5-8-2003

Real-time reach planning for animated characters using hardware acceleration

Ying Liu
University of Pennsylvania

Norman I. Badler
University of Pennsylvania, badler@seas.upenn.edu

Real-time reach planning for animated characters using hardware acceleration

Abstract

We present a heuristic-based real-time reach planning algorithm for virtual human figures. Given the start and goal positions in a 3D workspace, our problem is to compute a collision-free path that specifies all the configurations for a human arm to move from the start to the goal. Our algorithm consists of three modules: spatial search, inverse kinematics, and collision detection. For the search module, instead of searching in joint configuration space like most existing motion planning methods do, we run a direct search in the workspace, guided by a heuristic distance-to-goal evaluation function. The inverse kinematics module attempts to select natural posture configurations for the arm along the path found in the workspace. During the search, candidate configurations will be checked for collisions taking advantage of the graphics hardware – depth buffer. The algorithm is fast and easy to implement. It allows real-time planning not only in static, structured environments, but also in dynamic, unstructured environments. No preprocessing and prior knowledge about the environment is required. Several examples are shown illustrating the competence of the planner at generating motion plans for a typical human arm model with seven degrees of freedom.

Real-time Reach Planning for Animated Characters
Using Hardware Acceleration

Ying Liu   Norman I. Badler
Center for Human Modeling and Simulation
Department of Computer and Information Science
University of Pennsylvania
liuying, badler@graphics.cis.upenn.edu

Abstract

We present a heuristic-based real-time reach planning algorithm for virtual human figures. Given the start and goal positions in a 3D workspace, our problem is to compute a collision-free path that specifies all the configurations for a human arm to move from the start to the goal. Our algorithm consists of three modules: spatial search, inverse kinematics, and collision detection. For the search module, instead of searching in joint configuration space like most existing motion planning methods do, we run a direct search in the workspace, guided by a heuristic distance-to-goal evaluation function. The inverse kinematics module attempts to select natural posture configurations for the arm along the path found in the workspace. During the search, candidate configurations will be checked for collisions taking advantage of the graphics hardware – depth buffer. The algorithm is fast and easy to implement. It allows real-time planning not only in static, structured environments, but also in dynamic, unstructured environments. No preprocessing and prior knowledge about the environment is required. Several examples are shown illustrating the competence of the planner at generating motion plans for a typical human arm model with seven degrees of freedom.

1. Introduction

The issue of robot motion planning has been studied for a couple of decades and many important contributions to the problem have been made [7]. Most of these algorithms are based on the use of the configuration space (C-space): a structure that maps realizable joint angle ranges as dimensions in a Cartesian space. The inherent difficulty with the C-space approach is its high dimensionality. It is well known that the worst-case time bound for any complete motion planning algorithm is exponential in the dimensionality of its C-space [11]. Though reasonable performance can be achieved for low degree of freedom problems (low-dimensional C-space), motion planning algorithms typically run slowly when faced with many degrees of freedom [9].

Motion planning has applications in a variety of fields such as assembly planning, virtual prototyping [3], drug design [4], and computer graphics simulations [8, 10]. However, despite the applicability of motion planning techniques to computer graphics simulations, the problem has not been addressed much in the computer graphics community [2]. As stressed by Latombe [12], non-robotics applications (e.g. graphics animation, surgical planning, and computational biology) are growing in importance and are likely to shape future motion planning research at least as much as robotics.

In this paper, we address the motion planning problem in computer graphics while avoiding the explicit use of configuration space. Our focus is on inventing techniques aimed at making motion planning practical for real-time interactive animation of human arm reaching tasks in *constrained* workspaces. While there has been much robotics research on motion planning, our restriction to a human arm and torso provides an additional constraint as well as an opportunity to optimize such a difficult problem.

We describe a heuristic-based reaching algorithm to automatically generate motion paths for a human arm model in complex 3D environments. A system with such functions may be used in different domains. It allows the animator to direct the motion of the virtual human at a high level in computer animation. It is suitable for a variety of real-time applications involving autonomous animated characters. While inverse kinematics techniques are now common in computer animation, existing algorithms do not readily adapt to constrained or changing environments. Applications can also be found in ergonomic design and evaluation of workplaces for human operators, and maintenance facilities for service personnel [23].

