Part 7

Annotated Bibliographies

7.1 Graph-Based Motion Synthesis: An annotated bibliography by Michael Gleicher

Graph-Based Motion Synthesis: An annotated bibliography

Michael Lee Gleicher
Department of Computer Sciences
University of Wisconsin – Madison
gleicher@cs.wisc.edu
http://www.cs.wisc.edu/~gleicher

Version 0.1, May 28, 2008
Note: this is a work in progress. Hopefully, it has some value, even in its current form.

Caveat: this list is not meant to be encyclopedic, or the comments all-explaining.

I apologize in advance to everyone who has gotten left out. However, I think the list is more valuable if it isn't too overly long. I would like to say that these are the "most important" references, but really they are the ones that I had time to put in (biased towards things I am familiar with). If you want to find every last reference, look at the most recent papers and their references. Or use a good search tool and chase forward from the core references.

The intent is that this document is part of a longer tutorial/review/survey that tries to explain the methods listed here, and does some comparison between the approaches. But for now, I'm starting with this.

Citations are of the form [X.NNYY] where X is the section that the citation appears in.

The way I've divided this (the section headings) is:

A. The "Original" Motion Graph Papers (the trio from 2002)
B. Precursors to Motion Graphs (things that were Motion-Graph –like pre-2002)
C. Motion Editing Foundations (things that go into Motion Graphs)
D. Motion Blending (a foundation for Parametric Graphs)
E. Graphs for Interactive Control
F. Continuous (rather than discrete) control in graphs
G. Everything Else (a grab bag of other motion graph papers that I think are important)

A notable survey was put together by David Forsyth and some others:

[0.F++05] Forsyth, D. A.; Arikan, O.; Ikemoto, L.; O'Brien, J. & Ramanan, D. Computational studies of human motion: part 1, tracking and motion synthesis *Found. Trends. Comput. Graph. Vis.,* **2005***, 1*, 77-254.

This reference is particularly good at discussion some of the motion graph-like things done in the vision community. It has a very extensive (400+ reference) bibliography.

Annotated Bibliography

I have organized this list thematically. Sometimes, a paper might fall into more than one category, so I will need to use a cross-reference (I'll put papers where I think they fit best).

A. The "Original" Motion Graph Papers

At SIGGRAPH 2002, three papers came together that (arguably) marked the first real "motion graph" techniques in academia. While the core ideas had been kicking around (see Sections B and C), the idea of automatically assembling a graph and using this graph to synthesize motions that meet specific goals came together at this time. It is interesting that 3 groups came up with a very similar core idea at the same time independently. Some of this was that the area was ripe (with the fresh availability of data, etc.).

While these three papers all had the same core ideas (opportunistically build a graph from a collection of motion data, search the graph to construct motions that meet goals), the details were very different. And the notations used in each were quite different. A comparison of these approaches will be given in the Tutorial (when that gets written).

[A.KGP02] Kovar, L.; Gleicher, M. & Pighin, F. Motion Graphs. *ACM Transactions on Graphics,* **2002***, 21*, 473-482. (Proceedings SIGGRAPH 2002).

Lucas Kovar's thesis (and later papers) provide more details on some aspects of this work.

[A.AF02] Arikan, O. & Forsyth, D. A. Synthesizing Constrained Motions from Examples. *ACM Transactions on Graphics,* **2002***, 21*, 483-490. (Proceedings SIGGRAPH 2002)

See also [H.AFO03] which extends this work.

[A.L+02] Lee, J.; Chai, J.; Reitsma, P.; Hodgins, J. & Pollard, N. Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics,* **2002***, 21*, 491-500. (Proceedings SIGGRAPH 2002)

This paper had the most diverse set of example applications. The graph was "diven" by search (to follow a path, like in [A.KGP02]), user interaction, and computer vision.

Some places where the projects differed:

- *Transition scheme:* [KGP02] used blending, [AF02] used cuts and filtered the discontinuities, and [L+02] used cuts and displaced the endpoints to get continuity.
- *Method for getting sparsity in the graph:* [KGP02] found local minima of the cost function, [L+02] clustered poses and applied heuristics, and [AF02] clustered edges.
- *Search scheme:* [KGP02] used branch and bound, [AF02] used a probabalitic search of hierarchicy of simplified graphs, and [L+02] used greedy selection.

There were other papers at SIGGRAPH '02 that assembled motions to make new motions (and at least [B.LWS02] had a graph involved), but did not share the core ideas.

B. Precursors to Motion Graphs

Before us academics "invented" motion graphs, they were actually used in practice, particularly in the gaming industry. At the most basic level, computer games have almost always strung short movements together to make the continuously controllable characters that games need. The idea of encoding which movements could connect to which other movements (a motion graph) was in common practice – although the more common term was *Move Tree* (even though the graph usually was not a tree). What the 2002 (and later) academic motion graphs added was the idea of automatically (and opportunistically) building these graphs and using search methods to use them.

Unfortunately, there is little literature on what people were doing to create Move Trees from motion capture data in practice pre-2002. Generally, it involved careful planning to create motions that fit together, and a lot of manual effort to build the graphs themselves. There were several Game Developer Magazine articles about planning, but in general, there is little to document what is going on. A lot of the work was done by motion capture studios using proprietary tools. Some references that I am aware of:

[B.Men99] Menache, A. Understanding Motion Capture for Computer Animation and Video Games. *Morgan Kaufmann,* **1999.**

While some of this book is quite dated (referring to archaic hardware and considerations from old motion capture systems), its discussion of the planning process, and what really matters in motion capture data is still valuable.

[B.Kines98] Kines, M. Planning and Directing Motion Capture for Games. Game Developer Magazine, 1998. Also in GDC '98 proceedings, and in Gamasutra (online) at http://www.gamasutra.com/features/20000119/kines_01.htm.

[B.MBC01] Mizuguchi, M.; Buchanan, J. & Calvert, T. Data driven motion transitions for interactive games *Eurographics 2001 Short Presentations,* **2001**.

This is the commonly cited academic reference for the game industry practice.

Before "The Motion Graph Papers," several researchers had explored methods that assembled long motions from shorter segments, usually trying to create statistical models of possible motions.

[B.MTH00] Molina Tanco, L. & Hilton, A. Realistic synthesis of novel human movements from a database of motion capture examples. *In Proc. IEEE Workshop on Human Motion,* **2000.**

An early motion graph variant. Build an HMM on clusters of motion capture data, but retained the connection back to the original motions so that playback preserved the original poses.

[B.BH00] Brand, M. & Hertzmann, A. Style machines *Proceedings of ACM SIGGRAPH 2000,* **2000**, 183-192.

Build a Hidden Markov Model (which is a motion graph) from a collection of motion captured data. The density of this paper (it places a lot into the "black box" of the HMM) has made it less commonly referred to than subsequent papers, although many ideas seem to appear.

[B.GJH01] Galata, A.; Johnson, N. & Hogg, D. Learning Variable Length Markov Models of Behaviour. *Computer Vision and Image Understanding: CVIU,* **2001***, 81*, 398-413.

One of many computer vision papers that build a Hidden Markov Model for human movement. This one drew the attention of early motion graph projects, possibly because they had 3D mocap data as an example. See [0.F++05] for a more extensive list.

[B.LWS02] Li, Y.; Wang, T. & Shum, H. Motion texture: a two-level statistical model for character motion synthesis *ACM Tranactions on Graphics,* **2002***, 21*, 465-472. (Proceedings SIGGRAPH 2002)

While this paper appeared in the same session as the ones in Section A, it had a considerably different flavor because rather than being a kinematic model (that copied the frames), it produced a linear dynamic model that generated new movements (based on the statistics of the database). The switched-linear dynamic system is a kind of motion graph, but it doesn't resemble the other projects as much.

C. Motion Editing Foundations

Tools for working with motion capture data are the basis for motion graphs. Like Motion Graphs, the basic ideas were in practice before academics discovered them. Supposedly, a lot of the ideas of early motion editing research (presented in the mid- to late- 90s and afterwards) were part of the Symbolics animation system.

While there are quite a few papers on motion editing, I feel these early ones are particularly relevant.

[C.BW95] Bruderlin, A. & Williams, L. Motion signal processing *Proceedings of ACM SIGGRAPH 1995,* **1995**, 97-104.

[C.WP95] Witkin, A. P. & Popović, Z. Motion Warping. *Proceedings of SIGGRAPH 95,* **1995**, 105-108.

These papers really introduced (to the research community) the idea of thinking about editing motion capture data without having a model of what the various motion signals mean. I believe that many of the methods (particularly the blending and displacement or warping techniques) were in common practice – they just hadn't been shown to the research community.

[C.R++96] Rose, C.; Guenter, B.; Bodenheimer, B. & Cohen, M. F. Efficient generation of motion transitions using spacetime constraints. *Proceedings SIGGRAPH '96,* **1996**, 147-154.

This paper was the first (and still one of the only) papers in trying to develop better methods for creating transitions. Interestingly, while transitions are an essential piece of any Motion Graph method, there has been little research in developing more sophisticated transition techniques since this paper. In fact, almost nothing uses techniques as sophisticated as what was presented in this paper. [G.Z+05] is one thing that considers transitions more recently.

[C.RCB98] Rose, C.; Cohen, M. & Bodenheimer, B. Verbs and adverbs: multidimensional motion interpolation. *IEEE Computer Graphics and Application,* **1998***, 18*, 32-40.

This is the first academic example of using motion blending to do serious motion synthesis. They dealt with a high dimensional space (emotions, speed, …). In a sense, there is even a graph (since everything was looped). All of the connections were built manually (which is what will distinguish the later work), but this was a seminal early system. (Although, I think similar ideas had been in use in games, albeit with a mathematically simpler formulation).

[C.Gle98] Gleicher, M. Retargeting Motion to New Characters. *Proceedings of ACM SIGGRAPH 98,* **1998**, 33-42.

I think that this paper inspired a lot of people to start thinking about motion capture, since its one of the first that really talked about preservation of quality. Maybe I'm biased since I wrote it and overstate its importance. On the other hand, the techniques I was trying to use didn't scale, and have been replaced in later work.

[C.LS99] Lee, J. & Shin, S. Y. A Hierarchical Approach to Interactive Motion Editing for Human-Like Figures. *Proceedings of ACM SIGGRAPH 99,* **1999**, 39-48.

This paper is important because it was the first to show a really practical approach for motion editing, which is important since it's a backbone of motion graphs.

[C.Gle01] Gleicher, M. Motion Path Editing. *Proceedings 2001 ACM Symposium on Interactive 3D Graphics,* **2001**.

An alternative to motion graphs – edit motions rather than synthesize new ones. This was the thing that started our motion graph project, and I still think it's a valuable addition to the techniques – a first attempt to combine them was [F.SKG05].

D. Motion Blending

Motion blending is an important tool for creating new motions, but is usually distinct from the concatenation approaches that are the basis of Motion Graphs. I describe a few significant papers here, because they may provide a future for Motion Graph techniques (see the hybrid techniques in Section F). Motion blending simply does a pose-wise blending of each parameter (after appropriate alignments).

[C.RCB98] (as well as the things in the gaming world that used similar ideas) showed how powerful motion blending could be. However, these systems required manual construction of the blends (to find

the corresponding motions and the correspondences between motions). Methods from research have automated this process, and allowed for more complex blends.

[D.KG03] Kovar, L. & Gleicher, M. Flexible Automatic Motion Blending with Registration Curves. *Proceedings of the Symposium on Computer Animation,* **2003**.

This paper automatically found timing, coordinate frame, and constraint correspondences between two motions, and used these correspondences to make complex blends between dissimilar motions. It really showed that blending can be a very general tool.

[D.KG04] Kovar, L. & Gleicher, M. Automated extraction and parameterization of motions in large data sets. *ACM Transactions on Graphics,* **2004***, 23*, 559-568. (Proceedings SIGGRAPH 2004)

This paper searched a database for similar motions (where similar was roughly defined by "blendable"), and created blend-spaces of those motions.

[D.SH06] Safonova, A. & Hodgins, J. K. Analyzing the physical correctness of interpolated human motion. *Proceedings of the Symposium on Computer Animation,* **2005**, 171-180.

It is quite remarkable that something as simple as motion blending works at all. This paper tries to explain why it works so well, as well as giving some ideas as when it doesn't work, and suggesting some things to do about those cases. This paper only considers the trivial case of aligned blending (not the case where time warping or other alignment methods such as [D.KG03] have been applied).

E. Graphs for Interactive Control

The use of Motion Graphs has its roots in interactive applications (see Section B) where hand-constructed "Move Trees" have been used for quite some time. The first "Motion Graph" projects (and much of the research that followed) were predominantly *off-line* synthesis methods. That is, the entire set of goals is given at the beginning and the entire motion is produced. Such methods can be efficient (for example, even in 2002, [A.KGP02] could produce motions faster than real time – to create a 15 second animation would take less than 15 seconds). However, they are inappropriate for interactive settings where the future goals are not known, and instant response to control is required.

For interactive applications, we usually want local control: given the current situation, what is the right thing to do now. This is important for creating characters that respond to use input, and useful for making characters that efficiently move towards goals.

Part of the problem with most Motion Graph methods (particularly the 2002 ones) is that the graph is built opportunistically, so that finding good choices on it requires search. This is a key difference between the automatically constructed graphs of the academic research, and the manually constructed ones used in practice. Some techniques provide structured graphs automatically. These structured graphs are built such that the graph structure is simple and there are always plenty of choices.

[E.G+03] Gleicher, M.; Shin, H. J.; Kovar, L. & Jepsen, A. Snap Together Motion: Assembling Run-Time Animation. *Proceedings of the Symposium on Interactive 3D Graphics,* **2003**.

This system automates the construction of graphs that have the same kind of structure that the hand-made graphs used in games have. It automatically (or semi-automatically) finds hub nodes for which a large number of options (in and out edges) are available. The graphs are useful for interactive control, and their structure also makes them conducive to simple search methods (such as using a greedy selection) to drive characters towards goals (see [F.SKG05]).

[E.TLP07] Treuille, A.; Lee, Y. & Popović, Z. Near-optimal character animation with continuous control. *ACM Trans. Graph.,* **2007***, 26*, 7. (Proceedings SIGGRAPH 2007)

This paper uses a very simple motion graph (one specifically built with each node being a single step of locomotion, and all steps connectable by blends). However, it generates a sophisticated controller for steering the character towards a goal by figuring out what the optimal choice of the next step should be. It can make "optimal" choices by effectively looking infinitely far into the future (by assuming it will also make optimal choices on future steps). Note that the continuous in the title refers to continuity of the state space: like most other motion graph approaches this system makes discrete choices (the next graph arc to follow) at discrete times (the choice points). See section F for continuous approaches.

Rather than building graphs with special structure, many attempts to use Motion Graphs instead keep the unstructured graph and use some mechanism to process it such that efficient, local decision making can be done. Some examples of such an approach include:

[E.LL04] Lee, J. & Lee, K. H. Precomputing avatar behavior from human motion data. *SCA '04: Proceedings of the Symposium on Computer Animation,* **2004**, 79-87.

Use reinforcement-learning inspired searches to expand a number of search trees from given states so that they can cache the correct path on the motion graph to take in any situation.

[E.LK06] Lau, M. & Kuffner, J. J. Precomputed Search Trees: Planning for Interactive Goal-Driven Animation. *Proceedings of the Symposium on Computer Animation,* **2006**, 299-308.

This work similarly pre-computes searches and caches the best choice for different situations. It is used for doing efficient motion synthesis by coupling with a higher-level planner, but it could also be used for interactive control.

[E.MP07] McCann, J. & Pollard, N. Responsive characters from motion fragments *ACM Trans. Graph.,* **2007***, 26*, 6. (Proceedings SIGGRAPH 2007)

The basic idea is to build a table of what the right choice of motion to play next is given the current configuration. In addition to using the control inputs to determine what the next

motion to play is, it tries to predict future control inputs so that it can make even more informed choices. This method is (the first that I know of) capable of making tradeoffs of motion quality in order to achieve responsiveness – an issue that will certainly be considered more in the future.

F. Continuous Motion Graphs

All of the motion graph techniques described above make discrete choices: at each node in the graph, a decision from a small set (the outgoing edges) must be chosen. This limits the degree of precision in control: goals might be "in-between" the available choices. This means that a large number of choices (i.e. a big database) might be required to be able to get anywhere near goals, and also brings up the possibility of searches returning roundabout paths that get closer to the goals by taking indirect routes.

Hybrid techniques mix the discrete graph methods with some other mechanism that can provide a continuous range of choices.

One category of hybrid techniques is to use a discrete motion graph to create a rough motion that gets close to its goals, and then use motion editing to adjust these motions such that they precisely meet the goals. Surprisingly, this simple approach has not been considered too much in the literature. One example that I am aware of:

[F.SKG05] Sung, M.; Kovar, L. & Gleicher, M. Fast and accurate goal-directed motion synthesis for crowds *Proceedings of the Symposium on Computer Animation,* **2005**.

Uses a Probabilistic Roadmap (PRM) planner to get a rough path, then uses a Snap-Together Motion Graph [E.G+03] to greedily follow the path, and then adjusts the path using motion editing to precisely meet the goals.

Another category of hybrid techniques uses motion blending techniques to provide the continuously controllable choices and graph-based methods to connect the blended segments together. This is what is done in games that do motion blending, however the graphs (and blend) structures are typically built by hand. The Verbs and Adverbs system [C.RCB98] is an example of this (with a simple, hand-made graph structure). Special case hybrid "blending graphs" have been used for locomotion (some of the locomotion papers from KAIST are of this form).

A few recent papers have considered general approaches that automatically construct hybrid (i.e. with blending) graphs for situations more general than just locomotion.

[F.SO06] Shin, H. J. & Oh, H. S. Fat graphs: constructing an interactive character with continuous controls. *Proceedings of the Symposium on Computer Animation*, 2006, 291-298.

This paper extends Snap-Together Motion [E.G+03] such that the edges between nodes are blended to be continuously controllable. (in this notation, nodes are choice points and edges are motions). All motions that come into (or out of) a node share a common pose.

[F.HG07] Heck, R. & Gleicher, M. Parametric Motion Graphs. *Proceedings of the Symposium on Interactive 3D Graphics*, 2007.

This paper builds a graph where the nodes are motion blends, and the edges are transitions between the entire spaces of possible motions created by the blends.

The above two papers have been used for interactive control. In [HG07] we experimented with greedily driving characters towards goals, but this only works for goals of limited complexity. The following paper considers mixing blending and off-line motion graph search.

[F.SH07] Safonova, A. & Hodgins, J. K. Construction and optimal search of interpolated motion graphs. *ACM Transactions on Graphics,* **2007***, 26*, 106.

This creates a parametric (continuous control) motion graph by blending together all pairs of motions that can be blended using simple blending (as opposed to [F.HG07] that used the complex alignment procedure of [D.KG04], or [F.HG07] and [F.SO06] that both consider multi-way blends). However, it does use (very sophisticated) searching techniques to search these very large graphs to motions that precisely meet complex sets of goals.

Technically, this is still a discrete motion graph: the motion blends are sampled to provide a discrete set of motions. However, the intent of the work is to provide a searchable parametric graph, and the results impressively meet the goals, even though the implementation of the continuous part of the discrete/continuous optimization is done in a brute-force manner.

H. Everything Else

Since I don't have time to organize the rest of this, here's a grab-bag of assorted motion graph papers that I feel are significant. This is a sub-sampling of the papers out there. Just because something is missing doesn't mean it isn't significant.

[H.AFO03] Arikan, O.; Forsyth, D. A. & O'Brien, J. F. Motion Synthesis From Annotations. *ACM Transactions on Graphics,* **2003***, 22*, 402-408. (Proceedings SIGGRAPH 2003)

Adds to the original motion graph papers (especially [A.AF02]) by automatically figuring out labels that can be used to constrain the motion. Also provides a dynamic programming approach to searching for motions that seems less ad hoc than the original.

[H.Z+05] Zordan, V. B.; Majkowska, A.; Chiu, B. & Fast, M. Dynamic response for motion capture animation. *ACM Transactions on Graphics,* **2005***, 24*, 697-701. (Proceedings SIGGRAPH 2005)

This paper considers how to get "on and off" a motion graph. It significant because it provides methods of generating movement with a motion graph, using physical simulation (for example to have a character get hit), and then re-connecting with the motion graph. It is really the only thing since [C.R++96] to consider more complex transitions.

[H.IAF07] Ikemoto, L.; Arikan, O. & Forsyth, D. Quick transitions with cached multi-way blends. *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games, ACM,* **2007**, 145-151.

Provides a scheme for creating more blends for motion graphs by mixing in other motions over longer periods of time. Has some interesting ideas on how to evaluate the quality of transitions, and uses this to perform a generate and test approach.

[H.RP07] Reitsma, P. S. A. & Pollard, N. S. Evaluating motion graphs for character animation. ACM *Transactions on Graphics,* **2007***, 26*, 18.

Considers how good a motion graph is for locomotion by thoroughly evaluating the set of places that the discrete choices can get you to. One outcome is an interesting variant of a motion graph that embeds all the choices into an environment allowing for easy locomotion planning. This paper extends an earlier conference paper.

[H.WB08] Wang, J. & Bodenheimer, B. Synthesis and evaluation of linear motion transitions. *ACM Transactions on Graphics,* **2008***, 27*, 1-15.

This paper thoroughly considers the perceptual issues in the simple blending schemes used in many (most?) motion graph systems. While they focus on a specific portion of the design space, their careful consideration of the perceptual issues is interesting even if one considers using better choices for various parts of the algorithms.

7.2 Manipulation Planning for Virtual Humans Summary of Representative Papers by Marcelo Kallmann

Manipulation Planning for Virtual Humans
Summary of Representative Papers
Marcelo Kallmann

Several representative papers are discussed in each part of the course notes and I summarize here a selection of these papers, in order to provide a short document offering a quick roadmap of the relevant literature.

Once again, the reader should be warned that several other important publications are available and the proposed papers here are in no way the only relevant ones, and I apologize in advance for all eventual important omission.

1. Motion Interpolation Methods Relevant to Manipulation Planning

> [Rose et al 2001] C. Rose III, P-P. Sloan, and M. Cohen, "Artist-Directed Inverse-Kinematics Using Radial Basis Function Interpolation", Computer graphics Forum, 20(3), 2001, 239-250.

In this paper Rose and colleagues propose to extend the “verbs-and-adverbs” interpolation method based on Radial Basis Functions with additional techniques specifically designed for applying motion interpolation to control end-effectors, in particular the hand of a character. The presented techniques are therefore relevant to the development of data-based motion planning algorithms.

> [Kovar and Gleicher 2004] L. Kovar and M. Gleicher, "Automated Extraction and Parameterization of Motions in Large Data Sets", ACM Trans. on Graphics (Proc. of SIGGRAPH 2004).
>
> [Safonova and Hodgins 2007] A. Safonova and J. Hodgins, "Construction and Optimal Search of Interpolated Motion Graphs", ACM Trans. on Graphics (Proc. of SIGGRAPH 2007).
>
> [Heck and Gleicher 2007] R. Heck and M. Gleicher, "Parametric Motion Graphs", Proc. of Interactive 3D Graphics and Games (I3D 2007).

The three papers above well represent methods designed to reuse motion capture data in order to enable motion interpolation and parameterization. Although these methods are not in particular proposed for manipulation planning or for controlling arms, motion planners could be developed to operate in the created parametric spaces.

2. Inverse Kinematics

> [Maciejewski and Klein 1985] A. Maciejewski and C. Klein, "Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments," Intl. Journal of Robotics Research, 4(3), 1985, 109–117.
>
> [Baerlocher 2001] P. Baerlocher, "Inverse Kinematics Techniques for the Interactive Posture Control of Articulated Figures", PhD Thesis 2383, Swiss Federal Institute of Technology (EPFL), 2001.

Inverse Kinematics is the tool-of-choice for controlling open Kinematic chains for instance for manipulation planning. The work of Maciejewski and Klein provide important extensions

to the Jacobian-based IK formulation in order to integrate collision avoidance. The work of Baerlocher proposes extensions for handling several prioritized humanlike tasks in an unified framework and is also a comprehensive document on the topic (it is actually a PhD thesis).

[Tolani et al 2000] D. Tolani, A. Goswami, and N. Badler, "Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs", Graphical Models and Image Processing, 62(5), 2000, 353-388.

[Kallmann 2008] M. Kallmann, "Analytical Inverse Kinematics with Body Posture Control", Computer Animation and Virtual Worlds, 19(2), May 2008, 79-91.

The work of Tolani and colleagues introduces specific analytical solutions for humanlike arms and legs. As the analytical solution is extremely fast (requires only a fixed number of mathematical operations) it is therefore very useful for controlling arms and correcting constraints, in particular if IK has to be called from sampling routines of planners. The work in [Kallmann 2008] provides extensions for dealing with collision avoidance and joint limits avoidance, which turns out to be valuable for determining useful variations of the swivel angle.

3. Main Motion Planning Methods for Manipulation

[Koga et al 1994] Y. Koga, K. Kondo, J. Kuffner, and J.-C. Latombe, "Planning motions with intentions", In Proc. of SIGGRAPH 1994, 395-408.

[Kuffner and Latombe 2000] J. Kuffner and J.-C. Latombe, "Interactive Manipulation Planning for Animated Characters", In Pacific Graphics 2000 (short paper).

[Kallmann 2005] M. Kallmann, "Scalable Solutions for Interactive Virtual Humans that can Manipulate Objects", AIIDE 2005.

The work of Koga and colleagues represents the first application of a motion planner to an animated character. This work also presents excellent videos of cooperative arms manipulating objects and as well IK extensions for producing humanlike arm postures. Later on, Kuffner and Latombe present the first application of RRTs to virtual characters, and in [Kallmann 2005] options for integrating RRTs with the analytical IK are explored.

[Kallmann et al 2003] M. Kallmann, A. Aubel, T. Abaci, and D. Thalmann, "Planning Collision-Free Reaching Motions for Interactive Object Manipulation and Grasping", Proceedings of Eurographics 2003, 313-322.

[Yamane et al 2004] K. Yamane, J. Kuffner, and J. Hodgins, "Synthesizing Animations of Human Manipulation Tasks." ACM Trans. on Graphics (Proc. of SIGGRAPH 2004), 2004.

The two papers above present important extensions to the basic application of motion planners to virtual humans. In [Kallmann et al 2003] several techniques are presented for achieving full-body manipulations and in particular several heuristics for producing meaningful manipulation poses during sampling. Later on Yamane and colleagues propose an IK method which produces humanlike postures by relying on a posture database built with motion capture, greatly improving the realism of planned full-body motions for manipulations. Several excellent examples are produced by these works.

4. Coordination Planning for Humanoid Structures

[Kuffner et al 2001] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, "Footstep planning among obstacles for biped robots", In Proc. of IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS 2001).

[Kallmann et al 2004] M. Kallmann, R. Bargmann and M. Mataric´, "Planning the Sequencing of Movement Primitives", in Proc. of the Int'l Conference on Simulation of Adaptive Behavior (SAB), 2004, 193-200.

A coordination planning approach specifically addressing footstep placements for humanoid locomotion is proposed in [Kuffner et al 2001]. The method is based on a discrete search determining the best sequencing of pre-computed leg motions in order to plan locomotion sequences among obstacles. In [Kallmann et al 2004] this approach is re-formulated as a multi-mode RRT problem for generic primitive motion controllers, without the need of pre-computed motions.

[Kalisiak and van de Panne 2001] M. Kalisiak and M. van de Panne, "A Grasp-Based Motion Planning Algorithm for Character Animation", The Journal of Visualization and Computer Animation 12(3), 2001, 117-129.

[Bretl 2006] T. Bretl, "Motion Planning of Multi-Limbed Robots Subject to Equilibrium Constraints: The Free-Climbing Robot Problem", Int'l Jounal of Robotics Research 25(4), 2006, 317-342.

Coordination planning for climbing is equivalent to planning coordinated motions for manipulating the environment. A climbing planner for a 2D character has been proposed by [Kalisiak and van de Panne 2001] and algorithms for real robots that can climb are proposed in [Bretl 2006].

[Hauser et al 2007] K. Hauser, V. Ng-Thowhing, and H. Gonzalez-Baños, "Multi-Modal Motion Planning for a Humanoid Robot Manipulation Task", In Proc. of the Int'l Symposium on Robotics Research (ISRR) 2007.

A good example of a body coordination planner for manipulation tasks is given by Hauser and colleagues for the coordination of pushing objects with locomotion. The work is applied to Honda's ASIMO.

[Shapiro et al 2007] A. Shapiro, M. Kallmann, and P. Faloutsos, "Interactive Motion Correction and Object Manipulation", ACM SIGGRAPH Symposium on Interactive 3D graphics and Games (I3D), 2007.

The work proposed by Shapiro et al show how realistic results can be obtained by combining a sampling-based planner with motion captured locomotion sequences.

5. Learning for Motion Planning

[Leven and Hutchinson 2000] P. Leven and S. Hutchinson, "Toward Real-Time Path Planning in Changing Environments", Proc. of the fourth International Workshop on the Algorithmic Foundations of Robotics (WAFR), March 2000, pp. 363-376.

[Kallmann and Mataric' 2004] M. Kallmann and M. Mataric´, "Motion Planning Using Dynamic Roadmaps", In Proc. of ICRA 2004, 4399-4404.

[Burns and Brock 2005] B. Burns and O. Brock, "Sampling-Based Motion Planning Using Predictive Models", In Proc. of ICRA, 2005.

[Jiang and Kallmann 2007] X. Jiang and M. Kallmann, "Learning Humanoid Reaching Tasks in Dynamic Environments", In Proc. of IROS, 2007.

These four papers present different approaches for integrating learning with motion planning. The first two are based on the notion of addressing dynamic environments with dynamic roadmaps. The approach proposed in [Burns and Brock 2005] learns a model of the environment for improving sampling in difficult regions of the search space, and in [Jiang and Kallmann 2007] features are learned from previously planned motions and reused to help solving new tasks in environments and tasks with few variations.

Final Remarks

I hope that this compilation of papers give a good picture of the several methods available for each of the categories listed above, and that these notes will inspire exciting new developments in this area !

Part 8

Appendix

8.1 Motion Graphs

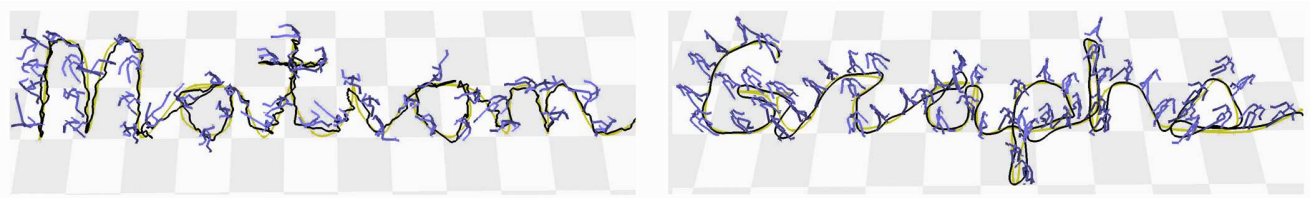
To appear in Proceedings of SIGGRAPH '02

Motion Graphs

Lucas Kovar
University of Wisconsin-Madison

Michael Gleicher*
University of Wisconsin-Madison

Frédéric Pighin†
University of Southern California
Institute for Creative Technologies



Abstract

In this paper we present a novel method for creating realistic, controllable motion. Given a corpus of motion capture data, we automatically construct a directed graph called a *motion graph* that encapsulates connections among the database. The motion graph consists both of pieces of original motion and automatically generated transitions. Motion can be generated simply by building walks on the graph. We present a general framework for extracting particular graph walks that meet a user's specifications. We then show how this framework can be applied to the specific problem of generating different styles of locomotion along arbitrary paths.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: motion synthesis, motion capture, animation with constraints

1 Introduction

Realistic human motion is an important part of media like video games and movies. More lifelike characters make for more immersive environments and more believable special effects. At the same time, realistic animation of human motion is a challenging task, as people have proven to be adept at discerning the subtleties of human movement and identifying inaccuracies.

One common solution to this problem is motion capture. However, while motion capture is a reliable way of acquiring realistic human motion, by itself it is a technique for *reproducing* motion. Motion capture data has proven to be difficult to modify, and editing techniques are reliable only for small changes to a motion. This limits the utility of motion capture — if the data on hand isn't sufficiently similar to what is desired, then often there is little that can be done other than acquire more data, a time-consuming and expensive process. This in particular is a problem for applications that require motion to be synthesized dynamically, such as interactive environments.

Our goal is to retain the realism of motion capture while also giving a user the ability to control and direct a character. For example, we would like to be able to ask a character to walk around a room without worrying about having a piece of motion data that contains the correct number of steps and travels in the right directions. We also need to be able to direct characters who can perform multiple actions, rather than those who are only capable of walking around.

This paper presents a method for synthesizing streams of motions based on a corpus of captured movement while preserving the quality of the original data. Given a set of motion capture data, we compile a structure called a *motion graph* that encodes how the captured clips may be re-assembled in different ways. The motion graph is a directed graph wherein edges contain either pieces of original motion data or automatically generated transitions. The nodes then serve as choice points where these small bits of motion join seamlessly. Because our methods automatically detect and create transitions between motions, users needn't capture motions specifically designed to connect to one another. If desired, the user can tune the high-level structure of the motion graph to produce desired degrees of connectivity among different parts.

Motion graphs transform the motion synthesis problem into one of selecting sequences of nodes, or *graph walks*. By drawing upon algorithms from graph theory and AI planning, we can extract graph walks that satisfy certain properties, thereby giving us control over the synthesized motions.

To demonstrate the potential of our approach, we introduce a simple example. We were donated 78.5 seconds of motion capture, or about 2400 frames of animation, of a performer randomly walking around with both sharp and smooth turns. Since the motion was donated, we did not carefully plan out each movement, as the literature suggests is critical to successful application of motion capture data [Washburn 2001]. From this data we constructed a motion graph and used an algorithm described later in this paper to extract motions that travelled along paths sketched on the ground. Characteristic movements of the original data like sharp turns were automatically used when appropriate, as seen in Figure 1.

It is possible to place additional constraints on the desired motion. For example, we noticed that part of the motion had the character sneaking around. By labelling these frames as special, we were able to specify that at certain points along the path the character must only use sneaking movements, and at other parts of the motion it must use normal walking motions, as is also shown in Figure 1.

*e-mail:{kovar,gleicher}@cs.wisc.edu
†e-mail:pighin@ict.usc.edu

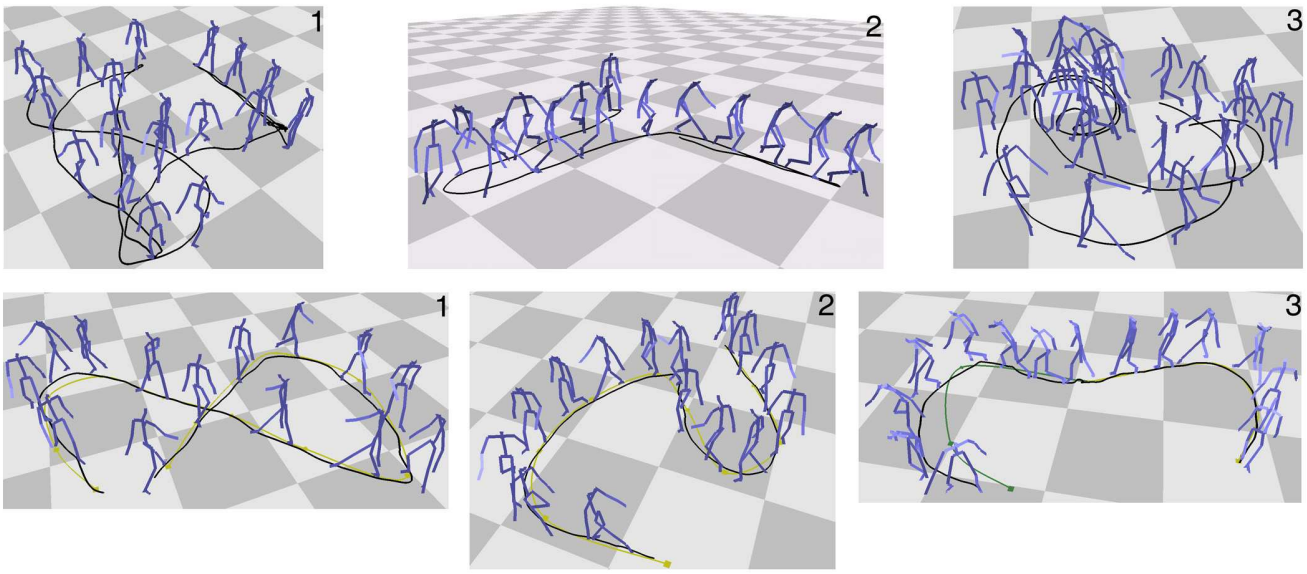


Figure 1: The top images show original motion capture data; two are walking motions and one is a sneaking motion. The black curves show the paths travelled by the character. The bottom images show new motion generated by a motion graph built out of these examples plus their mirror images. Images 1 and 2 show the result of having the motion graph fit walking motion to the smooth yellow paths. The black curve is the actual position of the center of mass on each frame. Image 3 shows motion formed by having the character switch from walking to sneaking halfway down the path.

The remainder of this paper is organized as follows. In Section 2 we describe related work. In Section 3 we describe how a motion graph is constructed from a database of motion capture. In Section 4 we set forth a general framework for extracting motion from the motion graph that meets user specifications. Section 5 discusses the specific problem of generating movements along a path and how it is handled in our framework. We conclude in Section 6 with a discussion of the scalability of our approach to large data sets and potential future work.

2 Related Work

Much previous work with motion capture has revolved around editing individual clips of motion. Motion warping [Witkin and Popović 1995] can be used to smoothly add small changes to a motion. Retargeting [Gleicher 1998; Lee and Shin 1999] maps the motion of a performer to a character of different proportions while retaining important constraints like footplants. Various signal processing operations [Bruderlin and Williams 1995] can be applied to motion data. Our work is different from these efforts in that it involves creating continuous streams of motion, rather than modifying specific clips.

One strategy for motion synthesis is to perform multi-target blends among a set of examples, yielding a continuous space of parameterized motion. Wiley and Hahn [1997] used linear interpolation to create parameterizations of walking at various inclinations and reaching to various locations. Rose et al. [1998] used radial basis functions to blend among clips representing the same motion performed in different styles. These works have a focus complementary to ours: while they are mainly concerned with generating parameterizations of individual clips, we are concerned with constructing controllable sequences of clips.

Another popular approach to motion synthesis is to construct statistical models. Pullen and Bregler [2000] used kernel-based probability distributions to synthesize new motion based on the statistical properties of example motion. Coherency was added to the model by explicitly accounting for correlations between parameters. Bowden [2000], Galata et al. [2001], and Brand and Hertzmann [2000] all processed motion capture data by constructing abstract "states" which each represent entire sets of poses. Transition probabilities between states were used to drive motion synthesis. Since these statistical models synthesize motion based on abstractions of data rather than actual data, they risk losing important detail. In our work we have tighter guarantees on the quality of generated motion. Moreover, these systems did not focus on the satisfaction of high-level constraints.

We generate motion by piecing together example motions from a database. Numerous other researchers have pursued similar strategies. Perlin [1995] and Perlin and Goldberg [1996] used a rule-based system and simple blends to attach procedurally generated motion into coherent streams. Faloutsos et al. [2001] used support vector machines to create motion sequences as compositions of actions generated from a set of physically based controllers. Since our system involves motion capture data, rather than procedural or physically based motion, we require different approaches to identifying and generating transitions. Also, these systems were mainly concerned with appropriately generating individual transitions, whereas we address the problem of generating entire motions (with many transitions) that meet user-specified criteria. Lamouret and van de Panne [1996] developed a system that used a database to extract motion meeting high-level constraints. However, their system was applied to a simple agent with five degrees of freedom, whereas we generate motion for a far more sophisticated character. Molina-Tanco and Hilton [2000] used a state-based statistical model similar to those mentioned in the previous paragraph to rearrange segments of original motion data. These segments were attached using linear interpolation. The user could create motion by selecting keyframe poses, which were connected with a high-probability sequence of states. Our work considers more general and sophisticated sets of constraints.

Work similar to ours has been done in the gaming industry to meet the requirements of online motion generation. Many companies use

move trees [Mizuguchi et al. 2001], which (like motion graphs) are graph structures representing connections in a database of motion. However, move trees are created manually — short motion clips are collected in carefully scripted capture sessions and blends are created by hand using interactive tools. Motion graphs are constructed automatically. Also, move trees are typically geared for rudimentary motion planning ("I want to turn left, so I should follow this transition"), as opposed to more complicated objectives.

The generation of transitions is an important part of our approach. Early work in this area was done by Perlin [1995], who presented a simple method for smoothly interpolating between two clips to create a blend. Lee [2000] defined orientation filters that allowed these blending operations to be performed on rotational data in a more principled fashion. Rose et al. [1996] presented a more complex method for creating transitions that preserved kinematic constraints and basic dynamic properties.

Our main application of motion graphs is to control a character's locomotion. This problem is important enough to have received a great deal of prior attention. Because a character's path isn't generally known in advance, synthesis is required. Procedural and physically based synthesis methods have been developed for a few activities such as walking [Multon et al. 1999; Sun and Metaxas 2001] and running [Hodgins et al. 1995; Bruderlin and Calvert 1996]. While techniques such as these can generate flexible motion paths, the current range of movement styles is limited. Also, these methods do not produce the quality of motion attainable by hand animation or motion capture. While Gleicher [2001] presented a method for editing the path traversed in a clip of motion capture, it did not address the need for continuous streams of motion, nor could it choose which clip is correct to fit a path (e.g. that a turning motion is better when we have a curved path).

Our basic approach — detecting transitions, constructing a graph, and using graph search techniques to find sequences satisfying user demands — has been applied previously to other problems. Schödl et al. [2000] developed a similar method for synthesizing seamless streams of video from example footage and driving these streams according to high-level user input.

Since writing this paper, we have learned of similar work done concurrently by a number of research groups. Arikan and Forsythe [2002] constructed from a motion database a hierarchical graph similar to ours and used a randomized search algorithm to extract motion that meets user constraints. Lee et al. [2002] also constructed a graph and generated motion via three user interfaces: a list of choices, a sketch-based interface similar to what we use for path fitting (Section 5), and a live video feed. Pullen and Bregler [2002] keyframed a subset of a character's degrees of freedom and matched small segments of this keyframed animation with the lower frequency bands of motion data. This resulted in sequences of short clips forming complete motions. Li et al [2002] generated a two-level statistical model of motion. At the lower level were linear dynamic systems representing characteristic movements called "textons", and the higher level contained transition probabilities among textons. This model was used both to generate new motion based on user keyframes and to edit existing motion.

3 Motion Graph Construction

In this section, we define the motion graph structure and the procedure for constructing it from a database of clips.

A clip of motion is defined as a regular sampling of the character's parameters, which consist of the position of the root joint and quaternions representing the orientations of each joint. We also allow clips (or, more generally, sets of frames) to be annotated with other information, such as descriptive labels ("walking," "karate") and constraint information (left heel must be planted on these frames).

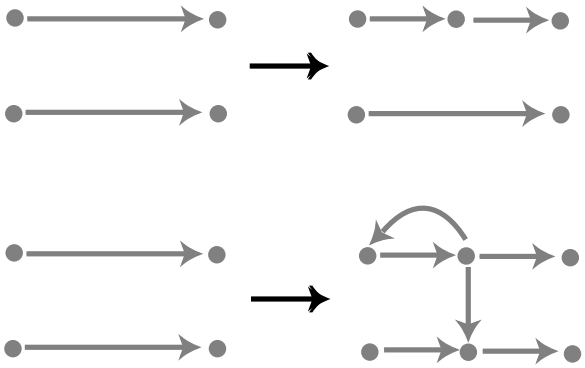


Figure 2: Consider a motion graph built from two initial clips. (top) We can trivially insert a node to divide an initial clip into two smaller clips. (bottom) We can also insert a transition joining either two different initial clips or different parts of the same initial clip.

A motion graph is a directed graph where all edges correspond to clips of motion. Nodes serve as choice points connecting these clips, i.e., each outgoing edge is potentially the successor to any incoming edge. A trivial motion graph can be created by placing all the initial clips from the database as arcs in the graph. This creates a disconnected graph with $2n$ nodes, one at the beginning and end of each clip. Similarly, an initial clip can be broken into two clips by inserting a node, since the later part of the motion is a valid successor to the earlier part (see Figure 2).

A more interesting graph requires greater connectivity. For a node to have multiple outgoing edges, there must be multiple clips that can follow the clip(s) leading into the node. Since it is unlikely that two pieces of original data are sufficiently similar, we need to create clips expressly for this purpose. *Transitions* are clips designed such that they can seamlessly connect two segments of original data. By introducing nodes within the initial clips and inserting transition clips between otherwise disconnected nodes, we can create a well-connected structure with a wide range of possible graph walks (see Figure 2).

Unfortunately, creating transitions is a hard animation problem. Imagine, for example, creating a transition between a run and a backflip. In real life this would require several seconds for an athlete to perform, and the transition motion looks little like the motions it connects. Hence the problem of automatically creating such a transition is arguably as difficult as that of creating realistic motion in the first place. On the other hand, if two motions are "close" to each other then simple blending techniques can reliably generate a transition. In light of this, our strategy is to identify portions of the initial clips that are sufficiently similar that straightforward blending is almost certain to produce valid transitions.

The remainder of this section is divided into three parts. First we describe our algorithm for detecting a set of candidate transition points. In the following two sections we discuss how we select among these candidate transitions and how blends are created at the chosen transition points. Finally, we explain how to prune the graph to eliminate problematic edges.

3.1 Detecting Candidate Transitions

As in our system, motion capture data is typically represented as vectors of parameters specifying the root position and joint rotations of a skeleton on each frame. One might attempt to locate transition points by computing some vector norm to measure the difference between poses at each pair of frames. However, such a simple approach is ill-advised, as it fails to address a number of important issues:

1. Simple vector norms fail to account for the meanings of the parameters. Specifically, in the joint angle representation some parameters have a much greater overall effect on the character than others (e.g., hip orientation vs. wrist orientation). Moreover, there is no meaningful way to assign fixed weights to these parameters, as the effect of a joint rotation on the shape of the body depends on the current configuration of the body.
2. A motion is defined only up to a rigid 2D coordinate transformation. That is, the motion is fundamentally unchanged if we translate it along the floor plane or rotate it about the vertical axis. Hence comparing two motions requires identifying compatible coordinate systems.
3. Smooth blends require more information than can be obtained at individual frames. A seamless transition must account not only for differences in body posture, but also in joint velocities, accelerations, and possibly higher-order derivatives.

Our similarity metric incorporates each of these considerations. To motivate it, we note that the skeleton is only a means to an end. In a typical animation, a polygonal mesh is deformed according to the skeleton's pose. This mesh is all that is seen, and hence it is a natural focus when considering how close two frames of animation are to each other. For this reason we measure the distance between two frames of animation in terms of a point cloud driven by the skeleton. Ideally this point cloud is a downsampling of the mesh defining the character.

To calculate the distance $D(\mathscr{A}_i, \mathscr{B}_j)$ between two frames $\mathscr{A}_i$ and $\mathscr{B}_j$, we consider the point clouds formed over two windows of frames of user-defined length k, one bordered at the beginning by $\mathscr{A}_i$ and the other bordered at the end by $\mathscr{B}_j$. That is, each point cloud is the composition of smaller point clouds representing the pose at each frame in the window. The use of windows of frames effectively incorporates derivative information into the metric, and is similar to the approach in [Schödl et al. 2000]. The size of the windows are the same as the length of the transitions, so $D(\mathscr{A}_i, \mathscr{B}_j)$ is affected by every pair of frames that form the transition. We use a value of k corresponding to a window of about a third of a second in length, as in [Mizuguchi et al. 2001]

The distance between $\mathscr{A}_i$ and $\mathscr{B}_j$ may be calculated by computing a weighted sum of squared distances between corresponding points $\mathbf{p_i}$ and $\mathbf{p_i'}$ in the two point clouds. To address the problem of finding coordinate systems for these point clouds (item 2 in the above list), we calculate the *minimal* weighted sum of squared distances given that an arbitrary rigid 2D transformation may be applied to the second point cloud:

$$\min_{\theta, x_0, z_0} \sum_i w_i \|\mathbf{p_i} - \mathbf{T}_{\theta, \mathbf{x_0}, \mathbf{z_0}} \mathbf{p_i'}\|^2 \tag{1}$$

where the linear transformation $\mathbf{T}_{\theta, \mathbf{x_0}, \mathbf{z_0}}$ rotates a point $\mathbf{p}$ about the y (vertical) axis by θ degrees and then translates it by (x_0, z_0). The index is over the number of points in each point cloud. The weights w_i may be chosen both to assign more importance to certain joints (e.g., those with constraints) and to taper off towards the end of the window.

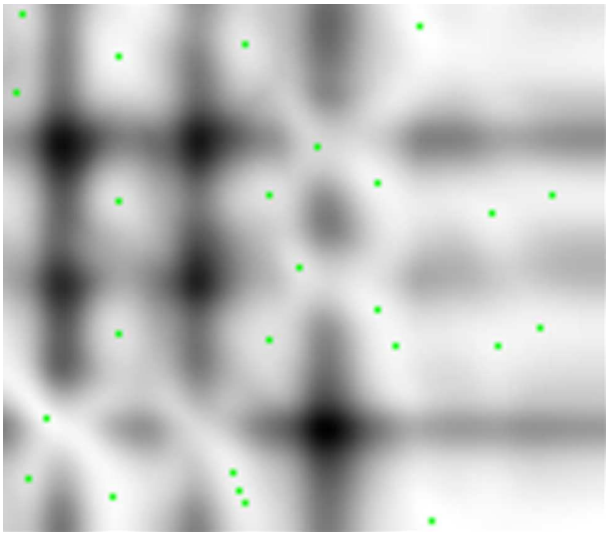


Figure 3: An example error function for two motions. The entry at (i, j) contains the error for making a transition from the i^{th} frame of the first motion to the j^{th} frame of the second. White values correspond to lower errors and black values to higher errors. The colored dots represent local minima.

