We represent sequences of motion capture clips as *high-level behaviors*, which are then connected together into an FSM of behaviors. There are a number of techniques that generate motion from a graph-like data structure built from motion capture data [AF02, KGP02, LCR*02]. These methods utilize search techniques that explore links between large databases of individual poses of motion data. In contrast, our planning algorithm searches over the behavior states of the

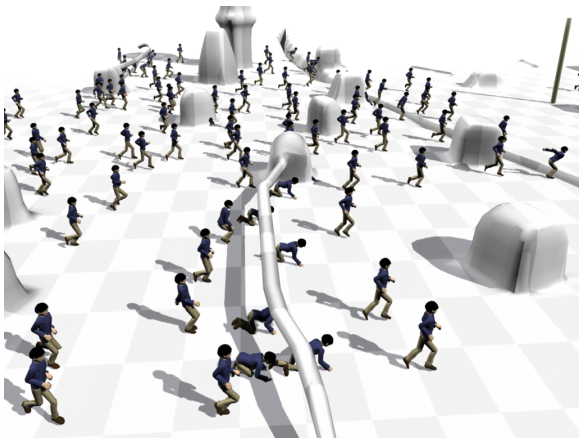


Figure 1: *Planned behaviors for 100 animated characters navigating in a complex dynamic environment.*

FSM. Each state consists of a collection of long sequences of individual poses that represent a high-level behavior. The output of the planner is a sequence of behaviors that moves the character to a specified goal location. Our planner does not need to know the details of the underlying motion or poses. This abstract representation of motion offers a number of advantages over previous methods detailed in Section 2 of the paper.

Our planning technique synthesizes motion by performing a *global search* of the behavior FSM. Previous methods synthesize final motion by using a *local policy* mapping from state to motion. Gleicher et al.'s [GSKJ03] method builds a data structure of motion clips and then replays these clips to generate motion. At runtime, the graph of motions is used as a policy that specifies a motion for the character depending on the state the character is in. The game industry uses *move trees* [Men99, MBC01] to generate motion for human-like characters. These approaches are reactive, and typically make decisions based only on local information about the environment. In contrast, our approach plans and searches through the FSM to explore candidate behavior sequence solutions. It builds a search tree of the states in the FSM, and considers many different possible paths in a global sense before deciding on the final output motion. This enables our algorithm to avoid many of the pitfalls that purely reactive techniques are prone to, such as escaping local minimas in the environment.

2. Background and Motivation

A number of methods have been proposed in the animation literature for generating motion for synthetic characters. Because virtual environments require continuous streams of motion, approaches that only create individual, static clips cannot be readily utilized. These include keyframing, motion capture editing [BW95, WP95, Gle98, LS99, PW99], and motion interpolation [WH97, RCB98]. By contrast, procedural animation techniques can generate arbitrarily long streams of motion. These include behavior scripting [PG96] and physically based methods that simulate natural dynamics [LP02, HWBO95, SHP04, FP03]. The key issues with using physically-based techniques for interactive environments have been high computational costs, and providing the appropriate amount of high-level control.

A variety of planning and search techniques have been used previously to create meaningful movements for animated characters. Planning approaches that preprocess static environment geometry with graph structures, and subsequently use motion capture data can produce human-like motion [CLS03, PLS03]. Preprocessing the environment with a roadmap has also been used for generating flocking behaviors [BLA02]. Animations of object manipulation tasks have been synthesized using planning techniques [KKKL94, YKH04]. Planning algorithms have also been used to generate cyclic motions such as walking and crawling [SYN01]. Techniques incorporating the search of control actions or motion primitives include examples involving grasp primitives [KvdP01], various body controllers [FvdPT01], and precomputed vehicle trajectories [GVK04].

Our work is closely related to recent techniques [AF02, KGP02, LCR*02, PB02, GSKJ03] that build graph-like data structures of motions. These approaches facilitate the re-use of large amounts of motion capture data by automating the process of building a graph of motion. In contrast, our algorithm requires the existence of a behavior FSM and motion data that has been appropriately segmented. However, we have found that it is not difficult to construct and re-use our FSMs because of their small size. More importantly, we believe that by abstracting the raw motion data into high-level behaviors, our approach offers a number of advantages over previous techniques:

Scalability and Efficiency: Our approach can scale to a large amount of data. Gleicher et al. [GSKJ03] described how unstructured motion graphs can be inappropriate for interactive systems that require fast response times. Lee et al. [LL04] explains similarly that for a large motion set, the time required for searching a motion graph is the bottleneck of the method. The number of behavior states in our FSMs (25 in our largest example) is relatively small compared to motion graph approaches, which typically have on the order of thousands of nodes corresponding to individual poses of motion (and potentially tens of thousands of edges). Because of the small size of our FSM, and the fact that our branching factor can be very small compared to that for unstructured graphs, our planner is able to generate long animation sequences very efficiently. Moreover, once we have an FSM that is large enough to produce interesting motions, the inclusion of additional motion data to existing behavior states does not change the complexity; it will only add to the variety of the synthesized motions. In Figure 2 for example, the addition of another *jog left* motion clip leads to the same FSM. This is in contrast to unstructured motion graph approaches [KGP02, LCR*02] that require recomputation of the graph if an additional motion data is added.

Memory Usage: Our method requires a relatively small amount of data in order to generate interesting motions, making it particularly appealing to resource-limited game systems. As an example, our synthesized horse motions (Figure 8) are generated from only 194 frames of data. While these motions do not have as much variety as the synthesized human motions, they may be appropriate for simple characters in some games.