The paper is organized as follows: in Section 2, a brief overview of the historical approaches to motion planning used by the computer graphics community is presented. Section 3 describes our algorithm in detail. Section 4 presents the experimental results of the implementation for several specific cases. Section 5 summarizes the state of our system. Section 6 outlines directions for future research.

2. Related Work

An early effort at addressing the motion planning problem in computer graphics is by J. Lengyel *et al*. [16]. They use standard graphics hardware to rasterize configuration space obstacles into a series of bitmap slices, and then use dynamic programming to create a navigation function and to calculate paths in this rasterized configuration space.

Working with a human figure model, [1] and [22] modify the inverse kinematics algorithm of [21] to include environment and body constraints. The constraints are inserted as a set of distance predicates checked during the solution search. The iterative gradient descent search simply respects the additional inequality constraints in each cycle. The speed of the algorithm is limited by the convergence rate of the inverse kinematics and the number of point-to-point distances checked each iteration.

In [8], a path planner is developed for several cooperating arms to manipulate a movable object between two configurations in the context of computer animation. In their work, a manipulation path is an alternating sequence of transit and transfer paths, where a transit path defines arm motions that do not move the object, while a transfer path defines arm motions that move the object. Transit paths are computed using a randomized path planner (RPP) in the arms' configuration spaces. For transfer paths, a collision-free trajectory in the movable object's configuration space is computed first, and then arm postures that are feasible to grasp the object are selected from a finite grasp set enumerated from an inverse kinematics algorithm.

Later, J. Kuffner designed a randomized path planner with a sampling heuristic designed for computing collision-free manipulation motions for animated human characters [10]. The sampling heuristic (known as RRT-Connect) is based on Rapidly-exploring Random Trees (RRTs) [13, 14].

Most recently, Foskey [5] propose a hybrid motion planning algorithm for rigid bodies translating and rotating in a 3D workspace. In their method, they generate a Voronoi roadmap in the workspace and combine it with "bridges" computed by randomized path planning with Voronoi-based sampling. The Voronoi roadmap is computed from a discrete approximation to the generalized Voronoi diagram of the workspace, which is generated using graphics hardware. Applications of the Voronoi diagram to motion planning for rigid bodies in 3D environments can also be found in [6, 18, 24].

3. Planning Algorithm

3.1. Problem Statement and Objectives

The problem of motion planning that we focus on in this work is described as follows: in a complex 3D environment (known as workspace in robotics literature), given a start position and a goal position of the end effector, compute an optimal collision-free motion path that a human arm can follow to reach from the start to the goal without interactive intervention from the animator.

The freedom of movement for the arm is constrained by the presence of obstacles in the environment. The arm should find a feasible obstacle-free path to the goal while satisfying certain criteria such as shortest path. For a human arm, its movement is also constrained by kinematics constraints such as joint limits.

The environment can be arbitrarily complex regarding possible solution paths and obstacles encountered. The obstacles are allowed to move unpredictably during planning.

The goal position can be changed interactively during planning. Once the goal is changed, the planner should be able to immediately adjust the plan toward the new goal.

As mentioned earlier, the worst-case time bound for any complete motion planning algorithm is exponential in the dimensionality of the configuration space [11]. This inherent difficulty makes many complete planning methods infeasible in practice. The planning framework presented here attempts to solve the problem in real-time by trading *completeness* for *efficiency*.

Instead of a complete algorithm, we are looking for fast spatial search and collision detection methods to perform reaches: if a reach is "easy", a solution should be found quickly; but if a reach is awkward or difficult, we can afford to indulge in more computation to find a path if one exists.

3.2. Algorithm Overview

To implement automatic reach space access planning in real time for a typical human arm model, we propose a heuristic-based reaching algorithm.

This algorithm consists of three modules: spatial search, inverse kinematics and collision detection. The spatial search module incrementally computes an optimal path from the start to the goal in discretized 3D workspace

for the end effector of the arm. Every time when a new node is generated in the path, it will be passed to the next module – inverse kinematics, which then computes a posture configuration to put the end effector in the new position. Candidate configurations will be checked for collisions taking advantage of the graphics hardware – depth buffer.

The following sections explain each of these modules in greater detail.

