An Overview of the Field of Inverse Kinematics

Joost van Dongen

23rd June 2007

Website: www.oogst3d.net
E-mail: tsgoo@hotmail.com
Course: Seminar Animation
Teacher: dr. ir. J. Egges
Study: Master Game & Media Technology
School: Utrecht University

Abstract

Animations, including those of human characters, are often created using skeletons. Calculating how a specific bone is positioned and oriented in space when the rotations of all bones relative to their parents are known is easy, but moving a bone to a certain position by rotating its parents is cumbersome. Inverse kinematics solves this by automatically calculating the required bone transformations to achieve a specific position for a specific bone. This literature study looks at several approaches to solving the inverse kinematics problem and looks at ways to handle situations where constraints cannot be achieved, or an infinite number of solutions exists.

Contents

1 Introduction

3D Computer graphics is a complicated business. Even relatively simple designs often require a skilled artist and a lot of time to be created. On the other hand, the volume and required quality of the work has increased immensely over the last two decades, as has the processing power that allows tools to take complicated tasks from the hands of the artist and automate them. A tool like ZBrush [1] has shown that introducing higher level tools to develop 3D artwork can have a large impact on both the quality of the produced work and the time required to produce it.

One high-level technique for creating animations that has been introduced into computer graphics early on, but still is far from a finished field of work, is inverse kinematics. Whereas normal skeletal animations requires the animator to rotate each bone by hand to create a certain pose, inverse kinematics allows the animator to set certain constraints, like the position of the feet and hand, after which an inverse kinematics algorithm can fill in the positions of all the other bones. This saves the animator a lot of work, while fine control is still maintained: if a pose is not correct, then it can be fixed by adding more constraints.

The flipside of inverse kinematics is that the constraints set by the artist are usually either over-constrained or under-constrained. If the constraints are over-constrained, then they are often either conflicting, which means it is not possible to achieve all of them and the algorithm will somehow have to choose how to handle this, or they are unachievable, which means that the algorithm should probably try to create the closest pose possible. If the constraints are under-determined, then an infinite number of different poses all satisfy the constraints and it is not known which one the artist wants. When animating humans, most of these solutions will probably look awkward and unnatural, so a wrong choice by the algorithm deteriorates the quality of the resulting animation or pose.

The usefulness of inverse kinematics is not limited to the aforementioned posing and animating of virtual characters for visual artwork and animation. In the field of ergonomics a task performed often is to check in a virtual model whether a person can reach a certain point. Rotating all the bones in the digital character's body is a tedious job, while just placing a constraint and see whether the character can reach it using inverse kinematics is a lot easier.

Another field where inverse kinematics can be of great help is motion capture. During a capture, moments were markers on the body of the motion actor are visible from too few or even no cameras at all happen and the motion capture software will have to fill in these gabs somehow. As the positions of most other markers are known, inverse kinematics can fill these gabs.

Probably the most difficult field to apply inverse kinematics in, is com-

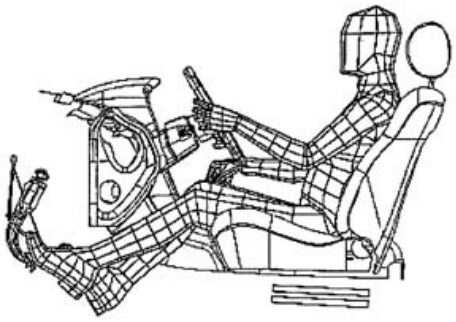

Figure 1: *A virtual character is used in ergonomics to see whether certain places on the dashboard can be reached. [2]*

puter games. Here inverse kinematics can be used to place the feet of a character on the ground when the slope of the ground varies, or make a character grab a certain object. This difficulty is that games requires high framerates and have many other tasks to do at the same time, so the inverse kinematics algorithm must be extremely efficient to be feasible.