Intuitive Structure: Because the FSM of behaviors is well-structured, the solutions that the planner returns can be understood intuitively. The high-level structure of behaviors makes it easier for a non-programmer or artist to understand and work with our system. For example, a virtual character that wants to retrieve a book from inside a desk in another room needs to do the following: exit the room it is

in, get to the other room, enter it, walk over to the desk, open the drawer, and pick up the book. It is relatively difficult for previous techniques to generate motion for such a long and complex sequence of behaviors. Because the FSM effectively partitions the motions into distinct high-level behaviors, planning a solution and synthesizing a resulting animation can be performed in a natural way.

Generality: We can apply our algorithm to different characters and environments without having to design new behavior FSMs. For example, we generated animations for a skateboarder and a horse (Figure 8) using essentially the same FSM (Figure 3). In addition, by representing the motion data at the behavior level, there is no dependence on any particular environment structure. While others have demonstrated synthesized motion for a single character navigating in a flat empty environment or heavily preprocessed terrain, we show examples of up to 100 characters moving simultaneously in a variety of complex dynamic environments.

Optimality: Our method computes optimal sequences of behaviors. The maze example in Lee et al. [LCR*02] uses a best first search technique to synthesize motion for a character to follow a user sketched path. The user specified path is useful for providing high-level control of the character. However, the greedy nature of these searches may cause unexpected or undesirable deviations from the path. Our approach overcomes this limitation because our FSM is small enough to perform optimal planning. In some cases, generating optimal sequences of motion may be undesirable. For these situations, we can relax the optimality criteria and use other non-optimal search techniques inside the planning module. In addition to providing optimality, carefully designed behavior FSMs can provide coverage guarantees, which is an issue for unstructured graphs [RP04]. Unstructured graphs have no pre-determined connections between motions, and can make no guarantees about how quickly one motion can be reached from another.

Anytime Algorithm: For game systems with limited CPU resources and real-time constraints, our planning algorithm can be interrupted at any time and asked to return the best motion computed up until that point as in [GVK04].

3. Behavior Planning

We explain our behavior planning approach in more detail. The algorithm takes as input an FSM of behaviors, information about the environment, and starting and goal locations for the character. It uses an A*-search planner [HNR68] to find a sequence of behaviors that allows the character to move from the start to the goal.

3.1. Behavior Finite-State Machine

The behavior FSM defines the movement capabilities of the character. Each *state* consists of a collection of motion clips that represent a high-level behavior, and each directed edge

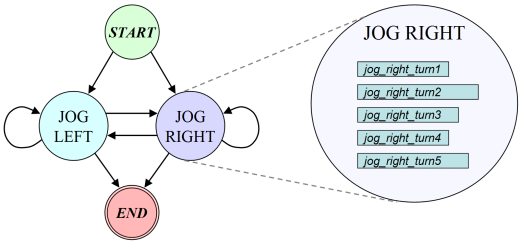


Figure 2: *A simple FSM of behaviors. Arcs indicate allowable transitions between behavior states. Each state contains a set of example motion clips for a particular behavior.*

represents a possible transition between two behaviors. Figure 2 shows a simple FSM. The *start* state allows the character to transition from standing still to jogging, and the *end* state allows the character to transition from jogging to standing still. We define a *behavior* to be the same as a state. However, we may have more than one state labeled, for example, "jog left". This is because it is possible to have multiple "jog left" states, each with different transitions and different connections within the FSM.

There can be multiple motion clips within a state. Having multiple clips that differ slightly in the style or details of the motion adds to the variety of the synthesized motions, especially if there are many characters utilizing the same FSM. However, clips of motions in the same state should be fairly similar at the macro scale, differing only in the subtle details. For example, if a "jog left" clip runs a significantly longer distance than another "jog left" clip, they should be placed in different states and assigned different costs.

Individual clips of motions may be extracted from longer clips of motion. Each motion clip has transition labels that correspond to the first and last frame of every clip along with their nearby frames, taking the velocities of the character into account. Transitions are possible if the end of one clip is similar to the beginning of another. It is advantageous to have the beginning and end of every clip to be similar. This means that every clip would be able to transition to all the others, allowing for a larger variety in the number of possible output motions. In practice, we have found that it is a good idea to include some motion clips that are relatively short compared to the length of the expected solutions. This makes it easier for the planner to globally arrange the clips in a way that avoids the obstacles even in cluttered environments.

Figure 3 shows an example of the FSM used for the human-like character. The most complicated FSM that we used has a similar structure, except for: (1) additional jogging states mostly connected to each other; and (2) more specialized behaviors such as jumping. We captured our human motion data using a Vicon optical system, at a frequency of 120 Hz. We used seven types of jogging behaviors: one moving forward, three types of turning left, and three of turning

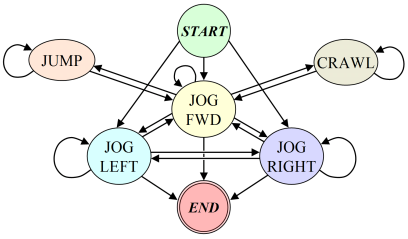


Figure 3: *An example FSM used for a human character that includes special jumping and crawling behavior states.*

right. In addition to "jump" and "crawl", there are also states for: duck under an overhanging obstacle, different types of jumps, and stop-and-wait. The "stop-and-wait" behavior allows the character to stop and stand still for a short while during a jog. These behaviors all have the same transition labels. We also have a few jogging behaviors that are relatively shorter in length; some of these have different transition labels from the other states at either the beginning or the end of the clip. For our experiments, the raw motion capture data was downsampled to 30 Hz. Our most complicated FSM has 1648 frames of data, 25 states, and 1 to 4 motion clips per state. Each of the seven types of jogging mentioned above has about 25 frames of data per motion clip. Each of the specialized behaviors has about 40 to 130 frames, and each of the shorter jogging states has about 15 frames. In addition, there are 8 frames of data before and after each motion clip that are used for blending.

