February 1992

Collision-Free Path and Motion Planning for Anthropometric Figures

Wallace S. Ching
University of Pennsylvania

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

University of Pennsylvania Department of Computer and Information Sciences Technical Report No. MS-CIS-92-09.

Collision-Free Path and Motion Planning for Anthropometric Figures

Abstract

This paper describes a collision free path planning and animation system for anthropometric figures. It can also take into consideration the strength limit of human figures and plan the motion accordingly. The algorithm breaks down the degrees of freedom of the figure into *Cspace groups* and computes the free motion for each of these groups in a sequential fashion. It traverses the tree in a depth first order to compute the motion for all the branches. A special playback routine is then used to traverse the tree in a reverse order to playback the final motion. Strength value measures are incorporated directly into the searching function so that path computed will obey strength availability criteria. The planner runs in linear time with respect to the total number of Cspace groups. The planner can interface with other simulation techniques to simulate complex human motions. We believe that the planner would find a path in most cases and is fast enough for practical use in a wide range of computer graphics applications.

Comments

University of Pennsylvania Department of Computer and Information Sciences Technical Report No. MS-CIS-92-09.

Collision-Free Path and Motion Planning For Anthropometric Figures

MS-CIS-92-09
GRAPHICS LAB 48

Wallace Ching
Norman I. Badler

University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

February 1992

Collision-Free Path and Motion Planning for Anthropometric Figures

Wallace Ching
Norman I. Badler
Phone: (215) 898-1976
Email: ching@graphics.cis.upenn.edu
badler@central.cis.upenn.edu
Computer Graphics Research Laboratory
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, Pennsylvania 19104-6389

Abstract

This paper describes a collision free path planning and animation system for anthropometric figures. It can also take into consideration the strength limit of human figures and plan the motion accordingly. The algorithm breaks down the degrees of freedom of the figure into *Cspace groups* and computes the free motion for each of these groups in a sequential fashion. It traverses the tree in a depth first order to compute the motion for all the branches. A special playback routine is then used to traverse the tree in a reverse order to playback the final motion. Strength value measures are incorporated directly into the searching function so that path computed will obey strength availability criteria. The planner runs in linear time with respect to the total number of Cspace groups. The planner can interface with other simulation techniques to simulate complex human motions. We believe that the planner would find a path in most cases and is fast enough for practical use in a wide range of computer graphics applications.

1 Introduction

Collision free path planning has applications in a variety of fields such as robotics task planning, computer aided manufacturing, human figure motion studies and computer graphics simulations. A collision free path for an articulated figure is the path along which the articulated figure moves from an initial configuration to a final configuration without hitting any obstacles residing in the same environment as the articulated figure.

A great deal of research has been devoted to the motion planning problem in the area of robotics within the last 10 years, e.g. [16] [17] [18] [5] [6] [4] [8] [9] [14] [11]. However, despite the applicability of motion planning techniques to computer graphics simulations, the problem has not been addressed much in the computer graphics community [13].

In this paper, we are going to present a motion planning system for anthropometric figures. Computer graphics human figure modeling has been used in different engineering applications like car occupant studies, space station design, product safety studies and maintenance assessment. An efficient system for task level simulation of human figures should possess automatic collision free path generation capability as this will alleviate the burden of the user to specify every key movement in the task study. Our motion planning system can also take into consideration the strength data of human figures so that the planned motion will obey strength availability criteria [15].

Human figures are articulated figures characterized by a branching tree structure with many degrees of freedom (DOFs). Existing algorithms in robotics fall short in handling some of the issues encountered when dealing with these types of figures. In this paper, we are going to present novel algorithms that can address all these issues. The basic idea is that instead of treating all the DOFs in the figure together, we divide them up into groups and treat these groups one by one. The algorithm is not a complete algorithm, but rather an approximate one. We justify this trade off in completeness for a gain in speed. The algorithm runs in $O(n)$ where n is the total number of groups in the figure rather than the conventional exponential time if all DOFs are treated together.

2 Background

Let us look at some of the related work done by robotics researchers. The major challenge of our problem is to handle a redundant branching articulated figures with many degrees of freedom. Many of the robotics algorithms deal with manipulators with relatively few degrees of freedom, e.g. the mobile robots which typically have three degrees of freedom and the PUMA type of robots which have six. Many of these algorithms are based on the use of the configuration space (C space) which is the space of the degrees of freedom of the robot [16, 18]. The inherent difficulties with this approach is due to the high dimensionality of the C space. It is well known that the worst case time bound for motion planning for a robot arm is exponential in the dimensionality of its C space [24, 25]. It is only during the last few years that motion planning algorithms that can handle manipulators with many degrees of freedom have been presented [2, 1, 3, 12, 10].

However, very few of the work consider articulated figures with branches. Barraquand *et al* gave an example involving a manipulator with 2 branches [2, 1, 3]. In their work, they create an artificial potential field in the 3-D workspace and the free path is found by tracking the valleys. A gain in efficiency is obtained as a result of the clever selection of potential functions and heuristics. However, it is not clear how these can be selected in general.

Faverjon *et al* [10] presented a method which partitions the free space into octrees and uses some probability measures to cut down the search tree during the A* search. Gupta [12] has presented another technique to handle sequential linkages with many degrees of freedom. It is a sequential search technique which basically treats the individual degrees of freedom one by one instead of considering all of them together. The initial stage of our path planner is based on his work.

In this paper we are going to present a path planning system that can provide a satisfactory and efficient solution to the collision avoidance problem for redundant branching articulated figures such as the human figures. The path found will also obey the strength limits of the human figures. Finally examples are presented to demonstrate the usefulness of the system in producing realistic human motion simulations.

3 The Approach

The main idea of our path planner is to handle the DOFs of the articulated figure not all at once but a certain number at a time. Gupta devised a sequential strategy that plans the motion of a sequential linkage by considering one DOF at a time [12]. We extend the idea further and introduce the notion of *Cspace groups* (C groups). We also come up with novel algorithms that can handle articulated figures with branches and many degrees of freedom.

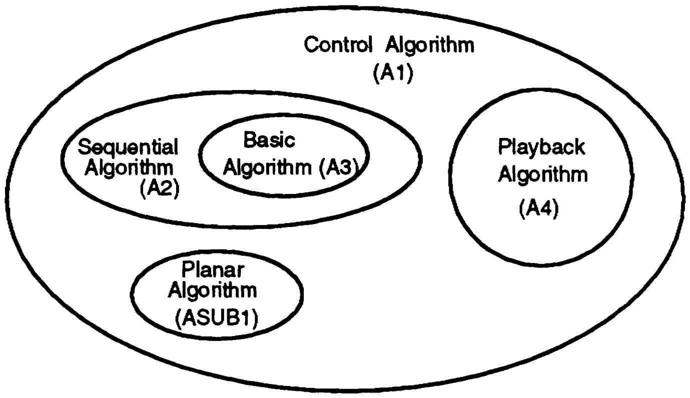


Figure 1: Different modules and their associated algorithms in the path planning system

The general principle of our path planner is first to divide up the DOFs in the articulated figure into a number of groups which we call *Cspace groups*. We then compute the collision free motion for the DOFs in each group successively, starting from the first group. After the motion of the DOFs in group i has been planned, we parameterize the resulting motion with a parameter t. The motion for the DOFs in group $i+1$ will then be planned along this path by controlling the DOFs associated with it. The problem is then posed in a $t \times \theta^k$ space if there are k DOFs in this group. We proceed in this manner along the whole figure structure and solve for the motion for all the groups. Finally a playback routine is invoked to playback the final collision free path for the figure.

Our system adopts a modular design in that it is made up of a number of modules each of which is based on an algorithm (Fig. 1). Each module carries out a particular function and contributes to the whole path finding process.

On a more global perspective, the path finding procedure can be viewed as consisting of two phases: the *computation* phase and the *playback* phase. All of the steps involved in these phases are performed by the algorithms described in later sections.

The overall path planning procedure is outlined as follows:

- **Computation** Phase:

  1. Partition the degrees of freedom of the articulated figure into *Cspace groups* according to a grouping scheme.
  2. Impose an order of traversal among the Cgroups. For human figure, we use a depth first traversal. This means we plan the motion for one arm and then another.
  3. Invoke the *control algorithm* that handles traversal of the tree and finds the final collision free path. This algorithm will actually call upon a subsidiary algorithm, *sequential algorithm*, to compute the free path along a branch of the tree structure. The *sequential* algorithm will in turn call another subsidiary algorithm, the *basic algorithm*, to compute the path for the DOFs within each Cgroup.

- **Playback** Phase:

  After all the Cspace groups have been considered, a special *playback algorithm* will be called upon to traverse the tree structure in a reverse order, collect and coordinate all the computed information and finally playback the overall collision free path in discrete time frames. These time frames can be futher interpolated to produce a smooth simulation.

