Inverse Kinematics

Article · February 2004

CITATION
1

READS
80

1 author:

Diego Park
Universidad de Buenos Aires
6 PUBLICATIONS **1** CITATION

SEE PROFILE

Inverse Kinematics

Diego Park

Computer Graphics, Department of Computer Science, University of Buenos Aires, Argentina

1. Introduction

In general, Kinematics studies the motion without considerations of the forces that produce it, like the flexion of a muscle. To understand different types of Kinematics, it is necessary to describe Forward and Inverse Kinematics.

As its name indicates, Forward Kinematics (FK) refers to the direct manipulation of the structure through rotations and translations until it reaches the desired final position. Its hierarchy consists in the assignation of an object as father of another. Say, the structure is basically transformed in a top-down transversal until the wanted effect is reached. FK is conceptually very easy to understand and it is many times very useful.

However, the main disadvantage of FK is what happens when, for instance, it is desired that the hip rotated but not the feet [Pit02]. In general, there are two solutions for it, although none of them visually satisfactory. First, the whole structure can be rotated, which means not only the hip is rotated but also the feet. The other possibility is that only the node of highest hierarchy (hip for instance) turned, leaving the other nodes without transformation. This produces even visually worse results because it can lead to separate knees from the rest of the body, which is of course unacceptable. Beyond this problem, some movements are hard to describe using FK, like the trajectory of a foot.

Inverse Kinematics (IK) solves the problem of the final position not through direct manipulation of the structure, but through the articulation of the structure necessary to reach the goal. For some time linked to robotics and to off-line animations packages later, the power of today's machines plus the development of different methods allows us to think of IK used in real-time. The most recommendable reference is found in [Wel93]. [Ebe01] and particularly [WP02] are good texts.

Automate robots of, for instance, an assembly-line in a factory, use the angles formed by their joints to reach the desired point (for example, certain part of a car) [Lan98]. The node that is wanted to reach certain point is called *end-effector*, for instance, a mechanical hand in the example. The first node in a parent-child chain (*manipulator*) is referred as *base*.

IK describes its hierarchy like an inverted tree, where a father is called *root* or, usually, son of the world. For example, the hip (normally chosen as root because it represents the center of gravity of the character) can have nodes called "left hip" and "right hip" as children. This introduces the concept of *skeleton*. A skeleton can be manipulated in a top-down approach using FK. However, the main advantage of the use of skeletons is the fact it enables to transfer the control of the hierarchy in a bottom-up direction. The use of IK considerably reduces the time consumed to generate, for instance, animations compared to FK.

As seen, the advantage of IK is that transformations in the end-effector produce changes in the rest of the nodes without necessity that the user had to manually calculate their transformations. For example, if it is wanted that a character reached an object, instead of calculating each transformation, the user can indicate the goal and the rest of the structure will be re-located in a way that the end-effector reached this point.

2. Solutions

Although there can be found more approaches, just the most significative are discussed in this text.

2.1. Analytic solution

One possible solution is to use the *Closed Form* or *Analytical Solution*. In this approximation, given a point in the space, the angles necessary to reach the target can be calculated *analytically*. The main advantages are that solutions can be quickly found

and they are exact, as it is possible to know if a point is reachable or not. A disadvantage consists in that, because it involves an exact solution, if the goal is not reachable, the system does not even try to approximate to the solution. This decision is not always desired, since it can be visually more interesting that an end-effector moved towards the goal, instead of staying quiet. Also, there is not a practical solution if the number of degrees of freedom (DOF) is increased, because it introduces a complexity that the system may not be able to solve. The analytic solution of IK [WP02] for a two-linked arm is:

$$\begin{aligned} \Theta_2 &= \cos^{-1}\left(\frac{x^2+y^2-l_1^2-l_2^2}{2l_1l_2}\right) \\ \Theta_1 &= \tan^{-1}\left(\frac{-l_2\sin\Theta_2 x+(l_1+l_2\cos\Theta_2)y}{l_2\sin\Theta_2 y+(l_1+l_2\cos\Theta_2)x}\right) \end{aligned} \tag{1}$$

where (x,y) is the position of the goal, l_1 y l_2 are the length of the two segments and Θ_1 y Θ_2 are the angels of the nodes.

