Application of Inverse Kinematics for Skeleton Manipulation in Real-time

Martin Fědor
fedor@fit.vutbr.cz
Faculty of Information Technology, Brno University of Technology

Abstract

Usual way of character's animation is the use of motion captured data. Acquired bones' orientations are blended together according to user input in real-time. Although this massively used method gives a nice results, practical experience show how important is to have a system for interactive direct manipulation of character's skeleton in order to satisfy various tasks in Cartesian space. For this purpose, various methods for solving inverse kinematics problem are used. This paper presents three of such methods: Algebraical method based on limbs positioning; iterative optimization method based on Jacobian pseudo-inversion; and heuristic CCD iterative method. The paper describes them all in detail and discusses practical scope of their use in real-time applications.

Keywords: skeleton, bone, joint, inverse kinematics, kinematic tree, Jacobian matrix, Newton-Raphson method, CCD method

1 Introduction

Graphics applications working strictly in real-time and demanding a fast on-input reaction are consisting of a variety of human-like figure animations, or generally speaking – the articulated structure animations. These structures are simplified abstraction of complex kinetic systems – the skeletons. Mostly they are used to simplify the manipulation with human bodies. Since the systems working in real-time are most often entertaining software and virtual reality systems, they can not be without some way of depiction of human beings. They must contain some sort of animation sub-system dealing with skeletons. Undoubtedly the most preferable way of human animation is the use of motion captured (*MoCa*) data for its undeniable realistic look and dynamics. These sets of data can be acquired by a various commercial devices which usually work either on magnetic or optic principle. *MoCa* data consists of recorded set of bones' orientations for every key-frame of animation. Implemented animation system then blends these orientations together that results in a smooth on-demand motion of a character. *MoCa* data in practice represents a motion library from which the animation system choose the appropriate animation or pose that most exactly reflects the action on user input.

A single key-frame in *MoCa* data comprises the set of orientation for every skeleton's bone. There are various ways to describe orientation in computer graphics. Most popular are these: transformation matrices, *Euler* angles, *quaternions* (or if the reducing of memory costs is necessary – the *exponential maps* [14]). Yet, in every of theses descriptions we say that the animation is defined in *Angular space*. That helps us to blend key-frames using some way of interpolation. However it is very unsuitable if constraints are given in *Cartesian space*[1]. Also in practice it is impossible to reduce the animation only to those dealing with prepared captured data and so that being restricted to the limited set of motions and their blending. Both these facts I demonstrate on following example:

Let us have an animated character whose task is to grab for an item placed at defined height in the scene. Although from *MoCa* data library one can combine an animation for inclining and grabbing, this motion surely will be suitable only for items lying at some heights - exactly the same as they were when *MoCa* data were acquired. In other cases the difference of item's and hand's position would be so large that the resulting animation tends to look very unpleasantly. Since in practice one cannot afford to store every pose for every item's height, the animation systems must have some other way how to deal with changing position constraints – constraints in Cartesian space. Unfortunately the angular space description is not much helping in fulfilling this task – computing the new skeleton configuration for a given set of position constraints is not a straight-forward and some sort of inverse kinematics method must be utilized.

---

[1] Usually these are conditions constraining the selected skeleton's point to be placed at the exact position in the scene space.

Let us call vector $\vec{\Theta}$ to be skeleton's configuration[2] in *Angular space*. Size of $\vec{\Theta}$ is denoted as structure's *Degree-of-Freedom (DOF).* The function f

$$\vec{x} = f(\vec{\Theta}) \qquad (1)$$

mapping angular space configuration into *Cartesian space* is called *Forward kinematics (FK). Inverse kinematics (IK)* is defined as follows

$$\vec{\Theta} = f^{-1}(\vec{x}) \qquad (2)$$

This work will describe the practical use of selected methods for solving inverse-kinematics with mutual use of animation captured by *MoCa*. A special emphasis will be put on sufficiently fast computation since the goal is to enhance the animation system to allow the real-time manipulation and editing of *MoCa* data *on-the-fly*. Described system will allow the user to define a set of constraints in Cartesian space and the inverse-kinematics solvers will compute appropriate skeleton's configuration in order to minimize the error below a given threshold.

In Section 2, selected previous works are discussed. Section 3 contains in-detail descriptions of implemented methods together with their advantages and drawbacks. Section 4 shows results of practical experiments from commercially used system and further discusses the scope of methods' use.

2 Previous Works

In the past, many authors and published works were focused on application of inverse kinematics in computer animation. The approaches to find skeleton's configuration in angular space for a given Cartesian space constraints can be split into a few groups according to a computational method used to solve it: analytical (determining a set of algebraical equations), iterative (root-finding or optimization of set of non-linear equations), hybrid (finding a minimum for a reduced skeleton set).

Early applications of inverse kinematics appeared in cybernetics in order to manipulate with robots (very early only with robotic arms). Many works come from this domain. Usually they described solutions for limited structure's DOF, surely it was because there had not been necessity for more complex robots. A set of algebraical equations for solving from 6 up to 9 DOF structures were derived in [16], [17]. In [4], [5], there are published methods for solving structures of 7 DOF. In practice, skeletons are usually more complex than robotic arms so that these approaches are valid only for interactive manipulation with limbs, or concurrently as a part of hybrid systems.