3.2.1. Spatial Search Module. One of the most perplexing problems in the study of biological motor control has been to determine in what type of coordinates movement plans are generated in the brain. Teleological concerns argue strongly for planning in workspace coordinates, since the characteristics of a movement are most naturally specified in terms of the layout of objects in the workspace [17].

Based on this observation, in our method, instead of searching in joint configuration space as most existing motion planning methods do, we propose to run a direct best-first search in the workspace, guided by a distance-to-goal evaluation function. It incrementally computes an optimal path from the start to the goal for the end effector of the arm.

In this module, the search is accomplished in the discretized workspace. Note that the discretization is not performed over the entire workspace, instead, only the discretized coordinates of the grid cells that the arm could possibly reach are computed, thus reducing the total computation time.

At each expansion step of the best-first search tree, six neighbors are considered for the current node. Suppose the discretized coordinate of the current node is (X, Y, Z), then its neighbors are defined as the face-adjacent nodes with the following coordinates in the discretized space:

$$(X - 1, Y, Z) \qquad (X + 1, Y, Z)$$
$$(X, Y - 1, Z) \qquad (X, Y + 1, Z)$$
$$(X, Y, Z - 1) \qquad (X, Y, Z + 1)$$

These six neighbors are sorted according to their distance to the goal and stored in a priority queue. The best-first search is continued with the nearest neighbor to the goal. Therefore, nodes with promising optimal costs are searched on a priority basis. The distance-to-goal evaluation function we use is simply the Euclidean distance from the current node to the goal.

To avoid circuits in the search graph, we mark all the visited nodes and only "unvisited" nodes are further expanded.

It is interesting to note that paradoxically but desirably, the search phase of the planner may run faster for complex, obstacle-cluttered environments, since such environments result in fewer free cells to search.

Figure 1 is an illustration of the searching process, where the end effector is at the wrist, the start position is shown in the blue sphere on the left and the goal is the bright green sphere on the right. Movement of the arm is constrained by the cube obstacle in the middle. The dotted green line is the computed path from the algorithm.

3.2.2. Inverse Kinematics Module. Choosing to work directly in the workspace makes inverse kinematics algorithm necessary because plans made in the workspace must eventually be translated into arm joint angles to drive the arm to move.

In order to achieve real-time performance, we prefer a method that can quickly solve inverse kinematics problems for a human arm. Currently, we are using the IKAN toolkit [19, 20]. It uses an analytical method to solve generalized inverse kinematics problems involving an anthropomorphic arm or leg and is faster and more reliable than numerical methods. It also allows the user to interactively explore all possible solutions using an intuitive set of parameters, which is also a favorable feature for our system.

3.2.3 Collision Detection Module. Instead of the traditional geometric collision detection methods, we take advantage of the computer graphics hardware depth buffer.

This module utilizes the hardware rendering pipeline commonly found on graphics accelerator cards to perform collision detection. Speed is gained by using the graphics hardware to quickly project the obstacle geometry into a 2D bitmap (depth map). Collision detection is done by depth comparisons between the depth values of the arm and the environment.

In 3D graphics, when rendering 3D objects onto a 2D screen, many different points in the objects may be mapped to the same pixel position on the screen, but with different depth values. Only the one with the smallest depth value is visible to the viewer and all the other ones are occluded. The depth buffer will be automatically updated with the nearest point's depth value every time when the scene is rendered. It is this property of the depth buffer that we use for collision detection.

This is a link-wise procedure. To check if the arm collides with any obstacles in the environment, collision detection will be performed link by link on upper arm, lower arm and hand sequentially.

For each link, a virtual perspective camera is placed at the proximal end of the link and the viewing direction is toward its distal end. This virtual camera is intended to observe the interior of the arm link geometry to detect any potentially penetrating obstacles. From the view of the camera, if it can see some objects other than the arm link

itself, then it indicates some obstacles penetrate the arm thus a collision will be reported. Accordingly, in depth buffer, the depth values of the part where the collisions occur will be the depth values of the corresponding intruding obstacles.

In other words, if we represent the depth map as a two-dimensional array, suppose the depth map of the arm link is $A[m][n]$ and the environment depth map is $E[m][n]$, then $A[i][j] >= E[i][j]$ means a collision exist between the obstacles and the arm. By comparing all the points in the bitmaps, we know whether there are any collisions between the environment and the current arm configuration. This procedure can be expressed in the following pseudo-code:

```
for i = 1 to m
    for j = 1 to n
        if A[i][j] >= E[i][j]
                return Collisions;
return No-Collisions.
```

For the algorithm to work, we assume that for each arm link, all interior surfaces are completely visible from the virtual camera location.

