Keyframe Animation of Articulated Figures Using Autocollision-Free Interpolation

Jean-Christophe Nebel
jc@dcs.gla.ac.uk

University of Glasgow
Computer Science Department
17 Lilybank Gardens
Glasgow, G12 8QQ, UK

Abstract

Keyframing is a popular method for animating articulated figures because it allows artistic expressiveness by providing control to the animator. The animator defines a movement by providing a set of poses. The motion is then obtained by simply interpolating between these keyframes, typically with cubic splines.

The drawback of this process is that it requires significant effort from the animator. Defining a movement often requires a high level of detail to ensure that the interpolation curves induce the desired motion. In particular, the animator has to focus on avoiding collisions between the limbs of his or her articulated figure.

The paper describes a new interpolation method producing autocollision-free paths. A first interpolation is computed using any classical inbetweening method. Autocollisions are then identified and corrected if necessary. At some collision times, autocollision-free sub-keyframes are automatically generated using geometric properties. They are then used for obtaining an autocollision-free interpolation.

Experimental results using a human model show that the animator can reduce the level of detail needed for describing a movement and still get realistic results at interactive speeds.

Keywords. Animation, articulated figures, keyframe

1 Introduction

Keyframing is a popular method for animating articulated figures, because it allows artistic expressiveness by providing control to the animator. The animator defines a movement by providing a set of poses. The motion is then obtained by simply interpolating between these keyframes, typically with cubic splines. These cubic splines may be controlled using kinematic [12], [2], [8] or dynamic constraints [9], [1] in order to have more realistic inbetweenings.

The drawback of this process is that it requires significant effort from the animator. Defining a movement often requires a high level of detail to ensure that the interpolation curves induce the desired motion. Hence recent work has been focused on high level functionalities such as adapting reference movements between keyframes [11], [7], [14]. Because the animator's creativity cannot be satisfied by only a set of gaits, specific movement still needs to be generated e.g. applications such as computer aided choreographic design [4].

In particular, the animator has to focus on avoiding collisions between the limbs of his or her articulated figure by adding irrelevant keyframes from the point of view of the creation, which appear to be necessary for realistic interpolation. The purpose of our research work is to offer a new interpolation method producing autocollision-free paths in order to free the animator from interpolation considerations and to let him or her to concentrate on what really matters.

In this paper we first describe the way we model an articulated figure, then the principle of our method and finally some experimental results.

2 The articulated figure

The actor is the key object in an animation system, so its design is very important. Our rigid articulated figure is modelled using the definition of D. Sturman [13]. Our model is a set of rigid objects jointed at nodes, organised hierarchically into an articulated figure. At each joint, a 3D-transformation matrix controls the position of the portion of the body below that joint. The motion of each joint is limited by degrees of freedom (DOFs) in order to produce only believable postures.

To evaluate our animation system, a humanoid was used. A very simplified skeleton, composed of 19 limbs and 15 joints, is used as a model. See figure 1.

In order to deal with autocollisions, we use the concept of a member:

- A member is a list of consecutive limbs starting from the centre of a figure (root) and finishing to a limb which does not have any children.
- Members can share common limbs.
- If A and B are limbs belonging to a same member and if A is closer to the root than B, then $A < B$.

In figure 1, the 5 members of our character are shown with different colours. The left upper member is composed of the trunk, the left shoulder, the left upper arm, the left forearm and the left hand.

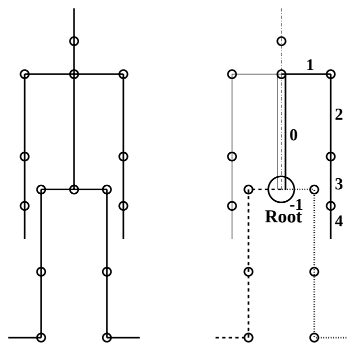


Figure 1: An articulated figure: joints and limbs (left) and members (right)

The software, we use for producing poses, has been developed by S. Etienne [6]. The production of poses of our articulated figure can be achieved in real-time by three different ways:

- Forward kinematics
- Inverse kinematics
- Interactive genetic algorithms [6]

The time spent for building a pose has been evaluated for these techniques by S. Etienne [6]. It appears a trained animator needs between 30 and 60 seconds to produce a copy of a given pose whatever the method.