One of the most general method published in computer graphics is work by Zhao and Badler ([3]). They solved IK task as a problem of finding a local minimum – optimization of a set of non-linear equations defining Cartesian space constraints. Modified BFGS method was used to solve the system. Most cited approach to solve IK is to differentiate a set of non-linear equations (i.e. to perform a linearization) by constructing *Jacobia*n matrix and inverting it. Differential equations are then solved by numerical integration. Since the *Jacobian* matrix inverse is performed by so called pseudo-inverse (assuring the least-square solution), this method provides minimal angular velocity of rotating bones of a skeleton. Pseudo-inverse also allows for application of generally any number of additional tasks that is often utilized for other minimization criteria - such as balancing, or minimizing inertia matrix (see [1], [2], [13]). These methods allow to fully control skeleton of any DOF undergoing any number of constraints, moreover with defined priority of tasks.

However the pseudo-inverse of matrices, especially when large, is very time consuming operation. Alternatively it is possible to find linear combinations of joints' angular velocities by a simple *Jacobian* matrix transpose, although this also means to take closer care of singularity configurations of the skeleton (see [8]). In [7], there is proposed a way to effectively deal with nearly-singular configurations by applying a correction function. This function is based on a priori knowledge of skeleton's structure and on definition of preferred direction of motion near singularities[3].

Approach published in [11] uses hybrid systems – optimization method is used over reduced set of bones, algebraical equations solve the limbs' configuration.

Inverse kinematics is often used in computer animation as a part of system for motion editing and re-targeting. This technique is usually based on transferring the captured motion on a different skeleton which differs in bones' lengths, joints' stiffness and other parameters. Kwang-Jin and Hyeong-Seok in [9] use *Jacobian* matrix inversion and *null-space* to apply angular velocities from original animation. In result, the motion of the second skeleton satisfies given space-time constraints. Simple algebraical approach is used in [12] to compute limbs' orientation, maintaining the torso motion intact. These computations, as published, are processed in real-time. Off-line editing (yet in real-time) is described in [11]. Hybrid IK method is used: Zhao, Badler optimization for reduced set of DOF, limbs are positioned analytically (as in [12]). Resulting skeleton's pose is smoothly inserted into the animation sequence as a new knot in approximation B-spline. B-splines are used to easily parameterize bones' rotation.

---

[2] In practice it can comprise of set of Euler angles.

[3] Singularities usually happen when a limb of a character is fully stretched. Preferred direction in this situation assures the structure to leave the singularity. For example this could be an elbow's motion.

3 Applied Methods

The goal of this work has been to satisfy these two requirements: 1.) edit *MoCa* data; 2.) perform all computation in such a speed that would allow to control dozens of computer character simultaneously. In order to satisfy these a mix of various method has been used. Its advantage lies in scalability – one can sort out methods according to their speed and use the most efficient for a given task. The fastest method is less visually accurate, and vice versa.

Three methods have been implemented: algebraical, that has been based on previous works published in [4], [5], [9] and [12]; iterative optimization based on work by Zhao and Badler [3] with the difference in using root-finding *Newton-Raphson* method instead of minimization; and finally *Cyclic Coordinate Descent (CCD)* method [18] offering a fair trade-off between speed and accuracy.

3.1 Skeleton definition

Articulated structure consists of bones connected by revolving joints (i.e. socket-ball joint). All bones are perfectly rigid of length l_i. Every *i*-th bone has an associated coordinate systems T_i^F and T_i^W. T_i^F is a coordinate system defined according to a previous bone in structure. T_i^W is coordinate system in WCS (World Coordinate System, i.e. in the scene). Under these condition following equation is valid:

$$T_i^W = \prod_{j=0}^{i} T_j^F , T_1^F = T_1^W \qquad (3)$$

The structure may have arbitrary number of child bones, although in practice it is sufficient to count at most with three children in a hierarchy (it means a joint connection up to four bones). This is the case of breast joint – one bone goes up to the neck, two bones for shoulders and one goes down to the waist. Stiffness can be defined as a three-part vector - one part for every rotational axis. Rotation limits may be treated using *Euler* angles or, alternatively, using so called *limit-cones* – cones with convex or concave base whose surface defines a boundary within which two connected bones can move.

Articulated structure defined in this way is an acyclic graph in a form of *n*-ary tree. In practice, the sub-graph is constructed to speed-up the traversal and reduce computations only to those bones which are possibly affected by a motion. This graph we would call *Kinematic Tree*.

