EUROGRAPHICS 2000 / M. Gross and F.R.A. Hopgood
(Guest Editors)

Volume 19 (2000), Number 3

Using an Intermediate Skeleton and Inverse Kinematics for Motion Retargeting

Jean-Sébastien Monzani, Paolo Baerlocher, Ronan Boulic and Daniel Thalmann[†]

Computer Graphics Laboratory, Swiss Federal Institute of Technology, Lausanne, Switzerland

Abstract

In this paper, we present a new method for solving the Motion Retargeting Problem, by using an intermediate skeleton. This allows us to convert movements between hierarchically and geometrically different characters. An Inverse Kinematics engine is then used to enforce Cartesian constraints while staying as close as possible to the captured motion.

Keywords: *Motion retargeting problem, adaptation to different characters.*

1. Introduction

1.1. Motivation

Recently, the growing demand of industry for realistic motions raised the problem of modifying a motion, once it has been captured. Even if it is fairly easy to correct one posture by modifying its angular parameters (with an Inverse Kinematics engine, for instance), it becomes a difficult task to perform this over the whole motion sequence while ensuring that some spatial constraints are respected over a certain time range, and that no discontinuities arise. When one tries to adapt a captured motion to a different character, the constraints are usually violated, leading to problems such as the feet going into the ground or a hand unable to reach an object that the character should grab. The problem of *adaptation* and *adjustment* is usually referred to as the *Motion Retargeting Problem*.

Some of the previous approaches addressed this with only geometrically different characters, that is, characters with different proportions. We present here a much flexible solution to both handle *geometrically* and *topologically* different characters, by using an *intermediate skeleton*. Adjustment is then performed through Inverse Kinematics.

This paper is organised as follows: section 1.2 presents the previous approaches, and is followed by an overview of our method in section 1.3. Sections 2 and 3 respectively focus on the conversions between different hierarchies and on how we are using Inverse Kinematics to enforce Cartesian constraints over a time-range. Section 4 shows implementation and results which are then discussed and compared to other approaches in section 5, before concluding.

1.2. Previous approaches

Andrew Witkin and Zoran Popović are often considered as pioneers in this area, as they presented in their SIGGRAPH 95 article[8] a technique for editing motions, by modifying the motion curves through warping functions and produced some of the first interesting results. In a more recent paper[7], they have extended their method to handle physical elements, such as mass and gravity, and also described how to use characters with different numbers of degrees of freedom. Their algorithm is based on the reduction of the character to an *abstract character* which is much simpler and only contains the degrees of freedom that are useful for a particular animation. The edition and modification are then computed on this simplified character and mapped again onto the end user skeleton. Armin Bruderlin and Lance Williams[4] have described some basic facilities to change the animation, by modifying the motion parameter curves. The user can define a particular posture at time t, and the system is then responsible for smoothly blending the motion around t. They also introduced the notion of *motion displacement map*, which is an offset added to each motion curve. The *Motion Retargeting Problem* term was brought up by Michael

[†] `http://ligwww.epfl.ch/ {jmonzani, baerloch, boulic, thalmann}@lig.di.epfl.ch`

Gleicher in his SIGGRAPH 98 article[5]. He designed a space-time constraints solver, into which every constraint is added, leading to a big optimisation problem. He mainly focused on optimising his solver, to avoid enormous computation time, and achieved very good results. Rama Bindiganavale and Norman I. Badler[2] also addressed the motion retargeting problem, introducing new elements: using the zero-crossing of the second derivative to detect significant changes in the motion, visual attention tracking (and the way to handle the gaze direction) and applying Inverse Kinematics to enforce constraints, by defining six sub-chains (the two arms and legs, the spine and the neck). Finally, Jehee Lee and Sung Yong Shin[6] used in their system a coarse-to-fine hierarchy of B-splines to interpolate the solutions computed by their Inverse Kinematics solver. They also reduced the complexity of the IK problem by analytically handling the degrees of freedom for the four human limbs.