This optimization has a closed-form solution:

$$\theta = \arctan \frac{\sum_i w_i (x_i z_i' - x_i' z_i) - \frac{1}{\sum_i w_i} (\overline{x}\overline{z'} - \overline{x'}\overline{z})}{\sum_i w_i (x_i x_i' + z_i z_i') - \frac{1}{\sum_i w_i} (\overline{x}\overline{x'} + \overline{z}\overline{z'})} \tag{2}$$

$$x_0 = \frac{1}{\sum_i w_i} (\overline{x} - \overline{x'}\cos(\theta) - \overline{z'}\sin\theta) \tag{3}$$

$$z_0 = \frac{1}{\sum_i w_i} (\overline{z} + \overline{x'}\sin(\theta) - \overline{z'}\cos\theta) \tag{4}$$

where $\overline{x} = \sum_i w_i x_i$ and the other barred terms are defined similarly.

We compute the distance as defined above for every pair of frames in the database, forming a sampled 2D error function. Figure 3 shows a typical result. To make our transition model more compact, we find all the local minima of this error function, thereby extracting the "sweet spots" at which transitions are locally the most opportune. This tactic was also used in [Schödl et al. 2000]. These local minima are our candidate transition points.

3.2 Selecting Transition Points

A local minimum in the distance function does not necessarily imply a high-quality transition; it only implies a transition better than its neighbors. We are specifically interested in local minima with small error values. The simplest approach is to only accept local minima below an empirically determined threshold. This can be done without user intervention. However, often users will want to

set the threshold themselves to pick an acceptable tradeoff between having good transitions (low threshold) and having high connectivity (high threshold).

Different kinds of motions have different fidelity requirements. For example, walking motions have very exacting requirements on the transitions — people have seen others walk nearly every day since birth and consequently have a keen sense of what a walk should look like. On the other hand, most people are less familiar with ballet motions and would be less likely to detect inaccuracies in such motion. As a result, we allow a user to apply different thresholds to different pairs of motions; transitions among ballet motions may have a higher acceptance threshold than transitions among walking motions.

3.3 Creating Transitions

If $D(\mathscr{A}_i, \mathscr{B}_j)$ meets the threshold requirements, we create a transition by blending frames $\mathscr{A}_i$ to $\mathscr{A}_{i+k-1}$ with frames $\mathscr{B}_{j-k+1}$ to $\mathscr{B}_j$, inclusive. The first step is to apply the appropriate aligning 2D transformation to motion $\mathscr{B}$. Then on frame p of the transition $(0 \le p < k)$ we linearly interpolate the root positions and perform spherical linear interpolation on joint rotations:

$$R_p = \alpha(p) R_{\mathscr{A}_{i+p}} + [1-\alpha(p)] R_{\mathscr{B}_{j-k+1+p}} \tag{5}$$

$$q^i_p = slerp(q^i_{\mathscr{A}_{i+p}}, q^i_{\mathscr{B}_{j-k+1+p}}, \alpha(p)) \tag{6}$$

where R_p is the root position on the p^{th} transition frame and q^i_p is the rotation of the i^{th} joint on the p^{th} transition frame.

To maintain continuity we choose the blend weights $\alpha(p)$ according to the conditions that $\alpha(p) = 1$ for $p \le -1$, $\alpha(p) = 0$ for $p \ge k$, and that $\alpha(p)$ has C^1 continuity everywhere. This requires

$$\alpha(p) = 2(\frac{p+1}{k})^3 - 3(\frac{p+1}{k})^2 + 1, \; -1 < p < k \tag{7}$$

Other transition schemes, such as [Rose et al. 1996], may be used in place of this one.

The use of linear blends means that constraints in the original motion may be violated. For example, one of the character's feet may slide when it ought to be planted. This can be corrected by using constraint annotations in the original motions. We treat constraints as binary signals: on a given frame a particular constraint either exists or it does not. Blending these signals in analogy to equations 5 and 6 amounts to using the constraints from $\mathscr{A}$ in the first half of the transition and the constraints from $\mathscr{B}$ in the second half. In this manner each transition is automatically annotated with constraint information, and these constraints may later be enforced as a post-processing step when motion is extracted form the graph. We will discuss constraint enforcement in more detail in the next section.

Descriptive labels attached to the motions are carried along into transitions. Specifically, if a transition frame is a blend between a frame with a set of labels L_1 and another frame with a set of labels L_2, then it has the union of these labels $L_1 \cup L_2$.

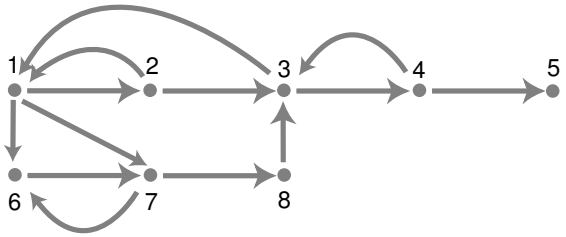


Figure 4: A simple motion graph. The largest strongly connected component is $[1,2,3,6,7,8]$. Node 4 is a sink and 5 is a dead end.

3.4 Pruning The Graph

In its current state there are no guarantees that the graph can synthesize motion indefinitely, since there may be nodes (called *dead ends*) that are not part of any cycle (see Figure 4). Once such a node is entered there is a bound on how much additional motion can be generated. Other nodes (called *sinks*) may be part of one or more cycles but nonetheless only be able to reach a small fraction of the total number of nodes in the graph. While arbitrarily long motion may still be generated once a sink is entered, this motion is confined to a small part of the database. Finally, some nodes may have incoming edges such that no outgoing edges contain the same set of descriptive labels. This is dangerous since logical discontinuities may be forced into a motion. For example, a character currently in a "boxing" motion may have no choice but to transition to a "ballet" motion.

To address these problems, we prune the graph such that, starting from any edge, it is possible to generate arbitrarily long streams of motion of the same type such that as much of the database as possible is used. This is done as follows. Every frame of original data is associated with a (possibly empty) set of labels. Say there are n unique sets. For each set, form the subgraph consisting of all edges whose frames have exactly this set of labels. Compute the strongly connected components (SCCs) of this subgraph, where an SCC is a maximal set of nodes such that there is a connecting graph walk for any ordered pair of nodes (u,v). The SCCs can be computed in $O(V+E)$ time using an algorithm due to Tarjan. We eliminate from this subgraph (and hence the original motion graph) any edge that does not attach two nodes in the largest SCC. Once this process is completed for all n label sets, any nodes with no edges are discarded.

A warning is given to the user if the largest SCC for a given set of labels contains below a threshold number of frames. Also, a warning is given if for any ordered pair of SCCs there is no way to transition from the first to the second. In either case, the user may wish to adjust the transition thresholds (Section 3.2) to give the graph greater connectivity.

4 Extracting Motion

By this stage we have finished constructing the motion graph. After describing exactly how a graph walk can be converted into displayable motion, we will consider the general problem of extracting motion that satisfies user constraints. Our algorithm involves solving an optimization problem, and so we conclude this section with some general recommendations on how to pose the optimization.

4.1 Converting Graph Walks To Motion

Since every edge on the motion graph is a piece of motion, a graph walk corresponds to a motion generated by placing these pieces one after another. The only issue is to place each piece in the correct location and orientation. In other words, each frame must be transformed by an appropriate 2D rigid transformation. At the start of a graph walk this transformation is the identity. Whenever we exit a transition edge, the current transformation is multiplied by the transformation that aligned the pieces of motion connected by the transition (Section 3.1).

As noted in Section 3.3, the use of linear blends to create transitions can cause artifacts, the most common of which is feet that slide when they ought to be planted. However, every graph walk is automatically annotated with constraint information (such as that the foot must be planted). These constraints are either specified directly in the original motions or generated as in Section 3.3, depending on whether the frame is original data or a transition. These constraints may be satisfied using a variety of methods, such as [Gleicher 1998] or [Lee and Shin 1999]. In our work we used the method described in [Kovar et al. 2002].

4.2 Searching For Motion

We are now in a position to consider the problem of finding motion that satisfies user-specified requirements. It is worth first noting that only very special graph walks are likely to be useful. For example, while a random graph walk will generate a continuous stream of motion, such an algorithm has little use other than an elaborate screen saver. As a more detailed example, consider computing an all-pairs shortest graph walk table for the graph. That is, given a suitable metric — say, time elapsed or distance travelled — we can use standard graph algorithms like Floyd-Warshall to find for each pair of nodes u and v the connecting graph walk that minimizes the metric. With this in hand we could, for example, generate the motion that connects one clip to another as quickly as possible. This is less useful than it might appear at first. First, there are no guarantees that the shortest graph walk is short in an absolute sense. In our larger test graphs (between a few and several thousand nodes) the average shortest path between any two nodes was on the order of two seconds. This is not because the graphs were poorly connected. Since the transitions were about one-third of a second apiece, this means there were on average only five or six transitions separating any two of the thousands of nodes. Second, there is no control over what happens during the graph walk — we can't specify what direction the character travels in or where she ends up.

More generally, the sorts of motions that a user is likely to be interested in probably don't involve minimizing metrics as simple as total elapsed time. However, for complicated metrics there is typically no simple way of finding the globally optimal graph walk. Hence we focus instead on local search methods that try to find a *satisfactory* graph walk within a reasonable amount of time.

We now present our framework for extracting graph walks that conform to a user's specifications. We cast motion extraction as a search problem and use branch and bound to increase the efficiency of this search. The user supplies a scalar function $g(w,e)$ that evaluates the additional error accrued by appending an edge e to the existing path w, which may be the empty path $\emptyset$. The total error $f(w)$ of the path is defined as follows:

$$f(w) = f([e_1,\ldots,e_n]) = \sum_{i=1}^{n} g([e_1,\ldots,e_{i-1}],e_i) \qquad (8)$$

where w is comprised of the edges $e_1,\ldots,e_n$. We require $g(w,e)$ to be nonnegative, which means that we can never decrease the total error by adding more edges to a graph walk.

In addition to f and g, the user must also supply a halting condition indicating when no additional edges should be added to a graph walk. A graph walk satisfying the halting condition is called *complete*. The start of the graph walk may either be specified by the user or chosen at random.

Our goal is find a complete graph walk w that minimizes f. To give the user control over what sorts of motions should be considered in the search, we allow restrictions on what edges may be appended to a given walk w. For example, the user may decide that within a particular window of time a graph walk may only contain "sneaking" edges.

A naïve solution is to use depth-first search to evaluate f for all complete graph walks and then select the best one. However, the number of possible graph walks grows exponentially with the average size of a complete graph walk. To address this we use a branch and bound strategy to cull branches of the search that are incapable of yielding a minimum. Since $g(w,e)$ by assumption never decreases, $f(w)$ is a lower bound on $f(w+v)$ for any v, where $w+v$ is the graph walk composed of v appended to w. Thus we can keep track of the current best complete graph walk w_{opt} and immediately halt any branch of the search for which the graph walk's error exceeds $f(w_{opt})$. Also, the user may define a threshold error ε such that if $f(w) < \varepsilon$, then w is considered to be "good enough" and the search is halted.

Branch and bound is most successful when we can attain a tight lower bound early in the search process. For this reason it is worthwhile to have a heuristic for ordering the edges we explore out of a particular node. One simple heuristic is to order the children greedily — that is, given a set of unexplored children $c_1,\ldots,c_n$, we search the one that minimizes $g(w,c_i)$.

While branch and bound reduces the number of graph walks we have to test against f, it does not change the fact that the search process is inherently exponential — it merely lowers the effective branching factor. For this reason we generate a graph walk incrementally. At each step we use branch and bound to find an optimal graph walk of n frames. We retain the first m frames of this graph walk and use the final retained node as a starting point for another search. This process continues until a complete graph walk is generated. In our implementation we used values of n from 80 to 120 frames ($2\frac{2}{3}$ to 4 seconds) and m from 25 to 30 frames (about one second).

Sometimes it is useful to have a degree of randomness in the search process, such as when one is animating a crowd. There are a couple of easy ways to add randomness to the search process without sacrificing a good result. The first is to select a start for the search at random. The second is retain the r best graph walks at the end of each iteration of the search and randomly pick among the ones whose error is within some tolerance of the best solution.

4.3 Deciding What To Ask For

Since the motion extracted from the graph is determined by the function g, it is worth considering what sorts of functions are likely to produce desirable results. To understand the issues involved, we consider a simple example. Imagine we want to lay down two clips on the floor and create a motion that starts at the first clip and ends at the second. Both clips must end up in the specified position and orientation. We can formally state this problem as follows: given a starting node N in the graph and a target edge e, find a graph walk

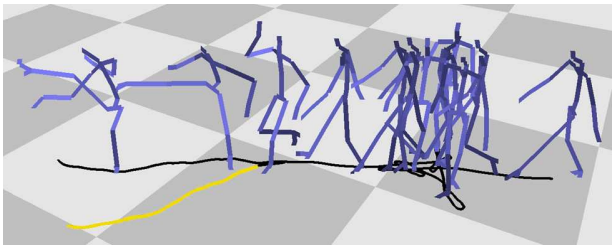


Figure 5: The above motion was generated using the search algorithm discussed in this section. The halting condition was to play a specific clip of two kicking motions. The error of a complete graph walk (which necessarily ended with the kicking clip) was determined by how far away this kicking clip was from being in a particular position and orientation. The character spends approximately seven seconds making minute adjustments to its orientation in an attempt to better align itself with the final clip. The highlighted line shows the the path of the target clip in its desired position and orientation.

that ends with e such that the transformation $\mathbf{T}$ applied to e is as close as possible to a given transformation $\mathbf{T}'$. What one will receive is a motion like in Figure 5, where the initial clip is a walking motion and the final clip is a kick. The character turns around in place several times in an attempt to better line up with the target clip.

While it's conceivable that given a larger database we would have found a better motion, the problem here is with the function we passed into the search algorithm. First, it gives no guidance as to what should be done in the middle of the motion; all that matters is that the final clip be in the right position and orientation. This means the character is allowed to do whatever is possible in order to make the final fit, even if the motion is nothing that a real person would do. Second, the goal is probably more specific than necessary. If it doesn't matter what kick the character does, then it should be allowed to choose a kick that doesn't require such effort to aim.

More generally, there are two lessons we can draw from this example. First, g should give some sort of guidance throughout the entire motion, as arbitrary motion is almost never desirable. Second, g should be no more restrictive than necessary, in order to give the search algorithm more goals to seek. Note the tradeoff here — guiding the search toward a particular result must be balanced against unduly preventing it from considering all available options.

5 Path Synthesis

We have cast motion extraction as an optimization problem, and we have given some reasons why the formulation of this optimization can be difficult. To demonstrate that it is nonetheless possible to come up with optimization criteria that allow us to solve a real problem, we apply the preceding framework to path synthesis. This problem is simple to state: given a path P specified by the user, generate motion such that the character travels along P. In this section we present our algorithm for path synthesis, present results, and discuss applications of the technique.

5.1 Implementing Path Synthesis

Given the framework in the previous section, our only tasks are to define an error function $g(w,e)$ and appropriate halting criteria. The basic idea is to estimate the actual path P' travelled by the character during a graph walk and measure how different it is from P. The graph walk is complete when P' is sufficiently long.

A simple way to determine P' is to project the root onto the floor at each frame, forming a piecewise linear curve[1]. Let $P(s)$ be the point on P whose arc-length distance from the start of P is s. The i^{th} frame of the graph walk, w_i, is at some arc length $s(w_i)$ from the start of P'. We define the corresponding point on P as the point at the same arc length, $P(s(w_i))$. For the j^{th} frame of e, we calculate the squared distance between $P'(s(e_j))$ and $P(s(e_j))$. $g(w,e)$ is the sum of these errors:

$$g(w,e) = \sum_{i=1}^{n} \|P'(s(e_i)) - P(s(e_i))\|^2 \qquad (9)$$

Note that $s(e_i)$ depends on the total arc length of w, which is why this equation is a function of w as well as e. The halting condition for path synthesis is when the current total length of P' meets or exceeds that of P. Any frames on the graph walk at an arc length longer than the total length of P are mapped to the last point on P.

The error function $g(w,e)$ was chosen for a number of reasons. First, it is efficient to compute, which is important in making the search algorithm practical. Second, the character is given incentive to make definite progress along the path. If we were to have required the character to merely be near the path, then it would have no reason not to alternate between travelling forwards and backwards. Finally, this metric allows the character to travel at whatever speed is appropriate for what needs to be done. For example, a sharp turn will not cover distance at the same rate as walking straight forward. Since both actions are equally important for accurate path synthesis, it is important that one not be given undue preference over the other.

One potential problem with this metric is that a character who stands still will never have an incentive to move forward, as it can accrue zero error by remaining in place. While we have not encountered this particular problem in practice, it can be countered by requiring at least a small amount of forward progress γ on each frame. More exactly, we can replace in Equation 9 the function $s(e_i)$ with $t(e_i) = max(t(e_{i-1}) + s(e_i) - s(e_{i-1}), t(e_{i-1}) + \gamma)$.

Typically the user will want all generated motion to be of a single type, such as walking. This corresponds to confining the search to the subgraph containing the appropriate set of descriptive labels. More interestingly, one can require different types of motion on different parts of the path. For example, one might want the character to walk along the first half of the path and sneak down the rest. The necessary modifications to accomplish this are simple. We will consider the case of two different motion types; the generalization to higher numbers is trivial.

We divide the original path into two smaller adjoining paths, P_1 and P_2, based on where the transition from type T_1 to type T_2 is to occur. If the character is currently fitting P_2, then the algorithm is identical to the single-type case. If the character is fitting P_1, then we check to see if we are a threshold distance from the end of P_1. If not, we continue to only consider edges of type T_1. Otherwise we allow the search to try both edges of type T_1 and T_2; in the latter case we switch to fitting P_2. Note that we only allow this switch to occur once on any given graph walk, which prevents the resulting motion from randomly switching between the two actions.

5.2 Results

While the examples shown in Figure 1 suggest that our technique is viable, it perhaps isn't surprising that we were able to find accurate fits to the given paths. As shown in the upper portion of the

[1] In our implementation we defined the path as a spline approximating this piecewise linear path, although this has little impact on the results.

figure, the input motion had a fair amount of variation, including straight-ahead marches, sharp turns, and smooth changes of curvature. However, our algorithm is still useful when the input database is not as rich. Refer to Figure 6. We started with a single 12.8-second clip of an actor sneaking along the indicated path. To stretch this data further, we created a mirror-image motion and then built a motion graph out of the two. From these we were able to construct the new motions shown at the bottom of the figure, both of which are themselves approximately 13 seconds in length.

Figure 7 shows fits to a more complicated path. The first example uses walking motions and the second uses martial arts motions; the latter demonstrates that our approach works even on motions that are not obviously locomotion. For the walking motion, the total computation time was nearly the same as the length of the generated animation (58.1 seconds of calculation for 54.9 seconds animation). The martial arts motion is 87.7 seconds long and required just 15.0 seconds of computation. In general, in our test cases the duration of a generated motion was either greater than or approximately equal to the amount of time needed to produce it. Both motion graphs had approximately 3000 frames (100 seconds) of animation.

Finally, Figure 8 shows paths containing constraints on the allowable motion type. In the first section of each path the character is required to walk, in the second it must sneak, and in the third it is to perform martial arts moves. Not only does the character follow the path well, but transitions between action types occur quite close to their specified locations. This example used a database of approximately 6000 frames (200 seconds).

All examples were computed on a 1.3GHz Athlon. For our largest graph (about 6000 frames), approximately twenty-five minutes were needed to compute the locations of all candidate transitions points. Approximately five minutes of user time were required to select transition thresholds, and it took less than a minute to calculate blends at these transitions and prune the resulting graph.

5.3 Applications Of Path Synthesis

Directable locomotion is a general enough need that the preceding algorithm has many applications.

Interactive Control. We can use path synthesis techniques to give a user interactive control over a character. For example, when the user hits the left arrow key the character might start travelling east. To accomplish this, we can use the path fitting algorithm to find the sequence of edges starting from our current location on the graph that best allow the character to travel east. The first edge on the resulting graph walk is the next clip that will be played. This process may then be repeated. To make this practical, we can precompute for every node in the graph a sequence of graph walks that fit straight-line paths in a sampling of directions (0 degrees, 30 degrees, ...). The first edges on these paths are then stored for later use; they are the best edges to follow given the direction the character is supposed to travel in.

High-Level Keyframing. If we want a character to perform certain actions in a specific sequence and in specific locations, we can draw a path with subsections requiring the appropriate action types. This allows us to generate complex animations without the tedium of manual keyframing. For this reason we term this process "high-level" keyframing — the user generates an animation based on what should be happening and where.

Motion Dumping. If an AI algorithm is used to determine that a character must travel along a certain path or start performing certain actions, the motion graph may be used to "dump" motion on top of the algorithm's result. Hence motion graphs may be used as a back-end for animating non-player characters in video games and interactive environments — the paths and action types can be specified by a high-level process and the motion graph would fill in the details.

Crowds. While our discussion so far has focused on a single character, there's no reason why it couldn't be applied to several characters in parallel. Motion graphs may be used as a practical tool for crowd generation. For example, a standard collision-avoidance algorithm could be used to generate a path for each individual, and the motion graph could then generate motion that conforms to this path. Moreover, we can use the techniques described at the end of Section 4.2 to add randomness to the generated motion.

6 Discussion

In this paper we have presented a framework for generating realistic, controllable motion through a database of motion capture. Our approach involves automatically constructing a graph that encapsulates connections among different pieces of motion in the database and then searching this graph for motions that satisfy user constraints. We have applied our framework to the problem of path synthesis.

As we had limited access to data, our largest examples used a database of several thousand frames of motion. While we believe this was sufficient to show the potential of our method, a character with a truly diverse set of actions might require hundreds or thousands of times more data. Hence the scalability of our framework bears discussion. The principle computational bottleneck in graph construction is locating candidate transitions (Section 3.1). This requires comparing every pair of the F frames in the database and therefore involves $O(F^2)$ operations. However, this calculation is trivial to parallelize, and distances between old frames needn't be recomputed if additions are made to the database.

It is the exception rather than the rule that two pieces of motion are sufficiently similar that a transition is possible, and hence motion graphs tend to be sparse. In our experience the necessary amount of storage is approximately proportional to the size of the database.

The number of edges leaving a node in general grows with the size of the graph, meaning the branching factor in our search algorithm may grow as well. However, we expect that future motion graphs will be larger mainly because the character will be able to perform more actions. That is, for example, having increasing amounts of walking motion isn't particularly useful once one can direct a character along nearly any path. Hence the branching factor in a particular subgraph will remain stationary once that subgraph is sufficiently large. We anticipate that typical graph searches will be restricted to one or two subgraphs, and so we expect that the search will remain practical even for larger graphs.

We conclude with a brief discussion of future work. One limitation of our approach is that the transition thresholds must be specified by hand, since (as discussed in Section 3.2) different kinds of motions have different fidelity requirements. Setting thresholds in databases involving many different kinds of motions may be overly laborious, and so we are investigating methods for automating this process. A second area of future work is to incorporate parameterizable motions [Wiley and Hahn 1997; Rose et al. 1998] into our system, rather than having every node correspond to a static piece of motion. This would add flexibility to the search process and potentially allow generated motion to better satisfy user constraints. Finally, we are interested in applying motion graphs to problems other than path synthesis.

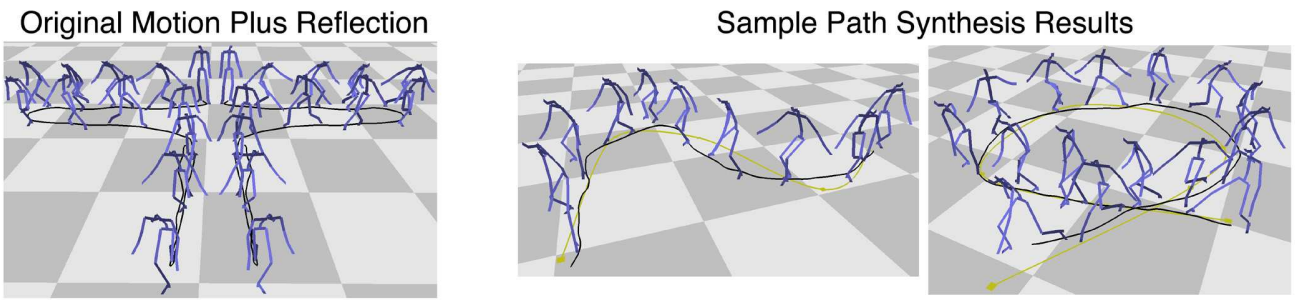


Figure 6: The leftmost image shows the original motion and its reflection and the following images show motion generated by our path synthesis algorithm. The thick yellow lines were the paths to be fit and the black line is an approximation of the actual path of the character. Note how we are able to accurately fit nontrivial paths despite the limited variation in the path of the original motion.

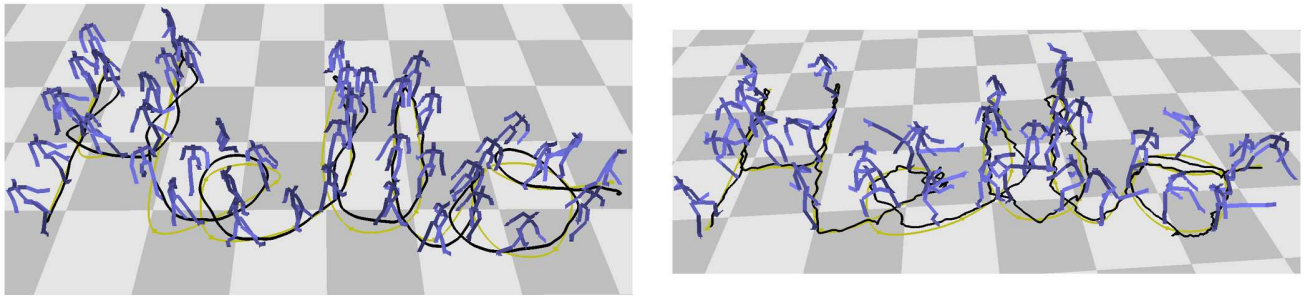


Figure 7: The left image shows a walking motion generated to fit to a path that spells "Hello" in cursive. The right image shows a karate motion fit to the same path. The total calculation time for the walking motion was 58.1 seconds and the animation itself is 54.9 seconds. The 87.7-second karate motion was computed in just 15.0 seconds. All computation was done on a 1.3gHz Athlon.

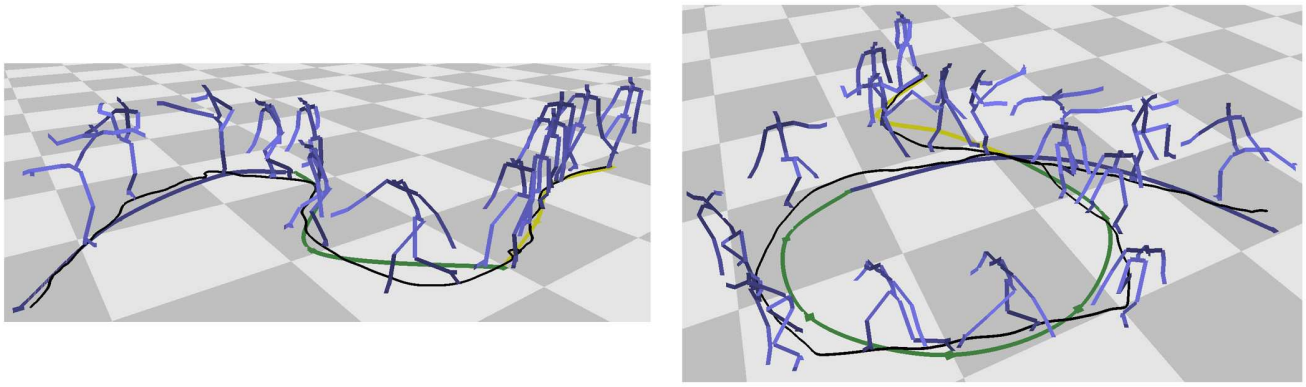


Figure 8: These images are both fits to paths wherein the character is required to walk, then sneak, and finally perform martial arts moves. The desired transition points are indicated by where the curve changes color. Note that the character both fits the path accurately and switches to the appropriate motion type close to the desired location.

Acknowledgements

We would like to acknowledge Andrew Gardner, Alex Mohr, and John Schreiner for assisting in video production, proofreading, and other technical matters. We also thank the University of Southern California's School of Film and Television for their support and the reviewers for their many useful suggestions. Our work was made possible through generous motion data donations from Spectrum Studios (particularly Demian Gordon), House of Moves, and The Ohio State University. This work was supported in part by NSF grants CCR-9984506 and IIS-0097456, the U.S. Army[2], the Wisconsin Alumni Research Fund's University Industrial Relations program, equipment donations from IBM, NVidia, and Intel, and software donations from Discreet, Alias/Wavefront, and Pixar.

References

ARIKAN, O., AND FORSYTHE, D. 2002. Interactive motion generation from examples. In *Proceedings of ACM SIGGRAPH 2002*, Annual Conference Series, ACM SIGGRAPH.

BOWDEN, R. 2000. Learning statistical models of human motion. In *IEEE Workshop on Human Modelling, Analysis, and Synthesis, CVPR 2000*, IEEE Computer Society.

BRAND, M., AND HERTZMANN, A. 2000. Style machines. In *Proceedings of ACM SIGGRAPH 2000*, Annual Conference Series, ACM SIGGRAPH, 183–192.

BRUDERLIN, A., AND CALVERT, T. 1996. Knowledge-driven, interactive animation of human running. In *Graphics Interface*, Canadian Human-Computer Communications Society, 213–221.

BRUDERLIN, A., AND WILLIAMS, L. 1995. Motion signal processing. In *Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, ACM SIGGRAPH, 97–104.

FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 2001. Composable controllers for physics-based character animation. In *Proceedings of ACM SIGGRAPH 2001*, Annual Conference Series, ACM SIGGRAPH, 251–260.

GALATA, A., JOGNSON, N., AND HOGG, D. 2001. Learning variable-length markov models of behavior. *Computer Vision and Image Understanding Journal 81*, 3, 398–413.

GLEICHER, M. 1998. Retargeting motion to new characters. In *Proceedings 0f ACM SIGGRAPH 98*, Annual Conference Series, ACM SIGGRAPH, 33–42.

GLEICHER, M. 2001. Motion path editing. In *Proceedings 2001 ACM Symposium on Interactive 3D Graphics*, ACM.

HODGINS, J. K., WOOTEN, W. L., BROGAN, D. C., AND O'BRIEN, J. F. 1995. Animating human athletics. In *Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, ACM SIGGRAPH, 71–78.

KOVAR, L., GLEICHER, M., AND SCHREINER, J. 2002. Footskate cleanup for motion capture editing. Tech. rep., University of Wisconsin, Madison.

LAMOURET, A., AND PANNE, M. 1996. Motion synthesis by example. *Computer animation and Simulation*, 199–212.

LEE, J., AND SHIN, S. Y. 1999. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of ACM SIGGRAPH 99*, Annual Conference Series, ACM SIGGRAPH, 39–48.

LEE, J., CHAI, J., REITSMA, P. S. A., HODGINS, J. K., AND POLLARD, N. S. 2002. Interactive control of avatars animated with human motion data. In *Proceedings of ACM SIGGRAPH 2002*, Annual Conference Series, ACM SIGGRAPH.

LEE, J. 2000. *A hierarchical approach to motion analysis and synthesis for articulated figures*. PhD thesis, Department of Computer Science, Korea Advanced Institute of Science and Technology.

LI, Y., WANG, T., AND SHUM, H.-Y. 2002. Motion texture: A two-level statistical model for character motion synthesis. In *Proceedings of ACM SIGGRAPH 2002*, Annual Conference Series, ACM SIGGRAPH.

[2]This paper does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred

MIZUGUCHI, M., BUCHANAN, J., AND CALVERT, T. 2001. Data driven motion transitions for interactive games. In *Eurographics 2001 Short Presentations*.

MOLINA-TANCO, L., AND HILTON, A. 2000. Realistic synthesis of novel human movements from a database of motion capture examples. In *Proceedings of the Workshop on Human Motion*, IEEE Computer Society, 137–142.

MULTON, F., FRANCE, L., CANI, M.-P., AND DEBUNNE, G. 1999. Computer animation of human walking: a survey. *The Journal of Visualization and Computer Animation 10*, 39–54. Published under the name Marie-Paule Cani-Gascuel.

PERLIN, K., AND GOLDBERG, A. 1996. Improv: A system for scripting interactive actors in virtual worlds. In *Proceedings of ACM SIGGRAPH 96*, ACM SIGGRAPH, 205–216.

PERLIN, K. 1995. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics 1*, 1 (Mar.), 5–15.

PULLEN, K., AND BREGLER, C. 2000. Animating by multi-level sampling. In *IEEE Computer Animation Conference*, CGS and IEEE, 36–42.

PULLEN, K., AND BREGLER, C. 2002. Motion capture assisted animation: Texturing and synthesis. In *Proceedings of ACM SIGGRAPH 2002*, Annual Conference Series, ACM SIGGRAPH.

ROSE, C., GUENTER, B., BODENHEIMER, B., AND COHEN, M. F. 1996. Efficient generation of motion transitions using spacetime constraints. In *Proceedings of ACM SIGGRAPH 1996*, Annual Conference Series, ACM SIGGRAPH, 147–154.

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

SCHÖDL, A., SZELISKI, R., SALESIN, D., AND ESSA, I. 2000. Video textures. In *Proceedings of ACM SIGGRAPH 2000*, Annual Conference Series, ACM SIGGRAPH, 489–498.

SUN, H. C., AND METAXAS, D. N. 2001. Automating gait animation. In *Proceedings of ACM SIGGRAPH 2001*, Annual Conference Series, ACM SIGGRAPH, 261–270.

WASHBURN, D. 2001. The quest for pure motion capture. *Game Developer* (December).

WILEY, D., AND HAHN, J. 1997. Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Application 17*, 6, 39–45.

WITKIN, A., AND POPOVIĆ, Z. 1995. Motion warping. In *Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, ACM SIGGRAPH, 105–108.

8.2 Snap-Together Motion: Assembling Run-Time Animations

To Appear in the 2003 Symposium on Interactive 3D Graphics

Snap-Together Motion: Assembling Run-Time Animations

Michael Gleicher * Hyun Joon Shin Lucas Kovar Andrew Jepsen

Computer Sciences Department, University of Wisconsin, Madison

Abstract

Many virtual environments and games must be populated with synthetic characters to create the desired experience. These characters must move with sufficient realism, so as not to destroy the visual quality of the experience, yet be responsive, controllable, and efficient to simulate. In this paper we present an approach to character motion called *Snap-Together Motion* that addresses the unique demands of virtual environments. Snap-Together Motion (STM) preprocesses a corpus of motion capture examples into a set of short clips that can be concatenated to make continuous streams of motion. The result process is a simple graph structure that facilitates efficient planning of character motions. A user-guided process selects "common" character poses and the system automatically synthesizes multi-way transitions that connect through these poses. In this manner well-connected graphs can be constructed to suit a particular application, allowing for practical interactive control without the effort of manually specifying all transitions.

Keywords: Motion Synthesis, Virtual Environments, Motion Capture

1 Introduction

Advances in graphics hardware and rendering software have made it possible to build visually rich virtual worlds, creating possibilities for entertainment and training applications. For many of these applications, the virtual worlds must be populated with believable synthetic characters. Creating such characters is challenging. To fit with the visual richness provided by virtual environments, characters must move in realistic ways. At the same time, in order to meet interactivity demands they must also be efficiently animated and controllable by the simulation.

In this paper we introduce a methodology that allows quality motions to be synthesized in a controllable manner with little run-time overhead. A corpus of motion capture data is processed into a set of short clips that can be "snapped together" (concatenated) into seamless streams of motions at run time. This process is guided by a human author who identifies (either independently or via help from our system) character poses that appear frequently in the corpus. Each such pose serves as a jump point at which any motion that enters can be followed by any motion that leaves, as shown in Figures 1 and 2. The result is a simple graph structure that allows clips to be connected into longer motions.

*gleicher@cs.wisc.edu, http://www.cs.wisc.edu/graphics

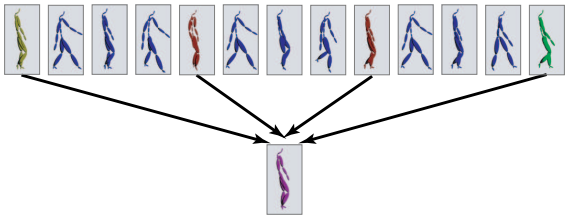


Figure 1: Schematic of the authoring process. A linear corpus of motion (here a single walking motion) has a common pose identified through a user-guided process.

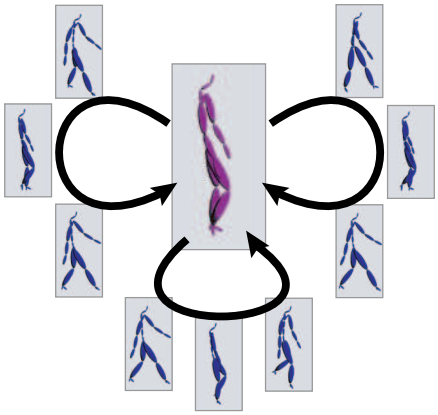


Figure 2: Transitions are generated around the common pose, forming a simple graph.

All transition generation and cleanup operations are performed automatically by our system. At run time, a character animation module need only play precomputed clips in a valid order as determined by the graph. User involvement in the graph construction process allows for the clips to connect in ways that facilitate control. That is, the animation designer guides the system into building a graph with a structure that is contrived to be easy to exploit at run time. In particular, if the designer creates a graph with a high branching factor, the run-time motion planner will have the flexibility to choose from several options when a new action must be taken.

Our approach is closely related to previous non-linear animation methods. In particular, our final graph structures are akin to the *move trees* common in computer games. The key difference is that our graphs are constructed opportunistically based on a provided data corpus and some user guidance on how to form a usable graph. In contrast, traditional move trees use specially contrived motions and hand-crafted transitions. In a sense, we provide a new approach for constructing the data structures used by existing approaches to real-time animation synthesis. The increased automation of our approach reduces the planning, effort, and skill required to author the graph structures, and it is possible to author graphs with a degree

of connectivity that would be extremely tedious to construct using traditional methods.

Our work involves two main contributions, each of which facilitate the authoring of character motion for virtual environments. First, we provide an improved authoring methodology where candidate transition points are identified automatically. This aids in the creation of graphs with a small number of *hub* nodes containing a large number of edges. We speed the process of adding clips to the graph by allowing an author to add entire hubs to a graph at a time, and we can further simplify construction by automatically suggesting hubs based on the original motion data.

The second main contribution of our work is to provide methods for generating multi-way transitions. Our framework allows *cut transitions*, which involve simply concatenating two clips together without further processing. This is done by adjusting the original motions such that these transitions are seamless, i.e., they are C_1 smooth and satisfy the appropriate constraints. The advantages of such an approach are that it keeps the resulting graph compact and allows efficient generation of motions at run time. The challenge is to connect multiple motions in a manner that avoids visual artifacts.

The remainder of this paper is divided into five sections. First, we clarify in Section 2 the limitations of current tools for constructing graphs and how we propose to address these limitations. In Section 3 we discuss related work. In Section 4 we describe our algorithms and explain how they fit into the overall process of building a graph. Finally, in Section 5 we present some example results and then we conclude in Section 6 with a discussion of the advantages and drawbacks of our approach.

2 Issues with Current Practices

In order to create streams of high-quality motion, current applications assemble static clips of motion created with traditional animation techniques such as motion capture or keyframing. The assembly process requires making transitions between motions. These transitions may be difficult to create, such as a transition between a running clip and one where the character is lying down, or trivial, if the end of one clip is identical to the beginning of the next. In practice, simple techniques such as linear blends are capable of creating transitions in cases where the motions are similar.

A set of motion clips and transitions between them form a graph where the edges are pieces of motions and the nodes are choice points connecting motions. A graph of this type called a *move tree* is common in computer games [14, 15]. Move trees are constructed by pre-planning movements such that the initial clips have similar beginning and end points. An artist then chooses the exact points in the clips where transitions are to occur and creates the transition motions. Most commercial motion editing tools, such as Character Studio, Softimage XSI, Diva, or Messiah:Animation, provide some support for applying simple transition methods (e.g., linear blends) at identified points.

The structure of a graph can have a significant impact on its usefulness. In general, the more well-connected a graph is, the more controllable the animation will be. Ideally, all clips of motions will connect, allowing any action to take place at any time. In practice, good transitions between radically different clips are prohibitively difficult to create. Tradeoffs must therefore be made between the quality of the transitions and the connectivity of the graphs.

While it may not be possible to have all clips connect directly, well-constructed graphs nonetheless typically have nodes with many incoming and outgoing edges. We call such nodes *hubs*. Hubs are desirable because they offer advantages in both flexibility and simplifying the problem of generating motion that meets high-level requirements. For example, a particular hub might contain several different kinds of punches and kicks, in which case a character could easily string together a sequence of strikes according to a high-level reasoning module (e.g., he should throw a punch combination since the opponent's guard is down). Similarly, there might be a "walking" hub that has several outgoing edges which each correspond to taking a step in some direction. Combined with jogging and running hubs, a planning module could move characters in the virtual environment simply by specifying a speed and direction.

Graphs containing hubs are difficult to construct. Authors must find places in the motion corpus where several motions come together and devise multi-way transitions, a much more difficult problem than making just two clips join smoothly. Current tools offer little support for the creation of hubs. Our framework, in contrast, explicitly supports the creation of hub nodes. Instead of having to hand-select a set of clip boundaries that are conducive to quality transitions, we are able to automatically provide the user with sets of clips whose starting and ending frames are "close". Moreover, given the desired transition locations we automatically modify the original database so cut transitions are possible. Specifically, at every transition the clips join seamlessly and any constraints in the motion (such as that a foot must planted on the ground) are enforced even if these constraints exist across transition boundaries.

In computer games and other virtual environments, move trees have demonstrated the utility of synthesizing motion based upon a hand-crafted graph. The main limitation of this technique is in the difficulty of the authoring process: the necessary manpower limits the complexity of the graphs and the range of applications that can afford to build them. Our framework provides an alternative to manual authoring that alleviates this problem.

3 Alternate Approaches

The computer animation literature provides a number of ways of generating motion for synthetic characters. Since virtual environments require continuous streams of motion, some approaches are clearly inappropriate. Two obvious examples are keyframing and motion capture, which only create individual, static clips. Similarly, while motion capture editing [4, 11, 3, 23, 19, 5] and multi-target motion interpolation [22, 20] allow one to adapt a motion to new circumstances, these methods are still only capable of producing individual clips.

Procedural approaches have the advantage, in principle, of being able to generate flexible motions of arbitrary length. Perhaps the largest class of such approaches is physically-based motion synthesis. While physically-based methods have been successful for many natural phenomena, they have failed to scale to the complexities of character motions, with the exception of a few particular actions such as running [6] and jumping [13]. More ad hoc procedural methods have succeeded at a larger range of character motions [17, 18], but they require each new motion to be generated by hand and often do not produce realistic results.

Some recent approaches to motion synthesis involve constructing mathematical models based on a set of motion capture data. In particular, researchers have used hidden markov models [2] and switched linear dynamic systems [12] to create new motion. Such methods provide a straightforward way of generating arbitrarily-long streams of motion, but as yet it is unclear how they can be adapted to provide the high-level control required for virtual environments.

A number of graph-based approaches to motion synthesis have recently been developed that fully automate the graph construction process [1, 8, 10]. These methods allow graphs to be constructed quite quickly at the expense of providing severely limited control over the graph structure; indeed the generated graphs were *unstructured*.

In contrast to the explicitly designer-structured graphs of the previous section, unstructured motion graphs have no pre-determined connections between movements, and can make no guarantees about how quickly one motion can be reached from another. The path between two motions might be complex. Therefore, methods for synthesizing motions from unstructured graphs rely on search. By looking ahead, the search algorithms make choices that not only meet current needs, but have paths to future goals.

Unstructured graphs are inappropriate for interactive applications for several reasons. Interactive systems preclude lookahead and therefore search algorithms. Another problem is that it is difficult to know what motions are possible in an unstructured graph, since the connectivity is complex. For example, if a designer knows that a certain set of transitions will be required for a character's actions, there is no way to ensure that they are contained in the graph. Third, the control approaches currently used in interactive applications rely on known structure. For these reasons, we believe that interactive applications demand designer control over the graph structure.

Recently the graph-based approach has been extended to manually-constructed graphs in which the fundamental unit is not a static clip of motion, but rather a set of carefully-chosen clips that can be interpolated [16]. Parameterizing these interpolations appropriately can given one a finer degree of control over a character; for example, in the work cited one could specify locomotion in a continuum of directions and speeds, rather than from a discrete set of choices. At present it is unclear how readily this approach generalizes to larger, more expressive sets of motions.

4 Constructing Graphs

We assume the user has a database of motion capture data in a standard skeletal format. The number of motions in the database is irrelevant; it might contain many short clips or a single long clip. Each frame of motion is represented by a vector of parameters $(\mathbf{p}, \mathbf{q_1}, \ldots, \mathbf{q_n}, \mathbf{o_1}, \ldots, \mathbf{o_n})$, where $\mathbf{p}$ is a three-vector specifying the position of the root joint in world coordinates, $\mathbf{q_i}$ is a quaternion specifying the orientation of the i^{th} joint in its parent's coordinate system, and $\mathbf{o_i}$ is a three-vector indicating the offset of the i^{th} joint in its parent's coordinate system[1]. We assume that there is some linear indexing of the corpus, so a particular frame's vector is denoted by $\mathbf{F_i}$ for frame i of the corpus.

We also assume the motions are annotated with relevant constraints on end-effector positions. In this paper we limit our attention to footplant constraints, which specify that either the heel or ball of a particular foot must be planted over some set of frames (hence a total of four possible constraints may exist on a given frame). These types of constraints are by far the most common in motion capture data, and so this restriction is minor.

In our framework each edge in a graph is a clip of motion and each node is defined by a group of frames at which transitions are to occur. This group of frames is called a *match set* and each element of the match set is a *match frame*. If the original database has n motions, then the corresponding graph has a trivial structure with $2n$ nodes and n edges; refer to Figure 3. Each match set naturally partitions the database into shorter clips, which in turn correspond to edges in the graph that attach at a common node.

[1]Most motion capture processing systems assume perfectly rigid skeletons, in which case $\mathbf{o_i}$ is not explicitly represented. We use this more general skeleton representation since we employ the constraint solver described in [9], which adds small length changes to bones.

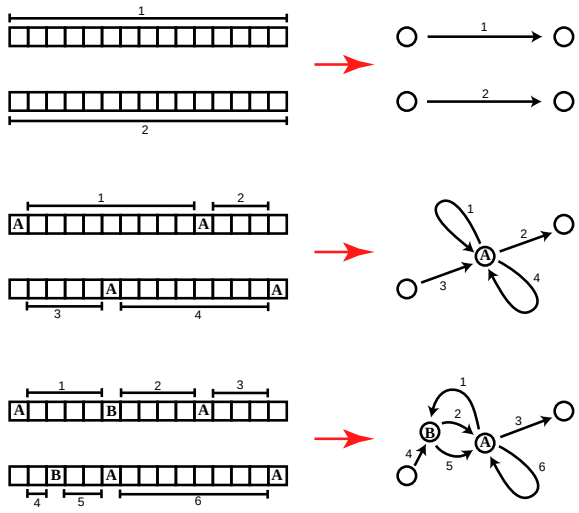


Figure 3: The top diagram schematically represents an initial database with two motions; on the left it is represented as two groups of frames and on the right is the corresponding graph. The middle diagram shows the result of making a match set out of four frames. This breaks the database into smaller clips and adds a new node to the graph. The bottom diagram demonstrates the addition of a second match set.

In our system graphs are built one node at a time by choosing match sets. If desired, an author can simply select the match frames manually. The author may also specify a particular frame and have the system automatically build a match set out of a group of similar frames. Finally, the author can have the system create a match set out of the largest collection of similar frames in the database.

Once the graph designer has finished creating match sets, our system automatically adjusts the motions so the corresponding transitions can be executed with simple cuts. This requires choosing a *common pose* for the match set, so that each match frame can be replaced by a rigid transformation of the common pose, and then transforming the surrounding frames such that this replacement is seamless. Any motion leading into the pose can then be followed by any of the motions exiting it, creating a multi-way transition.

The remainder of this section details our method. We first explain our process for helping a graph designer build match sets, then we describe our method for adjusting the original motions to generate seamless cut transitions, and finally we discuss the details of actually generating motion with the final graph.

4.1 Choosing Match Frames

Our system helps an author create match sets (and therefore nodes in the graph) by finding collections of frames that are similar to one another. This is accomplished through a scalar function $D(\mathbf{F_i}, \mathbf{F_j})$ that defines the distance between two frames $\mathbf{F_i}$ and $\mathbf{F_j}$. We use the same distance function as in [8], which has the advantage of automatically choosing a common coordinate system for $\mathbf{F_i}$ and $\mathbf{F_j}$. That is, since a motion is fundamentally unchanged by a rotation about the vertical axis and a translation along the floor plane, $\mathbf{F_j}$ needs to be "aligned" with $\mathbf{F_i}$ before the distance can be computed.