2.2. Cyclic coordinate descent (CCD)

As described in [WP02], this method heuristically moves towards a solution through small steps, using CCD (a standard non-linear programming tool).

This algorithm attempts to minimize the error transforming each joint at a time and it is shown to be fast. However, its main drawback is that it tends to priviledge the nodes near the end-effector. This can lead to artifacts or non-natural poses.

2.3. Differential methods using the matrix Jacobian

Considering that the position of the end-effector can be written as:

$$P(x,y) = f(\overrightarrow{\Theta}) \tag{2}$$

the problem of IK has to solve:

$$\overrightarrow{\Theta} = f^{-1}(P(x,y)) \tag{3}$$

Because of the non-linear nature of the problem, it cannot be solved directly. The problem is then linearized as follows:

$$\Delta x = J\Delta\Theta \tag{4}$$

where J is obviously the Jacobian matrix of differentials. Therefore, it tries, through *small* changes in the vector of angles $\overrightarrow{\Theta}$ of the manipulator, to introduce small changes in the position of the end-effector to finally reach the desired position:

$$\Delta\Theta = J^{-1}\Delta x \tag{5}$$

Because the Jacobian matrix is usually non-square, its inverse is not defined. However, its pseudoinverse can be calculated instead:

$$J^+ = J^T(JJ^T)^{-1} \tag{6}$$

where

$$J^+J = I \tag{7}$$

The algorithm can be applied iteratively to move (in the best case) the end-effector exactly to position of the goal:

While G is not reached and iterations<MAX_ITERATIONS:

- $\Delta x = G - P$
- Calculate J
- $J^+ = J^T(JJ^T)^{-1}$
- Subdivide Δx while $\|(I-JJ^+)\Delta x\| > \Sigma$
- $\Theta_t = \Theta_{t-1} + J^+\Delta x$

where J is the Jacobian matrix with the rotations of the bones of the manipulator, P the position of the end-effector, G the position of the goal and Σ an arbitrary error. Δx is subdivided to obviously reduce the tracking error [WP02].

3. Implementation

3.1. IK algorithm

Although angular constraints were not developed for the structure, restrictions of rotation about axis were included. Say, every node can rotate about X, Y and/or Z-axis.

For this purpose, the Jacobian is constructed as a matrix of $3 \times n$, where n is three times the length of the manipulator.

If $\overrightarrow{J_{3i}}$ is called the $3i^{th}$ column, with $0 \leq i < m$, (where m is the length of the manipulator) then $\overrightarrow{J_{3i}}$ indicates the possibility of rotation of the bone i of the manipulator about the X-axis, $\overrightarrow{J_{3i+1}}$ about Y-axis and $\overrightarrow{J_{3i+2}}$ about the axis Z. This way, if the first bone of the manipulator cannot rotate about the X-axis, the matrix J has its first column set to null:

$$\begin{pmatrix} \overrightarrow{0}^T & J' \end{pmatrix} \tag{8}$$

It was decided not to apply the IK algorithm when the goal is not within the range of the end-effector. Say, the length of the current manipulator (given by the base and the end-effector), is calculated. This value then acts like a radius of a sphere that represents the reachable range. This feature was thought to avoid *unnecessary* use of CPU, although it is also true that in many cases it would be preferable to obtain a (*bad*) approximation.

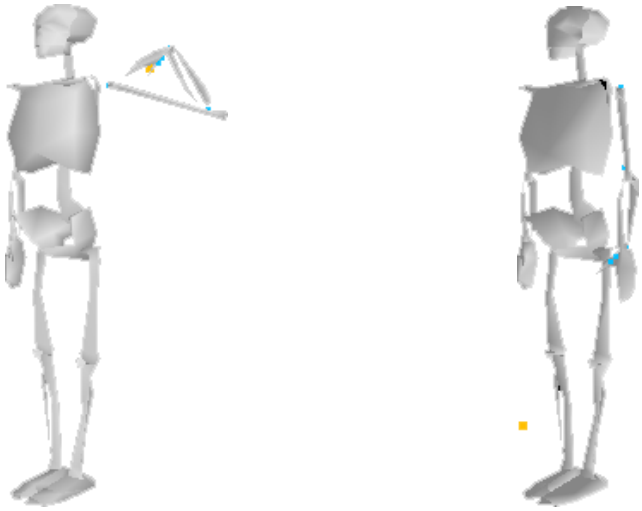

Figure 1: *The figure in the left shows a reachable goal (orange point). The light blue points are the nodes of the manipulator. The figure in the right shows a non-reachable goal. Therefore, the IK algorithm is not applied and the pose is that of the current animation.*