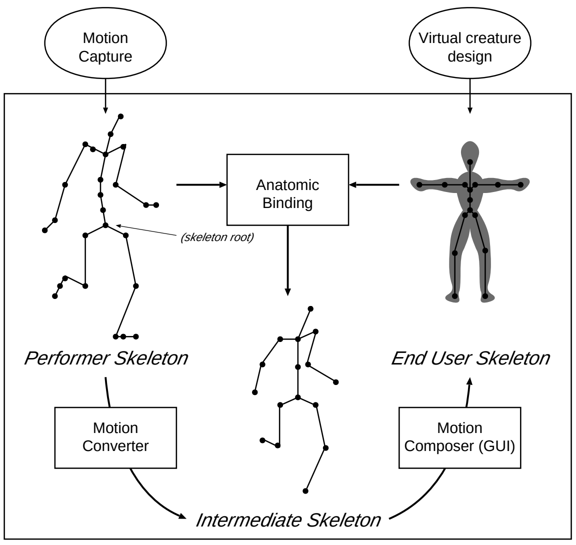


Figure 1: *Motion Conversion: overview*

1.3. Overview of the method

Given a captured motion associated to its Performer Skeleton, we decompose the problem of retargeting the motion to the **End User Skeleton** into two steps, as presented in sections 2 and 3: the first task is to convert the motion from one hierarchy to a completely different one. We introduce the **Intermediate Skeleton** model to solve this, implying three more subtasks: manually set at the beginning the correspondences between the two hierarchies, create the Intermediate Skeleton and convert the movement. We are then able to correct the resulting motion and make it enforce Cartesian constraints by using Inverse Kinematics, as described in section 3.

2. Direct motion conversion

2.1. The Intermediate Skeleton model

2.1.1. Overview

When considering motion conversion between different skeletons, one quickly notices that it is very difficult to directly map the Performer Skeleton values onto the End User Skeleton, due to their different proportions, hierarchies and axis systems. This raised the idea of having an **Intermediate Skeleton**: depending on the Performer Skeleton posture, we reorient its bones to match the same directions. We have then an easy mapping of the Intermediate Skeleton values onto the End User Skeleton, as presented on figure 1. The first step is to compute the Intermediate Skeleton (*Anatomic Binding* module). During the animation, motion conversion takes two passes, through the *Motion Converter* and the *Motion Composer* (which has a graphical user interface). The next sections discuss the creation of the Intermediate Skeleton, the motion conversion and demonstrate the importance of the initial Intermediate Skeleton posture.

2.1.2. Intermediate Skeleton creation

Our Intermediate Skeleton has the same number of nodes and the same local axis systems orientations as the End User Skeleton, but the bones are oriented as the Performer Skeleton nodes (the length of these bones are not significant, only their direction matters). Figure 5 shows a sample torso and right arm for the three models: note that the Performer Skeleton and the End User Skeleton do not necessarily have the same topology (number of nodes, represented as grey dots), nor the same coordinates systems. That is why the user has to indicate a one-to-one correspondence between the nodes of the two hierarchies before starting, as presented in figure 3. Some nodes of the Performer Skeleton can be ignored, like the scapula, the clavicle and the wrist in our example. This correspondence introduces what we call *virtual bones.* A virtual bone indicates a direction between two Performer Skeleton nodes, which are corresponding to two End User Skeleton nodes. When converting the motion, we align the Performer Skeleton virtual bones with the End User Skeleton bones. The right arm that we have defined previously has three virtual bones, as highlighted in figure 4.

2.1.3. Motion Converter and Motion Composer

For a particular Performer Skeleton's posture at time t, we first reorient each Intermediate Skeleton bone to point to the same direction as the corresponding Performer Skeleton virtual bone, starting from the skeleton root (located on the bottom of the spine) and propagating to the leaves. It is then easy to copy the local values of the Intermediate Skeleton nodes to the End User Skeleton nodes, and this does make sense, since both axis systems are the same.

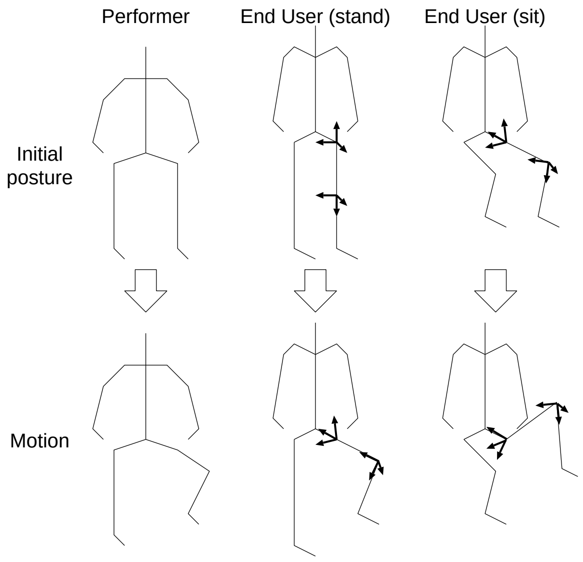


Figure 2: *Motion conversion with two differents initial postures for the Intermediate Skeleton.*

2.1.4. Virtual bones initial orientation

We discovered that it might be suitable for some applications to have different initial postures for the Performer Skeleton and the End User Skeleton, for instance the standing postures for an athlete and a sumo are not the same. These initial postures often give an impression about the character, and are important to give the illusion of variety. On the other hand, for extremely different initial postures (for example somebody standing for the Performer Skeleton and someone sitting for the End User Skeleton) problems may certainly arise if we directly copy values between them. Figure 2 shows the importance of the Intermediate Skeleton/End User Skeleton initial posture: if the Performer Skeleton moves its legs, the mapping will produce strange results when using a sit initial posture.