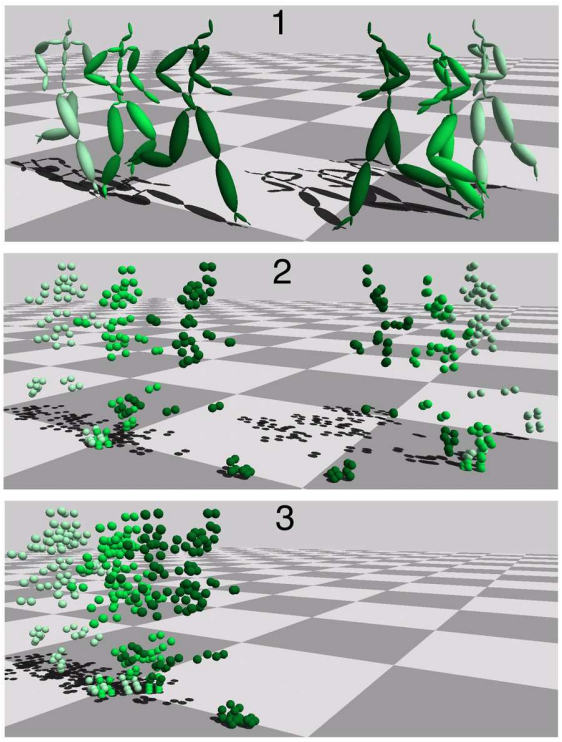


Figure 4: The distance between two frames $\mathbf{F_i}$ and $\mathbf{F_j}$ is calculated as follows. (1) Small neighborhoods of frames are extracted about $\mathbf{F_i}$ and $\mathbf{F_j}$. (2) These sets of frames are converted into two point clouds. (3) The optimal sum of squared distances between corresponding points is computed given that each point cloud can be rotated about the gravity axis and translated in the floor plane.

The distance calculation, motivated in [8], is shown in Figure 4. It works on clouds of points to avoid scaling issues in angle computations. First, small neighborhoods of frames are extracted around both $\mathbf{F_i}$ and $\mathbf{F_j}$. Two point clouds are then formed by attaching markers to the skeletons. Finally, the optimal weighted sum of squared distances is computed given that rigid 2D transformations may be applied to each point cloud. That is, we calculate

$$D(\mathbf{F_i}, \mathbf{F_j}) = \min_{\theta, x_0, z_0} \sum_k w_k \|\mathbf{p_{i,k}} - \mathbf{T}_{\theta, \mathbf{x_0}, \mathbf{z_0}} \mathbf{p_{j,k}}\|^2, \quad (1)$$

where $\mathbf{p_{i,k}}$ is the k^{th} point in the cloud generated from frame i and $\mathbf{T}_{\theta,\mathbf{x_0},\mathbf{z_0}}$ is a linear transformation consisting of a rotation of θ degrees about the y (vertical) axis followed by a translation of (x_0, z_0). The weights w_i sum to 1 and are chosen to give the most importance to $\mathbf{F_i}$ and $\mathbf{F_j}$ and less importance to frames toward the edges of the neighborhoods.

This optimization has the following closed-form solution:

$$\theta = \arctan \frac{\sum_i w_i (x_i z_i' - x_i' z_i) - (\overline{x}\overline{z'} - \overline{x'}\overline{z})}{\sum_i w_i (x_i x_i' + z_i z_i') - (\overline{x}\overline{x'} + \overline{z}\overline{z'})} \quad (2)$$

$$x_0 = (\overline{x} - \overline{x'}\cos(\theta) - \overline{z'}\sin\theta) \quad (3)$$

$$z_0 = (\overline{z} + \overline{x'}\sin(\theta) - \overline{z'}\cos\theta), \quad (4)$$

where $\overline{x} = \sum_i w_i x_i$ and the other barred terms are defined similarly.

For every pair of frames in the database there are two possible transitions, one that connects frames preceding $\mathbf{F_i}$ to frames following $\mathbf{F_j}$ and one that connects frames preceding $\mathbf{F_j}$ to frames following $\mathbf{F_i}$. D allows one to assign to each of these transitions a quality estimate and a coordinate transformation that aligns the ending motion with the starting motion. To speed interaction with our system, the distances and aligning coordinate transformations are precomputed for every pair of frames in the database.

Given a particular frame $\mathbf{F}$ and a user-defined threshold, we find a match set $S = \{\mathbf{F_1}, \ldots, \mathbf{F_n}\}$ as follows. For each motion in the database, we can form a 1D function by considering the distances between $\mathbf{F}$ and every frame of this motion. The local minima of these functions correspond to locally optimal transition points. We form a set S' of the frames corresponding to local minima whose values are below the threshold. These frames satisfy the similarity requirement for being match frames, but there is one more condition that must be met. Each match frame is associated with a displacement map that smoothly introduces the corresponding transitions into the motion database. As will be discussed more fully in Section 4.2, to create these displacement maps we require match frames to be at least w_{min} frames apart. So, in order of lowest distance to $\mathbf{F}$, we add to S the frames from S' that are at least w_{min} frames from every existing match frame.

If the graph designer wants $\mathbf{F}$ to serve as a hub node in the graph, then S determines the transitions that connect to this hub. By interactively choosing different thresholds the designer can determine an appropriate tradeoff between the number of edges attached to the hub and the quality of the resulting motions. The designer may also want to create a node based on the largest group of similar frames in the database. This can be found simply by forming S for every frame in the database and returning the one with the most elements. Ties are broken based on the lowest average distance between frames in S and the frame used to generate S.

4.2 Creating Transitions

Once the graph designer has finished creating match sets $S_1, \ldots, S_n$, our system adjusts the original database so the motions join seamlessly at all transitions points. Since transitions always occur between frames of a match set, it is sufficient to adjust the original motions such that the match frames are all identical, i.e., the values and velocities of each skeletal parameter are the same. If there are no constraints on the motions, this is accomplished solely through adaptation of displacement mapping techniques [23, 3]. If constraints *are* present, then matters are more complicated. Applying displacement maps will violate constraints, and if we subsequently use existing methods [4, 11, 9] to enforce them, the motions may change such that the match frames are no longer identical. We consider both of these cases, first treating transition generation in the absence of constraints and then when constraints exist.

4.2.1 Transitions Without Constraints

If constraints aren't present, then for each match set S_i our system creates an "average" frame $\mathbf{F_{S_i}}$ with a skeletal pose that is representative of the poses in the match frames. This pose is called the *common pose*. Our system then applies displacement maps that transform each match frame to have the common pose.

Figure 5 depicts our algorithm. In the original database the match frames are scattered about in a global reference frame. If we are to compute an average pose, the match frames must first be aligned. As discussed in Section 4.1, every pair of match frames $\mathbf{F_j}, \mathbf{F_k} \in \mathbf{S_i}$ has a rigid 2D transformation that aligns them for the

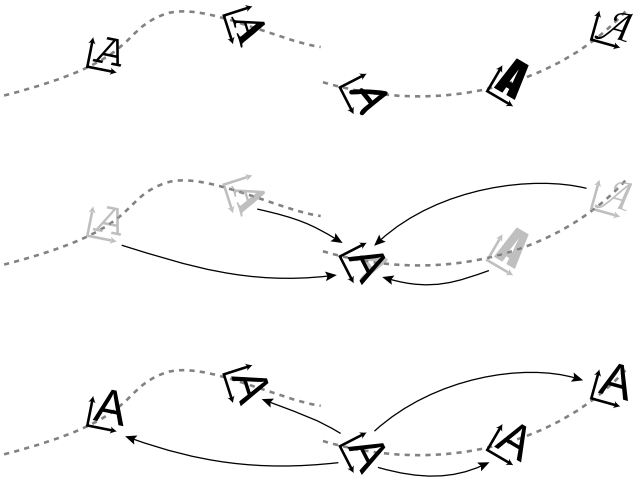


Figure 5: (top) In the original motions, match frames are scattered in the global coordinate system. (middle) We choose a particular match frame, align the others to it, and compute an average skeletal posture to serve as the common pose. (bottom) Using a set of displacement maps, each match frame is altered to have this common pose.

purposes of executing a transition. Let $\mathbf{T_{jk}}$ be the transformation that when applied to $\mathbf{F_j}$ aligns it with $\mathbf{F_k}$. Since each transformation was computed independently via equations $(2)-(4)$, in general they will be inconsistent in the sense that $\mathbf{T_{jk}T_{kl}} \neq \mathbf{T_{jl}}$. We could attempt to find a set of coordinate transforms that *are* consistent by, for example, adjusting Equation (1) to optimize simultaneously over several coordinate transforms. However, for more than two point clouds there is no simple closed form solution and an expensive nonlinear optimization would be necessary. On the other hand, we observe that if the match frames in S_i are sufficiently similar then the coordinate transformations will be *approximately* consistent. Hence we may simply select one particular match frame to define the coordinate transforms for every other match frame. Say we select $\mathbf{F_{j_{base}}}$. Then we redefine the $\mathbf{T_{pq}}$ to be

$$\mathbf{T_{pq}} := \mathbf{T'_{pq}} = \mathbf{T_{j_{base}p}T^{-1}_{j_{base}q}}. \quad (5)$$

These new coordinate transforms guarantee that $\mathbf{T_{pq}T_{qr}} = \mathbf{T_{pr}}$. We can now align the k^{th} match frame in S_i with $\mathbf{F_{j_{base}}}$ by applying the transformation $\mathbf{T_{kj_{base}}}$.

In practice $\mathbf{F_{j_{base}}}$ is not chosen arbitrarily. Rather, our system attempts to choose the match frame that is closest to being in the "center" of the other frames. This corresponds to choosing the match frame with the smallest sum of distances to the other match frames.

Once we have chosen $\mathbf{F_{j_{base}}}$, our system computes $\mathbf{F_{S_i}}$ by aligning the match frames into the coordinate system of $\mathbf{F_{j_{base}}}$. The root position, joint offsets, and joint orientations of $\mathbf{F_{S_i}}$ are the average of the corresponding quantities in the match frames. The average joint orientation is computed as in [16].

We can now form displacement maps that replace each $\mathbf{F_k} \in S_i$ with $\mathbf{T^{-1}_{kj_{base}}F_{S_i}}$ (Figure 5). Since each match frame is identical, motion is guaranteed to be continuous at transitions. This use of displacement maps is similar to previous work [10, 1] which used displacement maps to guarantee C_0 continuity at transitions. However, for motions with very different velocity characteristics C_0 continuity may be insufficient (Figure 6). For this reason we extend previous efforts by building displacement maps that preserve C_1 continuity. For each skeletal parameter we compute the average velocity over all match frames. We then construct displacement maps such that motions pass through the common pose with these parameter velocities. Since the motions are represented as discretely sampled signals, care must be taken in computing derivatives. Because continuity is most important at a scale greater than a single frame, we estimate derivatives by calculating finite differences at each point in a small window and filtering the results.

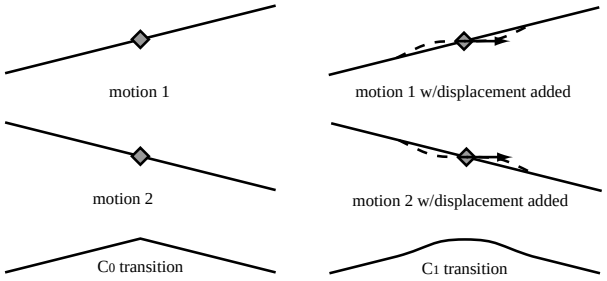


Figure 6: C_0 transitions can still cause discontinuities if motions have very different velocities. For this reason we use C_1 smooth displacement maps.

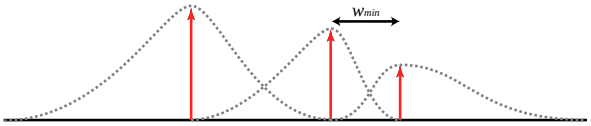


Figure 7: At each match frame a displacement map is used to smoothly alter the motion so as to facilitate transitions; this figure depicts a motion with three match frames and the corresponding displacement maps. On each side the displacement map extends up to either the next match frame or the motion boundary, whichever comes first. Displacement maps are required to extend at least w_{min} frames on either side, so match frames must be at least w_{min} frames apart.

Each side of a displacement map extends to either the nearest match frame or a boundary of the motion, whichever comes first (Figure 7). To ensure that changes do not occur too rapidly, we require match frames to be spaced at least w_{min} frames apart. If there are n joints in the skeleton, then the displacement map consists of $2n+1$ splines: one for the root position, n for the joint offsets, and n for the joint orientations. The ends of each spline have zero value and derivative and the center is chosen to map the relevant parameter to the target value and derivative. We construct these splines out of two Hermite cubic segments; for orientations we construct quaternion splines using the method in [7].

4.2.2 Transitions With Constraints

If displacement maps are applied to the original motions, then any constraints on those motions are likely to be violated. We now consider how to create smooth multi-way transitions while simultaneously preserving constraints. We focus on the most common kinds of constraints, which are footplant constraints. A footplant constraint specifies that either the left heel, right heel, left ball, or right ball must be fixed on the ground. To enforce a footplant constraint, two things must be done: 1) positions must be chosen for each constrained joint and then 2) the motion must be smoothly adjusted so the constrained joints are in these positions. We use the method of [9] to enforce footplant constraints. This algorithm has the important property that one can ensure that a particular frame is not altered by constraining the root, heels, and balls of the feet to remain in their current positions. We refer to this as *locking* the frame.

As in the previous section, our basic strategy is to construct a representative frame $\mathbf{F_{S_i}}$ for each match set S_i and use displacement maps to make the match frames identical to $\mathbf{F_{S_i}}$. We define a con-

straint to exist in $\mathbf{F_{S_i}}$ if and only if it exists on a majority of the match frames[2], which means that individual match frames may end up gaining and/or losing constraints.

Since constraints must be enforced in the final motion, $\mathbf{F_{S_i}}$ must satisfy all of its constraints, i.e., the constrained joints must be on the ground. Assume this is true. As in Section 4.2.1 we can apply displacement maps such that for all match sets each match frame is identical to the appropriate common pose. If we then lock each match frame and apply the constraint enforcement algorithm, our database of motions will have the desired properties: all constraints will be enforced and each match set will contain identical frames.

While we could choose the common poses using the same algorithm as in Section 4.2.1, this method fails to take into account constraint information. This is problematic since by locking each match frame, we are forcing the motions returned by the constraint solver to pass through the common poses. For example, say the left heel is *unconstrained* on some match frame that is only a few frames away from a region where the left heel must be planted. If the left heel happens to be far from the ground in the common pose, then the constraint solver will be forced to generate a motion where the foot leaves the ground with unnatural speed.

Intuitively, we would like to select the $\mathbf{F_{S_i}}$ such that when we replace each match frame with the appropriate common pose and lock it, the locking has as little effect as possible. That is, if we imagine *not* doing this locking and enforcing constraints, we would like the match frames to nonetheless remain unchanged. In light of this we use the following two-step iterative procedure for determining a particular $\mathbf{F_{S_i}}$. We start out by creating a "working set" that initially contains copies of the match frames as they appear in the original motions. Each iteration estimates the common pose by averaging the working set, and creates a variant of each motion that passes through this common pose using the same displacement map technique described in Section 4.2.1. This possibly violates constraints. Next, we apply the constraint enforcement algorithm to the modified motions, possibly adjusting the matched frames. After this the matched frames, which may no longer be identical, are copied back into the working set. Each iteration begins with the motion from the original database and evolves the common pose. At the end of the final iteration we set the common pose $\mathbf{F_{S_i}}$ to be the average of the poses in the working set. In our experiments only a small number of iterations (3-5) were necessary.

The $\mathbf{F_{S_i}}$ generated through the above algorithm will not necessarily satisfy their constraints. We can correct this by choosing positions for the constraints and applying inverse kinematics. However, constraint positions in general can not be found independently for each $\mathbf{F_{S_i}}$. In particular, if two common poses share a constraint and border a clip that has this constraint on each frame, then the constraint positions for the common poses must be chosen such that in this clip they are in the same location. This issue arises in many common situations, such as if the character stands in place. We describe a solution to this problem in the Appendix.

4.3 Generating Motion at Run Time

Each transition involves two pieces of information: the clip we're transitioning to and the coordinate transformation that aligns it with the current clip. At run time these coordinate transformations are the only information that needs to be kept track of. That is, to play the current clip, we simply adjust the root of every (precomputed) frame by the current coordinate transformation, and whenever we make a transition we update this transformation.

[2]We have found that requiring all match frames to have the same constraint state, as suggested by [10], forces us to exclude too many good potential matches.

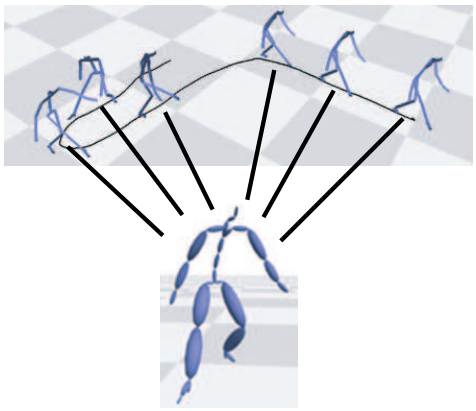


Figure 8: On top are the five frames of a match set generated automatically for a short sneaking motion. On the bottom is the corresponding common pose.

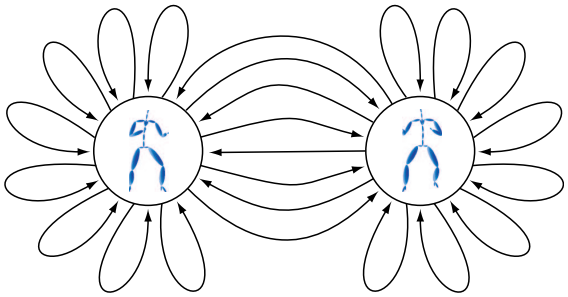


Figure 9: A schematic of the two-node martial arts graph generated with our system. Our algorithm for creating match sets automatically selected left and right "ready" stances as the hubs of the graph.

As discussed in previous graph-based approaches to motion synthesis [21, 10, 8], certain nodes of the graph may be dead ends in the sense that they are not part of any cycle. Once such a node is entered, there is a limit to how much further animation can be produced. This is unacceptable for virtual environments, since characters must be animated for arbitrarily long amounts of time. Our system notifies the graph designer of possible dead ends by finding the nodes that are not part of the largest strongly connected component [10, 8]. The designer may then decide to either add new transitions or remove these nodes.

5 Results

We have implemented a system based on the methods in Section 4 and applied it to a number of motion datasets. Figure 10 is a screenshot of the window seen by the graph designer. In the upper right is a visualization of the distance function; pixel (i, j) represents $D(\mathbf{F_i}, \mathbf{F_j})$, with darker pixels corresponding to lower distances. On the far upper right is a slice of the $2D$ distance function showing the distances between a frame selected by the user and the other frames in the database. The bottom of the window shows a schematic of the graph given the current match sets. The horizontal black lines represent original motions and the vertical lines indicate match frames. All frames of the same match set are the same color. Clicking on a segment in this schematic causes the corresponding clip to be displayed in the upper left window.

We created a set of graphs by having the system automatically create nodes based on the largest sets of match frames. To test the system, we started with a single motion of someone sneaking for thirteen seconds and built a graph with a single node and 7 clips; see Figure 8. We then moved on to larger data sets, constructing graphs and driving their input with a video game controller. We first built a two-node graph out of a dataset containing 900 frames (30 seconds) of martial arts motions (Figure 9). The common poses generated automatically by the system corresponded to two "ready" stances, one with the left foot forward and one with the right foot forward. We then mapped the clips to buttons on a gamepad, allowing a user to interactively direct the character to punch, kick, dodge, shuffle-step, and switch stances. We next built a one-node graph out of 3000 frames (100 seconds) of walking data. This graph allowed a user to guide a character by specifying the curvature of its path, where the options ranged from a gentle arc to a sharp about-face. Finally, we combined these two datasets into a larger graph that allowed all of the previous operations plus the ability to switch between walking and fighting modes.

The semi-automatic nature of our system makes it possible to produce graphs quite quickly. The total amount of time necessary to build the martial arts graph — from raw data to being able to interactively control a character — was about 12 minutes, and the walking graph took about 20 minutes. Most of this time was spent deciding how to map the clips to the gamepad.

6 Discussion

In this paper we have described a framework for synthesizing character motions in virtual environments by assembling clips built from a corpus of motion capture data. We meet the visual quality demands of virtual environments by preserving the fidelity of the original motions. We meet performance demands by performing all processing of the motions at authoring time, so at run time clips can simply be concatenated in appropriate orders. Finally, we meet controllability and responsiveness demands by allowing the user to guide the graph building process to ensure that the graph has a usable structure. Specifically, we support and encourage the creation of hub nodes that allow many different actions to be reachable from a common point.

Our approach automates tedious portions of the graph construction process and makes it possible to use data more opportunistically. This can allow graphs to be created from a wide range of data that was not specifically captured for graph construction, and it can also enable designers to build graphs of a scope that would otherwise be too expensive to produce.

The authoring tool described in this paper required several new techniques to be developed:

1. We automatically identify potential hub nodes, allowing a graph designer to avoid tedious parts of the construction process.

2. We introduce C_1 displacement maps as a means of creating higher quality cut transitions.

3. We provide a method for satisfying constraints as a preprocess, allowing the complexity of constraint satisfaction to be avoided at run time.

The run-time execution of our approach is intentionally similar to current (and successful) methods that use manually constructed graphs. We believe this will make it easier to apply our methods in practical virtual environments. Moreover, by reducing the effort required to construct graphs suitable for run-time synthesis, we hope to make run-time animation accessible to a broader array of applications.

Acknowledgements

We would like to thank everyone in the UW graphics group for their help with this project. Motion data was generously provided by Demian Gordon and House of Moves Studios. This research was supported in part by a Wisconsin University-Industry Relations Grant, NSF grants CCR-9984506 and CCR-0204372, and equipment donations from Intel. Lucas Kovar is supported by an Intel Foundation Fellowship.

References

[1] Okan Arikan and D.A. Forsythe. Interactive motion generation from examples. In *Proceedings of ACM SIGGRAPH 2002*, Annual Conference Series, July 2002.

[2] Matthew Brand and Aaron Hertzmann. Style machines. In *Proceedings of ACM SIGGRAPH 2000*, Annual Conference Series, pages 183–192, July 2000.

[3] Armin Bruderlin and Lance Williams. Motion signal processing. In *Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, pages 97–104, August 1995.

[4] Michael Gleicher. Retargeting motion to new characters. In *Proceedings 0f ACM SIGGRAPH 98*, Annual Conference Series, pages 33–42, July 1998.

[5] Michael Gleicher. Motion path editing. In *Proceedings 2001 ACM Symposium on Interactive 3D Graphics*, March 2001.

[6] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating human athletics. In *Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, pages 71–78, August 1995.

[7] Myoung-Jun Kim, Myoung-Soo Kim, and Sung Yong Shin. A general construction scheme for unit quaternion curves with simple high order derivatives. In *Proceedings of ACM SIGGRAPH 1996*, Annual Conference Series, pages 369–376, August 1996.

[8] Lucas Kovar, Michael Gleicher, and Fred Pighin. Motion graphs. In *Proceedings of ACM SIGGRAPH 2002*, Annual Conference Series, July 2002.

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

[10] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive control of avatars animated with human motion data. In *Proceedings of ACM SIGGRAPH 2002*, Annual Conference Series, July 2002.

[11] Jehee Lee and Sung Yong Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of ACM SIGGRAPH 99*, Annual Conference Series, pages 39–48, August 1999.

[12] Yan Li, Tianshu Wang, and Heung-Yeung Shum. Motion texture: A two-level statistical model for character motion synthesis. In *Proceedings of ACM SIGGRAPH 2002*, Annual Conference Series, July 2002.

[13] C. Karen Liu and Zoran Popović. Synthesis of complex dynamic character motion from simple animations. In *Proceedings of ACM SIGGRAPH 2002*, Annual Conference Series, July 2002.

[14] Alberto Menache. *Understanding Motion Capture for Computer Animation and Video Games*. Academic Press, San Diego, CA, 2000.

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

[16] Sang Il Park, Hyun Joon Shin, and Sung Yong Shin. On-line locomotion generation based on motion blending. In *ACM Symposium on Computer Animation 2002*, July 2002.

[17] Ken Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15, March 1995.

[18] Ken Perlin and Athomas Goldberg. Improv: A system for scripting interactive actors in virtual worlds. In *Proceedings of ACM SIGGRAPH 96*, pages 205–216, August 1996.

[19] Zoran Popović and Andrew Witkin. Physically based motion transformation. In *Proceedings of ACM SIGGRAPH 99*, Annual Conference Series, pages 11–20, August 1999.

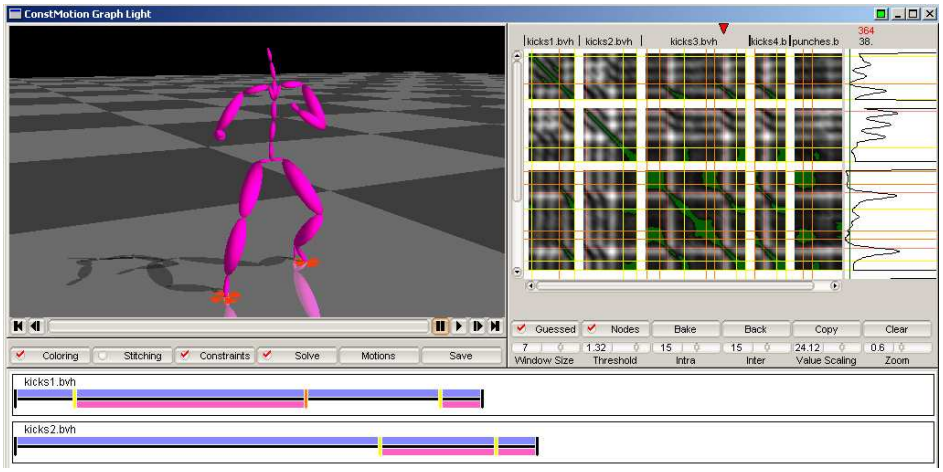


Figure 10: A screenshot of our graph-building interface.

[20] C. Rose, M. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Application*, 18(5):32–40, 1998.

[21] Arno Schodl, Richard Szeliski, David H. Salesin, and Irfan Essa. Video textures. In *Proceedings of ACM SIGGRAPH 2000*, Annual Conference Series, pages 489–498, August 2000.

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

[23] Andrew Witkin and Zoran Popović. Motion warping. In *Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, pages 105–108, August 1995.

Appendix

The $\mathbf{F_{S_i}}$ generated through the iterative algorithm of Section 4.2.2 will in general not satisfy their constraints unless these constraints are explicitly enforced. This amounts to identifying positions for each constraint and then performing inverse kinematics to ensure the relevant joints reach these positions. This process is complicated by the fact that the choice of constraint positions can not be made independently for each $\mathbf{F_{S_i}}$. Consider the case where $\mathbf{F} \in S_i$ and $\mathbf{F}' \in S_j$ share some constraints and border a clip that also has these constraints on every frame. For the resulting motion to be continuous, we require that $\mathbf{F_{S_i}}$ and $\mathbf{F_{S_j}}$ (when transformed to be aligned with $\mathbf{F}$ and $\mathbf{F}'$) place the constrained joints in the same location; $\mathbf{F_{S_i}}$ and $\mathbf{F_{S_j}}$ are *linked* on these constraints. Since constraints can exist anywhere in the original motions, common poses can be linked arbitrarily.

Linked constraints are not an artifact of having a bizarre set of motions. On the contrary, they occur in quite ordinary datasets. Consider, for example, a set of motions of someone waiting around impatiently. The character might shuffle its feet, tap its toes, and make subtle shifts in posture to redistribute its weight. In the likely event that constraints exist on every frame of the dataset, *every* common pose will have linked constraints with every other common pose.

To ensure continuous motion, linked constraint positions of $\mathbf{F_{S_i}}$ and $\mathbf{F_{S_j}}$ need only be identical up to a $2D$ rigid transform. Recall that when making a transition we first align the starting and ending motions so the match frames are in the same position and orientation. Section 4.2.1 explained how to determine these coordinate transformations based on the values we computed for D (Section 4.1). However, we are free to pick different transformations, and in particular we can select ones specifically to align constraint positions.

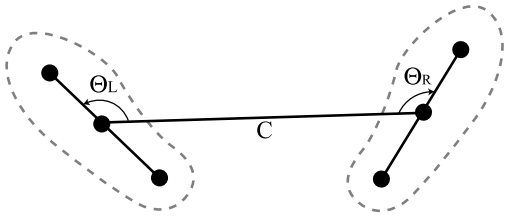


Figure 11: Up to a rigid $2D$ transformation, the configuration of two feet that are flat on the floor is uniquely defined by the distance between the centers of the feet and the orientation of each foot relative to the line connecting the centers.

So: the problem is to ensure that any linked constraint positions are identical up to a $2D$ rigid transformation, or *rigidly similar*. We can determine how common poses are linked simply by looking at every clip and determining whether the bordering match frames share constraints that exist throughout the clip. If the only linked constraints are on joints of the same foot, then since the foot is rigid the constraint positions are automatically rigidly similar. If linked constraints exist for all four joints on the feet, then let $\mathbf{C_1}$ and $\mathbf{C_2}$ be the segments connecting the centers of the feet in, respectively, the starting common pose and the ending common pose (Figure 11). Also, let Θ_{L_1} and Θ_{R_1} be the orientations of the left and right foot in the starting common pose relative to $\mathbf{C_1}$, and let Θ_{L_2} and Θ_{R_2} be defined similarly. To ensure rigid similarity it is sufficient to require $\|\mathbf{C_1}\| = \|\mathbf{C_2}\|$, $\Theta_{L_1} = \Theta_{L_2}$, and $\Theta_{R_1} = \Theta_{R_2}$. If there are only two or three linked constraints and they exist on joints of different feet, then the situation can be reduced to the four-joint case by rotating any foot with only one linked joint about that joint such that it is flat on the floor.

We can divide common poses into equivalence classes via constraint linkage. Each common pose in an equivalence class has linked constraints on both feet with at least one other common pose in that same class. For each equivalence class, we find the average foot orientations and distance between the foot centers. Each common pose is then adjusted to have these average parameters.

We have now ensured that every set of linked constraint position are rigidly similar. However, the coordinate transformations that align clips (as computed in Section 4.2.1) may not align the constraints positions. This can be addressed by redefining these coordinate transformations such that the constraint positions are identical for the last frame of the starting clip and the first frame of the ending clip.

8.3 Animation Planning for Virtual Characters Cooperation

Animation Planning for Virtual Characters Cooperation

CLAUDIA ESTEVES, GUSTAVO ARECHAVALETA, JULIEN PETTRÉ, and JEAN-PAUL LAUMOND
Laboratoire d'Analyse et d'Architecture des Systèmes, Toulouse, France

This paper presents an approach to automatically compute animations for virtual (human-like and robot) characters cooperating to move bulky objects in cluttered environments. The main challenge is to deal with 3D collision avoidance while preserving the believability of the agent's behaviors. To accomplish the coordinated task, a geometric and kinematic decoupling of the system is proposed. This decomposition enables us to plan a collision-free path for a reduced system, then to animate locomotion and grasping behaviors independently, and finally to automatically tune the animation to avoid residual collisions. These three steps are applied consecutively to synthesize an animation. The different techniques used, such as probabilistic path planning, locomotion controllers, inverse kinematics and path planning for closed kinematic chains are explained, and the way to integrate them into a single scheme is described.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-dimensional Graphics and Realism; G.3 [**Probability and Statistics**]: Probabilistic Algorithms

General Terms: Algorithms

Additional Key Words and Phrases: Autonomous characters, behavior modeling, motion control, motion planning

1. INTRODUCTION

Over the last few decades, many attempts to animate the human figure have been made [Parent 2001; Earnshaw et al. 1998], each of them attaining a certain degree of autonomy, interactivity, and user-controllability.

The human figure has been frequently represented in computer animation in the same way articulated mechanisms are represented in robotics [Badler et al. 1993]. Nevertheless, the techniques developed to generate the motions for these mechanisms have been different in both areas. In computer animation, the interest has been mainly in generating realistic-looking motions, while in robotics the goal has been to automate motion generation regardless of its realism.

In recent years, applications arising in both areas (e.g., ergonomics, interactive video games, etc.) have motivated researchers to automatically generate human-like plausible motion. Consequently, this work deals with the development of an automatic motion strategy for the cooperation of two or more virtual characters that transport an object in a 3-dimensional cluttered environment. The virtual characters are considered to be either human figures with walking capabilities, referred to in this work as *mannequins*, or mobile robots. This work finds its applications mainly, but not exclusively, in PLM (Product Lifecycle Management) as in the maintenance and operation of industrial facilities [Badler et al. 2002].

C. Esteves and G. Arechavaleta benefit from SFERE-CONACyT grants. This work is partially funded by the European Community Projects FP5 IST 2001-39250 Movie and FP6 IST 002020 Cogniron.
Authors' addresses: {cesteves,garechav,jpettre,jpl}@laas.fr.

In this context, several behaviors should be combined within a single animation sequence: agents should walk or slide while manipulating a bulky object coordinately with the other characters. Here, the main challenge is to achieve 3D collision avoidance while preserving the believability of the agents behaviors.

From an algorithmic perspective, our approach is a centralized one. Indeed, we show how to model the global task within a single system which contains all the degrees of freedom (DOFs) of the agents and the object. This system is automatically built by computing a *reachable cooperative space*. Then, three consecutive steps are performed.

(1) Plan a collision-free trajectory for a reduced model of the system.

(2) Animate, in parallel, locomotion and manipulation behaviors.

(3) Tune the generated motions to avoid residual collisions.

These steps are applied by making use of a probabilistic motion planner to compute the collision-free trajectories; motion controllers adapted to each kind of character for both locomotion and grasping behaviors; and path planning algorithms for closed kinematic chains to deal with coordinated behaviors.

The remainder of this article is structured as follows. Section 2 gives a brief overview of related work. Section 3 introduces the different techniques used in this work. In Section 4, our system is described and the underlying principles of our approach are stated. Section 5 details the three steps performed in order to generate an animation. Experimental results are shown and discussed in Section 6.

2. RELATED WORK AND CONTRIBUTION

In order to build our motion planner, we have chosen the techniques that best provide the desired characteristics to the resulting animation; that is, believable and automatic motion, precise manipulation and combined behaviors.

Our system is represented with a tree-like structure with a high number of degrees of freedom and kinematic constraints. This representation is well suited for probabilistic motion planning techniques. Several works have proposed motion planners based on this kind of probabilistic approach to produce collision-free human-like trajectories. However, they have mainly focused on generating trajectories either for locomotion or for manipulation. Our work addresses motion planning in the context of manipulating and walking at the same time.

2.1 Motion Planning for Walking Characters

In order to generate plausible human-like motions, techniques based on motion capture have frequently been used. These techniques have proven to be suitable for real-time motion generation. However, the examples in a motion library are limited in number, and it is often necessary to modify them to avoid repetitive motions. This is usually done by using interpolation procedures such as in Witkin and Popovic [1995]; Unuma et al. [1995]; Rose et al. [1998]. Because we are dealing with eye-believable locomotion generation which is repetitive by nature, controllers based on motion capture are particularly adequate.

Motion planners exploiting these kind of techniques have been described in the literature. In Kuffner [1998] and Choi et al. [2003], two-step motion planners for walking virtual mannequins are presented. The first approach consists in first planning a collision-free path for a cylinder in a two-dimensional world and then following it by means of a PD controller that uses cyclic motion capture data. The latter approach is capable of dealing with rough terrains by planning the mannequin's footprints and then applying and adapting captured motion to attain the planned footprints. Our work is based on the first approach, extended in Pettré et al. [2003]. Here, the 3-dimensionality of the environment is taken into account by applying a collision avoidance-based warping method described in Section 5.3.

2.2 Motion Planning for Object Manipulation

As far as manipulation and reach planning for virtual mannequins are concerned, the problem has been tackled using different approaches.

In Liu and Badler [2003], the authors plan and synthesize collision-free reaching motions for 7-DOF arms in real time by using graphics hardware to detect collisions. Their planning framework reasons directly in the workspace. However, this kind of planning approach has not proven to be more efficient than the configuration-space planning approaches used in Kallman et al. [2003] and Yamane et al. [2004].

In Kallman et al. [2003], the authors plan reaching motions for a 22-DOF system (9 DOFs for each arm and 4 additional DOFs to specify the mannequin's body posture). Inverse kinematics algorithms are used to specify the initial and final configurations, and a roadmap is constructed based on the cost computed for each configuration. This approach can manage obstacle displacements by dynamically updating the roadmap.

In Koga et al. [1994] and Yamane et al. [2004], a trajectory is first found for the object to be manipulated or grasped and then the loop is closed between the arms and the object to follow the computed trajectory. Our work is conducted in the same spirit. Our contribution here is to address the grasping task at the planning level itself.

2.3 Combining Behaviors

The problem of combining behaviors of articulated human figures has been frequently tackled by applying behavior-based controllers as in Perlin [1995]; Blumberg and Galyean [1995]; Brand and Hertzmann [2000]. Here, motion capture data is labeled as containing one particular behavior or characteristic (run, walk, scratch head, etc.). A new walking-scratching-head sequence can be generated by interpolating configurations of the original captured data.

Kovar et al. [2002] captures a set of labeled motion examples in a graph. A believable animation is produced by composing motion captures corresponding to chosen edges and the transitions between them (nodes). A path can be followed by minimizing the error between the path in the graph and the user-sketched one. In Kallmann et al. [2004], a sampling approach using parameterized motion primitives is used to satisfy a given task. Motion primitives are chosen by means of a tree in which nodes are valid configurations reachable by more than one primitive and edges represent the paths between them. Their chosen parameterization allows the computation of trajectories maintaining constraints such as balance along it.

Physically-based solutions are described in Shiller et al. [2001] where a simple motion planner is presented to combine behaviors (walking, crawling and side-walking) in a sequential manner. Another approach is presented in Faloutsos et al. [2001] where individual controllers generating different behaviors are managed by a supervisor controller.

In our work, behaviors are combined by using the geometric and kinematic decoupling approach described in Section 4 which consists of the decomposition of the DOFs of the character. In this work, we combine both grasping and locomotion behaviors in a continuous way.

Therefore, the main contribution of this article is to address all these issues in a single scheme and, at the same time, deal with three-dimensional collision avoidance.

3. TECHNIQUES OVERVIEW

In order to generate complete motion sequences of one or more virtual characters transporting a bulky object in cluttered environments, we use three main components:

—a motion planner that handles open and closed kinematic chains,

—motion controllers adapted for virtual mannequins and for virtual robots,

—a three-dimensional collision avoidance editing strategy.

Many existing techniques can be used to cover these requirements. In the paragraphs that follow, we describe those that we experienced as the best adapted to our problem.

3.1 Probabilistic Motion Planning Techniques

3.1.1 *Probabilistic Roadmaps.* The aim of this method is to obtain a representation of the topology of the collision-free space [Latombe 1991] in a compact data structure called a *roadmap*. Such a structure is used to find collision-free trajectories and is computed without requiring an explicit representation of the obstacles in the configuration space. A roadmap can be obtained by using two types of algorithm: *sampling* or *diffusion*.

The main idea of the sampling technique (introduced as *PRM* in Kavraki et al. [1996]) is to draw random configurations lying in the free space and to trace edges to connect them with neighbor samples. Edges or local paths should also be collision-free and their form depends on the kinematic constraints of the moving device.

The principle of diffusion techniques [LaValle 1998] consists of sampling the collision-free space with only a few configurations called *roots* and to diffuse the exploration in the neighborhood to randomly chosen directions.

In this work, we use a variant of the first approach, the *Visibility PRM* [Siméon et al. 2000]. Here, there are two types of nodes: the guards and the connections. Nodes are added if they are not visible from previously sampled nodes (guard nodes) or if they allow two or more connected components of the roadmap (connection nodes) to be linked. The generated roadmap is more compact than the one obtained using PRM alone.

3.1.2 *Planning for Open Kinematic Chains.* When using sampling-based methods, a trajectory is found in a roadmap by using a two-step algorithm consisting of a learning phase and a query phase. For an articulated mechanism, the roadmap is computed in the learning phase by generating random configurations within the range allowed for each DOF. In the query phase, the initial and final configurations are added as new nodes in the roadmap and connected with the chosen steering method. Then, a graph search is performed to find a collision-free path between the start and goal configurations. If a path is found, then it is smoothened to remove useless detours. Afterwards, it is converted into a trajectory (a time-parameterized path) by means of classical techniques (e.g., Lamiraux and Laumond [1997]).

The advantage of using sampling methods when dealing with complex static environments as in this work, is that the learning phase can be precomputed offline and then the roadmap is used online to find a trajectory, saving time on each query.

3.1.3 *Planning for Closed Kinematic Chains.* In order to handle the motions of closed kinematic mechanisms, some path planning methods have been proposed in the literature [LaValle et al. 1999; Han and Amato 2000; Cortés and Siméon 2003]. In our work, we have chosen to use the *Random Loop Generator (RLG)* proposed in Cortés and Siméon [2003] and summarized in Algorithm 1. In this method, a closed kinematic chain is decomposed into active and passive parts. The main idea is to progressively decrease the complexity of the closed kinematic chain processed at each iteration until only the configuration of the passive part of the chain remains to be solved by means of an inverse kinematics algorithm.

Algorithm 1. RLG_SINGLELOOP

Input : the loop L
Output : the configurations $q[n_{sol}]$
begin
  $q^a \leftarrow$ SAMPLE_$q^a(L)$;
  $q^p[n_{sol}] \leftarrow$ COMPUTE_$q^p(L, q^a)$;
  if $n_{sol} = 0$ then **return** Failure;
  else $q[n_{sol}] \leftarrow$ COMPOUNDCONF$(L, q^a, q^p[n_{sol}])$;
end

The joint values of the active chain q^a are computed sequentially by using Algorithm 2 where J_b through J_e are the active joints on the chain. The range at which each joint is sampled depends on the configuration of the previously processed joints. This closure range for each joint is approximated by a simple bounding volume whose parameters depend on the geometry of the chain. The *reachable workspace* of a kinematic chain is defined as the volume which the end-effector can reach. The reachable workspace is automatically approximated by a spherical shell, defined as the intersection of the volume between two concentric spheres and a cone. The parameters to construct this volume are mainly the origin of the chain and the maximum and minimum extensions of the chain. A guided random sampling of the configuration of the active part is done inside this volume by limiting the value for each joint to the interval computed in the function COMPUTECLOSURERANGE in Algorithm 2. In function COMPOUNDCONF, for a sampled value of q^a, the value of q^p is computed by solving an inverse kinematics problem. When several loops are present in the mechanism, they are processed as separate closed chains.

When the roadmap is constructed, a trajectory is found in the same way as for open kinematic chains.

Algorithm 2. SAMPLE_q^a

Input : the loop L
Output : the parameters q^a
begin
  $(J_b, J_e) \leftarrow$ INITSAMPLER(L);
  while not ENDACTIVECHAIN(L, J_b) **do**
    $I_c \leftarrow$ COMPUTECLOSURERANGE(L, J_b, J_e);
    if $I_c = \emptyset$ **then goto** line 2;
    SETJOINTVALUE$(J_b,$ RANDOM$(I_c))$;
    $J_b \leftarrow$ NEXTJOINT(L, J_b);
    if not ENDACTIVECHAIN(L, J_e)SWITCH(J_b, J_e);
  end
end

3.2 Motion Generation Techniques

Once a trajectory is found, the appropriate motions to follow it as accurately as possible should be produced. In the following paragraphs, we describe the techniques used to generate such motions for walking as well as for manipulating.

3.2.1 *Manipulation Control.* Kinematics-based techniques allow the motion to be specified independently of the underlying forces that produced them. Motion can either be defined by specifying the value of each joint (forward-kinematics) or it can be derived from a given end-effector configuration (inverse-kinematics). These kinds of techniques have been frequently used to generate the motions of virtual mannequins as in Zhao and Badler [1994]; Yamane and Nakamura [2003]; Baerlocher and Boulic [2004]. In our approach, IK is used to provide precise control of the grasping behavior. Several IK algorithms for 7-DOF anthropomorphic limbs have been developed based on comfort criteria in

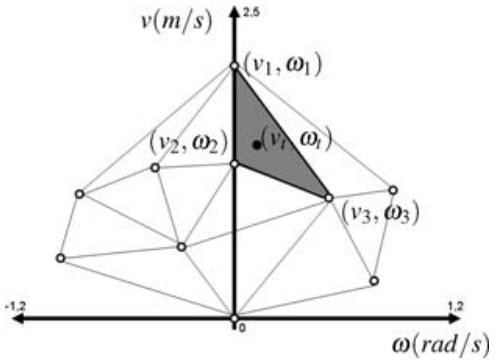


Fig. 1. Each captured source is represented as a point in the v–ω space according to their average linear and angular velocities. A new walking cycle will be obtained by interpolating its three closest sources.

order to best reproduce human-arm motions (e.g. Kondo [1991; Tolani et al. [2000]). We have chosen to use the analytic IK method presented in Tolani et al. [2000] to specify the configuration of our virtual mannequin's arms. To specify the configuration of virtual robot's 6-DOF arms, we use the algorithm described in Renaud [2000].

3.2.2 *Locomotion Control.* In order to specify the configuration of the walking virtual mannequin, we use the motion capture-based controller described in Pettré and Laumond [2005]. Here, the problem is modeled in such a way that the best motion examples to be blended as well as their respective weights are chosen by a simple geometric computation. The key idea is to represent all motion examples in a given library as single points lying in a 2-dimensional velocity space (linear and angular velocities) as in Figure 1. These points are then structured into a Delaunay triangulation, a well-known data structure in computational geometry, which allows efficient queries for point locations and nearest neighbor computations. The control scheme is based on a blending operator that combines three sources of motion capture. Their respective weights are automatically computed by solving a simple linear system with three unknown variables. The input parameters of the locomotion controller are, therefore, the sequence of sampled (v_t, ω_t), where v_t and ω_t are the linear and angular velocities of the virtual characters trajectory. Then, for each pair (v_t, ω_t) that represents one locomotion cycle, the three motion capture sources (v_1, ω_1), (v_2, ω_2), and (v_3, ω_3), with the closest average speeds to the sampled velocities are blended. As a result, a new locomotion cycle with the desired velocities is computed.

Finally, in order to obtain a continuous animation along the trajectory, the frequency characteristics of each cycle are computed and cycles are interpolated as described in Pettré and Laumond [2005].

4. MODELING THE SYSTEM

4.1 Virtual Characters

Our system is composed of two or more virtual characters and a movable object lying in a 3-dimensional cluttered environment. Each virtual character is classically represented as a hierarchy of rigid links connected by joints. In this work, we consider two kinds of virtual characters, human figures or mannequins and mobile robot manipulators.

The skeleton of our virtual mannequins is composed of 20 rigid bodies articulated by 18 joints with 53 degrees of freedom (DOFs). These joints and bodies are arranged in five kinematic chains that converge in the mannequin's root located on its pelvis (see Figure 2(a)).

Fig. 2. Our virtual characters are represented as tree-like structures formed of rigid bodies linked with spherical, rotational, and planar joints.

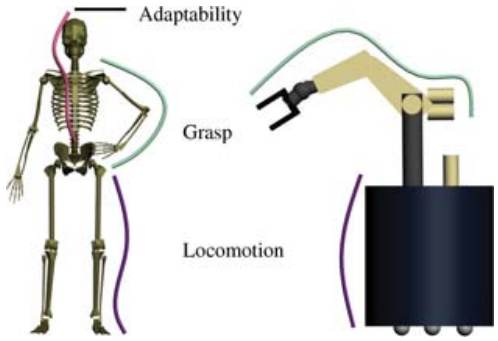


Fig. 3. Our system degrees of freedom are decomposed in three groups, locomotion, grasp, and adaptability.

Analogously, our virtual robot manipulators are composed of 7 rigid bodies linked by 6 joints and a mobile platform giving rise to a model with 9 degrees of freedom. These joints form one kinematic chain attached to the base of the robot (Figure 2(b)).

On top of the geometric model, kinematic constraints are imposed. For instance, we might want to consider that virtual human mannequins are allowed to walk only forwards or mobile manipulators to be differential drive robots. Kinematic constraints can also be imposed upon the object depending on the application, for example, keeping a tray with wine glasses horizontal. These constraints will be processed within the motion planning strategy described in Section 5.

4.2 Behavior-Based Kinematic Model

The main underlying principle of our work is a geometric decoupling of the system. This means that the system DOFs are decomposed in groups according to the main task they perform. The system contains the three groups of DOFs illustrated in Figure 3, locomotion, grasp, and adaptability.

Locomotion DOFs are the ones involved mainly in the steering of the character in the environment. For the virtual mannequin, these are the DOFs located in its legs and pelvis. For virtual robots, locomotion DOFs are the ones moving its base.

In a similar way, *Grasp* DOFs are in charge of the tasks that involve manipulation, that is, the arms of the characters. In this work, the mannequin's arms are redundant 7-DOF kinematic chains, and the virtual robots are equipped with 6-DOF nonredundant manipulators.

Adaptability DOFs are those involved in neither locomotion nor in manipulation but that allow a complementary posture control. For virtual mannequins, these DOFs lie in the head and spine. In our current virtual robots, there are no adaptability DOFs.

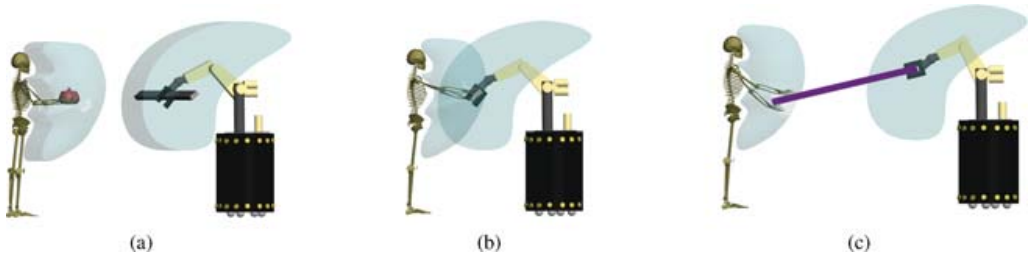


Fig. 4. (a) Individual reachable workspaces (b) To obtain a cooperative space for manipulation, the individual workspaces are intersected. (b) When manipulating a large object, the individual workspaces are enlarged by the objects size to compute the cooperative reachable workspace.

