Using a Half-Jacobian for Real-Time Inverse Kinematics

Michael Meredith & Steve Maddock
Department of Computer Science
University of Sheffield
United Kingdom
E-mail: M.Meredith@dcs.shef.ac.uk, S.Maddock@dcs.shef.ac.uk

KEYWORDS
Inverse Kinematics, Computer Character Animation, Real-time

ABSTRACT

Due to their scalability, numerical techniques often form part of an inverse kinematics (IK) solver. However, because of their iterative nature, such methods can be slow. So far research into the field of kinematics has failed to find a general non-numerical solution to the problem. Many researchers have proposed hybrid techniques yet these still rely on a numerical aspect. It is therefore important to find ways of using numerical techniques as efficiently as possible. In this paper we take a look at the Jacobian-based IK solver and techniques that allow this method to be used as an efficient real-time IK solver. We demonstrate how the half-Jacobian can be used effectively where normally the full Jacobian would be considered the principal technique. The result of this is much reduced computational costs when applying IK to articulated characters.

INTRODUCTION

The problem domain that is tackled by inverse kinematics solvers was first formulated in the mechanical engineering literature (Craig 1955) and more specifically research into the field of robotics. We are interested in its application in computer character animation. The issue that inverse kinematics attempts to resolve is to find a set of joint configurations of an articulated structure based upon a desirable end-effector location. This is expressed mathematically in Equation 1.1 where θ represents the set of orientation values for a structure and X is the global position of a given limb in the hierarchy.

$$\theta = f^{-1}(X) \qquad \textbf{(1.1)}$$

There have been many varied techniques used as an inverse kinematics solver. The fastest techniques, analytical algorithms, tend to suffer from poor scalability, whereas the scalable techniques, such as numerical iteration, suffer from poor solver times. Many techniques that have been proposed to offer speed advantages utilise numerical solvers therefore it is important to consider ways that such techniques can be used efficiently. A review of many of the present IK techniques is given in the following section.

We then present an analytical look at the iterative Jacobian approach to inverse kinematics and discuss techniques that allow the method to used effectively. Following this we present a real-time application that drives a walking character around rough terrain to demonstrate the effectiveness of our Jacobian interpretation.

RELATED WORK

We can identify 4 different categories of IK solver: geometric/analytical algorithms, cyclic co-ordinate descent (CCD) techniques, differential techniques, and hybrid methods (Tolani *et al.* 2000) which mix together various aspects of the first three techniques.

The geometric/analytical algorithms (Chin 1996, Kwang-Jin and Hyeong-Seok 2000, Paul and Shimano 1988) tend to be very quick because they reduce the IK problem to a mathematical equation that need only be evaluated in a single step to produce a result. However, for large chains of links the task of reducing the problem to a single-step mathematical equation is impractical. Therefore geometric/analytical techniques tend to be less useful in the field of character animation.

IK solvers that are based on CCD (Eberly 2001, Wang and Chen 1991, Welman 1993) use an iterative approach that takes multiple steps towards a solution. The steps that the solver takes are formed heuristically, therefore this step can be performed relatively quickly. An example of a possible heuristic would be to minimise the angle between pairs of vectors created when projecting lines through the current node and end-effector and current node and desired location. However, because the iterative step is heuristically driven, accuracy is normally the price paid for speed. Another issue with this technique is that only one joint angle is updated at a time, which has the unrealistic result of earlier joints moving much more than later limbs in the IK chain.

As with the CCD technique, differential-based techniques (Watt and Watt 1992, Zhao and Badler 1994) utilise an iterative approach that requires multiple steps to find a solution. The steps that the algorithm makes are determined via the use of the system Jacobian that relates small changes in joint configurations to positional offsets. Since all the joint angles are updated in a single step, the movements are dissipated over the whole chain which results in a more realistic looking posture.

By their nature, iterative-based techniques are generally slower at producing a desirable result when compared to their analytical counterparts. However the problem with the analytical methods is their lack of scalability. Fedor (Fedor 2003) explores this trade-off between speed, accuracy and scalability in an IK solver. One of the results from this work demonstrates that differential-based numerical solutions, although slower than both CCD and analytical techniques, provide better results for larger chains. This highlights the importance of refining numerical techniques such that we maintain accuracy and scalability but drive solution time down.

One such solution proposed by Tang *et al.* (Tang *et al.* 1999) makes use of the SHAKE algorithm (Ryckaert *et al.* 1997) to achieve a fast iterative-based IK solver. This technique treats a hierarchical structure as point masses that are related by system constraints. This is in contrast to the Jacobian-based technique that encapsulates the articulated information and thereby provides us the cohesion between links for free.

In order to achieve a desired end-effector location, the mass points of the SHAKE system are adjusted per cycle until a global goal has been reached. This includes meeting a threshold of acceptable error on the constraints. However because of the lack of node dependency of the algorithm, normally the points will lose their distance relationships between each other. To counter this issue, correcting forces are iteratively applied to each point to reassert cohesion between links therefore the accuracy of parent-child distances directly effects solver time. Without a reasonable level of accuracy at this point, the appearance of rigid links moving about each other would occur. This is an issue that the Jacobian-based techniques are not affected by.