On user's request, we propose that during the Intermediate Skeleton creation process, the *Anatomic Binding* module reorients the bones of the End User Skeleton/Intermediate Skeleton to match the same direction as the Performer Skeleton virtual bones. The resulting posture, which is the same for the three skeletons is what we call the *rest posture*. The user still has the possibility to keep her/his own initial posture if needed.

The reorientation is simple: for example (figure 6), if the orientation of the End User Skeleton's right forearm is not the same as the Performer Skeleton, we rotate it, and propagate the new orientation to the children nodes and axis systems. Let *u* be the direction from an End User Skeleton node to its child, and *v*, the corresponding virtual bone for the Performer Skeleton. The amount of rotation is given by the angle between these two vectors, and the rotation takes place around a vector, normal to *u* and *v*. Once this is done, we only have to copy the Intermediate Skeleton matrices local values to the End User Skeleton to perform the conversion. The automatic reorientation we present here does not take into account the twisting of the bones, but it can be corrected if the user provides more information while setting the nodes correspondences.

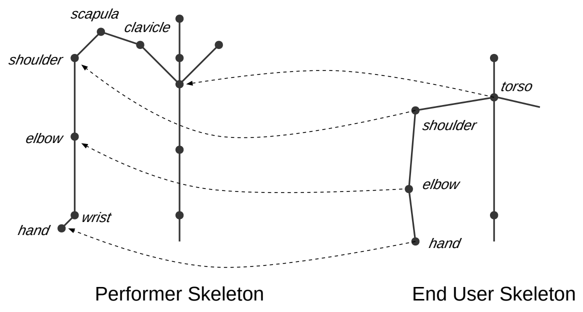


Figure 3: *Node to node correspondences between the Performer Skeleton and the End User Skeleton*

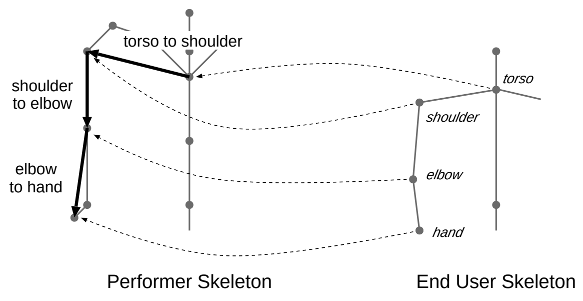


Figure 4: *Virtual bones for the right arm*

2.1.5. Summary

In summary, motion conversion at time *t* is done in two steps. First, computing the Intermediate Skeleton matrices by orienting the Intermediate Skeleton bones to reflect the Performer Skeleton posture (*Motion Converter*). Second, setting the End User Skeleton matrices to the local values of the corresponding Intermediate Skeleton matrices with the *Motion Composer* module. We have added a graphical user interface to this last module to be able to correct the problems caused by the conversion (either manually or automatically, as presented in section 3). We believe it is very important in computer animation to give the designers the ability of manual adjustment.

2.2. Motion Adjustment

The easiest way to correct the motion is to use a motion displacement map, as described by Bruderlin and Williams[4].

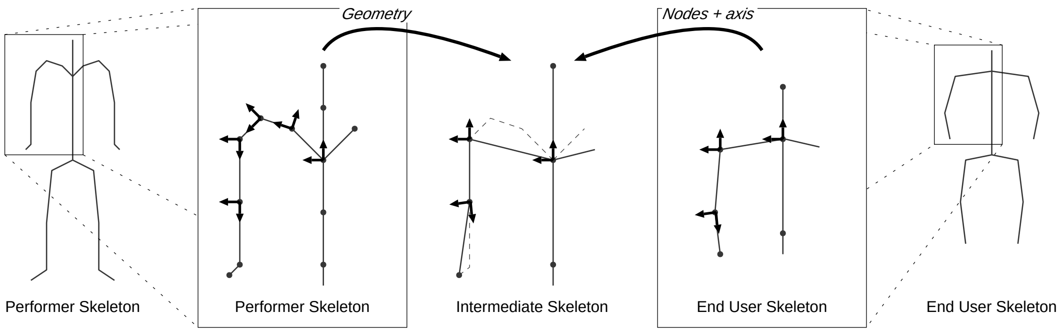


Figure 5: *Correspondences between Performer Skeleton, Intermediate Skeleton and End User Skeleton*

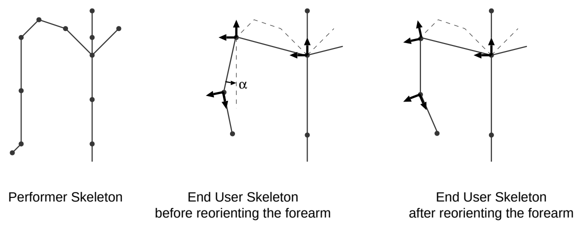


Figure 6: *Changing the End User Skeleton to the rest posture*

A displacement map works as an offset, added to each motion curve. For example, if the *z* translation coordinate of the skeleton root represents the position of the skeleton above the floor, and one would like to keep this coordinate above zero, one can define the corresponding motion displacement curve for *z* and add it to the previous one. In our implementation (a plug-in for 3D Studio Max), we added displacement maps and stored two values into each node: the original motion curves, and an optional offset, that the user can manipulate to edit the animation. When enforcing Cartesian constraints, these offsets are automatically computed as presented in the next section.