The interaction between the hierarchy of the model and the data created by IK was optimized to use the algorithm in specific situations. The results (transformations) are not stored between function calls, and the algorithm is applied from the current sequence. This approximation can be used to invoke IK to correct postures in real-time. Say, from generic animations created off-line, IK can be used to correct the pose of the model during the simulation. For instance, even if the model contains a (generic) animation sequence to hold certain objects, there can be artifacts because this pre-scripted pose can appear non-natural. Then, IK can be used to correct the posture in real-time.

3.2. Model

A converter of Half-Life's SMD reference files was developed. There are basically two variants of SMD: reference and animations [Val00]. The first provides information of the hierarchy and the geometry. The animation files, in the other hand, holds data about hierarchy (already included in reference files) but more important, about transformations for each bone along certain number of frames.

The displacements and rotations of each bone of the SMD file are indicated in their parent's local space.

The geometry of the SMD files indicate, for each vertex, the position and normals in world space, as the texture coordinates. The current implementation ignores these coordinates, because for the moment it does not support textures.

To represent the hierarchy of the skeleton and the geometry alike, an importer of SMD reference file was developed.

Because the geometry is represented in world space, it is necessary to convert them into the local space of the bones they belong to:

$$T_{relative} = R_i^{-1}(v - T_i) \tag{9}$$

where $T_{relative}$ indicates the translation of the vertex relative to the bone i, R_i^{-1} is the inverse of the (global) pure rotation matrix of the bone i, v is the vertex in world space and T_i, (global) translation or position of the bone i. However, because the pure rotation matrix is orthogonal, it yields:

$$R_i^{-1} = R_i^t \tag{10}$$

say, the inverse is equal to the transpose.

Although every vertex of the SMD format is rigid (i.e. they are associated to a single bone), the implementation enables to attach each vertex to a higher number of bones.

To calculate the position of a vertex attached to an arbitrary number of bones, it can be done as described in [WP02]:

$$\sum_{i=0}^{n} M_i d_i w_i \tag{11}$$

where M_i is the global transformation matrix of the bone i, d_i is the displacement or offset from the vertex to the bone i and w_i is the weight or influence of the attachment. One should note that the following restriction should be respected:

$$\sum_{i=0}^{n} w_i = 1.0 \tag{12}$$

3.3. Structures

In general terms, dynamic memory was avoided as much as possible, because of the overhead caused by the calculation of effective addresses of the pointers and the time required by the operating system to allocate memory. Therefore, following the recommendations found in [Int03], static arrays of an arbitrary length plus indices containing the number of valid elements were used.

Global variables were also frequently used to avoid the use of pointers. Say, one alternative would be to create a class with non-static member with a kind of (global) pointer holding the address of an instance of this class. As it can be inferred, every access to a method or member requires solving an indirect address (pointer) and for this reason the design attempted to avoid this overhead.

3.3.1. Model file format

The native file format for character of the first generation (*G1*) of the implementation, *.character*, is:

ID	'G'
VERSION	1
numbones	Number of bones
skeleton[*numbones*]	Array of *bone_t*
numanimations	Number of animations
animations[*numanimations*]	Array of *sequence_t*
keyframes[*numbones* $*$ *numanimations*]	Array of *spatial_t*
numvertices	Number of vertices
weights[*numvertices*]	Array of *weight_t*
normals[*numvertices*]	Array of normals
texture[*numvertices*]	Array of texture coordinates

3.3.2. Bone

The structure of *bone_t* contains information concerning to the hierarchy of the skeleton. Although this version does not support angular constraints for bones, the structure was designed to easily import biomechanical data about the minimum and maximum rotations in the future.

$$\begin{cases} dof[6] & \text{Array of reals} \\ parent & \text{Bone's parent} \\ options & \text{Combination of } X\_ROTATION, Y\_ROTATION \text{ and } Z\_ROTATION \end{cases}$$

It was also assumed that the bones cannot translate, which means that the degrees of freedom (DOF) are reduced to three: (minimum and maximum) rotations about X, Y and/or Z-axis. Once data of SMD files is loaded, the options of every bone (except the root) is initialized to allow rotation about the three axis.