The time complexity of the SHAKE algorithm is suggested by Tang *et a.l* to be $O(n^2)$ with respect to the number of constraints. However since each link in a hierarchical chain requires a constraint to impose cohesion, the time to solve a system is also minimally $O(n^2)$ with respect to the number of links in the chain. The inclusion of additional system constraints such as joint angle limits has a further detrimental effect on solution time therefore making the algorithm less applicable to real-time applications as the number of links increases.

Another real-time IK technique proposed by Shin *et al.* (Shin *et al.* 2001) that is used for computer puppetry makes use of a hybrid solution. This technique attempts to use analytical solutions where possible, except in cases where a large amount of body posturing is required, where a numerical implementation is invoked. The numerical solver only acts upon the IK chain defined between the root and the upper body while the analytical solver is used for the limbs of the character.

The hybrid use of IK solvers used by Shin *et al* demonstrates a good method for performing real-time IK. However the analytical aspect assumes some knowledge about the character's structure (Lee and Shin 1999, Tolani *et al.* 1996). This means that the overall IK technique is not a general one that can be applied to arbitrary IK chains. The other potential problem with the hybrid technique is similar to the CCD techniques in that not all joint angles are updated simultaneously which means unrealistic and unproportional posture configurations could result.

From the research done in the field of real-time IK, it is apparent that analytical solutions by themselves are not scalable enough to meet the demands of modern computer-based IK problems. Therefore numerical techniques are used as either a substitute or in serial with an analytical solution, which serves to highlight the importance of having fast numerical solutions. Furthermore these solutions should operate on the whole hierarchical structure equally to avoid unrealistic postures. These are the issues we address with our Jacobian-based approach for real-time IK.

OUR INVERSE KINEMATICS SOLUTION

Jacobian Inverse Kinematics

Our implementation of inverse kinematics is based upon the well-established Jacobian technique. The objective of this technique is to incrementally change joint orientations from a stable starting position towards a configuration state that will result in the required end-effector being located at the desired position in absolute space. The amount of incremental change on each iteration is defined by the relationship between the partial derivatives of the joint angles, θ, and the difference between the current location of the end effector, X, and the desired position, X_d. The link between these two sets of parameters leads to the system Jacobian, J. This is a matrix that has dimensionality $(m \times n)$ where m is the spatial dimensional of X and n is the size of the joint orientation set, θ. The Jacobian is derived from the equation for forward kinematics, Equation 1.2, as follows:

$$X = f(\theta) \qquad \textbf{(1.2)}$$

Taking partial derivatives of Equation 1.2:

$$dX = J(\theta)d\theta \qquad \textbf{(1.3)}$$

where

$$J_{ij} = \frac{\partial f_j}{\partial x_i} \qquad \textbf{(1.4)}$$

Rewriting Equation 1.3 in a form similar to inverse kinematics (Equation 1.1) results in Equation 1.5. This form of the problem transforms the under-defined system into a linear one that can be solved using iterative steps.

$$d\theta = J^{-1}dX \qquad \textbf{(1.5)}$$

The problem now is that Equation 1.5 requires the inversion of the Jacobian matrix. However because of the under-defined problem that the inverse kinematics technique suffers from, the Jacobian is very rarely square. Therefore, in our implementation we have used the right-hand generalised pseudo-inverse to overcome the non-square matrix problem, as given in equation 1.6.

Generating the pseudo-inverse of the Jacobian in this way can lead to inaccuracies in the resulting inverse that need to be reduced. Any inaccuracies of the inverse Jacobian can be detected by multiplying it with the original Jacobian then subtracting the result from the identity matrix. A magnitude error can be determined by taking the second norm of the resulting matrix multiplied by dX, as outlined in Equation 1.7.

If the error proves too big then dX can be decreased until the error falls within an acceptable limit.

An overview of the algorithm we used to implement an iterative inverse kinematics solution is as follows:

1) *Calculate the difference between the goal position and the actual position of the end-effector:*

$$dX = X_g - X$$

2) *Calculate the Jacobian matrix using the current joint angles: (using Equation 1.4)*
3) *Calculate the pseudo-inverse of the Jacobian:*

$$J^{-1} = J^T (JJ^T)^{-1} \qquad \textbf{\textit{(1.6)}}$$

4) *Determine the error of the pseudo-inverse*

$$error = \left\| (I - JJ^{-1})dX \right\| \qquad \textbf{\textit{(1.7)}}$$

5) *If error > e then*

$$dX = dX / 2$$

restart at step 4

6) *Calculate the updated values for the joint orientations and use these as the new current values:*

$$\theta = \theta + J^{-1}dX$$

7) *Using forward kinematics determine whether the new joint orientations position the end-effector close enough to the desired absolute location. If the solution is adequate then terminate the algorithm otherwise go back to step 1.*

The computational demand of the algorithm is relatively high over a number of iterations, so well-defined character hierarchies are advantageous. This means that each node in the articulation is defined by the minimum number of degrees of freedom (DOF) required thereby making θ as small as possible. For example, pivot joints such as an elbow would only be modelled using a single DOF whereas a ball and socket joint like the shoulder would need 3 Euler DOFs to represent the range of possible movements.