3 Principle

Once the animator has specified the keyframes of the animation, interpolations are achieved to produce the animation using cubic splines [5]. The task to achieve is to generate collision-free motion of articulated figures in a moving environment.

This problem is a classical problem of robotics: basic motion planning. It has been shown that a complete solution to this problem is computationally very expensive [3] and [10]. Much progress has been made, however, in producing fast planners by considering schemes that are not complete, i.e., may fail to find a path when one exists.

Because we wish to provide results at interactive speeds to allow the modification of the animation if the suggested interpolation does not fit the animator's desire, we propose a quick incomplete algorithm. It should generate automatically, at least, some of the keyframes that the animator previously had to add manually with a speed which would globally reduce the time the animator had to spend producing an animation, see section 2.

The principle of our scheme is, at first, to compute a first interpolation using any classical inbetweening method. Then, autocollisions are detected and corrected if necessary. At some collision times, autocollision-free sub-keyframes are automatically generated using geometric properties. Finally, these keyframes are used for a new classical interpolation. This process continues until an autocollision-free interpolation is obtained.

The following is an outline of the algorithm providing an autocollision-free path between two keyframes ((K_0, T_0) and (K_1, T_1)):

```
CheckAndCorrect( $(K_0, T_0), (K_1, T_1)$ )
{
    Interpolate between $K_0$ and $K_1$
    Get for each time step between $T_0$ and $T_1$ the list of autocollisions
    If(autocollision)
    {
        Select the first collision to be corrected $(T_{Collision}, ColType)$
        Move the limbs at $T_{Collision}$ in order to correct the collision
        Create a subkeyframe $(K_{Collision}, T_{Collision})$
        CheckAndCorrect( $(K_0, T_0), (K_{Collision}, T_{Collision})$ )
        CheckAndCorrect( $(K_{Collision}, T_{Collision}), (K_1, T_1)$ )
        }
}
```

In the next section, we explain how we correct collisions and produce subkeyframes.

4 Implementation

In order to apply the principle previously described, we have to define the way of ordering the tasks of intersection correction, then by which process an intersection is corrected for a given time. But first, we shall define the kinds of intersection which can be observed.

4.1 Types of intersections

In this paper, we define an intersection (A,B) as the intersection of 2 limbs A and B. We define 3 kinds of intersections:

- Intersection between two limbs belonging to a same member, figure 2(a,b): mono-intersection
- Intersection between two limbs not belonging to a same member:
  - The two members share some common limbs, figure 2(c): bi-shared-intersection
  - The two members do not share any common limbs, figure 2(d): bi-unshared-intersection

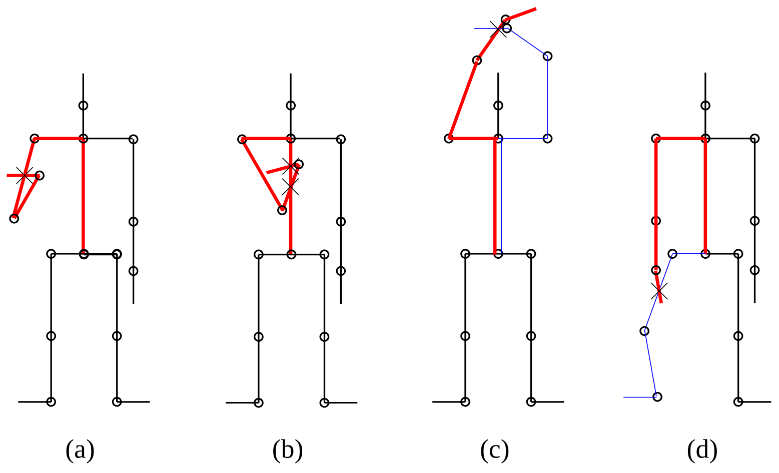


Figure 2: Types of intersections

4.2 Production of a subkeyframe

Once the first interpolation round has been performed between two keyframes, intersections between limbs are detected. Then they are grouped: to each couple of intersecting limbs, the type of the intersection and a list of intersection periods are computed. Then the first intersection to be corrected has to be chosen. We scan the intersection periods and select the longest one. The frame, with time index at the centre of the period, is the first frame to be corrected.