3.2 Algebraical method

This method has been chosen to control the skeleton whose end-effectors[4] are in proximity to constraints. Only limbs' orientations are affected by this method, assuming that torso animation is used from *MoCa* data without any modification. An informal definition of the method follows:

If the limb is constructed from pair of bones (T_i, T_{i+1}) of lengths (L_i, L_{i+1}), where $(\vec{p}_{T_i}, \vec{p}_{T_{i+1}})$ are original bones' positions, the point $\vec{p}$ is reachable if and only if $\left\| \vec{p} - \vec{p}_{T_i} \right\| < L_i + L_{i+1}$. Alternatively we can say that the joint $\vec{p}_{T_i}$ must lie in the center $\vec{p}$ of sphere of radius equal to sum of both bones' lengths. If we do not manipulate with other bones (we do not change inter-orientation of torso's bones) we can rewrite later condition so to be defined according to the center of whole character, i.e. the origin of the first bone:

The point $\vec{p}$ is reachable by a limb (T_i, T_{i+1}) if the character's origin $\vec{p}_{T_1}$ lies inside the sphere of radius $L_i + L_{i+1}$ and the center at $\vec{p} + \vec{p}_{T_1} - \vec{p}_{T_i}$.

If condition above is not satisfied then point $\vec{p}$ becomes reachable if $\vec{p}_{T_1}$ is translated onto the sphere's surface. In a case when more point conditions are defined (i.e. there exist points $\vec{p}_1 \ldots \vec{p}_n$) there also exist n possible spheres of reach. Then let us define a set *P* that is an intersection of all spheres $S_i \ldots S_n$. If point $\vec{p}_{T_1}$ lies in *P*, all points are simultaneously reachable. Otherwise, we must find a sufficient translation $\vec{t}$ that moves the origin onto the boundary of *P*. In a case that spheres $S_i \ldots S_n$ have no common and single intersection we must, at least, move the origin onto the border of the largest existing intersection[5] and simultaneously to the closest proximity of other spheres.

---

[4] End-effector is a point on skeleton (usually ulterior end of a bone) that is constrained by some condition in Cartesian space.

[5] Largest intersection in a meaning of the biggest number of participating spheres.

Unfortunately solution of this task is not trivial as it is not trivial to find and parameterize the surface of *n*-spheres' intersection.

Heuristic sub-optimal approach has been proposed. It needs only *n* steps to compute the translation vector (*n* is number of conditions). The approach consists of three steps, executed in order as follows:

1. We set appropriate bit in *w* word for every *i*-th sphere boundary in which the skeleton origin lies (we suppose up to 32 simultaneous conditions).
2. We test every successive bit *32-i* (it means in the descent order) and for every zero bit we find point $\vec{p}_{S_i}$ on sphere's surface S_i. This point lies on line connecting actual origin's position and the center of sphere S_i.
3. We do the test as in 1.). If $w_{new} < w_{old}$, we return to the old position of the origin and $w_{new} = w_{old}$. Otherwise we found new good position of the origin. We continue with step 2.).

This approach can assure that all condition with higher priorities will be satisfied. Priorities are indirectly defined by index value, i.e. these conditions with lower indices have higher priorities than those with higher indices. If we sort the conditions according to their priorities we are certain that found translation fulfils the maximum number of higher priority conditions. However, this translation cannot be optimal in a meaning of minimizing the residual difference of all defined conditions.

Approach described above has been successfully applied in practice and has been found to be sufficient. For sake of completeness it is likely to say that the priority can be effectively tuned by modifying spheres' reach radius: $r'_{S_i} = k_i \cdot r_{S_i}$. If $k_i < 1$, *i*-th condition would have a higher priority.

The next step is to find configuration of the limb (T_i, T_{i+1}). Orientation of every bone defines a sphere in space of a radius equal to bone's length. Distal end of bone (the other joint) lies on a surface of this sphere. If both spheres of the limb intersects then the common joint (e.g. an elbow or a knee) lies on a circle determined by this two-sphere intersection. We can choose every point on this circle for limb to be in a valid configuration[6]. Unfortunately this vague definition also means a serious problem in finding *exact* point where the joint might be placed. A good strategy is to choose the point that is the closest to previous (i.e. old) joint's position. Our system also allows for the definition of a vector $\vec{v}$ in WCS that further determines selected point on the circle: for a given vector $\vec{v}$ the point $\vec{p}_v$ on the circle is found; if the nearest point to the old position is denoted as $\vec{p}_n$ the resulting point on circle is computed as the linear combination $\vec{p} = \alpha\vec{p}_n + (1-\alpha)\vec{p}_v$ and normalized. This somehow "tricky" approach guarantees the minimal bone's rotation and allows to define the secondary condition for placing the joint in WCS.

3.3 Iterative optimization method

This iterative method has been applied because of necessity to manipulate with the kinematic tree of arbitrary DOF with arbitrary number of constraints. Method is similar to the approach published by Zhao and Badler ([3]). However instead of using BFGS numerical method for finding local minimum of set of non-linear equations, simple *Newton-Raphson* root-finding method has been used. The method can support any kind of constraint if it can be differentiated according to the rate of change of skeleton's configuration – i.e. the *Jacobian* matrix can be constructed. Described system supports these constraints: point-to-point, bone's orientation, center of mass placement.

