Real-Time Geometry Aware Motion Planning

Nithin Tharakan
GV2 Research Group
Trinity College Dublin
Dublin 2
Rep of Ireland
Email: tharakan@cs.tcd.ie

Cormac O'Brien
GV2 Research Group
Trinity College Dublin
Dublin 2
Rep of Ireland
Email: obrienc1@tcd.ie

Steven Collins
GV2 Research Group
Trinity College Dublin
Dublin 2
Rep of Ireland
Email: Steven.Collins@cs.tcd.ie

***Abstract*—Synthesising motions for user controlled characters whose motion is required to be constrained by the geometry of a virtual scene is a challenging problem in character animation. It is however a common requirement for applications such as games. We present a solution with a focus on: eliminating manual editing of animations to fit a scene; responsive user control; real-time motion synthesis and planning; and high fidelity interactions. Modelling the contacts that drive a motion gives us the basis for a fast run-time feasibility test that allows motion planning to an arbitrary depth.**

I. Introduction

In this paper we propose a novel approach for planning and synthesising motion for a controllable character given the geometric constraints of an arbitrary virtual scene.

Synthesising motion for a character is, in itself, a well-known problem in the field of animation. We focus on data driven approaches where motion is stitched together from clips of animation in an existing corpus [1]. Given a graph structure defining the connectivity between segments of animation, the problem becomes one of sequencing that animation by traversing edges constrained by the geometry of a scene. The essence of a data-driven solution is to embed suitable edges of the graph in the scene and traverse this embedded graph where edge traversal is guided by user input.

Some approaches completely unroll the motion graph into the scene and search for a globally optimum path across the graph, satisfying constraints such as start and end position and collision avoidance [2]. Due to the combinatorial increase in the number of possible paths this is very memory-intensive and the motion sequence must be computed offline [3].

Alternatively, geometry can be manually annotated with metadata specifying motion that can interact with it [4]. This approach leads to the construction of an animation sequence completely from pre-computed blocks of geometry-with-animation [5]. Such animations can have detailed interactions with the scene, but are fixed relative to the geometry and cannot react to user input.

A navigation mesh [6] gives a structure for path planning but does not provide for detailed interactions with particular features of a scene. This solution is popular in modern computer games, which add further requirements such as real-time synthesis and responsive player control. Games typically use highly complex animation state graphs where sequences and blends of animations and transitions require large amounts of artist input and are tailored to scene design. Furthermore, motion planning is typically very shallow, performing raycasts against scene geometry to determine Inverse Kinematics (IK) targets only for the next end-effector contact.

Our approach provides a framework for real-time motion synthesis for characters constrained by a scene while motion planning to an arbitrary depth in the animation graph. It also eliminates the need for manually editing the animations to fit the scene. Finally it facilitates high fidelity interactions between the character and the scene by employing a model of end-effector contact that we call docking.

II. Implementation

Our system assumes a user controlled animated character driven by a state graph. We perform medium range motion planning at run-time by unrolling the state graph dynamically in the region near the character and pruning infeasible graph edges. To determine the feasibility of a given motion we search for nearby geometry that can support a model of physical contacts for that animation, which we refer to as docks. The choice of which sequence of animations to execute from the set of feasible animations is made on the basis of user input. For responsiveness our animation clips are kept short, typically about 0.5s. Finally the support information for the chosen action informs an IK technique that enforces the desired contacts.

A. Docks

Moving through an environment requires physical contact by which force is transferred. We use the term docking to describe this interaction. We model the contact manifold between an end-effector and the environment as a mesh of contact point-and-normal pairs that we call a docking template (Figure 1). This represents the points at which forces are exerted during contact and the directions of those forces, normal to the surface.

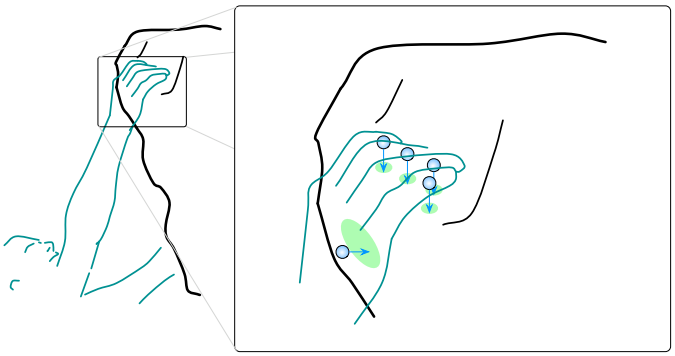

Fig. 1. Example of a hand docking template showing point-and-normal pairs for each finger and the palm.