The advantage of such a geometric decoupling approach is that a reduced model of the system is obtained for each of the different steps of the planner. In this way, the control and description of the current task is simplified.

4.3 Reachable Cooperative Space

To attain cooperation between characters, a description of the space where the object can be manipulated is needed.

For a single virtual character manipulating an object with both hands, the space in which the object has to lie in order to remain reachable is defined by the possible solutions of the inverse kinematics algorithm applied to the arms. In this way, the reachable space can be approximated as described in Section 3.1.3 with the *spherical shells* technique consisting of the intersection of simple geometric shapes with the limits given by the maximal and minimal arm extension. Figure 4(a) shows an approximation of such a space for each type of character.

In this work, as the characters are supposed to carry the same object, we consider the *reachable cooperative space* as the intersection of all individual spaces (see Figure 4(b)). In order to achieve this intersection, a nonrigid link between the characters is considered.

Note that in the case of a large object, as in Figure 4(c), individual reachable spaces are not intersecting, nevertheless both characters are still holding the object. This is achieved by considering the object as the end effector of the arms kinematic chain for each of the characters. The individual space is therefore enlarged and the cooperative workspace obtained.

5. THE THREE-STEP ALGORITHM

Our approach relies mainly on three stages: planning, animating, and tuning. Several techniques are suitable for solving each stage. We have adopted some of them in a hierarchical manner to finally achieve collision-free planning motions for the whole system.

The user-specified inputs for the algorithm are:

— a geometric and kinematic description of the system A,
— geometric description of the environment E,
— maximal linear velocity and acceleration P,
— number of the desired frame-rate in the animation also stored in P,
— a motion library containing captured data from different walking sequences $MLib$,
— the maximum number of failures before the algorithm stops $ntry$,
— an initial and a final configuration.

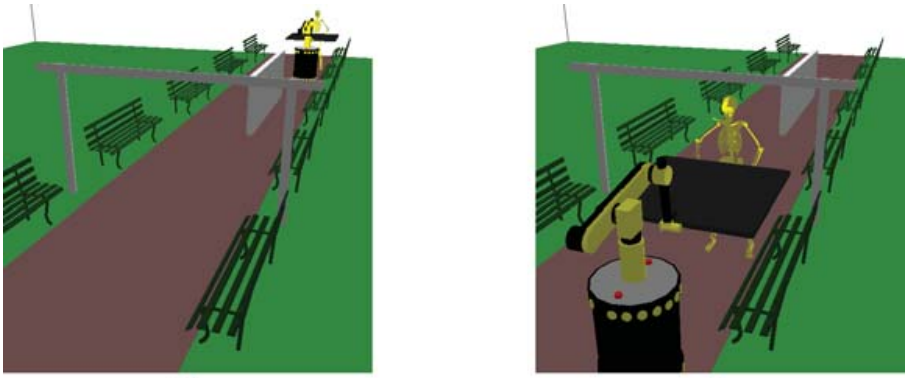


Fig. 5. Starting and goal configurations of the system for our workout example.

Algorithm 3. GLOBAL PLANNING

Input : the system A, the environment E, the list of parameters P
Output : the configurations $A(q_i)$
begin
  $stop \leftarrow$ False;
  while $\neg stop$ or $j < ntry$ **do**
    $path \leftarrow$ COMPUTEPATH(A_{simp}, E);
    if $path = \emptyset$ **then**
      $stop \leftarrow$ True;
    end
    else
      $traj \leftarrow$ SAMPLING($path$, P);
      $A(q_i) \leftarrow$ GENERATEANIMATION ($traj$, A, $MLib$);
      if $\neg$ TUNING($A(q_i)$) **then** $j \leftarrow j + 1$;
      else
        $stop \leftarrow$ True;
      end
    end
  end
end

The output is an animated sequence of the combined behaviors $A(q_i)$. Algorithm 3 summarizes the various steps. In the next paragraphs, each stage is described. We use the workout example of Figure 5 to illustrate the algorithm.

5.1 Path Planning

In the planning phase, a reduced geometric model of the system is used. This simplified model is defined by three different elements: two boxes bounding the locomotion DOFs of each character and the object DOFs (see Figure 6). This means that a 12-DOF system is considered at this level. Six of them are the 3-dimensional position and orientation of the object. The other six are the planar position and orientation of each character's box. Three DOFs will be added to the reduced model for each additional character present in the scene.

Given the user-defined initial and final configurations of the system in the 3D environment (Figures 5(a) and (b), respectively) , the first procedure, COMPUTEPATH, plans a path for the simplified

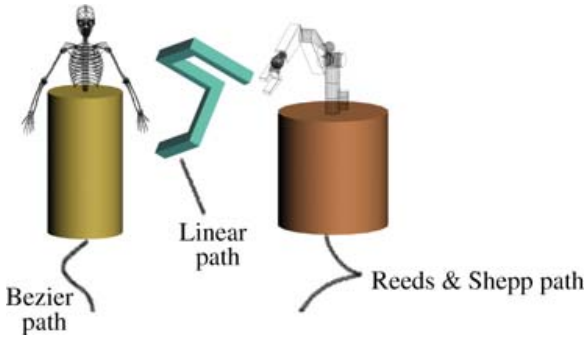


Fig. 6. A 12-DOF reduced geometric model of the system is employed in the planning phase. Each of the bodies will have an associated steering method while planning a path for the system.

model previously described. For this, the probabilistic roadmap method from Section 3.1.1 is used. This approach is performed in two steps, a learning and a query phase. The principle of the learning phase is to generate a random valid configuration for the system within the allowed range of the articulation limits for each DOF. In addition, in this stage, the cooperative reachable space as well as the maximal and minimal extension of the nonrigid link between the characters is computed. The random configurations are sampled within these limits.

During the sampling, a local planner tries to connect pairs of random configurations to incrementally construct a roadmap that captures the topology of the configuration space. Having the precomputed roadmap, the query phase performs a graph search to find feasible paths between the initial and final configurations.

The sampling strategy that we use is described in Section 3.1.1. This strategy captures well the topology of the configuration space in a compact roadmap. Local paths are computed by applying the adequate steering method. The selection of the steering method depends on the kinematic structure of each mobile entity that is part of the system. Let us consider, for instance, the construction of the local path for the virtual mannequin in the 12-DOF reduced model of Figure 6. Here, we chose Bezier curves of the third degree to approximate human-like trajectories in smooth curves that can be easily parameterized with four control points. Two of these points are the initial and final configurations of the mannequin's root and the mid-points are initial and final configurations translated by a user-defined distance in the initial or final root's direction. A Bezier curve is computed to connect each pair of intermediate configurations in the roadmap, therefore a path is a sequence of Bezier curves that share extremal control points. For the robot, we use Reeds and Shepp curves [Reeds and Shepp 1990] that account for the nonholonomic constraints (rolling without sliding) [Laumond 1998]. The object to be manipulated moves following straight line segments in its 6-dimensional configuration space. Note that no simplification is done for the model of the object to be carried. Its shape may be as complicated as desired, that is, there is no bounding box approximation for the object.

After the roadmap is constructed, a connecting path between the initial and final configurations is searched for. If the path is found, then the function SAMPLING transforms it into a trajectory (i.e., a time parametrized path) with user-defined velocity and acceleration constraints. It is important to note that at this level only the bodies attached to the locomotion and to the object DOFs are guaranteed to be free of collisions.

Figure 7 shows an example in the 3-dimensional environment. Here, the system is composed of a virtual mannequin, a mobile manipulator, and an object. The barrier in the middle of the walk path forces the robot to take the left side, while the human can still walk on the right side (Figure 7(a)). In

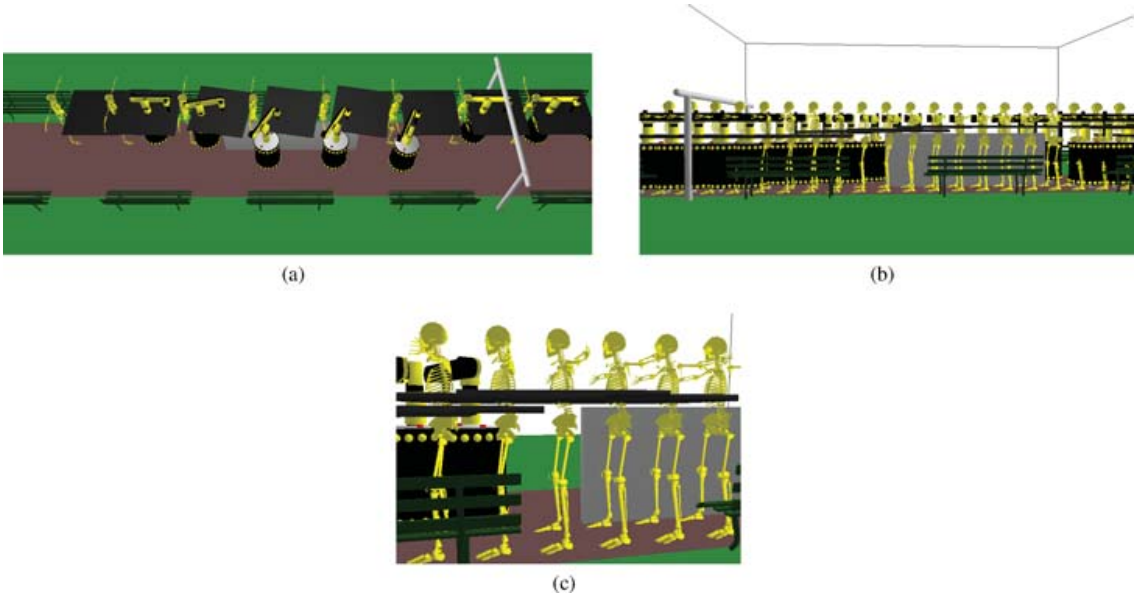


Fig. 7. (a) In the planning step, a trajectory is found for the 12-DOF system in a 3-dimensional environment. Here, the barrier in the middle of the scene causes the mannequins to take different paths. (b) Side view of the trajectory performed by the system. At this stage, locomotion and grasping behaviors are not yet synthesized. (c) The object is raised in order to avoid the barrier.

Figure 7, some of the configurations of the 12-DOF system are illustrated. It can be seen how the object is raised while traversing the barrier and lowered once it finishes. In this case, each of the characters follow the trajectory computed for their respective bounding cylinders according to their own steering methods. Note that the object remains within the reachable space between the virtual mannequin for the duration of the animation.

5.2 Behavior Control

At this stage, the trajectory for the 12-DOF simplified model is already planned. The next step is to synthesize the motions for the complete system involving the locomotion, adaptability, and grasp DOFs. This is done in the function GENERATEANIMATION by applying two different techniques: a locomotion controller to animate locomotion DOFs (pelvis and legs) and adaptability DOFs (spine and head) and an inverse kinematics algorithm adapted for each kinematic chain labeled as grasp DOFs (the mannequin's and the robot's arms).

The locomotion controller we have adopted is based on motion capture blending techniques. The result of this controller is a walking sequence for the virtual mannequin (see Section 3.2.2).

In order to synthesize the coordinated manipulation motions between the virtual characters, the IK solution for each arm is computed in order to reach the values imposed by the object configurations in the first stage (see Section 3.2.1). After applying the steps mentioned, a closed-chain mechanism is formed. As it is shown in Figure 8, two closed-loops exist. One is formed by the virtual mannequin and the object (body-arms-object) and the other by both characters (body-object-robot-floor).

Bearing in mind that the adaptability, as well as the grasp DOFs of each character could be in a collision state, a postprocessing stage is performed. In such a tuning phase, closure constraints are considered while the believability of the motions is preserved.

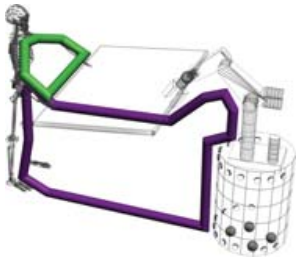


Fig. 8. In the animation phase, two closed kinematic chains are formed between the characters and the object.

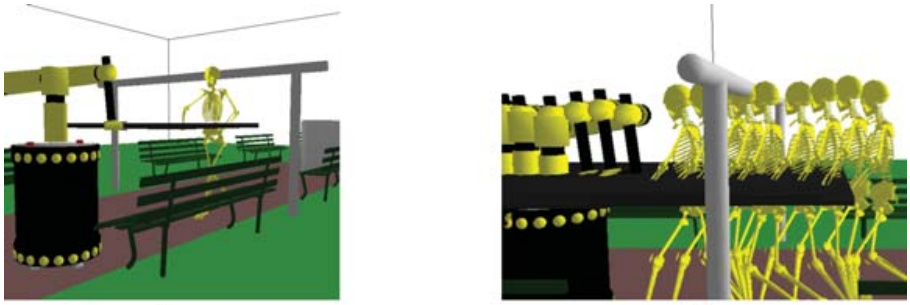


Fig. 9. (a) A configuration of the synthesized grasping and locomotion behaviors. (b) After the animation stage is performed, there can still be configurations where the upper body collides with the environment.

Figure 9(a) illustrates a configuration of the synthesized locomotion and grasping behaviors. Note that the character's head in Figure 9(b) still collides with the upper bar.

5.3 Automated Tuning

The purpose of this stage is to solve the possible residual collisions along the animated sequence. This is done by a local kinematic deformation of either the adaptability (spine-head) or grasp (arm-object) kinematic chains until a random collision-free configuration is reached. First, contiguous portions of the animation with collision in the same kinematic chain are identified and grouped into blocks. A new random configuration is generated for the colliding chain. Once the motion is modified there are three possible scenarios.

(1) The new motion contains a colliding configuration on the same kinematic chain. The random configuration is rejected and a new one is drawn.
(2) A collision for another kinematic chain is detected. In this case, the random configuration is stored as a solution for this chain.
(3) There is no collision detected. The configuration is kept and a new block is processed.

The collision-free motion generated randomly is usually not believable because of its high amplitude. To avoid this, a last step is required to progressively move the character away from the obstacle. Here we apply a warping procedure considering the two blocks, the original and the modified one. Such a procedure is typical in computer graphics to modify a sequence of key-frames when the two blocks have the same number of key-frames. The warping procedure consists in interpolating only the DOFs of the colliding chain. The parameters of the interpolation are controlled by the collision checker in

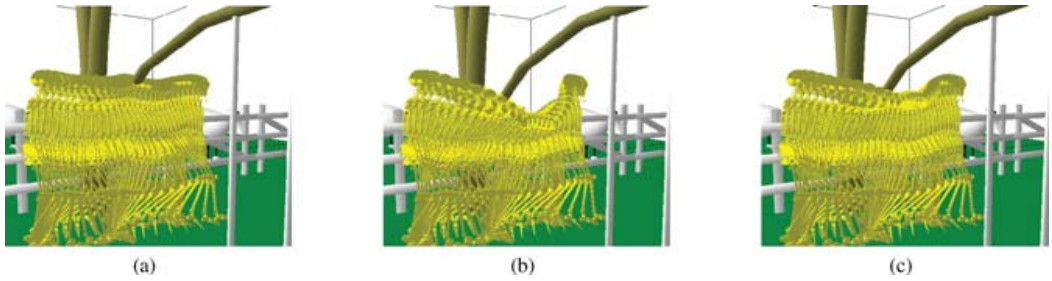


Fig. 10. (a) After synthesizing locomotion, a collision is found in the mannequin's head with the tree branch. (b) The collision is solved by finding a random collision-free configuration within the joint limits and warped with the previous animation. (c) The solution is optimized to obtain a smooth motion.

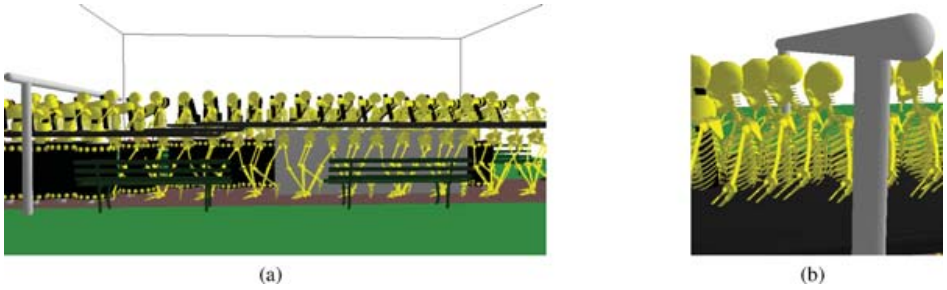


Fig. 11. (a) Some configurations of the resulting animation without residual collisions. (b) The collision with the upper bar is avoided in the tuning step.

order to provide a new configuration for the kinematic chain which is as close as possible to the original configuration while being free of collisions. This procedure is computed in function TUNING. Figure 10(a) shows a sequence where the mannequin's head collides with a tree branch. The colliding configurations are identified and a new, collision-free, solution is found as shown in Figure 10(b). Finally, the motion is optimized to preserve the eye-believability of the final animation as shown in Figure 10(c). In our barrier example, the tuning procedure was automatically applied at the end of the sequence of Figure 11(a), where the mannequin lowers its head in order to avoid the upper bar. This method has proven to work well when small deformations are needed.

If we consider the case of collisions involving the grasp DOFs, a local planner based on closed kinematic chains is used. In order to deal with multiloop closure constraints, we use the variant of RLG algorithm for multiple robots (see Section 3.1.3). For this, we consider the object DOFs as the active part of the closed kinematic chain, and the grasp DOFs as the passive part. The goal is to make the active chain reachable by all passive chains simultaneously. This is done by performing a guided-random sampling of the active chain within the intersection space formed between the reachable space of each passive chain. The configurations of passive chains are found using inverse kinematics. This procedure is illustrated in the pizza delivery example in Section 6.

5.4 Failure Recovery

After these three stages, if all configurations are not collision-free, the path generated in the first planning stage is invalidated. We remove from the roadmap the edge corresponding to the local path where the tuning step failed. Then a new global search is performed in order to find a new path.

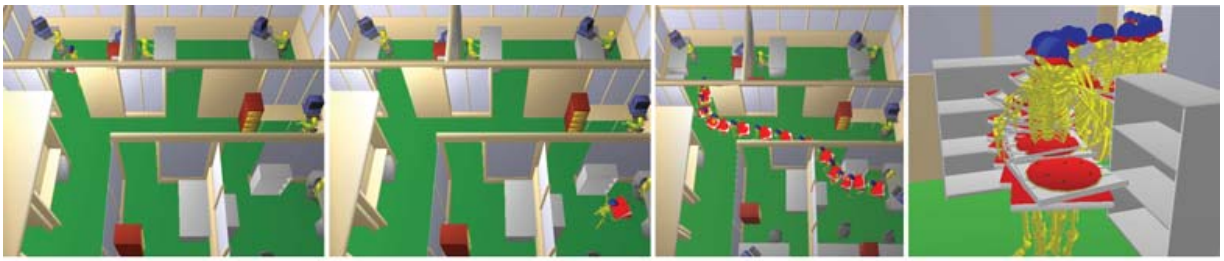


Fig. 12. (a) Initial configuration. (b) Final configuration. (c) Some configurations of the computed trajectory. (d) The mannequin moves his elbow in order to avoid collision with the bookshelf.

6. EXPERIMENTAL RESULTS

Our algorithm has been implemented within the motion planning platform Move3D [Siméon et al. 2001]. In the following paragraphs, examples of various scenarios and characters are presented.

6.1 Pizza Delivery Service

In this example, the virtual pizza delivery boy has to take a pizza box from one office to another. Here, we would like the mannequin to keep the boxes horizontal along the trajectory to prevent the pizzas from losing their toppings. For this, we impose kinematic constraints by restricting two of the six DOF of the free-flying object. This means that, in roll-pitch-yaw angle representation, we have removed the DOF allowing the object to pitch and roll.

Once the initial and final configurations (Figures 12(a) and (b)), velocity and acceleration constraints, and the number of frames are specified by the user, the algorithm is applied and the animation generated. In Figure 12(c), the resulting trajectory is shown as a sequence of configurations selected for image clarity.

After the locomotion and grasping behaviors were generated, a residual collision was found between the mannequin's arm and the bookshelf at the second office entrance. In this case, a solution was quickly found by modifying the elbow's configuration as shown in Figure 12(d). Here, this collision was not likely to be avoided in the original path because the doorways are narrow and the bookshelf is very near the final configuration.

In these examples, only four motion capture examples were used to synthesize locomotion. These capture examples included two straight lines at different speeds and two 45 degree turns, both left and right. This lack becomes evident when sharp turns are present in the path. This effect is apparent when the character turns and his feet slide on the floor. This can be corrected using more captured sources but it is interesting to note the good performance of the locomotion controller even with a low number of motion captured examples.

6.2 In the Factory

In this example, a virtual mannequin and a robot character have to transport a slab in a typical industrial environment with plenty of complex obstacles (pipes, drums, beams, ventilation units, etc.). Figure 13 shows some configurations of the resulting animation illustrating the trajectory followed by the system. As the system approaches the final configuration, the slab is turned as a result of the planning step in order to avoid the pipes.

In the animation stage, the locomotion behavior is animated and inverse kinematics is applied in order to grab the object in its new configuration. At the beginning of the trajectory, the human mannequin

Fig. 13. The characters deal with several obstacles while cooperatively transporting a slab.

Fig. 14. Some configurations extracted from the final animation.

head collides with the balcony. This collision is automatically solved in the tuning step by bending the spine and head of the mannequin. It should be noted that, if the balcony were lower, it would be very difficult, if not impossible, to find a bending collision-free configuration for the mannequin. This is due to the fact that the tuning stage is only applied to the upper part of the body and, in order to avoid a lower balcony, a knee flexion would be required. Once the obstacle is left behind, the mannequin smoothly regains its posture to continue the cooperative task until the final configuration is reached.

Figure 14 shows a set of frames extracted from the final animation. Here, we can see that the motion of the system remains plausible after applying the three steps in the algorithm. Note that the trajectory planned for each of the agents in the scene remains collision-free and their position close enough to keep holding the slab.

6.3 Buren's Columns

This example in our version of the Buren's Columns involves two virtual mannequins that handle a large plate. Here, we have introduced kinematic constraints on the plate in order to keep it horizontal. Because of this, a 10-DOF reduced model is considered in the planning step. Figure 15 shows a view of the complexity of the environment and the system.

Figure 16 illustrates some frames from the resulting animation. The virtual mannequins walk along the computed trajectory carrying their plate. The object configuration is then modified because of the increasing height of the columns. In the third frame, an upper view of the system shows that the computed trajectory for the plate is collision-free. The mannequins continue to raise their object until it is safe to put it down. The final two frames show that one of the agents collides with the bar. This collision is solved in the tuning step by bending the mannequins spine.

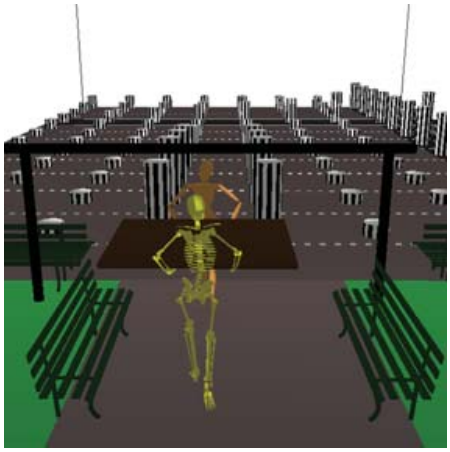


Fig. 15. In our version of Buren's Columns, two virtual mannequins interact to handle a plate.

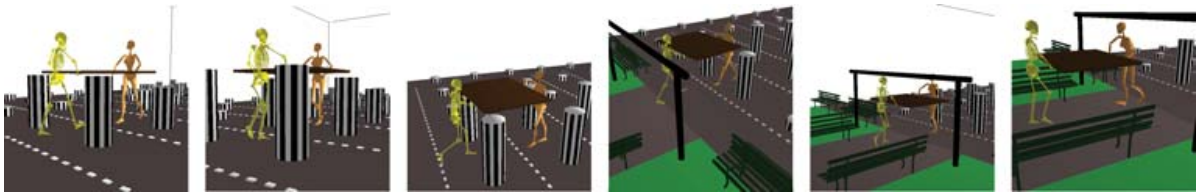


Fig. 16. Selected frames of the resulting animation.

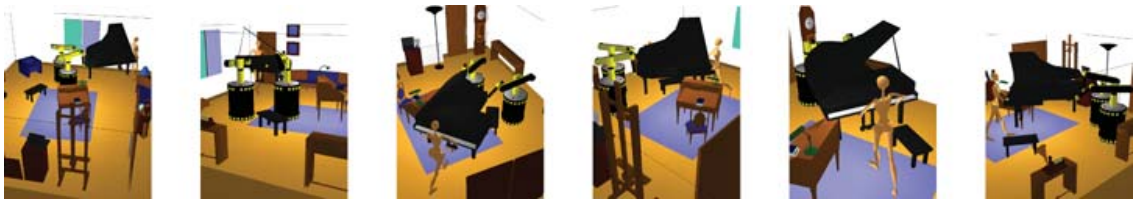


Fig. 17. Two mobile manipulators and a virtual mannequin cooperating to transport a piano.

6.4 Transporting the Piano

In our last example, we treat a version of the piano mover's problem with two mobile manipulators and a virtual mannequin. Here, we want to transport the grand piano in our living room environment. The livingroom is small and contains large objects (tables, a desk, etc.), the walkways are therefore narrow and collisions are likely to occur.

A collision-free path for the 15-DOF reduced system was found between the desk and the piano chair. Figure 17 illustrates some of the synthesized motions for the mobile manipulators and the human mannequin. The virtual robots move away from each other to avoid the stool and then regain their posture.

The cooperation among the characters is ensured during the animation. The 3-dimensionality of the resulting animation is mainly seen in the piano configuration. Figure 18 shows sequenced frames of the planned animation in order to avoid collision between the piano and the desk. In this example, no residual collisions were found.

Fig. 18. The piano should be raised in order to avoid collision with the desk.

Table I. Model Complexity (Number of Polygons)

	Environment	System
Office		
- Complete	148,977	17,239
Factory		
- Complete	159,698	18,347
- Col.Test	92,787	
Buren's Columns		
- Complete	44,392	24,837
Living Room		
- Complete	19,077	16,210

Table II. Office (time in seconds)

	In the Office
No. Frames	308 609
Stages	
I. Planner	
- Path	0.5–0.5
- Trajectory	2.0–3.8
II. Animation	
- Locomotion	0.8–1.6
- Manipulation	0.3–0.5
III. Residual Col.	0.2–0.4
Total	3.8–6.80

6.5 Computational Time

The planner has been tested on a workstation Sun-Blade-100 with a 500MHz UltraSparc-IIe processor and 512MB RAM. In Table I, the number of polygons in the different environments as well as the polygons in the system are presented. In the industrial environment, we have identified the polygons that participate in the collision test (i.e., the ones below the mannequin's head level). In the columns and in the living-room environment, all polygons are considered for collision tests purposes.

The required time to compute the examples presented are given in Tables II, III, IV and V (averaged over 100 runs). Here, the results of the planning step are expressed considering a precomputed graph of the environment product of the learning phase. The time taken to build this roadmap was 1.69 seconds, 31.4 seconds, 15.1 seconds and 3.2 seconds for the office, the factory, the columns and the living room, respectively. The time it takes to compute such graphs varies with the complexity of the environment and the fact that it is a probabilistic approach. The tables present only the results of the queries. Two different animations were generated for each trajectory, the second doubling the number of frames.

Table III. Factory (time in seconds)

	In the Factory
No. Frames	268 530
Stages	
I. Planner	
- Path	6.5–6.5
- Trajectory	2.1–4.5
II. Animation	
- Locomotion	0.8–1.6
- Manipulation	0.4–0.8
III. Residual Col.	5.7–11.4
Total	15.5–24.8

Table IV. Columns (time in seconds)

	Buren's Columns
No. Frames	204 405
Stages	
I. Planner	
- Path	0.01–0.01
- Trajectory	2.8–6.1
II. Animation	
- Locomotion	0.6–1.1
- Manipulation	0.6–1.3
III. Residual Col.	3.1–5.6
Total	7.11–14.11

Table V. Living Room (time in seconds)

	Transporting the Piano
No. Frames	78 151
Stages	
I. Planner	
- Path	3.3–3.3
- Trajectory	0.6–1.3
II. Animation	
- Locomotion	0.2–0.5
- Manipulation	0.1–0.3
III. Residual Col.	0.0–0.0
Total	4.2–5.4

Given the significantly higher size and complexity of the industrial environment, it is not surprising that it took more time to compute a collision-free path than the other examples. Note that the time it takes to compute the path at the planning stage is independent of the number of frames in the animation.

In the animation step, the fastest in the algorithm, it is clear that computational time increases proportionally with the number of frames.

The tuning step relies heavily on the complexity of the environment but also on the number of frames where there is a colliding configuration. Note that in the living-room example, the time it takes to tune the animation is zero because residual collisions were not found.

6.6 Videos

Animated sequences of the examples presented in this work can be found at: `http://www.laas.fr/Gepetto/motion-character`.

7. CONCLUSIONS AND FUTURE WORK

We presented an approach to plan and synthesize collision-free motions for virtual mannequins handling a bulky object in a 3-dimensional environment. To accomplish this coordinated task, a geometric and kinematic decoupling of the system is proposed. This decomposition enables us to plan collision-free paths for a reduced system, then to animate the locomotion and grasping behaviors in parallel and finally to clean up the animation from residual collisions. These three steps are applied consecutively by making use of different techniques such as motion planning algorithms, locomotion controllers, inverse kinematics techniques, and path planning for closed-kinematic mechanisms.

At this stage, behavior synthesis is based on the functional decomposition of the system's DOFs (locomotion, grasping, adaptability). The advantage of such decomposition is that the complexity of the system is reduced allowing, for instance, the use of very fast IK algorithms. This decomposition also greatly simplifies the combination of behaviors independently generated. The main limitation of this approach is that the motion planner is not as flexible as it could be. For instance, the characters can not bend to pick the object up from the floor because the grasping DOFs cannot modify the locomotion DOFs. This problem can be solved by using approaches that take into account the whole character's body, such as in Yamane et al. [2004] and Kallman et al. [2003], but at the cost of performance.

As future work, we intend to improve the believability of the resulting animation taking into account the forces working in the environment from the planning stage itself. In this work, we have seen that generated motions are not convincing when the characters handle heavy objects as in the Piano example. Our challenge is thus to combine motion-capture-based control with a physical model of the mannequin. This should be done while preserving as much as possible the current performance of the method.

ACKNOWLEDGMENTS

We thank Juan Cortés for his help on the implementation of the RLG algorithm.

REFERENCES

BADLER, N., ERIGNAC, C., AND LIU, Y. 2002. Virtual humans for validating maintenance procedures. *Commun. ACM 45*, 7, 56–63.

BADLER, N. I., PHILLIPS, C., AND WEBBER, B. 1993. *Simulating Humans: Computer Graphics Animation and Control*. Oxford University Press, Inc., New York, NY.

BAERLOCHER, P. AND BOULIC, R. 2004. An inverse kinematics architecture enforcing and arbitrary number of strict priority levels. *Visual Comput. 20*, 402–417.

BLUMBERG, B. AND GALYEAN, T. 1995. Multi-level direction of autonomous creatures for real-time virtual environments. *Comput. Graph. 29* (Annual Conference Series), 47–54.

BRAND, M. AND HERTZMANN, A. 2000. Style machines. In *Proceedings of SIGGRAPH*. ACM Press/Addison-Wesley Publishing Co., 183–192.

CHOI, M., LEE, J., AND SHIN, S. 2003. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Trans. Graph. 22*, 2, 182–203.

CORTÉS, J. AND SIMÉON, T. 2003. Probabilistic motion planning for parallel mechanisms. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

EARNSHAW, R., MAGNENAT-THALMANN, N., TERZOPOULOS, D., AND THALMANN, D. 1998. Guest editors' introduction: Computer animation for virtual humans. *IEEE Comput. Graph. App. 18*, 5, 20–23.

FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 2001. Composable controllers for physics-based character animation. In *Proceedings of SIGGRAPH*. ACM Press, 251–260.

HAN, L. AND AMATO, N. 2000. A kinematics-based probabilistic roadmap method for closed chain systems. In *Proceedings of the International Workshop on Algorithmic Foundations of Robotics (WAFR)*.

KALLMAN, M., AUBEL, A., ABACI, T., AND THALMANN, D. 2003. Planning collision-free reaching motions for interactive object manipulation and grasping. In *Proceedings of ACM SIGGRAPH/Eurographics*, P. Brunet and D. Fellner, Eds. Vol. 22. Eurographics Association.

KALLMANN, M., BARGMANN, R., AND MATARIC, M. 2004. Planning the sequencing of movement primitives. In *Proceedings of the International Conference on Simulation of Adaptive Behavior (SAB)*. Los Angeles, CA.

KAVRAKI, L. E., SVESTKA, P., LATOMBE, J.-C., AND OVERMARS, M. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robotics and Automat. 12*, 4, 566–580.

KOGA, Y., KONDO, K., KUFFNER, J., AND LATOMBE, J.-C. 1994. Planning motions with intentions. In *Proceedings of SIGGRAPH*. ACM Press, New York, NY, 395–408.

KONDO, K. 1991. Inverse kinematics of a human arm. *J. Robotics Syst. 8*, 2, 115–175.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. In *Proceedings of SIGGRAPH*. ACM Press, New York, NY, 473–482.

KUFFNER, J. 1998. Goal-directed navigation for animated characters using real-time path planning and control. Lecture Notes in Computer Science vol. 1537, 171–186.

LAMIRAUX, F. AND LAUMOND, J.-P. 1997. From paths to trajectories for multi-body mobile robots. In *Proceedings of the 5th International Symposium on Experimental Robotics (ISER)*. Barcelona, Spain, 237–245.

LATOMBE, J.-C. 1991. *Robot Motion Planning*. Kluwer Academic Press, Boston, MA.

LAUMOND, J.-P., Ed. 1998. *Robot Motion Planning and Control*. Springer-Verlag.

LAVALLE, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning. Tech. rep., Computer Science Department. Iowa State University. (Oct).

LAVALLE, S. M., YAKEY, J., AND KAVRAKI, L. E. 1999. A probabilistic roadmap approach for systems with closed kinematic chains. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

LIU, Y. AND BADLER, N. I. 2003. Real-time reach planning for animated characters using hardware acceleration. In *Proceedings of the International Conference on Computer Animation and Social Agents (CASA)*. Washington, DC, IEEE Computer Society, 86.

PARENT, R. 2001. *Computer Animation: Algorithms and Techniques*. Morgan-Kaufmann Publishers.

PERLIN, K. 1995. Real time responsive animation with personality. *IEEE Trans. Visualiz. Comput. Graph. 1*, 1, 5–15.

PETTRÉ, J. AND LAUMOND, J.-P. 2005. A motion capture-based control-space approach for walking mannequins. *Comput. Animat. Virtual Worlds* 16, 1–18.

PETTRÉ, J., LAUMOND, J.-P., AND SIMEÓN, T. 2003. A 2-stages locomotion planner for digital actors. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*. San Diego, CA, 258–264.

REEDS, J. AND SHEPP, R. 1990. Optimal paths for a car that goes both forward and backwards. *Pacific J. Mathemat. 145*, 2, 367–393.

RENAUD, M. 2000. A simplified inverse kinematic model calculation method for all 6r type manipulators. In *Proceedings of the International Conference in Mechanical Design and Production*. 15–25.

ROSE, C., COHEN, M., AND BODENHEIMER, B. 1998. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Comput. Graph. Appli. 18*, 5, 32–41.

SHILLER, Z., YAMANE, K., AND NAKAMURA, Y. 2001. Planning motion patterns of human figures using a multi-layered grid and the dynamics filter. In *Proceedins of the IEEE International Conference on Robotics and Automation (ICRA)*.

SIMÉON, T., LAUMOND, J.-P., AND LAMIRAUX, F. 2001. Move3d: A generic platform for motion planning. In *Proceedings of the 4th International Symposium on Assembly and Task Planning (ISATP)*.

SIMÉON, T., LAUMOND, J.-P., AND NISSOUX, C. 2000. Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics 14*, 6.

TOLANI, D., GOSWAMI, A., AND BADLER, N. I. 2000. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical Models 62*, 353–388.

UNUMA, M., ANJYO, K., AND TAKEUCHI, R. 1995. Fourier principles for emotion-based human figure animation. In *Proceedings of SIGGRAPH*. ACM Press, 91–96.

WITKIN, A. AND POPOVIC, Z. 1995. Motion warping. In *Proceedings of SIGGRAPH'95*. ACM Press, 105–108.

YAMANE, K., KUFFNER, J., AND HODGINS, J. K. 2004. Synthesizing animations of human manipulation tasks. In *Proceedings of SIGGRAPH*. ACM Press, New York, NY.

YAMANE, K. AND NAKAMURA, Y. 2003. Natural motion animation through constraining and deconstraining at will. *IEEE Trans. Visualiz. Comput. Graph. 9*, 3, 352–360.

ZHAO, J. AND BADLER, N. I. 1994. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Trans. Graph. 14*, 4.

Received October 2004; revised September, October 2005; accepted November 2005

8.4 Crowds of Moving Objects: Navigation Planning and Simulation

Crowds of Moving Objects: Navigation Planning and Simulation

Julien Pettré, Helena Grillon and Daniel Thalmann

Abstract— **This paper presents a solution to interactive navigation planning and real-time simulation of a very large number of entities moving in a virtual environment. From the environment geometry analysis, we deduce a structure called *navigation graph*, which is the base to our method. After the description of this structure, we introduce a set of algorithms dedicated to answer navigation queries with a set of various solution paths and to execute the planned navigation in an efficient manner. We equally demonstrate method performance and robustness over several examples.**

I. INTRODUCTION

Robotics has made great efforts to develop motion planning methods in order to allow mechanical systems to navigate autonomously in their environment. Recently, the application field of these methods significantly expanded, such as to Biochemistry, Architecture, Ergonomics, and Computer Graphics (CG). For CG applications, motion planning increases the autonomy of digital actors, eases a user's navigation in a virtual world and solves user queries from high level directives (e.g., commanding an army in a video game) among other functionalities. A specificity of CG applications is a frequent need for interactivity: performance and robustness are main issues. We principally consider the specific case of a large number of moving entities evolving on the terrain of a given virtual environment. We have developed an architecture able to solve users' navigation queries interactively, and to update the position of each entity in real time.

A number of criteria and objectives have conditioned our technical choices. First of all, *performance*: computer power increase has doubtlessly improved computational times to those of previous approaches. However, the complexity of environments' geometric models and the number of moving entities in them has increased even more. Our method allows the abstraction of environment geometries. Then, *transferability*: our method is applicable to a large class of environments and moving entities. It is based on simple geometrical expressions and properties. It is thus easily adaptable. Finally, *scalability*: our architecture is scalable. In other words, the user can distribute the computing resources locally in space and time during simulations, and focus them where most needed.

J. Pettré performed this work at EPFL-VRlab, sponsored by Marie Curie Action, grant "RAGA" n.11166 FP6-2004-Mobility-5

J. Pettré is with IRISA-INRIA, Bunraku Team, Campus de Beaulieu, F-35042 Rennes, France julien.pettre@irisa.fr

H. Grillon and D. Thalmann are with EPFL-VRlab, CH-1015 Lausanne, Switzerland {helena.grillon, daniel.thalmann}@epfl.ch

Our method's contributions are, first, a cell-decomposition method which captures the environment's topology and geometry and is adapted to environments combining uneven terrains and multi-layered surfaces. Second, it proposes a data structure designed for both navigation planning and simulation. Third, it offers a specific navigation planning technique which searches for various solutions to a single problem. Finally, it proposes a set of algorithms to respond to user queries such as navigation, neighbor or visibility requests.

II. RELATED WORK

Navigation planning is a specific case of motion planning, [1], [2], [3], which is mainly studied by the Robotics community. However, we will once again focus on their application or development by the CG community [4]. Basically, three main classes of solutions to the motion planning problem can be distinguished: local approaches, probabilistic ones and deterministic ones.

Local approaches fit the problem of collision avoidance between two moving entities: the potential fields method [5] is very popular and has inspired many solutions [6]. However, when used for global path planning, they generate many problems: parameters tuning (naturalness vs. collision freeness), goal conflicts and local minima. ***Probabilistic approaches*** randomly explore the free configuration space [7] of a given problem, while capturing its connectivity into a roadmap. Multi-query (PRM) [8] or single query (RRT) [9] solutions exist, as well as many variants. These have been successfully applied to computer animation problems such as to the locomotion of human-like characters [10], [11], [12], [13]. Such approaches fit high-dimensional problems, with systems having numerous degrees of freedom, whilst the navigation problem is frequently lowered to a 3 or 4-dimensional problem (position and orientation of the system). Paths synthesized by PRMs require optimization which dramatically increases the solution's computational cost. Moreover, when dealing with multiple moving entities, PRM solutions tend to always produce similar paths (the ones captured by the roadmap), which increases the potential number of collisions between entities and may have an impact on path naturalness. ***Deterministic approaches*** use an exact and continuous representation of the space and of the considered system mobility. This generally results in a very complex system, making them unenforceable. Environments are therefore discretized to overpass these limitations. The use of a grid laying on the floor to capture obstacle position and navigable space is the most frequent solution. A* or Dijkstra's algorithms are used to search for solution paths [14], [15]. To optimize

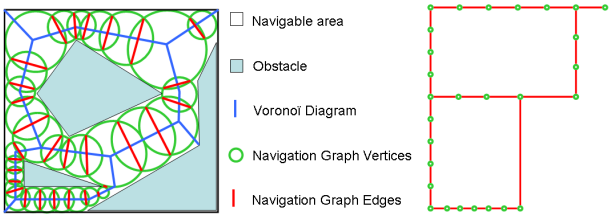


Fig. 1. *left:* Geometric representation of a $\mathcal{NG}$ in a 2D academic environment. *right:* $\mathcal{NG}$ itself for the same example.

search times, multi-resolution or hierarchical grids can be used [16], [17]. A classical drawback to these solutions is the low aspect quality of the results: jagged paths and sharp corners. A solution consists in post-processing the solution paths, which increases computation time. Our approach falls into the deterministic class by decomposing the environment into a set of 3D navigable cells. However, we compute this decomposition from a discrete space representation. Decomposition methods have already been used for crowd simulation, like in [18], and proved to be efficient. Our method improves such approaches by considering a larger class of terrains (uneven and/or multi-layered), by extending data structures to efficient navigation, neighborhood and visibility queries, and by providing scalable simulations.

III. NAVIGATION GRAPHS

A. Principle

A Navigation Graph ($\mathcal{NG}$) results from a cell decomposition of the navigable parts $\mathcal{C}_{nav}$ of a considered environment. As in classical decomposition methods, $\mathcal{NG}$ vertices are cells free of obstacle and adjacent cells are linked by an edge. The decomposition is not exact as the computation is based on a discrete representation of the obstacles (see next section). The method is dedicated to entities moving on a floor or terrain. $\mathcal{C}_{nav}$ is thus the union of the environment surfaces:

- flat enough: for a surface to be navigable, its slope must be within the bounds of a user defined limit angle,
- free of obstacle,
- with a high enough free-space above them - still according to considered entities characteristics.

Cells are cylinders laying on $\mathcal{C}_{nav}$ and are the $\mathcal{NG}$ vertices. Cells are adjacent when the corresponding cylinders overlap. Adjacency is naturally modeled by the $\mathcal{NG}$ edges. In order to capture $\mathcal{C}_{nav}$ in a compact manner, cylinders are centered on the Voronoï diagram [19] of the navigable surfaces and their center is excluded from any other cell. As for others decomposition-based techniques, a property of $\mathcal{NG}$ is that any point belonging to a given navigable cell can reach any point belonging to an adjacent cell, passing by any point of their intersection (depending on conditions). Thus, the navigation planning is reduced to a graph search problem.

Fig. 1 schematically illustrates a $\mathcal{NG}$ computed for a simple 2D environment. However, a novelty of our method is to fit 3D environments, even for those combining uneven and multi-layered surfaces.

B. $\mathcal{NG}$ computation

We previously introduced a technique to compute $\mathcal{NG}$s [20] using an intermediate grid and graphics hardware-based operators. The method consists in 5 main steps:

1) *environment geometry sampling*: we create a regular grid of points matching the environment surfaces. As multi-layered environments may be considered, a simple elevation is insufficient since several elevations may correspond to given horizontal coordinates.
2) *grid mesh*: two neighboring grid points are interconnected when the slope of the in-between space is beneath the user-defined maximum slope angle and free of obstacle. With this stage, we provide a mesh capturing $\mathcal{C}_{nav}$ in a discrete manner.
3) *clearance map*: we compute the clearance for each grid point, i.e. the distance to the nearest obstacle or high-slope. Given the the grid mesh computation method, this distance is approximated by the distance to the nearest grid point not connected to all of its direct neighbors. Indeed, The lack of connection reveals the presence of an obstacle or a high-slope.
4) $\mathcal{NG}$ *deduction*: we then use a subset of grid points to compute the $\mathcal{NG}$ vertices. A graph vertex (cylinder) is created from a grid point by using it as centre and its corresponding clearance as radius. The grid point with maximum clearance is selected to create the graph vertex and all the grid points covered by the vertex are disabled for selection. The process is then reiterated until no more grid points remain for selection. Doing so, a majority of cylinders are centered on the Voronoï diagram corresponding to the environment.
5) *visibility pre-computation*: for each $\mathcal{NG}$ vertex, we compute the visibility of all other vertices according to the four main cardinal points. We can then use this information for visibility queries between moving entities.

Some examples of environments and corresponding $\mathcal{NG}$, as well as computing times, are given in the Results section.

C. Data Structures

In order to allow the implementation of our method, and for a better understanding of the algorithms we use, we here detail the contents of our data structures. $\mathcal{NG}$ captures the following information:

1) For each vertex. *Navigable cylinder geometry*: center, radius and height. *Local elevation map*: the portion of the intermediate grid - used for $\mathcal{NG}$ computation - located under the cylinder is copied and stored. As $\mathcal{NG}$ handles multi-layered environments, this allows to solve elevation queries efficiently. *Visible vertices*: the list of all vertices that can be partially or totally seen from the current vertex. This allows to solve visibility queries efficiently. *Included moving entities*: the list of all moving entities currently navigating in the vertex. This information is managed and updated by the simulation loop. *Adjacent vertices*: the list of

vertices sharing an edge with the current vertex. This allows to solve neighbor queries efficiently.

2) For each edge. *Linked vertices*: the pair of vertices sharing this edge. *Gate geometry*: the coordinates of a line segment at the linked cylinders' intersection. *Cost*: the distance between the two linked cylinders' centers.
3) For each mobile entity. *Steering methods*: able to steer the mobile entity toward a way-point, whilst avoiding other mobile entities. Optionally, we can scale the steering method, i.e., we can change its complexity and precision with a parameter. This will be discussed in the Simulation part of the next section. *Currently crossed vertex*: in order to know in which navigable area the mobile entity is currently located. This information is managed by the simulation loop. *Currently followed path*: resulting from a navigation query. *Currently followed way-point*: which is selected along the currently followed path.

D. Discussion

As mentioned before, our method is inspired by cell-decomposition motion planning techniques. However, our method is not perfectly accurate: the decomposition does not capture the complete navigable space, and the graph computation uses a discrete representation of the environment. A high-precision grid may therefore be required in order to capture narrow passages, which results in long comuptations. Nevertheless, advantages of our method are its robustness (we have tested it on many environments whose meshes were crude from design), its ability to consider both uneven and multi-layered environments, and the lack of need for expertise to use it.

Another advantage of our method is its use of very simple geometric expressions to model the navigable space: cylinders and line segments for the graph vertices and edges respectively. This representation is not perfect in all cases: for example, long corridors require many adjacent cylinders to be captured. However, our model allows to solve basic queries efficiently. For example, a simple distance test allows to determine if a moving entity is contained in a vertex or not. It equally requires little memory for storage.

The next section illustrates the use of $\mathcal{NG}$ for crowd navigation planning.

IV. NAVIGATIO AND OTHER INTERACTIVE QUERIES

A. Navigation Planning Queries

We use 2 navigation planning algorithms, one to plan the navigation of a single entity between two locations (Alg. 1) and one to create a navigation flow between two locations, i.e., for a large number of entities all moving between identical locations (Alg. 2).

We solve Single Navigation Queries (Alg. 1) in a classical manner: given a navigation graph $\mathcal{NG}$, and two locations to join (for which the corresponding $\mathcal{NG}$ vertices v_i and v_d are found), we launch a Dijkstra's shortest path search. We deduce the resulting path from the set of edges between both

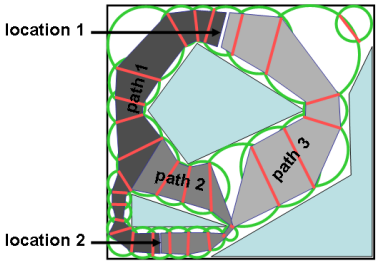


Fig. 2. Solution paths to a navigation flow query between locations 1 and 2: the proposed algorithm provides several solutions to a single query. The objective is to dispatch many entities moving between identical locations. Each entity can select its own path, and use the full width of the corridors composing the path to obtain a unique trajectory.

Algorithm 1: Single Navigation Query

Data: current location p_{init}, destination p_{dest} and $\mathcal{NG}$
Result: a solution path (set of edges)
$P_{sol} = \{e_1, ..., e_n\}$

```
begin
    v_i ← the vertex including p_init
    v_d ← the vertex including p_dest
    P_sol ← Dijkstra(v_i, v_d, NG)
end
```

locations. As edges are line segments (we also call them *gates*), we can model the path as a corridor between the two desired locations. An example of Navigation Query is illustrated in Fig. 2. This first algorithm provides *path 1* as unique solution.