3. Using Inverse Kinematics to enforce constraints

3.1. Introduction

There is a growing demand, especially in the video-games and movies fields to retarget motions while ensuring that spatial constraints are enforced. This includes, for instance, people resting on a balustrade, climbing a ladder, picking an object, etc... All of these constraints can be expressed as desirable locations for body elements, such as hands, elbows, or feet, and we refer to these constraints as *Cartesian constraints*. Numerous works proved that Inverse Kinematics (IK) is a powerful tool to synthesise motion by defining special postures (keys) and interpolate between them to produce the animation. Key-frame animation is not really suitable for our problem, since our goal is to correct, and not to synthesise motion. The naive approach of solving one IK problem for each frame with a regular IK engine gives awkward results, as the original End User Skeleton posture may be lost while computing an IK solution over the whole hierarchy. Furthermore, discontinuities will arise because IK can produce a posture at time t and a very different one (but also correct) at time $t+1$, due to different IK solutions for the same problem.

However, we demonstrate in the next sections how to address these problems, and how to ensure a smooth animation by using an IK solver.

3.2. Using IK

The user defines some Cartesian constraints on the End User Skeleton, which are activated over a specified time range. Parameters specified by the user include which particular node of the hierarchy is constrained, which object is the *goal* of the constraint, and what kind of constraint it is. For instance, it is easy to define that the hand of the End User Skeleton has to stay above a table object, by declaring that the constrained node is the right hand, the goal is the table plane, and the constraint is of type “above a plane”.

An other requirement is to try to maintain the original captured posture (we will call this posture the *attraction posture*), while satisfying the Cartesian constraints. This is possible as long as the system is redundant. The general solution provided by Inverse Kinematics is[3]:

$$\Delta\theta = \mathbf{J}^{+}\Delta\mathbf{x} + (\mathbf{I} - \mathbf{J}^{+}\mathbf{J})\Delta\mathbf{z} \qquad (1)$$

where $\Delta\theta$ is the joint variation vector, $\mathbf{I}$ is the identity matrix, $\mathbf{J}$ is the Jacobian matrix of the set of Cartesian constraints, $\mathbf{J}^{+}$ is the pseudo-inverse of $\mathbf{J}$, $\Delta\mathbf{x}$ represents the variations of the set of Cartesian constraints (main task), and the secondary task $\Delta\mathbf{z}$ is used to minimise the distance to the attraction posture.

It is now possible to solve the problem by applying the following algorithm:

```
For each frame within the time range:
  - Create the main IK tasks, based on Cartesian constraints (if necessary).
  - Get the Intermediate Skeleton posture, and set it as the attraction posture.
  - Successively compute solution to equation 1 until a convergence criterion is satisfied.
```

We do not solve separate IK problems for the arms or the legs, but set one global problem, with an attraction to the Intermediate Skeleton posture, that is, the original motion-captured posture. The resulting posture computed by IK gives us a new displacement map that the user can modify later. Moreover, since IK is an incremental method, we ensure continuity by computing the solution at t starting with the posture we had at time $t-1$. Consequently, it is always better to compute the solution over the whole time range, rather than sporadically.

3.3. Smoothing with multiples constraints

3.3.1. Introduction

Ensuring smooth transitions between captured and corrected postures is one of the key issues when retargeting the motion. We assume that, for each constraint i:

- The constraint is **active** over the time range $[start_i, end_i]$
- The constraint is **eased-in** over $[smooth\_in_i, start_i]$ (*input ramp*) before the activation of the constraint.
- The constraint is **eased-out** over $[end_i, smooth\_out_i]$ (*output ramp*) after the activation of the constraint.

Imagine that the End User Skeleton has to reach a ball with the right hand, with values $smooth\_in = 20$, $start = 30$, $end = 50$ and $smooth\_out = 60$. This means that the posture is the original motion-captured posture until time 20. Over $[20, 30]$, there is a smoothing between the motion-captured posture and the corrected posture in order to reach the ball at time 30, and over $[30, 50]$, the arm follows the ball. Finally, over $[50, 60]$, the smoothing occurs between the posture that we obtained when reaching the ball at time 50 and the motion-captured posture at time 60. As one notice with this example, ensuring the smoothing requires to know in advance the postures that will occur in the *future*. To retrieve these postures, we suggest a two-pass computation. During the first pass, no smoothing is performed. This gives us the key postures at times $smooth\_in$, $start$, end and $smooth\_out$. The second pass handles the smoothing. In order to be able to handle multiple constraints, different smoothing methods are applied: for the *input ramp*, the smoothing takes place in the Cartesian space, whereas it smoothes the angular values for the *output ramp*.