The use of well-defined hierarchies further helps to prevent the inverse kinematics solver from producing unnatural-looking postures. However this still does not cover all of the potential unnatural poses the solver can return. In order to restrict the IK solver to the orientation space of only possible character configurations, joint orientation restrictions can be enforced within the scope of the existing algorithm. The simplest way of incorporating such constraints is to crop the joint angles. This requires Step 6 of the algorithm to be modified in the following way:

6) *Calculate the updated values for the joint orientations and use these as the new current values:*

$$\theta = \begin{cases} lowerbound & if \quad \theta + J^{-1}dX < lowerbound \\ upperbound & if \quad \theta + J^{-1}dX > upperbound \\ \theta + J^{-1}dX & otherwise \end{cases}$$

The time to complete the IK algorithm for a given end-effector is an unknown quantity due to an arbitrary number of iterations required. However the time to complete a single iteration is constant with respect to the dimensionality of X and θ which is unchanged under a complete execution of the algorithm. Therefore by placing an upper limit on the number of iterations we can set a maximum time boundary for the algorithm to return in. If the solver reaches the limit then the algorithm returns the closest result it has seen.

In 3-dimensional space, the dimensionality of X in a Jacobian-based inverse kinematics solver is generally either 3 or 6. The 6-dimensional X vector is normally used as it contains both positional and orientation information whereas a 3-dimensional vector only contains positional information for an end-effector.

From the inverse kinematics algorithm outlined above, it is clear that the 3-dimensional X vector is quicker over its counterpart and should always be used when orientation is not required. However there are times that orientation is required but it is still possible to use the 3-dimensional vector which is demonstrated in our application of the algorithm present later.

To see how much of a cost difference there is between the two sizes of X vector, the corresponding complexity analysis of them is illustrated in the following section.

Complexity Analysis Of The *X* Vector

We need a technique to perform the inverse of the square matrix JJ^T. For the 3-dimensional X vector, which uses a (3 x 3) matrix we use an analytical solution whereas for the 6-dimensional X vector, which uses a (6 x 6) matrix, we use LU Decomposition.

LU Decomposition and Analytical Inversion
LU decomposition can be used to determine the inverse of a square matrix by using the matrix identity, $AA^{-1} = I$, where I is an identity matrix (and in this case it has dimensionality (6 x 6)). The application of LU decomposition to this equation requires matrix A to be split into 2 further matrices that have the form of lower and upper matrices as illustrated in Equation 1.8.

$$A = LU = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ * & 1 & 0 & \cdots & 0 \\ * & * & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ * & * & * & \cdots & 1 \end{bmatrix} \begin{bmatrix} * & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ 0 & 0 & * & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & * \end{bmatrix}$$

(1.8)

The decomposition of A into the two matrices allows the original matrix identity to be rewritten into the form of Equation 1.9, which can be solved using forward and backward substitution.

$$LUA^{-1} = I \quad \Rightarrow L(UA^{-1}) = I$$

$$\Rightarrow LY = I \qquad \textbf{(1.9a)}$$

$$\wedge UA^{-1} = Y \qquad \textbf{(1.9b)}$$

Our algorithm for performing a (6 x 6) LU Decomposition inverse gives a complexity of 619 flops (Meredith and Maddock 2004).

In comparison, the analytical inversion of a (3 x 3) matrix is given in Equation 1.10. This equation can be directly encoded. The complexity of calculation is 51 flops (Meredith and Maddock 2004): 36 multiplications, 1 division and 14 additions & subtractions.

$$\begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix}^{-1} = \frac{\begin{bmatrix} r_{11}r_{22} - r_{21}r_{12} & r_{21}r_{02} - r_{01}r_{22} & r_{01}r_{12} - r_{11}r_{02} \\ r_{20}r_{12} - r_{10}r_{22} & r_{00}r_{22} - r_{20}r_{02} & r_{10}r_{02} - r_{00}r_{12} \\ r_{10}r_{21} - r_{20}r_{11} & r_{20}r_{01} - r_{00}r_{21} & r_{00}r_{11} - r_{10}r_{01} \end{bmatrix}}{r_{00}(r_{11}r_{22} - r_{12}r_{21}) + r_{01}(r_{12}r_{20} - r_{10}r_{22}) + r_{02}(r_{10}r_{21} - r_{11}r_{20})} \quad \mathbf{(1.10)}$$

The decision to use an analytical solver for the smaller matrix and LU decomposition for the larger one is demonstrated in Figure 1.1. The results given in Figure 1.1 were obtained using a matrix with all elements non-zero so the analytical technique was unable to make use of zeros to cut off the co-factor expansions. This is a valid assumption because it would be most unlikely that the (6 x 6) matrix that needs to be inverted in the pseudo-inverse would actually contain any zeros.

Figure 1.1 shows that the analytical approach to solving matrix inversion is only better for matrices that have dimensionality equal to or less than 3. After this size, the number of flops required to solve an analytical inverse increases in a cubic fashion with respect to dimensionality whereas the LU technique increases at the lower squared rate. This analysis justifies the use of an analytical solution for the (3 x 3) matrix while using LU decomposition for the inversion of the larger (6 x 6) matrix.