Let us have a kinematic tree configuration $\vec{x}$. Its parts can be quaternion parameters, matrix parameters or, as in our case, three *Eule*r angles for every bone in a tree. Next, let us have a vector function $F(\vec{x})$ returning a difference between actual state of conditions and constraints. The solution is found if $F(\vec{x}) = \vec{0}$. Function *F* can be expanded in Taylor series

$$F(\vec{x} + \delta\vec{x}) = F(\vec{x}) + J \cdot \delta\vec{x} + O(\delta\vec{x}^2) \quad (4)$$

By neglecting terms of order $O(\delta\vec{x}^2)$ and higher and by setting $F(\vec{x} + \delta\vec{x}) = 0$, we obtain a set of linear equations:

$$J \cdot \delta\vec{x} = -F(\vec{x}) \quad (5)$$

By inverting *Jacobian* matrix *J* we obtain the Newton step $\delta\vec{x}$:

$$\delta\vec{x} = -J^{-1} \cdot F(\vec{x}) \quad (6)$$

The step is then added to old vector yielding a new configuration:

$$\vec{x}_{new} = \vec{x}_{old} + \delta\vec{x} \quad (7)$$

In [1],[2], the equation (6) is in homogenous form

$$\delta\vec{x} = -J^{-1} \cdot F(\vec{x}) + (I - J^{-1} \cdot J)\Delta z \quad (8)$$

and is integrated so that to give a resulting tree's configuration. The presented system uses *line-search* method ([6]) which determines Newton's step length $\delta\vec{x}$ so that it minimizes the norm of difference

---

[6] This is valid only if no rotational limits are defined.

$$f = \frac{1}{2} F \cdot F \qquad (9)$$

As already mentioned, this method requires the *Jacobian* matrix to be constructed for every constraint. It can be easily derived how to do it for a various types. For sake of brevity I would mention only three most used constraints (see more in [2], [3]):

Point-to-point constraint

Rows of the matrix are linear velocities of the end-effector according to angular velocity in every joint of kinematic tree: $\vec{v}_i = (\vec{o}_i \times \vec{r}_i)$, where $\vec{o}_i$ is a rotational axis of a joint *i*, $\vec{r}_i$ is a line between the end-effector and joint *i*.

Bone's orientation

Rows of the matrix are linear velocities of rotation of a bone *i* with respect to angular velocity in every joint: $\vec{v}_{xi} = \vec{o}_i \times \vec{x}_i, \vec{v}_{yi} = \vec{o}_i \times \vec{y}_i, \vec{v}_{zi} = \vec{o}_i \times \vec{z}_i$, where $\vec{o}_i$ is a rotational axis of a joint *i*, $\vec{x}_i, \vec{y}_i, \vec{z}_i$ are bone's axes.

Center of mass placement

Rows of the matrix are linear velocities of the center of mass of augmented[7] body. Because *i*-th joint rotates only by a part of kinematic tree, the weight of augmented body is modified by its ratio to the global weight of whole kinematic tree:

$\vec{v}_{gi} = \frac{m_{gi}}{m_g} (\vec{o}_i \times \vec{r}_{gi})$, where m_{gi} is weight of augmented body, m_g is weight of whole kinematic tree, $\vec{o}_i$ is rotational axis of a joint *i*, $\vec{r}_{gi}$ is a line between the center of mass and joint *i*.

The kinematic tree is usually highly redundant structure (i.e. DOF is much higher than the number of conditions). Therefore it means that *Jacobian* is non-square matrix and it is necessary to use some method of *pseudo-inversion* in equation (6) and (8). The author of this work uses *Moore-Penrose* pseudo-inverse in a form as follows:

$$J^{-1} \approx J^T (JJ^T + \lambda^2 I)^{-1} \qquad (10)$$

where λ is so called *dumping factor*. Damping is necessary mainly near the singular configuration where is no linear combination of angular velocities leading to requested linear velocity of an end-effector. Damping lowers computed angular velocity (Taylor series is valid only for small steps) and it increases overall stability of the method, on the other hand it reduces the convergence speed. More on this topic can be found in [8].

3.4 Cyclic Coordinate Descent method

This method is a good compromise between speed of algebraical method (Section 3.2) and accuracy of iterative optimization method (Section 3.3). It was introduced by Wang and Chan in [18]. CCD method removes the most time consuming operation – *Jacobian* pseudo-inverse, so that it provides the results much faster. Further advantage, when compared to algebraical method, is the possibility to compute configuration of arbitrary large kinematic tree. The main drawback might be the necessity to re-compute whole body orientation in every iterative step and the fact that it cannot converge in such a speed in general.

CCD is heuristic iterative method, which searches for appropriate bone's rotation in every single step in order to minimize defined constrain condition. The difference between actual condition's position and constrained position is computed and fitting rotation is applied. This process is repeatedly applied on every bone up (or down) the kinematic tree until defined conditions are met. When compared to the previous method, the iterative scheme is rapidly quicken and iterations are stopped in every step whenever the conditions' differences are below some threshold. Bones' rotations are dealt with independently. The previous method solved for every angular velocities simultaneously and this property gives it the power of finding optimal (i.e. minimal) rotations in contrast to CCD. From animator point of view this fact is also the most unpleasant feature of CCD method: rotations are not equally distributed throughout the whole kinematic tree, conversely a rotation is applied only to the first *n* bones if conditions are met. The animation then might look unnaturally. However the big advantage of CCD is relatively fast computation and in a case of small conditions' differences very quick convergence[8].