3.3.2. Smoothing before the constraint starts

Smoothing is achieved by taking the two locations of an end effector at times $smooth\_in$ and $start$ and linearly interpolating the end effector position between these two positions. In our example, the first point is the location of the hand in space at time 20, and the second one is the position of the ball at time 30. A linear interpolation for the hand end effector then take place on the line joining these points during the time range $[20, 30]$.

3.3.3. Smoothing after the constraint ends

Unfortunately, the same method cannot be applied to the *output ramp*, mainly because of conflicts between multiple tasks: the position for constraint A at time $smooth\_end_1$ (computed during the first pass) might be disturbed by an other constraint B, for instance if B remains active while A is smoothly deactivated. This occurs for example when both constraints try to control a common subset of the hierarchy, e.g. when both hands try to reach something: each end effector tries to control the spine and it might be possible that the spine significantly changes its posture when one of the constraints is removed to favour the other one.

It is not easy to find a general solution to avoid this. However, the method that we have implemented gives good results when the adapted motion remains quite close to the original one: to achieve this, we linearly interpolate between the postures at time end_i and $smooth\_end_i$, that is, between angles for every node of both postures. This interpolation only takes place on the subchain controlled by constraint i (the subchain starts at the end effector node and goes back to the root). In our example, it will control the right arm, the right shoulder, and the spine.

4. Implementation and results

4.1. Implementation

Rather than starting from scratch, we developed a plug-in for 3D Studio Max, and extended its user interface with our own tools. Some of the functionalities provided by 3D Studio Max, like motion curves and multiple viewpoints helped a lot to test and correct the animations. We noticed that performances are highly dependent on the number and types of constraints. For complex cases, such as the karateka side kick presented in section 4.3, the computation took up to two minutes for five seconds of animation. On the other hand, Direct Motion Conversion, which is not an iterative process, was always applicable in real time.

4.2. Direct Motion Conversion

We applied successfully Direct Motion Conversion on various characters. As planned, the animation quality is highly dependent on the original End User Skeleton rest posture, but even without constraints, we achieved interesting results on characters with very different proportions and postures, as discussed in figure 7.

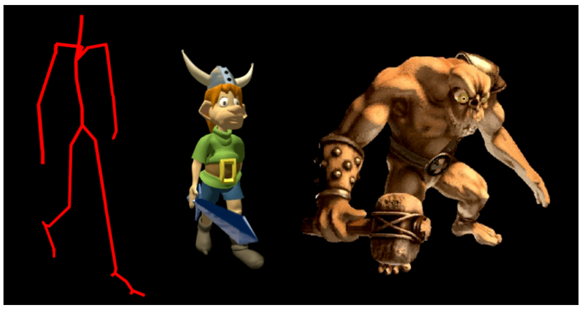


Figure 7: *We applied motion conversion to various characters (the orginal motion was captured on the red skeleton on the left). Even if the Vicking and monster's topologies, proportions and rest postures are very different, we successfully mapped a walking sequence on both characters (models courtesy of Art & Magic).*

4.3. Constraints and smoothing

This section presents some results with Cartesian constraints and smoothing. In the next figures, the original motion of the Performer Skeleton is represented by the smallest skeleton (on the right in figures 8, 9 and 11 and on the left in figures 12 and 13) and the corrected motion is applied to the other skeleton. We have tested the following types of constraints:

- a particular location in space (this can be a trajectory to follow).
- a position, relatively to a plane (above, on, or below).

These types can be extended, as long as they are converted to desirable locations and/or orientations for the nodes.

A good smoothed solution can only be obtained if the user carefully chooses the start and end times for a constraint. For instance, if our motion has two constraints, like picking and releasing a glass on a table, retargeting the pick-up and release locations involves to add two constraints. In our example, we first choose the release start time without any care. Playing the animation shows that the constraint should start sooner, since the Performer Skeleton hand at frame 126 for instance is closer to its target than the End User Skeleton hand (figure 8). By simply setting the starting time sooner, we produced a better solution (figure 9).

We also applied constraints to adapt a motion where the performer jumped from a platform to the ground. We used two different kinds of constraints here: when looking at the original motion, we noticed that the performer's feet were not moving while he started his jump, while there was a litle slide when he started to hit the ground. Thus, we set one point constraint for each foot on the platform at the beginning of the sequence, and two planar constraints to force the feet to be on the plane at the end of the animation (figure 10). The in-between jump is computed by smoothing (figure 11).