This is sufficiently general to describe detailed end-effector shapes. Matching such shapes to plausible supporting features in the scene is described in section II-D. The docking templates are constructed manually in our prototype system but work is underway to automate this process. To represent the hands and feet we use simple docking templates consisting of two points and their associated normals. For each hand, the two points are located by mapping the wrist and middle-finger joint positions to the skinned character geometry. A similar mapping with the ankle and toe joint positions are used for each foot (Figure 2).

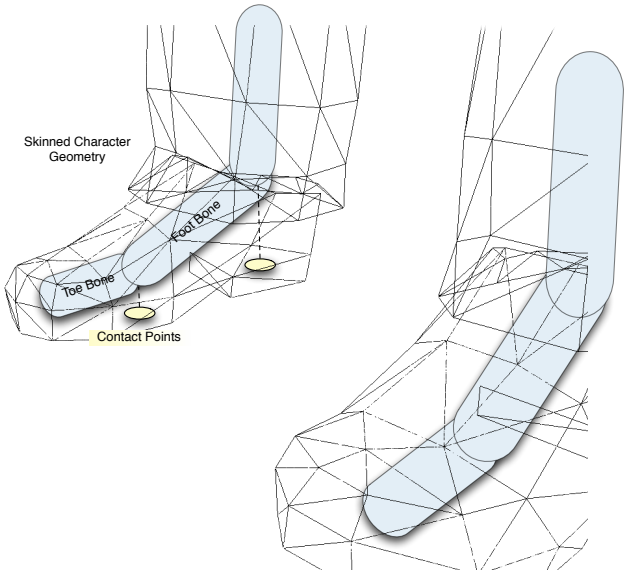

Fig. 2.

The space-time locations of the docking contacts associated with an animation are determined by finding intervals where the end-effectors seemed to be stationary. For each clip in our animation state graph we bundle the docking model (templates and positions) and animation data (joint angles and root trajectory) together and call this an action. An action can be considered feasible if all of its docks have a valid support over the entire duration of the action given the surrounding scene geometry.

B. Unrolling & Pruning the State Graph

For planning the performance of actions we unroll the state graph into the scene to form a spatial hierarchy in which each node represents an action. Unrolling a dense graph leads to a combinatorial explosion in the number of actions that must be evaluated.

Each node (action) has a rigid transform relative to its parent (preceding action) according to the final translation and rotation of the preceding action's root trajectory. The unrolled graph locates actions in scene space, providing the expected locations of the docks, and hence target regions for locating supports via geo-queries (Section II-D).

To obtain a motion planning graph where all edges are valid, we perform a feasibility test on each edge, and discard those edges whose docks are not supported by corresponding geometry (Figure 3). When a node is marked infeasible, all its descendants are considered infeasible and the whole branch can be pruned from the tree without further geo-queries. If all descendants of a node are infeasible then that node is infeasible and can also be pruned. This leaves us with a tree of feasible paths, to a given depth, which may be executed to move the character through the scene (Figure 4).

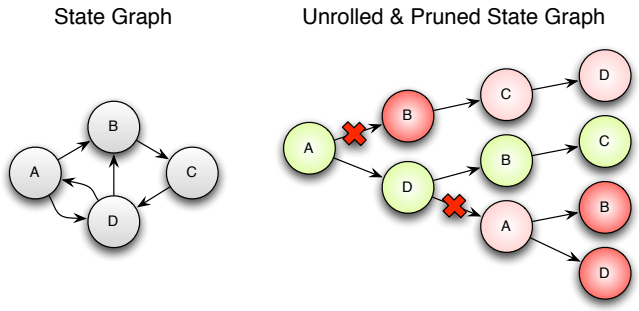

Fig. 3. State graph unrolling and pruning. Green and red circles indicate feasible and infeasible states respectively.

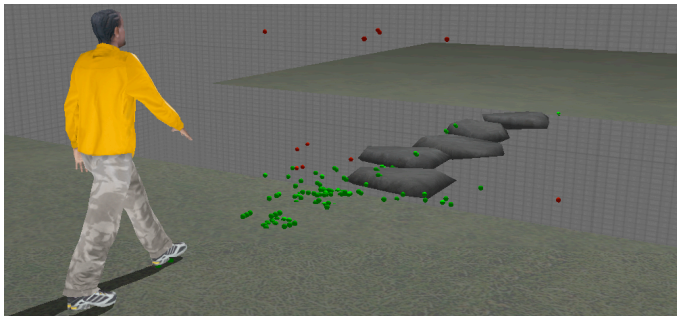

Fig. 4. State graph unrolling and pruning in our system. Green and red markers indicate supported and unsupported docks respectively.