Inverse kinematics is actually older than 3D computer graphics, as it started out as part of the field of robotics. There it is used for the purpose of controlling robot arms. However, as these arms are not virtual but real, some of the problems in that field are different from those in graphics. This allows for example robotics to simply avoid certain difficulties in the algorithms. These problems have to be solved in order to create believable humanlike virtual characters.

This article is a literature study into the field of inverse kinematics. Inverse kinematics is a difficult problem both from a algorithmic and from an aesthetic point of view. Several approaches to solving the problem in the first place will be discussed, followed by a number of ways to make the results look more aesthetically correct. Finally, some interesting alternatives to the common approach will be analyzed. Before it gets to the juice, though, it is time to formulate exactly what inverse kinematics is.

2 Problem definition

Inverse kinematics is usually considered to be a tool for the animation of skeletons, although some of the techniques that are discussed later on do not use skeletons. To exactly define what inverse kinematics is, it is therefore necessary to first specify what is meant with a skeleton. Thorough introductions into skeletons and inverse kinematics can be found in [3] and [2].

A skeleton is a tree of nodes that are linked together. Each node has

its own local transformation and the full transformation of a node is the concatenation of the nodes own local transformation and the transformations of all its ancestors. An example that shows what this means, is the human body. Some of the nodes in the body are the shoulder, the elbow and the wrist. If the shoulder rotates, then the elbow and wrist follow this rotation. If the elbow rotates, then the wrist follows this rotation as well. So the total rotation of the wrist consists of its own rotation and of its ancestors, being the elbow and the shoulder. If the other arm is added to this, then the shoulders of both arms are linked to the spine. Thus the whole becomes a tree.

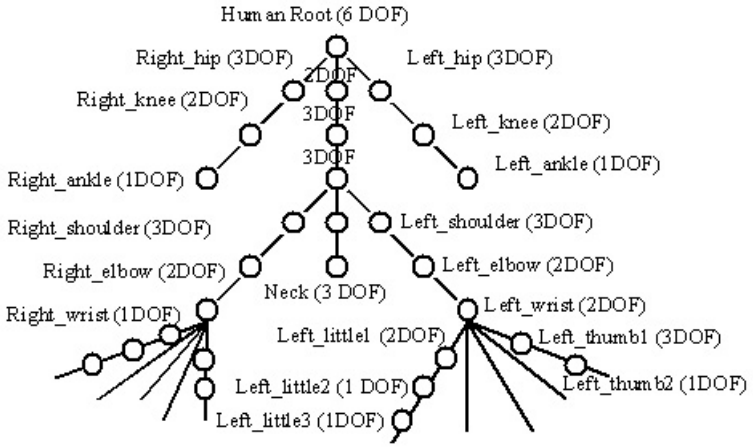

Figure 2: *An example of how the skeleton for a human character is modelled as a tree. [4]*

The joints in the skeleton can often only rotate. A shoulder joint can rotate in all directions and therefore has three degrees of freedom. An knee joint can only rotate over one axis and thus has only one degree of freedom. Depending on the application, bones might also be allowed to move relatively to each other. Such joints are called sliding joints. In computer animation, most uses of inverse kinematics concern the human body. The full inverse kinematics problem is not limited to any type of skeleton, though, as long as the skeleton has a tree structure. The topic is often found in robotics, where it handles the positioning of robot arms.

As matrix multiplication is a very simple operation, it is easy to calculate the total transformation of a single bone, simply by multiplying the transformation matrices of all the parents up to the root. Calculating the transformation of a specific bone is called forward kinematics.

As simple as forward kinematics is, so difficult is its opposite. Inverse kinematics is the problem of how to transform a set of bones to make sure an end bone is positioned or oriented in a certain way. While forward kinematics

has only one solution, inverse kinematics often has an infinite number of solutions. A set of bones can be transformed in many different ways to position an end-bone. Inverse kinematics is not limited to positioning or rotating a single end-bone. Any combination of constraints can be set for the algorithm to achieve. Some combinations will not be possible to achieve, in which case some way of handling this will have to be chosen.