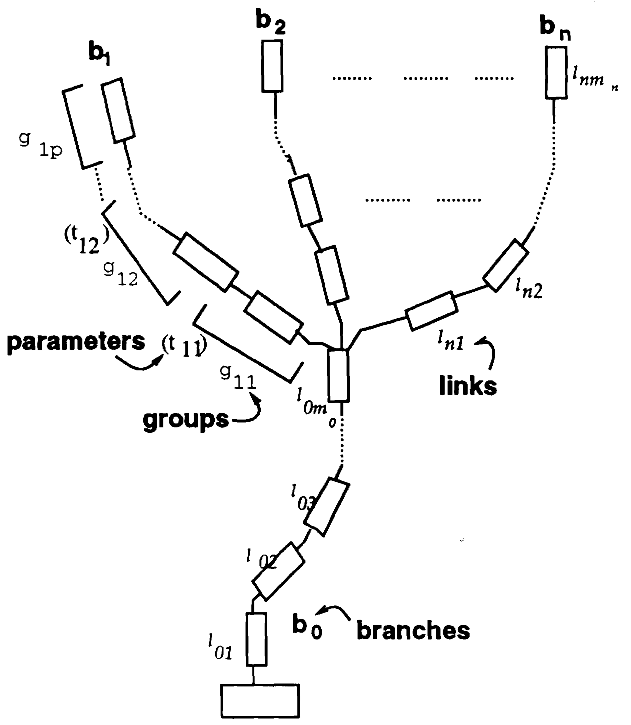


Figure 2: The redundant branching articulated figure considered in this work

The translational movement of the articulated figure as a whole on a plane can also be planned with our planner. In this case, the figure resembles a mobile robot with two degrees of translational freedom and one degree of rotational freedom. The module that handles this case is named the *Planar Algorithm.*

Fig. 2 shows the general redundant branching articulated structure that we will use for reference in the description of our path planning algorithms. The figure also shows the symbols that we will be using in our explanations. Our discussions will mainly focus on the upper body of the human figure. The system can be easily applied to the legs to provide stepping movement, as shown in our last rock climbing example (Fig. 18).

4 The Basic Algorithm

The particular algorithm we have chosen is the one presented by Lozano-Perez in [17] due to its simplicity and intuitiveness. It first constructs the C space for the articulated figure. For the sake of completeness, the process is described below.

If the manipulator has n links, its configuration space can be constructed as follows:

1. $i = 1$.
2. While $(i < n)$ do
   (a) Ignore links beyond link i, and find the ranges of legal values of q_i by rotating link i around the position of joint i determined by the current value ranges of $q_1, ..., q_{i-1}$ and check for collision with the surrounding obstacles. Self collision can be avoided by checking collision with linkages from the same figure as well. This is for handling self

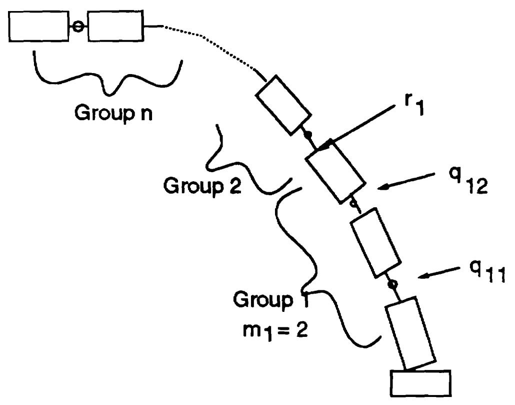


Figure 3: A Sequential linkage

collision in the figure. Mark those joint values at which link i will have a collision as forbidden.

(b) Sample the legal range of q_i at the specified resolution.

(c) Increment i and repeat step (a) for each of these value ranges.

The free space is then represented by the data structure called *regions* to explore the connectivity between the cells. A graph is then built on these region nodes and an A* search is conducted to search for a free path from the start node to the goal node.

5 The Sequential Algorithm

5.1 Overview

The *Sequential Algorithm* handles the motion planning problem for the Cspace groups along a sequential branch. This algorithm is based on Gupta's work. We will discuss some of the differences after presenting the algorithm.

5.2 The Algorithm

Referring to Fig. 3, let n be the total number of *Cspace groups* on this branch. Let the joint DOFs associated with the groups be represented by q_{ij} where i is the group number and j is from 1 to m_i where m_i is the maximum number of DOFs group i has. Let r_i be the reference vertex for group i. It is basically the distal vertex of the link associated with the DOFs in the group i. Let $r_i(t)$ denote the trajectory of the reference vertex r_i. The initial and goal configurations of the arm are given as q^s_{ij} and q^g_{ij}, i=1..n; j = 1..m_i.

The algorithm is as follows:

1. Compute a collision free trajectory for the links associated with group 1. The trajectory of the reference vertex on its link will be $r_1(t)$.

2. $i = 2$.

3. While ($i < n$)

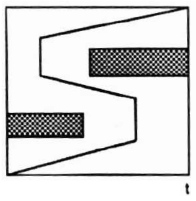


Figure 4: An example showing the case that a path can only be found with backtracking which means the parameter takes on a non-temporal interpretation.

(a) along $r_{i-1}(t)$, discretize the path according to a pre-specified resolution. Compute a collision-free trajectory for the DOFs in the ith group from q_{ij}^s to q_{ij}^g for $j = 1..m_i$ using the *basic* algorithm described in the last section.

(b) given $q_{1j}(t), q_{2j}(t), ..., q_{ij}(t)$, compute $r_i(t)$ using forward kinematics.

(c) Increment i.

5.3 Interpretation of the Parameter

The parameter used in parameterizing the path already computed can be either interpreted as temporal or non-temporal. For a temporal interpretation of the parameter, the path computed has to be monotonic with respect to the parameter t simply because we cannot travel backward in time. Hence backtracking is not allowed and the chance of finding a path is greatly restricted. In the example shown in Fig. 4, we will not be able to come up with a path without backtracking.

In our system, we have adopted a non-temporal interpretation of the parameter in most cases as this will increase the chance of finding a path.

5.4 Discussion

- Gupta considered one DOF at a time in his work. We extend this idea further and introduce the notion of *Cspace groups* (C groups). Each C group deals with one parameter and a certain number of DOFs. The number of DOFs can vary between C groups so as to fit into the structure of the figure. For example, the shoulder joint can be handled by one C group with 3 DOFs.

  The number of DOFs handled at a time also affects the degree of optimality of the resulting path (with respect to some criteria). Theoretically, the optimal path can only be obtained by searching through the n-dimensional C space built from considering all n DOFs together. However, such an algorithm has been proven to be exponential in the dimensionality of its C space [24]. That is also the reason that motivates us to treat a subset of the DOFs at a time. Apparently, there is a trade off between speed and optimality. The compromise selection from the spectrum of choices is usually dictated by the hardware available as well (Fig. 5).

  Dealing with more than 1 DOF at a time also greatly enhances the chances of finding a free path. With a higher dimensional space, backtracking is easier and more room is available for exploration.

- Gupta has chosen to use *visibility graph* for representing the free space and searching for a solution. The path found will have the undesirable effect of being very close to an obstacle. This will leave very little room for the next link to manuever (Fig. 6). Our choice of using

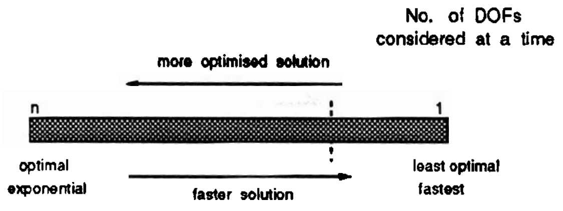


Figure 5: A spectrum of choices in picking the number of DOFs in a group

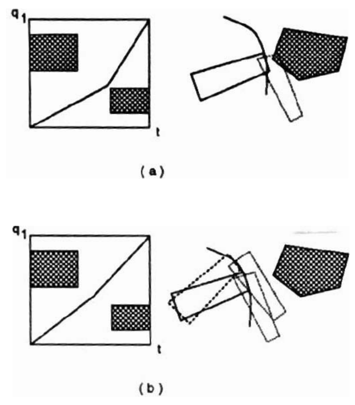


Figure 6: (a) A path that is too close to an obstacle. This leave little room for the next linkage to maneuver. (b) A better path that is farther away from the obstacle

the region graph allows for the path to be positioned farther away from the obstacles, hence leaving more room for the next linkage.

6 The Control Algorithm

6.1 The Control Algorithm (A1)

1. Apply Algorithm $ASUB1$ (the Planar Algorithm, see next section) to the whole figure to obtain the planar collision free translational movement of the figure taken as a whole.

2. Parameterize the resulting motion. The normalized constant for this motion is called $s_{initial}$.

3. Apply Algorithm $A2$ (the *Sequential* algorithm) to branch b_0 with $s_{initial}$ as the first parameter for the first group of the branch, i.e. group $g_{0,0}$.

4. Parameterize the resulting path computed for group g_{0,p_0} according to some prespecified resolution. The normalized parameter is named s_0. The trajectory of the reference point on branch b_0 is referred to as $r_0(t_1)$.