Calculating The Jacobian

If the Jacobian definition of Equation 1.4 is divided by a differential time element, the resulting equivalence provides a mapping between angular velocities in state space, θ, and linear velocities in Cartesian space, X. This result is illustrated in equation 1.11.

$$\dot{X} = J(\theta)\dot{\theta} \quad \mathbf{(1.11)}$$

In the case of a 6-dimensional X vector, $\dot{X}$ consists of linear velocity, V, and angular velocity, Ω, components, whereas the 3-dimensional X vector only includes the linear velocity. Both the linear velocity and angular velocity are with respect to a global frame of reference as too are the partial derivatives of the Jacobian. The Jacobian linking the linear and angular velocity of the end-effector, with the intermediary local angular velocities, is given in equation 1.12, where there are i DOFs in the IK chain.

$$\begin{bmatrix} V \\ \Omega \end{bmatrix} = \begin{bmatrix} b_1, b_2, \ldots, b_i \\ a_1, a_2, \ldots, a_i \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \vdots \\ \dot{\theta}_i \end{bmatrix} \quad \mathbf{(1.12)}$$

In Equation 1.12, the a components are of the local axes for a given link transformed into the global frame of reference. The b elements of the Jacobian are the cross products of the corresponding a axis with the spatial difference between the global origin of the current limb and the absolute location of the end of the articulation, P_e (Equation 1.14). The DOFs within the state space are normally ordered such that limbs from the root are considered first followed by their children, following this pattern to the end of the chain. Using this pattern, the orientation values of the required axes for each limb can be obtained from a transformation matrix, ${}^{0}T_j$, that converts points defined in the limb's local orientation into a global position. Equation 1.13 illustrates this for the jth limb

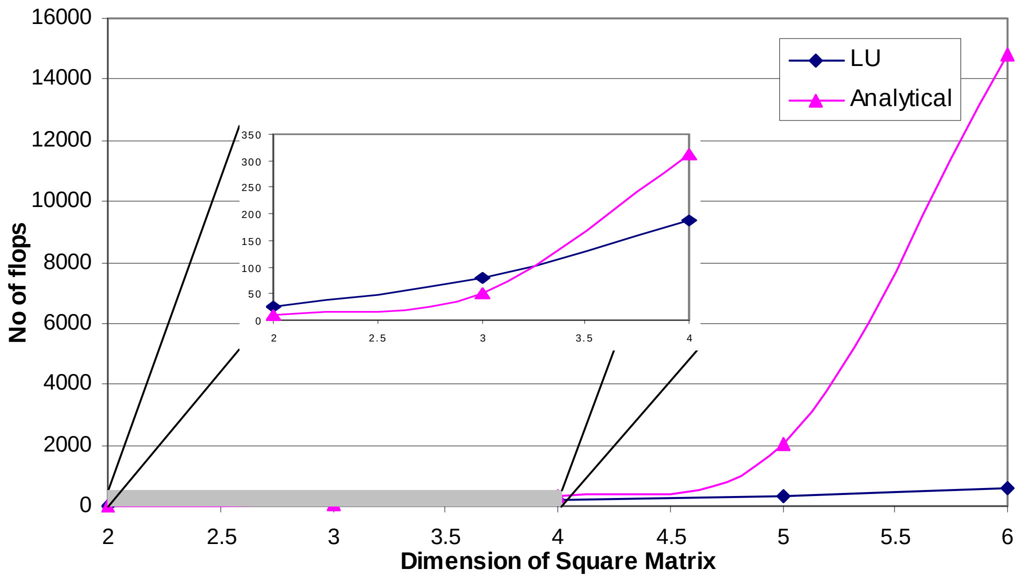

Figure 1.1: Demonstration of the complexity of solving a square matrix using an analytical and LU decomposition technique.

in the IK chain (note that this assumes a right-handed coordinate system):

$$^{0}T_{j} = \begin{bmatrix} a_{xj} & a_{yj} & a_{zj} & P_{j} \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(1.13)}$$

The parameter P_j in Equation 1.13 also gives the global position of the origin of the limb thereby aiding in the determination of the b components in Equation 1.14.

$$b = a_i \times (P_e - P_j) \qquad \textbf{(1.14)}$$

Using the chaining principle for calculating the transforms of the local axes into a global reference frame ($^{0}T_{j} = {}^{0}T_{1} \times {}^{1}T_{2} \times \ldots \times {}^{j-1}T_{j}$.), the direct implementation of this subsection in 3-dimensional space yields a constant complexity. Assuming that each link has 3 DOFs, the complexity associated with each limb in the IK chain is given by 162 flops: 98 multiplications and 64 additions & subtractions. The assumption of 3 DOFs does not add a great deal of complexity if it is an overestimate since each DOF contributes only 9 flops to the overall result (where the 9 flops is the calculation of the cross product).