If the intersection to correct is a mono-intersection, there is only one member which has to move, but if it is a bi-intersection two members may eventually move. In this paper, we correct an intersection only my moving a single member. So in the case of a bi-intersection, the member, which has to move, is the member which has the most moving limbs between the previous and the next frame, since it has a large movement allowing more freedom for moving a member.

Then, we have to define which limb should be moved first for the correction, and which part of the member should stay immobile:

- For a mono-intersection (A,B), if we suppose that A is closer to the root of the member than B, $A < B$:
  - In the best possible case, correcting the intersection would be achieved by only moving B
  - Moving A or any other limb closer to the root would not have any effect on the intersection

  B is the starting limb and A is the ending limb
- For a bi-intersection, if we suppose that A is the limb which belongs to the selected member:
  - For a bi-unshared-intersection, A is the starting limb and the root is the ending limb
  - For a bi-shared-intersection, A is the starting limb and the first common limb is the ending limb

Then, we look for any related intersections at the same time in order to correct them during the same step.

If there are other intersections on the selected member, starting limbs and ending limbs are calculated for each of these intersections (for bi-intersections these computations are made for the selected member). Then the new starting limb is the starting limb which is the closest to the root and the new ending limb is the ending limb which is the closest to the root:

$$Intersections = \bigcup_i I_i(Starting_i, Ending_i)$$
$$NewStartingLimb = Min_i(Starting_i)$$
$$NewEndingLimb = Min_i(Ending_i)$$

Once the member is selected and the starting and ending limbs are known, the computation of a new pose of the member can start. The process begins with the motion of the starting limb. New positions for the end of the limb are computed (see subsection 4.3). For each of these new values, the limb is moved and intersections between this limb and the others are checked. As soon as a position is valid (no intersection), the process continues with the search of a valid position for the next child limb of the member. When the last limb of the member has been validated the member is collision-free and the associated subkeyframe is produced.

However, if at any step, no valid position may be found, new positions are computed for the parent limb, which becomes the starting limb.

If the starting limb becomes the ending limb, no move can be performed because this limb has to be immobile. So the process has finished and has failed. No subkeyframe is produced.

Once the subkeyframe has been produced autocollisions are checked again for the whole figure. If there are remaining intersections, they are sorted and the intersection to be corrected is chosen. And the subkeyframe is modified accordingly to the process previously described.

The following is an outline of the algorithm:

```
Move(end, start)
{
    limb = member(start)
    limb→GetPositions()
    OrderPositions()
    For each position
    {
        limb→MoveToPosition()
        If(limb→DoNotIntersect())
        {
            for(i =start+1 to nblimbs)
            {
                limb = limbs(i)
                If(limb→DoIntersect())
                    If(Move(start, j)==0)
                    Break
            }
            if(i==nblimbs)
                return SUCCESSFUL
        }
    }
    start−−
    limb→MoveBackToStartingPosition()
    if(start==end)
        return FAILED
    else
        return Move(end,start)
}
```

For example, in case (a) of figure 2, the intersection to be corrected is the mono-intersection (Right hand, Right upper arm). The way of resolving it is, at first, to move the right hand. If a valid position is found, the process is over and the subkeyframe is generated.

But if no valid position can be found, the forearm should be moved to another valid position before the hand is moved again. If no valid position can be found for the forearm, the process has to stop and is unsuccessful because the upper arm has to stay immobile - its movement would not affect this intersection. If a valid position of the hand is found for a valid position of the forearm, a subkeyframe is generated, otherwise this intersection cannot be corrected.