The FSMs for the skateboarder and the horse are similar to the one in Figure 3. For the skateboarder, there are five gliding behaviors: one moving forward, two types of left turns, and two types of right turns. In addition, there are states for jumping, ducking, and stopping-and-waiting. All of the motion clips have the same transition labels. The FSM has 835 frames of data, 11 states, and 1 motion clip per state. Each motion clip has about 60 frames of data, and an additional 16 frames used for blending. For the horse, we only had access to a single keyframed motion of a forward gallop. We defined a total of five galloping behaviors: one moving forward, two types of turning left, and two turning right. All turning motions were keyframed from the forward motion. The FSM has 194 frames of data, 5 states, and 1 motion clip per state. Each of the clips consists of 20 frames of data, and an additional 12 frames used for blending.

3.2. Environment Abstraction

For *static environments*, we represent the environment e as a 2D heightfield gridmap. This map encodes the obstacles that the character should avoid, the free space where the character can navigate, and information about special obstacles such as an archway that the character can crawl under. This information can be computed automatically given the arrangement of obstacles in a scene. The height value is used so that we can represent terrains with slopes or hills. For each of the special obstacles, we compute: a set of *near regions* where the character is near the obstacle and some special motions such as jumping can be performed, and a *within region* where the character can be in the process of executing the special motions. We assume our environments are bounded by obstacles that prevent the character from navigating into infinite space.

The virtual character is bounded by a cylinder with radius r. The character's root position is the center of this cylinder. The character is not allowed to go anywhere within a distance r of an obstacle. As is standard in robot path planning, we enlarge the size of the obstacles by r so that the character can then be represented as a point in the gridmap [LaV].

Each of the special motions such as crawling need to be pre-annotated with the type of corresponding special obstacle. In addition, the motion clips that are more complex can be optionally pre-annotated with the time that the special motion is actually executed. For example, a long jumping motion clip where the character might take a few steps before the jump can be annotated with the time where the jump actually takes place. If there is no such annotation, we can simply assume that the jump occurs in the middle of the motion clip.

Our algorithm handles *dynamic environments*, given that we know a priori how each obstacle moves. Given the motion trajectories of all the moving objects, we define a function $E(t)$ that given a time t, returns the environment e at that time. For static environments, this function is constant. Note that if the motion of moving objects is unknown, the planner can utilize estimates of the future trajectories (e.g. by extrapolation) to predict their future positions and plan accordingly. Our method is generally fast enough to allow iterative replanning should these predictions turn out to be wrong.

3.3. Behavior Planner

The search algorithm uses two interrelated data structures: (1) a tree with nodes that record explored states in the FSM that is continually expanded during the search; and (2) a priority queue of FSM states ordered by cost, which represent potential nodes to be expanded during the next search iteration. Each node in the tree stores the motion clip or action a chosen at that state, and the position, orientation, time, and cost. This means that if we choose the path from the root node of the tree to some node n, the position stored in n corresponds to the character's global position if it follows the sequence of actions stored along the path. The purpose of the queue is to select which nodes to expand next by keeping track of the cost of the path up to that node and expected cost to reach the goal. The priority queue can be implemented efficiently using a heap data structure.

In order to get the position, orientation, time, and cost at each node during the search, we first compute automatically

Algorithm 1: BEHAVIOR PLANNER

```
Tree.Initialize(s_init);
Queue.Insert(s_init,DistToGoal(s_init,s_goal));
while !Queue.Empty() do
    s_best ← Queue.RemoveMin();
    if GoalReached(s_best,s_goal) then
        return s_best;
    end
    e ← E(s_best.time);
    A ← F(s_best, e);
    foreach a ∈ A do
        s_next ← T(s_best, a);
        if G(s_next, s_best, e) then
            Tree.Expand(s_next,s_best);
            Queue.Insert(s_next,DistToGoal(s_next,s_goal));
        end
    end
end
return no possible path found;
```

the following information for each action a: (1) the relative change in the character's root position and orientation $(x(a), y(a), \theta(a))$, (2) the change in time $t(a)$ (represented by the change in number of frames), and (3) the change in cost $cost(a)$.

Pseudocode for the behavior planner is shown in Algorithm 1. The planner initializes the root of the tree with the state s_{init}, which represents the starting configuration of the character at time $t = 0$. Most of our synthesized results were generated using A* search, so the total cost is the sum of the cost of the path up to that node and the expected cost to reach the goal (DistToGoal). In addition to A*-search, we also experimented with using truncated A* and inflated A* search. For inflated A*, we set the relative weight for the estimated cost to reach the goal to be twice the weight of the cost of the path taken so far. The planner iteratively expands the lowest cost node s_{best} in the queue until either a solution is found, or until the queue is empty, in which case there is no possible solution. If s_{best} reaches s_{goal} (within some small tolerance ε), then the solution path from the root node to s_{best} is returned. Otherwise, the successor states of s_{best} are considered for expansion.

The function F returns the set of actions A that the character is allowed to take from s_{best}. This set is determined by the transitions in the FSM. Some transitions may only be valid when the character's position is in the *near regions* of the special obstacles. Moreover, F can add more variety to the synthesized motion by randomly selecting a motion clip within each chosen state, if there are multiple clips in a state.