5. $i = 1$

6. While $(i < n)$ do

   (a) Apply Algorithm $A2$ to branch b_i. Use s_{i-1} as the parameter for the first group of this branch, i.e. group $g_{i,0}$.

   (b) Parameterize the resulting path with some prespecified resolution.

(c) Invoke the Algorithm $A4$ (the Playback Algorithm) to branch b_i to obtain the sequence of joint angle values of branch b_i when moving along the computed path.

(d) Record this sequence of joint angle values in the array $FREEANGLES_i[1..N_{free}]$ where N_{free} is the total number of joint angle values recorded along this path.

(e) A normalized parameter, s_i, is defined to index into the array $FREEANGLES_i$ through the linear mapping:

$$Map(s_i) : \{0..1\} \rightarrow \{1..N_{free}\}$$

(f) Increment i

7. (Now $i = n$, the last branch) Apply Algorithm $A2$ (the *Sequential* algorithm) to branch b_n. Discretize the resulting path according to some resolution.

8. Apply Algorithm $A4$ (the playback algorithm) to the whole figure, starting from this very last group of the very last branch.

9. The angle values obtained can then written into frames for continuous playback.

7 The Planar Algorithm (ASUB1)

The articulated figure can translate and rotate on a plane, navigating around obstacles. The whole figure behaves just like a mobile robot. The path planning algorithm in this case deals with a 3-dimensional (2 translational, 1 rotational) Cspace. We can handle this case simply with our *basic* algorithm (A3) or other existing mobile robot path planning techniques.

8 Resolving Conflicts between Different Branches

Although the different branches are attached to the same rear link of branch b_0, we do not use the same parameter t_{0,p_0} that parameterizes the motion of branch b_0 in all these branches. The reason is that the parameters t_{ij} are interpreted as non-temporal in general. Hence, backtracking is allowed and the values of t_{ij} along the computed path can be nonmonotonic. If we use the same parameter in computing the motion for the first groups in all other branches, some of the joint angle values cannot be obtained uniquely during the final playback phase. This reasoning may become clear after looking at the playback algorithm.

Our solution to this problem is to further *parameterize* the already *parameterized path* of the previous branch. This is described in step 6(e) of the *Control Algorithm* listing in the last section and is further explained below.

Let us look at Fig. 7(a). After we have computed motion for branch b_0, we parameterize the resulting path with the parameter t_{0,p_0}. We create another variable s_0 here and set it equal to t_{0,p_0}. The curved path shown in the diagram represents the computed motion for branch b_0. Next, we go on to compute the path for branch b_1. The parameter value used in the first group, i.e. $t_{1,1}$, is assigned to be s_0 which is the parameter passed along from the previous branch. We proceed in a similar manner and compute the path for the whole branch. The resulting configuration space and the free path for the *last* group of the branch, i.e. g_{1,p_1}, may look like the one in Fig. 7 (a).

As we follow the computed path, it can be seen that the values taken by the parameter t_{1,p_1} along the path are not necessarily monotonic. We choose to parameterize this path again before going on to compute the path for the next branch. We will call this parameter along the path s_1

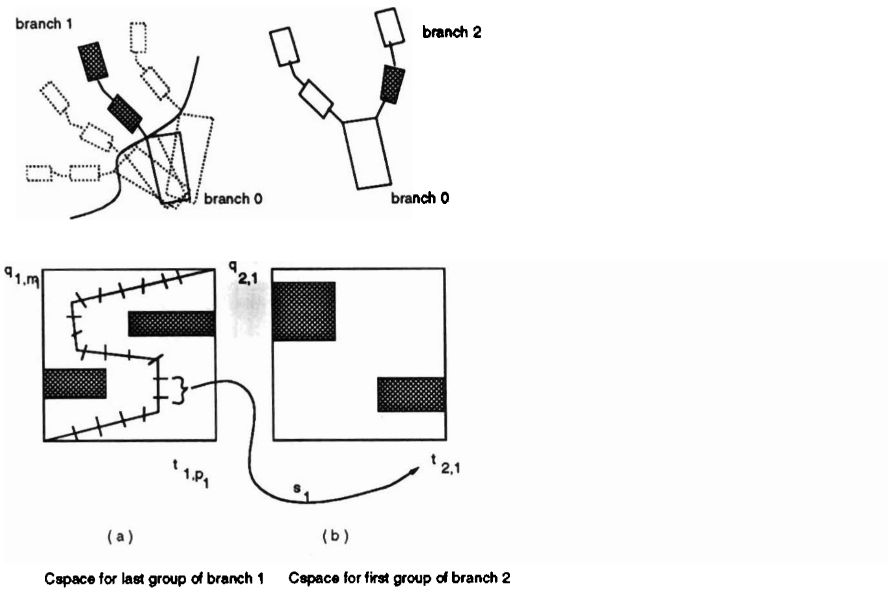


Figure 7: (a) Diagram above shows the parameterized path of the torso and the computed path for branch b_1. Diagram below shows the corresponding Cspace for the last group of branch b_1 (b) The Cspace and computed path for the first group of the branch b_2.

and it is this parameter that we are using when dealing with the first Cspace group of the next branch as shown in Fig. 7 (b).

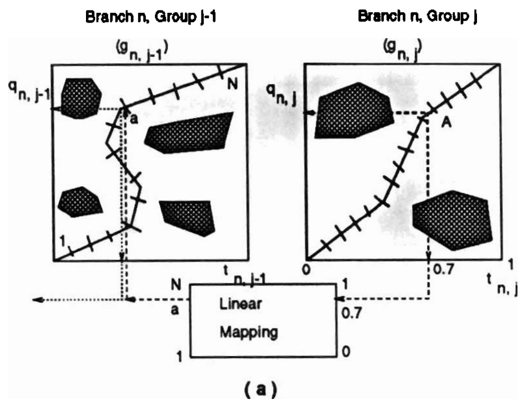
9 Playing back the free path

During the playback phase, we start from the *last group* of branch b_n and then traverse the branches in a backward manner along branch b_{n-1}, b_{n-2} and so on and finally to branch b_0.

For example, let Fig. 8 (a) represent the configuration space for the *last group* of the last branch, i.e. group g_{n,p_n} of branch b_n. We then discretize the free path according to a pre-specified playback resolution. The number of discretization intervals for this last group will be equal to the number of *time frames* for the final simulation.

At every discretized point, say A, there is a corresponding (q, t) pair: the q value is what we should set the last joint DOF to, and the *parameter t* is used to deduce the motion of the preceding group. We first set the last DOF to the q value. Then we use the parameter t in the pair to trace back to the *preceding* (proximal) group. Note that within this preceding group, the parameter t is *monotonic* by defintion. Hence we can uniquely determine the corresponding (q, t) pair within this preceding group. By the same token, we can continue tracing back to the groups further preceding this one (Fig. 8 (a)). We carry on in this fashion recursively until we come to the first group within this branch.

Note that at this point, all joint DOFs along this branch will have been set to their correct value for this simulation time frame. The sequence of joint values along the free path for all the other branches should have also been recorded in the array $FREEANGLES_i$. The parameter value left unused is the first parameter of the first group of the last branch, i.e. $t_{n,1}$. It is actually the same as s_{n-1} as described in the last section. As discussed then, it is monotonic in values by definition. Hence, we can use this as an index into the recorded joint angle array and uniquely determine the set of angles corresponding to the movement of the preceding branch.



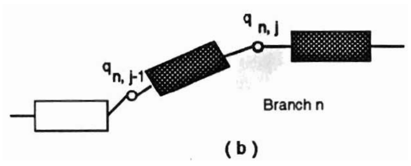


Figure 8: An example showing how the final joint angle values of the whole figure are obtained from the Cspace associated with the Cspace groups.

After setting the angles in branch b_{n-1}, we proceed to treat all other branches in a similar manner.

The details of the playback algorithm is as follow:

9.1 The PlayBack Algorithm (A4)

For easy understanding, we will explain the algorithm by looking into its two components separately: the Final Playback Algorithm (A4a) and the Single Branch Single Frame Playback Algorithm (A4b).

9.1.1 The Final Playback Algorithm (A4a)

In this algorithm, we enter into a simulation loop and will generate all the time frames for this simulation. The variable k in the outer loop can be thought of as the frame index.

- Discretize the path computed for the last group in the last branch into N_{final} discrete points according to some pre-specified resolution. This is illustrated in the diagram on the right in Fig. 8 (a). This number also determines the total number of key postures or time frames we will generate for the final simulation.
- Let $k = 1$.
- While ($k < N_{final}$) do
  1. Apply Algorithm *A4b* (the Single Branch Single Frame Playback Algorithm) to branch b_n, the last branch in the figure.
  2. The parameter value, $t_{n,1}$, will be obtained at the termination of the Algorithm *A4b*. We then use this parameter as an index into the array $FREEANGLES_{n-1}$. The joint angles recorded for branch b_{n-1} will be read off from the array element pointed to by this parameter value.