A bone that has a constrained position and or orientation is called an *end-effector*. The bone chain from the root to the end-effector is called a *manipulator*. The *dimension* of the manipulator is the addition of all of its degrees of freedom, so if the bone chain is the shoulder, the elbow and the wrist, then that means $3 + 1 + 3 = 7$ degrees of freedom for the whole manipulator. If the end-effector defines both the position and orientation of the wrist, then the end-effector has six degrees of freedom. This makes the manipulator *redundant*: is has one degree of freedom more than the end-effector. If the wrist were not involved and the end-effector were applied to the elbow, then the six degrees of freedom of the end effector would be more then the 4 degrees of freedom left in the manipulator. Such a situation is called *over-constrained*.

As part of the definition of the skeleton, many systems also allow setting some extra constraints. Often the rotation of bones can be limited by the designer of the skeleton to remain within certain limits. This is used to prevent the skeleton from taking on unnatural poses, like an elbow that bends backwards. More complex constraints, like that parts of a body cannot interpenetrate, can sometimes also be defined.

3 Strategies to solving inverse kinematics

The problem of inverse kinematics is not linear, as rotations are involved. This means that analytical solutions are only available in limited situations. In all other cases, alternative methods will have to be employed. The most-used alternatives are both numerical solutions: linearization and optimisation.

3.1 Analytical solutions

Analytical solutions are the best option to use when available, as they are the fastest and most reliable inverse kinematics solvers. The problem of analytically solving inverse kinematics is that it does not scale to more complex bone sets and is therefore only an option for simple situations, like robot arms with few joints, or a single human leg. An example of an analytical solver for a single human arm can be found in [5]. This paper is also a nice example of the difficulty of find an analytical solution. Although the paper is called "Analytical inverse kinematics with body posture control",

the proposed algorithm is in fact not fully analytical: it is a combination of an analytical part and an iterative part.

3.2 Linearization

For more complex situations, numerical solutions are needed. Linearization is one such solution, as is shown in [3], [6] and [2]. This is an iterative technique that moves the end-effector closer to its goal transformation through many iterations. The idea is that although the full problem is not linear, if the iteration steps are made small enough they become increasingly linear. The set of bones is made linear through the use of a so called Jacobian matrix. This is an m*n matrix, where n is the number of degrees of freedom in the entire chain of bones, and m is the dimension of the end-effector. Using this Jacobian matrix, the bone chain is slightly transformed towards to goal of the end-effector. At the start of each iteration, the Jacobian matrix is re-calculated to create a new linearization.

This technique has several problems. The first is that to use the Jacobian matrix, is has to be inverted, which is only possible if it is square. However, as the manipulator often has more degrees of freedom than the dimension of the end-effector, this is rarely the case. This can be solved using the Moore-Penrose pseudoinverse [6].

Another problem is that of singularity. A matrix is called singular when two or more of its rows are linearly dependent [3] and a manipulator is in a singular configuration if the Jacobian matrix is singular. This happens when the set of bones is almost fully stretched. In this case, no single rotation of a bone brings the whole any closer to the goal of the end-effector. The result is that close to the singular situation, the bones will quickly flip between different solutions. This is not a problem when posing a manipulator, but it is during animation. In robotics, the solution to this oscillating bone movement around the singular situation is often solved by simply not bringing the robotic arm close to this state. This is not a solution in computer animation, though, as a character needs to be able to stretch its arms. [7] has shown that this problem is solvable, but at a higher computational cost.

3.3 Optimization

The third solution that can be utilized to solve the inverse kinematics problem is through the use of existing optimization algorithms. The problem is formulated as an optimization problem and the optimization itself is considered to be a black box that returns the resulting transformation of the manipulator. Examples of this can be found in [3] and [8].