Usually in the literature, the CCD method is described for only one constraint and for a single kinematic chain ([15]). Offered generalized method is able to deal with arbitrary number of constraints with defined joints' stiffness. In the following text, only point-to-point constraints will be taken into account:

Let us define a position constraint $\vec{p}_{di}$ and a point on the skeleton $\vec{p}_{si}$ that must fulfil following condition $\|\vec{p}_{di} - \vec{p}_{si}\| = 0$. Next, if there exists a bone T_i with a joint at position $\vec{s}_i$, then we can construct two vectors – from joint to actual position and from joint to constrained position: $\vec{r}_{si} = \frac{\vec{p}_{si} - \vec{s}_i}{\|\vec{p}_{si} - \vec{s}_i\|}$, $\vec{r}_{di} = \frac{\vec{p}_{di} - \vec{s}_i}{\|\vec{p}_{di} - \vec{s}_i\|}$. Easily we can construct the quaternion q_i that represents the rotation necessary to align $\vec{r}_{si}$ with $\vec{r}_{di}$:

---

[7] This means the part of kinematic tree that is affected by rotation of bone *i*.

[8] It is worth to note that CCD is also immune to singularities.

$$\vec{v}_i = \vec{r}_{si} \times \vec{r}_{di}$$

$$\alpha = \frac{1}{2}\sin^{-1}(\|\vec{v}_i\|) \cdot \eta \qquad (11)$$

$$q_i = (\cos(\alpha), \sin(\alpha) \cdot \frac{\vec{v}_i}{\|\vec{v}_i\|})$$

If the bone *i* is predecessor of *n* bones which all have defined conditions $\vec{p}_i$ it is possible to construct *n* quaternions q_i^n. Combination of all quaternions yields the resulting rotation:

$$q_i = Slerp(q_{i,}^{n} Slerp(q_i^{n-1}, \ldots Slerp(q_i^2, q_i^1, \lambda^1) \ldots, \lambda^n)$$

where λ^i is a weighting factor (i.e. the priority – influence of *i*-th rotation). New bone's orientation T_i is computed by applying quaternion q_i. Parameter η in (11) is stiffness. *Slerp* is the spherical linear interpolation as defined in [19].

When new T_i is obtained, the whole kinematic tree down to the leaves is recomputed. This way the new end-effectors' positions are determined and the whole iterative process starts again. It is important to say that the order in which iterations are taken affects the new found configuration – the order indirectly defines the priority of every bone in a kinematic tree. As already mentioned, the iteration stops whenever conditions are met. The described system traverses the tree from leaves up to the root. This way assures the shorter bones (which are mainly in limbs) to have higher priority than bones in torso.

In a case that the structure is defined as a *free-floating* (i.e. allowing for root's translation), it is also possible to compute a translation vector that minimizes residual errors.

4 Practical Experiments

Described methods were successfully tested in practice (system was inserted into a commercial product). Data acquired by *MoCa* were interactively blended together and replayed in real-time – a character performed a typical sort of tasks like walking, running, crawling. Upon this, the real-time IK system was modifying the character poses so to meet a given set of conditions. Different methods were used in order to choose the best compromise between the speed and accuracy:

1. Algebraical method (Section 3.2) is used whenever the fast limbs positioning is necessary. Only bones in limbs are affected by computation. Character's torso is intact since it is assumed that a new pose is sufficiently close enough to the actual pose. This assumption proves to be valid for a large set of practical tasks. See Figure 1a-2d.
2. CCD method is used in cases when multiple conditions are defined and/or it is necessary to manipulate with more complex kinematic trees (i.e. the system happens to be redundant). See Figure 3a-3d: Both hands are placed on steering-wheel. When the wheel moves, the hands remain at defined places. Since the system have to deal with various vehicles (with steering wheels at different position and orientation) it is necessary for the character to move its back as well, otherwise the motion looks unrealistic. To accomplish this task, the kinematic tree was introduced with the root in hip joint. In order to assure that both elbows are not extremely twisted for some wheel orientation, the approach similar to that described in Section 3.2 is used – elbow position on a circle is modified by an a priori given vector pointing down. Described process was successfully applied simultaneously on dozens of characters with no visible slow-down.
3. Iterative optimization method (Section 3.3) is used when a complex manipulation is necessary. It gives the best results – the most realistically looking poses, but it is the slowest. See Figure 4a-4d and Figure 5a-5i. When the center of mass constraint is applied, the character remains in naturally looking pose in virtually every situation. Presented experiment constrains positions of all limbs (one of them was interactively moved) and a usual set of actions is taken (*MoCa* animations for walking, running, etc. - all in standing, kneeling and lying stance). The pose is computed so that all constraints are satisfied simultaneously. The center of mass is placed on a vertical axis going through the hip joint. Since the method contains very time consuming computation (from which the matrix pseudo-inversion is the most), the resulting animation is sufficiently fast for only two character at the same time.