Figure 2 is an illustration of the collision detection procedure for the upper arm. For explanation purpose, we represent the link as a cylinder. The virtual camera is placed at the shoulder, and the viewing direction is toward the elbow. The blue cone represents the viewing volume. The green torus represents an obstacle. In this projection scheme, points P1 on the link and P2 on the obstacle are projected to the same position P on the screen. The depth value of P2 will be stored in the depth buffer, and is smaller than P1's. This only happens when the obstacle penetrates the link as shown in the figure.

Note that this is an approximate algorithm. The approximation is mainly from the finite resolution of the depth buffer. With higher resolution, it can detect collisions more accurately, but requires more computations accordingly.

Another limitation of this method is that it cannot detect collisions for the part of the link that is outside the viewing volume, as the zone Z1 and Z2 shown in figure 2. This problem can be solved by increasing the view angle so that the link is covered by the viewing volume as much as possible. But unfortunately, this will cause the resolution of the depth map to decrease. Thus there is a tradeoff between the two factors. Another alternative is to move the camera proximally along the segment axis and adjust the near clipping plane accordingly, as the dashed lines shown in Figure 2. In our implementation, the view angle is set to be 45 degrees, and the resolution of the depth buffer is 100*100.

3.2. Planning Procedure

The following is a more detailed planning procedure involving the three modules described above. Figure 3 shows the high-level description of the planning framework.

1. Initialization: A general 3D polygon mesh model of the environment and character suitable for rendering are provided, along with a starting position and a goal position.
2. Projection of the arm: For each link, set the camera at the proximal end and view direction toward its distal end. Render the link only and get the depth map from depth buffer for later comparisons.
3. Discretization: Compute the discretized coordinate for the current position of the end effector (wrist in the model) according to the user-specified resolution level.
4. Spatial search: Search next position for the end effector to reach. The possible candidates are the six neighbors of the current end effector cell less any already visited cells. The one with the shortest distance to the goal position is picked and marked as "visited". If all the candidates are "visited", then backtrack.
5. Inverse kinematics: Compute joint angles for the arm using IKAN to put the end effector in position.
6. Forward kinematics: Compute the positions of the shoulder, elbow and wrist in the workspace corresponding to the new configuration computed from step 4. The information will be used to position and orient the virtual camera in the following collision detection step.
7. Collision detection: Perform collision detection utilizing the depth-buffer for the upper arm, lower arm and hand sequentially:
   (a) Upper arm collision detection:
   - Setup: Position a virtual perspective camera at the shoulder. The view direction is toward the elbow. The near clipping plane of the camera is set to be slightly away from the shoulder, and the far clipping plane is slightly away from the elbow.
   - Projection: Render the obstacles in the environment and update the depth buffer
   - Read back: Read the depth map into memory from the depth buffer.
   - Comparisons: Compare the depth values between the upper arm and the environment as stated in Section 3.2.3. If no collisions, go to the next step; otherwise, go back to step 3.
   (b) Lower arm collision detection:
   Execute the same procedure for the lower arm as stated in step 7.a). The virtual camera is placed at the elbow and the view direction is

toward the wrist. The near and far clipping planes are slightly away from elbow and wrist, respectively.

 (c) Hand collision detection:
  Repeat the same procedure 7.a) for hand. The camera is placed at the wrist toward the middle-finger tip. The clipping planes are set accordingly.

8. If no collisions are found for all 3 links, the current configuration is put into the solution path and the arm is moved to this configuration, then repeat from step 3 until the goal position is reached; Otherwise, go back to step 4 and continue the search in the workspace.

Note that once the arm model is given, the depth map for each link will not change throughout the planning as long as the segment is not deformed during the movement. Therefore they only need to be computed once in practice, thus saving some computation time.

4. Implementation and Experimental Results

We implemented the basic algorithm described in the previous sections and tested it on a few sample environments. It works well in the designed scenarios.

The algorithm is implemented on a 1500MHz Dell PC using C++ and OpenGL. It is applied to a seven degree-of-freedom human arm model with the shoulder position fixed.