3.3.3. Sequence

The structure *sequence_t* contains the following information:

$$\begin{cases} fps & \text{Frames per second of the sequence} \\ index & \text{Position of the sequence in the array } keyframes \text{ of } character\_t \\ numframes & \text{Number of frames of the sequence} \end{cases}$$

3.3.4. Transformations

The module *spatial_t* can be described as a pure rotation matrix of 3×3 and a position vector. Although these two components can be combined into a single homogeneous transformation matrix, the use of matrices of 4×4 was avoided because of the both spatial and computational inefficiency because of the redundancy of the last row $(0,0,0,1)$.

$$\begin{cases} rotation & \text{Pure rotation matrix of } 3 \times 3 \\ position & \text{Position vector} \end{cases}$$

3.3.5. Skinning

The structure *weight_t* has an arbitrary amount of attachments and each position of the arrays *offset*, *weights* and *bone* refers to the i^{th} attachment.

$$\begin{cases} offset[MAX\_WEIGHTS] & \text{Array of displacements to the } i^{th} \text{ bone} \\ weights[MAX\_WEIGHTS] & \text{Array of weighting factors} \\ bone[MAX\_WEIGHTS] & \text{Arrays of bones} \end{cases}$$

3.4. Memory management

When it was required to work with dynamic memory, a basic memory manager was implemented, similar to video card's. Certain amount of memory is initially reserved and every request is taken from these positions of memory. Although its main drawback is that objects should be explicitly initialized and destroyed (the *new* and *delete* operators are by-passed), the advantages are its efficiency (it only takes an index update), non-segmentation due to successive request of memory and a better use of cache lines (since the data is stored in adjacent positions). A relatively new feature is also included: the manager allows temporary allocations through *push()* and *pop()*. Thought to be used within a function scope, it provides an efficient way to allocate temporary memory. This feature is typically exploited when accesses to hard-disc are required. Memory to contain the whole file is reserved and the content of this file is transferred to the temporary memory. Thus, expensive accesses to disc are avoided.

3.5. Input

Because this project was designed to do the calculations in real-time, efficiency is obviously crucial. For this reason, DirectInput is used, a component of DirectX that allows direct access to devices like mouse or keyboard. Although the COM interface is not the most efficient, its performance is better than the traditional API of the operating system.

3.6. Multimedia

Different modules were developed to play multimedia to benchmark the IK algorithm in environments where the CPU was required by other processes and where there were also frequent accesses to disc or CD. This situation is not abnormal in certain simulations like 3D games. Music and videos are examples of media-streaming, which means that the file not always can be thoroughly stored in main memory and therefore there are frequent accesses to secondary memory. CD audio requires frequent accesses to CD-ROM drive too.

The IK algorithm shows to be efficient enough to be applied in real-time, even in low-end personal computers.

A short video (or cinematic) is shown at start-up.

3.7. BVH format

Instead of using SMD files, another option would be to import BVH (Biovision Hierarchical Data) files. This format is commonly used to represent data taken from Motion Capture (MoCap) [WP02] [MM]. Even if this format includes the hierarchy of the skeleton, its principal drawback is that it does not contain information about geometry.

4. Instructions

For the moment, the applications run only under Microsoft Windows.

OpenGL is used to render the model. If required features are also detected, the project uses the extensions of OpenGL for vertex buffers [VBO02], which is potentially less time-consuming because the data may be stored in video card's local memory.

The system requirements are:

- OpenGL 1.1
- DirectX 8.0

To compile the project, the user also needs:

- Compilator for Microsoft Windows
- DirectX SDK 8.0

Because the models of this project do not preserve bones' names and only works with indices, it is recommendable to execute MilkShape 3D with the original SMD file to see the correspondence between bone index and name.