3. Set the joint angles in branch b_{n-1} to the values read off from the array.
4. Let $z = t_{n-1,1}$, the first parameter value of branch b_{n-1}, which is also read off from the array $FREEANGLES_{n-1}$.
5. Let $i = n - 2$, the third last branch in the articulated figure.
6. While $(i > 0)$ do
   (a) Use the value of z as an index into the array $FREEANGLES_i$. Read off the joint angle values stored in the array elements.
   (b) Set the joint angles in this branch to the values obtained from the previous step.
   (c) Set $z = t_{i,1}$, the first parameter which is also read off from the array $FREEANGLES_i$.
   (d) Decrement i.
7. Now we have finished setting the values for all the branches except the first branch b_0. Apply Algorithm *A4b* to this branch to get back its computed joint angles values and set the corresponding joints to these values.
8. Now all the joint angles in the articulated figure have been set to their appropriate values in this time frame. What is left is the *position* of the whole figure. The last parameter value obtained from the last step is used to index into the path computed from the Planar Algorithm. Then we set the whole figure location to that indexed position.
9. Advance the simulation time step by incrementing k and repeat the whole playback process for the next time frame.

9.1.2 The Single Branch Single Frame Playback Algorithm (A4b)

Let the branch index we are considering be i. Here branch i has a total of p_i groups. This playback algorithm is called only after the motion for the last group in the branch is computed.

We present the algorithm as follow:

1. This algorithm only deals with one discretized point, and hence only one time frame. Let this discretized point be the kth point on the path.

2. Let $j = p_i$. j here is the group number index. We start from the last group in the branch and go *down* the branch by decrementing j.

3. Let $r = k$. r is used as a loop variable to pass down the time frame index down the branch.

4. While $(j > 0)$ do

   (a) From the rth discrete point on the computed path, read off the values of the $q_{i,j}$s associated with this Cspace group from the axes of the Cspace. This is illustrated in Fig. 8 (a) with a 2-dimensional Cspace as an example.
   (b) Set the joints in the articulated chain corresponding to these qs variables to the values we have just found.
   (c) Then read off the normalized parameter value $t_{i,j}$ from the t axis.
   (d) Through a linear mapping, we can obtain the corresponding discretized point on the path computed for group $g_{i,j-1}$ from this parameter value.
   (e) Set $r = index$
   (f) Decrement j

Note after this algorithm terminates, all the joint angles on the this branch will be set to the appropriate values for this simulation time step.

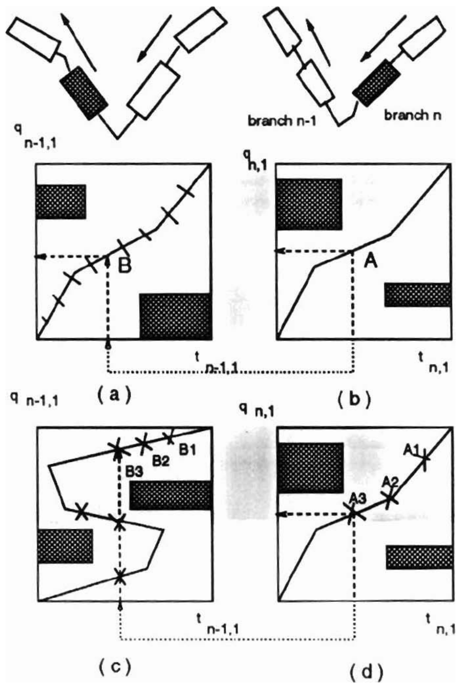


Figure 9: The procedure for setting the angles in the final playback phase. (a,b) shows cases where no backtracking is used. (c,d) show cases where backtracking occurs in one branch

10 Further explaining the conflict between different branches

Now that the playback mechanism is explained, we can explain in more detail the potential conflict that can happen between branches attached to the same linkage.

Let us look at Fig. 9. Fig. 9 (a) shows the Cspace for the first group of branch b_{n-1}, the second to last branch. Fig. 9 (b) shows the one for the first group of branch b_n which is the last branch of the tree.

Recall the previous argument that since the two branches are attached to the same linkage b_0, we may be tempted to use the same parameter t_{0,p_0} in computing the motion for both branches. If this is the case, then $t_{n-1,1} = t_{n,1}$. Let us see what kind of potential conflict can result from this arrangement.

During the playback phase, when we come to the first group of the last branch, (i.e. group $g_{n,1}$ of branch b_n) we read off the parameter value $t_{n,1}$ from the horizontal axis (point A in Fig. 9 (b)). Since this is equal in value to $t_{n-1,1}$, point B must be the corresponding point for branch b_{n-1} (Fig. 9 (a)). Then the corresponding value of q can be read off from the vertical axis.

This looks fine in Fig. 9 (a) and (b). However, since we have allowed the parameters t_{ij} to be nontemporal, they need not be monotonic along the computed path. This can create a problem as shown in Fig. 9 (c) where backtracking is adopted to find a path. When using the $t_{n,1}$ value obtained in Fig. 9 (d) to get the corresponding point for the branch b_{n-1} shown in Fig. 9 (c), we get multiple values and several possibilities. This is still solvable by keeping track of the history of points obtained so far and look for the nearest match. For example, points B1, B2 are the corresponding match for points A1, A2. Then naturally point B3 (shown in diagram (c)) should be the one for A3.

This may work in some cases but not in cases where backtracking is adopted in both Cspaces as shown in Fig. 10 (a) (b). In this case, we can find only only one match point in Fig. 10 (a) (for

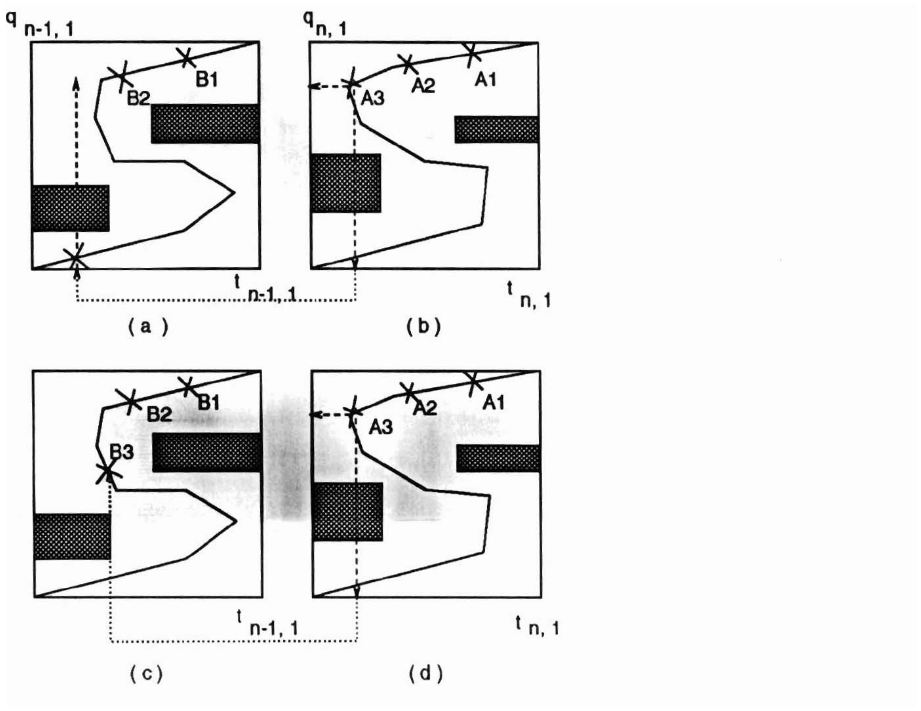


Figure 10: (a,b) A conflict occurs resulting in no feasible solution. (c,d) A solution for resolving such conflicts

point A3) which however is not a feasible match as the middle segment of the computed path will be truncated.

The solution to all these problems is to add another level of parameterization. Instead of using the same parameter value in both Cspaces, we use the parameter obtained from parameterizing the first path in computing motion for the second one. As shown in Fig. 10 (c) and (d), the parameter $t_{n,1}$ can now be used to uniquely determine the corresponding match point as its value is guaranteed to be monotonic along the first path by its defintion.

11 Strength Guided Motion

Lee *et al* [15] have demonstrated that realistic animation of lifting motions can be generated by considering the strength of the human figure. The basic premise of the method is that a person tends to operate within a *comfort* region which is defined by the amount of available torque. Their method makes use of a strength model of the human body. The path planner incrementally updates the next joint angles values according to the *available torque* at the current configuration based on a number of motion strategies.

However, their method does not consider any obstacle avoidance during the path planning process, and it is not obvious how the method can be extended to handle the issue. We are here proposing a solution that incorporate the strength model into our collision free path planner in a bottom up fashion. The problem poses a unique challenge that has not been addressed before.

11.1 Computation of Required Torque

The static torque values at all the joints required to sustain a load is a function of all its joint angles. That is, it is a function of the figure configuration.