All tests were run on a common PC equipped with AMD Athlon 1GHz processor. Influence of graphical sub-system on tests was marginal.

5 Conclusion and future work

Performed tests approved the practical helpfulness of inverse kinematics for many tasks in real-time applications. Interactive exact positioning of character's limbs is a necessity in major of nowadays entertaining programs. This work showed that the best results give a combination of methods for solving IK – the most suitable method is selected according to the given conditions, complexity of skeleton and a planned motion it should undergo. Also it was shown that the most accurate iterative method based on *Jacobian* construction and inversion cannot be massively used in real-time yet. Although it is able to manage the animation of one or two character at once, the usual demand on the system is to be able to manage dozens of them at iterative speed. It also should be mentioned that it is necessary to introduce even more constraints to assure that the result animation looks realistic. These constraints are usually the center of mass positioning and inertia matrix minimization. These additional constrains more extend system's dimension and slow down the computational speed.

The problems with realistic look can be generally solved by animating the character using the dynamics. It can be performed both by using forward or inverse dynamics. Yet, both these methods in practice (it means in very large scenes containing a lot of moving objects) lack of speed that is caused by large amount of collision tests and necessity of small integration steps to assure a suitable accuracy and stability. However it should be said that the dynamics simulation of animated characters will be certainly incorporated into the real-time applications very soon as the computational speed of common PCs is dramatically increasing every year.

Acknowledgements

This work was partially supported by grant no. GA102/02/0507, Czech Science Foundation

References

[1] Baerlocher, P., Boulic, R., *Task-Priority Formulations for the Kinematic Control of Highly Redundant Articulated Structures*, Proceedings of IEEE Conference Intelligent Robots and Systems IROS 98, Victoria (Canada), pp. 323 - 329, Oct. 1998.

[2] Baerlocher, P.,Boulic, R., *Kinematic Control of the Mass Properties of Redundant Articulated Bodies*, Proceedings of IEEE Conference on Robotics and Automation, ICRA 2000, San Francisco (CA), pp. 2557 - 2562, Apr. 2000.

[3] Zhao, X., Badler, N., I., *Inverse kinematics positioning using non-linear programming for highly articulated figures,* ACM Transactions on Graphics, 13(4):313–336, 1994.

[4] Zhao, X., *Kinematic Control of Human Postures for Task Simulation*, Dissertation thesis, University of Pennsylvania, December 1996

[5] Tolani, D., Goswami, A., Badler, N., *Real-time inverse kinematics techniques for anthropomorphic arms*, Graphical Models, Vol. 62, pp. 353 - 388, 2000.

[6] Press, et al, *Numerical Receipes in C*, Second Edition, Cambridge University Press, 1997

[7] Schreiber, G., Hirzinger, G., *Singularity Consistent Inverse Kinematics by Enhancing the Jacobian Transpose*, Advances in Robot Kinematics: Analysis and Control, Kluwer Academic Publishers, 1998, pp. 209-216

[8] Maciejewski, A., A., *Dealing with the Ill-Conditioned Equations of Motion for Articulated Figures*, IEEE Computer Graphics and Applications, Vol. 10, No. 3, pp. 63 - 71, May 1990

[9] Kwang-Jin, C., Hyeong-Seok, K., *On-line Motion Retargetting*, Proceedings of the International Pacific Graphics '99, Seoul Korea, October 5-7, 1999

[10] Dam, E., B., Koch, M., Lillholm, M., *Quaternions, Interpolation and Animation*, Technical Report, University of Copenhagen, July 17, 1998

[11] Lee, J., Shin, S., Y., *A Hierarchical Approach to Interactive Motion Editing for Human-like Figures*, Proceedings of SIGGRAPH 99. pp. 39-48, 1999

[12] Shin, H., J., Lee, J., Shin, S., Y., *Computer Puppetry: An Importance-Based Approach*, ACM Transactions on Graphics, Vol. 20, No. 2, April 2001, pp. 67-94

[13] Aydin, Y., Nakajima, M., *Realistic Articulated Character Positioning and balance Control in Interactive Environments*, Computer Animation, pp.160-168, May 26-29, 1999

[14] Grassia, F., S., *Practical Parametrization of Rotations Using the Exponential Map*, The Journal of Graphics Tools, volume 3.3, 1998

[15] Welman, C., *Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation*, Master thesis, Simon Fraser University, 1993

[16] Paul, R., P., Shimano, B., Mayer, G., E., *Kinematic Control Equations for Simple Manipulators*, IEEE Trans. Systems Man Cybernetics SMC-116: 449-455, 1981

[17] Manocha, D., Canny, J., F., *Efficient Inverse Kinematics for General 6R Manipulators*, IEEE Trans. On Robotics and Automation

