Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2012)
P. Kry and J. Lee (Editors)

Precomputed Motion Maps for Unstructured Motion Capture

Mentar Mahmudi and Marcelo Kallmann

University of California, Merced

Abstract

We present in this paper a solution for extracting high-quality motions from unstructured motion capture databases at interactive rates. The proposed solution is based on automatically-built motion graphs, and offers two key contributions. First, we show how precomputed expansion trees (or motion maps) coupled with new heuristics and backtracking techniques are able to significantly improve the time taken to search for motions satisfying user constraints. Second, we show that when feature-based transitions are employed for constructing the underlying motion graph, the connectivity of motion maps is greatly increased, allowing the overall method to perform search and synthesis at interactive frame rates. We demonstrate the effectiveness of our approach with the problem of extracting path-following motions around obstacles from a motion graph structure at interactive performances.

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

1. Introduction

High quality motion search and synthesis from unstructured motion capture examples among obstacles has proven to be a challenging problem. Successful methods are often based on motion graph structures [KGP02, AF02, LCR*02], which represent a popular approach for the purpose of character animation. Motion graphs have several good properties: they can be built automatically, they can be built to represent any kind of motion, and most importantly, they are able to produce realistic high fidelity results.

However, one of the main drawbacks of motion graphs is that they can easily become too large, preventing search algorithms from quickly finding motions that satisfy user constraints. Not only is the size of the graph a limiting factor, but also its connectivity since graphs with many transitions will have a higher branching factor eventually slowing down the underlying search algorithms. As such, motion graphs are mostly unable to support search and synthesis at interactive rates. For applications where interactivity is essential, the practical alternative often involves relying on a small hand-crafted set of motions with guaranteed transitions between themselves.

We present a solution for allowing motions to be extracted from a motion graph structure at interactive rates. We achieve this with the use of precomputed motion maps (see Figure 1) coupled with new heuristic and backtracking techniques that significantly improve motion search performance. To eliminate the need of manually-crafted motion representations with full connectivity, we also show that a feature-based segmentation of the motion capture database [MK11] is able to produce motion maps with enough density and connectivity for the overall method to successfully search and synthesize motions at interactive frame rates.

In this paper we focus on applying motion maps for solving the problem of navigation around obstacles. The obtained motions are realistic and are computed at interactive rates. Furthermore, due to the employed precomputed motion maps, motion search is often reduced to processing only the next best motion map, allowing search and synthesis to be performed in parallel.

Our overall method is divided in three main phases: motion map precomputation, path finding and path following. The precomputation phase is an off-line phase where the motion capture database is transformed into a feature-based motion graph (FMG) according to a locomotion feature segmentation [MK11], and motion maps are then computed for each node of the FMG. Given a query in run-time, the path finding phase will employ an efficient triangulation-based path planning method [Kal10] able to quickly return paths

© The Eurographics Association 2012.

DOI: 10.2312/SCA/SCA12/127-136

delivered by
EUROGRAPHICS DIGITAL LIBRARY
www.eg.org diglib.eg.org

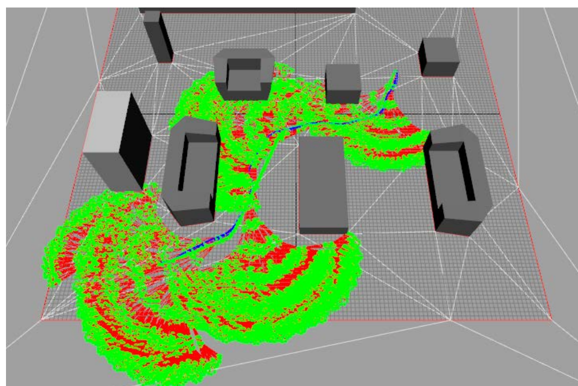

Figure 1: *The image shows four motion maps used in a path following query. Motion maps are search tree expansions efficiently precomputed and stored, and then employed in runtime queries.*

with guaranteed clearance from any given set of polygonal obstacles. The returned paths, therefore, provide channels with enough clearance for the character to move; minimizing collision checking queries after this point. In the path following phase, an optimized motion search algorithm is employed taking into account the precomputed motion maps, which are transformed and superimposed on the path during the search process. See Figure 1 for an example.

As a result, precomputed motion maps can be applied to efficiently solve path following queries from unstructured motion databases. Path following is one basic behavior important to several animation areas such as in simulation of populated environments and computer games. Our proposed solutions are automatic and applicable to generic locomotion data, and can therefore impact several of these applications.

2. Related Work

Motion graphs are built by connecting frames of high similarity in a database of motion capture examples [KGP02, AF02, LCR*02, PB02, LWS02]. Once the motion graph is available, a graph search is performed in order to extract motions with desired properties.

Many methods based on motion graphs have then been proposed. Kovar et al. [KGP02] used branch and bound to find motions that follow a user specified path by considering the motion synthesis problem as an optimization problem. Arikan and Forsyth [AF02] used a randomized method to extract motions from a hierarchy of motion graphs. Lee et al. [LCR*02] constructed a cluster forest of similar frames in order to improve the motion search efficiency. Dynamic programming was used by Arikan et al. [AFO03] to search for motions satisfying user annotations (such as first *run* and then *jump*).

The approach of Safonova and Hodgins [SH07] and Zhao and Safonova [ZS08] is based on an *interpolated motion graph* with *anytime A** used to search for solutions. The resulting motion is an interpolation of two time-scaled paths through the motion graph. Although it represents an improvement in respect to satisfying constraints, this approach requires additional search time for finding solutions. The difficulty to run at interactive frame rates is a common limitation of motion graph approaches.