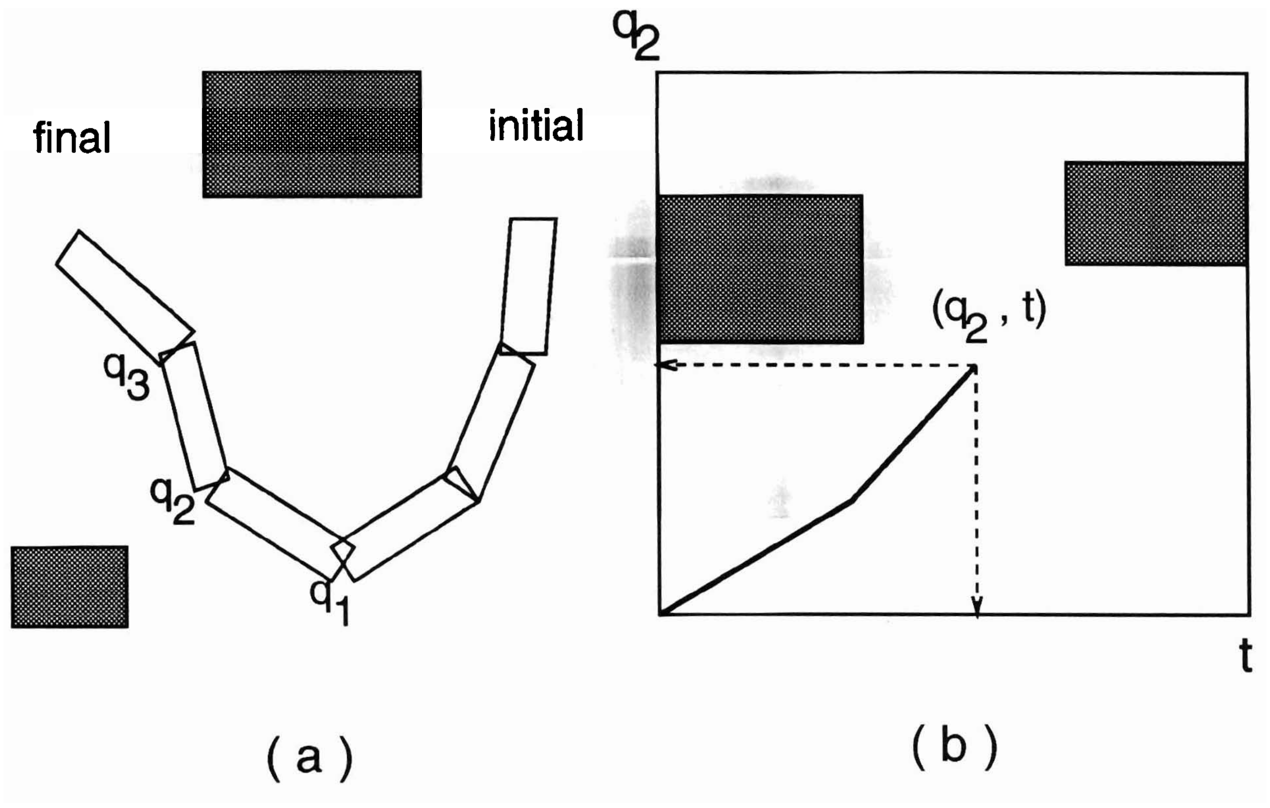


Figure 11: An illustration for the torque computation method

The methodology employed by our algorithms is to divide up the degrees of freedom into groups and plan the motion for each group sequentially. Therefore, only after the complete algorithm terminates do we have the complete path for each degree of freedom. However, we need to make use of the strength information *during* the planning process. This requires values of all joint angles at a certain configuration. Since we would have computed only a subset of all the joint angles, we need to have a procedure to realistically estimate the rest of the joint angles at this configuration.

This is illustrated in Fig. 11 in which a 3 link manipulator is taken as an example. The manipulator has 3 joint angles: q_1, q_2 and q_3. Assume we are planning the joint angles one at a time and we have planned the motion for joint 1. Now we are planning the motion of joint 2. To compute the required torque at a certain point, say point A, we need the values of all 3 angles at that point. As shown on the diagram, q_2 can be read off directly from the axis. q_1 can be obtained through the current value of the parameter t. However, q_3 is still unknown.

The solution is to use the value of the current parameter and project it over to the rest of the angles that have not yet be computed. Referring to Fig. 12, let the set of angles we have computed be denoted Φ and those that are yet to be computed be Ψ. Let the value of the normalized parameter we are currently working with be t. We obtain the projected values for the angles in Ψ by using the following projection function:

$$\forall \theta \in \Psi,\ \theta_{proj} = PROJ(t)$$

where $PROJ(t)$ is :

$$PROJ(t) : \{0..1\} \rightarrow \{\theta_{initial}..\theta_{final}\}$$

The projection function $PROJ$ can be a simple linear interpolation mapping as:

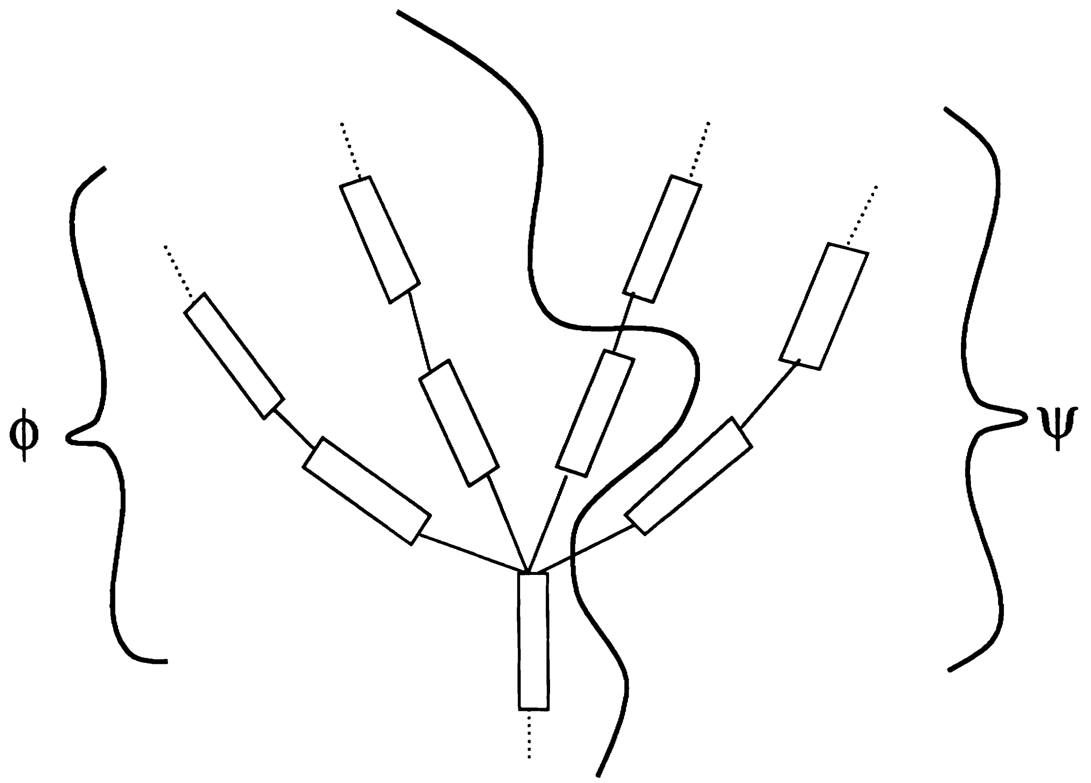


Figure 12: The two set of angles, one computed and the other to be computed

$$PROJ(t) = \theta_{initial} + t(\theta_{final} - \theta initial)$$

Beside linear interpolation, other types of interpolation relationship may be explored. For example, a quadratic relationship may be used instead as shown on Fig. 13. This implies that the particular degree of freedom under consideration will spend more time near its final value than its initial value.

We can explore this idea further by using a different interpolation relationship for each joint based on its movement characteristics.

11.2 The Strength Function used in Heuristic Search

Recall that in the searching process using the A* algorithm, we evaluate a heuristic function at every step and expand the node that has the smallest value. So far we have been using only a distance function in this heuristic function. The path found will be optimal up to the size of the *regions* used.

Now that we have a means to compute the required torque value of a particular configuration, we can add to this heuristic function terms that represent the strength information. The weighing factors attached to these terms represent the relative importance of the quantities they represent. Possible terms to include are as follow:

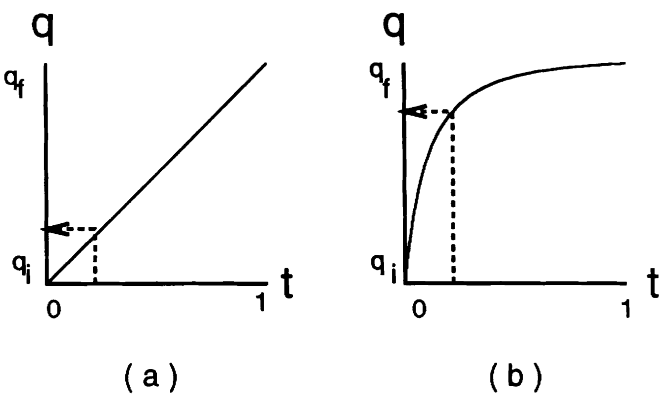


Figure 13: Different projection schemes: (a) a linear interpolation (b) a quadratic interpolation

11.2.1 Work Done

The total work done or energy expanded can be measured by the term $\int T(\vec{\theta}) \cdot d\vec{\theta}$. The integration is done over the path taken.