In case (b) of figure 2, we suppose the first selected intersection to be corrected is the mono-intersection (Right hand, Trunk). Its correction may not remove the (Right forearm, Trunk) intersection. These two intersections involve the same member; they are related and should be corrected together. The starting limb is then the right forearm and the ending limb is the right upper arm.

4.3 Production of new positions

In order to produce collision-free subkeyframes, limbs are moved to new positions. The realism of the interpolation produced depends on the generation of these positions. At first we assume that the interpolation algorithm used has been carefully chosen by the animator to provide the maximum of realism in motion needed for his or her purpose. So any new generated position of a limb should be related to its previously interpolated curve. Secondly, the intersection detected is due to an obstacle, which should be passed, the limb cannot stay on one of its sides.

We define, here, a limb with two positions:

- The beginning of the limb, B, which does not move when the limb moves
- The end of the limb, E, which should be moved to correct the detected collision.

In order to keep the new suggested positions of the limb connected with an interpolation curve, we use the previous position, (B_0, E_0), and the next position, (B_1, E_1), of the limb on the interpolation curve, see figure 3.

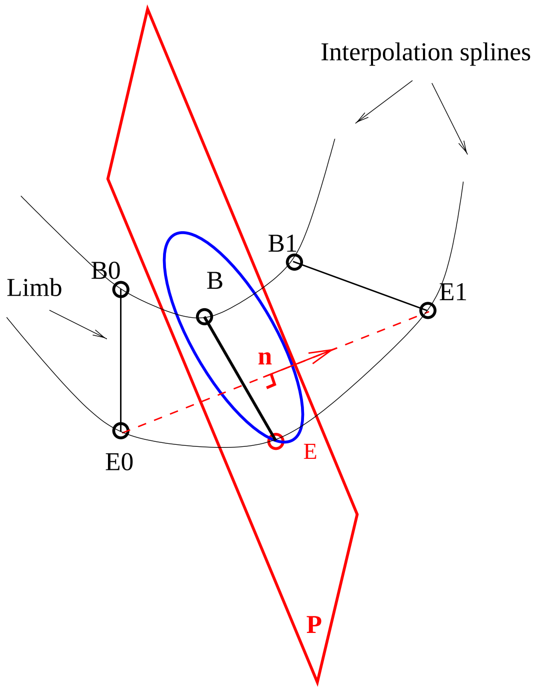


Figure 3: New positions generated by the plane P (an ellipse, a circle or a point)

Because we want the obstacle to be passed, we give the constraint that the new position should be in the space between the 2 planes ($P_0(\overrightarrow{E_0E_1}, E_0)$ and $P_1(\overrightarrow{E_0E_1}, E_1)$) perpendicular to $\overrightarrow{E_0E_1}$ and passing respectively through E_0 and E_1.

Practically, we build a first plane perpendicular to $\overrightarrow{E_0E_1}$ and passing through E, $P(\overrightarrow{E_0E_1}, E)$ and then build other planes, P_i at regular intervals between E and E_1, and E_0 and E.

The figure has a rigid skeleton and hence the distance between the beginning of the limb (B) and the end (E) is constant, moreover B is fixed, so the positions should belong to the sphere $S(B, \overrightarrow{BE})$.

The space of new positions is the intersection between these planes and this sphere:

$Positions = \bigcup_i P_i \cap S$

A sampling of this space is made and positions which do not suit the DOFs are removed. Then they are sorted by distance to the initial position in order to offer first, positions which are close to the interpolated position.

5 First results

We tested our algorithm using an articulated figure whose skeleton is described figure 1. The keyframes of the different animations are given in table 1 and the animations themselves are presented in appendix A. Of course these animations do not pretend to be a representative sample of all possible animations, but could give an idea of the first results obtained thanks to our algorithm. In these examples, the space of new positions is sampled to 180 positions (using 9 planes, see section 4.3), this number being a compromise between the level of precision and the speed. Interpolations are made using classical cubic splines.