Determining The Pseudo-Inverse Of The Jacobian

Using the complexity derivation of the inversion of a square matrix from the above sections, the complexity of the pseudo-inverse of the Jacobian, as given in Equation 1.6, can be calculated. Table 1.1 outlines the number of flops required to calculate the pseudo-inverse depending on the size of the *X* vector. The variable *n* is the size of the state space, θ (i.e. the sum of all the links' DOFs). It should also be noted that there is no inclusion of complexity to calculate the transpose of matrices when they are required as this can be handled at no extra cost be simply swapping out indexing parameters.

Complexity Of The Whole IK Solver

The complexity of a single loop of the IK algorithm described above can be derived using the complexity analyses of the smaller parts of the algorithm already determined. This is shown in Table 1.2 where *m* is the number of inner loops executed at stage 5 of our algorithm.

As Table 1.2 illustrates, the use of a 3-dimensional *X* vector appears to be about 2½ times less computationally demanding than its 6-dimensional counterpart. Considering only the major factor of the complexity, which is the size of the state space, *n*, the 3-dimensional *X* vector should be 238.9% quicker than the alternative. However, the complexity of each algorithm is not only dependent on the size of the state space but also on the number of inner loops which are required to make the inversion of the Jacobian stable enough to provide meaningful results. Therefore it needs to be shown that the use of a smaller Jacobian in the 3-dimensional *X* vector case does not adversely affect the pseudo-inverse. This does not appear to be the case as illustrated with the empirical dataset present in the following section.

Size of *Matrix Operation*	(3 x n) 3D *X* Vector	(6 x n) 6D *X* Vector
$A = JJ^T$	18n - 9	72n - 36
$B = A^{-1}$	51	619
$J^T B$	15n	33n
$J^{-1} = J^T (JJ^T)^{-1}$	33n + 42	105n + 583

Table 1.1: Number of flops required to calculate the pseudo-inverse of a non-square matrix.

Since the smaller *X* vector can be shown to be less computationally demanding by a significant factor, it raises the issue of whether the smaller *X* vector can be used even when orientation is important. The following section gives a brief discussion of possible application areas for using the half-Jacobian over the full-Jacobian. Thereafter we illustrate an example for which we have used the half-size Jacobian in an application that would normally be considered a full-sized Jacobian problem domain.

USING THE HALF- OVER THE FULL-JACOBIAN

An obvious application of the half-Jacobian is in applications that do not discriminate against the orientation of the final link in an inverse kinematics chain. In applications of inverse kinematics where the orientation of the end-effector has little consequence, the 3-dimensional *X* vector should always be used to reduce the computation effort required. For example, when configuring a spider's legs using IK, because the spider effectively walks on the tips of its legs, the orientation of this end point is immaterial. Therefore only the 3-dimensional *X* vector would be required. As illustrated in Table 1.2, using the full-sized Jacobian in such cases would be less efficient than the half-sized Jacobian.

Another, more subtle, application of the half-sized *X* vector is

Size of X Vector / ***Algorithm Stage***	3	6
1. Calc. increment	3 flops	6 flops
2. Calc. Jacobian	162 flops	162 flops
3. Calc. Pseudo-Inverse	33n + 42 flops	105n + 583 flops
4. Check for convergence	18n + 15 flops	72n + 66 flops
5. Reduce *dX*	18m flops	72m flops
6. Update joint angles	6n flops	12n flops
7. Calc. new position	38n flops	38n flops
Total	95n + 18m + 252 flops	227n + 72m + 817 flops

Table 1.2: Complexity analysis of our Jacobian based IK solver

in situations where the penultimate link in the IK chain has unlimited and full use of all 3 DOFs (in 3 dimensional space). In this scenario the first step is to calculate the position of the penultimate link based on the desired position and orientation of the final node. The 3-dimensional *X* vector can then be used to position the penultimate node in the chain. Once this is done the desired orientation of the final node can be specified thereby allowing the correct end configuration of the chain.

Other applications where the half-size Jacobian would prove a better technique to employ over the full-size version is in situations of low resolution modelling. For example, if a complex articulated model is being animated as a background entity in a scene, it would be advantageous to switch to the quicker half-Jacobian to solve its configuration. This means that more avatars can be animated in the background of a scene.

There are many other applications where the half-sized Jacobian could substitute for the traditional full-sized version. Currently we have applied the quick half-Jacobian inverse kinematic solver to motion capture retargetting and IK-driven character walking. Both of these applications can easily run in real-time as demonstrated in the following section which describes the latter of our applications.

IK-GENERATED HUMANIOD WALKING

The coupling of a procedural model and an inverse kinematics solver provides the basic building blocks needed to generate the walking motion of a computer character. The procedural model describes the path through which the foot travels during a stride while the IK solver positions (and orientates) the foot along this path over time. The task of tracing the foot along the path would initially appear to require the full-sized Jacobian, inherently requiring the foot to be orientated in a forward facing direction. Without the orientation of the foot taken into account, there are an infinite number of anatomically correct positions the heel could take in order to meet a simple positional constraint. This is possible because the hip joint for a leg can rotate about the axis of the femur approximately ±90 degrees from the forward facing pose, as illustrated in Figure 1.2.