11.2.2 Comfort

The *comfort level* of the resulting motion can be measured by the *available torque* which is the amount obtained by subtracting the required torque from the strength limit at that particular joint configuration. We can sum up all the contributions along the path as $\int AvailTorque(\vec{\theta}) \cdot d\vec{\theta}$ where the available torque is defined in terms of its elements:

$$AvailTorque(\vec{\theta})_i = \begin{cases} Str(\vec{\theta})_i - T(\vec{\theta})_i & \text{if } Str(\vec{\theta})_i > T(\vec{\theta})_i \\ 0 & \text{otherwise} \end{cases}$$

The subscript i stands for the i-th element in the vector. Str is the strength limit vector. This integral value will then represent the overall comfort level.

This term will properly be useful only in the g function as it only affects future actions.

11.2.3 Fatigue

Humans are not like robots in that their strength will decrease with time as a result of fatigue. We may include a term like $\int \|T(\vec{\theta})\| dt$ to avoid taking a path that has a high torque value maintained over a prolonged period of time. Since we do not maintain an explicit notion of time, the normalized parameter value can be used as an approximation.

11.3 Regions vs. Uniform Grids

Recalling the fact that once we have mapped the obstacles onto the configuration space, we group the cells in the free space into basic entities called *regions* to maximize the usage of the connectivity between the cells. This has the advantage of having a smaller number of *nodes* in the graph to search for. Nevertheless, the use of regions instead of a uniform grid will also reduce the *granularity* of the C space and hence will have fewer possible paths.

Since we use *regions* as the basic entities, we choose representative paths within the regions and evaluate the integrations of the different strength measure components mentioned in the last section along these chosen paths. Fig. 14 (a) shows a sample Cspace with four regions. Region

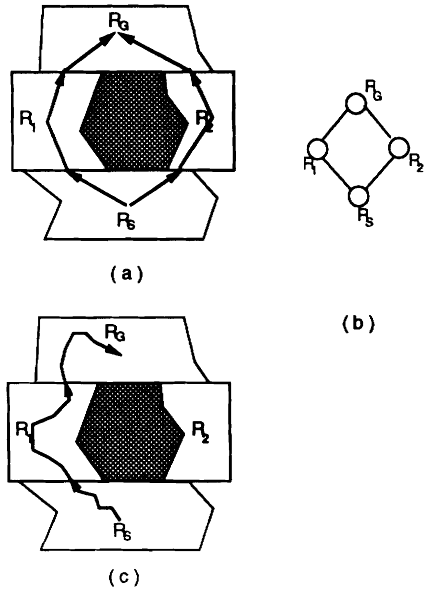


Figure 14: (a) Part of a sample C space showing two possible paths leading from the start to the goal node. (b) the corresponding regions graph.

R_S is the start region and regiion R_G is the goal region. The figure also shows the chosen paths within the four regions. Typically a path is made up of straight line segments and is chosen to go from one side of the region boundary to the next through the region *kernel* (the center area of the region). Integration is computed along the path with pre-specified increments.

11.4 Finding a Smooth Optimal Path

The path found so far is the best path found up to the size of the *regions* (the basic entities). Paths within regions are chosen by the system rather than by the search process. We can further refine this computed path by treating it as a feasible path. For example, in Fig. 14 (a), the left path may be chosen to be a better one than the one on the right. This path can then be further refined by examining local strength values and comfort level and by invoking one of the motion heuristics such as *Available Torque, Reducing Moment* and *Pull Back* as described in [15].

12 Hybrid Simulation Techniques

We believe that simulating the wide range of human motions requires a number of different simulation techniques, such as those that simulate walking [7, 19], grasping [23], lifting [15]. Therefore, it is important that our simulating techniques can interface with other existing techniques in simulating more complex human behaviors. Fig. 18 shows a human figure climbing up a rocky surface. The limbs' climbing movement and the torso translation are produced from our path planning system. At one time, only one limb is considered by the path planner. The other three limbs are held in place by imposing a hold constraint to them [22]. The positions for the hand grasp and foothold have to be specified by the user. In this case, even though our path planner cannot handle closed loop systems, , with the help of other simulation techniques, motions that involve closed loop mechanism can still be simulated.

13 Results and Discussions

Fig. 15 shows a human figure reaching through two apertures with both arms. The path computed is collision free and involves more than 20 DOFs. Fig. 16 shows a human figure lifting two objects from the lower shelves of a chest to the upper shelves with both arms. Fig. 17 shows a human figure lifting objects on a different shelf. The examples are run on a Silicon Graphics Personal Iris workstation under the *Jack*^{TM} environment - a graphics environment developed for human figure modeling at the Computer Graphics Research Laboratory at Penn [20, 21]. The examples shown here take about 12 to 30 mintues wall clock time to compute, depending on the complexity of the environment.

The main advantages of our algorithm described here are that it can deal with redundant articulated figures with branches and many degrees of freedom. It can also incoporate the strength data for human figures to produce strength guided motions.

The algorithm cannot handle closed loop mechanisms. It is an approximate algorithm, not a complete one, in the sense that it may not succeed to find a path even though there exists one. However, We can still get around this apparent disadvantage by regrouping the DOFs under a different grouping scheme and try finding the solution again if a path cannot be found for the previous grouping scheme. Backtracking can be employed between groups too.

The *basic* algorithm within a Cspace group is $O(r^{k-1}(mn)^2)$ where k is the number of DOFs, r is the discretization intervals, m is the number of faces and edges for the robot and n for the environment as shown in [17]. Since the number of DOFs in a Cspace group is bounded, the run time for the *basic* algorithm can be treated as a constant. Consequently the whole algorithm runs in $O(p)$ time where p is the total number of groups in the tree structure.

We believe that the run time of the algorithm is fast enough for practical use and that it will contribute to applications in robotics task level planning and computer graphics human figure motion simulations.

References

[1] J. Barraquand, B. Langlois, and J. Latombe. *Numerical Potential Field Techniques for Robot Path Planning.* Technical Report STAN-CS-89-1285, CS Dept, Stanford University, 1989.

[2] J. Barraquand, B. Langlois, and J. Latombe. Robot Motion Planning with Many Degrees of Freedom and Dynamic Constraints. In *Fifth Intl. Sym. on Robotics Research (ISRR), Tokyo*, pages 1–10, 1989.

[3] J. Barraquand and J. Latombe. *Robot Motion Planning: A Distributed Representation Approach.* Technical Report STAN-CS-89-1257, CS Dept, Stanford University, May 1989.

[4] Rodney A. Brooks. Planning Collision-Free Motions for Pick-and-Place Operations. *Int. Journal of Robotics Research*, 2(4):19–44, Winter 1983.

[5] Rodney A. Brooks. Solving the Find-Path Problem by Good Representation of Free Space. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-13(3):190–197, Mar 1983.

[6] Rodney A. Brooks and Tomas Lozano-Perez. A Subdivision Algorithm in Configuration Space for Findpath with Rotation. In *Proc. 8th Int. Joint Conf. Artificial Intelligence*, pages 799–806, 1983.

[7] Armin Bruderlin and T. W. Calvert. Goal-directed, Dynamic Animation of Human Walking. *Computer Graphics*, 23(43), 1989.

[8] Bruce Donald. *Motion Planning with six degrees of freedom.* Technical Report 791, MIT AI Lab, 1984. pp.504-512.

[9] Bruce Donald and Patrick Xavier. A Provably Good Approximation Algorithm for Optimal-Time Trajectory Planning. In *IEEE Intl. Conf. on Robotics and Automation*, pages 958–963, 1989.

[10] Bernard Faverjon. Obstacle Avoidance using an Octree in the Configuration Space of a Manipulator. In *IEEE Intl. Conf. on Robotics and Automation*, pages 504–512, 1984.

[11] Laurent Gouzenes. Strategies for Solving Collision-free Trajectories Problems for Mobile and Manipulator Robots. *Int. Journal of Robotics Research*, 3(4):51–65, Winter 1984.

[12] Kamal Kant Gupta.. Fast Collision Avoidance for Manipulator Arms: A Sequential Search Strategy. *IEEE Transactions of Robotics and Automation*, 6(5):522–532, Oct 1990.

[13] Bruce R.. Donald Jed Lengyel, Mark Reichert and Donald Greenberg. Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware. *Computer Graphics*, 25(4):327–336, July 1991.

[14] Kamal Kant and Steven W. Zucker. Toward Efficient Trajectory Planning: The Path-Velocity Decomposition. *Int. Journal of Robotics Research*, 5(3):72–89, Fall 1986.

[15] Philip Lee, Susanna Wei, Jianmin Zhao, and Norman I. Badler. Strength Guided Motion. *Computer Graphics*, 24(4), 1990.