In table 2, some experimental results are given with the following variables:

- Motion: number of the studied animation (see table 1). Each animation is produced using only two keyframes.
- Frame: Number of frames composing the animation
- Inter/frame: Number of intersections occurring during the classical interpolation divided by the number of frames
- SubK: Number of subkeyframes generated by the algorithm
- Free: Is the resulting animation collision-free? (yes or no)
- Quality: Quality of the realism of the generated motion (good, fair or poor)
  - Good: the animation generated can be used directly
  - Fair: the animation generated is possible, but not very realistic or not smooth enough, the animator could keep most of the frames and modify one of them in order to get the desired animation
  - Poor: the animation generated is not possible, new keyframes should be created
- Times: All times given are expressed in seconds
  - T_{algo}: Computation time of our algorithm. This work being in progress, the times given should only be seen as indicative. They have been obtained with a 300 MHz Ultra2 processor of a Sparc 450 workstation. No optimisation has been implemented yet.

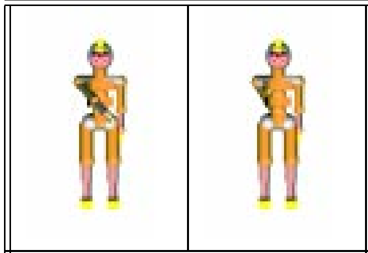
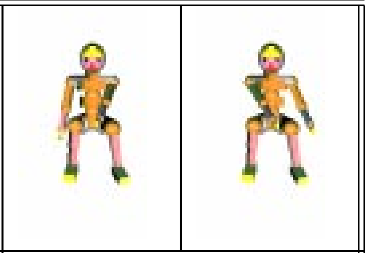
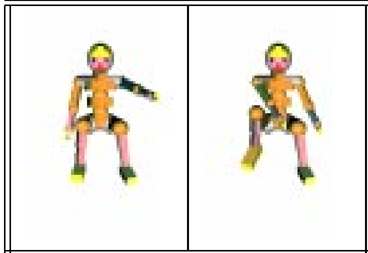
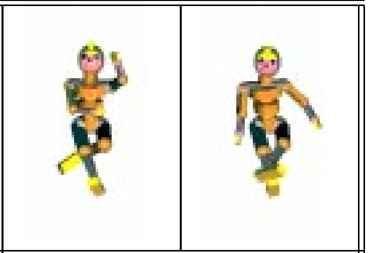
	
Animation 1	*Animation 2*
	
Animation 3	*Animation 4*

Table 1: Keyframes of the studied animations

- T_{old}: Minimum theoretical time needed by the animator for creating the missing keyframes in order to have a similar collision-free animation. It is approximated by multiplying the number of subkeyframes generated by the minimum time needed for the creation of a pose (30 seconds [6]).
- T_{new}: Theoretical time needed in order to have a realistic collision-free animation using our algorithm. Its approximation depends on the quality of the animation produced by our algorithm:
    - Good quality: T_{new} is the computation time of our algorithm.
    - Fair quality: T_{new} is the computation time of our algorithm plus the time needed for the modification of one pose by the animator, which are approximated by the minimum time needed for the creation of a pose (30 seconds [6]).
    - Poor quality: T_{new} is the computation time of our algorithm plus the minimum theoretical time.

The meaning of the variables having been given, we can start analysing the results of the table 2.

At first, the computation times are between 1 and 39 seconds, most of them are below 20 seconds: the interpolations are produced at interactive speed. The quality of a result and its computation time depend on the keyframes and the number of frames to interpolate. However our algorithm seems to have a positive impact in most cases:

- Good quality: Our algorithm is very time saving for the animator, the speed-up is between 3.1 and 20, and allows him or her to focus on the creative process.

- Fair quality: Our algorithm is time saving for the animator, the speed-up is between 1.3 and 2.8, and eases the task of creating new keyframes by providing frames where only minor corrections are needed.