The function T takes the input state s_{in} and an action a as parameters and returns the output state s_{out} resulting from the execution of that action (Equation 1). The function f represents the translation and rotation that may take place for each clip of motion. The cost of each clip is computed by the distance that the root position travels multiplied by a user weight. The distance that the root position travels is estimated by the Euclidean distance between the start and end frame projections onto the ground plane. If there are multiple clips in a state, their costs should be similar; otherwise they should be in different states. Each state has only one cost. For multiple clips in the same state, we take the average of the cost of each clip. The search algorithm is optimal with respect to the states' cost.

$$\begin{aligned} s_{out}.pos &= s_{in}.pos + f(s_{in}.ori, x(a), y(a), \theta(a)) \\ s_{out}.ori &= s_{in}.ori + \theta(a) \\ s_{out}.time &= s_{in}.time + t(a) \\ s_{out}.cost &= s_{in}.cost + cost(a) \end{aligned} \quad (1)$$

The user weights assigned to each action correspond to the character's preference for executing a particular action. For example, the weights for jogging-and-turning are set slightly higher than the weights for jogging forward. The weights for jumping are higher than any jogging motions, reflecting the relatively higher effort required to jump. Hence the character will prefer to jog rather than jump over the same distance whenever jogging is possible. The stopping and waiting motions have the highest weights; the character should prefer the other motions unless it is much better to stop and wait for a short time. There is also an additional cost for walking up or down a sloped terrain. This makes it more preferable to choose a path that is flat, if such a path exists, than one that requires traversing a rough terrain. The user can easily change these relative weights in order to define a particular set of preferences for the character's output motion. Because the number of behavior states in the FSM is relatively small, we have not found parameter tuning to be a major issue in practice.

The function G determines if we should expand s_{next} as a child node of s_{best} in the tree. First, collision checking is performed on the position of s_{next}. This also checks the intermediate positions of the character between s_{best} and s_{next}. The discretization of the positions between these two states should be set appropriately according to the speed and duration of the action. The amount of discretization is a tradeoff between the search speed and the accuracy of the collision checking. For the special actions such as jumping, we also check to see if the character is inside the *within regions* of any corresponding obstacles during the execution of the action. In the case of a jumping motion, for example, since we have annotated when the jump occurs, we can add this time to the accumulated time at that point (s_{best}.time) and use the total time to index the function E.

As a final step for function G, we utilize a state-indexed table to keep track of locations in the environment that have previously been visited. If the global position and orientation of a potential node s_{next} has been visited before, the function G will return false, thereby keeping it from being

expanded. This prevents the search from infinitely cycling between different previously explored positions. Hence the algorithm will terminate in finite time if no solution exists.

3.4. Motion Generation and Blending

After the search algorithm returns a sequence of behaviors, we convert that sequence into an actual motion for the character. In order to smooth out any slight discontinuities where the transitions between behaviors occur, we linearly interpolate the root positions and use a smooth-in, smooth-out slerp interpolation function for the joint rotations. More sophisticated blending techniques could be used, such as [RCB98,GSKJ03]. We use the height map of the terrain to adjust the elevation of the character; we smooth these elevation values by averaging any adjusted value with those in the neighboring frames. It is also possible to utilize constraints or inverse kinematics techniques to fix the locations of certain end effectors in the skeleton, although we did not perform this in our experiments.

We synthesized motions for multiple characters by doing prioritized planning. We plan the motion for the first character as usual; each additional character's motion is then synthesized by assuming that all previous characters are moving obstacles. Prioritized planning does not guarantee a *globally optimal* solution for a given group of characters, as solving this multi-agent planning problem is known to be PSPACE-hard [LaV]. Although it is neither fully general nor optimal, we have found that prioritized planning is efficient and performed very well in our experiments. However, if the environment contains narrow tunnels or bridges that will generate path conflicts and impasse, then prioritized planning may not be appropriate.

4. Results

Figure 4 shows an example of a search tree generated by the planner. We used here the behavior FSM for the human-like character described in Section 3.1. The larger green sphere is the starting position of the character, the smaller green sphere represents the starting orientation, and the large red sphere is the goal position. The obstacle near the bottom of the figure and two of the smaller obstacles near the middle are rendered at a lower height than they really are. This was done to show the search nodes more clearly. The long obstacle on the right side of the figure has two parts: one where the character must crawl under, and one where the character must jump over. The points that are plotted represent the nodes of the tree, which are the states in the FSM. The tree in Figure 4 has 2241 nodes. Each point's location is the projection of the character's root position onto the ground. The color represents the total cost of each node, and it ranges from blue to red.

We compare the search trees generated for A*-search, truncated A*, and inflated A* search (Figure 5). The tree for