[16] T. Lozano-Perez and M. A. Wesley. An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles. *Communications of ACM*, 22(10):560–570, October 1979.

[17] Tomas Lozano-Perez. A Simple Motion Planning Algorithm for General Robot Manipulators. *IEEE Journal of Robotics and Automation*, RA-3(3):224–238, June 1987.

[18] Tomas Lozano-Perez. Spatial Planning: A Configuration Space Approach. *Transactions on Computers*, c-32(2):26–37, Feb 1983.

[19] Jessica K. Hodgins Marc H. Raibert. Animation of Dynamic Legged Locomotion. *Computer Graphics*, 25(4):349–358, July 1991.

[20] Cary B. Phillips and Norman I. Badler. Interactive Behaviors for Bipedal Articulated Figures. *Computer Graphics*, 25(4), 1991.

[21] Cary B. Phillips and Norman I. Badler. JACK: A Toolkit for Manipulating Articulated Figures. In *Proceedings of ACM SIGGRAPH Symposium on User Interface Software*, Banff, Alberta, Canada, 1988.

[22] Cary B. Phillips, Jianmin Zhao, and Norman I. Badler. Interactive Real-time Articulated Figure Manipulation Using Multiple Kinematic Constraints. *Computer Graphics*, 24(2), 1990.

[23] Hans Rijpkema and Michael Girard. Computer Animation of Knowledge-Based Human Grasping. *Computer Graphics*, 25(4):339–348, July 1991.

[24] J. T. Schwartz and M. Sharir. On the Piano Movers' Problem: I. The case of a Two Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers. *Communications on Pure and Applied Mathematics*, 36:345–398, 1983.

[25] J. T. Schwartz and M. Sharir. On the Piano Movers' Problem: II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds. *Advances in Applied Mathematics*, 4:298–351, 1983.

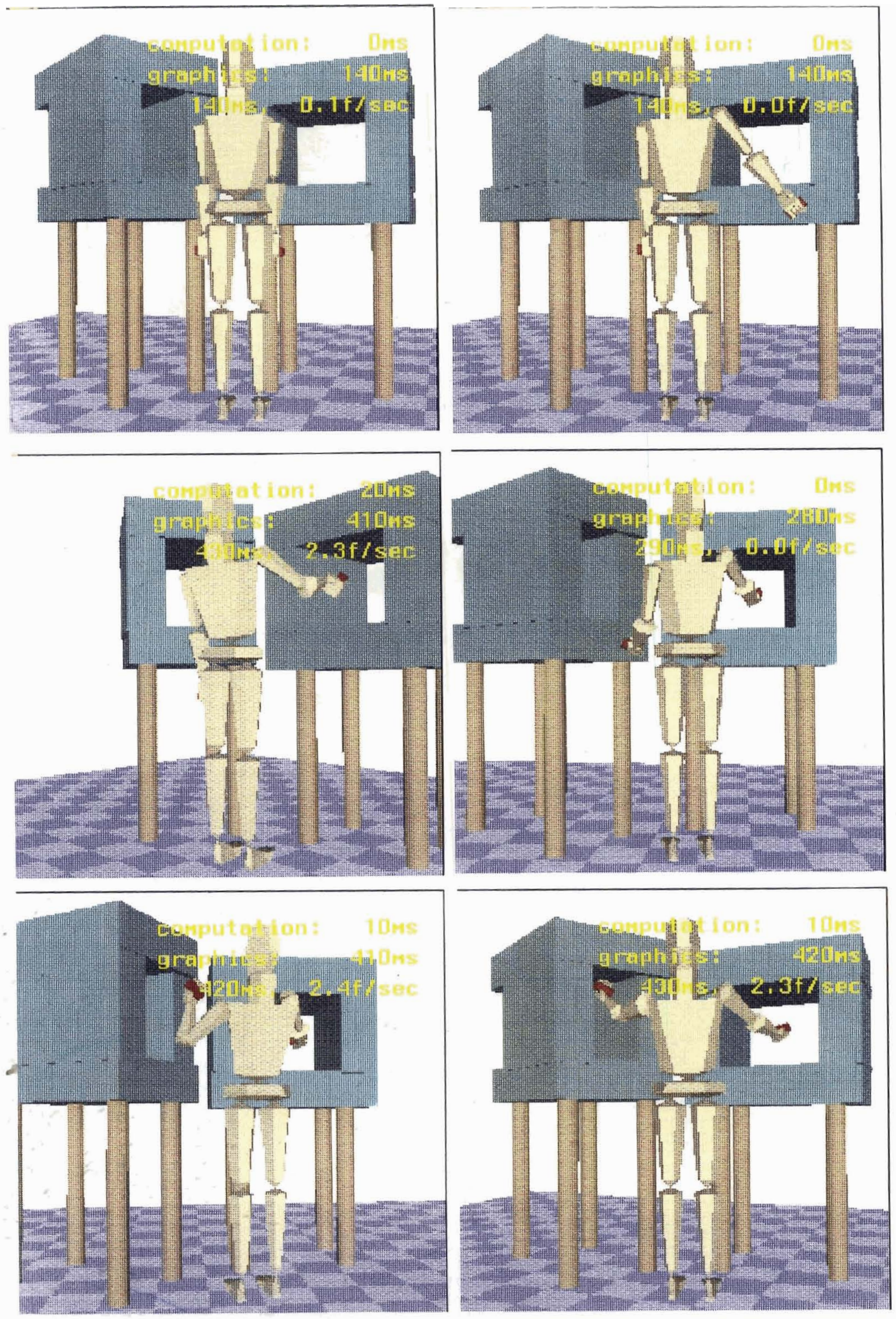


Figure 15: A human figure reaching through two apertures with both arms.