Many other planning methods have been proposed for synthesizing full-body motions among obstacles [EAPL06, PZLM10, KL00]. In particular, Choi et al. [CLS02] combine motion capture data with probabilistic roadmaps [KSLO96] to generate motions for a given start and goal positions. Although such methods improve the planning capabilities for addressing constraints, the quality of the results are not improved in respect to motion graph approaches. Sung et al. [SKG05] make use of probabilistic roadmaps to guide a bidirectional search, achieving realistic results but relying on unrolling a manually-crafted motion graph structure.

Structures based on motion graphs still carry the benefit of minimally deforming the original database of collected motions, therefore achieving high quality results. In general, the main drawback of motion graphs is that a prohibitively large structure would be needed in order to produce motions satisfying many constraints, such as around obstacles and addressing precise goal locations. The problem of increasing the motion database is that, as the size of the graph grows, the underlying search methods will require additional computation time for finding solutions.

This inherent difficulty of motion graph structures is well-known and methods based on simplifying the database have also been proposed. In particular, Gleicher et al. [GSKJ03] note that one main difficulty of motion graphs is its unstructured nature, and they propose a method to simplify the graph to a small structured graph suitable to interactive control. In the same direction, fat graphs [SO06] and parametric graphs [HG07] have been proposed as attempts to improve the structure of the motion capture data so that interactive controllers can be devised.

The approach taken in this paper seeks to fully handle the entire given motion capture database, and to use precomputed search trees in order to achieve interactive performances. Precomputation of search expansions has been already employed for the problem of motion synthesis using motion capture data. For instance, the work of Lau and Kuffner [LK06, LK10] efficiently employs precomputed search trees for synthesizing goal-driven interactive motions. However, in order to achieve efficiency, their approach is designed around a manually-built finite state machine of motions [LK05] that is fully connected. In contrast, the specific techniques we propose enable precomputation to be applied to unstructured motion graphs.

Srinivasan et al. [SMM05] apply precomputed trees for

© The Eurographics Association 2012.

all nodes of a motion graph, and report difficulties handling tight spaces around obstacles due the standard search expansion technique employed. The related approach of precomputed avatar behavior policies has also been proposed by Lee at al. [LL04].

Our approach is most similar to these works, however, we focus on handling unstructured motions around obstacles at interactive rates. Our method introduces several benefits: 1) it works with generic automatically built Feature-based Motion Graphs (FMGs) [MK11] and hence does not require manually-built finite state machines of motions; 2) it precomputes expansion trees for all nodes in the FMG and does not require full connectivity, i.e. it does not require all leaves of a precomputed search tree to link to the root of the tree for seamless transitions; 3) it employs a fast triangulation-based path computation [Kal10] that computes paths within a collision free corridor with given clearance, allowing the search to be pruned to the corridor and minimizing collision detection queries during the search; and 4) it employs a search strategy that considers backtracking for well handling difficult situations involving tight spaces, turns and precise arrivals; achieving excellent overall performance due the improved search and the well-formed FMGs.

The proposed method, therefore, well handles unstructured motions around obstacles and is able to produce long paths efficiently even in complicated environments with many obstacles. The method is suitable to interactive applications and the results are always of high quality.

3. Finding Paths with Clearance

Given an initial point p_{init}, a goal point p_{goal}, and a clearance distance r, the collision-free path $P(p_{init}, p_{goal}, r) = (p_0, p_1, \ldots, p_n)$ is described as a polygonal line with vertices p_i, $i \in \{1, \ldots, n\}$ describing the solution path. As the path turns around obstacles with distance r from the obstacles, every corner of the path is a circle arc which is approximated by points, making sure that the polygonal approximation remains of r clearance from the obstacles.

Any path planning method with clearance can potentially be used to compute $P(p_{init}, p_{goal}, r)$. Due its efficiency, we employ the Local Clearance Triangulation (LCT) method [Kal10], which leverages 2D meshing algorithms for maintaining a suitable triangulation of the free space for path planning with clearance. We are using an extended version of the method that includes dynamic obstacle updates only requiring local updates each time an obstacle changes position (details will be available soon [Kal12]). We can, therefore, handle dynamic environments very easily, with updates and path queries in relatively complex environments being computed in the order of milliseconds. Figure 2 shows examples of collision-free paths with clearance obtained by the method.

Given that path determination takes obstacle clearance into account while solving at the path finding level, we almost entirely eliminate the need of using costly collision checking queries during the FMG search and synthesis.

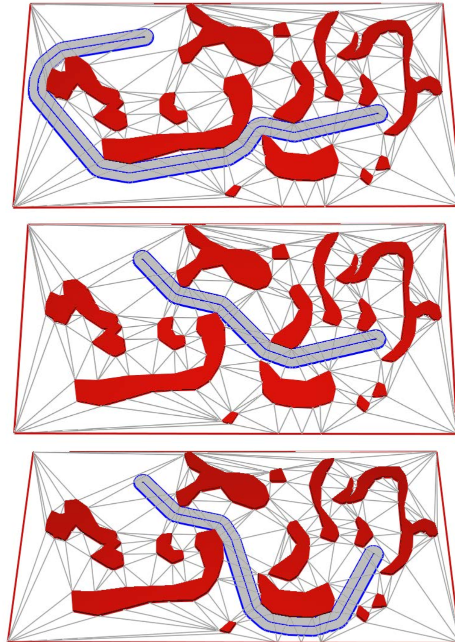

Figure 2: *Different paths obtained to connect the same initial and goal points, as they adapt to a few changes in the obstacles. The clearance of the paths is always maintained, and the LCT representation is updated only with local operations for each time an obstacle moves.*

In our path following application the obtained 2D path is used for guiding and pruning the FMG search and a fine polygonal approximation of the curved sections of the path is not needed. We, therefore, approximate the curved sections very coarsely with only a few points.