Another problem is to compute a trajectory for a moving entity which is always confined to the free space. For example, if way-points are picked within each successive gate and a linear steering is used to join them, the trajectory is contained in the solution path (the solution corridor). However, in the case of mechanical constraints (e.g. non-holonomy), the user must take care of this issue.

Our objective is to address the problem of numerous entities navigating in a same environment and particularly, the case were many of them navigate between identical destinations. In this case we create a Navigation Flow between the two locations. We avoid the concentration of all entities on a same trajectory in order to limit the potential number of inter-collisions (solving interactions also has a high computational cost) and increase realism. In order to do this, we can exploit corridor width to dispatch the entities. However, this can be insufficient, especially when the gates are narrow: a congestion may appear. The second navigation planning algorithm Alg. 2 is aimed at providing a second level of variety by answering queries with a set of paths instead of a single one. As in Alg. 1, the shortest path is our first solution path. We then search for an alternative path avoiding the narrowest gate: the gate's cost is increased and Dijkstra's search is invoked again. Edge cost can only be modified once and the algorithm stops when no more edge cost can be modified. Optionally, the process stops when a

Algorithm 2: Navigation Flow Query

Data: current position p_{init}, destination p_{dest}, $\mathcal{NG}$ and optionally a max. number of paths to find N_{max}
Result: a set of solution paths $F_{sol} = \{P_{sol_1}, ..., P_{sol_n}\}$

```
1  begin
2      v_i ← the vertex including p_init
3      v_d ← the vertex including p_dest
4      E_inc ← {∅}
5      Stop ← false
6      while Stop is false and card(F_sol) < N_max do
7          Stop ← true
8          P_sol ← Dijkstra(v_i, v_d, NG)
9          if P_sol ∉ F_sol then
10             F_sol ← F_sol ⋃ {P_sol}
11         if
           ∃e \ e ← Thinnest({e | e ∈ P_sol ∧ e ∉ E_inc})
           then
12             e_cost = e_cost × 10
13             E_inc ← E_inc ⋃ {e}
14             Stop ← false
15 end
```

Algorithm 3: Simulation Loop

Data: simulation initialized
Result: updated situation

```
1  begin
2      ComputeLoS
3      for all vertex V ∈ NG do
4          if UpdateRequired then
5              for all MovingEntity M∈ V do
6                  SteeringMethod
7                  if WayPointReached then
8                      if EndOfPath then
9                          GoBackward
10                         ChooseCurrentBestPath
11                     ComputeNewWayPoint
12                     MoveToNextVertex
13     UpdatePathsCosts
14     RenderScene
15 end
```

number of solution paths is reached. Note that we developed a variant stop criterion where the relative path lengths are used: the algorithms stop when the last found path length is a times longer than the first and shortest one, $a \in [1..\infty]$.

At the end of the planning stage, the data structure is completed: the entities are placed at their initial location and each vertex's list of *included moving entities* is setup accordingly. The *currently crossed vertex, followed path and way-point* are listed for each entity as well. The real-time simulation, i.e., the iterative udpate of the situation for each entity according to the results of the planning stage then starts.

B. Simulation

The objectives of our simulation loop are specific. We want the highest performance possible in order to allow real-time updates ($25Hz$ at least) along with visualization. In our case, we look for believability. A spectator observing the scene with moving entities should be presented with a simulation of the best quality level on the foreground. For background areas, we want the entities to achieve their goal, but assumptions and simplifications are permitted.

Algorithm 3 is looped to update the simulation. Whereas classical solutions loop over each moving entity, a specificity of our simulation is to consider each area successively. These areas are delimited by the $\mathcal{NG}$ vertices. Before update (line 2), Levels of Simulation (LoS) are computed. A LoS is a score assigned to each $\mathcal{NG}$ vertex which depends on the the point of view location (view centrality and distance). Vertices having a high LoS are close to the point of view and located in the central part of the screen. Vertices having low LoS are far from the point of view, on the borders of the screen or even invisible.

When a vertex is visited, update is required or not (line 4), depending on the LoS. If the LoS is high, update is required frequently ($25Hz$). On the contrary, if the LoS is low, the udpate is done at lower frequencies (from 1 to $15Hz$).

When update is required, the position of each moving entity included in the vertex is updated. Once again, update quality depends on the LoS (line 6). For low LoS, entities are steered in a simplified manner: we use linear steering and do not consider collisions between entities since we assume they are not detectable at a far distance and even less in invisible areas. For high LoS, we steer entities using smooth trajectories and velocity accelerations. We equally take into account inter-collisions by using Reynolds' [21] steering method.

Steering methods require way-points to lead the entities along the followed corridors. For a given entity, once the tracked way-point is reached, a new one is picked within the next gate to be crossed (line 7). We use an individual parameter to compute the way-point (line 11), so that the entity crosses a gate always on the same side (the parameter continuously changes from 0 to 1 while the way-point position in the gate line segment continuously moves from left to right). The moment a gate has just been crossed also corresponds to a change of area for the entity. The vertices *included moving entities* lists are updated accordingly (line 12), as well as the other entity-related variables. Doing so, we always know where a given entity is, and which entities are in a given place. This consists in crucial information in order to solve interactive queries, presented in the next section.

If the end of the path is reached, the entity turns around and is sent back to its original location. Entities go back and forth indefinitely. However, new goals can be assigned

interactively. A different path can be assigned to the entity, within the set of solutions provided by Alg. 2. The best solution is not necessarily the shortest one. We consider the current occupation of each path to compute the best solution. We take into account both the distance to a given edge and the local population density to compute an average travel time. The lowest travel time path is selected and assigned to the entity. The paths' travel times are recomputed at the end of the update loop (line 13) if necessary: as it is not a highly dynamic variable, it can be updated at low rates. Finally, the situation is rendered to the screen, however, this is not this paper's main issue.

C. Neighbor and Visibility Queries

During the simulation we use neighbor queries in order to solve collisions between moving entities. Alg. 4 allows to compute the list of moving entities potentially in contact with a considered entity E. As each vertex stores the list of entities currently included in it, the complete list can be computed with a limited number of distance tests. Note that adjacent vertices are visited since they may contain close enough entities. Recursively, the search may be extended to other linked vertices, at a deeper level, if neighboring entities at a farther distance are to be searched for.

Algorithm 4: Neighbor Query

```
   Data: an entity E
   Result: list of entities L_neighb potentially in collision
1  begin
2      V_E ← E :: current vertex
3      for all entity e_i ∈ V_E :: included entities do
4          if DistanceTest(E, e_i) then
5              Add e_i to L_neighb
6      for all vertex v_i ∈ V_E :: linked vertices do
7          for all entity e_i ∈ v_i :: included entities do
8              if DistanceTest(E, e_i) then
9                  Add e_i to L_neighb
10 end
```

Each $\mathcal{NG}$ vertex equally refers to a list of visible areas. A list of visible entities to a given entity E can be computed with a limited number of tests (Alg. 5). Such queries may remain too complex to be solved interactively in the case of complex environments and high density areas. Indeed, visibility tests (line 5) complexity depend on the number of triangles composing the environment. Tests may be done by casting rays (using a collision checker) or by occlusion tests using OpenGL extensions.

V. Results

We have applied our technique to crowds of virtual pedestrians. Fig. 3 illustrates some of the tested environments, with snapshots of the computed $\mathcal{NG}$. Outcomes to Navigation Queries are also shown. The Stonehenge-like environment is representative: the presence of pillars in the middle of

Algorithm 5: Visibility Query

```
   Data: an entity E
   Result: the list of all visible entities L_vis
1  begin
2      V_E ← E :: current vertex
3      for all vertex v_i ∈ V_E :: visible vertices do
4          for all entity e_i ∈ v_i :: included entities do
5              if VisibilityTest(E, e_i) then
6                  Add e_i to L_vis
7  end
```

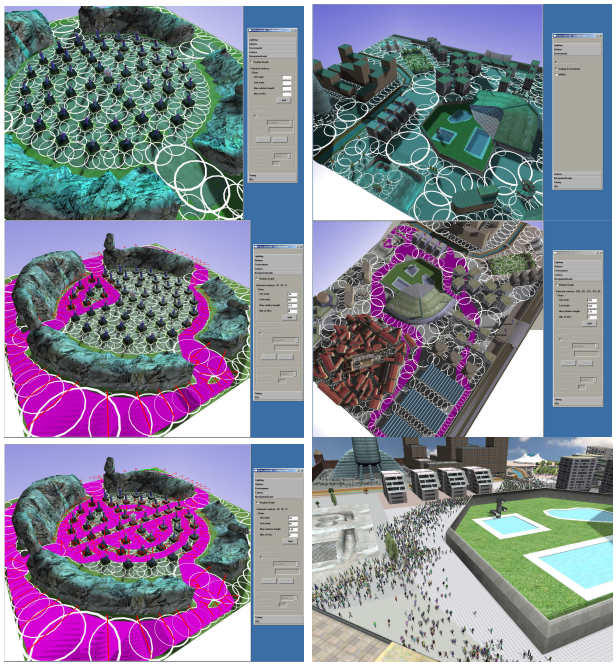


Fig. 3. *left column:* A Stonehenge-like environment and a set of paths solution to a Navigation Flow query with varying maximum relative path length values (Alg. 2). *right column:* A virtual city, a set of paths solution to a Navigation Flow query and a crowd navigating in the city.

the scene creates many congestion points. In such a case, it is important to obtain alternative solutions to a navigation flow query. Indeed, many entities navigating along a single solution path would result in congestions whereas parts of the environment close to the pedestrians would remain empty. This would seem very unrealistic to a spectator. Figure 3 shows solutions to a query for which we have limited the number of solutions (limitation is stronger in the middle image). Without limitation, the union of solution paths covers the whole environment, as shown in the accompanying video. The accelerated part of the video illustrates a real-time simulation of 1000 pedestrians all navigating between identical locations, but dispatched according to the distribution done line 10 of the Algorithm 3. In the virtual city environment, our simulator is able to reach a 35'000 pedestrians crowd with interactive rates (10-20Hz, including rendering tasks, according to the point of view). The obtained performance

is possible thanks to the scalable simulation and rendering: at the forefront, complex articulated characters are rendered whereas in the background simplified representations of humans are used. The crowd is dispatched on the whole city using 7 Navigation Flows of 5'000 pedestrians each, joining main buildings (hotel, church, train station, circus, etc.). Each navigation flow is created in a second (the corresponding $\mathcal{NG}$ is made of 1'500 vertices), which allows interactive crowd setup. The environment geometries are complex: 10'000 and 100'000 triangles approximately for the Stonehenge-like and the City examples respectively. $\mathcal{NG}$ allows to abstract the geometries of the environment and the complexity of both our planning and simulation. Complexity then becomes mainly dependent on the number of vertices and edges composing the graph.

VI. CONCLUSIONS AND FUTURE WORKS

We have presented a method to plan and simulate the navigation of crowds of moving entities in large virtual environments. Our solution is based on a structure called *Navigation Graphs*, which decomposes an environment of any kind in sets of interconnected navigable areas. A specific navigation planning technique allows to dispatch a crowd of moving entities navigating between any pair of given locations. We have also introduced a scalable simulation loop, which allows crowd situation update while distributing the available computational resources in space and time. We have equally been able to preserve quality at its best in the foreground of the central area of the screen as well as real-time rates. Finally, we have presented algorithms to solve useful neighbor and visibility queries. Our method is efficient in the case of large crowds. Indeed, the moving entities' positions are updated block by block, according to their current relative position to the actual observation point of view. We save precious computation time by not accessing each of the entities at each update loop. Our method is demonstrated on crowds of virtual pedestrians with real-time visualization experience, however, its principle is general.

In Robotics, the method is applicable to the simulation of a robot in presence of a crowd of virtual humans (VHs). The simulation of VHs can then be scaled using our method: their behavior is complex enough to simulate interactions with the robot, while VHs in the background only execute a navigation task. Another application for the Robotics field is to adapt the navigation flow algorithm to obtain several solutions to a single query. Thus, additive criteria could be used to select a solution path to a robot navigation query: width of the passages to cross, visibility over given areas along the path, access goal destination from given direction, etc.

Further works are in progress to address dynamic environments, and more specifically to treat the case in which a passage is partially or totally obstructed by a new obstacle (permanently or not) while the simulation runs. In this case, the Navigation Graph must be adapted interactively (deletion, split or addition of vertices and edges) and previously computed paths must be reconfigured (validity checks, new path searches), as well as moving entities' path reconfiguration.

VII. ACKNOWLEDGMENTS

The authors would like to thank Barbara Yersin and Jonathan Maïm for developing the crowd simulator we have used to produce the images in Figure 3 and for their help during experiments, Mireille Clavien and Renaud Krummenacher for environment design, and Mireille again for video production.

REFERENCES

[1] J.-C. Latombe. *Robot Motion Planning*. Boston: Kluwer Academic Publishers, 1991.

[2] Jean-Paul P. Laumond. *Robot Motion Planning and Control*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.

[3] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[4] Xuejun Sheng. Motion planning for computer animation and virtual reality applications. *CA*, 00:56, 1995.

[5] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

[6] Parris K. Egbert and Scott H. Winkler. Collision-free object movement using vector fields. *IEEE Computer Graphics and Applications*, 16(4):18–24, 1996.

[7] Tomas Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, 32(2):108–120, 1983.

[8] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Proceedings of IEEE Transactions on Robotics and Automation*, pages 566–580, 1996.

[9] James Kuffner and Steven LaValle. Rrt-connect : An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*, 2000.

[10] M.G. Choi, J. Lee, and S.Y. Shin. Planning biped locomotion using motion capture data and probabilistic roadmaps. *SIGGRAPH'03: ACM Transactions on Graphics*, 22(2):182–203, 2003.

[11] Julien Pettré, Jean Paul Laumond, and Thierry Siméon. A 2-stages locomotion planner for digital actors. *SCA'03: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 258–264, 2003.

[12] A. Kamphuis and M.H. Overmars. Finding paths for coherent groups using clearance. *SCA'04: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 19–28, 2004.

[13] M. Sung, L. Kovar, and M. Gleicher. Fast and accurate goal-directed motion synthesis for crowds. *SCA'05: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 291–300, 2005.

[14] Steve Rabin. *AI Game Programming Wisdom*. Charles River Media, Inc., Rockland, MA, USA, 2002.

[15] James Kuffner. Goal-directed navigation for animated characters using real-time path planning and control. *CAPTECH*, pages 171–186, 1998.

[16] R. Bohlin. Path planning in practice; lazy evaluation on a multi-resolution grid. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001.

[17] W. Shao and D. Terzopoulos. Autonomous pedestrians. *SCA'05: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 19–28, 2005.

[18] F. Lamarche and S. Donikian. Crowds of virtual humans : a new approach for real time navigation in complex and structured environments. *Eurographics'04: Computer Graphics Forum*, 23(3):509–518, September 2004.

[19] S. J. Fortune. Voronoi diagrams and delaunay triangulations. *CRC Handbook of Discrete and Computational Geometry*, pages 377–388, 1997.

[20] Julien Pettré, Pablo de Heras Ciechomski, Jonathan Maïm, Barbara Yersin, Jean-Paul Laumond, and Daniel Thalmann. Real-time navigating crowds: scalable simulation and rendering: Research articles. *Comput. Animat. Virtual Worlds*, 17(3-4):445–455, 2006.

[21] C. W. Reynolds. Steering behaviors for autonomous characters. *Proc. of Game Developers Conference*, pages 763–782, 1999.

8.5 Real-time Path Planning for Virtual Agents in Dynamic Environments

Real-time Path Planning for Virtual Agents in Dynamic Environments

Avneesh Sud Erik Andersen Sean Curtis Ming Lin Dinesh Manocha

{sud,andersen,seanc,lin,dm}@cs.unc.edu
Department of Computer Science, University of North Carolina at Chapel Hill

ABSTRACT

We present a novel approach for real-time path planning of multiple virtual agents in complex dynamic scenes. We introduce a new data structure, *Multi-agent Navigation Graph* (MaNG), which is constructed from the first- and second-order Voronoi diagrams. The MaNG is used to perform route planning and proximity computations for each agent in real time. We compute the MaNG using graphics hardware and present culling techniques to accelerate the computation. We also address undersampling issues for accurate computation. Our algorithm is used for real-time multi-agent planning in pursuit-evasion and crowd simulation scenarios consisting of hundreds of moving agents, each with a distinct goal.

Keywords: crowd simulation, Voronoi diagram, motion planning.

Index Terms: I.3.5 [Computing Methodologies]: Computational Geometry and Object Modeling—Geometric algorithms; I.3.7 [Computing Methodologies]: Three-Dimensional Graphics and Realism—Animation, virtual reality

1 INTRODUCTION

Crowds, ubiquitous in the real world from groups of humans to schools of fish, are vital features to model in a virtual environment. Realistic simulation of virtual crowds have diverse applications in architecture design, emergency evacuation, urban planning, personnel training, education and entertainment. Existing work in this area can be broadly classified into *agent-based methods* that focus more on individual behavior, or *crowd simulations* that aim to exhibit emergent phenomena of the groups.

In this paper, we address the problem of collision-free path computation for agents moving in a complex virtual environment. Since individuals constantly adjust their behavior according to dynamic factors (e.g. another approaching individual) in the environment, agent-based techniques that focus on modeling individual behaviors and intents offer many attractive benefits. They often result in more realistic and detailed simulations. One of the key challenges in a large-scale agent-based simulation is global path planning for each virtual agent. The path planning problem can become very challenging for real-time applications with a large group of moving virtual characters, as each character is a dynamic obstacle for other agents. Many prior techniques are either restricted to static environments or perform local collision avoidance computations. The latter can result in unnatural behavior or "getting stuck" in local minima. These problems tend to be more visible in a dynamically changing scene with multiple moving virtual agents.

Main Results: In this paper, we present a novel, real-time algorithm for path planning of multiple virtual agents in a dynamic environment. We introduce a new data structure called "multi-agent navigation graph" or MaNG and compute it efficiently using GPU-accelerated discrete Voronoi diagrams. Voronoi diagrams have been widely used for path planning computations in static environments [6, 20] and we extend these approaches to dynamic environments.

Voronoi diagrams encode the connectivity of the space and provide a path of maximal clearance for a robot from other obstacles. In order to use them for multiple moving agents in a dynamic scene, prior approaches compute the Voronoi diagram for each agent separately by treating the other agents and the environment as obstacles. This approach can become very costly as the number of virtual agents increases. Instead, we compute the *second order* Voronoi diagram of all the obstacles and agents, and show that the second order Voronoi diagram provides *pairwise* proximity information for all the agents simultaneously. Therefore, we combine the first and second order Voronoi graphs to compute the MaNG for global path planning of multiple virtual agents.

The MaNG computes paths of maximal clearance for a group of moving agents with different goals *simultaneously* and does not require a separate path planning data structure for each virtual agent. Furthermore, we compute a discrete approximation to this graph structure by using the rasterization hardware and propose an adaptive culling technique to accelerate the computation. We also address the undersampling issues that arise due to discretization. Some of our key results include:

1. A new global data structure, the "multi-agent navigation graph" (MaNG) for parallel computation of maximal clearance paths among multiple virtual agents moving independently;
2. Interactive global path planning and local collision avoidance for multiple virtual agents, each with possibly different goals, in a complex virtual environment;
3. A fast two-pass algorithm with adaptive culling techniques for computing a discrete MaNG using GPUs;
4. Resolving undersampling issues in discrete graph computation.

The resulting technique is scalable for global path planning of many dynamic agents in a complex virtual world, not necessarily moving in groups. Although our approach is specifically well suited for simulating multiple virtual agents with distinct intentions, it can also be used in conjunction with a crowd simulation. We have demonstrated our algorithm on two challenging scenarios: a pursuit-evasion game of many fruit pickers chased by farmers and crowd simulation. In both these environments, our algorithm is able to perform real-time global path planning and collision avoidance simultaneously for hundreds of virtual agents with distinct goals.

Organization: The rest of the paper is organized as follows. Section 2 reviews prior literature in related areas. In Section 3, we define our notation and give an overview of our approach. We introduce our data structure, MaNG, and show how it can be used for path planning of multiple agents in Section 4. Section 5 describes our efficient algorithm to compute the MaNG in real-time using GPUs. We describe the implementation and highlight two applications of our planning algorithm to complex virtual environments in Section 6, and analyze the algorithm performance in Section 7.

2 Related Work

In this section, we briefly survey related work on multi-agent simulation and Voronoi diagrams for path planning.

2.1 Multiple Agent Simulation

Agent-based methods, such as the seminal work of Reynolds [28], generate fast, simple local rules that can create visually plausible flocking behavior. Numerous extensions that account for social forces [7], psychological models [26], directional preferences [34], sociological factors [23], etc. have been proposed. Interesting techniques for collision avoidance have also been developed based on grid-based rules [22] and behavior models [37].

Most agent-based techniques perform local collision avoidance. However, global path planning techniques are needed to provide goal seeking capability. In practice, global planning algorithms typically use graph search techniques for each agent [2, 11, 19, 35]. Pettre et al. [27] proposed a graph structure that decomposes the space into multi-layered terrains to support fast graph search for multiple characters.

Most recently, a novel approach for crowd simulation based on continuum dynamics has been proposed by Treuille et al. [36]. This work computes a dynamic potential field that simultaneously integrates global navigation with local obstacle avoidance. The resulting system runs at interactive rates and demonstrate smooth traffic flows for three to four groups of large crowds that are moving with common goals. However this work does not address the case where each person's or agent's path needs to be computed separately. Multi-agent path planning has also been investigated extensively in robotics, mostly for performing collaborative tasks [3, 21, 25]. In addition, crowd simulation has also been heavily studied in other fields [14, 18, 29, 30]. We provide detailed comparisons with some closely related methods in Section 7.

2.2 Voronoi Diagrams and Path Planning

Voronoi diagram is a fundamental proximity data structure used in computational geometry and related areas [24]. Generalized Voronoi diagrams (GVD) of polygonal models have been widely used for motion planning [5, 20]. The boundaries of the generalized Voronoi diagram represent the connectivity of the space. Moreover, they are used to compute paths of maximal clearance between a robot and the obstacles based on potential field approaches [4, 16] or bias the sample generation for a randomized planner [10, 13, 39]. However, sampling-based methods are limited to static environments and the potential-field based planners have been used for 2D environments with very few robots or agents.

A disadvantage of using the GVD is the practical complexity of computing it efficiently and robustly. Hence, several approaches have been proposed to compute an approximation of the GVD. Vleugels and Overmars[38] use adaptive spatial subdivision. Choset and Burdick [5] define a related structure called *hierarchical generalized Voronoi graph* which is computed using continuation methods. Wilmarth et al. [39] compute points on the GVD without explicitly computing a representation of the entire set. Another set of approaches compute a discrete Voronoi diagram along a uniform grid using graphics hardware [15, 33, 8].

3 Background and Notation

In this section we introduce the notation used in the paper, give a background on Voronoi diagram based motion planning, and present an overview of our approach.

3.1 Notation

A geometric primitive or an object (in 3-dimensions) is called a *site*. In this work, a site refers to a point, an open edge, an open triangle, or a connected polygonal object, and we restrict ourselves to 2D environments. An entity for which a path needs to be computed is called an *agent* (or a robot). All obstacles and agents are represented as sites. The center of mass of a site p_i is denoted as $\pi(p_i)$.

Given a site p_i, the scalar distance function $d(\mathbf{q}, p_i)$ denotes the distance from the point $\mathbf{q} \in \mathbb{R}^n$ to the closest point on p_i. Given a set of sites P in domain D, and a subset T of P, with $|\mathsf{T}| = k$, the *k-th order Voronoi region* is the set of points closer to a site in T than to any other site:

$$\mathrm{Vor}^k(\mathsf{T}|\mathsf{P}) = \{\mathbf{q} \in \mathsf{D} \mid d(\mathbf{q}, p_i) \leq d(\mathbf{q}, p_j)\ \forall\ p_i \in \mathsf{T}, p_j \in \mathsf{P} \setminus \mathsf{T}\}.$$

The *k-th order Voronoi diagram* is a partition of of the domain D into the k-th order Voronoi regions:

$$\mathrm{VD}^k(\mathsf{P}) = \bigcup_{p_i \in \mathsf{P}} \mathrm{Vor}^k(\mathsf{T}, \mathsf{P})\ ,\ |\mathsf{T}| = k.$$

The standard Voronoi diagram is the same as the 1^{st} order Voronoi diagram $\mathrm{VD}^1(\mathsf{P})$. We are specifically interested in the 1^{st} and 2^{nd}order Voronoi diagrams, denoted as $\mathrm{VD}^1(\mathsf{P})$ and $\mathrm{VD}^2(\mathsf{P})$, respectively. A 1^{st}order Voronoi region $\mathrm{Vor}^1(p_i|\mathsf{P})$ contains points closest to site p_i, and the 2^{nd}order Voronoi region $\mathrm{Vor}^2(\{p_i, p_j\}|\mathsf{P})$ contains points closest to two sites p_i and p_j. For ease of notation, we drop the superscript for the 1^{st} order Voronoi diagram $\mathrm{VD}(\mathsf{P})$. The complement of a sub-domain X is denoted as X^c and given by $\mathsf{D} \setminus \mathsf{X}$.

The set of closest k-tuples of sites to a point is called the k-th order *governor set*. For a point $\mathbf{q} \in \mathsf{D}$, let the set of closest k-tuple of sites be $\mathsf{U} = \{\mathsf{T}_0, \ldots, \mathsf{T}_m\}, |\mathsf{T}_i| = k$, i.e. $\mathbf{q} \in \mathrm{Vor}^k(\mathsf{T}_i|\mathsf{P})$. Then the k-th order governor set of $\mathbf{q}$ is denoted as $\mathrm{Gov}^k(\mathbf{q}|\mathsf{P}) = \mathsf{U}$. The 1^{st} order governor set is the set of closest sites, while the 2^{nd} order set of a point is the set of closest pairs of sites.

In 2D, the boundaries of Voronoi regions consist of Voronoi edges which are subsets of the bisector between two sites, and Voronoi vertices which are equidistant from three or more sites. The arrangement of all Voronoi edges and vertices in the k-th order Voronoi diagram is called the k-th order *Voronoi graph*, denoted $\mathrm{VG}^k(\mathsf{P})$. Formally, $\mathrm{VG}^k(\mathsf{P}) = (\mathsf{V}, \mathsf{E})$, where,

$$\begin{aligned}\mathsf{V} =& \{\mathbf{v} \in \mathsf{D} \mid |\mathrm{Gov}^k(\mathbf{v}|\mathsf{P})| \geq 3\}\\ \mathsf{E} =& \{e \mid e = (\mathbf{v_1}, \mathbf{v_2}), \mathbf{v_1} \in \mathsf{V}, \mathbf{v_2} \in \mathsf{V}, \exists \text{ connected curve } c,\ s.t.\\ & c = \mathrm{Vor}^k(p_i|\mathsf{P}) \cap \mathrm{Vor}^k(p_j|\mathsf{P}), \mathbf{v_1} \in c, \mathbf{v_2} \in c\}\end{aligned}$$

The k-th order Voronoi diagram is closely related to the *k-th nearest neighbor diagram*. The k-th nearest neighbor diagram is the partition of D into k-th nearest neighbor regions. The k-th nearest neighbor region of site p_i is the set of points for which p_i is the k-th nearest site. Similarly, the arrangement of the vertices and edges in the k-th nearest neighbor diagram is called the k-th nearest neighbor graph, denoted $\mathrm{NG}^k(\mathsf{P})$. Examples of the 1^{st} and 2^{nd} order Voronoi diagrams, Voronoi graphs, and nearest neighbor diagrams are shown in Figure C.2. The 1^{st} nearest neighbor diagram is identical to the 1^{st} order Voronoi diagram. Further properties of higher order Voronoi diagrams are presented in [9, 24].

3.2 Motion Planning Using Voronoi Diagrams

Voronoi diagrams have been used in motion planning in many ways, including roadmap computation, sample generation, or combined with potential field methods. The set of sites P is the set of obstacles, and the Voronoi diagram of the workspace $\mathrm{VD}(\mathsf{P})$ is computed. The Voronoi graph $\mathrm{VG}(\mathsf{P})$ captures the connectivity of the workspace and provides paths of maximal clearance between the obstacles. The Voronoi vertices closest to the robot and goal are classified as source and destination and the minimum weight path is then computed.

For complex 3D environments, an approximate Voronoi diagram is computed. The computation of discrete Voronoi diagrams and discrete Voronoi graphs can be accelerated using GPUs and has been used for motion planning in dynamic 2D [17] and 3D environments [33]. The Voronoi vertex closest to the agent is set as an intermediate goal and the Voronoi diagram is recomputed as the obstacles move.

However, these approaches are inefficient for computing the path of multiple agents in a dynamic environment. For an agent p_i, the remaining agents need to be considered as obstacles, i.e. the set of obstacles is $\mathsf{P} \setminus \{p_i\}$. Hence to compute the path for agent p_i, the modified Voronoi graph $\mathrm{VG}(\mathsf{P} \setminus \{p_i\})$ needs to be computed. Thus the cost of computing the path for all agents is $O(nc)$, where n is the number of agents and $O(c)$ is the cost of computing each modified Voronoi graph $\mathrm{VG}(\mathsf{P} \setminus p_i)$ for $1 \leq i \leq n$.

3.3 Overview

Our approach for motion planning of multiple agents uses the 1st and the 2nd order Voronoi diagrams to compute a global navigation data structure, the MaNG. The MaNG graph is the union of the 1st and the 2nd order Voronoi graphs and is formally presented in Section 4. We treat each agent as a site (in addition to other obstacles in the environment) and the MaNG is computed. The MaNG can be computed in time $O(c)$, and provides a path of maximal clearance for each agent. In addition, we compute the proximity information from the second order Voronoi diagram [31] and apply it within a potential-field based simulator [16].

4 Multiple Agent Planning using Hybrid Voronoi Graph

In this section we introduce the multi-agent navigation graph and demonstrate its application to multiple agents planning.

4.1 Multi-agent Navigation Graph (MaNG)

For multi agent planning, each moving agent represents a dynamic obstacle for the remaining agents. Hence, our goal is to compute a global navigation data structure that provides the clearance information for each agent. In particular, for each agent we want to compute a graph that provides maximal clearance to the obstacles and remaining agents.

We partition the set of sites P into two subsets - the set of obstacles P_o and the set of agents P_a. The multi-agent navigation graph (MaNG), denoted $\mathrm{MG}(\mathsf{P})$, is a union of the first order Voronoi graph $\mathrm{VG}^1(\mathsf{P})$ and a subset of the second order Voronoi graph $\mathrm{VG}^2(\mathsf{P})$ contained inside the 1st order Voronoi region of each agent. Formally,

$$\begin{aligned}\mathrm{MG}(\mathsf{P}) &= (\mathsf{V},\mathsf{E}), \text{ where}\\ \mathsf{V} &= \{v \mid v \in \mathsf{V}^1 \cup (\mathsf{V}^2 \cap \mathrm{Vor}(p_i|\mathsf{P}))\forall p_i \in \mathsf{P}_a\},\\ \mathsf{E} &= \{e \mid e \in \mathsf{E}^1 \cup (\mathsf{E}^2 \cap \mathrm{Vor}(p_i|\mathsf{P}))\forall p_i \in \mathsf{P}_a\},\\ \mathrm{VG}^1(\mathsf{P}) &= (\mathsf{V}^1,\mathsf{E}^1) \text{ and } \mathrm{VG}^2(\mathsf{P}) = (\mathsf{V}^2,\mathsf{E}^2).\end{aligned}$$

The $\mathrm{MG}(\mathsf{P})$ consists of vertices and edges from the 1st and the 2nd order Voronoi graphs $\mathrm{VG}^1(\mathsf{P})$ and $\mathrm{VG}^2(\mathsf{P})$. Some vertices in $\mathrm{MG}(\mathsf{P})$ are common to both $\mathrm{VG}^1(\mathsf{P})$ and $\mathrm{VG}^2(\mathsf{P})$, however $\mathrm{VG}^1(\mathsf{P})$ and $\mathrm{VG}^2(\mathsf{P})$ do not share any edge [24].

We assign a coloring to each edge and vertice in $\mathrm{MG}(\mathsf{P})$ based on its membership in $\mathrm{VG}^1(\mathsf{P})$ or $\mathrm{VG}^2(\mathsf{P})$. Edges from $\mathrm{VG}^1(\mathsf{P})$ are colored **red** and edges from $\mathrm{VG}^2(\mathsf{P})$ are colored **black**. Further, vertices in $\mathrm{VG}^1(\mathsf{P})$ are colored red, and vertices in $\mathrm{VG}^2(\mathsf{P}) \setminus \mathrm{VG}^1(\mathsf{P})$ are colored black. Finally, each edge in the MaNG is assigned weight based on the cost of traveling that segment. Details on weight computation are presented in Section 5. A 2D example of the MaNG for some point agents and obstacles is shown in Figure C.2.

$\mathrm{MG}(\mathsf{P})$ is closely related to the 2nd nearest neighbor graph $\mathrm{NG}^2(\mathsf{P})$. In particular, we state the following result about their relation. Detailed proofs are provided in a technical report.

Lemma 1. *Given a set of sites* P*, the 2nd nearest neighbor graph* $\mathrm{NG}^2(\mathsf{P})$ *and the MaNG* $\mathrm{MG}(\mathsf{P})$*:*

- $\mathrm{MG}(\mathsf{P}) \subseteq \mathrm{NG}^2(\mathsf{P})$,
- *Given an edge* $e \in \mathrm{NG}^2(\mathsf{P})$ *incident on two 2nd nearest neighbor regions of sites* p_i *and* p_j*. For any point* $\mathbf{x} \in e$*: If* $d(\mathbf{x},p_i) = d(\mathbf{x},p_j) = d(\mathbf{x},\mathsf{P}) \Rightarrow e \in \mathrm{VG}^1(\mathsf{P})$*. If* $d(\mathbf{x},p_j) = d(\mathbf{x},p_i) > d(\mathbf{x},\mathsf{P}) \Rightarrow e \in \mathrm{VG}^2(\mathsf{P})$.

As a consequence of lemma 1, both the 1st and 2nd order Voronoi graphs can be extracted from the 2nd nearest neighbor diagram. We use this result to efficiently compute the MaNG from the 2nd nearest neighbor diagram in Section 5.

4.2 Multiple Agent Planning

In this section, we present our approach for efficient path planning of multiple agents using the MaNG. The path planning problem for each agent is defined as follows: we are given an agent $p_i \in \mathsf{P}_a$, its current position in the workspace given by its center of mass $\pi(p_i)$, and a goal position of the center of mass $\mathbf{g_i}$. We wish to compute a path for p_i from $\pi(p_i)$ to $\mathbf{g_i}$, which is maximally clear and collision free to the remaining sites $\mathsf{P} \setminus \{p_i\}$. Such a path can be computed using the Voronoi graph $\mathrm{VG}^1(\mathsf{P} \setminus \{p_i\})$. We state a result on the equivalence of the paths computed using the 1st order Voronoi graphs and the MaNG.

Lemma 2. *Given an agent* p_i*, and the Voronoi graphs* $\mathrm{VG}^1(\mathsf{P} \setminus \{p_i\})$, $\mathrm{MG}(\mathsf{P})$*:*

1. $\mathrm{VG}^1(\mathsf{P} \setminus \{p_i\}) \subseteq \mathrm{MG}(\mathsf{P})$
2. $\mathrm{VG}^1(\mathsf{P} \setminus \{p_i\}) \cap \mathrm{Vor}(p_i|\mathsf{P}) = \mathrm{MG}(\mathsf{P}) \cap \mathrm{Vor}(p_i|\mathsf{P})$.

Lemma 2 provides an approach for extracting the Voronoi graph $\mathrm{VG}^1(\mathsf{P} \setminus \{p_i\})$, for each agent p_i, from $\mathrm{MG}(\mathsf{P})$. The complete algorithm for computing a path for an agent p_i is given in Algorithm 1, and an example path is shown in Figure C.4. The function *LocatePoint*($\mathbf{g_i}$) returns the 1st order Voronoi region which contains $\mathbf{g_i}$. The source and goal positions are connected to vertices in the MaNG using green edges. *ShortestPath*(p_i,$\mathbf{g_i}$,$\mathrm{MG}(\mathsf{P})$) computes

the minimum weight path from $\pi(p_i)$ to $\mathbf{g_i}$ following only the green and red edges in MG(P). This is equivalent to computing the shortest path by following the 2nd order Voronoi graph inside the 1st order Voronoi region of agent p_i, and the 1st order Voronoi graph everywhere else (see Figure C.4). The first vertex along this path is chosen as an intermediate goal for agent p_i.

Input: Agent p_i, Goal position $\mathbf{g_i}$, Set of sites P, MaNG MG(P)
Output: Path S_i from current position to goal position

$k \leftarrow$ LocatePoint($\mathbf{g_i}$)
if $k = i$ **then**
  $S_i \leftarrow edge(\pi(p_i), \mathbf{g_i})$
  return
Compute $V_i \leftarrow$ set of black vertices in Vor(p_i|P)
Compute $E_i \leftarrow$ set of black edges in Vor(p_i|P)
if $V_i \neq \emptyset$ *and* $\pi(p_i) \notin V_i$ **then**
  Augment MG(P) with green edges
  $e_j = (\pi(p_i), v_j) \forall v_j \in V_i$
  Assign weight to $e_j, w(e_j) \leftarrow d(\pi(p_i), v_j)$
else
  foreach *edge* $e_j \in E_i$ **do**
    Compute $v_j \leftarrow$ closest point on e_j to $\pi(p_i)$
    Augment MG(P) with green edge $e_j = (\pi(p_i), v_j)$
    Assign weight to $e_j, w(e_j) \leftarrow d(\pi(p_i), v_j)$
  end
Compute $V_k \leftarrow$ set of red vertices in Vor(p_k|P)
Augment MG(P) with green edges $e_j = (\mathbf{g_i}, v_j) \forall v_j \in V_k$
Assign weight to $e_j, w(e_j) \leftarrow d(\mathbf{g_i}, v_j)$
Add green labels to each edge $e_j \in E_i$
$S_i \leftarrow$ ShortestPath (p_i,$\mathbf{g_i}$,MG(P))
Remove green labels from each edge $e_j \in E_i$
Remove all green edges from MG(P)

Algorithm 1: ComputePath(p_i, $\mathbf{g_i}$, P, MG(P)): Computes a path for agent p_i to goal $\mathbf{g_i}$ given the set of sites P and the MaNG MG(P)

5 MANG COMPUTATION

In this section we present our approach for efficiently computing the MaNG, which is based on the 1st and the 2nd order Voronoi diagrams. However, exact computation of generalized Voronoi diagrams of polygonal models is non-trivial. Rather, we compute the discrete Voronoi diagram along a uniform grid using graphics hardware [15]. The 2nd nearest neighbor diagram is computed using a second pass with depth peeling, as presented in [9]. We compute the generalized 2nd nearest diagram of higher order sites (lines, polygons) by rendering the generalized distance function for each site [32]. We compute the 1st order Voronoi diagram in the first pass, and compute the 2nd nearest neighbor diagram in the second pass. Finally we extract the 1st and the 2nd order Voronoi graphs from the 2nd nearest neighbor diagram and compute the MaNG.

5.1 Culling Techniques

The distance field is computed by evaluating the distance function to each site at each pixel, and this computation is efficiently performed using the rasterization capabilities of the GPU. However, for a large number of sites, this leads to redundant computation for each pixel, and the computation becomes fill bound. Hence, we use culling techniques to compute conservative bounds on the 1st and the 2nd order Voronoi regions. The distance function to each site is computed on the pixels that are contained within its conservative Voronoi region.

Our goal is to efficiently derive a tight upper bound on the 1st and the 2nd order Voronoi regions for each site. We compute these bounds by determining the closest site (and closest 2 sites) along each principle direction ($+X, -X, +Y, -Y$). We compute the bounds using a quadtree, which subdivides the domain. Each node in the quadtree contains the number of sites contained in the subtree rooted at the node. Using this quadtree we can efficiently determine the set of nearest neighbors for each site.

The quadtree is constructed as follows. Each leaf nodes contains the number of sites contained within the node. Let δ be the size of a leaf node. Each intermediate node contains the number of sites contained in each of its 4 children. Let the function $E(l)$ compute the closest non-empty leaf node to the right of node l in the quadtree. Similarly, $W(l)$ and $N(l)$, $S(l)$, respectively, return closest leaf nodes to the left, top and bottom of node l. To compute the bound along $+X$ for the 1st order Voronoi region of a site p_i, we first identify the leaf node l_i that contains the centroid of the site $\pi(p_i)$. Next we compute the closest leaf nodes $E(l_i)$, $W(l_i)$, $N(l_i)$ and $S(l_i)$. Finally we compute the bisectors of the pairs of nodes $E(l_i), S(l_i)$ and $W(l_i), S(l_i)$. Then the bound on the first order Voronoi region along X axis is given by the intersection of these two bisectors. Similarly, the bounds along Y-axis are computed, and the first order Voronoi region of site p_i is bounded by a quad covering these bounds. In addition, for a leaf node l_i, we store the locations of its closest neighbors $E(l_i)$, $W(l_i)$, $N(l_i)$ and $S(l_i)$.

We compute the bounds on all the 2nd order Voronoi regions of site p_i in the second pass as follows. Along $+X$ axis, we check the number of sites stored in the closest node $E(l_i)$. If the number of sites in node $E(l_i)$ is 2 or more, then the bound along $+X$ is $\Delta X^+ = d(l_i, E(l_i)) + 2\delta$. If number of sites in node $E(l_i)$ is less than 2, then we lookup the node $E(E(l_i))$ (this has been computed in the 1st pass), and the bound along $+X$ is $\Delta X^+ = d(l^i, E(E(l_i))) + 2\delta$. Similarly we compute bounds along $-X, +Y$ and $-Y$ axes and compute the distance function of site p_i in a quad that covers these bounds.

To compute the bounds for a higher order site (a line segment or a convex polygon), we store the position of the centroid of the site in the quadtree. We compute the distance bounds for the centroid using the quadtree, and add the distance between the centroid and a vertex to compute the distance bounds for the site.

5.2 Undersampling Errors

Computation of the Voronoi graph on a uniform grid may result in undersampling errors, which may lead to the Voronoi regions to become disconnected [33],and the computed discrete Voronoi graph may have many small disjoint components [16]. As a result, for complex environments with a large number of sites, the combinatorial complexity of the MaNG becomes very high.

We address the issue of undersampling for motion planning, by reducing the combinatorial complexity of the MaNG without changing its connectivity. We reduce the complexity by appropriately modifying the MaNG near undersampled areas. We rely on the fact that when two Voronoi edges are arbitrarily close, then the agent might follow either edge, as long as the path connectivity does not change. Such edges can be removed from the MaNG provided their removal does not change the connectivity of the MaNG.

We present the details of our algorithm for reducing the complexity of MaNG. We treat an edge with an adjacent edge less than one pixel away as a candidate for removal. Such edges are exactly those edges that bound a discrete Voronoi region of width 1 pixel. Thus the test for eliminating such edges is equivalent to removing certain pixels from a discrete Voronoi region, which does not change the

connectivity of the Voronoi graph. Hence our test for removal of a pixel from a discrete Voronoi region relies on a local 3×3 stencil around a pixel. Let p_a be the governor of a pixel (i, j), and the set α denote the governor set of the 4 adjacent pixels $(i-1, j), (i+1, j), (i, j-1), (i, j+1)$. Then the pixel (i, j) can be removed if either of the following conditions holds (see Figure C.5):

1. $p_a \notin \alpha$. Then site p_a has an isolated discrete Voronoi region at pixel (i, j), with 4 Voronoi edges surrounding it. Removal of this Voronoi region does not change the path connectivity in the stencil.
2. $p_a \in \alpha$ and p_a occurs in α exactly once. Then the pixel (i, j) represents an end point of a discrete Voronoi region of site p_a and its removal does not change the path connectivity in the stencil.

After a pixel (i, j) satisfies the criteria for removal, we assign it to another discrete Voronoi region to maintain the connectivity of Voronoi edges. The pixel is assigned to a site in $\alpha \setminus \{p_a\}$ with the minimum distance to pixel (i, j). The distance of a site in α to pixel (i, j) can be efficiently computed by relying on the fact that distance vectors are bi-linearly interpolated [32]. Thus distance computation involves a vector summation with a basis vector and vector norm computation.

The operation performed at each pixel is a read followed by a conditional write, and the output of one pixel may affect the connectivity of adjacent pixels. Thus an efficient parallel algorithm is not feasible, and we perform a sequential scan of the discrete Voronoi diagram to update the Voronoi graph.

5.3 Graph Construction

We now present our algorithm to compute the MaNG. We compute the 1^{st} order Voronoi diagram $\mathrm{VD}^1(\mathsf{P})$ and the 2^{nd} nearest neighbor diagram on the GPU, and refine the connectivity information based on the algorithm described in Section 5.2. We then perform sequential tracing of vertices and edges to compute the 2^{nd} nearest neighbor graph [15].

We use the result presented in Lemma 1 to classify the edges in the 2^{nd} nearest neighbor graph, $\mathrm{NG}^2(\mathsf{P})$. An edge is classified as belonging to the 1^{st} order Voronoi graph if the distance to closest site for all pixels on the edge is identical in $\mathrm{VG}^1(\mathsf{P})$ and $\mathrm{NG}^2(\mathsf{P})$. Due to pixel resolution errors, we treat two distance values as identical if they are within one pixel width of each other. Each edge is assigned a weight proportional to its length and inversely proportional to the minimal clearance along the edge. An edge belonging to $\mathrm{VG}^1(\mathsf{P})$ is labeled red, and the remaining edges are labeled black. A vertex is labeled red if it has at-least one red edge incident on it, otherwise it is labeled black. These colors are used by Algorithm 1 to search for an optimal path.

6 Implementation and Results

In this section we describe the implementation of our multi agent planning algorithm and highlight its application to various multi-agent simulations.

6.1 Implementation

We have implemented our algorithm on a PC running Windows XP operating system with an AMD Opteron 280 CPU, 2GB memory and an NVIDIA 7900 GPU. We used OpenGL as the graphics API and Cg language for implementing the fragment programs. The discrete Voronoi diagram and distance field are computed at 32-bit floating point precision using floating point buffers. The Voronoi diagram is stored in the red channel, and the distance field in the depth buffer. We use stencil tests to disable 2^{nd} order Voronoi diagram computation in the 1^{st} order Voronoi regions of the obstacles. In the first pass, the stencil mask is set for all pixels in the 1^{st} order Voronoi regions of the agents. In the second pass, distance functions are evaluated at pixels with stencil mask set. This optimization speeds up both discrete Voronoi diagram computation and MaNG construction. We perform readback of the discrete Voronoi diagrams and construct the MaNG on the CPU. The optimal path is computed using an A^* search with Euclidean distance metric to guide the search.

We use a complete quadtree for Voronoi region culling described in Section 5.1. The depth of the quadtree is set such that one leaf node corresponds to a block of 32×32 pixels. We need to determine if a node contains up to 2 sites - hence the number of sites per node is encoded in 1 byte. By using a complete quadtree, the node addresses can be efficiently computed using bit shifts, avoiding pointer addressing.

6.2 Demos

We describe two multi-agent simulations, demonstrating the effectiveness of the MaNG for real-time path planning. The first simulation involves a coverage problem, while the second one is of a crowd simulation.

Fruit Stealing Game: The first simulation is of fruit stealing in a dense orchard (see Figure C.1). There are several agents (thieves) which attempt to steal the fruit on the trees. The environment also contains some old farmers who chase the thieves. As the thieves move through the orchard, they steal fruit in close proximity. The goal is for each thief to move towards denser regions of fruit while avoiding the farmers, the trees and other thieves. The thieves, farmers and trees are treated as cylindrical sites. The trees are fixed obstacles, farmers are dynamic obstacles and the thieves are the agents. A coarse density map is used to track the density of fruit remaining in the orchard. Trees with desirable fruit are assigned higher density. The agents are initially spread near the boundary of the orchard, and the goal position is set to a distant high density region. The goal position for each agent is also dynamically updated as the density of the current goal drops below a certain threshold.

The global path of each agent is computed using the approach presented in Algorithm 1. We compute the proximity to nearest site for each agent from the 2^{nd} nearest neighbor diagram, which is used in a potential planner for local planning. Finally, we also use the 2^{nd} order Voronoi diagram to compute the closest agent (thief) for each farmer. This is set as the goal for each farmer and the farmer moves directly towards this agent. The farmers do not use the MaNG for path planning, however they use the potential and repulsive forces to stay clear of other farmers and trees. A thief is eliminated if caught by a farmer. Hence it is desirable for each thief to compute shortest paths of maximal clearance from the farmers (dynamic obstacles) and other thieves (agents) in order to collect the most fruit.

Crowd Simulation: We simulate a crowd of people moving in an urban environment with dynamic obstacles (Figure C.3). We simulate only the individual behavior and not the group behavior. The set of sites consists of buildings, cars and humans. The humans enter the scene from one of the buildings and exit through another building or the sidewalks. Each human is an individual agent with an independent goal. The cars are dynamic obstacles, while the buildings, benches, fountains are static obstacles. Similar to fruit picking, the proximity information for local planning is computed using the 2^{nd} order Voronoi diagram. The total force applied on each agent is a sum of an attractive force to move it towards the intermediate goal computed by the MaNG, and the repulsive forces

from the nearest neighbors. For goals in close proximity, only the local potential field planner is used, disregarding the MaNG.

6.3 Results

We now highlight the performance of our algorithm in complex dynamic environments. Our approach can perform real-time path planning for each agent in environments up to 200 agents with different destinations, at the rates of 5 to 20 fps. The discrete Voronoi diagrams are computed on grid of resolution $1K \times 1K$ pixels. The fruit stealing simulation has 64 trees with a varying number of thieves and farmers. The crowd simulation has 15 static obstacles and between 2 and 5 moving cars, with a varying number of humans. The performance of our approach, with a timing breakup is presented in Table 1.