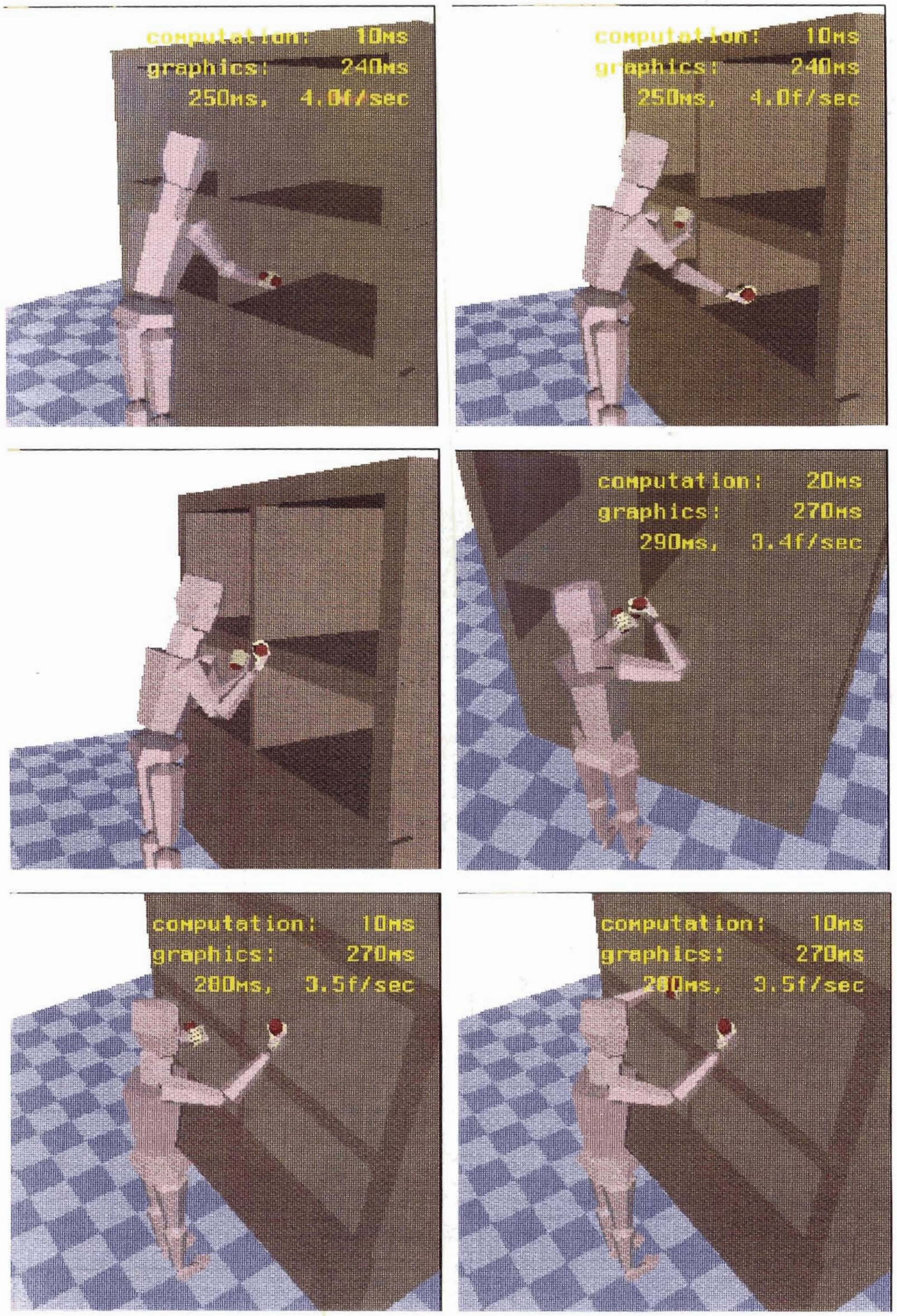
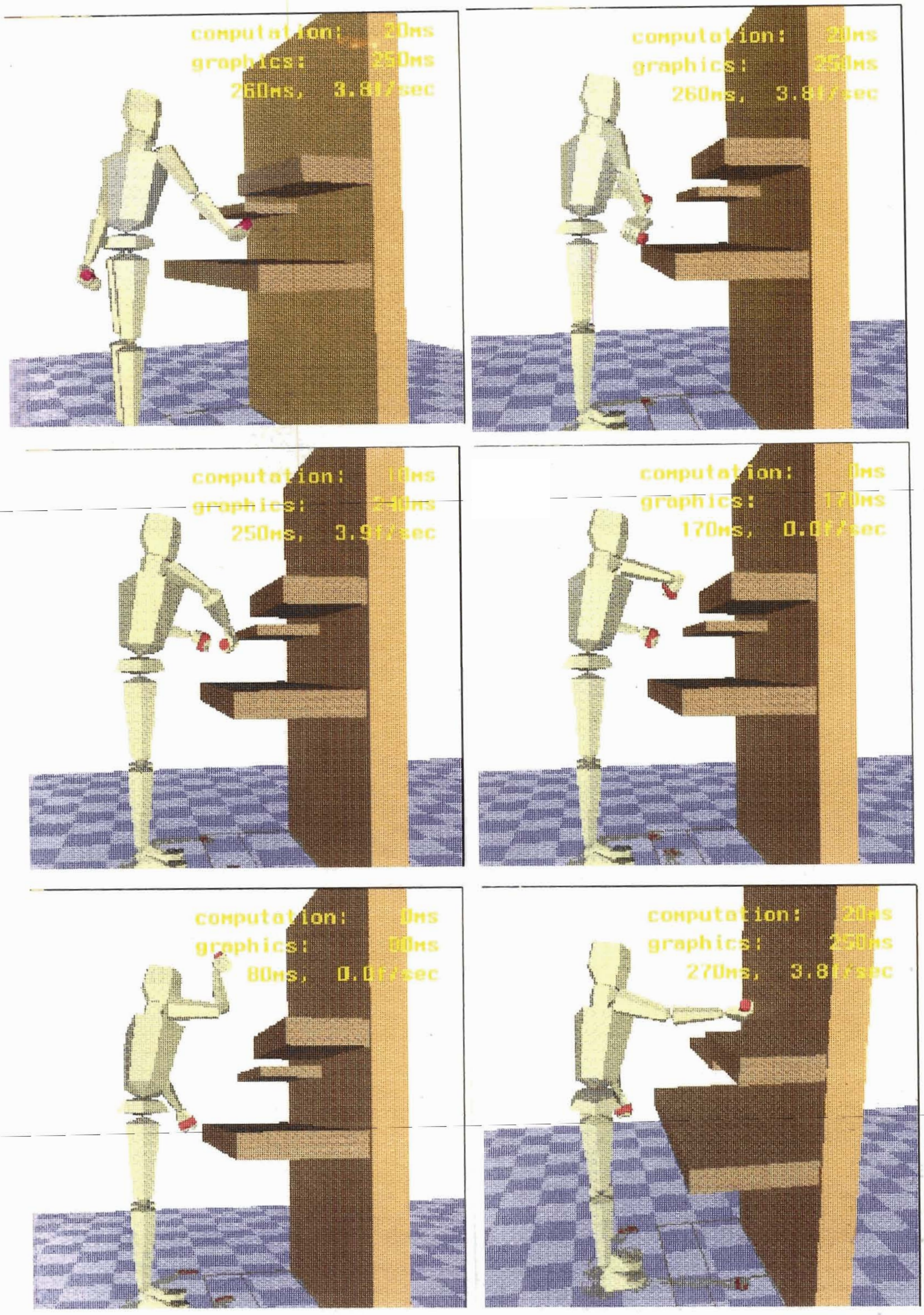


Figure 16: A human figure lifting objects from the lower shelves of a chest to the upper shelves.



```
computation:     20ms
graphics:       250ms
   260ms,  3.8f/sec
```

```
computation:     20ms
graphics:       250ms
   260ms,  3.8f/sec
```

```
computation:     10ms
graphics:       240ms
   250ms,  3.9f/sec
```

```
computation:      0ms
graphics:       170ms
   170ms,  0.0f/sec
```

```
computation:      0ms
graphics:        80ms
    80ms,  0.0f/sec
```

```
computation:     20ms
graphics:       250ms
   270ms,  3.8f/sec
```

Figure 17: A human figure lifting objects on a different shelf.

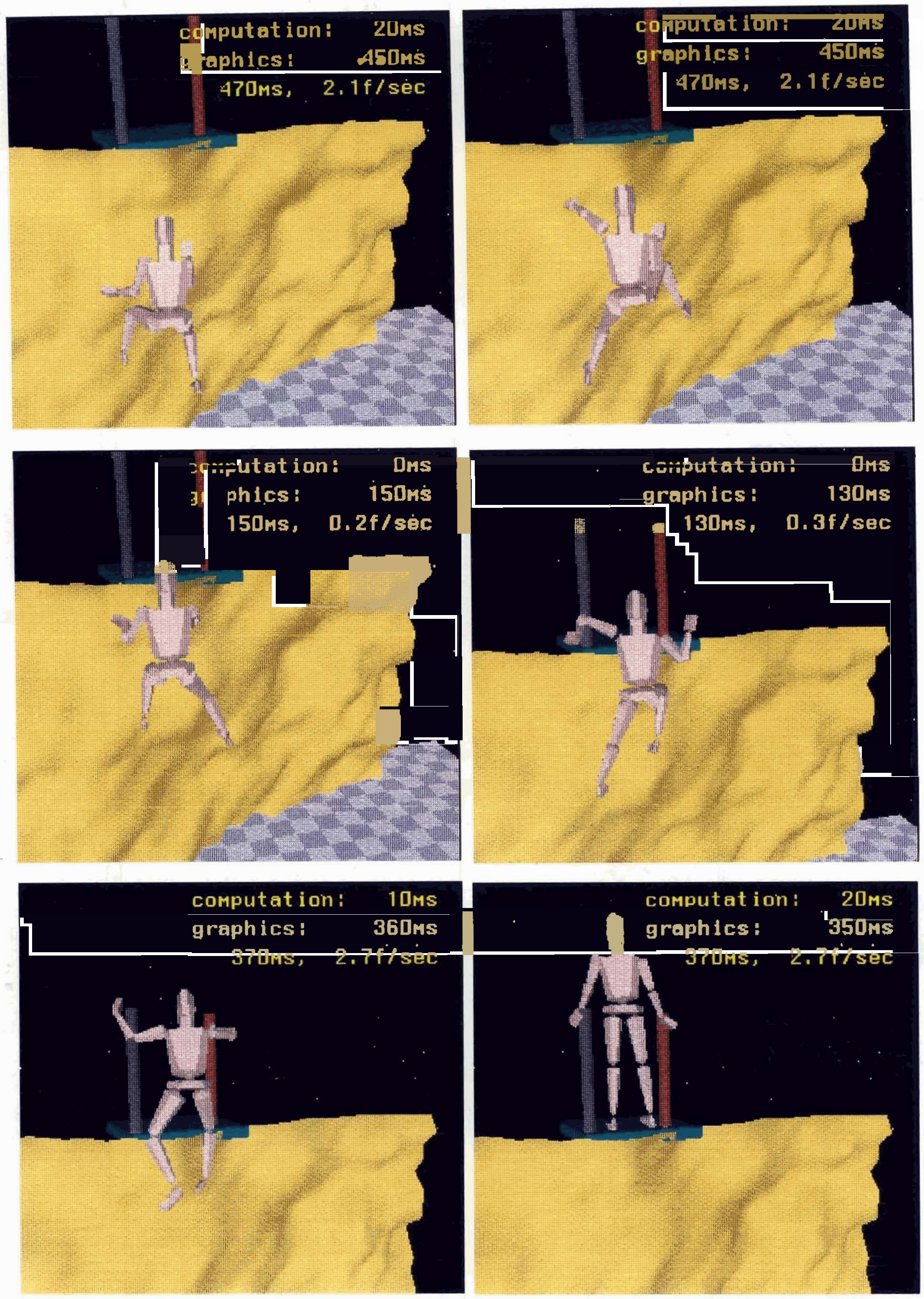


Figure 18: A human figure climbing up a rocky surface. This animation is created with a hybrid of simulation techniques.