4.1. Configuration

This program should be executed at least once to create the necessary entries in the Windows Registry. However, it is strongly recommended to open it every time.

The configurator, which is also a launcher of the main application, automatically selects the best option: safe mode or normal. The safe mode is designed to allow OpenGL to be executed even by software, because it invokes only OpenGL 1.1 functions. If the system supports OpenGL extensions for vertex buffer [HA01] [ATI02], the configurator loads *ik.exe*. Although it may be better to dynamically load DLLs according to the existence or not of these extensions, this idea was discarded because the executables do not occupy big space and because calls to DLL functions are slower.

The required extensions to run in normal mode are:

- *glDrawRangeElements()*
- *wglChoosePixelFormatARB()*
- *glBindBufferARB()*
- *glBufferDataARB()*

If any of these were no found, the configurator launches the safe mode.

To uninstall the program, the following Registry key should be deleted:

```
HKEY_CURRENT_USER\Software\G1
```

The user can select the screen resolution (at least $800 \times 600 \times 32$ is recommended) and set the mouse sensitivity.

The program also looks for MP3 files within the current directory, and automatically creates a play list.

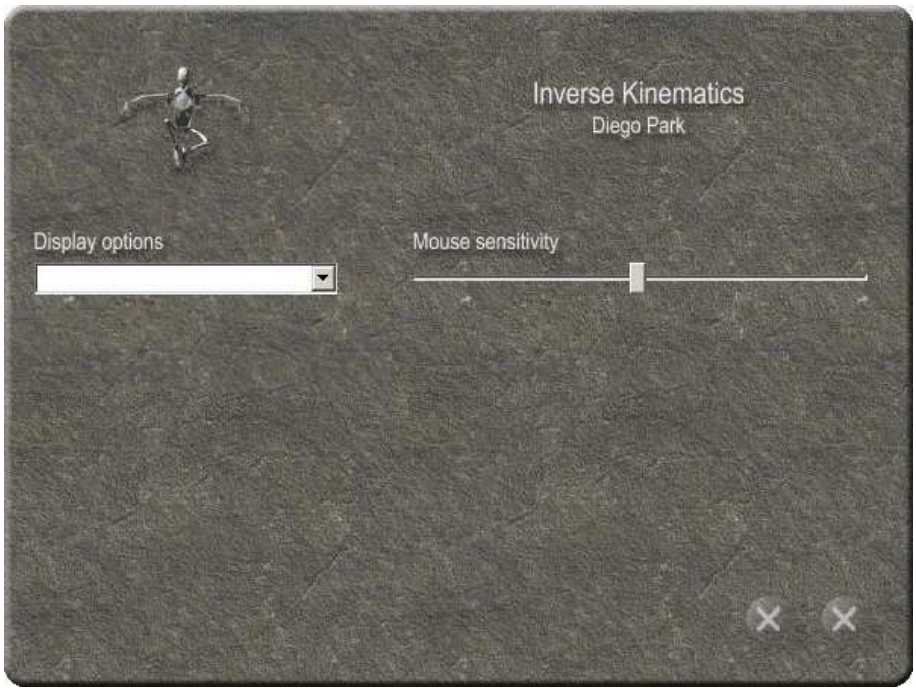

Figure 2: *Image of the configurator.*

4.2. Main application

The current implementation does not have a graphical interface to change the options in real-time.

The mouse and the W and S keys control the goal point, shown as an orange dot.

The program runs in fullscreen mode by default and recreates the 16:9 widescreen effect (although not visible because of the black background).

To quit the application, the ESCAPE key should be pressed.

4.3. Utilities

They are the applications located in the folder */tools*. To quit the programs, the ESCAPE key should be pressed.

4.3.1. Regions

This application creates regions for Windows's windows. To create a region, the user should run the program and then drag a BMP file into the window. A new *.region* file is created in the same folder of the bitmap. The transparent color is (255,0,255).

The application was initially created to generate the region used in the configurator.

4.3.2. SMD converter

Once this program is opened, only SMD reference files will create a valid output (*.character* file) to be used with the project.