Motion	Frame	Inter/frame	SubK	Free	Quality	T_{algo}	T_{new}	T_{old}
1	5	1.2	1	yes	good	2	*2*	*30*
1	9	1.7	3	yes	good	13	*13*	*90*
1	14	1.7	2	yes	fair	15	*45*	*60*
1	20	1.5	2	yes	fair	17	*47*	*60*
2	5	1.4	2	yes	good	3	*3*	*60*
2	9	1.8	4	yes	fair	13	*43*	*120*
2	14	1.7	3	yes	good	29	*29*	*90*
2	20	1.8	3	yes	fair	22	*52*	*90*
3	5	0.6	2	yes	fair	16	*46*	*60*
3	9	1.0	3	yes	good	10	*10*	*90*
3	14	0.9	3	yes	good	12	*12*	*90*
3	20	1.0	5	yes	poor	39	*189*	*150*
4	5	0.8	3	yes	fair	4	*34*	*90*
4	9	1.0	0	no	poor	1	-	-
4	14	0.9	0	no	poor	1	-	-
4	20	0.9	0	no	poor	1	-	-

Table 2: Experimental results

- Animation with auto-collisions: If the algorithm is not able to build a collision-free motion, this result is given, generally, instantly and information about the auto-collisions detected are provided. Knowing this, the animator can then add some new keyframes at specific times.
- Poor quality: Our algorithm has been time consuming and does not provide any valuable information except for the list of auto-collisions originally detected. These poor results can be explained by mainly two reasons:
  - The process being based on time steps instead of continuous motion, if the sampling of a motion is not thin enough, the path between two successive valid positions can cross an obstacle.
  - If there are several ways of passing an obstacle at a given time step the chosen one may be a dead-end for the next frames.

These first results show that our interpolation method, producing autocollision-free motions, can be an interesting tool for keyframe animation. This tool should be used interactively by an animator, who should decide if the offered interpolation should be kept, modified or if more keyframes should be provided. In spite of some time consuming cases, this algorithm spares a lot of time for the animator, who does not have to produce as many keyframes as classical interpolations ask, and lets him or her work on what really matters: the creation process.

6 Conclusion and future work

We presented an algorithm of interpolation assisting the animator in his or her production of autocollision-free animation. This algorithm, having to provide results at interactive speed, is not complete, but most of the time offers an improved animation.

Experimental results using a human model show that with our algorithm, the animator can generally reduce the level of detail needed for describing a movement and still get realistic motions at interactive speed. They can be used directly or with minor corrections by the animator. However the animator sometime still has to give an higher level of detail to get the desired motions. Globally the animator saves a lot of time and can focus more on the creation process.

In the future, we will work on the improvement of this algorithm, and in particular on the way of selecting the best position among the positions produced. We will also implement some optimisations in order to provide results at near real-time.

Acknowledgements

We gratefully acknowledge the support of the European Union's Training and Mobility of Researchers (TMR) programme in funding this work.

References

[1] K. Arai. Keyframe animation of articulated figures using partial dynamics. In N. Magnenat-Thalmann and D. Thalmann, editors, *Models and techniques in computer animation*, pages 243–256. Springer-Verlag, 1993.

[2] R.H. Bartels and I. Hardtke. Speed adjustement for key-frame interpolation. In *Graphics Interface '89*, pages 14–19, 1989.

[3] J.F. Canny. *The complexity of robot motion planning*. MIT Press, 1988.

[4] J. Lopez D. Rodriguez D. Meziat M. Carbajo A. Casillas and J. L. Bosque. A user interface for the design of human figures multimedia animations. In *HIM97*, 1997.

[5] D. Eberly. Key frame interpolation via splines and quaternions, 1998.

[6] S. Etienne. *Positioning articulated figures*. PhD thesis, University of Glasgow, 1998.