4. Precomputation of Motion Maps

We start by constructing a motion graph similarly to Mahmudi et al. [MK11]. Although, standard motion graphs might very well be used with our path following methods, our results indicate that FMGs have higher success rates because they are better at generating denser and more evenly distributed motion maps (see Figures 3 and 4). As later shown in Section 5, good density and distribution in motion maps are important requirements for the path following procedure to run successfully.

The next step is to precompute motion maps for all the nodes of the FMG. The rationale behind the precomputation is to be able to follow the input 2D path by repeatedly making efficient queries to motion maps for partial solutions that can follow the path closely. This process is repeated

© The Eurographics Association 2012.

until the goal is found or all the possible candidates are exhausted. Unlike previous methods that precompute only one expansion tree, we precompute motion maps for all the nodes of the motion graph. The benefit of precomputing all the nodes is that we do not have to rely on manually crafted motion graphs with full connectivity but can instead use any automatically-built motion graph.

Let c represent a node of the FMG, which in our representation contains one segmented motion clip. A motion map T_c of a node c represents a tree T_c of motions that can be generated starting from the node c. The motion maps span a three dimensional space X defined as:

$$X = \{(x, z, \theta) \in \mathbb{R} \times \mathbb{R} \times (-\pi, \pi]\}. \quad (1)$$

The x and z parameters determine the position of the character on the floor and the θ specifies the orientation about the Y vertical axis of the root joint of the skeleton; the positive Y axis points up on the XZ plane. For efficient storage, we discretize the motion maps into cells of 10 cm by 10 cm for the x and z parameters and into 12 groups of 30 degrees for the θ orientation. Furthermore, we represent motion maps as hashed maps instead of 3D grids. There are two reasons for this: first, by using hashed maps we avoid setting spatial limits on how far our motion maps can be unrolled; second, motion maps populate X sparsely and unevenly and, as such, a significant number of cells in our discretization remain empty. Hash maps will only store occupied cells and, therefore, no space will be allocated for unoccupied cells.

Algorithm 1 illustrates the overall precomputation process. We start by *unrolling* each node c of the FMG. The procedure starts by placing the first frame of the node c at the origin, with the character facing the positive Z axis, and building a motion map T_c with its root set to the node c (lines 1-3). Then, we run a Dijkstra search starting on the node c and expand nodes that minimize the arc-distance traveled by the expanded motions (lines 13-15). At every expansion, the motion map T_c is augmented with the position (x_i, z_i) and orientation θ_i of the character and the corresponding cell $T_c(x_i, z_i, \theta_i)$ is annotated with the path P_i that leads the character from the initial node c to the current state (x_i, z_i, θ_i) (line 12).

Note that since the same cell might be reached by different paths we store all possible paths at each stage. This increases the number of available candidates during the path following search phase. The unrolling stops when the Dijkstra search reaches a user-defined depth (line 6). Examples of typical precomputed motion maps obtained for FMGs are shown in Figure 3. As comparison, Figure 4 shows the obtained motion maps for a standard motion graph built from the same motion capture database. It is possible to note that the well-defined segmentation rule of FMGs lead to much denser motion maps, an essential property to guarantee that

Algorithm 1 Precompute(c, max_depth)

```
1:  F.init();
2:  Q.init(c);
3:  T.init(c);
4:  while ( Q not empty ) do
5:      node ← Q.pop();
6:      if ( node.depth ≥ max_depth ) then
7:          continue;
8:      if ( F(node) is not occupied ) then
9:          F.occupy(node);
10:     else
11:         continue;
12:     T.insert(node, path(node));
13:     node.expand();
14:     for all ( child ∈ children(node) ) do
15:         Q.push(child, child.length);
16: return T;
```

solutions are mostly always found. Additional comparisons are discussed in Section 7.

Each motion map T_c also stores the transformation Φ that aligns the map to the origin with the positive Z axis direction and the average length of a motion map, which is defined as:

$$\mathcal{A}(T_c) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(T_c^i), \quad (2)$$

$$\mathcal{L}(m) = \sum_{i=0}^{n-1} ||m_i - m_{i+1}||, \quad (3)$$

where T_c^i is the i-th path of T_c and $\mathcal{L}(P)$ returns the length of the 2D path P. The average length of T_c will determine the sampling frequency of the path P during the path following phase (Section 5).

Moreover, to speed up the precomputation, we also maintain a 4D frontier $F = \{(x, z, \theta, c_{id}) \in X \times \mathbb{N}\}$, which prevents expanding duplicate branches (lines 8-9). If the same node c_{id} is about to revisit a cell $x \in X$ then the we can safely cull this branch as expanding c_{id} at x would not lead to any new paths. Once the motion map is built, the frontier F is no more needed and it is discarded. Another significant speed up is to cache cells that were already queried. This facilitates the search during the path following phase since many queries are made to cells that were already used in prior queries.

The computational complexity of the precomputation procedure is $O(nb^d)$, where n is the number of nodes, b is the average branching factor of the motion graph and d is the chosen horizon depth of the motion map. Depth d is a significant factor influencing the precomputation time, however,

© The Eurographics Association 2012.