From the evidence of Figure 1.2, it would seem that the full-sized Jacobian is the only choice of IK solver to drive the walking motion of a humanoid character. However, by realising that in the course of a walking motion, any large hip joint rotations result in unnatural postures, additional constraints can be added to restrict movement to only plausible ranges. This would allow the half-sized Jacobian to be used to calculate the position of the heal and thus simultaneously reduce the potential for orientation error and increase the performance of the solver.

This approach has been used in our implementation of an IK-driven humanoid character, *MovingIK* (Meredith and Maddock 2004) which has the ability to walk over uneven terrain in real-time.

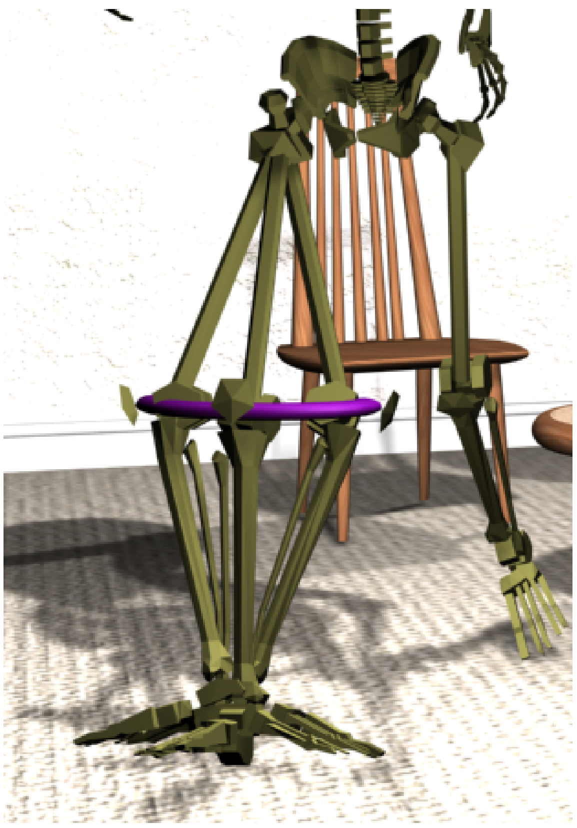
Figure 1.2: Infinite number of positional solutions to fixing a heel plant without regard to the orientation of the foot. The purple ring shows the location of all possible knee positions.

MovingIK, makes use of a procedural stride model to define how the foot moves over time as the character is walking. The source for the procedural stride model in our application comes from a simple mathematical equation whose form is illustrated in Figure 1.3 where a complete cycle ranges between 0 and 3π.

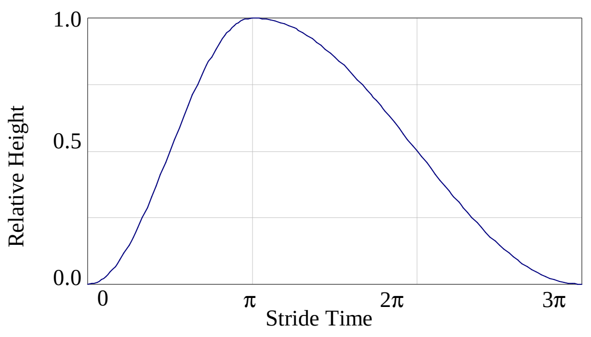

Figure 1.3: Graph of procedural stride used in *MovingIK*

Along with the procedural model used to drive the character's foot through the air, we have a pre-flight stage that rolls the foot from a heel supporting phase to a complete foot supporting phase. This uses the inverse kinematics algorithm to simultaneously plant the heel of the character and gravitate the toes towards the ground. This extra bit of the walking cycle increases the realistic-looking nature of the resulting animation and gives us the ability to model the complete foot as opposed to just the heel.

The character is driven around an uneven terrain in real-time using an analogue joystick that determines parameters such as stride length, stride speed and direction of travel.

Using MovingIK we are able to compare the half-sized and full-sized Jacobian techniques for both performance and realism.

Empirical Results

The results in Table 1.4 are obtained from running *MovingIK* on a Pentium 4 1.4GHz processor with a GeForce2 Ultra. There was a maximum iteration count imposed on the IK solver for the outer loop of 200 cycles while the inner loop was subject to a 20-cycle ceiling. These limits were determined by the empirical running of the IK solver to determine over what limits a solution was very rarely found. The character driven by the user is made up of 18 hierarchical segments where only naturally-occurring DOFs within the human body were permitted. The constraints on each remaining DOF were further limited to joint angles within the scope of normal human movement.

The results given were obtained by driving the character around both flat and uneven terrains. In the case of the uneven terrain, several randomly-generated surfaces were used (including a flight of steps) and the overall results were obtained by averaging the results. Each of the uneven terrains had the same number of vertices and polygons in the model (64,082 polygons compared to 2 polygons for even terrain). The character displayed was that of either a stick figure or a 3D model consisting of 11,101 polygons.