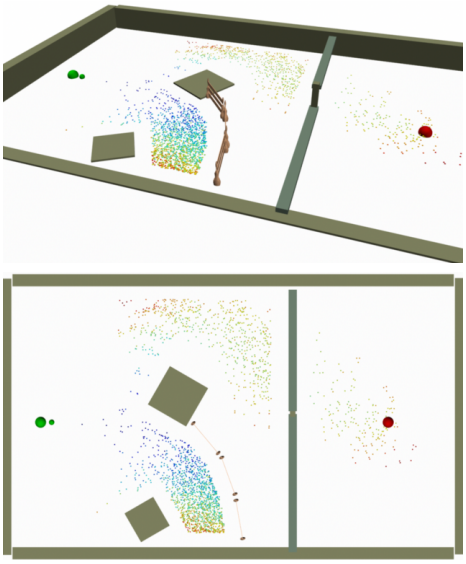
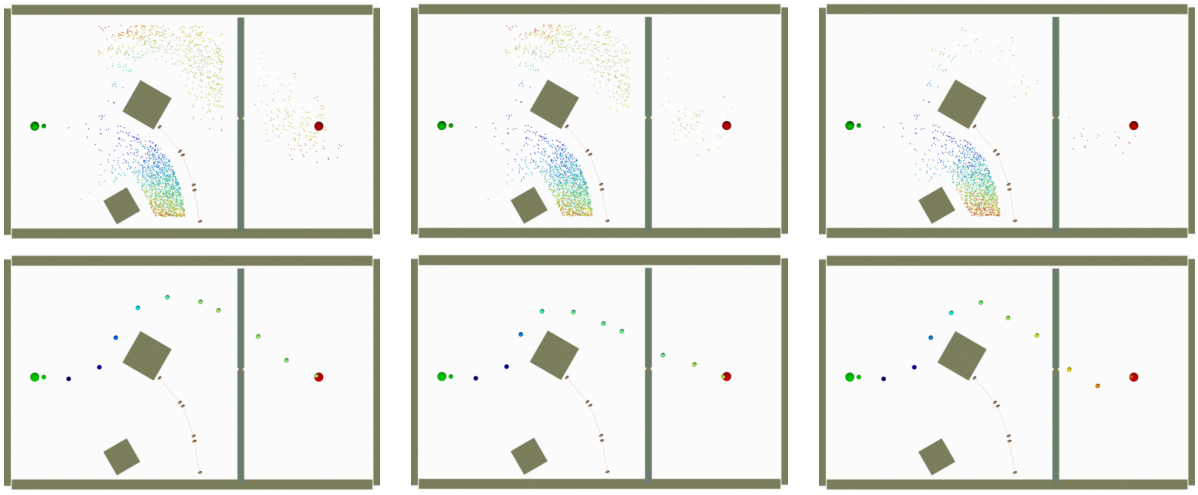


Figure 4: *Perspective and top views of an example search tree generated by the A* search planner (2241 nodes).*

truncated A* has 1977 nodes, while the one for inflated A* has 1421. The three searches each took less than 0.4 seconds, and they all spend a large proportion of the time exploring the area where there is a local minimum. The bottom row of images shows the final paths the planner returned. Each plotted point represents the end of the selected motion clip, or the global position of the corresponding node in the search tree. The three final paths look fairly similar, except for the last part of the path for inflated A*. This last part is not a direct path towards the goal, which reflects the non-optimal nature of inflated A*. These results demonstrate the typical characteristics of the three searches. In general, as we move from A*-search to truncated A* to inflated A*: (1) the size of the search tree decreases, (2) the search time decreases, and (3) the quality of the solution decreases. Hence there is a tradeoff between the search time and the quality of the output motion.

We now present some experimental results that demonstrate the effectiveness of our approach. Figure 6 shows three human characters navigating in an environment with a cylinder-shaped tree obstacle that gradually falls down. The first character jogs past this obstacle before it falls, while the two that follow jump over it after it has fallen. Our planner takes less than one second to synthesize about 10 seconds of animation for each character. In general, the amount of search time is significantly less than the amount of motion that the planner generates.

The example in Figure 7 demonstrates three concepts. First, our algorithm can deal with simple rough terrain environments: the character successfully jogs up and down the small rough terrain area. However, if the terrain consists of



A* search truncated A* inflated A*

Figure 5: *A comparison of search methods. The top row illustrates the complete search trees colored according to cost, and the bottom row shows the waypoint nodes along the final paths returned by the planner for each search method.*

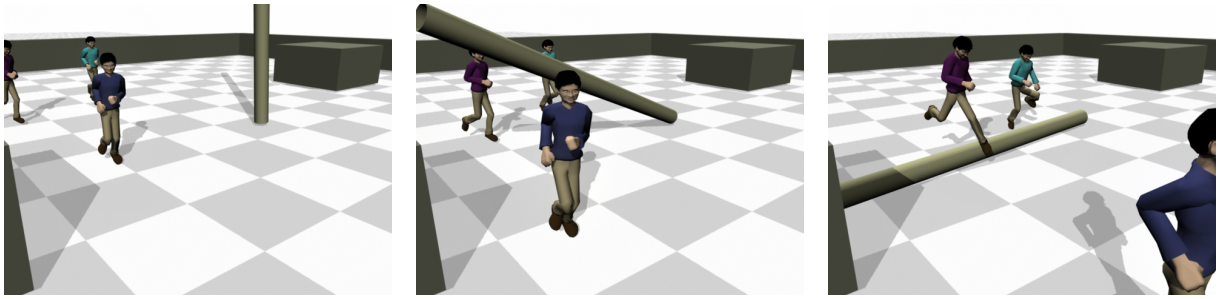


Figure 6: *A dynamic environment with a falling tree.* Left: *Before it falls, the characters are free to jog normally.* Center: *As it is falling, the characters can neither jog past nor jump over it.* Right: *After it has fallen, the characters can jump over it. See the accompanying video for the animation.*

steeper slopes, it would be best to include a "jog up terrain" behavior in our FSM. Second, an animator can control a character's preferences for certain types of behavior by adjusting cost functions that the planner uses to evaluate candidate motions. In the example, one solution (the light blue path) is generated using the normal costs for jumping and navigating uphill. Another solution (the dark blue path) is generated using very high costs for these two motions, with everything else being the same. The one with normal costs produces a more direct path: the character jogs through the terrain and executes a jumping motion. The one with higher costs produces a longer path, but it is optimal with respect to the costs given that the character prefers neither to jump nor navigate the elevated part of the terrain if possible. Third, our technique scales efficiently to large motion databases since the search performance is primarily dependent on the complexity of the behavior FSM rather than on the amount of motion data. We artificially create an FSM with over 100000 motion clips by including the same motion clip into a state 100000 times. This does not affect the resulting motion, but it simulates a large database of motion. For this artificial example, the database size increased by about 1000 times (from 2000 frames to 2000000), and the search time only doubled (from about 2 seconds to 4).