The trick is to formulate the inverse kinematics problem as a minimization problem. The vector that stores the rotations of all the bones over all their degrees of freedom is called q. When for example moving an end-

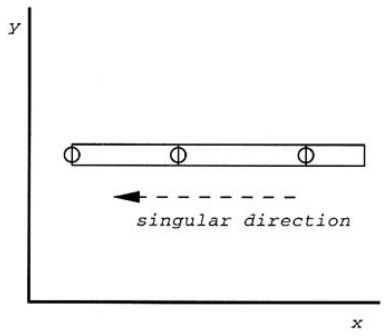

Figure 3: *A manipulator in singular direction does not work well with the Jacobian matrix approach. [3]*

effector x to position p, the distance from the current position to the goal position can be used as an error measurement that needs to be minimized. Using $x(q)$, which is the position of the end-effector x under the manipulator's vector of transformations q, this formula can be used for the optimization [3]:

$$minimize : E(q) = (p - x(q))^2$$

This simply means that the goal is to minimize the distance between the actual position of the end-effector and the desired position. The constraints on the problem can be the limits on the bone rotations. With l_i the lower rotational limit of degree of freedom i, u_i its upper rotational limit and q_i the rotation over a specific degree of freedom of a specific bone, this yields these constraints:

$$l_i <= q_i <= u_i i = 1...n$$

As $E(q)$ can in fact be any function, the optimization approach allows the definition of any kind of goal to be achieved by the inverse kinematics. [9] shows that for example keeping a bone on a line or plane, or keeping a bone within a half-space, are examples that can easily be expressed as goal functions to be minimized.

3.4 Comparison

If it is possible to use it, then the analytical solution is always the best choice. It is the only technique that can achieve exact solutions and it is very fast, as it does not require large amounts of iterations. However, most problems are too complex or have varying constraints and are therefore not fit for an analytical solution, while numerical solutions can handle more complex situations as well.

The problem with the solution based on linearization through the Jacobian matrix is that it does not scale well to situations were several constraints are active simultaneously, and that it works best when the initial setup is already relatively close to the final setup. This makes it a great fit for situations were a manipulator smoothly follows an animated constraint over time, for example in an animation or in a user interface where the user can drag the end-effector around. Calculating the Jacobian matrix for each iteration is computationally expensive, though, and this method has some problems around singularities that can only be solved by either avoiding them or significantly increasing computation time.

Unlike both other techniques, the optimization-based one scales easily to much more complex situations. The algorithm itself is very expensive, though, and it can sometimes be a problem that most non-linear optimization solving algorithms only find local minima, which means that the best solution is sometimes not found. This is often solved by starting from several different configurations to find more then one local minimum and use the best one found. Another problem of optimization is that the algorithm itself is considered to be a black-box that might do anything, including jumping from one configuration to a totally different one. This means that continuity of the solution is not guaranteed. This is problem when an artist drags the end-effector around and the bones quickly flip to different solutions, while the end-effector positions are in fact very close to each other.

4 Handling redundant and over-constrained manipulators

As was shown earlier, manipulators often have less or more degrees of freedom than the end-effector. When the end-effector has more degrees of freedom than the manipulator, the chain is over-constrained and cannot achieve the goal. When the end-effector has less degrees of freedom than the manipulator, then many situations accept an infinite number of different solutions. However, these are of different quality to the user, as some poses are less natural than others. Both redundant and over-constrained manipulators need to be handled in a way that makes sense to the user and gives the best result. An undisputed definition of what this "best result" is does not exist, so defining what the best result is, is part of the problem.

4.1 Over-constrained manipulators

The easier of the two problems is the over-constrained one. Here it is not possible to achieve all goals and some way of handling this has to be chosen. Common choices are on the one hand trying to get close to all constraints but reaching none, and on the other hand choosing certain constraints to

be achieved, while not achieving others. [2] suggests a system where each constraint has a certain priority and priorities are solved downwards, so the constraint with the highest priority can never be disturbed by those of lower priority. In the over-constrained case, what works best really depends on the application that the algorithm will be used for.