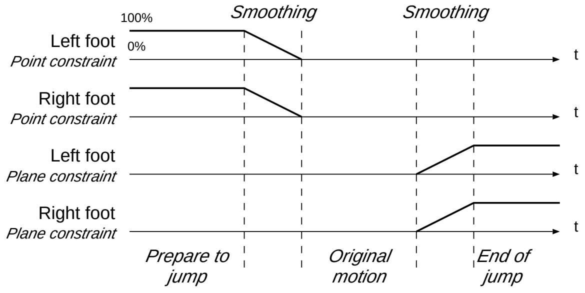


Figure 10: *Constraints for the jump.*

Our last example is a karateka side kick, a typical gesture in many fighting video-games. When retargeting the motion, we noticed that the original Performer Skeleton's support points on the ground were lost, since the End User Skeleton proportions were different. Moreover, the End User Skeleton hands, feet and knees were sometimes under the virtual floor, which is unacceptable (figure 12). Consequently, we added constraints to ensure that these body parts were above the floor. The corrected motion is presented in figure 13. We have to say that this sequence was a bit difficult to handle, because this kind of motion is quite fast and we do not have much key frames.

5. Discussion

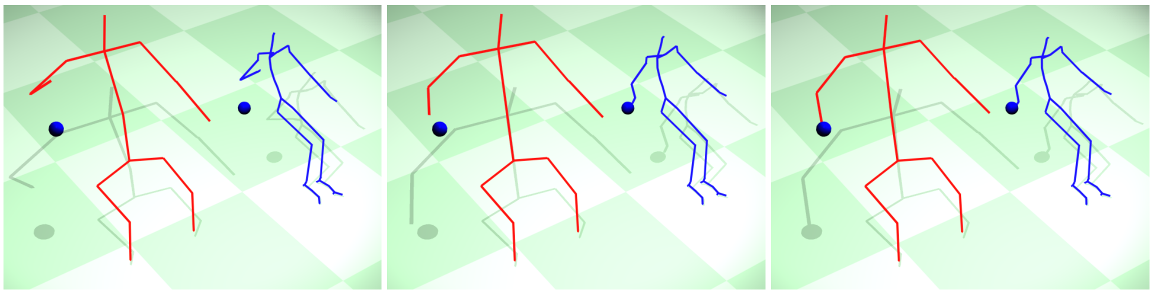
This section emphasises what is different between our method and previous approaches.

Our Intermediate Skeleton is different from Andrew Witkin and Zoran Popović's *abstract character*: it must be defined only once, whereas the *abstract character* depends on the animation and the properties that the animator wants to keep and has to be specified for each new motion. On the other hand, our model is much simpler and does not use physical elements, but this can be included in the IK solver.

Due to the way we preserve the motion-captured posture while enforcing new constraints, we have to apply IK to the whole hierarchy, starting from the root, rather than defining sub-chains for the arms, the legs and the spine, as Bindiganavale and Badler[2] did.

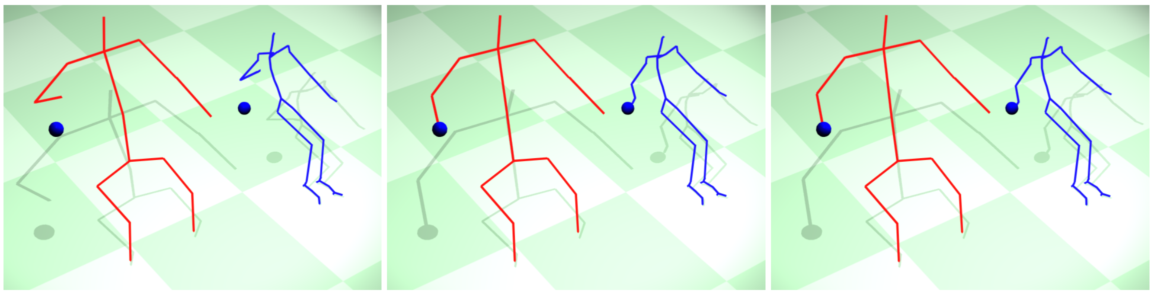
Regarding the adaptation to more complex structures (End User Skeleton having more nodes than the Performer Skeleton), a simple approach could be to lock some unused degrees of freedom in the End User Skeleton and make a one to one correspondance for the remaining DOFs using the Anatomic Binding. Modules like Motion Converter and Motion Composer will work without any problem on this structure. Other schemes are possible, for instance, distributing one DOF value onto multiple End User Skeleton DOFs according to a predefined mapping.