Figure 8(top) shows a skateboarder stopping and then jumping over a moving hook-shaped obstacle. It executes a "stop and wait" motion so that it can prepare for the timing of its jump. The planner takes about one second to generate 20 seconds of motion.

An example of three horses simultaneously avoiding each other and a number of moving obstacles is shown in Figure 8(bottom). The motions of the moving obstacles are pregenerated from a rigid-body dynamics solver. Their positions at discrete time steps are then automatically stored into the time-indexed gridmaps representing the environment. In

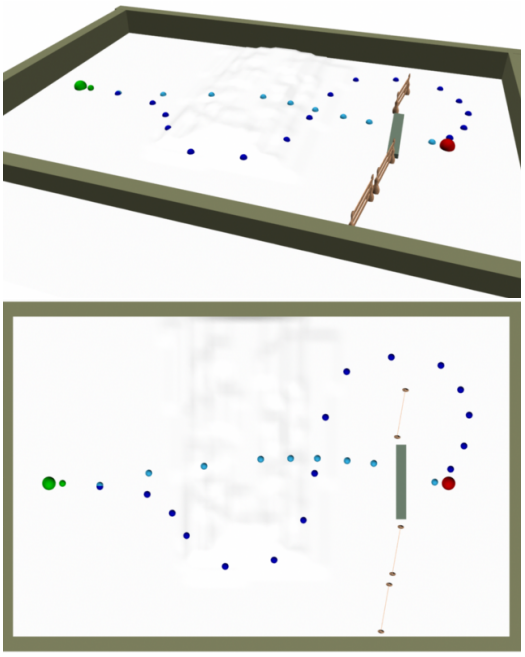


Figure 7: *An environment with an area of elevated terrain. The two synthesized paths were generated with different relative costs for jumping and climbing uphill and downhill.*

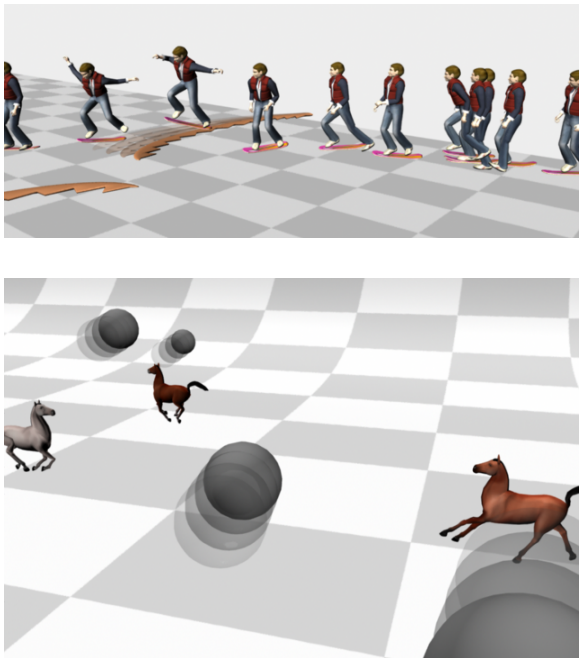


Figure 8: Top: *Snapshots of a skateboarder stopping and jumping over a moving obstacle.* Bottom: *Three horses galloping across a field of fast-moving obstacles.*

the animation, the horses appear to move intelligently, planning ahead and steering away from the moving obstacles in advance. Our algorithm takes roughly half of one second to generate approximately 16 seconds of motion for each horse.

Figure 1 shows synthesized motion for 100 human characters. In this example, it is important that the characters can exhibit a "stop-and-wait" motion. If a character is temporarily surrounded by moving obstacles (the other humans in this case), it can stop and wait for them to move past before resuming forward motion. The high-level data representation makes it intuitive to plan such a long and complex sequence of motion. Each of the 100 characters used the same FSM with approximately 55 seconds of animation, and the planner synthesized a total of 4000 seconds of motion. Given that there were only 55 seconds of data, the resulting motion has enough variety and complexity to be interesting. It takes on average approximately 4 seconds to plan 40 seconds of motion for each character.

The basic structure of the behavior FSM used to generate the motion of the skaterboarder, the horses, and all 100 of the human characters is similar to the one presented in Figure 3. These examples demonstrate the ability of our behavior planning approach to adapt to characters with vastly different kinematics and movement styles. In addition, the same FSM can be re-used for any number of characters with their own unique motion styles.

5. Discussion

We have presented a behavior planning approach to automatically generate realistic motions for animated characters. We model the motion data as abstract *high-level behaviors*. Our behavior planner then performs a *global search* of a data structure of these behaviors to synthesize motion. Although a designer has to connect these behaviors together into a behavior FSM, the tradeoff that comes from a well organized data structure is that the graph size, behavior semantics, search time, and level of control over the resulting motion is vastly improved.