Demo	Agents	Graph		Time(ms)			
		$\|\mathsf{V}\|$	$\|\mathsf{E}\|$	DVD	MaNG	Plan	Total
Crowd	10	206	1051	7	20	0.23	52
Crowd	25	330	1949	9	22	0.8	57
Crowd	50	560	3500	10	36	2.0	73
Crowd	100	946	7058	15	65	5.6	110
Crowd	200	1927	14669	20	150	18	213
Fruit	10	565	2282	8	25	1.0	59
Fruit	100	1378	6099	15	70	20	130

Table 1: Performance of multi-agent path planning algorithm (average over all frames): $|\mathsf{V}|$ and $|\mathsf{E}|$ denote number of vertices and edges in the MaNG. DVD = Time to compute the 2^{nd} order discrete Voronoi diagram on the GPU, and removing undersampled regions. MaNG = Time to extract the MaNG from the discrete Voronoi diagram. Plan = time for path planning for all agents. Time for readback of discrete Voronoi diagram and depth buffers at $1K \times 1K$ resolution = 25ms.

7 Analysis and Comparison

In this section, we analyze the performance of our algorithm. We highlight its computational complexity and compare it with other approaches for multi-agent path planning.

7.1 Analysis

Let the number of sites be n, and the size of the grid used to compute the discrete Voronoi diagrams be $m \times m$. We assume the number of agents $|\mathsf{P}_a| = O(n)$. We now present the time complexity of each stage in our algorithm.

The cost of computing the 1^{st} and 2^{nd} order discrete Voronoi diagrams is as follows. The size of the quadtree is $O((\frac{m}{32})^2)$, and depth $= O(\log m)$. Then the cost of computing the bounds for each site (see Section 5.1) is $O(\log m)$. The cost of rasterizing the distance function for a site p_i is $O(r|\mathrm{Vor}^k(p_i|\mathsf{P})|)$, where $|\mathrm{Vor}^k(p_i|\mathsf{P})|$ is the number of pixels in the Voronoi region of p_i and r depends on the tightness of the computed Voronoi region bounds, $1 < r < O(n)$. Typically, we have observed $r = O(1)$. Then the cost of computing the Voronoi diagram is $O(n \log m + \Sigma_{i=1}^{n}(r|\mathrm{Vor}^k(p_i|\mathsf{P})|)) = O(rm^2 + n\log m)$.

The cost of reading back the framebuffers is $O(m^2)$. The cost of extracting the MaNG is $O(|\mathsf{E}|)$, where $|\mathsf{E}|$ is number of edges in MaNG. From lemma 2, the number of edges in MaNG, $|\mathsf{E}| \leq |\mathsf{E}^1| + |\mathsf{E}^2|$, where $|\mathsf{E}^k|$ is number of edges is $\mathrm{VD}^k(\mathsf{P})$, and $|\mathsf{E}^k| = O(kn)$ [9]. Thus cost of extracting the MaNG is $O(n)$. The cost of path planning using A^* is typically polynomial in $O(|\mathsf{E}| + |\mathsf{V}|)$. Therefore cost of computing all paths is $O(n(|\mathsf{E}| + |\mathsf{V}|)) = O(n^2)$. In practice, as shown by Table 1 the associated constant with path planning is much smaller and the bottleneck is the discrete Voronoi diagram computation and graph construction.

7.2 Comparisons

Next we provide qualitative comparisons of our approach with prior methods for multi-agent planning.

Comparison with 1^{st} order Voronoi diagram: Our approach provides a global solution for path planning of each agent using the MaNG. The MaNG computes a roadmap of maximal clearance collision free paths for each agent in $O(1)$ passes, as compared to $O(n)$ passes for computing $O(n)$ Voronoi roadmaps. In particular, using the 2^{nd} order Voronoi graph for path planning guarantees that the position selected as the first intermediate goal along the computed path is unique. This prevents adjacent agents from moving towards the same intermediate goal and getting stuck in a local minimum of the potential function. An example is presented in Figure C.6. In this example, adjacent agents select the same intermediate goal from 1^{st} order Voronoi diagram, whereas the intermediate goals from the 2^{nd} order Voronoi diagram are unique. In addition, the path computed has maximal clearance. More specifically, vertices on the Voronoi diagram are used to compute the area of maximum coverage for a new site [1]. Hence by following the vertices on the MaNG, our planning approach ensures a maximum coverage region for each agent.

The closest related work by Pettre et al. [27] computes an initial roadmap of a static environment using Voronoi diagrams, and constructs a set of homotopic paths for a group of agents. This work implicitly groups agents by their origins and goals. Furthermore, local collision avoidance is not guaranteed. In contrast our algorithm is able to handle dynamic environments as the roadmap is updated in real-time, and the use of 2^{nd} order Voronoi diagrams provides pairwise proximity information which is used to guarantee collision avoidance.

The work on continuum crowds [36] computes a dynamic potential field and updates the position of each agent by moving along the gradient of the potential function. The potential field is computed for a small number of groups of agents moving with common goals. However, due to the use of a potential function the agents may get stuck in a local minimum. In contrast, our approach allows for an independent goal for each agent.

In comparison to agent based methods, our MaNG based path planning algorithm provides global paths, and may be combined with rule-based techniques to simulate more complex and realistic agent behavior.

7.3 Limitations

There are some limitations of our work. We compute the MaNG in the workspace, hence the approach does not scale well for agents with many degrees of freedom (e.g. snakes). We use an A^* graph search algorithm, which may not be optimal. Finally, we compute an optimal path for each frame, however there is no guarantee on coherence of paths across frames, or on convergence over a period of time. In fact, the optimal paths across two time steps may not be coherent, potentially resulting in noisy motions.

8 Conclusions and Future Work

We have presented a novel approach for real-time path planning of multiple virtual agent, based on a new data structure - the Multi-agent Navigation Graph (MaNG). The MaNG is used to simultaneously compute the paths of maximal clearance for a set of moving agents with independent goals. The MaNG is constructed dynamically using discrete Voronoi diagrams. We also presented culling techniques for accelerating the discrete Voronoi diagram computation and addressed undersampling issues due to discretization. We have demonstrated the application of our approach to real time simulation involving a large number of independent agents, each with an individual goal.

There are several avenues for future work. One relevant avenue is to constrain the graph search to compute temporally coherent paths which are guaranteed to converge to the final goal. We would like to exploit coherence in graph search when many agents have similar goals and initial positions. Efficient parallel algorithms for simplifying the discrete Voronoi graphs and computing the MaNG would be useful for accelerating the computation. Finally, we would like to extend our approach to handle agents with high degrees of freedom.

References

[1] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, Sept. 1991.

[2] O. B. Bayazit, J.-M. Lien, and N. M. Amato. Better group behaviors in complex environments with global roadmaps. *Int. Conf. on the Sim. and Syn. of Living Sys. (Alife)*, 2002.

[3] M. Bennewitz and W. Burgard. Finding solvable priority schemes for decoupled path planning techniquesfor teams of mobile robots. *Proceedings of the 9th Int. Symposium on Intelligent Robotic Systems (SIRS)*, 2001.

[4] J. Champagne and W. Tang. Real-time simulation of crowds using voronoi diagrams. *EG UK Theory and Practice of Computer Graphics*, 2005.

[5] H. Choset and J. Burdick. Sensor based motion planning: The hierarchical generalized Voronoi graph. In *Algorithms for Robot Motion and Manipulation*, pages 47–61. A K Peters, 1996.

[6] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[7] O. C. Cordeiro, A. Braun, C. B. Silveria, S. R. Musse, and G. G. Cavalheiro. Concurrency on social forces simulation model. *First International Workshop on Crowd Simulation*, 2005.

[8] M. Denny. Solving geometric optimization problems using graphics hardware. In *Proc. of Eurographics*, 2003.

[9] I. Fischer and C. Gotsman. Fast approximation of high order Voronoi diagrams and distance transforms on the GPU. Technical report CS TR-07-05, Harvard University, 2005.

[10] M. Foskey, M. Garber, M. Lin, and D. Manocha. A voronoi-based hybrid planner. *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2001.

[11] J. Funge, X. TU, and D. Terzopoulos. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. *Proc. of ACM SIGGRAPH*, 1999.

[12] P. Glardon, R. Boulic, and D. Thalmann. Dynamic obstacle clearing for real-time character animation. *Computer Graphics International*, 2005.

[13] L. Guibas, C. Holleman, and L. Kavraki. A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach. In *Proc. of IROS*, 1999.

[14] D. Helbing, L. Buzna, and T. Werner. Self-organized pedestrian crowd dynamics and design solutions. *Traffic Forum 12*, 2003.

[15] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast computation of generalized voronoi diagrams using graphics hardware. *Proceedings of ACM SIGGRAPH 1999*, pages 277–286, 1999.

[16] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. Interactive motion planning using hardware accelerated computation of generalized voronoi diagrams. *IEEE Conference on Robotics and Automation*, pages pp. 2931–2937, 2000.

[17] K. Hoff, A. Zaferakis, M. Lin, and D. Manocha. Fast and simple 2d geometric proximity queries using graphics hardware. *Proc. of ACM Symposium on Interactive 3D Graphics*, pages 145–148, 2001.

[18] A. Kamphuis and M. Overmars. Finding paths for coherent groups using clearance. *Proc. of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2004.

[19] F. Lamarche and S. Donikian. Crowd of virtual humans: a new approach for real-time navigation in complex and structured environments. *Computer Graphics Forum*, 23(3 (Sept)), 2004.

[20] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[21] T.-T. Li and H.-C. Chou. Motion planning for a crowd of robots. *Proc. of IEEE Int. Conf. on Robotics and Automation*, 2003.

[22] C. Loscos, D. Marchal, and A. Meyer. Intuitive crowd behaviour in dense urban environments using local laws. *Theory and Practice of Computer Graphics (TPCG'03)*, 2003.

[23] S. R. MUSSE and D. Thalmann. A model of human crowd behavior: Group inter-relationship and collision detection analysis. *Computer Animation and Simulation*, 1997.

[24] A. Okabe, B. Boots, and K. Sugihara. *Spatial tessellations: concepts and applications of Voronoi diagrams*. Wiley & Sons, 1992. ISBN 0 471 93430 5.

[25] L. E. PARKER. Designing control laws for cooperative agent teams. *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1993.

[26] N. Pelechano, K. O'Brien, B. Silverman, and N. Badler. Crowd simulation incorporating agent psychological models, roles and communication. *First International Workshop on Crowd Simulation*, 2005.

[27] J. Pettre, J.-P. Laumond, and D. Thalmann. A navigation graph for real-time crowd animation on multilayered and uneven terrain. *First International Workshop on Crowd Simulation*, 2005.

[28] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In M. C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 25–34, July 1987.

[29] M. SOFTWARE. http://www.massivesoftware.com, 2006.

[30] G. Still. *Crowd Dynamics*. PhD thesis, University of Warwik, UK, 2000. Ph.D. Thesis.

[31] A. Sud, N. Govindaraju, R. Gayle, I. Kabul, and D. Manocha. Fast proximity computation among deformable models using discrete voronoi diagrams. *ACM Trans. Graph. (Proc ACM SIGGRAPH)*, 25(3):1144–1153, 2006.

[32] A. Sud, N. Govindaraju, R. Gayle, and D. Manocha. Interactive 3d distance field computation using linear factorization. In *Proc. ACM Symposium on Interactive 3D Graphics and Games*, pages 117–124, 2006.

[33] A. Sud, M. A. Otaduy, and D. Manocha. DiFi: Fast 3D distance field computation using graphics hardware. *Computer Graphics Forum (Proc. Eurographics)*, 23(3):557–566, 2004.

[34] M. Sung, M. Gleicher, and S. Chenney. Scalable behaviors for crowd simulation. *Computer Graphics Forum*, 23(3 (Sept)), 2004.

[35] M. Sung, L. KOVAR, and M. Gleicher. Fast and accurate goal-directed motion synthesis for crowds. *Proc. of SCA 2005*, pages 291–300, 2005.

[36] A. Treuille, S. Cooper, and Z. Popovic. Continuum crowds. *Proc. of ACM SIGGRAPH*, 2006.

[37] X. Tu and D. Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. In A. Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 43–50. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[38] J. Vleugels and M. H. Overmars. Approximating Voronoi diagrams of convex sites in any dimension. *International Journal of Computational Geometry and Applications*, 8:201–222, 1998.

[39] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. Maprm: A probabilistic roadmap planner with sampling on the medial axis of the free space. *IEEE Conference on Robotics and Automation*, pages 1024–1031, 1999.

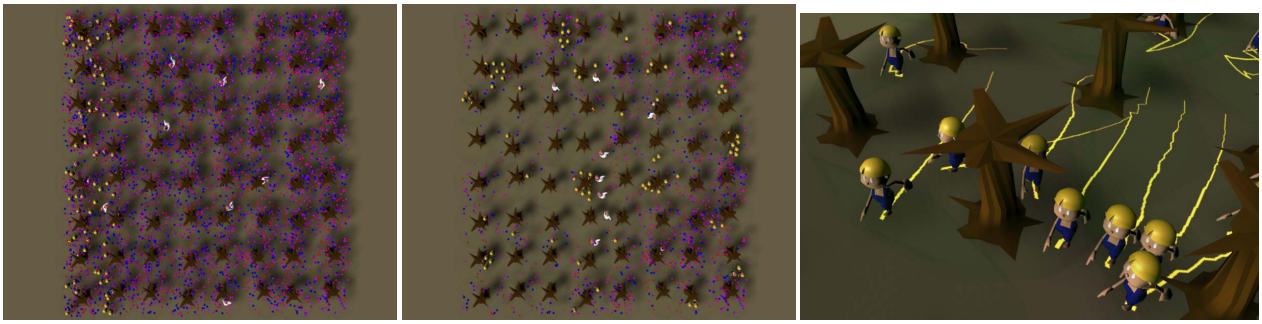


Figure C.1: Fruit stealing simulation: A simulation of 96 *fruit pickers (with yellow hair) in an orchard with* 64*K fruit (dark blue and purple) on* 64 *trees (brown trunks) and* 4 *farmers (in white shirts). Each agent maintains an independent goal.* Left*: Initial top view of the orchard.* Middle*: Top view during the middle of simulation with many fruit collected.* Right*: Perspective view of path traces of the agents in fruit stealing simulation. The yellow curves trace the position of each agent over a range of time steps. The trace demonstrates lane formation as the agents move around obstacles.*

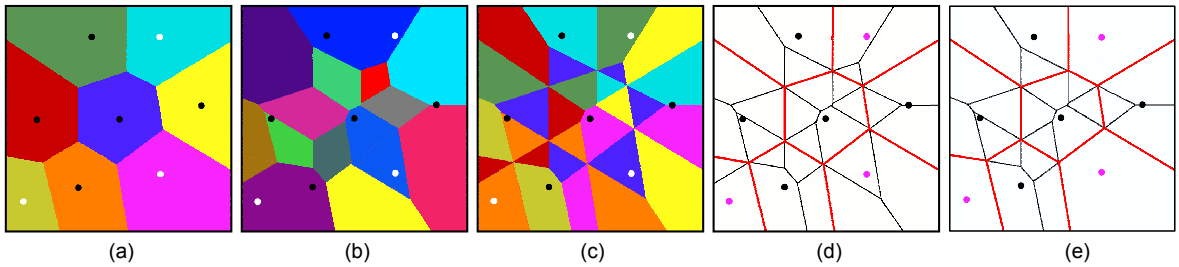


Figure C.2: Voronoi Diagrams and Voronoi Graphs: 8 *point sites consisting of* 3 *obstacles (shown in white) and* 5 *agents (shown in black). (a)* 1^{st} *order Voronoi diagram (b)* 2^{nd} *order Voronoi diagram of the* 8 *sites. Each region is closer to one of a pair of sites than to any other site (c)* 2^{nd} *nearest neighbor diagram. Each region has the same site as the second closest site. (d)* 2^{nd} *nearest neighbor graph. Red edges denote edges from* 1^{st} *order Voronoi graph, black edges are edges from* 2^{nd} *order Voronoi graph (e) the Multi-agent Navigation Graph (MaNG) for the* 5 *agents, which is a subset of the* 2^{nd} *nearest neighbor graph.*

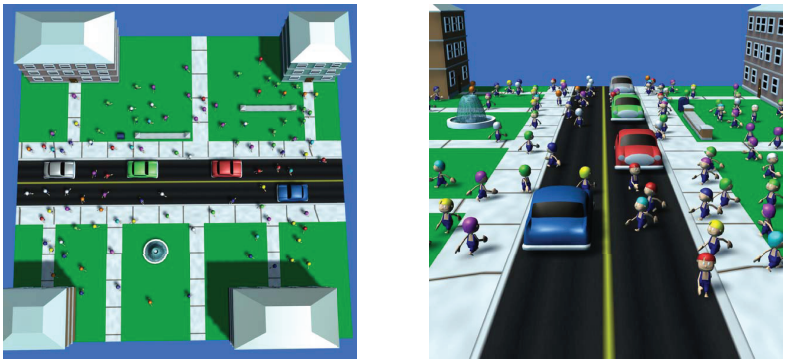


Figure C.3: Crowd Simulation: Two scenes of a crowd simulation with agents moving between buildings and the sidewalks. The cars represent dynamic obstacles. Our MaNG based algorithm can perform path planning on 200 *agents, each with distinct goals, at* 5 *frames per second.*

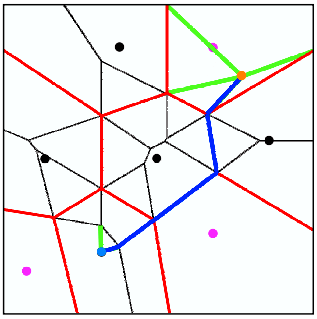


Figure C.4: Multi-Agent Path Planning using the MaNG. The MaNG is augmented with green edges connecting the start position (blue dot) to the goal position (orange dot). The computed shortest path for one agent is shown with blue edges.

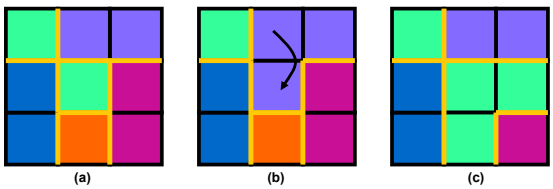


Figure C.5: Discrete Voronoi region shrinking for under-sampling errors: A 3×3 *pixel neighborhood of a discrete Voronoi diagram. The discrete MaNG is shown in thick orange lines. (a) The green discrete Voronoi region is disconnected. (b) The center pixel may be assigned to an adjacent Voronoi region reducing complexity of the MaNG, without changing its connectivity (c) Reassigning the center pixel will change connectivity of the MaNG.*

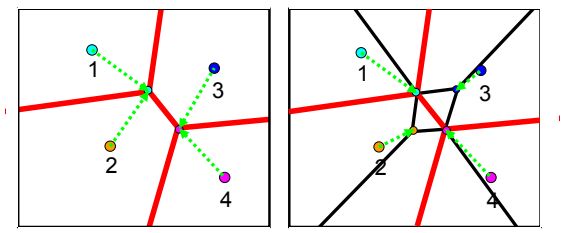


Figure C.6: Comparison of 1^{st} *order Voronoi graph and MaNG:* 4 *agents, with goals in opposite corners.* Left: *Intermediate goals computed from* 1^{st} *order Voronoi graph. Pairs of agents move towards same goal.* Right: *Intermediate goals from MaNG. Each agent has a unique intermediate goal.*

8.6 Real-time Navigation of Independent Agents Using Adaptive Roadmaps

Real-time Navigation of Independent Agents Using Adaptive Roadmaps

Avneesh Sud* Russell Gayle* Erik Andersen* Stephen Guy* Ming Lin* Dinesh Manocha*

Dept of Computer Science, University of North Carolina at Chapel Hill

Abstract

We present a novel algorithm for navigating a large number of independent agents in complex and dynamic environments. We compute adaptive roadmaps to perform global path planning for each agent simultaneously. We take into account dynamic obstacles and inter-agents interaction forces to continuously update the roadmap by using a physically-based agent dynamics simulator. We also introduce the notion of 'link bands' for resolving collisions among multiple agents. We present efficient techniques to compute the guiding path forces and perform lazy updates to the roadmap. In practice, our algorithm can perform real-time navigation of hundreds and thousands of human agents in indoor and outdoor scenes.

1 Introduction

Modeling of multiple agents and swarm-like behaviors has been widely studied in virtual reality, robotics, architecture, physics, psychology, social sciences, and civil and traffic engineering. Realistic visual simulation of many avatars requires modeling of group behaviors, pedestrian dynamics, motion synthesis, and graphical rendering. In this paper, we address the problem of real-time motion synthesis for large-scale independent agents in complex, dynamic virtual environments. These agents may correspond to virtual or digital characters that consist of *non-uniform distributions* of many *distinct* entities, each with *independent* behavior, characteristics, and goals. Examples of such environments include virtual humans in large exposition halls, avatars in wide festival arenas, digital actors in busy urban streets, etc.

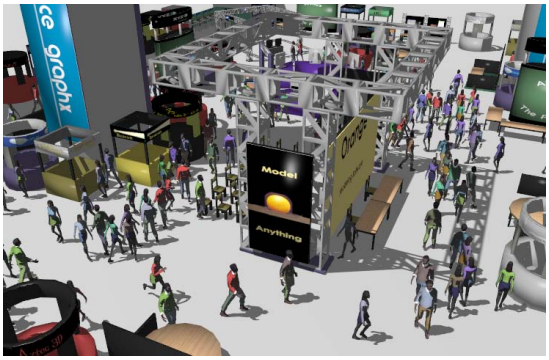


Figure 1: Navigation within an indoor environment: An exhibit hall of a trade show that consists of 511 *booths and* 1,000 *human agents. Each agent has a* distinct *goal (i.e. visiting one of the booths) and behavior characteristic. Our navigation algorithm based on AERO can compute collision-free paths simultaneously for all* 1,000 *agents at* 22 *fps on a PC with* 3*Ghz Pentium D CPU.*

*e-mail: {sud,rgayle,andersen,sjguy,lin,dm}@cs.unc.edu

A major challenge is automatic navigation of each agent through a complex, dynamic environment. More specifically, real-time global path computation for each agent can become a bottleneck, as the number of independent agents in the environment increases. The route planning problem can become intractable, as each individual character moving independently needs to perform collision avoidance with remaining agents. The problem of computing a collision-free path has been extensively studied in robot motion planning, crowd simulation and character animation. Prior global motion planning algorithms are mainly limited to static environments with a few moving robots. Most algorithms for dynamic scenes are based on local collision-avoidance methods, which suffer from convergence and local minima problems. Most of existing work in crowd simulation has been applicable to only a few groups or swarms of agents with the same goal, and not a large number of *independent* agents with *different* intentions [Treuille et al. 2006]. Our work is complementary to these work by addressing the navigation problem simultaneously for many virtual agents with distinct goals and individualized behavior characteristics.

Main Results: In this paper, we present a new algorithm for real-time navigation of large-scale heterogeneous agents in complex dynamic environments. Our approach is based on a novel representation called **"Adaptive Elastic ROadmaps" (AERO)**. AERO is a global connectivity graph that deforms based upon obstacle motion and inter-agent interaction forces.

We use AERO to perform dynamic, global path planning simultaneously for independent agents. We also take into account moving obstacles and the local dynamics among the agents. AERO continuously adapts to dynamic obstacles and deforms according to local force models and global constraints, in order to compute collision-free paths in complex environments. In addition, we introduce the notion of *link bands* to augment the local dynamics and resolve collisions among multiple agents. Due to lazy and incremental updates to the roadmap and efficient computation of guiding-path forces using link bands, our approach can scale to hundreds and thousands of individual agents and perform real-time global navigation of many independent agents in complex, changing environments, with generic obstacles and no restrictions on agent motion.

We demonstrate our approach on complex indoor and outdoor scenarios, including a city scene consisting of 2,000 pedestrians with 50 moving cars and an exhibition hall with 511 stationary booths and 1,000 individual agents on foot and avoiding each other. In order to highlight global navigation, we also place upto 1,000 agents in a dynamic maze environment. Our initial proof-of-concept implementation is able to perform motion simulation of independent agents for these highly challenging scenes in a fraction of a second per frame on a PC with an 3Ghz Pentium D CPU and 2GB memory. As compared to the prior approaches, our algorithm is perhaps among the first to interactively navigate upto thousands of *independent* agents each with *distinct* goals and individualized behavior characteristics, by providing real-time global path planning for all agents *simultaneously* and performing fast local collision avoidance among them.

Organization: The rest of the paper is organized as follows. Section 2 presents related work on multi-agent planning and crowd simulation. We describe "adaptive elastic roadmaps" in Section 3 and use them to navigate multiple virtual agents simultaneously in

Section 4. We describe our implementation and highlight its performance on different benchmarks in Section 5. We analyze the performance of our algorithm and compare it with earlier approaches in Section 6.

2 Related Work

In this section, we give a brief overview of prior work related to multi-agent planning, character animation and crowd simulation.

2.1 Multiple Agent Planning

Extensive literature exists on path planning for multiple agents in robot motion planning and virtual environments [LaValle 2006]. At a broad level, these methods can be classified into *global* (or centralized) and *local* (or distributed) methods. The global paths represent the connectivity of collision-free space in terms of a graph or a roadmap, and require search algorithms to compute a path for each agent [Bayazit et al. 2002; Funge et al. 1999; Kamphuis and Overmars 2004; Lamarche and Donikian 2004; Pettre et al. 2005; Sung et al. 2005; Sud et al. 2007]. Most roadmap based algorithms have been designed for motion planning for a single robot in a static environment and are generated based on random sampling techniques [LaValle 2006]. Recently, some algorithms have been proposed to extend the roadmap-based methods to dynamic environments, multiple agents and deformable models [Gayle et al. 2005; Garaerts and Overmars 2007; Li and Gupta 2007; Pettre et al. 2005; Rodriguez et al. 2006; Gayle et al. 2007; Zucker et al. 2007]. However, they have only been applied to relatively simple environments composed of a few robots and restricted obstacles. These approaches may not scale well to environments with a large number of independent agents.

As compared to global methods, local methods are mostly reactive style planners based on variants of potential fields [Khatib 1986]. They can handle large dynamic environments, but suffer from 'local-minima' problems and may not be able to find a collision-free path, when one exists [LaValle 2006]. Often these methods do not give any kind of guarantees on their behavior. Other route planning algorithms are based on path or roadmap modification, which allow a specified path for an agent to move or deform based upon obstacle motion. These methods include Elastic Bands [Quinlan and Khatib 1993] and Elastic Roadmaps [Yang and Brock 2006]. Our approach bears some close resemblance to these techniques, but AERO is lazily updated to deal with dynamic obstacles and is a significant extension of these algorithms to plan paths for multiple agents simultaneously. We will describe it in detail in Section 4.

2.2 Crowd Dynamics and Human Agents Simulation

Many different approaches have been proposed for modeling movement and simulation of multiple human agents or crowds or individual pedestrians. [Ashida et al. 2001; Schreckkenberg and Sharma 2001; Shao and Terzopoulos 2005; Thalmann et al. 2006; Reynolds 2006]. They can be classified based on specificity, problem decomposition (discrete vs continuous), stochastic vs deterministic, etc.

2.2.1 Discrete methods

Discrete methods rely on a sampling of the environment or of the agents. Some common approaches include:

Agent-based methods: These are based on seminal work of Reynolds [1987] and can generate fast, simple local rules that can create visually plausible flocking behavior. Numerous extensions have been proposed to account for social forces [Cordeiro et al. 2005], psychological models [Pelechano et al. 2005], directional preferences [Sung et al. 2004], sociological factors [MUSSE and Thalmann 1997], etc. Most agent-based techniques use local collision avoidance techniques and cannot give any guarantees on the correctness of global behaviors.

Cellular Automata methods: These methods model the motion of multiple agents by solving a cellular automaton. The evolution of the cellular automata at next time step is governed by static and dynamic fields [Hoogendoorn et al. 2000]. While these algorithms can capture emergent phenomena, they are not physically based. Different techniques for collision avoidance have been developed based on grid-based rules [Loscos et al. 2003] and behavior models [Tu and Terzopoulos].

Particle Dynamics: Computing physical forces on each agent is similar to N-body particle system [Schreckkenberg and Sharma 2001; Helbing et al. 2003]. Sugiyama et al. [2001] presented a 2D optimal velocity (OV) model that generalizes the 1D OV model used for traffic flow. Our formulation is built on some of these ideas and we elaborate them in Section 4.

2.2.2 Continuous Methods

The flow of crowds or multiple agents can be formulated as fluid flows. At low densities crowd flow is like gases, at moderate densities it resembles fluid flow, and at high densities crowd has been compared to granular flow [Helbing et al. 2005]. Most recently, a novel approach for crowd simulation based on continuum dynamics has been proposed by Treuille et al. [2006]. We compare our approach with these methods in Section 7.

3 AERO: Adaptive Elastic ROadmaps

In this section, we describe our representation for navigating heterogeneous agents. We first introduce the notation used in the rest of the paper. Next, we give an overview of our representation and present algorithms to compute it.

3.1 Definitions and Notation

We assume that the multiple agents are contained within a domain D. Each *agent* is denoted as p_i and let the environment consist of k agents, the set of all agents is denoted $\mathscr{A}$. We assume that each agent p_i has a finite radius r_i, a goal position denoted $\mathbf{g_i}$. The dynamics state of each agent at time t consists of its position $\mathbf{x}_i(t)$ and velocity $\mathbf{v}_i(t)$. For ease of notation, we shall not indicate the time dependency of the simulation terms as they are implicitly defined.

In addition to agents, the environment also consists of a set of static and dynamic obstacles. Each obstacle is denoted o_i and set of obstacles is denoted $\mathscr{O}$. The *free space*, is the empty space in the domain, and is given as $D_f = D \setminus (\mathscr{O} \cup \mathscr{A})$. The motion of each agent is restricted to the free space.

The unit normal vector from a point $\mathbf{p} \in D$ to an agent p_i is given by $\mathbf{n}_i(\mathbf{p}) = \frac{\mathbf{p}-\mathbf{x}_i}{\|\mathbf{p}-\mathbf{x}_i\|}$. The agent's *velocity bias field* $\phi_i(\mathbf{p})$ is defined as the angle between the normal to the agent and its velocity, $\phi_i(\mathbf{p}) = \cos^{-1}(\mathbf{n}_i(\mathbf{p}) \cdot \frac{\mathbf{v}_i}{\|\mathbf{v}_i\|})$. Given a pair of agents (p_i, p_j), we define the following: separation distance $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$, separation normal $\mathbf{n}_{ij} = \frac{\mathbf{x}_i-\mathbf{x}_j}{d_{ij}}$. For the sake of simplicity, we assume that all agents have same radius, $r_i = r_j = r_a$ and our algorithm can be easily modified to account for varying radii.

We use the generalized Voronoi diagrams to compute proximity information for the roadmap and link bands. A *site* is a geometric

primitive in $\mathbb{R}^2$. Given a set of sites $\mathscr{S}$, and a domain D, the Voronoi region of a site s_i is denoted $\mathscr{V}(s_i|\mathscr{S})$, and the Voronoi diagram of all sites is VD($\mathscr{S}$). For our work, the sites are the edges on the roadmap, and the domain is the free space.

Multi-agent navigation problem: Given the state of each agent p_i at time t_0, and goal $\mathbf{g_i}$, we compute a sequence of states $\mathbf{q}_i(t_0), \mathbf{q}_i(t_1), \ldots, \mathbf{q}_i(t_f)$, such that $\mathbf{x}_i(t_f) = \mathbf{g_i}, \mathbf{x}_i(t_j) \in D_f(t_j)$ for all i, j. Our goal is to compute a collision-free path for each agent and ensure that its behavior conforms with prior results in pedestrian dynamics. Each agent is also assigned a desired velocity $\mathbf{v}_i^d$, with the magnitude equal to the maximum velocity, v_{max}, of the agent and direction determined by its state and the environment.

3.2 Global Path Planning

Our goal is to use global path planning methods that can help each agent to reach its goal. Prior global approaches are slow in terms of handling complex environments with hundreds of independent agents in real-time. At the same time, local methods can give no guarantees in terms finding a collision-free path and can get stuck in local minima. In order to overcome these problems, we use a novel path planning data structure called Adaptive Elastic ROadmaps (AERO). AERO provides a global roadmap that is updated instantaneously in response to motion of the agents and obstacles in the environment.

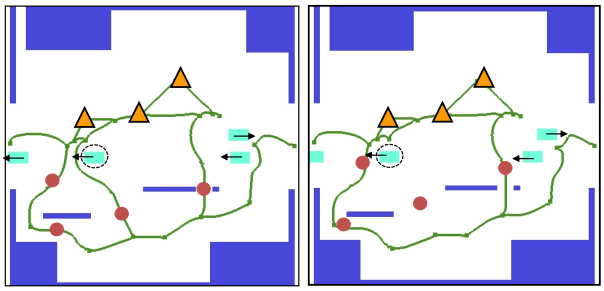


Figure 2: **Adaptive Elastic ROadmap (AERO)***: An example with* 4 *agents (red circles) and goals (yellow triangles). The static obstacles are dark blue rectangles and dynamic obstacles are cyan rectangles with arrows indicating direction of motion. The green curves represent links of the reacting deforming roadmap. The dynamic obstacles represent cars. As the highlighted car (circled) moves, the affected link in the roadmap is removed.*

AERO is a time-varying roadmap, or a connectivity graph of milestones ($\mathscr{M}$) and links ($\mathscr{L}$), $\mathscr{R} = \{\mathscr{M}, \mathscr{L}\}$, and is used to compute the collision-free guiding path for each agent. Each milestone is a position $\mathbf{x}_i \in D_f$, and each link l_{ab} connects two milestones $\mathbf{x}_a, \mathbf{x}_b$ along a path (see Fig. 2). A link l_{ab} is a closed (including the end points) curve in D_f. In path planning literature, this path is often a straight-line for simplicity. Each agent queries the roadmap to compute a path between two configurations in D_f by using a graph search algorithm (such as A*).

3.3 Particle-based graph representation

As the agents and dynamic obstacles move, we compute and update AERO using a physically-based particle simulator. The main components of the graph, dynamic milestones and adaptive links, are built from particles. Particle i, denoted m_i, is a point-mass in D_f which responds to applied forces. The state of the particle at time t is described by its position $\mathbf{x}_i(t)$ and velocity $\mathbf{v}_i(t)$. In the connectivity graph, the dynamic milestones $\mathscr{M}$ are each represented as a particle. Similarly, the adaptive links are represented using a sequence of particles connected by linear springs. The number of particles in each link can vary as a function of link length. As the obstacles (which may include other agents) move, the repulsive forces cause the milestones to move, and the links to deform toward the open areas of the navigation domain (as shown in Fig. 2).

3.4 Applied Force Computation

We apply forces to move the milestones and links away from obstacles while simultaneously maintaining the connectivity of the roadmap. For each particle i, we consider two forces for each particle, roadmap internal forces and repulsive external forces,

$$\mathbf{F_i} = \mathbf{F_i}^{int} + \mathbf{F_i}^{ext},$$

where $\mathbf{F}_i^{int}$ denotes the internal forces and $\mathbf{F}_i^{ext}$ describes external forces.

The internal forces maintain the length of the links, and are simulated using standard damped Hookean spring. Given two particles m_i and m_j, the force on particle m_i from an incident particle m_j is given as:

$$\mathbf{F_i}^{int} = -(k_s(\|\mathbf{x_{ij}}\| - L) + k_d(\frac{\mathbf{v_{ij}} \cdot \mathbf{x_{ij}}}{\|\mathbf{x_{ij}}\|}))\frac{\mathbf{x_{ij}}}{\|\mathbf{x_{ij}}\|},$$

where $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$, $\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$, k_s is a spring constant, k_d is the damping constant, and L is the initial distance between the particles. To prevent the entire roadmap from drifting as a result of moving obstacles, additional springs are attached between the dynamic milestones and a particle is fixed at its initial location. Note that the dynamics milestones are not fixed - instead they are allowed to move to avoid dynamics obstacles.

The external force is repulsive potential force from the obstacles. For each obstacle o_j, we apply a force on particle i, m_i, if it is sufficiently close to o_j. This force given is:

$$\mathbf{F_i}^{ext} = \begin{cases} \frac{b}{d(m_i,o_j)}\mathbf{n} & \text{if } d(m_i,o_j) < \delta, \\ \mathbf{0} & \text{otherwise,} \end{cases}$$

where $d(m_i, o_j)$ is the minimum distance between a particle m_i and obstacle o_j, b is a repulsive scaling constant, $\mathbf{n}$ is the normal from the obstacle to the particle, and δ is a repulsive force threshold.

Given the applied forces, we update the state of AERO, the position and velocity of all particles, using numerical integration. In order to prevent undesirable oscillation in the adaptive links, Verlet integration is used [Verlet 1967; Jakobsen 2001]. This method considers the particles to be at rest, $\mathbf{v}(t) = \mathbf{0}$, during integration. Based on forward Euler integration and Newtonian motion, $\mathbf{F_i} = m\mathbf{a}$, the update rule for particle m_i with unit mass is given as:

$$\mathbf{x}_i(t+dt) = \mathbf{x}_i(t) + \frac{1}{2}\mathbf{a}_i(dt)^2 \quad = \mathbf{x_i}(t) + \frac{1}{2}(\mathbf{F_i}^{int} + \mathbf{F_i}^{ext})(dt)^2.$$

Since velocity is necessary to compute $\mathbf{F_i}^{int}$ we can locally approximate it as

$$\mathbf{v}_i(t+dt) = \frac{\mathbf{x}_i(t+dt) - \mathbf{x}_i(t)}{dt}$$

This formulation describes how a roadmap can adapt to the motion of obstacles. In order to compute the initial roadmap, we can use any of the well known methods in the motion planning literature [LaValle 2006]. In our current implementation, the initial roadmap is generated based on edges and vertices of the generalized Voronoi diagram of the free workspace. This provides good initial clearance from the obstacles and captures all the passages in the environment.

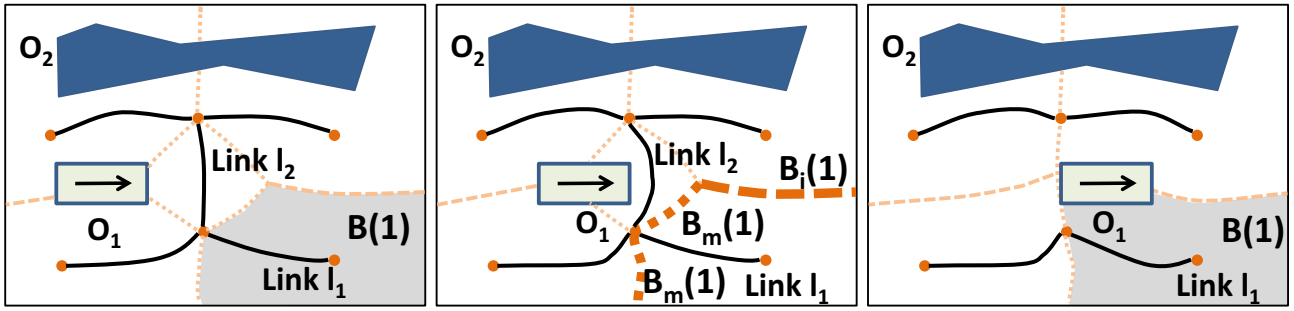


Figure 3: **Roadmap Link Bands***: Link bands are a partition of the freespace based on the links of AERO. (a) Several AERO links, in solid black lines, respond to a static obstacle, O_2 and a dynamic obstacle O_1. The link band, $B(1)$, for link l_1 is shaded, and the link boundaries are shown as dashed lines. (b) As O_1 approaches link l_2 it deforms. Link band $B(1)$'s boundary is highlighted in bold dashed lines and shows the two segments of the milestone boundary, $B_m(1)$, and the intermediate boundary $B_i(1)$. (c) Link l_2 is removed due O_1's motion while the link band $B(1)$ changes to reflect the removal.*

3.5 Roadmap Update

The previous section described the general representation for AERO and how it adjusts to moving obstacles. However, this type of deformation can not guarantee a valid roadmap or that a path can be found at all times, e.g. when a moving obstacle moves directly into a link. These cases require additional link removal or link addition steps to update the roadmap.

3.5.1 Link Removal

In order to maintain a valid roadmap, we remove the links based on both physically-based and geometric criteria. This combination works well since the roadmap computation is a combination of physically-based and geometric approaches.

The physically-based criteria attempts to prune links which have been considerably deformed. A natural measure of deformation for a link is its potential energy. The spring potential energy is a measure of the amount of deformation of a spring. For an adaptive link l_{ab} with n springs $\mathscr{S} = \{s_1, s_2, \ldots, s_n\}$, the average potential is given as

$$P_{ab} = \frac{1}{n} \sum_{s_i \in \mathscr{S}}^{n} \frac{k_s^i}{2} (\|s_i\| - L_i)^2$$

where k_s^i is the spring constant of spring i, $\|s_i\|$ is its current length, and L_i is its rest length. A link is removed when $P_{ab} > \varepsilon_s$, for a spring energy threshold ε_s.

The geometric criteria removes links based on proximity and intersection with the obstacles. Proximity is measured by the nearest distance from an adaptive link l_{ab} to the obstacles,

$$d_{ab} = min_{s_i \in \mathscr{S}, o_j \in \mathscr{O}}(d(s_i, o_j)).$$

Links are removed when this distance is less than the largest radius assigned to an agent, i.e. $d_{ab} < r_a$.

3.5.2 Link Addition

As the links are removed, AERO may no longer capture the connectivity of the free space and maybe unable to find paths. To remedy the situation, it is necessary to repair and add links to the graph. Since our initial roadmap is based on Voronoi diagram, it almost captures the connectivity of the freespace for *static* obstacles. Based on this assumption, we can bias the link addition step to repair links which have been previously removed. When a link is removed, it is placed in a list and re-inserted into AERO when the straight-line path between link's two milestones becomes collision-free. This has the added advantage that our roadmap will try to maintain the connectivity of the freespace of the static environment.

However, in a realistic scenario, it may not be the case that the static and dynamic obstacles are known ahead of time. In this case we need the ability to add milestones to help explore the freespace as the environment changes. In this case, we use random sampling techniques to generate new milestones and additional links [LaValle 2006].

We modify this approach by biasing our search toward areas *behind* obstacle motion. Given the velocity $\mathbf{v_j}$ of an obstacle o_j, we generate a sample outside of the obstacle's axis aligned bounding box in the direction $-\frac{\mathbf{v_j}}{\|\mathbf{v_j}\|}$ from the box's center. This sample is randomly perturbed to help remove uniformity among samples, which can lead to overly regular connectivity graphs. The milestones near to this sample are found and checked to see if an adaptive link can be added. Since a large number of new links can become computationally expensive, link addition is only performed when no path exists.

3.6 Numerical Stability

Since AERO relies upon numerical integration, stability is a concern due to the possibility of applying large spring or repulsive forces. However, when combined with the removal rules, this has not been an issue in practice. In general, using Verlet integration greatly helps in stability by treating the particle to be at rest during integration. Also, our link removal steps also help to ensure stability. Any link which is likely to become unstable will also likely be close to an obstacle or otherwise highly deformed. These types of links will be removed, helping the system remain in a stable state.

4 Navigation using AERO

In this section we describe our approach to compute collision-free paths for independent agents using AERO. In order to allow the agents to occupy the entire free space for navigation, we relax the restriction of constraining the agent's position to the links of the roadmap. Rather, we introduce *link bands* defined by each link of AERO, and use them for path planning as well as local dynamics simulation of each agent (See Figure 5).

4.1 Link Bands

The *link band* associated with a link of the roadmap is the region of free space that is closer to that link than to any other link on the roadmap. Formally, the link band of a link l_i is given by $B(i) = \mathscr{V}(l_i|\mathscr{L}) \cap D_f$. The width of the link band is the minimum clearance from the link to the obstacles in the environment, $B_w(i) = \min_{o_j \in \mathscr{O}}(d(l_i, o_j))$. The link bands form a partitioning of the free space based on proximity to the links. Each link band specifies a collision free zone in a well-defined neighborhood of each link of the roadmap. Additionally, link bands provide the nearest link, which is required for path search (Section 4.2), and distance to the roadmap that is used to compute guiding forces to advance the agent along the path (Section 4.3).

In a dynamic environment, an agent's path might require recomputation. We use link bands to detect such events. In particular we keep track of an agent's motion across a link band boundary. We classify points on a link band boundary into *milestone boundary* and *intermediate boundary* (see Fig. 3). A point on the milestone boundary belongs to two adjacent link bands whose links share a common milestone. On the other hand, the intermediate boundary is all points on the boundary that do not belong to the milestone boundary (see Fig. 3(b)). Formally, the milestone boundary of a link l_i, $B_m(i) = B(i) \cap B(j),\ \forall l_i \cap l_j \neq \emptyset$, and the intermediate boundary of a link l_i, $B_i(i) = \delta B(i) \setminus B_m(i)$. In the next section we show how the link bands are used for global path planning and to detect replanning events.

4.2 Path Planning

We use AERO for global path computation for each agent. Since an agent is not constrained to the roadmap, we initially compute the link band it belongs to. This link is set as the source link. Similarly the link band containing the goal position is computed and the corresponding link is set as the goal link. We assign a weight to each link as a combination of the link length, the reciprocal of the link band width, and the agent density on the link as:

$$w(l_i) = \begin{cases} \infty & \text{if } 2B_w(i) < r_a, \\ \alpha|l_i| + \beta\frac{1}{B_w(i)} + \gamma\frac{n}{|l_i|B_w(i)} & \text{otherwise,} \end{cases}$$

where α, β, γ are constants, $|l_i|$ is the length of link l_i, and n is the number of agents on link $B(i)$. The third term approximates the agent density on the link and causes the agents to plan using less crowded regions. The relative values of the constants are determined by the behavior characteristics of individual agents. A high relative value of α allows for choice of shortest path, a high value of β avoids narrow passages and a high value of γ demonstrates preference of less crowded passages. In our experiements, we used a high value of α for slow agents, whereas aggressive agents are assigned a higher value of γ. Given a weighted roadmap, an A* graph search is performed to compute the minimum weight path from the source to goal link band, which is stored by the agent. Once the agent reaches its goal link band, it proceeds to its goal position within the band.

As the simulation progresses, the nearest link to an agent may change. Based on link boundaries, we determine events that require a path recomputation. Crossing a milestone boundary indicates agent motion along the global path, and does not require a path recomputation. However, it is also possible for an agent to cross over the intermediate boundary. This typically occurs as a result of roadmap modification. In this case, it is possible that an alternative path to the goal exists. However, we allow the agent to move back to its previous path since this should be its path of least cost.

4.3 Local Dynamics Computation

Given a path on AERO, the motion of each agent is computed using a local dynamics model. In this section, we describe the local dynamics model used to guide an agents along the computed path. Our local dynamics model is based on the generalized force model of pedestrian dynamics proposed by Helbing et al [2003]. This force model has been shown to capture emergent crowd behavior of multiple agents at varying densities of crowds. We define the social force model in terms of force fields that are defined over each link band.

We modify the social force model, to include a force $\mathbf{F}^r$ that guides an agent along a link band on the roadmap. In addition, there is a repulsive force $\mathbf{F}^{soc}$ to the nearby agents, an attractive force $\mathbf{F}^{att}$ to simulate the joining behavior of groups, and a repulsive force from dynamic obstacles $\mathbf{F}^{obs}$. Let the agent p_i belong to link band $B(k)$, then the force field at a point $\mathbf{p}$ is given as

$$\mathbf{F}(\mathbf{p}) = \sum_j \left[\mathbf{F}_j^{soc}(\mathbf{p}) + \mathbf{F}_j^{att}(\mathbf{p})\right] + \mathbf{F}_k^r(\mathbf{p}) + \Sigma_o \mathbf{F}_o^{obs}(\mathbf{p}),$$
$$p_j \in \mathscr{A}, j \neq i, o \in \mathscr{O}$$

where,

$$\mathbf{F}_j^{soc}(\mathbf{p}) = A_i \exp^{(2r_a - \|\mathbf{p}-\mathbf{x}_j\|)/B_i} \mathbf{n}_j(\mathbf{p}) \left(\lambda_i + (1-\lambda_i)\frac{1+\cos(\phi_j(\mathbf{p}))}{2}\right),$$
$$\mathbf{F}_j^{att}(\mathbf{p}) = -C_j \mathbf{n}_j(\mathbf{p})$$
$$\mathbf{F}_o^{obs}(\mathbf{p}) = A_i \exp^{(r_a - d(\mathbf{p},o))/B_i} \mathbf{n}_o(\mathbf{p}) \left(\lambda_o + (1-\lambda_o)\frac{1+\cos(\phi_o(\mathbf{p}))}{2}\right)$$
$$\mathbf{F}_k^r(\mathbf{p}) = \frac{\mathbf{v}_k^d(\mathbf{p}) - \mathbf{v}_i}{\tau_i} + D_i d^4(\mathbf{p}, l_k)\mathbf{n}_{l_k}(\mathbf{p})$$

where A_i and B_i denote interaction strength and range of repulsive interactions and C_j strength of attractive interaction, which are culture-dependent and individual parameters. λ_i reflects anisotropic character of pedestrian interaction. The obstacle force field $\mathbf{F}^{obs}$ simulates the repulsion of the agents from other obstacles in the environment. Since the obstacles may be dynamic, we introduce an additional anisotropic term which biases the repulsive forces along the motion of the obstacles. This effect has also been modeled in other approaches by creating a 'discomfort zone' in front of dynamic obstacles [Treuille et al. 2006]. For efficient computation of repulsive force $\mathbf{F}^{soc}$ and obstacle force $\mathbf{F}^{obs}$, we compute forces to agents and obstacles within a radius B_i.