Finally, this implementation of Inverse Kinematics does



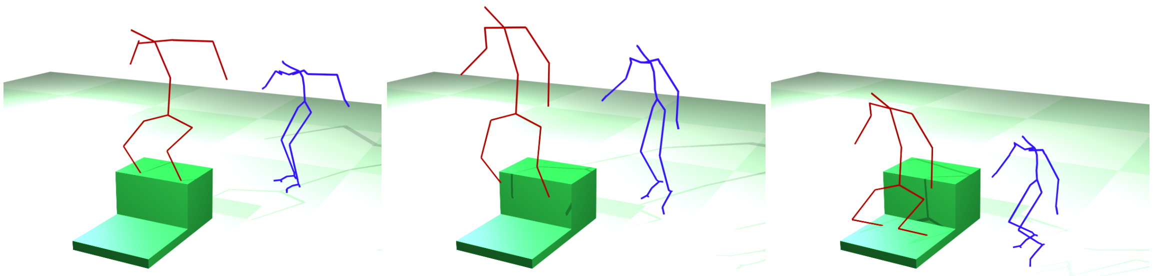
(a) Frame 112 (b) Frame 126 (c) Frame 137

Figure 8: *The Performer Skeleton (on the right) is going to release an object at the location specified by the ball, and we retarget the motion onto the End User Skeleton (on the left), by constraining its right hand to also reach an other location. When we first set the starting time for the constraint, we notice that at frame 126, the Performer Skeleton hand is very close to the ball, while the End User Skeleton right hand is too far from it.*



(a) Frame 112 (b) Frame 126 (c) Frame 137

Figure 9: *By simply decreasing the starting time for the constraint, we achieve a better result: the End User Skeleton right hand is now closer to the ball at frame 126.*



(a) Frame 55 (b) Frame 77 (c) Frame 88

Figure 11: *We retarget the Performer Skeleton's jump (on the right) to the End User Skeleton (on the left). Since platforms supporting the motion are different for the End User Skeleton, we had to set constraints to avoid the End User Skeleton feet to go into them.*

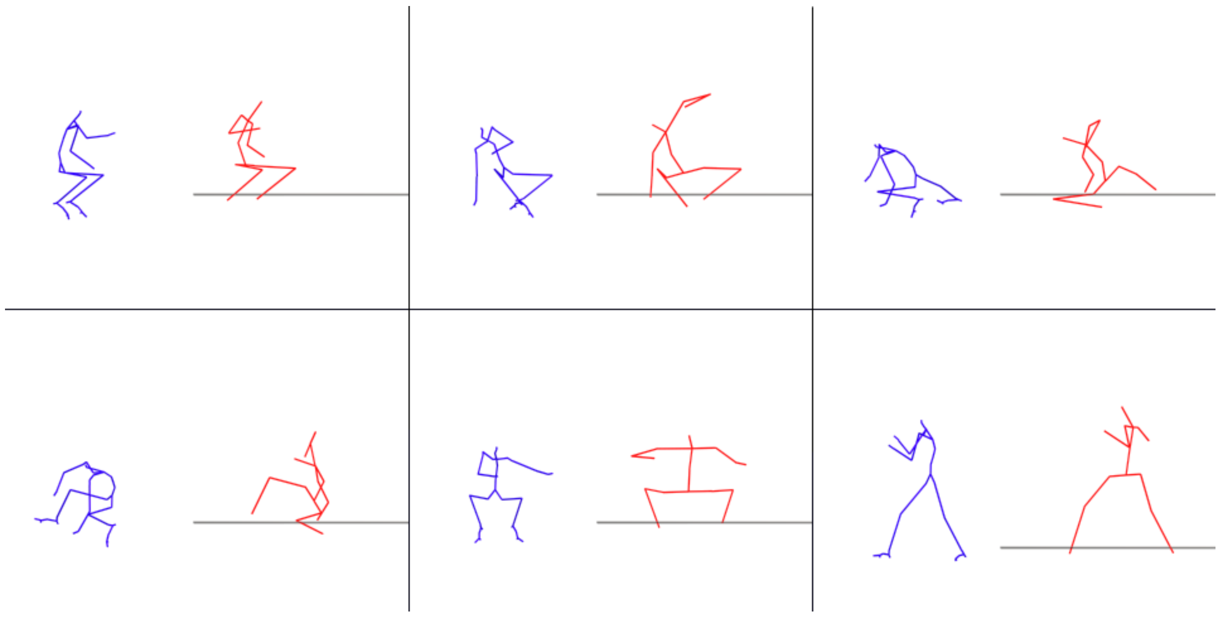


Figure 12: *Karateka motion: before correction. Without any constraint, the End User Skeleton's hand, feet, and knee go into the ground (represented by the horizontal line).* Note: *the Performer Skeleton is in on the left and the End User Skeleton on the right, and frames are ordered from top-left to bottom-right.*