Our behavior planning approach can provide guarantees of completeness and optimality. It is complete in the sense that if a solution exists, the algorithm will find it, and if no solution exists, it will fail in a finite amount of time. The method is optimal with respect to the behavior costs defined for the A* search. This optimality criteria, however, is not a necessity. We have shown that other non-optimal searches such as truncated or inflated A* can be used to produce reasonable results. A limitation to being complete and optimal is that A* search is an exponential search method. But this does not present a problem in practice when applied to small data structures such as our behavior FSMs. Our planner can search through several thousand states in a fraction of a second, and cover fairly long distances in a virtual environment.

The primary drawback of our approach compared to motion graphs [AF02, KGP02, LCR*02] is that motion graphs

can be constructed automatically from a general set of motion data. Our behavior planning approach relies on the existence of a behavior FSM, and motion data clips that have been appropriately categorized into sets of behaviors. In practice, we have found the FSM design process to be relatively simple, and once the FSM is constructed, it can be re-used for different characters and environments. Furthermore, we can add individual clips of motions to the states easily once its basic structure is defined. The majority of the work is, therefore, in the segmentation of data. We do require that the motions be carefully segmented and transition labels be identified. However, the added benefit of having abstracted motion clips organized into behaviors outweighs the extra effort in manually segmenting the data; there are many advantages (see Section 2) that come from building a well defined FSM.

An issue regarding the organization of the motion clips is that it is difficult to know when to group similar clips into the same state. We currently assign a clip to an existing state if the motion matches the high-level description of the behavior and it has the same transition labels of that state. Otherwise, we assign a new state for the clip, or we eliminate the clip completely if it does not fit into the states we want to have in the FSM. This grouping process is qualitative and is a potential candidate for automation in the future. Arikan et al. [AF02] clustered the edges in their graph of motion data. Lee et al. [LCR*02] have tried clustering to group similar poses. However, it is difficult to define an automatic algorithm to perform clustering on long sequences of motion accurately for a general set of data. It is difficult even for humans to group similar clips of motions together, and hence getting the ground truth for comparison purposes will be a challenge. The segmentation process requires identifying poses of the data that are similar, and can potentially be automated in the future. Identifying similar poses from a set of motion data is already a step that is automated when building a motion graph, and this is an active area of research [GSKJ03, BSP*04].

Our examples show planned motions in dynamic environments whose obstacle motions were known a priori. This is reasonable for applications in the movie industry that are offline, and with some interactive games where the motions of non-player entities are known in advance. Other interactive applications benefit from "on-the-fly" motion generation in environments where human-controlled characters move unpredictably. For this case, we suggest an anytime algorithm (Section 2), in which the planning process is repeatedly interrupted and returns the best motion computed at that point (obtained by examining the state at the top of the planning priority queue). Provided that replanning happens at a high enough frequency relative to the maximum speed of the unpredictable entities, the characters will be able to avoid obstacles "on-the-fly". There exists a tradeoff between the computation time and the optimality of the generated motion. For interactive game systems, the designer can minimize the planning time as much as possible, while keeping the synthesized motion reasonable.

Our behavior planner does not allow the virtual character to exactly match precise goal postures. Our focus in on efficiently generating complex sequences of large-scale motions across large, complex terrain involving different behaviors. Given a small number of appropriately designed "go straight", "turn left", and "turn right" actions, our planner can generate motions that cover all reachable space at the macro-scale. No motion editing is required to turn fractional amounts or traverse a fractional distance because we are computing motion for each character to travel over relatively long distances (compared to each motion clip). The algorithm globally arranges the motion clips in a way that avoids obstacles in cluttered environments while reaching distant goals. The character stops when it is within a small distance ε from the goal location. If matching a precise goal posture is required, motion editing techniques [BW95, WP95] may be used after the blending stage.

Although our results include animations of multiple characters, we did not originally intend to build a system for generating motions for crowds. We do not claim our method to be better than the existing specialized commercial crowd animation systems. Nevertheless, the application of our planning technique to multiple characters produces results that are quite compelling. In addition, to the best of our knowledge, existing crowd animation systems utilize local reactive steering methods [Rey87] rather than a global planning approach.

A possible direction for future work is to parametrize the states in the FSM. Instead of having a "jog left" behavior state, we can have a "jog left by x degrees" state. Such a state might use interpolation methods [WH97, RCB98] to generate an arbitrary turn left motion given a few input clips. This can decrease the amount of input data needed, while increasing the variety of motion the planner can generate. We can also have behaviors such as "jump forward x meters over an object of height h". This would allow our system to work in a larger variety of environments.

Acknowledgements: We thank Moshe Mahler for the human character models and for helping with video production. We thank Justin Macey for cleaning the motion capture data, and the CMU Graphics Lab for providing many resources used for this project. Finally, we are grateful to Alias/Wavefront for the donation of their Maya software. This research was partially supported by NSF grants ECS-0325383, ECS-0326095, and ANI-0224419.

References

[AF02] ARIKAN O., FORSYTH D. A.: Interactive motion generation from examples. *ACM Transactions on Graphics 21*, 3 (July 2002), 483–490.

[BLA02] BAYAZIT O. B., LIEN J.-M., AMATO N. M.:

Roadmap-based flocking for complex environments. In *Pacific Conference on Computer Graphics and Applications* (2002), pp. 104–115.

[BSP*04] BARBIČ J., SAFONOVA A., PAN J.-Y., FALOUTSOS C., HODGINS J. K., POLLARD N. S.: Segmenting Motion Capture Data into Distinct Behaviors. In *Proceedings of Graphics Interface 2004* (July 2004), pp. 185–194.