4.2 Redundant manipulators

Much more complex to handle is the case of redundant manipulators, especially in the case of human-like figures. Humans have preferences for certain poses and movements and are very adept at noticing oddities of this type in virtual characters. A common approach in robotics is to prefer manipulator configurations that are far from a joint's extreme angles. This does not work for humans, though, as many natural poses involve arms that are almost fully stretched or bent.

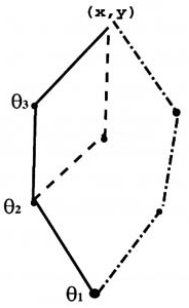

Figure 4: *In this situation, several different bone configurations can all achieve the goal at (x, y). [3]*

Some extra constraints that prevent the worst poses from occurring have already been mentioned: setting limits to the angles that joints can bend to and avoiding collisions between parts of the body. This chapter discusses some more complex solutions to the problem.

4.2.1 Parameterization of redundant degrees of freedom

A solution that is seen for example in [5] and in the commercial 3D design package 3D Studio MAX [10] is to parameterize the redundant degrees of freedom. In [5], this parameterization is called the swivel angle. The case that is analyzed there is that of the arm, which has seven degrees of freedom (shoulder, elbow, wrist), while the end-effector has only six degrees of freedom. This extra degree of freedom allows the elbow to twist to several positions. The swivel angle is a parameter that defines which position the elbow will take and the artist can set this parameter. If the desired situa-

tion is not achievable, for example due to collision with the body, then the position closest to the desired swivel angle is chosen.

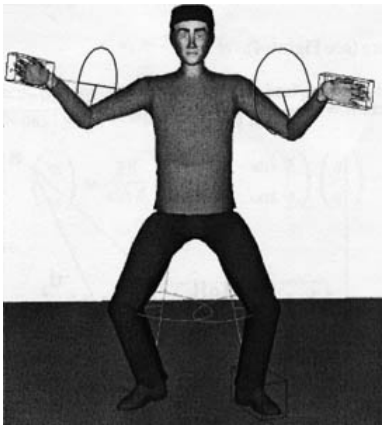

Figure 5: *The circles at the elbows can be rotated to define the swivel angle. [5]*

3D Studio MAX uses this same swivel angle to parameterize any number of redundant constraints at the same time. Together with the vector from the start of the manipulator to the end-effector, the swivel angle defines an infinite plane. The algorithm uses this to attempt to position the entire solution inside this plane. This does not remove all redundancy from the system, though.

4.2.2 Physics based optimisation

[8] proposes a solution to the redundancy where the body is modelled and simulated using physical properties. Muscles exert forces on the joints to make them rotate and forces like gravity and the friction of the foot soles with the ground are taken into account. Movement is defined by a number of constraints, like the positions of the feet on the ground during a walking animation. The algorithm now attempts to find a movement of the body that uses as little energy as possible. This is based on the idea that human locomotion is energy optimal. The benefit of this solution if that it prevents sudden changes in posture that often occur in other inverse kinematics solutions.

The drawbacks of this solution are significant. Foremost, it is extremely complex to calculate. The algorithm in [8] takes ten to twenty minutes to calculate some ten seconds of motion, which is orders of magnitude from real-time and is thus not useable in either games or situations where an artist needs the possibility to tweak things. A much more significant drawback of this solution is that it has not been proven that human locomotion is energy

optimal. In fact, the way a human being moves even varies with his or her mood. [8] attempts to solve this by defining a total of 147 parameters for the whole body. These variables capture the style of a motion. Deriving proper values for these parameters is very complex, though, so the result is that a style is not only dependent on a person, but also on the type of movement this person is making. So person A walking has a different style than that same person A running.

4.2.3 Posture blending

A solution that places the responsibility for defining what a "good solution" is in the hands of the artist, is posture blending. Here the artist defines a number of poses and feeds these into the inverse kinematics algorithm. When choosing among different solutions, the algorithm now finds the one that looks most like one of the example poses. An example of this can be found in [11].