A sample environment is shown in Figure 4, where the arm is at the start configuration. Two reaching tasks are specified in this environment. Task A is to reach for the thermos in the middle, task B is to reach for the teacup on the right from the thermos.

This is a relatively open environment, but is composed of more than 30,000 polygon primitives. The main purpose is to show the planner's potential to plan motions at interactive rates in complex environments.

In Figure 4, the movements of the arm are constrained by the desktop and the computer. It tries to avoid the collisions with these objects during planning.

	Step Size	Visited Nodes	Solution Nodes	Total Time	Time Per Step
Task A	0.06	57	39	1,129	19.81
Task B	0.06	46	32	1,015	22.06
Task A	0.1	37	27	797	21.54
Task B	0.1	32	22	687	21.47

Table 1. Experimental results

The experimental results are listed in Table 1. Total Time is the average total planning and execution time for the designed task. Times listed in the table are in milliseconds. Visited Nodes is the number of the total nodes visited while marching through the workspace grid during the planning. Solution Nodes is the number of "good" nodes in the final computed planning path. The Step Size is the marching cell size of the workspace grid. The arm radius is 0.18, so the two step sizes are relatively fine enough. The depth buffer resolution used is 100*100.

Figure 5 shows another sample environment. The goals are represented as orange spheres. The image sequence shows some key configurations of the character computed from the algorithm.

Animation examples can be found at http://www.seas.upenn.edu/~liuying/animations/.

During an interactive session, the user can click and drag on the goal location or obstacles, and the path planner will calculate an updated, collision-free path toward the new goal at interactive rate.

Among the three key modules of the algorithm, the collision detection dominates the computation time per step and plays a key role in achieving real time performance. It involves three major operations: geometry projection, read-back and depth comparisons. The projection rate mostly depends on the complexity of the environment, and the resolution of the depth buffer affects the computation time for read-back and comparisons. In the environment shown in Figure 4, when the depth buffer resolution is increased to 600*600, the planning and execution time per step is around 28ms.

Note that when rendering the environment for collision detection purpose, since we only care about the depth values of the objects, the rendering time can be further reduced by disabling some expensive graphics computations such as lighting. For programming simplicity, we keep the lights on throughout the planning.

The effectiveness of the planner and the total planning time is determined by the search and inverse kinematics modules, where the performance of the search depends on the discretization resolution of the workspace. The finer the resolution is, the longer it takes to find a solution. On the other hand, with a coarser discretization, it will run faster, but in some cases when narrow passages are involved, it may fail to find solutions.

5. Discussion

In this paper, we present a real-time planner that has been specially designed to quickly solve path planning queries involving human arms. It is straightforward and easy to implement. The examples shown illustrate the competence of the planner at generating motion plans for a typical human arm model with seven degrees of freedom.

This method differs from existing motion planning methods in several ways. In our method, the arm starts out without any knowledge of the environment. It explores the environment and generates goal-directed motion on the fly, based upon the information acquired thus far, until the goal is reached. A feasible collision-free pose in the goal position is not required in advance. During exploration of the environment, the algorithm just takes the immediate surroundings of the arm into account, thus saving planning time and boosting efficiency by only exploring the grid cells that the arm could possibly reach.

Furthermore, our method avoids expensive pre-processing steps associated with traditional geometric techniques that partition configuration space into obstacle space vs. free space. Instead, a much less expensive potential field computation is carried out that provides a heuristic distance-to-goal evaluation function over the workspace. In actuality, our method does not fully and explicitly pre-compute such a potential function over the entire workspace, but rather computes a crude measure of remaining distance to the goal at each step during planning that serves as a virtual potential field. The crucial point is that an easily computed potential field can be of good heuristic value, making it possible to search directly for paths without excessive backtracking [17].

While being inherently local, the algorithm does not suffer from local minima like most potential field based methods since the search is always continued with one of the six neighbors that is closest to the goal position. When no neighbors are accessible, the algorithm will backtrack.

In addition, this algorithm allows planning not only in static, structured environments, but it is also useful in dynamic, unstructured environments where no prior information is available. Each time obstacles move, the depth buffer will be dynamically updated automatically. This way, the real-time hardware computation enables local motion planning through dynamic environments.

This method also allows interactive changes of the goal position during the planning process. The planner will immediately adjust the motion plan for the arm toward the new goal.