[18] Wang, L., C., Chen, C., C., *A Combined Optimization Method for Solving the Inverse Kinematics Problem of Mechanical Manipulators*, IEEE Trans. On Robotics and Applications, vol. 7, no. 4, pp. 489-499, 1991

[19] Shoemake, K., *Animating rotation with quaternion curves,* Computer Graphics, 19(3):245--254, July 1985. Proceedings of SIGGRAPH'85

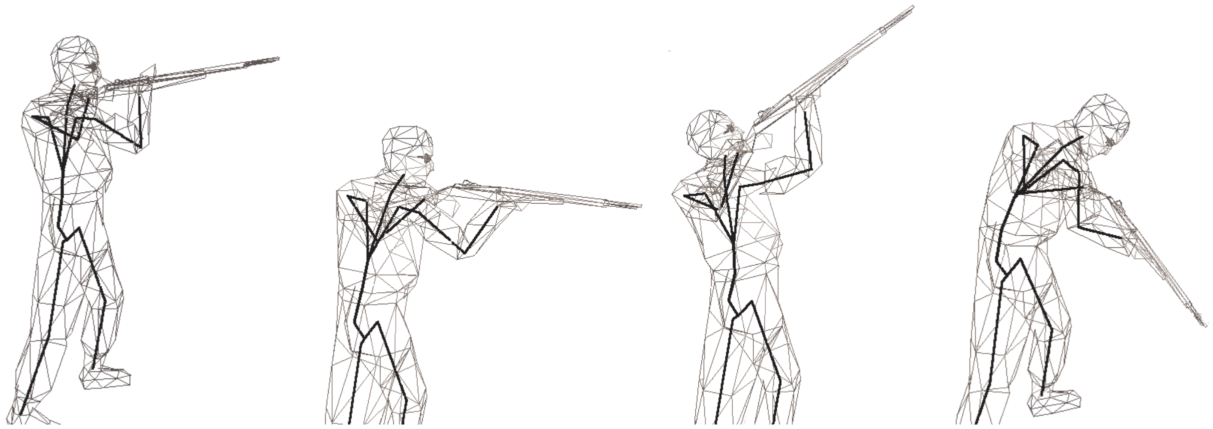

Figure 1 a ***Figure 1 b*** ***Figure 1 c*** ***Figure 1 d***

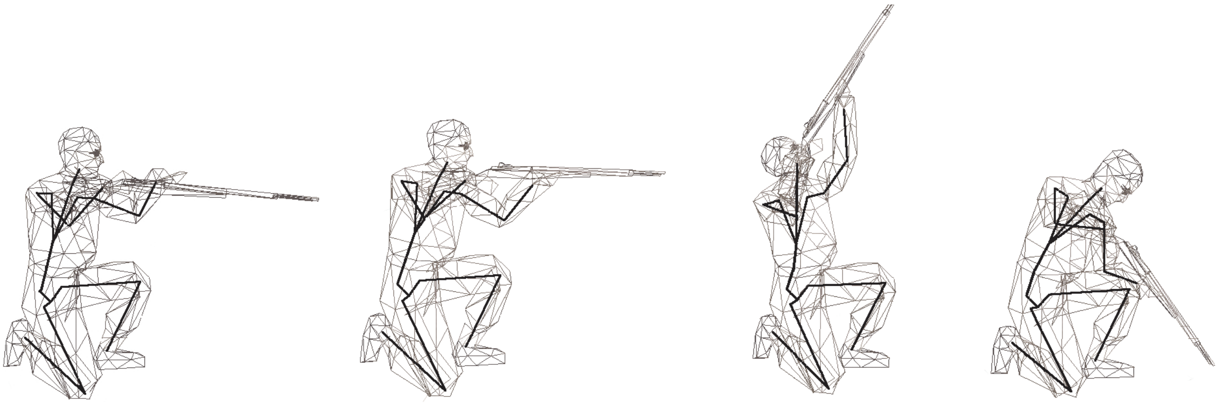

Figure 2 a ***Figure 2 b*** ***Figure 2 c*** ***Figure 2 d***

Figure 1a- 2d:

Algebraical method was used to fit hands on the weapon. Weapon is placed to the right shoulder with an exact orientation. Torso's joints were rotated in weighted amount so that to fit a given direction. Figures 1a and 2a show original pose without IK system. In order to maintain the left elbow down for every target direction, an additional vector pointing down was provided (see Section 3.2)