[12] takes this idea step further and solves the constraints by interpolating between the example poses without using any of the previously described algorithms. Each example pose is parameterized by the position of certain bones. According to the parameters of the desired end-effector, several poses are blended using radial basis functions. This is implemented in such a way that a solution for an end-effector that is outside the range of the example poses can still be found through a form of extrapolation.

Posture blending is taken even further by the computer game Mechwarrior 3 [13], as is explained in [14]. Here complete animations are made for robots walking on different slopes in all directions. These complete animations are then blended in the game to allow the robots to walk on any kind of terrain. The algorithm is greatly simplified by defining exactly what animations are required to make it work, instead of the use of arbitrary sets of poses in [12]. This comes at a cost, though: each robot requires 152 different animations. The paper proposes some ways to speed up the animation process and automate some of the tasks, but the number of animations the animators have to create remains very high. The result in the game is very convincing, though, and works smoothly in real-time.

A mayor problem with blending between existing poses or animations, is that when blending two poses with equal weight, the resulting pose does not necessarily have its end-bone exactly in the middle of the end-bone positions in the two original poses. [12] measures errors of up to 24% in a number of example cases and proposes a number of solutions to decrease this error. Exact solutions can only be achieved by adding a final step of traditional inverse kinematics, though.

Figure 6: *Figure: An image of a robot in the computer game Mechwarrior 3. [14]*

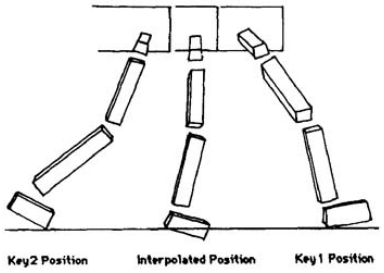

Figure 7: *Blending the left and right pose of the leg results in the middle pose. Although the original poses both touched the ground correctly, the blended pose intersects with the ground. [6]*

5 Mesh-based inverse kinematics

So far, everything that has been discussed was based on skeletal animation. An interesting example of a technique that uses a wholly different approach, is mesh-based inverse kinematics. In [15] a technique is introduced where some standard deformations of a mesh are created by the artist and based on this, the artist can constrain any vertex and the algorithm tries to find a mesh-deformation that fits the constraints. These solutions are found by interpolating and extrapolating the given examples. Mesh-based inverse kinematics has the benefit that the artist need not define a skeleton and a translation from skeleton to mesh. It also allows the use of inverse kinematics on objects that do not lend themselves easily to skeletons. An example given in [15] is that of a flag that ripples in the wind, but should achieve some positional constraints at the corners for a slightly odd animation of a walking flag.

Mesh-based inverse kinematics also fits well when more detail is needed

in the human body. If the deformation of the skin due to the contracting of muscles needs to be modelled, then simple skeletons cannot do the job. By delivering a human figure in several example poses, where each pose has the correct shape of the skin, mesh-based inverse kinematics can animate this correctly. These examples can be acquired using a 3D scanner, but this does require that the topology of the triangles in the mesh is the same for each example.

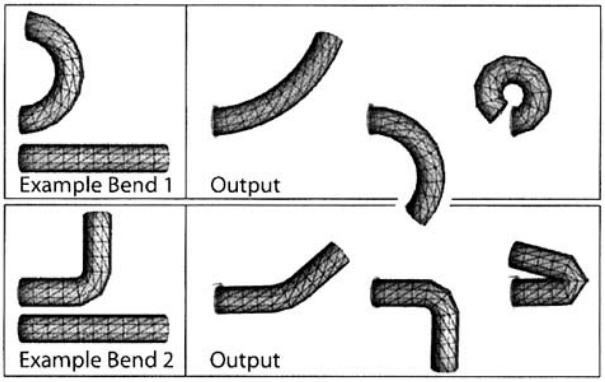

Figure 8: *On the left some example bends are created for the mesh-based inverse kinematics algorithm. On the right the algorithm is applied to achieve certain constraints. Depending on the given examples, the bend will work very differently. [15]*

Not only does it often take quite some work to create example meshes, it is also the case that the mesh-based inverse kinematics algorithm as presented in [15] is too slow to manipulate meshes of more than 10.000 triangles and 10 example poses in real-time. [16] manages to increases the efficiency of the algorithm to allow the manipulation of a model with 42.000 triangles in real-time. The technique used there is to automatically search for near-rigid parts of the body and grouping these together, thus greatly reducing the complexity of the mesh.