[7] S. Guo and J. Roberge. A high-level control mechanism for human locomotion based on parametric frame space interpolation. In *Computer animation and simulation'96*, pages 95–107. Springer Computer Science, 1996.

[8] Z. Liu and M. F. Cohen. Keyframe motion imization by relaxing speed and timing. In *1995 Eurographics Workshop on Animation*, Maastrich, Holland, 1995.

[9] X. Pintado and E. Fiume. Grafield: Field-directed dynamic splines for interactive motion control. In *Eurographics'88*, pages 43–54, 1988.

[10] J.H. Reif. Complexity of the mover's problem and generalizations. In *26th IEEE Symposium on foundations of computer science*, pages 144–154, 1979.

[11] J. Roberge S. Guo and T. Grace. Controlling movement using parametric frame space interpolation. In N. Magnenat-Thalmann and D. Thalmann, editors, *Models and techniques in computer animation*, pages 216–227. Springer-Verlag, 1993.

[12] S. N. Steketee and N. I. Badler. Parametric keyframe interpolation incorporating kinetic adjustement and phrasing control. In *Siggraph'85*, volume 19, pages 255–262, San Francisco, 1985.

[13] D. Sturman. Interactive keyframe animation of 3d articulated models. In S. MacKay, editor, *Graphics Interface'84*, pages 35–40, 1984.

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

A Animations

We presents here, for each of the 4 animations studied, the frames composing the 5 frame animations. For each animation, we start by presenting the frames obtained by a classical spline interpolation and then the frames issued from our scheme.

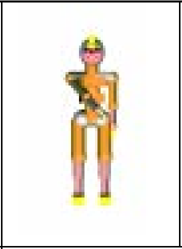
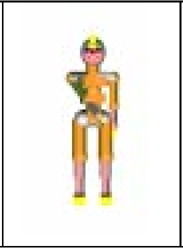
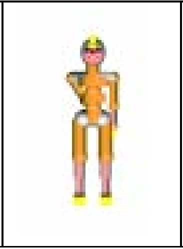
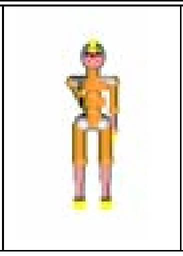
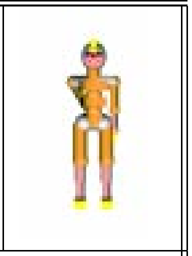
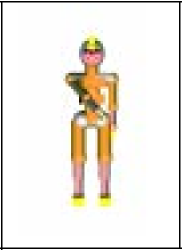
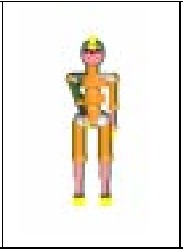
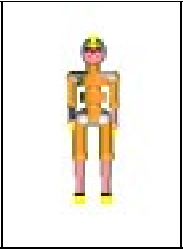
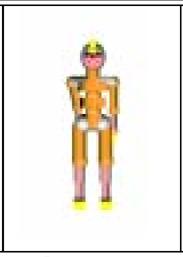
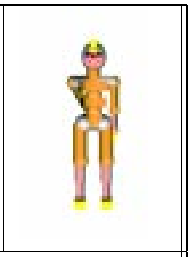
				
Spline interpolation				
				
Autocollision-free interpolation				

Table 3: Animation 1

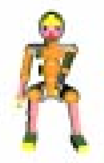
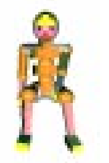
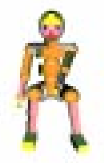
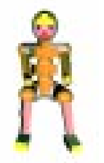
				
Spline interpolation				
				
Autocollision-free interpolation				

Table 4: Animation 2

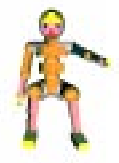
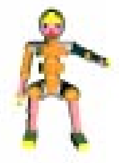
				
Spline interpolation				
				
Autocollision-free interpolation				

Table 5: Animation 3

Spline interpolation				
Autocollision-free interpolation				

Table 6: Animation 4