C. Executing & Warping Actions

In our prototype system the final choice of which sequence of actions to execute is made by mapping user input to requests for specific actions.

For each dock in the chosen animation we use geo-queries (Section II-D) to return points on the scene geometry surface corresponding to the dock's contact points. We find the minimal rigid transformation between these point sets, and thus calculate the target position for the end-effector during that docking interval [7].

We adapt the action to the displaced docks as follows: reorient the entire action to minimise the displacements; find the root bone position that minimises the sum of squared distances between end-effectors and docks during constrained intervals; drive the root bone towards this target by integrating a damped-spring model; and finally, pose each constrained limb by analytical IK [8].

D. Geo-Queries

Given a docking template, we wish to find a location in the scene where the geometry can support the dock. The location should be as near as possible to the expected location of the dock so as to minimise the amount of motion deformation required to achieve a registration between motion and geometry. Currently we use a simple distance cut-off, determined experimentally, beyond which we expect the warped animation to be of unacceptable quality.

1) Version I: Query and Probe: To find the nearest feasible support within this allowed maximum distance we probe the region around the expected dock by sampling at spatial and angular intervals over the space. Angular sampling is done around the up-axis only as human motion is invariant under rotation about the gravitational axis but not other axes. For each sample (with position and angle) we test the feasibility of a dock at that location by the geo-query criterion described in Algorithm 1.

Algorithm 1 Feasibility Geo-Query

```
for all dock's points do
    ray cast along normal in positive and negative directions
    if (ray hit) and (hit distance is less than sampling increment) and (face normal is acceptable) then
        record hit
    end if
end for
if number of hits equals number of constraint points then
    support exists at this location
else
    support does not exist at this location
end if
```

The algorithm searches by raycast for a matching face for each docking contact point. The normal of the face must be within an error tolerance[1] cone about the docking contact normal. Raycasts need not proceed farther than sampling distance since scene features further away will be detected at other sampling locations.

The variable parameters of the algorithm, linear sampling increment, angular sampling increment and face-normal angle tolerance were determined experimentally depending on the scene and or type of docking template.

2) Version I: Results: Table I shows the total number of animations and the time taken to evaluate them (by probing for support locations) in an unrolled state graph at different unrolling depths. Increasing the linear and or angular sampling increments causes a linear growth in the evaluation time. The results did not allow for real-time animation. Therefore we explored ways of embedding meta-data in the scene using a pre-process to achieve real-time performance.

Tree Depth	Animations to Prune	Version I: Total Prune Time	Version II: Total Prune Time
1	5	0.9754 s	0.0005 s
2	32	9.1953 s	0.0056 s
3	197	75.1461 s	0.0538 s
4	1236	588.1479 s	0.4969 s

TABLE I
ANIMATION TREE DEPTH VS. PRUNING TIME. IN ORDER TO TEST UPPER BOUNDS AND EVALUATE ALL ANIMATIONS IN THE UNROLLED TREE, BRANCHES THAT FAILED EVALUATION WERE NOT ACTUALLY REMOVED.

3) Version II: Static Scene Optimisation: By shifting the expensive probing operation into a pre-process, we can make the feasibility queries more efficient but incur the overhead of needing to store the pre-processed data. The current algorithm also assumes static scene geometry.

Given the docking templates from our animation database we pre-process the environment to find regions where these docks are supported[2]. We store this information in octree structures, one for each type of docking template, which we refer to as support-trees. Given an expected dock location we perform a nearest-neighbour search over the support-tree to find whether the dock is supported in or near that location. This search replaces the expensive query-and-probe support search with a fast octree traversal. All other aspects of our system remain unchanged. An overview of our final system is shown in Figure 5.

4) Version II: Results: We present results for offline processing time cost, support-tree memory usage and geo-query time cost.

For a single dock the offline processing time and the number of ray-triangle intersection tests increases linearly with the number of dock points (Table II). Detecting increasingly complex docks only incurs a cost in the offline processing time.

[1]While face normals are compared to within a tolerance of 10 degrees in our implementation, this tolerance should be determined by the friction cone associated with contact surface.

[2]Similar docking templates are generalised and grouped, for example all flat foot docks.