MovingIK was not optimised to use either the half- or full Jacobian but instead provided the ability to switch between the two techniques at run-time. There are three different configurations possible to switch between. The first two modes use only the half- or full Jacobian respectively to calculate the configuration of the character to position the leading foot and trailing toes. The third mode uses a hybrid approach that uses the full Jacobian to determine the configuration of the leading foot and the half-Jacobian to anchor the trailing toes.

The empirical results of driving the computer character within *MovingIK* are illustrated in Table 1.4. It should be noted that during a single frame, *MovingIK* solves two IK chains – one for each leg. An illustration of *MovingIK* is given in Figure 1.4.

The speed-up factor between the full Jacobian and the half-Jacobian, based on the empirical average time per iteration, is 238.5% which when compared to the analytical computed result of 238.9% reinforces the advantages of using the half-Jacobian over the full Jacobian whenever possible.

A further conclusion that can be obtained from these results is that the use of the full Jacobian does not necessarily make the IK solver any more stable. This logical conclusion comes form the fact that the analytical speed-up factor calculated assumes that the inner loop is executed an equal number of times for both algorithms. If this were not the case then the empirical results would show a larger difference in speed up factor due to one algorithm executing the inner loop more times than the other.

CONCLUSIONS & FUTURE WORK

From this analysis of the empirical and analytical results, there is no proven stability advantage from using the full Jacobian compared to that of the half-Jacobian. Therefore there is a definite argument for using the half-sized Jacobian when only the position of an end-effector is needed.

As we have shown, there is also scope for using the quicker half-Jacobian for limited domains when orientation is required as well as position. Although we have only demonstrated this for a walking motion, this represents one of the most fundamental movements in computer character animation. In other work we have also applied this technique to the field of motion capture retargeting with similarly successful results in both speed and visual accuracy. There are many other conceivable domains in which this application can be used by placing extra dynamic constraints on joint angles to prevent the orientation from deviating too much from a natural-looking configuration. An arm, for example, would prove just as suitable a subject for the technique.

IK Mode / ***Measurement***	All Half Jacobian (3D X Vector)	All Full Jacobian (6D X Vector)	Hybrid Method
Flat Floor with Stick Character	260 fps	95 fps	115 fps
Flat Floor with Skeleton	140 fps	69 fps	83 fps
Uneven Terrain with Stick Character	97 fps	54 fps	64 fps
Uneven Terrain with Skeleton Character	75 fps	42 fps	53 fps
Average time to execute each IK solver	0.24 ms	5.5 ms	- - - -
Average Number of iterations	18.15	180	- - - -
Average time per iteration	0.013 ms	0.031 ms	- - - -

Table 1.4: Empirical Results from *MovingIK*

The advantages of using dynamic constraints to transform an orientation and positional IK problem into a position-only task are a speed-up factor of about 238%. There is also no extra cost to adding in constraints to the half-sized Jacobian algorithm because its framework already operates using joint restrictions. Effectively you get the dynamic constraints for free in the Jacobian-based IK solver.

We have already integrated our quick real-time inverse kinematics solver into a motion capture retargeting application where the next step will be to use the solver to simultaneously individualise the character. For this we are looking into the application of weighted IK chains such that different parts of the articulation change with a varying rate to the others. This would give rise to the very simple and quick production of injuries or even varying character builds in computer figures.

REFERENCES

Chin, K.W., "Closed-form and generalized inverse kinematic solutions for animating the human articulated structure.", *Bachelor's Thesis in Computer Science, Curtin University of Technology*, 1996

Craig, J. J., "Introduction to Robotics: Mechanics and Control", *Addison-Wesley*, 1955

Eberly, D. H., "3D Game Engine Design", *Morgan Kaufmann*, 2001

Fedor, M., "Application of Inverse Kinematics for Skeleton Manipulation in Real-time", *International Conference on Computer Graphics and Interactive Techniques*, p.203-212, 2003

Kwang-Jin, C., Hyeong-Seok, K., "On-line Motion Retargetting", *The Journal of Visualization and Computer Animation*, Vol. 11, p.223-235, 2000

Lee, J., Shin, S. Y., "A Hierarchical Approach to Interactive Motion Editing for Human-Like Figures", *Siggraph 99*, p.39-48, 1999

Meredith, M., Maddock, S., "Real-Time Inverse Kinematics: The Return of the Jacobian", Technical Report No. CS-04-06, *Department of Computer Science, The University of Sheffield*, 2004

Paul, R. P., Shimano, B., Mayer, G. E., "Kinematic Control Equations for Simple Manipulators", *IEEE Transactions on System, Man & Cybernetics*, Vol. 11, No. 6, 1988

Ryckaert, J. P., Ciccotti, G., Berendsen, H. J. C., "Numerical Integration of the Cartesian Equations of Motions of a System with Constraints: Molecular Dynamics of n-Alkanes", *Journal of Computational Physics*, Vol. 23 p.327-341, 1977

Shin, H. J., Lee, J., Gleicher, M., Shin, S. Y., "Computer Puppetry: An Importance-Based Approach", *ACM Transactions On Graphics*, Vol. 20, No. 2, p.67-94, April 2001

Tang, W., Cavazza, M., Mountain, D., Earnshaw, R., "A Constrained Inverse Kinematics Technique for Real-time Motion Capture Animation", *The Visual Computer*, Vol. 15, p.413-425, 1999