The roadmap force field $\mathbf{F}_k^r$ guides the agent along the link l_k. The link band $B(k)$ is used to define the region which is used to compute the force field for l_k. The first term in $\mathbf{F}_k^r$ makes the agent achieve a desired velocity along the link, whereas the second term attracts the agent within the link band. $\mathbf{n}_{l_k}(\mathbf{p})$ is the unit normal from point $\mathbf{p}$ to the closest point on l_k, $d(\mathbf{p}, l_k)$ is the distance from $\mathbf{p}$ to l_k. The desired velocity $\mathbf{v}_k^d(\mathbf{p}) = v_{max}\mathbf{e}_k(\mathbf{p})$, where $\mathbf{e}_k(\mathbf{p})$ is a unit vector field orthogonal to $\mathbf{n}_{l_k}(\mathbf{p})$. The direction of the normal is chosen such that $\mathbf{e}_k(\mathbf{p})$ points along the roadmap towards the next milestone on an agents path. D_i is a weighting term and the attractive force term keeps an agent inside the link band, reducing toggling across intermediate boundaries.

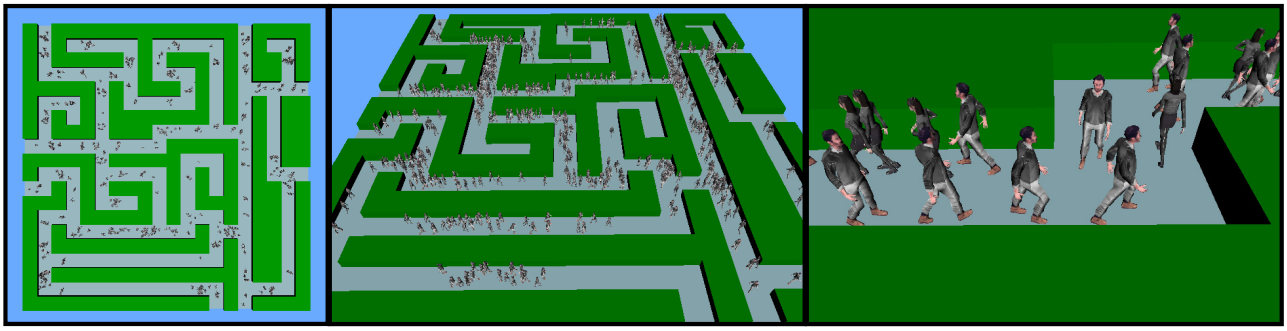


Figure 5: Left: Navigation of 500 *virtual agents in a maze consisting of 8 entrance and 8 exit points. Center: Each agent computes an independent path to the nearest exit using adaptive roadmaps. Right: Our local dynamics simulation framework based on link bands captures emergent behavior of real crowds, such as forming lanes. We perform real-time navigation of* 500 *agents at* 100*fps.*

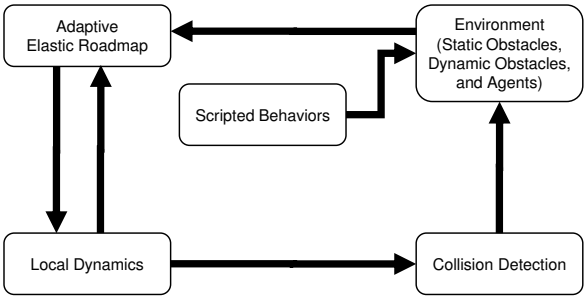


Figure 4: **Navigation System***: Given a description of the environment, an AERO is computed and updatehd. This is used in conjunction with our local dynamics model to simulate the motion of each agent.*

4.4 Behavior Modeling

Once the agent motion has been determined by local dynamics, this motion needs to be animated. Behavioral modeling allows us to translate from this motion to an animated character. To accomplish this task, we use a minimal set of predetermined behaviors; a stop, walk, and jog. A finite state machine is then used to transition and switch between them, as shown in Fig.4.

Transitions between states are determined by an agent's velocity and predefined thresholds. When the velocity is at or very close to zero, the agent moves to a stop state. Similarly, as the agent's speed increases, it transitions to a walk state and then to a jog state a higher speeds. To prevent oscillation between states, the threshold for increasing speeds is different than that of decreasing speeds. This is analogous to the idea that a slow jog can be the same speed as a fast walk.

Depending on the application, some agents are set to be more aggressive by specifying a higher maximum velocity. These agents will be more likely to be in the jogging motion in order to reach their goals.

5 Implementation and Results

In this section we describe the implementation of our multi-agent navigation system and highlight its performance on various environments. We have implemented our algorithm on a PC running Windows XP operating system, with an 3Ghz Pentium D CPU, 2GB memory and an NVIDIA 7900 GPU. We used Visual C++ 7 programming language and OpenGL as the graphics API for rendering the environments. The initial Adaptive Elastic Roadmap (AERO) for an environment is initialized by computing the Voronoi diagram of the static obstacles in the environment. This computation helps initialize the roadmap with links that are optimally clear of obstacles when the simulation begins. To simulate particle dynamics of the agents, we used a semi-implicit verlet integrator [Verlet 1967; Jakobsen 2001].

Proximity computations to dynamic obstacles are accelerated using a spatial hash data structure to identify the nearby objects. We maintain a spatial hash table of all dynamic obstacles, agents and links. Briefly, spatial hashing uses a hash function and table to compress and update a regular spatial decomposition. This step enables efficient lookups and proximity computation. In addition, to accelerate proximity computations to static obstacles, we precompute a discretized Voronoi diagram of the obstacles using the GPU [Sud et al. 2006]. The discrete Voronoi diagram provides proximity information to the nearest obstacle. To get the set of all obstacles within a given radius r, we scan the discrete Voronoi diagram (and distance field) within a window of size $r \times r$ and check if the distance value at the discrete samples is less than r. Thus the proximity computation is reduced to a small number of table lookups.

5.1 Benchmarks

We demonstrate our system on three complex scenarios.

- **Maze**: The maze scenario considers the case of multiple agents navigating a maze. The maze has 8 entry and 8 exit locations, and 1288 polygons. The initial roadmap consists of 113 milestones and 281 links. By using AERO, they have complete knowledge of how to navigate the maze despite its complexity and thus are able to quickly move toward their goals. See Fig. 5.

- **Tradeshow**: The tradeshow scenario is an indoor environment of an exhibit hall in a trade show. The exhibit consists of 511 booths and 110K polygons. The initial roadmap consists of 3996 links and 5996 milestones. Numerous agents walk around and visit multiple booths. The goals for each agent are updated as the agent arrives at a booth. Some agents stop when they reach their goal in order to simulate observation of a particular point of interest. After a certain amount of time, the agents will resume walking towards their next goal. Also, certain booths have fixed agents whose orientation changes according to passing agents. As agents move freely through the floor, they act as dynamic obstacles, and update the AERO. See Fig. 7.

- **City**: The city scenario is an outdoor scene consisting of multiple city blocks. The model consists of 924 buildings and 235K triangles. The initial roadmap for the environment con-

sists of 4K links. The environment also consists of 50 moving cars as dynamic obstacles. As the cars move through the urban setting, links on the path deform around the obstacles and get invalidated. We add a higher potential in front of the cars along their direction of motion, which decreases the probability of the agents from selecting paths in front of moving obstacles. The environment is populated with a non-uniform density of agents moving along the side walks or crossing the streets. Additional behavior characteristics of each agent are assigned at run-time. These individualized behavors includes updating the goals, varying the maximum speed, and changing interaction range of the agents. See Fig. 6.

Demo	Agents	Sim	Path Search	AERO Update	Total Time
Maze	500	9.1	0.005	0.58	9.64
Maze	1000	31.2	0.01	0.58	31.79
Trade Show	500	8.73	3	5.5	17.23
Trade Show	1000	32.95	7	5.5	45.45
City	500	9.75	7.4	15.1	32.25
City	1000	35	13.1	15.1	63.2

Table 1: **Performance on each scenario.** *Timings reported here are the average simulation time per frame (step) broken down into the time for simulating local dynamics (***Sim***), performing path search (***Path Search***), and updating AERO on the fly (***AERO Update***). All timings are in milliseconds.*

5.2 Results

We highlight the performance of our algorithm on the complex benchmarks. Our approach can perform real-time simulation of crowds with up to $1,000$ independent agents at interactive rates – ranging from 16 to 104 frames per second, depending on the scene complexity and the crowd density. Our current implementation is unoptimized and does not make use of all optimized computations on the GPU. The performance of our algorithm in the environments (with different complexity) and varying number of agents is highlighted in Table 1.

6 Analysis and Discussion

In this section we analysis the time complexity of various stages of our algorithm, and provide a qualitative comparison with prior work.

6.1 Analysis

Performance of our approach depends on a number of factors. At each time-step, AERO's complexity is $O(|M|+|E|)$, or linear in the number of particles and edges. The tasks per timestep include force computations, numerical integration, as well as path search and roadmap maintenance. Agent motion also depends linearly in the number of agents, but each agent also performs a path search, thus making the agent portion of computation complexity $O(|N|+|E||N|)$. But, in all of these cases, the performance scales linearly with the number of agents or the complexity of the roadmap. Therefore, this approach should be able to scale well to a large number of agents.

6.2 Comparison and Limitations

We compare some of the features of our approach with prior algorithm and highlight some of its limitations. Our adaptive roadmap based agent navigation algorithm is designed to perform real-time global navigation for a large number (e.g. hundreds or thousands) of independent or *heterogeneous* agents, each with different goals. We also take into account dynamic obstacles and the local dynamics among the agents. AERO continuously adapts to dynamic obstacles and is used to compute collision-free paths in dynamic environments. As compared to local or potential field methods, AERO can compute a global path for each agent. In addition, we use elastic bands and link bands to augment the local dynamics and resolve collisions among multiple agents. Due to lazy and incremental updates to the roadmap and efficient computation of guiding-path forces with link bands, this approach can scale well to hundreds or thousands of agents.

Comparisons: Our work is complementary to several existing works on crowd simulation and multi-agent planning. Continuum Crowds [Treuille et al. 2006] targets navigation for a small umber $(2-5)$ groups of human agents, where each group consists of many (upto thousands) agent with *identical* goals and behavior characteristics. Moreover, this approach uses local collision avoidance and its accuracy is governed by the underlying grid resolution. This approach has not been shown to extend well to a large number of groups or when there are challenging narrow passages in the free space, as shown in our maze and trade show benchmarks.

Graph based approaches[Pettre et al. 2005; Lamarche and Donikian 2004; Li and Gupta 2007] use proximity graphs to capture the connectivity of the navigable space and use it for agent coordination. However, the navigation graphs are precomputed and thus are mainly restricted to static environments. Multi-agent Navigation Graphs [Sud et al. 2007] efficiently compute dynamic navigation graphs for simple agents. However, this approach is limited to a few hundred agents and does not scale with the number of agents. It cannot guarantees coherent and smooth paths, as shown in our video. Corridor Maps [Garaerts and Overmars 2007] use similar proximity ideas as link bands to define navigable space, and can adapt to dynamic obstacles. However, corridor maps cannot easily handle dynamic topology of the roadmap and model emergent behaviors like agents following each other in lanes. Local agent-based and potential-field methods [Reynolds 1987; Shao and Terzopoulos 2005] perform well for a large number of agents and exhibit interesting crowd-like behaviors, but cannot provide same guarantees in path finding as global approaches.

Limitations: Our approach has some limitations. Our current implementation address collision-free navigation of a large number of 3-DoF agents, therefore our work does not produce realistic motion in situations where each human is modeled as a high DoF avatar. Although AERO uses a global roadmap at each given time step for path computation, the local dynamics formulation to update the links can potentially result in an agent getting stuck in a local minimum across space-time. In other words, our work may not be able to provide convergence guarantees or provide completeness on the existence of a collision-free path for each agent in all environments. Furthermore, we currently treat each agent as an individual agent and do not exploit all the behavior-related characteristics of real crowds such as grouping. Finally, the performance of proximity queries is sensitive to choice of hash function parameters and theoretical analysis can be of potential interest for challenging scenarios with many varying parameters.

7 Conclusion

We present a novel approach for real-time navigation of independent agents in complex and dynamic environments. We use adaptive roadmaps and present efficient algorithms to update them. These roadmaps are augmented with link bands to resolve collisions among multiple agents. The algorithm has been applied to complex

indoor and outdoor scenes with hundreds or thousands agents and dynamic obstacles. Our preliminary results are encouraging and the algorithm can compute collision-free paths for each agent towards its goal in real time.

There are many avenues for future work. First of all, we would like to develop multi-resolution techniques to handle a very large number of agents, e.g. 10-20K independent agents at interactive rates. Secondly, we would like to use better models for local dynamics and behavior modeling that can result in more realistic crowd-like behavior. Instead of its current simple model, we would like to use higher DoF articulated models for each agent to generate more realistic motion. However, this would increase the dimensionality of the configuration space and significantly increase the compleixty of the navigation algorithm. Finally, it may be useful to extend these results to generate truly heterogeneous crowd behavior [Bon 1895], using example based models to guide the simulation [Lerner et al. 2007].

Acknowledgments

This work was supported in part by Army Research Office, National Science Foundation, RDECOM, and Intel. We would like to acknowledge members of UNC GAMMA group for useful discussions and feedback. We are also grateful to the anonymous reviewers for their comments.

References

ASHIDA, K., LEE, S. J., ALLBECK, J., SUN, H., BADLER, N., AND METAXAS, D. 2001. Pedestrians: Creating agent behaviors through statistical analysis of observation data. *Proc. Computer Animation*.

BAYAZIT, O. B., LIEN, J.-M., AND AMATO, N. M. 2002. Better group behaviors in complex environments with global roadmaps. *Int. Conf. on the Sim. and Syn. of Living Sys. (Alife)*.

BON, G. L. 1895. *The Crowd: A Study of the Popular Mind*. Reprint available from Dover Publications.

CORDEIRO, O. C., BRAUN, A., SILVERIA, C. B., MUSSE, S. R., AND CAVALHEIRO, G. G. 2005. Concurrency on social forces simulation model. *First International Workshop on Crowd Simulation*.

FUNGE, J., TU, X., AND TERZOPOULOS, D. 1999. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. *Proc. of ACM SIGGRAPH*.

GARAERTS, R., AND OVERMARS, M. H. 2007. The corridor map method: Real-time high-quality path planning. In *ICRA*, 1023–1028.

GAYLE, R., LIN, M., AND MANOCHA, D. 2005. Constraint based motion planning of deformable robots. *IEEE Conf. on Robotics and Automation*.

GAYLE, R., SUD, A., LIN, M., AND MANOCHA, D. 2007. Reactive deformation roadmaps: Motion planning of multiple robots in dynamic environments. In *Proc IEEE International Conference on Intelligent Robots and Systems*.

HELBING, D., BUZNA, L., AND WERNER, T. 2003. Self-organized pedestrian crowd dynamics and design solutions. *Traffic Forum 12*.

HELBING, D., BUZNA, L., JOHANSSON, A., AND WERNER, T. 2005. Self-organized pedestrian crowd dynamics: experiments, simulations and design solutions. *Transportation science*, 1–24.

HOOGENDOORN, S. P., LUDING, S., BOVY, P., SCHRECKLENBERG, M., AND WOLF, D. 2000. *Traffic and Granular Flow*. Springer.

JAKOBSEN, T. 2001. Advanced character physics. In *Game Developer's Conference*.

KAMPHUIS, A., AND OVERMARS, M. 2004. Finding paths for coherent groups using clearance. *Proc. of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*.

KHATIB, O. 1986. Real-time obstable avoidance for manipulators and mobile robots. *IJRR 5*, 1, 90–98.

LAMARCHE, F., AND DONIKIAN, S. 2004. Crowd of virtual humans: a new approach for real-time navigation in complex and structured environments. *Computer Graphics Forum 23*, 3 (Sept).

LAVALLE, S. M. 2006. *Planning Algorithms*. Cambridge University Press (also available at http://msl.cs.uiuc.edu/planning/).

LERNER, A., CHRYSANTHOU, Y., AND LISCHINSKI, D. 2007. Crowds by example. *Computer Graphics Forum (Proceedings of Eurographics) 26*, 3.

LI, Y., AND GUPTA, K. 2007. Motion planning of multiple agents in virtual environments on parallel architectures. In *ICRA*, 1009–1014.

LOSCOS, C., MARCHAL, D., AND MEYER, A. 2003. Intuitive crowd behaviour in dense urban environments using local laws. *Theory and Practice of Computer Graphics (TPCG'03)*.

MUSSE, S. R., AND THALMANN, D. 1997. A model of human crowd behavior: Group inter-relationship and collision detection analysis. *Computer Animation and Simulation*.

PELECHANO, N., O'BRIEN, K., SILVERMAN, B., AND BADLER, N. 2005. Crowd simulation incorporating agent psychological models, roles and communication. *First International Workshop on Crowd Simulation*.

PETTRE, J., LAUMOND, J.-P., AND THALMANN, D. 2005. A navigation graph for real-time crowd animation on multilayered and uneven terrain. *First International Workshop on Crowd Simulation*.

QUINLAN, S., AND KHATIB, O. 1993. Elastic bands: Connecting path planning and control. *Proc. of IEEE Conf. on Robotics and Automation*.

REYNOLDS, C. W. 1987. Flocks, herds, and schools: A distributed behavioral model. *Comput. Graph. 21*, 4, 25–34. Proc. SIGGRAPH '87.

REYNOLDS, C. 2006. Big fast crowds on ps3. In *sandbox '06: Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, ACM Press, New York, NY, USA, 113–121.

RODRIGUEZ, S., LIEN, J.-M., AND AMATO, N. M. 2006. Planning motion in completely deformable environments. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (May).

SCHRECKKENBERG, M., AND SHARMA, S. D. 2001. *Pedestrian and Evacuation Dynamics*. Springer.

SHAO, W., AND TERZOPOULOS, D. 2005. Autonomous pedestrians. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 19–28.

SUD, A., GOVINDARAJU, N., GAYLE, R., AND MANOCHA, D. 2006. Interactive 3d distance field computation using linear factorization. In *Proc. ACM Symposium on Interactive 3D Graphics and Games*, 117–124.

SUD, A., ANDERSEN, E., CURTIS, S., LIN, M., AND MANOCHA, D. 2007. Real-time path planning for virtual agents in dynamic environments. *Proc. of IEEE VR*.

SUGIYAMA, Y., NAKAYAMA, A., AND HASEBE, K. 2001. 2-dimensional optimal velocity models for granular flows. In *Pedestrian and Evacuation Dynamics*, 155–160.

SUNG, M., GLEICHER, M., AND CHENNEY, S. 2004. Scalable behaviors for crowd simulation. *Computer Graphics Forum 23*, 3 (Sept).

SUNG, M., KOVAR, L., AND GLEICHER, M. 2005. Fast and accurate goal-directed motion synthesis for crowds. *Proc. of SCA 2005*, 291–300.

THALMANN, D., O'SULLIVAN, C., CIECHOMSKI, P., AND DOBBYN, S. 2006. *Populating Virtual Environments with Crowds*. Eurographics 2006 Tutorial Notes.

TREUILLE, A., COOPER, S., AND POPOVIC, Z. 2006. Continuum crowds. *Proc. of ACM SIGGRAPH*.

TU, X., AND TERZOPOULOS, D. Artificial fishes: Physics, locomotion, perception, behavior. In *Proceedings of SIGGRAPH '94*, A. Glassner, Ed., 43–50.

VERLET, L 1967. Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. *Phys. Rev.*, 159, 98-103.

YANG, Y., AND BROCK, O. 2006. Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation. *Proceedings of Robotics: Science and Systems* (August).

ZUCKER, M., KUFFNER, J., AND BRANICKY, M. 2007. Multipartite rrts for rapid replanning in dynamic environments. *Proc. IEEE Int. Conf. on Robotics and Automation*.

Figure 6: Crowd simulation in an urban landscape: A street intersection in a virtual city with 924 *buildings,* 50 *moving cars as dynamic obstacles and* 1,000 *pedestrians. We show a sequence of four snapshots of a car driving through the intersection. As the car approaches a lane of pedestrians (top), the lane breaks (middle two images) and the pedestrians re-route using alternate links on the adaptive roadmap. Once the car leaves the intersection (bottom) the pedestrians reform the lane using the adaptive roadmap. We are able to perform navigation of* 1,000 *pedestrians in this extremely complex environment at* 16*fps on a* 3*Ghz PC.*

Figure 7: Sequence of 4 snapshots from Tradeshow demo. The environment contains 511 booths with 110K polygons. The agents move toward different booths and avoid each other using link bands.

8.7 Interactive Motion Correction and Object Manipulation

Interactive Motion Correction and Object Manipulation

Ari Shapiro*
University of California, Los Angeles

Marcelo Kallmann†
Computer Graphics Lab
University of California, Merced

Petros Faloutsos‡
University of California, Los Angeles

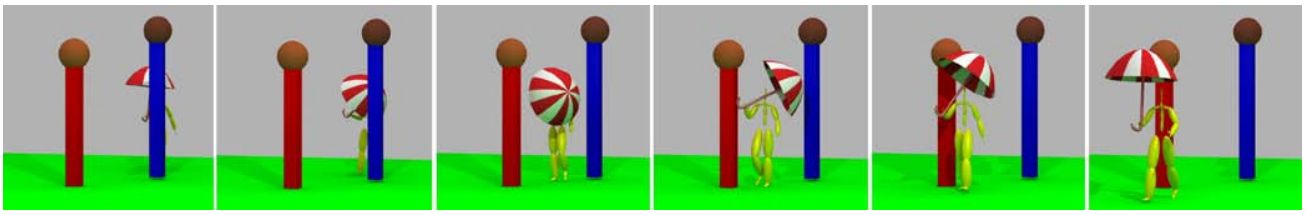


Figure 1: In order to avoid collisions between the umbrella and the two posts the arm motion was planned in sync with a walking sequence.

Abstract

Editing recorded motions to make them suitable for different sets of environmental constraints is a general and difficult open problem. In this paper we solve a significant part of this problem by modifying full-body motions with an interactive randomized motion planner. Our method is able to synthesize collision-free motions for specified linkages of multiple animated characters in synchrony with the characters' full-body motions. The proposed method runs at interactive speed for dynamic environments of realistic complexity. We demonstrate the effectiveness of our interactive motion editing approach with two important applications: (a) motion correction (to remove collisions) and (b) synthesis of realistic object manipulation sequences on top of locomotion.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques.

Keywords: character animation, motion capture, motion editing, virtual humans, object manipulation

1 Introduction

Techniques based on libraries of motion capture produce the most realistic animations of virtual humans to date. However one of the main drawbacks of such techniques is their inability to offer, without additional mechanisms, any variations from the exact recorded motions. The virtual environments where the playback of motion occurs differ from the environment in which the motion was captured. Virtual environments often contain obstacles and characters that were not present in the motion capture studio. The motion capture data must be modified to accommodate the virtual environment in order to preserve the appearance of realism. For example, a virtual human may need to swing its arm away to avoid a virtual object or lift its leg higher in order to step over an obstacle that lies on the ground.

Designing in advance all required motions for a given virtual environment or scenario involves tedious and time-consuming design work. Furthermore, it is not practical to rely on pre-designed motions when object grasping and manipulation are required for arbitrarily located objects in arbitrary scenes. A recorded motion captured of a person catching a ball with two hands at chest-level will not be effective for catching a different sized ball with one hand at waist-level. The problem is even more complex when the character, the target and the obstacles in the environment move. This is the problem that we address in this work.

We introduce a new motion editing approach that combines recorded motions with motion planning in order to produce realistic animations of characters avoiding and manipulating moving objects in dynamic environments. The approach has two applications: 1) *Motion correction*, where a prerecorded motion played on a virtual human is automatically corrected to respect obstacles in the virtual environment, and 2) *Object manipulation*, where virtual humans are instructed to grab, drop and touch various objects, either moving or fixed, while playing back recorded motion and respecting both fixed and moving obstacles As an example of motion correction, Figure 1 shows a walking character manipulating an umbrella so that it can walk through the two posts without hitting them. The motion of the character's arms was synthesized interactively by our planner on top of the original locomotion.

Our approach is based on a motion planner that generates collision-free motions for specified linkages of a character, making them reach desired targets at specified times. In order to address the time constraints, the planner considers time as one additional dimension in the search space. Therefore moving targets, moving obstacles and synchronization with keyframe animations can all be taken into account together. Our method proves to be very efficient for producing object manipulation sequences as well as for adjusting motions to avoid collisions with obstacles. We are also able to control stylistic aspects of the resulting motions by customizing the search heuristics that our planner employs when exploring the space of possible configurations. Furthermore, we employ an anatomically meaningful skeleton parameterization that helps us enforce realistic limits on the motion of the character's joints. To demonstrate the effectiveness of our approach, we present several complex examples solving highly dynamic tasks.

*e-mail: ashapiro@cs.ucla.edu
†e-mail:mkallmann@ucmerced.edu
‡e-mail:pfal@cs.ucla.edu

Contributions. We present: (a) a hybrid motion synthesis solution that combines recorded motions with motion planning, and (b) how to control the obtained results by choosing different configuration sampling strategies for the motion planner.

In general our method can be applied to generic task-oriented motion editing problems respecting collision-free and spatio-temporal constraints. It requires about one second of time to compute motions of average complexity making it suitable for interactive use.

2 Related Work

Motion planning research for animated characters has traditionally been segmented into two different areas; 1) full-body motion planning for the purpose of locomotion, and 2) reach and arm planning for the purpose of object manipulation.

Full-body locomotion planning. Motion synthesis, whose primary goal is to generate plausible motion that adheres to given constraints, such as a movement path, has been explored by past research [Lau and Kuffner 2005; Kwon and Shin 2005; Lai et al. 2005; Kovar et al. 2002; Arikan and Forsyth 2002; Choi et al. 2002]. The goal of our method differs in that rather than generating locomotion sequences as these other methods do, it targets arm and leg movements corrections that either adhere to constraints in the virtual environments or allow object manipulations not included in the original motion. Our method uses the time dimension in planning in order to handle dynamic obstacles. Of particular note, [Lau and Kuffner 2005] uses the time dimension in order to plan for dynamic obstacles. In contrast, our method uses the time dimension in concert with a locomotion clip to handle object manipulation and arm linkage adjustments, rather than to generate the underlying locomotion clip. Also, our method does not require a preprocessed set of motion clips that have already been segmented and transformed into an FSM, and can use any motion clip that propels the animated character. The inclusion of the time dimension in a motion planner has also already been proposed in Robotics [Hsu et al. 2002]; however, we present a planner that uses the time dimension for a specific set of tasks; to plan the motions of some limbs in synchronization with external motions affecting the same character.

Given the complementary nature of our work to the research in locomotion synthesis, our method could be enhanced with the inclusion of such methods as a preliminary motion editing stage. The locomotion generation method would create a motion clip, which would subsequently be used as input into our system which would in turn edit and arm and leg movements of the resulting animation.

Of relation to our work, [Pettré et al. 2003] uses a two-stage locomotion planner to first plan the movement of the character, then correct the upper body for collisions. Our method differs in it can handle simultaneously moving targets, moving obstacles and moving characters.

Reach and arm planning Since the first application of motion planning to computer animation [Koga et al. 1994] which included grasp planning, several motion planning methods have been proposed specifically addressing human-like characters manipulating objects.

One approach is to search for a sequence of intermediate suitable hand locations in the workspace and use Inverse Kinematics (IK) to derive arm postures reaching each intermediate hand location [Yamane et al. 2004; Liu and Badler 2003; Bandi and Thalmann 2000]. The final valid motion is obtained through interpolation of the postures. Another approach is to search directly in the configuration space [Koga et al. 1994; Kuffner and Latombe 2000], yielding simpler algorithms (not requiring IK during the search) that can address the entire solution space. As the search space grows exponentially with its dimension, simplifying control layers can be specified for synthesizing whole-body motions [Kallmann et al. 2003].

Hardware acceleration has been used to generate arm linkage paths for manipulation purposes [Liu and Badler 2003] for stationary characters. Our work is similar in that we use a similar analytically-based IK algorithm [Tolani and Badler 1996], however our method works with both non-stationary characters as well as moving objects.

A key feature of our method is efficiency. We choose to perform the search in the configuration space, relying on a *Rapidly-Exploring Random Tree* (RRT) planner [LaValle 1998; LaValle and Kuffner 2000] in its bidirectional version [Kuffner and LaValle 2000] along with adding the time dimension to the search. This allows our method to be used interactively by an animator.

The problem of synthesizing human-like arm movements is addressed by [Yamane et al. 2004] by using examples from motion capture examples to generate velocity profiles of natural arm motions. Unlike this method, our method is able to plan motions that involve moving feet and moving characters. Our method does not use example motions and thus our arm movements are less likely to look as natural. However, we are able to generate a solution with much greater speed, on the order of seconds rather than minutes, and thus are much better suited for interactive use.

3 Problem Formulation

We represent the character as an articulated figure composed of hierarchical joints. Let C^F be the space of all full configurations of the character. Let $c^f = (p, q^1, \ldots, q^{r-1}, q^r, \ldots, q^n) \in C^F$ be one full configuration of the character, where $p \in \mathbb{R}^3$ is the position of the root joint, and $q^i \in \mathbb{S}^3, i \in \{1, \ldots, n\}$, is the rotational value of the i^{th} joint, in quaternion representation. The components of c^f are organized in such way that the rear part $c^p = (q^r, \ldots, q^n) \in C^P$ denotes the degrees of freedom (DOFs) controlled by the planner, and the fore part $c^m = (p, q^1, \ldots, q^{r-1}) \in C^M$ contains the remaining DOFs, which are controlled by an external motion. Therefore $C^F = C^M \times C^P$ and $c^f = (c^m, c^p)$.

An external motion controller affecting the DOFs in C^M is defined as a time-varying function $m^m(t) = (p(t), q^1(t), \ldots, q^{r-1}(t))$. Therefore $p(t) \in \mathbb{R}^3$ describes the translational motion of the root joint, and $q^i(t) \in \mathbb{S}^3$, $1 \leq i < r$, describes the rotational motion of the affected joints in their local frame coordinates. We assume that m^m is completely defined over a given time interval $I \subset \mathbb{R}$, as is the case for motions defined as keyframe animations. We furthermore assume that $m^m(t)$ is collision-free for all $t \in I$.

In order to take into account moving objects in the environment, all object motions are required to be parameterized by the same time parameter $t \in I$ of motion m^m. We therefore construct a function $w(t)$, which sets the state of the world to the desired time t.

Let $c^p_{init} \in C^P$ and $c^p_{goal} \in C^P$ be initial and goal configurations specified to be reached at times t_{init} and t_{goal} respectively, $[t_{init}, t_{goal}] \subset I$. Our search space includes the time dimension and is defined as $C^S = C^P \times [t_{init}, t_{goal}]$. Configuration $c^s = (c^p, t) \in C^S$ is *valid* if the character's posture $(m^m(t), c^p) \in C^F$ respects joint limits and is collision-free when the world's state is $w(t)$. We denote by C^S_{free} the subspace of all valid configurations in C^S.

Consider now $c^s_{init} = (c^p_{init}, t_{init})$ and $c^s_{goal} = (c^p_{goal}, t_{goal})$ be initial and goal configurations in C^S_{free}. Our problem is then reduced to finding a path in C^S_{free} connecting c^s_{init} to c^s_{goal}.

Our planner solves the problem by searching for a sequence of valid landmarks $c^s_i = (q^r_i, \ldots, q^n_i, t_i) \in C^S$, $1 \leq i \leq k$, such that:

1. $c^s_1 = c^s_{init}$, and $c^s_k = c^s_{goal}$,
2. the time parameter is monotone, i.e., $t_i < t_{i+1}$, $1 \leq i < k$,
3. for all pairs of adjacent landmarks (c^s_i, c^s_{i+1}), $1 \leq i < k$, the motion obtained through interpolation between c^s_i and c^s_{i+1} remains in C^S_{free}.

Let q^j_i, $r \leq j \leq n$, be the j^{th} quaternion of landmark c^s_i, $1 \leq i \leq k$. Motion $m^p(t)$ can then be constructed as:

$$m^p(t) = (q^r(t), \ldots, q^n(t)),$$

$$\text{with } q^j(t) = slerp(q^j_i, q^j_{i+1}, \tfrac{(t-t_i)}{t_{i+1}-t_i}),$$

$$\text{and } t_i \leq t < t_{i+1}.$$

The composite motion $m^f(t) = (m^m(t), m^p(t))$, $t \in [t_{init}, t_{goal}]$, will be a valid motion satisfying constraints c^p_{init} and c^p_{goal} at times t_{init} and t_{goal} respectively, and therefore solving our problem. We present in the following section our motion planner, which finds the sequence of landmarks c^s_i required for constructing $m^p(t)$.

4 Synchronized Motion Planner

The goal of our planner is to find a sequence of landmarks connecting c^s_{init} to c^s_{goal} in C^S_{free}. For solving this problem we propose a bidirectional RRT planner algorithm that supports landmarks with monotone time parameters.

4.1 Algorithm

Algorithm 1 summarizes our implementation. Two search trees T_{init} and T_{goal} are initialized having c^s_{init} and c^s_{goal} respectively as root nodes, and are sent to the planner. The trees are iteratively expanded by adding valid landmarks. When a valid connection between the two trees can be concluded, a successful path in C^S is found. Otherwise when a given amount of time has passed, the algorithm fails.

Algorithm 1 SYNCPLANNER (T_1, T_2)

```
1:  while elapsed time ≤ maximum allowed time do
2:    c^s_sample ← SAMPLECONFIGURATION().
3:    c^s_1 ← closest node to c^s_sample in T_1.
4:    c^s_2 ← closest node to c^s_sample in T_2.
5:    if INTERPOLATIONVALID (c^s_1, c^s_2) then
6:      return MAKEPATH (root(T_1), c^s_1, c^s_2, root(T_2)).
7:    end if
8:    c^s_exp ← NODEEXPANSION (c^s_1, c^s_sample, ε).
9:    if c^s_exp ≠ null and INTERPOLATIONVALID (c^s_exp, c^s_2) then
10:     return MAKEPATH (root(T_1), c^s_exp, c^s_2, root(T_2)).
11:   end if
12:   Swap T_1 and T_2.
13: end while
14: return failure.
```

Line 2 in algorithm 1 requires a sampling routine in C^S for guiding the search for successful landmarks. Our sampling routine is customized for human-like characters and is explained in detail in section 4.2.

Lines 3 and 4 require searching for the closest configurations in each tree. A linear search suffices as the trees are not expected to grow much. The metric used is a weighted sum of time and arm posture metrics. Let c^s_1 and c^s_2 be two configurations in C^S, such that $c^s_j = (q^r_j, \ldots, q^n_j, t_j)$, $j \in \{1,2\}$. Let p^i_j be the position (in global coordinates) of the joint affected by rotation q^i_j, $r \leq i \leq n$. The distance between c^s_1 and c^s_2 is computed as:

$$dist(c^s_1, c^s_2) = w_t|t_1 - t_2| + w_a \max_i \|p^i_1 - p^i_2\|,$$

where w_t and w_a are the desired weights.

Lines 5 and 9 check if the interpolation between two configurations is valid. It is considered valid if two tests are successful:

1. the configuration in T_{init} has to have its time component smaller than the configuration in T_{goal},
2. the interpolation has to remain in C^S_{free}.

The simplest approach for testing item 2 above is to perform several discrete collision checks along the interpolation between the two configurations. In order to promote early detection of collisions, we use the popular recursive bisection for determining where to perform the discrete tests, until achieving a desired resolution. Note that continuous tests not requiring a resolution limit are available and can be integrated [Schwarzer et al. 2002].

The algorithm tests at lines 5 and 9 if a valid connection between T_1 and T_2 has been found, and in such cases a path in C^S is computed and returned as a valid solution. The path is computed using routine MAKEPATH$(c^s_1, c^s_2, c^s_3, c^s_4)$ (lines 6 and 10), which connects the tree branch joining c^s_1 with c^s_2 to the tree branch joining c^s_3 with c^s_4, with the path segment obtained with the interpolation between c^s_2 and c^s_3.

The node expansion in line 8 uses c^s_{sample} as growing direction and computes a new configuration c^s_{exp} as follows:

$$c^s_{exp} = interp(c^s_1, c^s_{sample}, t), \text{ where}$$

$$t = \varepsilon/d,\ d = dist(c^s_1, c^s_{sample}).$$

Null is returned in case the expansion is not valid, i.e. if the interpolation between c^s_1 and c^s_{exp} is not valid or if the time component in the configurations do not respect the monotone condition. Otherwise c^s_{exp} is linked to c^s_1, making the tree grow by one node and one edge. The factor ε represents the incremental step taken during the search. Large steps make the trees grow quickly but with more difficulty in capturing the free configuration space around obstacles. Inversely, too small values generate roadmaps with too many nodes, slowing down the algorithms.

Path Smoothing. When a solution is found, a final step for smoothing the path is required. We use here the popular approach of applying several linearization steps. Each linearization consists of selecting two random configurations c^s_a and c^s_b along the solution path in C^S (not necessarily landmarks) and replacing the subpath between c^s_a and c^s_b by the straight interpolation between them, if the replacement is still a valid path. Note that the time component in c^s_a and c^s_b are as well interpolated and smoothed. The process is repeated until valid replacements are difficult to find or until a time threshold is reached. This simple process works well in practice and has both the effect of smoothing and shortening the path, which are obvious properties expected in natural motions.

4.2 Configuration Sampling

The sampling routine guides the whole search and is of extreme importance in determining the quality of a solution and how fast it is found. It is therefore important to define meaningful joint parameterizations, joint limits and search heuristics for both reducing the search space and guiding the search to more realistic postures. We pay particular attention here to the joints of the arm linkages due to their importance for object manipulation.

Joint Parameterization. The first step for ensuring anatomically plausible postures is to impose meaningful joint range limits on the articulations of the skeleton. For anatomical articulations with a 3 DOF rotation, e.g. the shoulder, we use the natural swing-and-twist decomposition [Grassia 1998]. The remaining joints are either parameterized with Euler angles or by a swing rotation.

For instance in the arm linkages the swing-and-twist decomposition is used to model the shoulder (3 DOFs). The elbow has flexion and twist rotations defined with two Euler angles (2 DOFs), and the wrist has a swing rotation (2 DOFs) parameterized exactly as a swing-and-twist, however considering the twist rotation to be always 0. The linkages of the legs are similarly parameterized.

Joint Limits. The swing parameterization allows the use of spherical polygons [Korein 1985] for restricting the swing motion. Spherical polygons can be manually edited for defining a precise bounding curve. However we follow a simpler, more efficient, and still acceptable solution for bounding swing limits based on spherical ellipses [Grassia 1998]. In this case, a swing rotation can be checked for validity simply by replacing the axis-angle parameters into the ellipse's equation. The twist and flexion rotations of the remaining DOFs are correctly limited by minimum and maximum angles.

Collision Detection. In order to achieve complex collision-free motions, we take into account the full geometries of the characters and the environment when checking for collisions. The VCollide package [Gottschalk et al. 1996] is employed for querying if body parts self-intersect or intersect with the environment.

Search Heuristics. We control the overall quality of the planned motions by properly adjusting sampling heuristics. Uniformly sampling valid postures has the effect of biasing the search toward the free spaces. For example, in several cases where the character manipulates objects, there are obstacles in front of the character and larger volumes of free space are located at the sides of the character. Although these are indeed valid areas, realistic manipulations are mainly carried out in the smaller free spaces in front of the character.

A simple correction technique for such cases consists of highly biasing the sampling towards the bent configuration of the elbow. This has the effect of avoiding solutions with the arm outstretched, resulting in more natural motions. As we perform a bidirectional search, it also contributes to decomposing the manipulation in two distinct phases: bringing the arm closer to the body and then extending it towards the goal. Our biasing method starts sampling the elbow flexion DOF with values in the interval between 100% and 90% of flexion, and as the number of iterations grow, the sampling interval gets larger until reaching the joint limits.

For other less important joints, e.g. the wrist or spine joints if used, the sampling is also similarly biased to a smaller range than their validity range, resulting in more pleasant postures as these joints are usually secondarily used for avoiding obstacles.

The sampling routine can be even interactively customized by choosing different values for the sampling intervals used for sampling each considered DOF. For example by adjusting the intervals of the shoulder DOFs we are able to control the overall location of the obtained arm motion. The top row of Figure 2 shows an example where the x-component of the shoulder swing was sampled between 50 and 100 degrees, generating only relatively low arm postures during the search. In the solution shown in the bottom row however, we choose to sample higher arm postures. Such example illustrates that we are able to control the overall quality of the motion and avoid repetitive results.

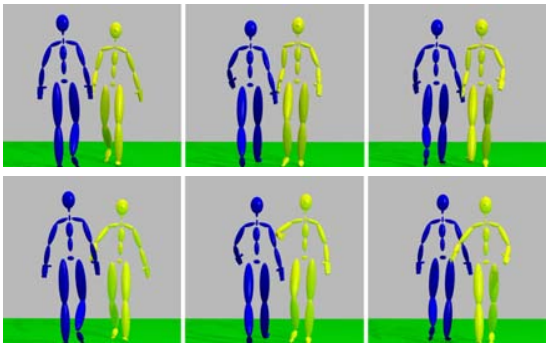


Figure 2: Two alternative solutions for correcting arm collisions obtained by choosing different sampling heuristics.

Final Sampling Routine. The final sampling routine can be summarized as follows:

1. Configuration $c^s_{rand} = (c^p_{rand}, t_{rand}) \in C^S$ is generated having the values in c^p_{rand} randomly sampled in the described parameterizations based on swings, twists and Euler angles; inside individually placed range limits and following the appropriate sampling heuristics. The time component t_{rand} is uniformly sampled in $[t_{init}, t_{goal}]$.
2. The state of the world is set with $w(t_{rand})$ and configuration $m^m(t_{rand})$ is applied to the character.
3. Configuration c^p_{rand} is applied to the character.
4. Finally the positions of all objects are updated and tested for collisions; if no collisions are found c^s_{rand} is returned as a successful valid configuration, otherwise the sampling routine returns to step 1.

5 Inverse Kinematics

Inverse Kinematics is an important component of our overall method. Although the planner does not require the use of IK during its execution, our IK allows us to easily (and interactively) specify goal arm and leg postures to be used as input to the planner. In particular for interactive grasping, the use of IK allows the user to define goal arm postures for the planner on-line, by simply selecting goal hand positions in the workspace.

In order to obtain realistic and fast results, we implemented an analytical IK formulation [Tolani and Badler 1996] that produces joint values directly in our arm and leg parameterizations with meaningful joint limits based on swings and twists. Note that for each arm or leg, there are 7 DOFs to be determined for reaching a given hand position and orientation goal. The problem is under-determined and the missing DOF is modeled as the *swivel angle*, which is an extra parameter specifying the desired rotation of the elbow (or knee) around the wrist-shoulder (or ankle-hip) axis.

We have furthermore integrated in the IK a simple search strategy that automatically searches for a swivel angle leading to a valid (and therefore collision-free) configuration. Equipped with such automatic posture search, the IK and the planner are able to produce complex collision-free animations for reaching given hand targets interactively.

We start solving the IK with the desired initial swivel angle, which is usually extracted from the current character posture. Then, the posture given by the IK solver is checked for validity. If the posture is not valid, the swivel angle is incremented and decremented by δ and the two new postures given by the IK solver are again checked for validity. If a valid posture is found the process successfully stops. Otherwise, if given minimum and maximum swivel angles are reached, failure is returned. Faster results are achieved in a greedy fashion, i.e. when Δ increases during the iterations. As the search range is small this simple process is very efficient and the whole process can be limited to a few number of tests. Note that both joint limits and collisions are avoided in an unified way.

6 Applications and Results

We have integrated the methods described in this paper in the DANCE animation system [Shapiro et al. 2005]. Multiple arms and leg targets can be specified and solved by our planner interactively. Targets can be dynamic and/or attached to any objects or body parts. Characters can be instructed to grab, drop and move objects. Several tasks can be specified simultaneously and synchronized with arbitrary keyframe motions applied to the characters.

In the remainder of this section we present several results obtained with our system. We group them by two key applications that demonstrate the versatility and the effectiveness of our approach. For a better presentation of the results we refer the reader to the accompanying video and our website (removed for anonymity).

6.1 Motion Correction

Our planner introduces an effective way to correct portions of motions that are found to produce collisions with new objects in the environment or with new objects attached to the character. Such situations are common when reusing motions in new environments or new characters. Our planner is able to search for an alternative motion for the problematic limb which is both valid and in synchronization with the original motion and any moving objects.

Let m be a given motion affecting the full configuration space C^F of the character. We want to correct a portion of m that was found to obtain collisions. For solving this kind of problem, we define times t_{init} and t_{goal} such that interval $[t_{init}, t_{goal}]$ spans the problematic period of the motion.

Let m be decomposed in two parts, such that $m(t) = (m^m(t), m^p(t)) \in C^M \times C^P$. The problem is then solved by planning a new path between $(m^p(t_{init}), t_{init})$ and $(m^p(t_{goal}), t_{goal})$ in C^S. If the planner is successful, the result will be a collision-free motion that is used to replace m^p during interval $[t_{init}, t_{goal}]$.

We present several examples in this paper. Figure 4 (a) presents a valid walking motion that becomes invalid when an umbrella is attached to the right hand of the character. The umbrella collides with the post in several frames of the sequence. Figure 4(b) presents the corrected motion after the planner is applied to produce a new synchronized motion for the joints of the right arm. The same walking motion was also successfully corrected by our planner in a new environment containing two posts (Figure 1). Other correction examples are shown in Figure 2 and Figure 4(c,d).

6.2 Interactive Object Manipulation

Object manipulation tasks for moving characters can be complex and computationally expensive to synthesize. Our planner generates realistic results of such highly complex tasks by synchronizing synthesized arm motions with locomotion sequences. For instance in Figure 4(e) we generate a motion where the character grasps a dynamic target through a moving ring while under the influence of an *idle* motion affecting its body. Figure 4(f) shows an even more complex motion where the character is asked to solve the same task but while transitioning from walking to running. Figure 4(g) shows a character walking and at the same time grabbing the hat of another walking character.

For this kind of problem we first specify hand targets on the objects to be grasped. Let $h = \{p, q\}$, $h \in \mathbb{R}^3 \times \mathbb{S}^3$, be a *hand target* described as a target position and orientation for the wrist joint of the character in global coordinates, to be reached at a given time t_b.

Let again $m(t) = (m^m(t), m^p(t)) \in C^M \times C^P$ be a motion as described in Section 6.1. We want now to modify motion m such that at time t_b the character wrist joint is located at the given hand target h, and as the modified motion has to be performed in a cluttered environment, it has to be collision-free.

We now determine times t_a and t_c such that $t_a < t_b < t_c$. Then the problem is solved in two steps: first a path in C^S is planned between $(m^p(t_a), t_a)$ and (c_h^p, t_b) and then a second path in C^S is planned between (c_h^p, t_b) and $(m^p(t_c), t_c)$. Configuration $c_h^p \in C^P$ is determined by employing our IK (Section 5), in order to determine the best arm configuration that reaches the hand target h.

The sequences in Figure 4(h-j) show several complex manipulation examples. The obtained results show realistic motions where planned arm manipulation sequences are perfectly synchronized with the walking motion. In this example, 10 planned sequences were used for synchronizing 5 different object manipulations: grasping a piece of cheese from inside a box, dropping it on the table, grasping a hat with the right hand, turning off the lights with the left hand and then placing the hat on the head.

We have also implemented a system to interactively instruct a character to reach for arbitrarily located objects, in synchronization with an on-line locomotion planner. We therefore compute the final motion in two steps: first a path is planned such that the character arrives close enough to the object to grasp with the hand. A motion captured sequence is then deformed to fit the computed path, and before the locomotion is finished, we compose a synchronized arm motion with the locomotion.

7 Discussion

One important characteristic of our method is the random nature of the planner. It ensures that the obtained motions are always different, greatly improving the realism in interactive applications of autonomous characters. At the same time, we are also able to control the overall aspect of the obtained results by choosing different body motions to synchronize with and search heuristics (Figure 2).

The performance of the planner greatly depends on the complexity of the environment. For instance in the complex scenario of Figure 4(h-j), the collision detection is handling 30K triangles and the

planner took about 2 seconds to both compute and smooth each of the planned motions. In the simpler environments the performance is about two times faster.

Limitations and Extensions. Although our results are realistic, further processing could still be employed. For instance, dynamic filters could be applied for ensuring the balance of the character. However, this would penalize the overall performance of the method. We chose not to employ a more time-consuming method, such as those described by [Yamane et al. 2004] in the interests of speed. Our method currently serves as an interactive application whereby an animator can quickly edit and change the motion within seconds to his or her tastes.

Although the examples presented here show the planner is mainly applied to arms and legs, it can also be applied to any set of open linkages. It can be as well employed sequentially, for example for synchronizing the motions of several limbs: first, the motion of one limb is planned in synchronization with the given external motions, resulting in a new composite motion. Then, a second limb motion can be synchronized with the previously obtained motion. The process can be repeated until all limbs are planned and synchronized. The result achieved is a decoupled priority-based (due to the chosen order) planning process. Note that limbs may belong to different characters, as in the example shown in Figure 3.

The examples here could also use longer linkages on the same character, such as those that include the arm and torso to accommodate bending and twisting of the waist and trunk. The risk of using larger IK linkages is the deteriorating effect on the resulting realism that such a solution would provide. Since many IK solutions do not take into account physics or changes to the COM or momentum of the body, the longer the IK chain used, the less realistic the final motion will be. This could be overcome by either using an additional dynamic filtering as a postprocessing step, or employing an IK that accommodates changes to the rest of the body, such as shown by [Grochow et al. 2004].

Figure 3: The motion of the character on the left side was planned after the motion of the character on the right side, achieving synchronized simultaneous graspings.

8 Conclusion

We have presented a new approach for motion editing based on planning motions in synchronization with pre-designed (or recorded) motion sequences and moving objects. In general, our method is able to solve arbitrary spatio-temporal constraints among obstacles and takes into account dynamic environments.