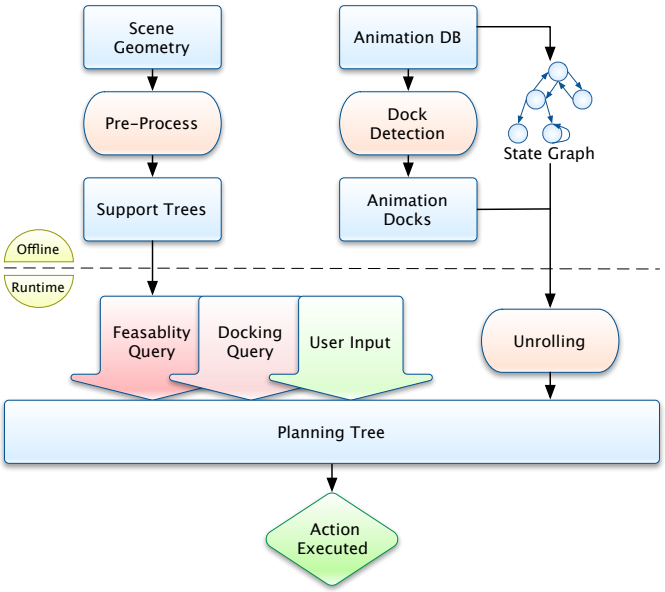

Fig. 5. An overview of the system with pre-process optimisation.

Dock Points	Ray-Triangle Tests	Processing Time
2	29956064	5.8797 s
4	58514624	12.118 s
6	95037024	16.6224 s
8	123839360	23.6544 s

TABLE II
NUMBER OF POINTS PER DOCK VS. PROCESSING TIME. THE SAMPLING WAS DONE AT 0.1 M INTERVALS IN A SCENE WITH VOLUME OF 48 M^3 AND SURFACE AREA OF 213 M^2.

The memory required to store a single support-tree node is approximately 112 bytes. The number of support-tree nodes for a given docking template in a scene depends on the sampling density and the scene surface area (or more generally, the number of docking support locations). The memory required to store a full support-tree, in this case for a foot plant dock was measured for several scenes (Table III). We choose a foot plant dock because it is a common requirement for all animations in our corpus.

Scene	Surface Area	Volume	Nodes	Memory
Stepping	213 m^2	48 m^3	13665	1.46 MB
Climbing	309 m^2	240 m^3	20009	2.14 MB
Typical Counter Strike Level	13240 m^2	127428 m^3	218017	23.28MB

TABLE III
MEMORY USAGE FOR A SUPPORT OCTREE.

At runtime, the number of animations to evaluate and consequently the time taken increases exponentially with search depth as shown in Table I. Future branches evaluated in the tree remain valid as branching choices are made. Hence for increasingly large unrolling depths a reentrant tree traversal and pruning strategy would aid in amortising time costs. The average time to evaluate a single animation using feasibility geo-query is approximately 0.1 ms, making real-time motion planning possible while remaining responsive to user input.

III. Conclusions & Future Work

Our results show that real-time motion planning with high fidelity interactions is possible but only with an offline pre-process. Pre-processing allows for the complexity of the docking templates to be de-coupled from run-time cost.

The system described allows real-time performance in planning for a responsive character in a scene. The following are areas of future work to extend our current system: auto generate docking templates and contact locations by analysing end-effector motion/physics; optimise memory usage by merging feasible regions in a support-tree; and extend solution to dynamic scenes.

Acknowledgments

The authors would like to thank Àlex Collado i Castells for his generous help in obtaining the motion capture data used in this paper. This research is funded by Science Foundation Ireland (SFI), under the Research Frontiers Programme Grant No. SFI/08/RFP/CMSF243.

References

[1] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," *ACM Transactions on Graphics (TOG*, vol. 21, no. 3, Jul 2002.
[2] M. Lau and J. Kuffner, "Precomputed search trees: planning for interactive goal-driven animation," *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Sep 2006.
[3] P. S. A. Reitsma and N. S. Pollard, "Evaluating motion graphs for character animation," *ACM Trans. Graph.*, vol. 26, no. 4, p. 18, 2007.
[4] T. Abaci, J. Cíger, and D. Thalmann, "Planning with smart objects." in *WSCG (Short Papers)*, 2005, pp. 25–28.
[5] K. H. Lee, M. G. Choi, and J. Lee, "Motion patches: building blocks for virtual environments annotated with motion data," in *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*. ACM, 2006, pp. 898–906.
[6] D. H. Hale, G. M. Youngblood, and P. N. Dixit, "Automatically-generated convex region decomposition for real-time spatial agent navigation in virtual worlds," in *AIIDE*, 2008.
[7] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-D point sets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, no. 5, pp. 698–700, September 1987.
[8] D. Tolani, A. Goswami, and N. I. Badler, "Real-time inverse kinematics techniques for anthropomorphic limbs," *Graph. Models Image Process.*, vol. 62, no. 5, pp. 353–388, 2000.