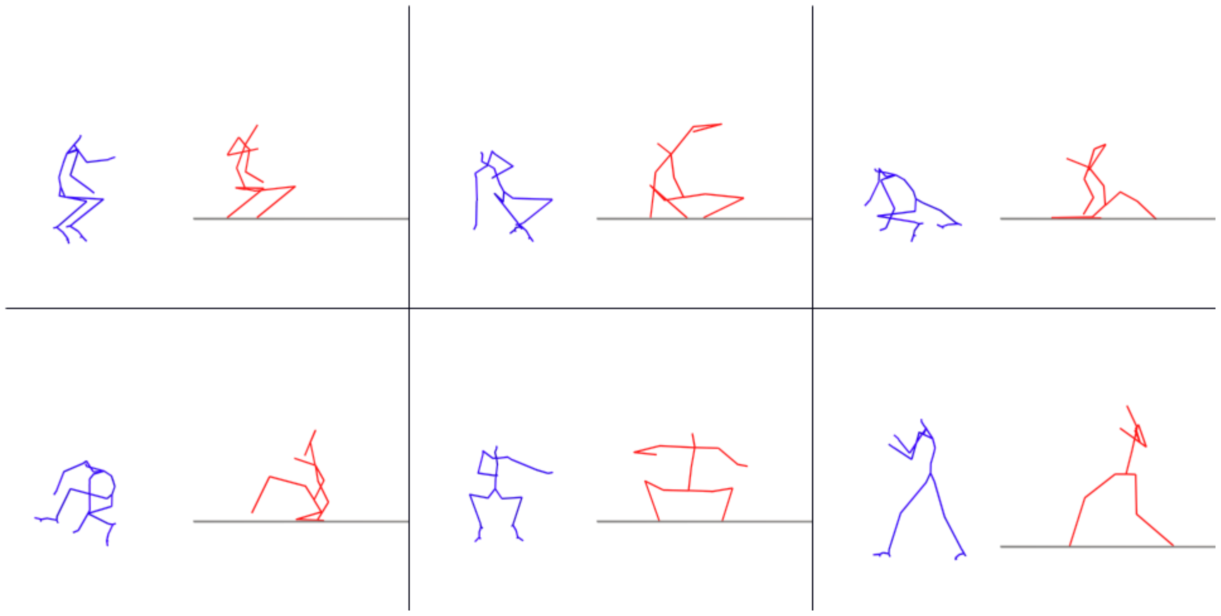


Figure 13: *Karateka motion: after correction. Constraints set on the End User Skeleton give a more pleasant result.*

not converge if *goals* of the tasks are part of the character's hierarchy. For instance, there would be no solution if the goal of the character's left hand is to touch its own right hand.

6. Conclusion

We have presented in this paper a method to retarget a motion from one character to a geometrically and topologically different one, and demonstrate how Inverse Kinematics can conserve the motion-captured posture while enforcing Cartesian constraints. Our tool has been successfully applied to video-games projects. Extensions could be to define more complex mapping functions, rather than one-to-one correspondences, to be able to handle coupled DOFs. Priorities could be assigned to the tasks so as to favour one task rather than another when they come into conflict, as Baerlocher[1] *et al.* This is very useful for conflicting tasks, like trying to grab distant objects when only one hand can reach the desired location at a specified time: in such cases, priorities are an easy way to favour one task, for instance, grab the object with the left or the right hand.

Acknowledgements

The authors would like to thank Art & Magic for their 3D characters. The research was supported by the Federal Office for Education and Science in the framework of the MOCA ESPRIT project.

References

1. Paolo Baerlocher and Ronan Boulic. Task priority formulations for the kinematic control of highly redundant articulated structures. In *IEEE IROS '98*, pages 323–329, 1998.

2. Rama Bindiganavale and Norman I. Badler. Motion abstraction and mapping with spatial constraints. In N. Magnenat-Thalmann and D. Thalmann, editors, *Modeling and Motion Capture Techniques for Virtual Environments*, Lecture Notes in Artificial Intelligence, pages 70–82. Springer, November 1998. held in Geneva, Switerland, November 1998.

3. Ronan Boulic and Daniel Thalmann. Combined direct and inverse kinematic control for articulated figure motion editing. *Computer graphics forum*, 2(4):189–202, 1992.

4. Armin Bruderlin and Lance Williams. Motion signal processing. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 97–104. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

5. Michael Gleicher. Retargeting motion to new characters. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 33–42. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.

6. Jehee Lee and Sung Yong Shin. A hierarchical approach to interactive motion editing for human-like figures. *Proceedings of SIGGRAPH 99*, pages 39–48, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.

7. Zoran Popović and Andrew Witkin. Physically based motion transformation. *Proceedings of SIGGRAPH 99*, pages 11–20, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.

8. Andrew Witkin and Zoran Popović. Motion warping. *Proceedings of SIGGRAPH 95*, pages 105–108, August 1995. ISBN 0-201-84776-0. Held in Los Angeles, California.