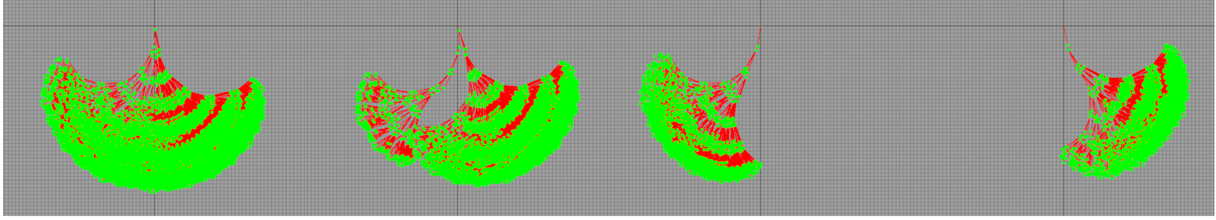

Figure 3: *Four typical motion maps precomputed for a Feature-Based Motion Graph (FMG). The used expansion depth is 12. Note the high density achieved in the regions covered by the motion maps.*

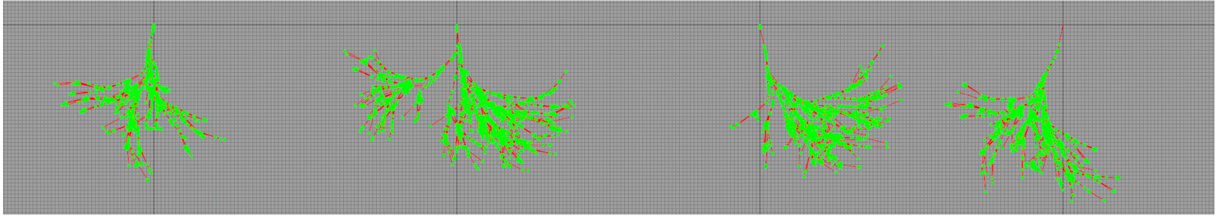

Figure 4: *The four equivalent motion maps to the ones shown in Figure 3, when precomputed for a standard Motion Graph (SMG). These maps were precomputed with depth 22 in order to achieve a size (number of nodes) equivalent to the FMG motion maps.*

we have noticed that after a certain depth is reached, increasing the horizon depth does not translate into further improvements to our path following procedure.

5. Path Following

As described in Section 3, the input path P is a collision-free path with clearance r, with starting point p_0 and goal point $p_{goal} = p_n$. The start and goal orientations are also defined by the input path. The start orientation is determined by the vector $p_1 - p_0$ and the goal orientation is determined by the vector $p_n - p_{n-1}$, where $p_i \in P(p_0, p_n, r)$ are points from the polygonal path as returned by the LCT path planner.

The path following procedure assumes that all the nodes of the FMG are precomputed up to a depth d and all their paths are stored in their corresponding motion maps. After the precomputation is finished, the goal of the path following procedure is to search for a sequence of nodes that generates a smooth motion closely following the collision-free input path $P(p_0, p_n, r)$.

The search procedure is presented in Algorithm 2, and all line references that follow are in respect to this algorithm. The search starts by selecting an initial node i from the FMG as the starting node of the search, and aligning its precomputed motion map T_i with the input path P. The alignment is determined by the transformation Φ_i associated with the motion map T_i and a transformation Ψ_j as determined by the last position and orientation of the partial solution up to the current point. Initially, Ψ_0 is set to the start position and orientation of the path P. Subsequent alignments are performed in a similar manner by concatenating the accumulated transformation Θ with the product of Φ_i and Ψ_j, as defined by the partial solution of the j-th iteration of the algorithm:

$$\Theta_j = \Theta_{j-1} \Phi_i \Psi_j. \quad (4)$$

The next step is to sample path P to obtain query points $q \in X$ which are a subset of the space X (line 3). The query points are equally spaced between the beginning of the residual input path up to the point with length along the path equal to the average length of T_i, as defined in Equation 2. In our experiments most points were about 10 cm apart, see Figure 6 for an example. Note that the sampled points are in world coordinates and before querying the motion maps they have to be transformed to the motion map's local frame. This is done by transforming the query points $q_i \in Q$ by the Θ_j^{-1} transformation.

At every iteration of the path following procedure, we first query for the goal p_n and then all the query points $\Theta^{-1} p_i$ against T_i to check for possibles partial paths (line 4). If none of the queried cells in the motion map are occupied, the algorithm backtracks to consider different candidates from its previous iteration. If during backtracking the root node is reached and all its alternatives are exhausted then the algorithm stops and reports failure (lines 5-9). When the goal query is successful then the final partial path is appended to

© The Eurographics Association 2012.

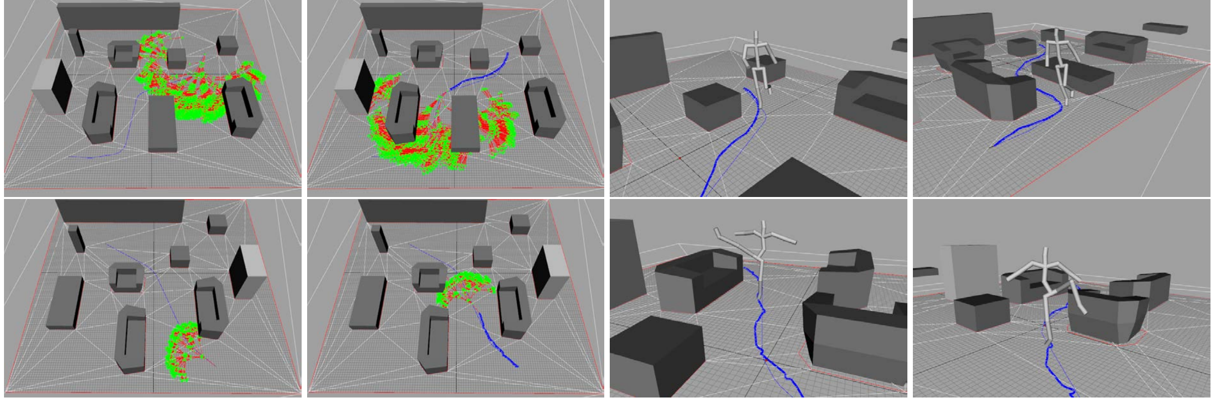

Figure 5: *Motion maps and their corresponding generated motions for a happy walking motion (top) and ballet motion (bottom).*