The collision detection module runs fast due to the utilization of the graphics hardware depth buffer. The algorithm also makes parallel computations possible, which we expect to result in an even better performance for the planner.

In the direct heuristic search formulation, extra work is introduced in the search process only when backtracking becomes necessary. In our case, the cost is expected to be acceptable empirically by adding appropriate constraints, such as setting up a time limit that one can afford to plan reaches. If the time limit is exceeded, the planner simply returns with a failure, meaning either no solutions exist, or it could not find one.

6. Future Work

Although some promising results are shown in its present form, the algorithm could be improved in a number of important ways.

One of the problems of our method is from the usage of inverse kinematics. The major difficulty with inverse kinematics stems from the fact that the problem is ill posed [17]. In other words, there are in general many (possibly infinite) solutions that will put the end effector of a redundant arm at a given point in the workspace. However, in our method, we only choose a single solution returned from IKAN that may not be a good one in the sense that it will collide with obstacles. This is a tradeoff between efficiency and completeness. How to explore all the possible solutions based on the environment information and make the algorithm resolution-complete while still running at interactive rates will be the next task we will work on. This will greatly enhance the planner's ability to solve various planning problems in complex environments.

The evaluation function for the grid search is simple. It does not take the environment information into account. In cases where the arm has to reach around an obstacle, the algorithm will rely on backtracking to go around the obstacle, which increases the entire planning time. A more sophisticated metric may be helpful.

The resulting end effector path computed from our method is jerky because of the cellular property of the grid search. Measures need to be taken to generate smoother path.

Another limitation of the current searching procedure is from the fixed discretization resolution of the workspace. It does not distinguish relatively open environments and rather cluttered ones. To help the algorithm automatically adapt to environments with different complexities, perhaps a multi-resolution or an adaptive strategy would be more appropriate.

For collision detection purposes, the environment is rendered completely 1~3 times (for 3 links) per step and the rendering time increases with the complexity of the environment. This is one of the critical factors that affect the performance of the algorithm. We are planning to use some geometry culling scheme to improve the rendering efficiency. Optimizations can also be made to reduce the depth buffer read-back time.

In computer animation, natural motions are vitally important for animated characters. The issue is not addressed in its current implementation of the algorithm. As suggested in [9], it may be possible to encode

aesthetics as search criteria to use during planning. Efforts toward generating natural motions for the human arm are in the progress, mainly focusing on using strength information [15].

References

[1] Badler, N., Bindiganavale, R., Granieri, J., Wei, S., Zhao, X. "Posture Interpolation with Collision Avoidance", *Computer Animation*, Geneva, Switzerland, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 13-20.

[2] Breen, D. E., "Choreographing Goal-Oriented Motion Using Cost Functions", In N. Magnenat-Thalmann and D. Thalmann, editors, *State of the art in Computer Animation*, Springer-Verlag, New York, NY, 1989, pp. 141-151.

[3] Chang, H., Li, T., "Assembly Maintainability Study With Motion Planning", *Proceedings of International Conference on Robotics and Automation,* 1995.

[4] Finn, P. W., Kavraki, L. E., Latombe, J. C., Motwani, R., Shelton, C., Venkatasubramania, S., Yao, A., "Rapid: Randomized Pharmacophore Identification for Drug Design", *Proceedings of 13th ACM Symposium on Computational Geometry (SoCG'97)*, 1997.

[5] Foskey, M., Garber , M., Lin, M. C., Manocha D., "A Voronoi-Based Hybrid Motion Planner for Rigid Bodies", *Proceeding of IEEE/RSJ International Conference on Intelligent Robots and Systems,* 2001.

[6] Hoff, K. III, Culver, T., Keyser, J., Lin, M. C., Manocha, D., "Interactive Motion Planning Using Hardware-Accelerated Computation of Generalized Voronoi Diagrams", *Proceedings of IEEE International Conference on Robotics and Automation*, 2000.

[7] Hwang, Y. K., "Gross Path Planning – A Survey", *ACM Computing Surveys*, Vol. 24, No. 3., 1992.

[8] Koga, Y., Kondo K., Kuffner J. J., Latombe, J.-C., "Planning Motions with Intentions", *Proceedings of ACM SIGGRAPH 1994*

[9] Kuffner, J. J., *Autonomous Agents for Real-Time Animation*. Ph.D. thesis, Stanford University, 1999.