Tolani, D., Badler, N. I., "Real-time Inverse Kinematics of the Human Arm", *Presence*, Vol. 5, No. 4, p.393-401, 1996

Tolani, D., Goswami, A., Badler, N., "Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs", *Graphics Models*, Vol. 62, No. 6, p.353-388, 2000

Wang, L., Chen, C., "A Combined Optimisation Method for Solving the Inverse Kinematics Problem of Mechanical Manipulators", *IEEE Transactions on Robotics & Applications*, Vol. 7, No. 4, p.489-499, 1991

Welman, C., "Inverse kinematics and geometric constraints for articulated figure manipulation", Master of Science Thesis, *School of Computing Science, Simon Fraser University*, 1993

Watt, A., Watt, M., "Advanced animation and rendering techniques", *Addison-Wesley*, 1992

Zhao, J., Badler, N. I., "Inverse Kinematics Positioning Using Nonlinear Programming for Highly Articulated Figures", *ACM Transactions on Graphics*, Vol. 13, No. 4, p.313-336, 1994

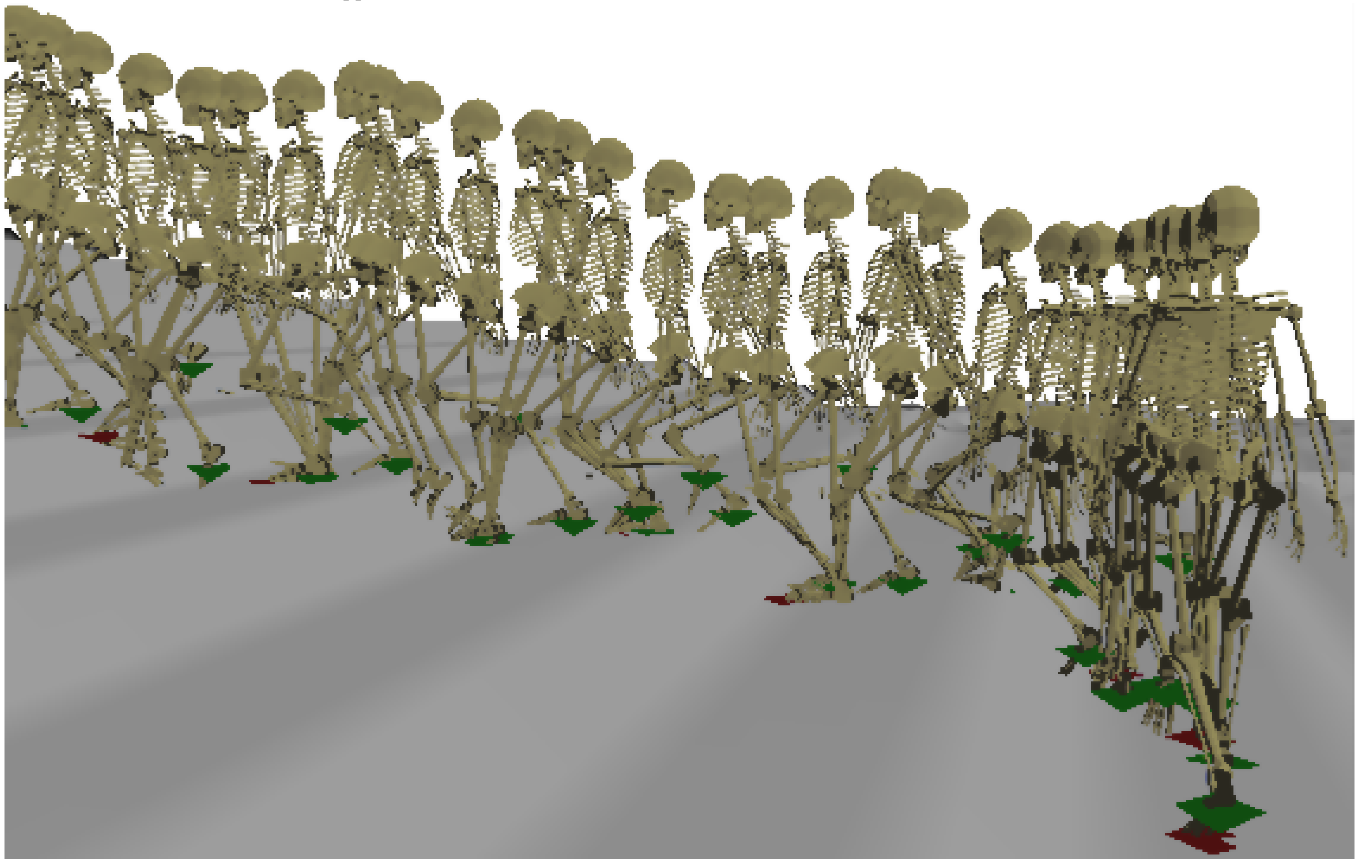

Figure 1.4: Analogue joystick-controlled real-time IK over uneven terrain; green pyramids represent the intended position of the leading foot while the red pyramids indicate desired location of the training toes.