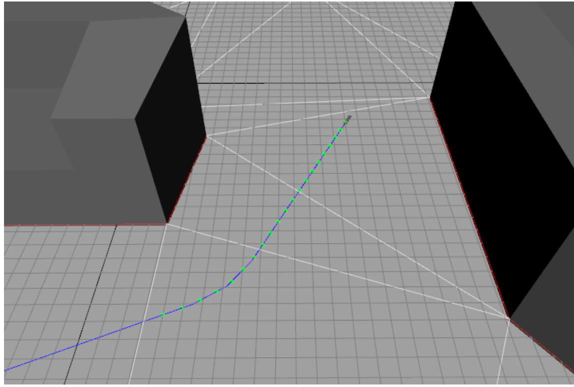

Figure 6: *Sample points from the input path used during the path following search.*

the solution and success is reported (lines 19-20), otherwise, all the queried candidate paths are retrieved and sorted for their suitability (line 11).

The sorting function that picks the best partial path among the available candidates could be tuned to suit a particular application. Since our goal is to follow the path as closely as possible, we have defined our sorting function to compute the area formed between the candidate partial path m and the input path P. However, since longer paths will lead to a larger area, we divide the cost by the square of the length of the candidate path to avoid favoring shorter paths.

The sorting function $f(m,M,P)$, is shown in Equation 5. It takes three parameters: a candidate path m, the current partial solution M and the input path P. The $\mathcal{L}(m)$ function computes the length of a path m (if an index of a frame is indicated then it computes the length up to that frame). Function $\mathcal{P}(P,l)$ returns a point $p \in P$ at distance l along the input path P. These functions read as follows:

$$f(m,M,P) = \frac{\sum_{i=0}^{n} ||m_i,\ \mathcal{P}(P,\ \mathcal{L}(M)+\mathcal{L}(m_i)))||}{\mathcal{L}(m)^2}, \quad (5)$$

$$\mathcal{P}(P,l) = \left\{(x,z) \in \mathbb{R}^2 \mid 0 \leq l \leq \mathcal{L}(P)\right\}. \quad (6)$$

Once all the candidate paths are sorted, we iterate through the sorted candidate paths and pick the best path m_{best} (line 12). Once the best candidate path m_{best} is chosen, we run collision checking on m_{best} to determine if it is collision-free; otherwise, we consider the next best candidate. Once such candidate path is found, it gets appended to the current partial solution M (line 16) and the last node of the best path m_{best} determines which motion map gets used in the next iteration of the path following procedure (line 18). At this stage, the partial solution path M is a motion that follows the input path P up to the point p_{best}, with an orientation facing the residual input path P. In case all the sorted candidates are unsuitable, the procedure backtracks and resumes considering other alternatives from the previous iteration (lines 13-14). The path following search iterates through this procedure until either the goal is reached (lines 19-20) or the lengths of the partial solutions M exceed the length of the input path P. Figure 7 overviews the main stages of the overall algorithm and Figure 5 depicts two such examples.

As the path following advances towards the goal, a stack of sorted candidates are stored at different stages. When backtracking needs to occur, the failed stage is popped from the stack and the next best candidates left from the previous stages are reconsidered without having to re-query the motion maps associated with the previous stage. Storing partial paths in stages offers a significant speed up and allows for efficient reuse of prior partial path. In this manner, as a whole, our motion search algorithm can be seen as an efficient cached A* search with very effective pruning by the 2D channel computed from the triangulation planner.

© The Eurographics Association 2012.

Depth	12	13	14	15	16	17	18	19	20	21	22
Time	5.72s	8.14s	11.58s	16.36s	22.93s	31.92s	44.23s	1m 0s	1m 23s	1m 55s	2m 38s
Cells	9600	12141	15361	19351	24367	30521	38391	48193	60356	75506	93992
Size	29MB	42MB	59MB	83MB	116MB	162M	225M	311MB	428MB	588MB	805MB

Table 1: *Motion maps with various depths in a SMG: preprocessing time, average number of occupied cells, and average size.*

Depth	2	3	4	5	6	7	8	9	10	11	12
Time	0.03s	0.09s	0.19s	0.57s	1.21s	3.26s	6.76s	17.39s	35.38s	1m 28s	3m 4s
Cells	161	360	667	1396	2458	4956	8312	16173	25552	47413	70117
Size	0.1MB	0.39MB	0.9MB	3MB	7MB	17M	35MB	88M	179M	444MB	889MB

Table 2: *Motion maps with various depths in a FMG: preprocessing time, average number of occupied cells, and average size.*

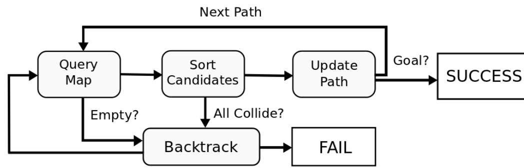

Figure 7: *Main stages of the path following search procedure.*

Algorithm 2 FollowPath(T, P, i)

1: $i_{last} \leftarrow \emptyset$
2: **while** $\mathcal{L}(M) \leq \mathcal{L}(P)$ **do**
3: $Q \leftarrow$ sample_path(P);
4: $C \leftarrow$ query_map(T_i, Q);
5: **if** C is empty **then**
6: **if** $i_{last} == \emptyset$ **then**
7: **return** FAIL;
8: **else**
9: $i \leftarrow i_{last}$;
10: **else**
11: $S \leftarrow$ sort_paths(C);
12: $m \leftarrow$ best_path(S);
13: **if** $m ==$ NONE **then**
14: $i \leftarrow i_{last}$;
15: **else**
16: M.append(m);
17: $i_{last} \leftarrow i$;
18: $i \leftarrow m$.last();
19: **if** M.goal_reached() **then**
20: **return** M;
21: **end while**

6. Concurrent Motion Synthesis

Since the time spent on searching is significantly lower than the duration of the synthesized motions, we can search and play motions concurrently. When a query is made to the path following procedure, the first node to be used is either specified by the user or automatically chosen by the procedure. Since this first node is known in advance, and is not subject to change for a particular query, our method can immediately start the path following search procedure while the first node is being played. When the first node has finished playing, the path following is already way ahead of the nodes that need to be played next. Usually the entire path following search concludes before the first few nodes are played.