[10] Kuffner, J. J., Latombe, J-C., "Interactive Manipulation Planning for Animated Characters", *Pacific Graphics'00*, Hong Kong, 2000.

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

[12] Latombe, J.-C., "Motion Planning: A Journey of Robots, Molecules, Digital Actors, and Other Artifacts", *International Journal of Robotics Research*, 1999, 18(11): 1119-1128.

[13] Lavalle, S. M., Kuffner, J. J., "Randomized Kinodynamic Planning", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'99)*, Detroit, MI., 1999.

[14] Lavalle, S. M., "Rapidly-Exploring Random Trees: A New Tool for Path Planning", Preliminary manuscript available at http://janowiec.cs.iastate.edu/~lavalle/, 1999.

[15] Lee, P., Wei, S., Zhao, J., Badler, N. I., "Strength Guided Motion", *Proceedings of ACM SIGGRAPH*, Vol. 24, No. 4., 1990.

[16] Lengyel, J., Reichert, M., Donald, B. R., Greenberg, D. P., "Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware", *Proceedings of ACM SIGGRAPH 1990*, Vol. 24, pp. 327-335.

[17] Mel, B. W., *Connectionist Robot Motion Planning: A Neurally-Inspired Approach to Visually-Guided Reaching,* Academic Press, Inc., 1990.

[18] Pisula, C., Hoff, K. III, Lin, M. C., Manocha, D., "Randomized Path Planning for a Rigid Body Based on Hardware Accelerated Voronoi Sampling", *Proceedings of Workshop on Algorithmic Foundations of Robotics,* 2000.

[19] Tolani, D., *Analytic Inverse Kinematics Techniques for Anthropometric Limbs*. Ph.D. theis, University of Pennsylvania, 1998.

[20] Tolani, D., Goswami, A., Badler, N. I., "Real-time Inverse Kinematics Techniques for Anthropomorphic Limbs", *Graphical Models*, 2000, 62: 353-388.

[21] Zhao, J., Badler, N. I., "Inverse Kinematics Positioning Using Nonlinear Programming for Highly Articulated Figures", *ACM Transactions on Graphics* 1994, 13(4): 313-336.

[22] Zhao, X., Badler, N. I., "Near Real-Time Body Awareness", *Computer-Aided Design*, 1994. 26(12): 861-868.

[23] Badler, N. I., Phillips, C. B., Webber, B. L., *Simulating Humans: Computer Graphics, Animation, and Control*, Oxford University Press, 1999.

[24] Garber, M., Lin, M. C, "Constraint-Based Motion Planning Using Voronoi Diagrams", *Proceedings of Fifth International Workshop on Algorithmic Foundations of Robotics (WAFR),* 2002
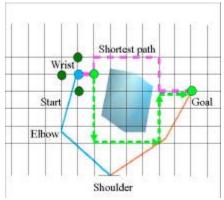
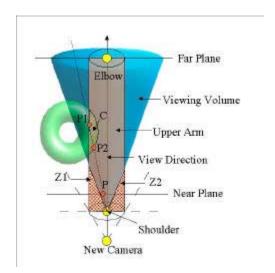Figure 1. Search path for end effector

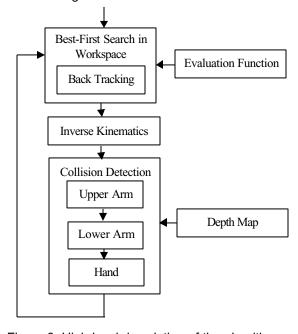Figure 2. Collision detection

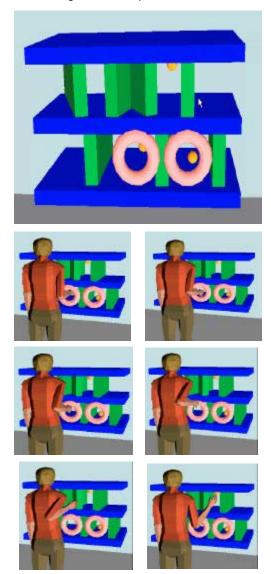Figure 3. High-level description of the algorithm

Figure 4. Sample environment 1

Figure 5. Sampled image sequence from a reaching task