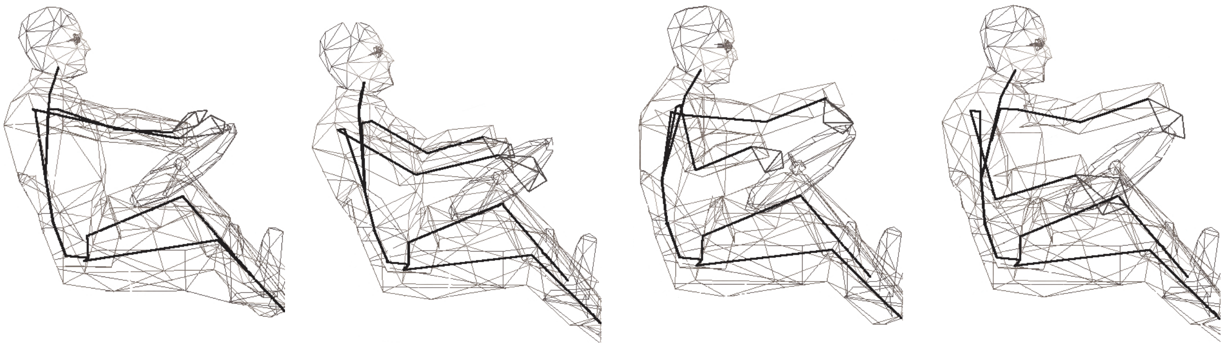

Figure 3 a ***Figure 3 b*** ***Figure 3 c*** ***Figure 3 d***

Figure 3a-3d:
CCD method was used to place both hands on the steering-wheel. Figure 3a shows original pose with no IK; Figure 3c – turning left, 3d – turning right. Note how the character's back is moving according to constraints' positions on the wheel. In order to maintain both elbows down, an additional vector pointing down was provided and a computation similar to that described in Section 3.2 was performed.

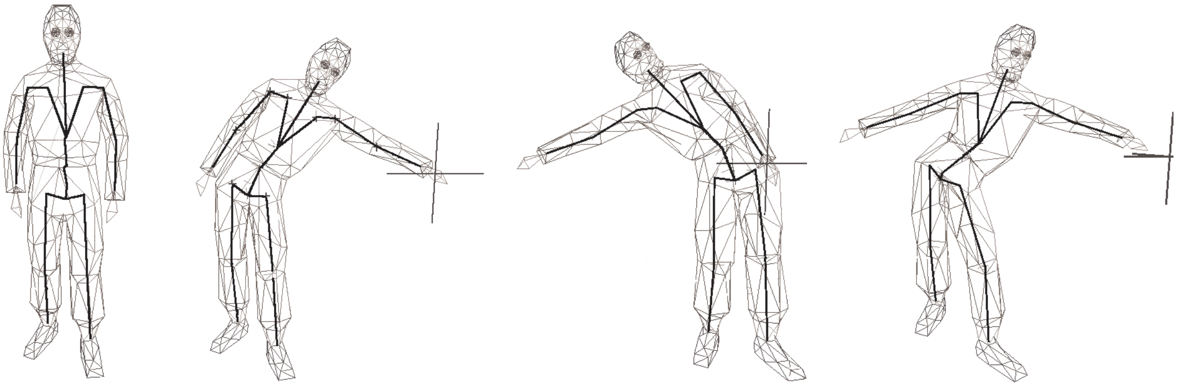

Figure 4 a ***Figure 4 b*** ***Figure 4 c*** ***Figure 4 d***

Figure 4a-4d:
Demonstration of implemented iterative optimization method (Section 3.3): Both hands and legs were constrained to be at an exact position, left wrist was interactively moved away. In order to remain realistically looking pose, the center of mass position constraint was also introduced – the center of mass was placed on vertical axis going through the hip joint. Figure 4a shows unconstrained pose (no IK). Figure 4b and 4c show same constraints but character interactively strafed from right to left. Figure 4d is a pose when left hand constraint was moved out of range.

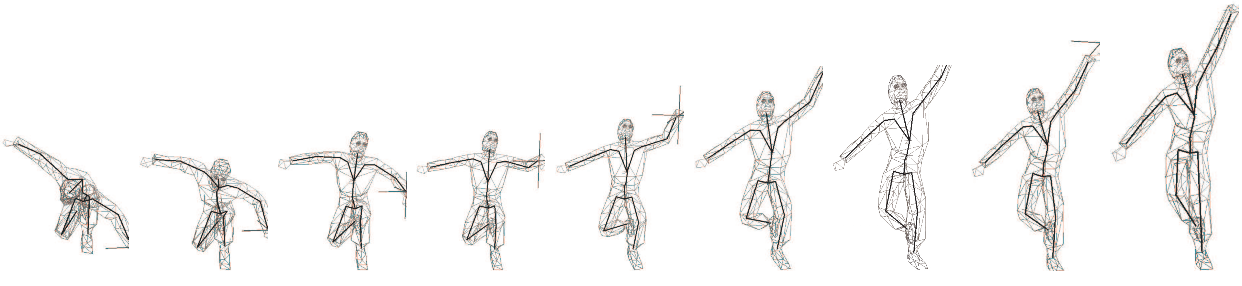

Figure 5 a ***Figure 5 b*** ***Figure 5 c*** ***Figure 5 d*** ***Figure 5 e*** ***Figure 5 f*** ***Figure 5 g*** ***Figure 5 h*** ***Figure 5 i***

Figure 5a-5i:
As in previous picture set, both hands and legs together with the center of mass were constrained to remain at a given place. The character was set to the kneel pose, then the left wrist was moved full range up. Note that during this test the character was interactively controlled by the user (i.e. the character walked, strafed right and left, etc.).