In order to achieve concurrent playing and search we have two separate threads: the search thread and the rendering thread. The search thread is assigned the task of carrying the path following search procedure and the rendering thread renders the obtained partial solutions. The rendering thread initially awaits a signal from the search thread notifying the completion of the first partial path. This is usually done almost instantaneously. Then, after finishing rendering the first partial path, the rendering thread continuously polls the search thread for new partial paths until the goal is reached. The rendering threads tries to achieve a fixed frame rate of 30 frames per second. After the visualization buffer is updated the rendering thread is blocked and the path following searched is resumed until the next rendering cycle. The parallelization has shown to scale well and we have successfully run about 10 characters searching and playing motions at 30 frames per second in a quad-core CPU.

Due to the backtracking possibility of our path following search procedure, the rendering thread might start rendering a partial path that has been backtracked in the search thread. In that event, the rendering thread stops playing the motion and reports a failure. This does not happen very often as the path following search is very efficient, however, it may happen if the motion database lacks motions suitable for navigating in all areas of the environment. The user might always

© The Eurographics Association 2012.

decide to improve the database, or to disable simultaneous search and animation in order to maximize the use of the backtracking mechanism.

7. Results and Discussion

For our experimental setup we have built three FMGs from different types of motions: regular walking database, happy walking database and one ballet motion database. Each of these motion capture databases contained 1 straight, 1 left sharp turning, 1 left gentle turning, 1 right sharp turning and 1 right gentle turning motion. Our method showed to work well with this relatively small number of turn variations. Additional variations would improve the already high success rate of the algorithm, and the only drawback would be additional storage space for the precomputed motion maps. The motions were sampled at 60Hz using 18 Vicon cameras in an environment of 5.5 m by 4 m. The total number of frames in the motion capture databases ranged from 1079 to 3181, corresponding to 20-60s of motions. The motions were captured and processed into the FMGs without any manual editing involved. All the measurements took place on a 2.7 GHz Intel i7 computer with 4GB of memory.

We built the FMGs as described in [MK11], and for comparison we also built a standard Motion Graph (SMG) as described in [KGP02]. For achieving adequate comparisons we have used the same error threshold for deciding transitions in both graphs. In our trials, for the regular walking database, we have set the maximum error threshold to 6 cm, obtaining a FMG with 72 nodes and average branching factor of 1.56, and a SMG with 277 nodes and an average branching factor of 1.30. Similar graphs were obtained for the happy walking and ballet motions.

We then precomputed motion maps for all the nodes of both FMG and SMG, and at various depths. As mentioned earlier, motion maps were discretized into cells of 10 cm by 10 cm for the (x,z) positional parameters and into 12 cells of 30 degrees for the orientation parameter. Table 1 (SMG) and Table 2 (FMG) show the time taken to generate all the motion maps. Also reported is the total number of occupied cells and the combined size of all the motion maps.

As it can be seen from the Tables 1 and 2, SMGs require higher depths, in comparison to FMGs, to occupy motion maps with the same amount of entries. The reason for this is the fact that the average length of a node in SMG is shorter than the average length of a node in FMG. In Figures 3 and 4 we show motion maps precomputed from a FMG (of depth 12) and from a SMG (of depth 22) and notice that FMGs are significantly better at evenly spanning the covered space of X and thus they present themselves as a better choice for the path following algorithm. They offer a wider range of different candidate motions and ensure global connectivity (and concatenation success) during the path following search. This FMG property is essential for successful execution of the path following method.

size	depth	success	t (ms)	len (m)	inp (m)
33	8	63%	412.6	10.15	10.12
86	9	71%	356.7	10.96	10.94
174	10	87%	188.7	12.13	12.10
433	11	90%	148.3	12.75	12.66
867	12	93%	145.8	12.36	12.27

Table 3: *The effect of the size (in MB) of motion maps on the success rate and performance of the path following search for FMGs. The right-most four columns show the success rate of the search, the average time taken to search for the solutions, the average length of the solutions, and the average length of the input paths.*

We also measured the importance of the motion map depth and size in respect to the effectiveness of the path following method by running 1000 trials on the environment shown on Figure 8. Each trial consisted of: randomly selection of start and goal locations on the environment, query to the LCT planner for a path P with clearance of 50 cm, and full execution of the path following in respect to the input path P. The same trials were conducted for both the FMG and SMG, and the results are shown in Tables 3 and 4.

Table 3 reports the results for FMG. We can notice that the size of motion maps initially had a strong influence on the success rate and search time of the path following method, however, once a certain horizon was reached larger motions maps did not improve the path following method as effectively. We also notice that the lengths of the solution motions returned by the path following method were very close to the length of the input path, showing that our method is capable of generating solutions that are very close to optimal solutions. This is as expected since we sample the input path and only admit candidates that follow the path closely.

The same trials for SMG are shown on Table 4. Here, however, we see that SMG did not scale as well as FMGs. For maps of relatively same sizes, SMGs failed to achieve acceptable success rates and ran slower than FMGs. As noted earlier, the main drawback of SMGs were that they failed to provide a variety of candidates for the path following method to consider and thus failed to find solutions following the input that.