[BW95] BRUDERLIN A., WILLIAMS L.: Motion signal processing. In *SIGGRAPH 95 Conference Proceedings* (1995), ACM SIGGRAPH, pp. 97–104.

[CLS03] CHOI M. G., LEE J., SHIN S. Y.: Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics 22*, 2 (Apr. 2003), 182–203.

[FP03] FANG A. C., POLLARD N. S.: Efficient synthesis of physically valid human motion. *ACM Transactions on Graphics (SIGGRAPH 2003) 22*, 3 (July 2003), 417–426.

[FvdPT01] FALOUTSOS P., VAN DE PANNE M., TERZOPOULOS D.: The virtual stuntman: dynamic characters with a repertoire of autonomous motor skills. *Computers and Graphics 25*, 6 (2001), 933–953.

[Gle98] GLEICHER M.: Retargeting motion to new characters. In *Proc. ACM SIGGRAPH 98 (Annual Conference Series)* (1998), pp. 33–42.

[GSKJ03] GLEICHER M., SHIN H. J., KOVAR L., JEPSEN A.: Snap-together motion: assembling run-time animations. *ACM Transactions on Graphics 22*, 3 (July 2003), 702–702.

[GVK04] GO J., VU T., KUFFNER J.: Autonomous behaviors for interactive vehicle animations. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aug. 2004).

[HNR68] HART P., NILSSON N., RAFAEL B.: A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Sys. Sci. and Cyb. 4* (1968), 100–107.

[HWBO95] HODGINS J., WOOTEN W., BROGAN D., OŠBRIEN J. F.: Animating human athletics. In *Proc. ACM SIGGRAPH 95 (Annual Conference Series)* (1995), pp. 71–78.

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. *ACM Transactions on Graphics 21*, 3 (July 2002), 473–482.

[KKKL94] KOGA Y., KONDO K., KUFFNER J. J., LATOMBE J.-C.: Planning motions with intentions. In *Proceedings of SIGGRAPH 94* (July 1994), pp. 395–408.

[KvdP01] KALISIAK M., VAN DE PANNE M.: A grasp-based motion planning algorithm for character animation. *J. Visualization and Computer Animation 12*, 3 (2001), 117–129.

[LaV] LAVALLE S. M.: *Planning Algorithms*. Cambridge University Press (also available at http://msl.cs.uiuc.edu/planning/). To be published in 2006.

[LCR*02] LEE J., CHAI J., REITSMA P. S. A., HODGINS J. K., POLLARD N. S.: Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics 21*, 3 (July 2002), 491–500.

[LL04] LEE J., LEE K. H.: Precomputing avatar behavior from human motion data. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2004), ACM Press, pp. 79–87.

[LP02] LIU C. K., POPOVIC Z.: Animating human athletics. In *Proc. ACM SIGGRAPH 2002 (Annual Conference Series)* (2002).

[LS99] LEE J., SHIN S. Y.: A hierarchical approach to interactive motion editing for human-like figures. In *Proc. ACM SIGGRAPH 99 (Annual Conference Series)* (1999), pp. 39–48.

[MBC01] MIZUGUCHI M., BUCHANAN J., CALVERT T.: Data driven motion transitions for interactive games. *Eurographics 2001 Short Presentations* (September 2001).

[Men99] MENACHE A.: *Understanding Motion Capture for Computer Animation and Video Games*. Morgan Kaufmann Publishers Inc., 1999.

[PB02] PULLEN K., BREGLER C.: Motion capture assisted animation: Texturing and synthesis. *ACM Transactions on Graphics 21*, 3 (July 2002), 501–508.

[PG96] PERLIN K., GOLDBERG A.: Improv: A system for scripting interactive actors in virtual worlds. In *Proc. ACM SIGGRAPH 96 (Annual Conference Series)* (1996), pp. 205–216.

[PLS03] PETTRE J., LAUMOND J.-P., SIMEON T.: A 2-stages locomotion planner for digital actors. *Symposium on Computer Animation* (Aug. 2003), 258–264.

[PW99] POPOVIĆ Z., WITKIN A.: Physically based motion transformation. In *Proc. ACM SIGGRAPH 99 (Annual Conference Series)* (1999), pp. 11–20.

[RCB98] ROSE C., COHEN M., BODENHEIMER B.: Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Application 18*, 5 (1998), 32–40.

[Rey87] REYNOLDS C. W.: Flocks, herds, and schools: A distributed behavioral model. In *Computer Graphics (SIGGRAPH '87 Proceedings)* (July 1987), vol. 21, pp. 25–34.

[RP04] REITSMA P. S. A., POLLARD N. S.: Evaluating motion graphs for character navigation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aug. 2004).

[SHP04] SAFONOVA A., HODGINS J. K., POLLARD N. S.: Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics (SIGGRAPH 2004) 23*, 3 (Aug. 2004).

[SYN01] SHILLER Z., YAMANE K., NAKAMURA Y.: Planning motion patterns of human figures using a multi-layered grid and the dynamics filter. In *Proceedings of the IEEE International Conference on Robotics and Automation* (2001), pp. 1–8.

[WH97] WILEY D., HAHN J.: Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Application 17*, 6 (1997), 39–45.

[WP95] WITKIN A. P., POPOVIĆ Z.: Motion warping. In *Proceedings of SIGGRAPH 95* (Aug. 1995), Computer Graphics Proceedings, Annual Conference Series, pp. 105–108.

[YKH04] YAMANE K., KUFFNER J. J., HODGINS J. K.: Synthesizing animations of human manipulation tasks. *ACM Transactions on Graphics (SIGGRAPH 2004) 23*, 3 (Aug. 2004).