By relying on a hybrid approach, we are able to address the difficult constraints imposed by object manipulations, achieve realistic results and still leave space for designers to customize and personalize the underlying motion sequences.

9 Acknowledgements

The work in this paper was partially supported by NSF under contract CCF-0429983. We would also like to thank Intel Corp., Microsoft Corp., Alias Wavefront and ATI Corp. for their generous support through equipment and software grants.

References

ARIKAN, O., AND FORSYTH, D. A. 2002. Interactive motion generation from examples. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 483–490.

BANDI, S., AND THALMANN, D. 2000. Path finding for human motion in virtual environments. *Computational Geometry: Theory and Applications 15*, 1-3, 103–127.

CHOI, M. G., LEE, J., AND SHIN, S. Y. 2002. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics 22*, 2, 182–203.

GOTTSCHALK, S., LIN, M. C., AND MANOCHA, D. 1996. Obbtree: A hierarchical structure for rapid interference detection. *Computer Graphics SIGGRAPH'96 30*, Annual Conference Series, 171–180.

GRASSIA, S. 1998. Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools 3*, 3, 29–48.

GROCHOW, K., MARTIN, S., HERTZMANN, A., AND POPOVI, Z., 2004. Style-based inverse kinematics.

HSU, D., KINDEL, R., LATOMBE, J., AND ROCK, S. 2002. Randomized kinodynamic motion planning with moving obstacles. *International Journal of Robotics Research 21*, 3, 233–255.

KALLMANN, M., AUBEL, A., ABACI, T., AND THALMANN, D. 2003. Planning collision-free reaching motions for interactive object manipulation and grasping. *Computer graphics Forum (Proceedings of Eurographics'03) 22*, 3 (September), 313–322.

KOGA, Y., KONDO, K., KUFFNER, J. J., AND LATOMBE, J.-C. 1994. Planning motions with intentions. In *Proceedings of SIGGRAPH'94*, ACM Press, 395–408.

KOREIN, J. U. 1985. *A Geometric Investigation of Reach.* The MIT Press.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 473–482.

KUFFNER, J. J., AND LATOMBE, J.-C. 2000. Interactive manipulation planning for animated characters. In *Proceedings of Pacific Graphics'00*. poster paper.

KUFFNER, J. J., AND LAVALLE, S. M. 2000. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings of IEEE Int'l Conference on Robotics and Automation (ICRA'00)*.

KWON, T., AND SHIN, S. Y. 2005. Motion modeling for on-line locomotion synthesis. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 29–38.

LAI, Y.-C., CHENNEY, S., AND FAN, S. 2005. Group motion graphs. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

LAU, M., AND KUFFNER, J. 2005. Behavior planning for character animation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

LAVALLE, S., AND KUFFNER, J., 2000. Rapidly-exploring random trees: Progress and prospects. In Workshop on the Algorithmic Foundations of Robotics.

LAVALLE, S. 1998. Rapidly-exploring random trees: A new tool for path planning. Tech. Rep. 98-11, Iowa State University, Computer Science Department, October.

LIU, Y., AND BADLER, N. I. 2003. Real-time reach planning for animated characters using hardware acceleration. In *Proceedings of Computer Animation and Social Agents (CASA'03)*, 86–93.

PETTRÉ, J., LAUMOND, J.-P., AND SIMEÓN, T. 2003. A 2-stages locomotion planner for digital actors. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 258–264.

SCHWARZER, F., SAHA, M., AND LATOMBE, J.-C. 2002. Exact collision checking of robot paths. In *Proceedings of the Workshop on Algorithmic Foundations of Robotics (WAFR'02)*.

SHAPIRO, A., FALOUTSOS, P., AND NG-THOW-HING, V. 2005. Dynamic animation and control environment. In *GI '05: Proceedings of the 2005 conference on Graphics interface*, Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 61–70.

TOLANI, D., AND BADLER, N. 1996. Real-time inverse kinematics of the human arm. *Presence 5*, 4, 393–401.

YAMANE, K., KUFFNER, J. J., AND HODGINS, J. K. 2004. Synthesizing animations of human manipulation tasks. *ACM Transactions on Graphics (Proceedings of SIGGRAPH'04) 23*, 3, 532–539.

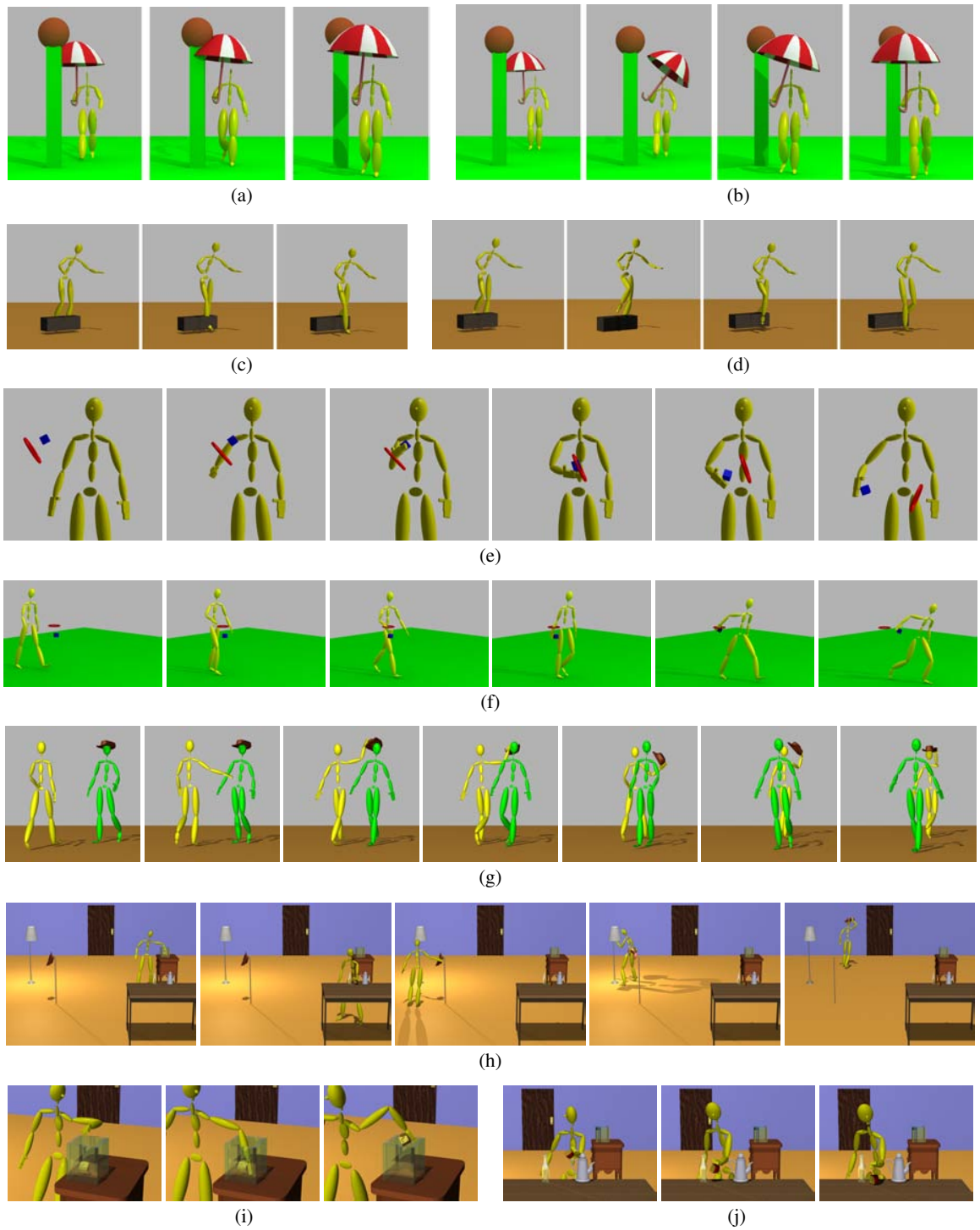


Figure 4: Sequences (a) and (c) have collisions and are corrected by our planner, which produced (b) and (d). Sequences (e) and (f) show examples of a moving cube being grasped from inside a moving ring. The character in sequence (g) steals the hat of another character while both are walking. Sequence (h) shows several object manipulations planned around obstacles and in synchronization with a long walking motion. Details of grabbing and dropping the cheese are shown in sequences (i) and (j).

8.8 Planning Collision-Free Reaching Motions for Interactive Object Manipulation and Grasping

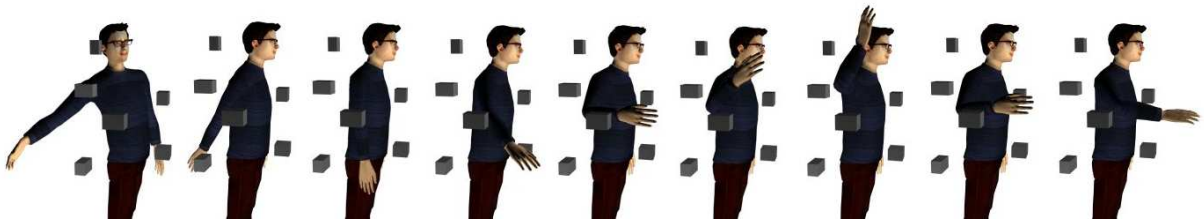
EUROGRAPHICS 2003 / P. Brunet and D. Fellner
(Guest Editors)

Volume 22 (*2003*), *Number 3*

Planning Collision-Free Reaching Motions for Interactive Object Manipulation and Grasping

Marcelo Kallmann,[1†] Amaury Aubel,[2†] Tolga Abaci[3] and Daniel Thalmann[3]

[1] Robotics Research Lab, University of Southern California, Los Angeles, United States, kallmann@usc.edu
[2] DreamWorks Animation, Glendale, United States, aaubel@anim.dreamworks.com
[3] Virtual Reality Lab, Swiss Federal Institute of Technology, Lausanne, Switzerland, {tolga.abaci|daniel.thalmann}@epfl.ch



Abstract

We present new techniques that use motion planning algorithms based on probabilistic roadmaps to control 22 degrees of freedom (DOFs) of human-like characters in interactive applications. Our main purpose is the automatic synthesis of collision-free reaching motions for both arms, with automatic column control and leg flexion. Generated motions are collision-free, in equilibrium, and respect articulation range limits. In order to deal with the high (22) dimension of our configuration space, we bias the random distribution of configurations to favor postures most useful for reaching and grasping. In addition, extensions are presented in order to interactively generate object manipulation sequences: a probabilistic inverse kinematics solver for proposing goal postures matching pre-designed grasps; dynamic update of roadmaps when obstacles change position; online planning of object location transfer; and an automatic stepping control to enlarge the character's reachable space. This is, to our knowledge, the first time probabilistic planning techniques are used to automatically generate collision-free reaching motions involving the entire body of a human-like character at interactive frame rates.

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

1. Introduction

Recent research in the character animation domain has mainly concentrated on the generation of realistic movements using motion capture data. Probably on account of its difficult nature, the problem of automatically synthesizing collision-free motions for object manipulation has received little attention from the Computer Graphics community. Most of the techniques developed[1] have not sufficiently explored this domain.

The automatic generation of collision-free grasping sequences has several direct applications in virtual reality, games, and computer animation. And yet, producing collision-free grasping motions currently involves lots of tedious manual work from designers.

Motion planning originated in Robotics, with an emphasis on the synthesis of collision-free motions for any sort of robotic structure[2]. Some works have applied mo-

† Work done while at EPFL - Virtual Reality Lab

tion planning to animate human-like characters manipulating objects[3, 4]. However, the nature of the articulated structures being controlled is usually not taken into consideration. Most often, only one arm is used for reaching while the rest of the body remains static.

We present in this paper a collection of new techniques based on probabilistic motion planning for controlling human-like articulated characters. Our goal is to synthesize valid, collision-free grasping motions while taking into account several human-related issues, such as: control of the entire body (including leg flexion, spine and clavicle-shoulder complex), joint coupling (e.g. spine), articulation limits, and comfort criteria.

We mainly concentrate on the reaching phase problem, i.e., how to compute a valid collision-free motion between two postures. Our method operates on 22 degrees of freedom (DOFs) of an abstract control layer, mapped to the actual DOFs of the character (over 70).

In order to deal with the reduced yet high dimensional configuration space defined by the abstract control layer, we make use of a pre-computed Probabilistic Reaching Roadmap encoding comfort criteria. This roadmap is constructed with a carefully tailored sampling routine that efficiently explores the free regions in the configuration space and favors postures most useful for grasping.

In addition, we present several extensions applied to the problem of object manipulation: a probabilistic inverse kinematics method used to automatically propose goal postures for designed grasps, a technique to dynamically update the roadmap when obstacles change position, a method for planning transfer motions when objects are attached to the character's hands, and an automatic stepping control mechanism to enlarge the character's reachable space.

We have fully implemented the methods proposed herein as integrated interactive tools for the production of object manipulation sequences. After a preprocessing of a few minutes (to compute the required roadmaps and manipulation postures), manipulation motions are quickly synthesized. Several animation sequences of a character reaching for and manipulating objects are presented to demonstrate the effectiveness of our method.

2. Related Work

It is common sense that the use of motion captured data is the best approach to achieve realistic human-like motions for characters. Several advances have been proposed on this subject[5, 6, 7, 8, 9]. However, for motions such as object manipulation, the main concern is on the precise control and correctness of motions, and thus captured data are hard to reuse.

The key problem for object manipulation is to solve the reaching phase for a given target 6-DOF hand location. The most popular approach is to somehow solve the underlying inverse kinematics (IK) problem[9, 10, 11, 12]. However, three main difficulties appear when devising IK algorithms. First, as the problem is under determined, additional criteria are needed in the system formulation in order to select valid and natural postures among all possible ones. Second, IK algorithms alone do not ensure that generated postures are free of collisions. Last, IK algorithms are more suitable for synthesizing postures than animations.

Computing collision-free reaching motions is in fact a motion planning problem[2]. Among the several existing methods, those based on probabilistic roadmaps[13, 14, 15, 16] are particularly amenable to high-dimensional configuration spaces. Roadmaps can typically be computed in a preprocessing step and re-used for fast on-line querying.

Different strategies have been proposed to construct roadmaps. Visibility-based Roadmaps[16] use a visibility criterion to generate roadmaps with a small number of nodes. Rapidly-Exploring Random Trees (RRTs)[14, 17] generate a tree that efficiently explores the configuration space. Because of this important property, we make use of RRTs as the growing strategy to construct our roadmap.

Other kinds of motion planners have been applied to the animation of human-like characters[3, 4, 18]. However, these works are limited to the control of only one arm at a time. In another direction, a posture interpolation automaton[19] was proposed, however with collision avoidance treated as a post process based on a force-field approach, which is highly sensitive to local minima.

In order to compute complete grasping and object manipulation sequences, several extensions are required. We follow the idea of predefining grasps (hand locations and shapes) for each object to be manipulated[20, 21, 22], and developed a probabilistic IK algorithm to automatically propose goal postures matching the predefined grasps. The IK algorithm is inspired by some approaches based on Genetic Algorithms[23, 24].

Other works have also addressed the dynamic update of roadmaps[25], allowing to cope with object displacement in the workspace. We present here a similar technique except that we maintain a roadmap that is always a single connected component.

3. Method Overview and Paper Organization

A character posture is defined using a 22-DOF abstract control layer divided as follows: 9 DOFs to control each arm, 3 DOFs to control spine and torso movements, and 1 DOF to control leg flexion. Hereafter, when we speak of configurations, postures and DOFs, we are referring to configurations, postures and DOFs of the abstract control layer.

Let C be the 22-dimensional configuration space of our

control layer. Let C_{free} denote the open subset of valid configurations in C. A configuration is said to be valid if the corresponding posture: is collision free, respects articulation range limits, and is balanced.

A sampling routine is responsible for generating random valid configurations in C_{free}. As the dimension of C is high, several heuristics are implemented in order to favor the generation of postures most useful for grasping. The configuration sampling routine and the abstract control layer specification are presented in Section 4.

The sampling routine is used to construct our roadmap. We first run the standard RRT algorithm[14, 17] to build a roadmap from the initial rest posture, and then apply a process which adds extra valid edges (or links) to existing nodes. Valid extra edges are added only if they represent a shorter path in the roadmap, i.e. if they represent a shortcut. The final step is to perform a proper weighting according to comfort criteria. As a result we obtain a Probabilistic Reaching Roadmap, hereafter simply referred to as roadmap. The roadmap construction process is detailed in Section 5.

Let R be a roadmap which was computed during an off line phase. Let q_c be the current character configuration and let q_g be a given valid goal configuration. A path P in C_{free} joining q_c and q_g is determined by finding the shortest path in R joining the nearest nodes of q_c and q_g in R. Path P is said to be valid if all configurations interpolated along P are valid, and in this case a final smoothing process is applied in order to obtain the final reaching motion. This entire process is described in Section 6.

As configurations have 22 DOFs, we allow designers to specify only target 6 DOFs hand locations to be reached, and a probabilistic IK algorithm automatically proposes valid goal configurations. This probabilistic IK algorithm and several other extensions useful for creating animations involving grasping and displacement of objects are presented in Section 7.

Section 8 presents and discusses obtained results, and finally Section 9 presents conclusions and future work.

4. Configuration Sampling

Configuration Definition. A complete configuration of our abstract control layer is defined by a set of 22 DOFs. Each arm is defined by 9 DOFs, five of which are devoted to the shoulder complex, the four remaining ones being equally distributed on the elbow and wrist. The character's spine, which comprises the lumbar and thoracic vertebrae, is completely determined by three DOFs, each of which controlling a unique rotational direction. Finally, 1 translational DOF controls the flexion of the legs.

We represent the arm's kinematic chain by four rotational joints: clavicle, shoulder, elbow and wrist. Except for the elbow, which is parameterized by two Euler angles (flexion and twist), we use the natural swing-and-twist decomposition defined by Grassia[26]:

$$R = R^{twist} R^{swing}, \text{ where}$$

$$R^{twist} = R_z(\theta), \text{ and } R^{swing} = \begin{bmatrix} S_x & S_y & 0 \end{bmatrix}$$

The swing motion is performed by a rotation parameterized by the above axis-angle. Note that the rotation axis for the swing always lies in the x-y plane perpendicular to the skeleton segment. The axial rotation (or twist) that follows occurs around the (arbitrarily chosen) z-axis of the local frame. In practice, the axial rotation is not used for clavicle and wrist joints and we simply set $\theta = 0$.

The one singularity of the parameterization is reached when the swing vector has norm π. Consequently, the singularity is easily avoided for the motion range of human joints by choosing an appropriate zero posture (i.e., when $S_x = S_y = 0$). For the shoulder joint for instance, we choose a reference posture in which the arm is outstretched.

We place limits on each arm joint individually. Elbow flexion, elbow twisting and shoulder twisting are limited by confining the corresponding angles to a given range. The direction of the upper arm is restricted to the interior of a spherical polygon[27]. The same kind of directional limit is applied to the clavicle and wrist joints.

The many joints in the spine are controlled by a reduced set of three DOFs, which determine respectively the total spine flexion (bending forward and backward), roll (bending sideways) and twist. While roll and twist are distributed uniformly over the spine joints, we apply the flexion mainly on the lumbar vertebrae. This distribution ensures that when the character bends forward, its back remains straight and not unnaturally hunched. This approach gives good results (see Figure 2) and is simpler than other spine coupling strategies[28].

Leg flexion is determined through the use of an IK solver. We first constrain the position and orientation of the feet to remain fixed with respect to the ground. Then we analytically compute the required rotations at the hips, knees and ankles to lower the waist according to the value of the DOF, which represents the vertical translation of the pelvis. Note that the same DOF could be used to make the character stand on tiptoes so as to reach higher.

Sampling. The sampling routine is responsible for generating random valid configurations. The random generation takes place in the 22-DOFs parameter space. For the most part, parameters are randomly generated directly within the allowed range (i.e., articulation limits). For DOFs that control a swing movement, however, directly generating parameters that respect directional limits is difficult for lack of an analytic formulation of the spherical polygon. In such cases, we simply keep iterating until the directional limits are respected (shoulder), or project the current direction onto the borders of the spherical polygon (wrist). Before accepting

the random posture, we finally check if it is balanced and free of collisions.

The balance test performs a projection of the character's center of mass onto the floor, and check if it lies inside the support polygon defined as the convex hull of the feet base.

Rigid objects representing body parts are attached to the skeleton and used for collision checking. We first deactivate collision checking for all pairs of body parts that intersect in the rest posture (assumed to be valid collisions). Deactivated pairs are mainly adjacent body parts in the skeleton. After this initialization process, a collision is said to take place if any activated pair of body parts intersects, or if any body part collides with the environment. Note that rigid body parts are used for collision checking but that a regular skinning technique is used for displaying a realistic character with deformable skin. We employ the V-Collide library[29] for collision checking.

As the dimension of our configuration space is high, we bias the random distribution of configurations in order to favor postures most useful for reaching and grasping. More specifically, we distinguish two posture types:

- Regular postures have little spine motion, little leg flexion, no clavicle motion and random arm poses.
- Distant-reaching postures have large spine motion and/or large leg flexion, little elbow flexion, shoulder-clavicle coupling, arm-legs coupling, and arm-torso coupling.

In our current implementation, regular and distant-reaching postures are generated with a respective likelihood of 60% and 40%. Also, 66% of distant-reaching postures use the right hand as if the character were right-handed (see Figure 1).

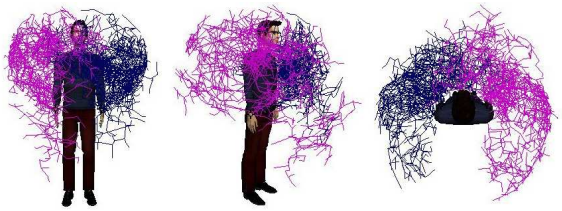


Figure 1: *Example roadmap (with little knee flexion). Each configuration in the roadmap is graphically represented with two graph nodes, which are the positions of the right (purple color) and left (blue color) wrist. Asymmetry results from the preference given to the right arm.*

Regular postures are useful when spine motion is not needed to reach a location with the hand. Note that, however, we always generate a small amount of spine motion to avoid robotic-like motions due to a completely static vertebral column.

Distant-reaching postures are suitable for remote objects that cannot be reached with arm motion only. Example postures can be seen in Figure 2. An important concept for generating distant-reaching postures is the use of couplings. These serve to favor configurations that expand the reachable space of the character. Leg flexion for instance, is authorized if one arm points downwards as if to reach for a low object. Similarly, the spine is bent so that the up-most thoracic vertebra travels in roughly the same direction as that of the arm. Finally, the clavicle is moved in such a way that another few centimeters are gained toward the imaginary location the arm points at.

Figure 2: *Examples of distant-reaching postures.*

Instead of computing two separate roadmaps for the right and left arms, we generate a single roadmap encoding motions of both arms. Besides reducing memory consumption and storage costs for large roadmaps, we guarantee that while reaching with one arm, possible motions of the spine will not induce collisions with the other arm. A further benefit is that our roadmap can also handle multi-hand reaching motions.

5. Roadmap Computation

The roadmap construction relies mainly on three functions: the sampling routine, the distance function and the interpolation function. We now describe the latter two.

Distance function. Good results are usually obtained with distance functions based on the sum of the Euclidean distances between corresponding vertices lying in the shape of the articulated structure[2, 7]. We use a similar yet simplified approach based on a selected set of articulations: the bottom-most lumbar vertebra, the top-most thoracic vertebra, the articulations of both arms (shoulder, elbow and wrist), and finally the thumb and pinky base joints to capture arm twisting and wrist rotations. Let q_1 and q_2 be two configurations. Our distance function $dist(q_1, q_2)$ returns the average sum of the Euclidean distances between the corresponding selected articulations at configurations q_1 and q_2.

The primary advantage of our distance function is its

speed. Another important property is that it remains independent of both the skin deformation module and the vertices density distribution in the skin mesh.

Interpolation function. The interpolation function $interp(q_1, q_2, t)$ returns, for each $t \in [0,1]$ a configuration varying from q_1 $(t = 0)$ to q_2 $(t = 1)$. The interpolation function applies spherical linear interpolation between corresponding joints, except for the translational joint controlling leg flexion and the elbow, which is parameterized with Euler angles. For these joints, linear interpolation is applied.

The interpolation is said to be valid if, for all values of $t \in [0,1]$, the *interp* function returns a valid configuration. The implementation of the interpolation validity test is approximate: An interpolation is considered valid if n equally spaced interpolated configurations between $t = 0$ and $t = 1$ are valid. The number n trades computation precision for speed, and its value is also adjusted according to the distance between the two configurations.

Roadmap growing process. The roadmap construction starts with the tree growing process of the RRT algorithm[14, 17]. The initial posture is the character's rest posture, i.e. standing straight with arms lying by the side. This rest posture is well suited for generating a tree with nearly uniform branch depth.

In the RRT algorithm a random configuration q_{rand} is used as a growing direction. The nearest configuration q_{near} in the current tree is determined and a new configuration q_{new} is computed as:

$$q_{new} = interp(q_{near}, q_{rand}, t), \text{ where}$$
$$t = \varepsilon/d,\ d = dist(q_{near}, q_{rand})$$

If q_{new} is valid and the interpolated path to q_{near} is also valid, q_{new} is linked to q_{near}, making the tree grow by one node and one edge. The new edge is assigned the cost ε. The factor ε represents the roadmap edge length, i.e. the incremental step by which the tree is grown. Large steps make the roadmap grow quickly but with more difficulty to capture the free configuration space around obstacles. Inversely, too small values generate roadmaps with too many nodes, thus slowing algorithms down. Good values for ε mainly depend on the complexity of the environment.

The tree generation process runs until a specified number of nodes is reached. In our experiments, we worked with graphs made up of around 1500 nodes.

Shortcuts. Once the tree is constructed we transform it into a graph with the process of shortcuts creation: for each pair of leaf nodes (l_1, l_2) in the tree, an edge linking l_1 to l_2 is added to the roadmap if:

- the interpolation between l_1 and l_2 is valid,
- $dist(l_1, l_2) <$ shortest path in the tree joining l_1 and l_2,
- $dist(l_1, l_2) < r$.

Shortest paths are easily determined by running an A^* algorithm[2] over the roadmap, taking into account the costs associated to the roadmap edges. The radius r is a parameter that specifies a limit on the length of shortcuts created and serves also to control the total number of shortcuts added to the roadmap.

Figure 3 illustrates the effect of adding shortcuts in the simpler 2-dimensional configuration space problem. In this example, the square is the sole obstacle and the root of the tree is on the left side of the square.

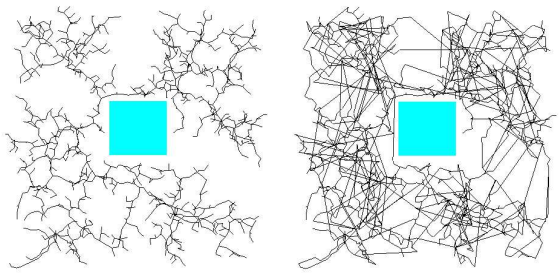


Figure 3: *The roadmap before (left image) and after (right image) the insertion of shortcuts.*

Single arm costs. The roadmap constructed so far encodes in each edge a cost defined as the distance between the two configurations linked by the edge. These configurations contain random positions for both arms of the character. As we use the same roadmap to determine single-arm motions as well, we also store in each edge of the roadmap two additional costs. The right arm motion cost is calculated with a distance function that simply does not take into consideration the joints in the left arm. Conversely, the left arm motion cost is determined by ignoring joints in the right arm.

Weights. In the final stage, we assign to each roadmap edge a proper weight to favor the determination of paths with better comfort characteristics. Different heuristics can be used to determine such weights. We use the central idea of favoring motions passing near the rest posture. We first compute the distance between the rest posture and the midpoint of each roadmap edge. Each edge is assigned this distance scaled to the interval $[k, 1], k \in [0, 1)$. Parameter k gives the amount of influence of the weighting and is controlled through the user interface. The weighting scheme helps, among others, to generate motions with the arm closer to the body.

6. Roadmap Querying

Let R be a roadmap computed during an off line phase, as described in the previous section. Let q_c be the current (valid) character configuration and let q_g be a given valid goal configuration to reach. Note that q_c and q_g are not necessarily contained in R (and in fact, normally they are not). The desired reaching motion is obtained by determining a valid

path in C_{free} having q_c and q_g as endpoints. The path is determined in two phases: path finding and path smoothing.

Path finding. We first find $N(q_c) \in R$ and $N(q_g) \in R$, which are respectively the nearest nodes to q_c and q_g in R. It is required that the interpolation between $N(q_c)$ and q_c and the interpolation between $N(q_g)$ and q_g are valid. If not, it means that the graph has not grown sufficiently, or that the goal configuration is not reachable.

Then, the correct cost in the roadmap is activated according to the desired grasping type (right, left or both hands), and the shortest path in R joining $N(q_c)$ and $N(q_g)$ is found. Note that R ensures that the shortest path is valid. Valid paths are represented as a sequence of configurations, where the interpolation of each pair of consecutive configurations is valid. The final reaching path P is obtained by adding to the shortest path configurations q_c and q_g as end nodes.

Path Smoothing. Because of the random nature of the nodes in R, P normally does not represent a useful motion and a smoothing process is required. We basically smooth P by incremental linearization.

Let q_1, q_2 and q_3 be a corner of P, i.e three consecutive configurations in P, and let $q = interp(q_1, q_3, t)$ where t is set according to the relative distances of the three configurations. If the interpolation between q_1 and q, and between q and q_3 are both valid, a local smooth can be applied and q_2 is replaced with q.

Our smoothing algorithm always selects the corner of P that deviates most from a "straight line". The distance between q_2 and q gives us a measure of the deviation. This process quickly results in a smooth path. After a while, it also tends to bring the path closer to obstacles.

We propose additional operations, which are applied during the basic smoothing process:

- Whenever two consecutive configurations in P get too close, they get merged into a single one; conversely, if they are too distant, a new configuration is inserted in-between by interpolation with $t = 0.5$.
- At every k steps during the iterative local smoothing process, we try to apply a group smoothing: two configurations in P are randomly selected and, if their interpolation is valid, all nodes between them are removed from P (note that a re-sampling may occur due to the operation described in the previous item). This procedure greatly accelerates the process in many cases, and even permits to escape from local minima. In our experiments we have used k as the number of nodes in P. Finally, we also obtained significant speed-ups by applying group smoothing hierarchically before entering the iterative loop: from both endpoints in P, we perform a recursive binary partition until pairs are smoothed or until consecutive pairs are reached.
- Last but not least, when the application of one of the smoothing procedures fails due to non-valid interpolation between configurations, we test again the same interpolation on different combinations of groups of DOFs (and not on all DOFs at the same time). Groups of DOFs are defined as: left arm, right arm, spine and legs. This process keeps smoothing for instance the motion of one arm when spine motion cannot be smoothed anymore because of obstacles.

7. Extensions for Object Manipulation and Grasping

Probabilistic IK. It is not an easy task for the artist to specify a realistic 22-DOFs goal configuration q_g (used as input for roadmap querying). To overcome this difficulty we allow designers to simply place a three-dimensional model of a hand anywhere in the workspace, and run a probabilistic IK algorithm to automatically propose goal configurations matching the specified 6-DOFs hand location. A probabilistic approach is effective because we already have nodes in the roadmap with the hand close to the required posture. In addition, our framework enables us to easily generate collision-free postures. In contrast, Jacobian-based methods exhibit better convergence but cannot guarantee collision-free postures.

Our method is inspired by some Genetic Algorithms implementations[23, 24]. Let H be a target hand location. We first select the k closest configurations q_i, $i \in 1,\ldots,k$ in the roadmap, according to a distance function that only considers the distance between H and H_i, where H_i is the same object H, but placed at the hand location specified by configuration q_i. The distance function takes the average sum of the Euclidean distances between each corresponding pair of vertices in H_i and H.

Configurations q_i, $i \in 1,\ldots,k$ constitute the initial population that converges towards H by minimizing the distance function. Usually the configurations in the initial population are already very close to H, and thus, instead of developing all usual operators of a Genetic Algorithm approach, we obtain satisfactory results with simple and faster strategies (see Figure 4).

We use a variation of the roadmap growing procedure. Random configurations q_{rand} are generated, and if the incremental interpolation from q_i towards q_{rand} gives a valid and closer configuration to H, q_i is replaced by the incremented version. However, if H is located close to the limits of the reachable workspace, the convergence of this method may become problematic. In such cases, we adopt a different strategy and apply perturbations on random DOFs of every q_i. A perturbed configuration is kept only if it is valid and closer to H. In both methods, backtracking is applied when a local minimum is reached.

Grasping. We follow the usual approach of having predesigned hand shapes for every object to be grasped[20, 21, 22]. A complete grasping sequence results from the concatenation of any number of reaching motions and a final grasping.

Figure 4: *The closest configuration in the roadmap in relation to the target hand (left column), and after the probabilistic IK process (right column).*

The final grasping is generated like a reaching motion except that it moreover includes the interpolation of finger joint angles towards target angles defined in the grasping hand shape. An additional finger-object collision test can be used to ensure perfect grasping as well as to diminish the required design precision of hand shapes.

Dynamic roadmap update. Each time an object in the workspace moves, the roadmap needs to be updated accordingly. The approach is to detect and remove the nodes and edges that become invalid after a change in the workspace. As a result, disconnected roadmap components may appear. Instead of managing disconnect parts[25], we follow the philosophy of keeping a single connected roadmap that represents the reachable configuration space C_{free} at all times. Whenever an obstacle in the workspace is inserted, removed, or displaced, a global roadmap validation routine is performed in three steps:

- All invalid edges and nodes are removed. Disconnected components may appear.
- We try to connect each pair of disconnected components by adding valid links joining the k closest pairs of nodes in each component (in our experiments, $k = 0.2n$ where n is the number of nodes in the smaller component of a pair). If disconnected components still linger, we simply keep the largest component.
- Due to the operations described above, the roadmap may no longer cover C_{free} very well. "Holes" in the coverage of C_{free} are likely to appear because some regions become free due to an obstacle displacement, and because of removed components in the roadmap. Hence, the roadmap is grown again (see Figure 5) as described in Section 5. Note that the sampling routine can easily be biased to generate postures only in the parts of C_{free} that are insufficiently covered.

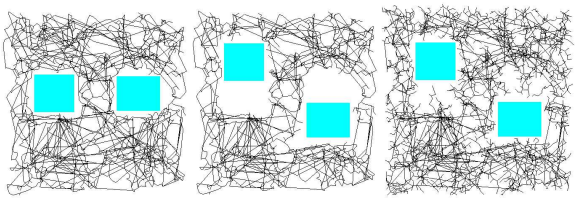


Figure 5: *Left: original roadmap. Middle: roadmap exhibiting "holes" due to obstacles displacements. Right: roadmap is grown again.*

Transfer paths. We call a transfer path a motion enabling the character to move an object from one place to another.

The main difficulty of computing transfer paths is that roadmap nodes can no longer be guaranteed to be valid because of objects attached to the character's hand(s). One first option is to pre-compute specific roadmaps for each object that needs to be carried by the character. This solution could be used, for instance, to plan motions for a character with a sword in its hand.

We developed an alternative method that returns transfer paths on the fly. We first compute a reaching path P between the first and the last posture of the desired transfer path, without considering the object being transferred. Then the object is attached to the character's hand and P is checked for validity, this time taking the attached object into account. If the object is small, P may still be valid and directly useful as a transfer path. If not, the invalid nodes and edges of P are removed, and disconnected nodes in P are reconnected at run time using a standard single query RRT[17].

The performance of this method greatly depends on the complexity of the transfer path to be computed, ranging from extremely fast in simple cases to extremely slow in complex transfer cases.

After a transfer path has been computed, the displaced object needs to be removed from the roadmap and inserted again at its new position. This must be done in both methods, i.e. when using additional pre-computed roadmaps or when planning transfer motions on the fly.

Stepping control. Although we control the entire body of the character when generating motions, we are still limited to grasping sequences with the feet fixed on the floor. In some cases, much more realistic results are achieved if the character takes some steps. Typically, steps are used for obtaining better balance and for reaching distant objects.

We developed a multi state approach to let the character step. As a pre-process, we create k short stepping animations with different lengths and directions. Each of these animations transfers the character's rest posture p_r into the final

posture p_i of the stepping animation i, $i \in 1,\ldots,k$. Each final posture is considered to be a valid state if the animation linking p_r and p_i can be played without collisions. Furthermore, adjacent states are also linked with pre-defined stepping animations if these do not incur collisions.

For each valid p_i, a roadmap R_i is computed considering p_i as the start node of the roadmap generation. In the end, several roadmaps co-exist, the original reaching roadmap R being centered at p_r while adjacent roadmaps R_i are centered at various positions p_i around R (see Figure 6).

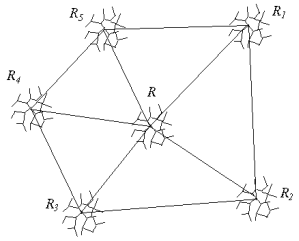


Figure 6: *Distinct roadmaps are generated and connected with predefined stepping animations.*

Let q_i and q_g be the initial and goal configurations of a reaching animation to be determined with stepping control. We first detect the closest configurations to q_i and q_g in any of the roadmaps, determining the initial and goal roadmaps to be used (R_i and R_g respectively). Then we determine path P_1 joining q_i and p_i in R_i, and path P_2 joining p_g and q_g in R_g. The final path P is obtained by concatenating P_1, the shortest sequence of animations joining p_i and p_g, and finally P_2.

A final smoothing process is applied over P, taking into account only the motions of the upper body limbs. Further explanations and examples are omitted for lack of space.

8. Analysis and Results

Results. Figure 7 presents animation stills of a virtual character reaching for objects in a fridge, and as well an example of object relocation. In these examples and others shown in the videos accompanying this article, we have grown roadmaps using an incremental distance ε of 4 cm until reaching 1500 nodes. In each example, the design work was limited to the definition of a few goal postures. Then, the motions were automatically generated by the planner without any user intervention. The only exception is the head orientation, which was specified by hand in some sequences.

All motions were produced with constant velocity along planned paths. The timing could easily be improved by the designer or automatically adjusted e.g. according to Fitts' law[30]. Note also that the left arm is not animated while the right hand is reaching, which creates a somewhat stiff look in some postures. Additional controllers need to be integrated in order to correct this, for instance simulating dynamics over that arm.

The method works extremely well providing that enough free space exists between obstacles. The roadmap encounters some difficulty to explore portions of the workspace containing many obstacles, especially when these are situated on the borders of the character's reachable space. This is exemplified by the refrigerator sequence. Even for small values of ε, postures where the hand reaches inside the refrigerator are not present in the first computed roadmap. Our probabilistic IK routine elegantly solves this problem. The designer simply specifies 6-DOF hand postures within the refrigerator and thus forces the roadmap to grow inside the refrigerator.

The encoded comfort criteria help to favor natural looking movements. However, the simple criterion currently used cannot capture all the subtleties of human movement. More complicated comfort criteria e.g. from biomechanics should be introduced. It is also important to note that the efficacy of the weighting scheme is restricted to cases where several paths exist.

Performance. With roadmaps consisting of around 1500 nodes, shortest paths are instantaneously determined (a few milliseconds) with an A^* algorithm. Construction times are listed in Table 1. In order to accelerate the computation of the distance function, joint positions relative to configurations in the roadmap are cached.

The smoothing phase gives good results in less than a second. Actually, in the presented results, we stopped the smoothing process when the time limit of 1 second was reached. It is important to mention that the group smoothing strategy tremendously accelerates the process.

The convergence of the probabilistic IK procedure greatly depends on the distances between the nodes in the initial population and the target hand posture. In our examples, we had cases ranging from a few seconds to a few minutes. Transfer paths calculated in relatively clear spaces, such as the example in Figure 7(b), could be computed in approximately 1 second.

The results and times reported in this paper were obtained with tools developed in a Maya plug-in, running on a Pentium PC at 1.7 GHz.

9. Conclusions

Contribution. This work makes a number of contributions that allow to plan grasping motions for a 22-DOF human-like character in interactive applications. More specifically, our main contributions are:

- A 22 DOFs abstract control layer that poses the entire body of the character: arms, shoulders, torso, spine and leg flexion.

Scene	Number Triangles	Roadmap Computation	Shortcuts Computation	Shortcuts Created
No Obstacles	0	70	6	565
Only Body Parts	10153	78	7	559
Cubes	10249	78	11	508
Kitchen	26088	145	18	486

Table 1: *Performance measurements. The second column gives the number of triangles considered for collision detection. The third and fourth columns give computation times in seconds. The last column lists the number of shortcuts created using the maximum shortcut length of 16 cm.*

- A biased sampling method that efficiently covers most-used parts of the free configuration space with random human-like grasping postures.
- A new roadmap structure: the probabilistic reaching roadmap, which is dense in connections between nodes and encodes comfort criteria.
- Several extensions for generating complete object manipulation sequences.

Future Work. We believe that the techniques presented herein open several new research directions, and show that motion planning can greatly benefit computer animation.

The notion of balance can be extended to take into account supports when hands or other body parts get in contact with objects. Motion constraints can be added to allow the displacement of objects with fixed orientation, e.g. like a glass of water. Additional DOFs can be included to control movements that are coordinated with objects, e.g. to control chair translation (when sitting), or to plan motions like opening a drawer or pressing a button.

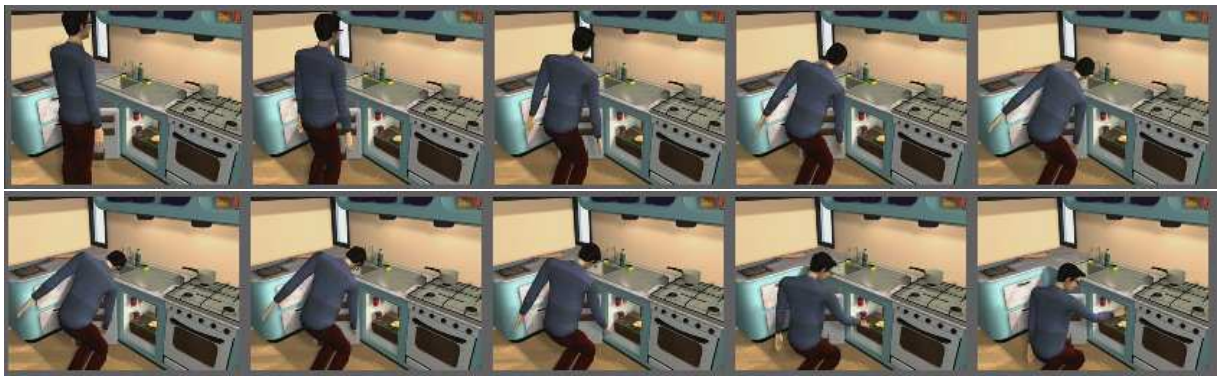
Finally, motion capture data could improve the personality of movements and could be included at two levels: in the posture sampling routine during the generation of the roadmap[7], and during the smoothing process by adding motion texturing[8].

Acknowledgements

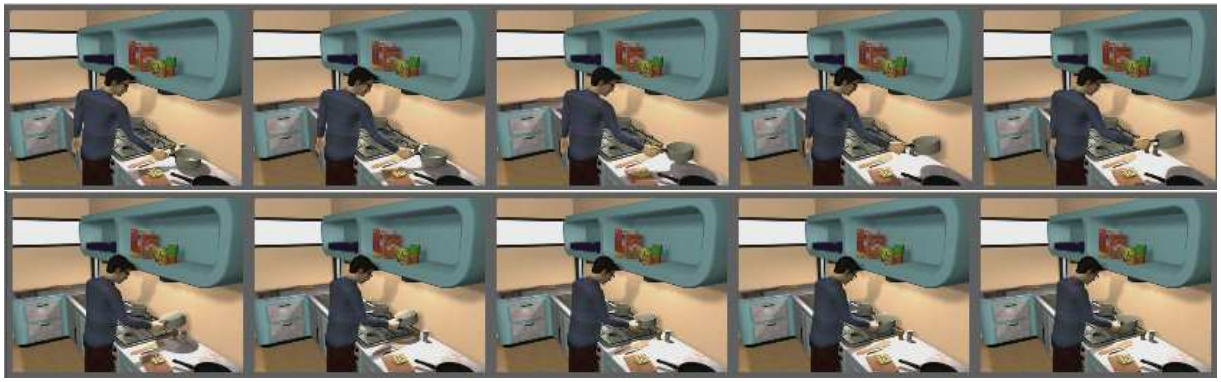
We express our thanks to Alias/Wavefront for the granted Maya SDK educational licence. This research has been partially funded by the Federal Office for Education and Science in the framework of the European project STAR (Service and Training through Augmented Reality), IST-2000-28764.

References

1. N. I. Badler, C. B. Phillips, and B. L. Webber. Simulating Humans: Computer Graphics, Animation and Control. ISBN 0-19-507359-2, 1993.
2. J.-C. Latombe. Robot Motion Planning. ISBN 0-7923-9206-X, Kluwer Academic Publishers, 1991.
3. Y. Koga, K. Kondo, J. Kuffner, and J. Latombe. Planning Motions with Intentions. *Proc. of SIGGRAPH'94*, 395-408, 1994.
4. J.J. Kuffner and J.C. Latombe. Interactive manipulation planning for animated characters. *Proc. of Pacific Graphics'00*, poster paper, Hong Kong, October 2000.
5. A. Witkin and Z. Popovic. Motion Warping. *Proc. of SIGGRAPH'95*, 1995.
6. M. Gleicher. Retargeting Motion to New Characters. *Proc. of SIGGRAPH'98*, 1998.
7. L. Kovar, M. Gleicher, and F. Pighin. Motion Graphs. *Proc. of SIGGRAPH'02*, 2002.
8. Y. Li, T. Wang, and H.-Y. Shum. Motion Texture: A Two-Level Statistical Model for Character Motion Synthesis. *Proc. of SIGGRAPH'02*, 2002.
9. C. F. Rose, P.-P. J. Sloan, and M. F. Cohen. Artist-Directed Inverse-Kinematics Using Radial Basis Function Interpolation. *Proc. of Eurographics*, **20**(3), 2001.
10. D. Tolani, and N. Badler. Real-Time Inverse Kinematics of the Human Arm. *Presence*, **5**(4):393-401, 1996.
11. P. Baerlocher, and R. Boulic. Task-priority Formulations for the Kinematic Control of Highly Redundant Articulated Structures. *Proc. of IROS'98*, Victoria, Canada, Oct. 1998.
12. X. Wang, and J.-P. Verriest. A Geometric Algorithm to Predict the Arm Reach Posture for Computer-aided Ergonomic Evaluation. *Journal of Vis. and Comp. Animation*, **9**(1):33-47, 1998.
13. L. Kavraki, P. Svestka, J. Latombe, and M. Overmars. Probabilistic Roadmaps for Fast Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, **12**:566-580, 1996.
14. S. La Valle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. *Technical Report* 98-11, Computer Science Dept., Iowa State University, Oct. 1998.
15. R. Bohlin and L. Kavraki. Path Planning using Lazy PRM. In Proc. of IEEE Int. *Conference on Robotics and Automation*, ICRA, 2000.
16. T. Simeon, J. P. Laumond, and C. Nissoux. Visibility Based Probabilistic Roadmaps for Motion Planning. *Advanced Robotics Journal*, **14**(2), 2000.
17. J.J. Kuffner and S.M. La Valle. RRT-Connect: An efficient approach to single-query path planning. *In Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA'2000)*, San Francisco, CA, April 2000.
18. S. Bandi and D. Thalmann. Path Finding for Human Motion In Virtual Environments. *Computational Geometry* **15**(1-3):103-127, 2000.



(a)



(b)

Figure 7: *Animation sequences within a kitchen scenario. In the first example (a), leg flexion is used for reaching two specified locations in the fridge: at the door, and inside it. The second example (b) shows the object relocation of a saucepan.*

19. R. Bindiganavale, J. Granieri, S. Wei, X. Zhao, and N. Badler. Posture interpolation with collision avoidance. *Computer Animation '94*, Geneva, Switzerland, 1994.
20. Y. Aydin, and M. Nakajima. Database guided computer animation of human grasping using forward and inverse kinematics. *Computer & Graphics*, **23**:145-154, 1999.
21. H. Rijpkema and M. Girard. Computer Animation of Knowledge-Based Human Grasping. *Proc. of SIGGRAPH'91*, 339-348, 1991.
22. M. Kallmann. Object Interaction in Real-Time Virtual Environments. DSc Thesis 2347, EPFL, January 2001.
23. A. A. Khwaja, M. O. Rahman, and M.G. Wagner. Inverse Kinematics of Arbitrary Robotic Manipulators using Genetic Algorithms. J. Lenarcic and M. L. Justy, editors, *Advances in Robot Kinematics: Analysis and Control*, 375-382. Kluwer Academic Publishers, 1998.
24. M.-H. Lavoie, and R. Boudreau. Obstacle Avoidance for Redundant Manipulators Using a Genetic Algorithm. *CCToMM Symp. on Mechan., Machines, and Mechatronics*, Canada, 2001.
25. T.-Y. Li, and Y.-C. Shie. An Incremental Learning Approach to Motion Planning with Roadmap Management. *Proc. of International Conference on Robotics and Automation (ICRA)*, 2002.
26. S. Grassia. Practical Parametrization of Rotations Using the Exponential Map. *Journal of Graphics Tools*, **3**(3):29-48, 1998.
27. J. U. Korein. A Geometric Investigation of Reach. The MIT Press, Cambridge, 1985.
28. G. Monheit and N. Badler. A Kinematic Model of the Human Spine and Torso. *IEEE Computer Graphics and Applications*, **11**(2):29-38, 1991.
29. S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. *Proc. of ACM SIGGRAPH*, 171-180, 1996.
30. P. Fitts. The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement. *Journal of Experimental Psychology*, **47**:381-391, 1954.