We then compared our path following method against two different search methods: A* search and A* search with channel pruning (A*-Ch) as described in [MK11]. The A* search represents a popular technique to generate optimal solutions when unrolling the graphs. A*-Ch runs the A* search only inside a channel around the input path by pruning all branches that go outside the path channel. It represents a simple way to improve the A* search, however loosing optimality. A comparison of the methods illustrating their speed of computation is shown on Figure 8: the A* search took 186s, A*-Ch took 9.4s and the search with motion maps took

© The Eurographics Association 2012.

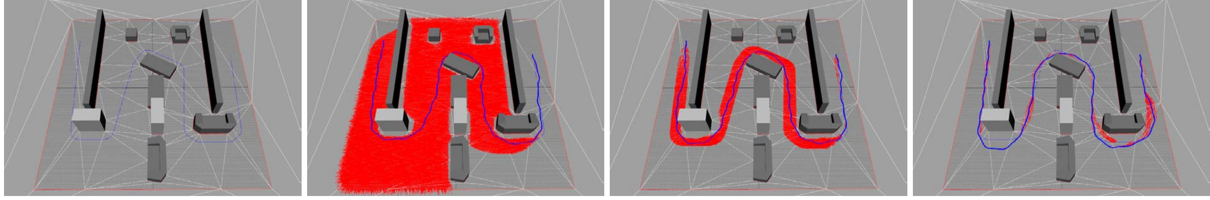

Figure 8: *Comparison of three search techniques. The left-most image shows the used environment and its blue path is the input path as returned by the LCT planner. The subsequent images depict the search tree expansion of A*, A*-Ch and our proposed method based on motion maps. The blue path in these last three images is the projection of the returned motion solutions. The A* search solution took 186s to be computed, the A*-Ch solution took 9.4s, and the search with motion maps took 0.760s.*

size	depth	success	t (ms)	len (m)	inp (m)
28	12	17%	175.6	6.39	6.19
81	15	32%	120.0	8.24	8.03
158	17	45%	335.6	7.82	7.66
417	20	55%	352.1	9.40	9.10
767	22	70%	556.9	10.13	9.86

Table 4: *The effect of the size (in MB) of motion maps on the success rate and performance of the path following search for SMGs. The columns are the same as in Table 3.*

only 760ms. In this example our method gave an improvement of 245 and 12.3 folds respectively.

We have also performed numerical comparisons between the three search methods, with 1000 random trials on the environment shown in Figure 8. We used motion maps of depth 11 for the FMG and of depth 20 for the SMG. We have only compared trials where all the three methods were able to successfully return a solution. The results are shown in Tables 5 and 6.

Table 5 shows the results for the FMG. The average length of the input path was 14.34m. As it can be seen from the table, motion maps were significantly faster than the other methods, and the length of the solutions were also very close to the original path length. Motion maps did show a reduced success rate of 93%. However, this rate is still excellent for a method based on precomputed search trees, and it is higher than reported results in previous related works. Furthermore, this rate can be improved to 100% by making sure that the motions in the database include all needed variations for a given environment. For instance in our paths around obstacles, more motions of different turning angles would be needed.

Table 6 shows the results obtained from running the same experiments with a SMG. The average length of the input paths was 12.87m. We notice that, although motion maps are always faster than the other methods, when SMGs are used, the success rate is decreased to 58%. FMGs are essential for motion maps to be effective, however, it is important to

method	success	t (ms)	len (m)	speed up
A*	100%	23527.8	13.95	157.6x
A*-Ch	98%	1742.1	13.95	11.7x
M. Maps	93%	149.3	14.51	1.0x

Table 5: *Search performance comparisons for A*, A*-Ch and motion maps in 1000 random trials using FMG. The columns show the overall success rate, the average computation time taken, the average length of the obtained solutions, and the relative speed improvement obtained with motion maps.*

method	success	t (ms)	len (m)	speed up
A*	100%	44184.0	12.48	50.6x
A*-Ch	96%	3192.4	12.58	3.6x
M. Maps	58%	872.4	13.30	1.0x

Table 6: *Search performance comparisons as described in Table 5, but here based on SMG.*

note that using SMGs with more elaborate precomputation techniques, such as the method by Lau et al. [LK10], could make SMGs more suitable for our path following method. However, we have not run any experiments to verify this.

The presented experiments clearly demonstrate the performance of the precomputed motion maps. The success rate of the overall method is highly related to the motion capture database used to build the underlying FMG. In our presented examples no special care was taken when selecting motion clips for building the used FMGs, and still the success rate achieved was of 93%. The presented experiments can also serve as a way to evaluate the suitability of the motion capture database, and additional methods can also be employed for measuring coverage [RP07].

Another important observation is that the granularity of the precomputed motion maps does not influence how well precomputed maps can store the motions. The reason for this is that we do not disregard paths that might lead to a cell that has been already occupied, and thus we never fail to capture all the available paths from the root of a precomputed mo-

© The Eurographics Association 2012.

tion map. Changing the resolution of the motion map might redistribute the paths into neighbouring cells or pack neighbouring cells into a larger cell but the number of entries in the motion map will not be altered.

However, the granularity of the motion maps does have a slight influence on the odds of finding a candidate path while sampling the motion map. If the resolution is very coarse, the sampling should be adjusted accordingly so that neighbouring cells are not queried, otherwise overly long motions might be considered. On the other hand, if the resolution of the motion maps is overly fine then a finer sampling should be employed in order to void missing possible candidates. Across all our experiments we kept the resolution of the precomputed motion maps constant and equal to the resolution of the sampling routine. One limitation of our method is that the storage space may become high for large databases.