6 Conclusion

Inverse kinematics is a tool that is of great use to computer animation and is already being used extensively for many years. Nevertheless, it is also a field of research that is far from finished. This is exemplified by the inverse kinematics tools provided in 3D Studio MAX [10], one of the most widely used 3D design and animation tools. In over ten years of development of different versions of this application, two main inverse kinematics tools have been introduced into the program. Of these, the so-called HD-solver

becomes very slow in long animations and gives far from continues results, suddenly flipping the bones around certain positions. The other algorithm, called the HI-solver, does not have these problems, but only allows for a single end-effector per manipulator.

Most of the papers discussed in this literature study solve these problems and support much more complex uses of inverse kinematics, but each of the papers introduces new drawbacks, thus never resulting in a final solution to the problem. Some techniques are too slow for real-time, while others require too much work from the artist.

Several algorithms that solve inverse kinematics in the first place have been developed. The most important problem is coping with redundant manipulators and many different approaches to handling this have been proposed. More knowledge of human locomotion might help here. Hopefully, some of these techniques will be developed far enough to soon be efficiently applicable in production tools like 3D Studio MAX and in computer games.

References

[1] Zbrush. Pixologic, 2000.
www.pixologic.com.

[2] Paolo Baerlocher. Inverse kinematics techniques for the interactive posture control of articulated figures. Ecole Polytechnique Federale de Lausanne, 2001.

[3] Chris Welman. Inverse kinematics and geometric constraints for articulated figure manipulation. Simon Fraser University, 1993.

[4] Jiyong Ma / Jie Yan / Ron Cole. Cu animate - tools for enabling conversations with animated characters. Center for Spoken Language Research, 2002.

[5] Marcelo Kallmann. Analytical inverse kinematics with body posture control. Computer Animation and Virtual Worlds, 2005.

[6] Michael Girard / A.A. Maciejewski. Computational modeling for the computer animation of legged figures. International Conference on Computer Graphics and Interactive Techniques, 1985.

[7] A. A. Maciejewski / C. A. Klein. The singular value decomposition: Computation and applications to robotics. International Journal of Robotics Research, Vol. 8, No. 6, pp. 63-79, 1989.

[8] C. Karen Liu / Aaron Hertzmann / Zoran Popovc. Learning physics-based motion style with nonlinear inverse optimisation. ACM SIGGRAPH, 2005.

[9] Jianmin Zhao / Normal I. Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. University of Pennsylvania, 1994.

[10] 3d studio max 8. Autodesk, 2005.
www.autodesk.com/3dsmax.

[11] Keith Grochow / Steven L. Martin / Aaron Hertzmann / Zoran Popovc. Style-based inverse kinematics. ACM SIGGRAPH, 2004.

[12] Charles F. Rose III / Peter-Pike J. Sloan / Michael F. Cohen. Artist-directed inverse kinematics using radial basis function interpolation. Eurographics, 2001.

[13] Mechwarrior 3. Microprose / Zipper Interactive, 1999.
www.mech3.org.

[14] Jerry Edsall. Animation blending: Achieving inverse kinematics and more. Gamasutra, 2003.

[15] Robert W. Sumner / Matthias Zwicker / Craig Gotsman / Jovan Popovc. Mesh-based inverse kinematics. Harvard University, 2005.

[16] Keving G. Der / Robert W. Sumner / Jovan Popovc. Inverse kinematics for reduced deformable models. Massachusetts Institute of Technology, 2006.