8. Conclusion

We have presented new preprocessing and search techniques enabling unstructured motion graph structures to be efficiently employed for locomotion synthesis around obstacles in complicated environments. Several experiments were presented demonstrating the benefits of the proposed methods. The results are always of highly quality and are computed at interactive rates.

Acknowledgments This work was partially supported by NSF Award IIS-0915665.

References

[AF02] ARIKAN O., FORSYTH D. A.: Synthesizing constrained motions from examples. *Proceedings of SIGGRAPH 21*, 3 (2002), 483–490. 1, 2

[AFO03] ARIKAN O., FORSYTH D. A., O'BRIEN J. F.: Motion synthesis from annotations. *Proceedings of SIGGRAPH 22*, 3 (2003), 402–408. 2

[CLS02] CHOI M. G., LEE J., SHIN S. Y.: Planning biped locomotion using motion capture data and probabilistic roadmaps. *Proceedings of SIGGRAPH 22*, 2 (2002), 182–203. 2

[EAPL06] ESTEVES C., ARECHAVALETA G., PETTRÉ J., LAUMOND J.-P.: Animation planning for virtual characters cooperation. *ACM Transaction on Graphics 25*, 2 (2006), 319–339. 2

[GSKJ03] GLEICHER M., SHIN H. J., KOVAR L., JEPSEN A.: Snap-together motion: assembling run-time animations. In *Proceedings of the symposium on Interactive 3D graphics and Games (I3D)* (NY, USA, 2003), pp. 181–188. 2

[HG07] HECK R., GLEICHER M.: Parametric motion graphs. In *Proc. of the Symposium on Interactive 3D Graphics and Games (I3D)* (New York, NY, USA, 2007), ACM Press, pp. 129–136. 2

[Kal10] KALLMANN M.: Shortest paths with arbitrary clearance from navigation meshes. In *Proc. of the Eurographics / SIGGRAPH Symposium on Comp. Animation (SCA)* (2010). 1, 3

[Kal12] KALLMANN M.: Path planning with dynamic and robust local clearance triangulations. *Manuscript in preparation.* (2012). 3

[KGP02] KOVAR L., GLEICHER M., PIGHIN F. H.: Motion graphs. *Proc. of SIGGRAPH* (2002). 1, 2, 8

[KL00] KUFFNER J. J., LATOMBE J.-C.: Interactive manipulation planning for animated characters. In *Proceedings of Pacific Graphics* (Hong Kong, October 2000). poster paper. 2

[KSLO96] KAVRAKI L., SVESTKA P., LATOMBE J.-C., OVERMARS M.: Probabilistic roadmaps for fast path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation 12* (1996), 566–580. 2

[LCR*02] LEE J., CHAI J., REITSMA P., HODGINS J. K., POLLARD N.: Interactive control of avatars animated with human motion data. *Proceedings of SIGGRAPH 21*, 3 (July 2002), 491–500. 1, 2

[LK05] LAU M., KUFFNER J. J.: Behavior planning for character animation. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (Aug. 2005), pp. 271–280. 2

[LK06] LAU M., KUFFNER J. J.: Precomputed search trees: planning for interactive goal-driven animation. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation (SCA)* (2006), pp. 299–308. 2

[LK10] LAU M., KUFFNER J.: Scalable precomputed search trees. In *Motion in Games*, vol. 6459. Springer Berlin / Heidelberg, 2010, pp. 70–81. 2, 9

[LL04] LEE J., LEE K. H.: Precomputing avatar behavior from human motion data. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation (SCA)* (2004), pp. 79–87. 3

[LWS02] LI Y., WANG T.-S., SHUM H.-Y.: Motion texture: a two-level statistical model for character motion synthesis. *Proceedings of SIGGRAPH 21*, 3 (2002), 465–472. 2

[MK11] MAHMUDI M., KALLMANN M.: Feature-based locomotion with inverse branch kinematics. In *Proc. of the 4th International Conf. on Motion In Games (MIG)* (2011). 1, 3, 8

[PB02] PULLEN K., BREGLER C.: Motion capture assisted animation: Texturing and synthesis. *Proceedings of SIGGRAPH* (2002), 501–508. 2

[PZLM10] PAN J., ZHANG L., LIN M., MANOCHA D.: A hybrid approach for synthesizing human motion in constrained environments. In *Conference on Computer Animation and Social Agents (CASA)* (2010). 2

[RP07] REITSMA P. S. A., POLLARD N. S.: Evaluating motion graphs for character animation. *ACM Trans. Graph. 26* (October 2007). 9

[SH07] SAFONOVA A., HODGINS J. K.: Construction and optimal search of interpolated motion graphs. *ACM Transactions on Graphics (Proceedings. of SIGGRAPH) 26*, 3 (2007). 2

[SKG05] SUNG M., KOVAR L., GLEICHER M.: Fast and accurate goal-directed motion synthesis for crowds. In *Proceedings of the Symposium on Computer Animation (SCA)* (jul 2005). 2

[SMM05] SRINIVASAN M., METOYER R. A., MORTENSEN E. N.: Controllable real-time locomotion using mobility maps. In *Proceedings of Graphics Interface 2005* (2005), pp. 51–59. 2

[SO06] SHIN H. J., OH H. S.: Fat graphs: constructing an interactive character with continuous controls. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer animation (SCA)* (2006), pp. 291–298. 2

[ZS08] ZHAO L., SAFONOVA A.: Achieving good connectivity in motion graphs. In *Proc. of the 2008 ACM/Eurographics Symp. on Computer Animation (SCA)* (July 2008), pp. 127–136. 2

© The Eurographics Association 2012.