5. Discussion

The current implementation can be improved to include angular constraints. This way, more realistic poses can be achieved. A system to avoid self-collisions would also be required (to avoid, for instance, a hand placed *inside* the chest).

The instructions of the function *frame()* of the module *character_t* were isolated so they can be eventually processed directly

in the GPU using vertex shaders [HGI01]. Luckily, programming the video card is not more difficult than programming C now, because both DirectX and OpenGL [KBR03] [LK03] provide high-level languages, although at different stages. High-Level Shading Language (HLSL) [Mic02] of Microsoft DirectX is already implemented, while OpenGL Shading Language (*glslang*) is still being reviewed (however, some video cards of 3DLabs y ATI do support *glslang* programming). Obviously, the implementation would be benefited by the use of vertex shaders, thus partially freeing processor's work.

Although this version does not support textures, the use of them would noticeably enhance the visual appearance. Even more, using normal maps a higher realism would be achieved without necessity of increasing the number of polygons.

The generation of SIMD instructions for the CPU would also improve the efficiency of the algorithm, because it constantly executes vectorial operations. And with the imminent diffusion of 64-bit personal computers (possibly through the AMD's Athlon64 architecture), the algorithm would be enormously benefited because the possibility of even more parallelism.

It is also a somewhat unfortunate situation the fact that it did not exist higher support and implementation of OpenML [RBBC01]. It would be expected that this interface allowed an efficient and simple way to play-back media (possibly faster than DirectShow).

6. Naming convention

To avoid conflicts with another libraries, the namespace *ik* was used.

In general, names in lower cases were used for variables, functions and classes alike. For the variables, whenever it was clear enough, a single word resuming its function was used. In the case that more than a word was needed, the underscore (_) was attached to separate them.

The suffix _t for the classes means that it is a template. In most cases, these classes were created to contain float or doubles.

7. Conclusions

The problem of finding a transformation of a manipulator for the end-effector to reach certain position can be solved using IK. In particular, the differential method using the Jacobian matrix was explored. This algorithm should not have any problem to be applied in real-time.

Normally, a higher number of iterations of the algorithm leads to a better solution, although this may attempt against its velocity.

This version does not implement angular constraints for the nodes, although their incorporation in the future should not be complicated. Even if the existence of angular constraints would obviously mean higher complexity, it would still be viable to apply the algorithm in real-time. Say, the efficiency of the current algorithm should still enable the incorporation of angular constraints at interactive times.

Potentially, the algorithm can be applied during a simulation in special situations; for instance, it can be used to correct poses in real-time, using a generic and pre-scripted pose as an initial solution (which in practice should be a *good* approximation).

8. Acknowledgements

The author would like to thank Valve Software for their character.

References

[ATI02] ATI TECHNOLOGIES INC.: *ATI OpenGL Extension Support*, 2002.

[Ebe01] EBERLY D. H.: *3D game engine: a practical approach to real-time computer graphics*. Morgan Kaufmann Publishers, 2001.

[HA01] HAWKINS K., ASTLE D.: *OpenGL Game Programming*. Prima Tech, 2001.

[HGI01] HART E., GOSSELIN D., ISIDORO J.: Vertex shading with direct3d and opengl. 3D Application Research Group, ATI Research, Inc.

[Int03] INTEL CORPORATION: *IA-32 Intel Architecture Optimization Reference Manual*, 2003.

[KBR03] KESSENICH J., BALDWIN D., ROST R.: *The OpenGL Shading Language*. 3DLabs, 2003.

[Lan98] LANDERS J.: Oh my god, i inverted kine. *Game Developer Magazine* (1998).

[LK03] LICEA-KANE B.: Gl2 shading language. ATI Research, Inc.

[Mic02] MICROSOFT CORPORATION: *DirectX 9.0 Programmer's Reference*, microsoft directx 9.0 sdk ed., 2002.

[MM] MEREDITH M., MADDOCK S.: Motion capture file formats explained. *Department of Computer Science, University of Sheffield*.

[Pit02] PITZEL S.: Character animation: Skeletons and inverse kinematics. *Intel Corporation* (2002).

[RBBC01] ROST R., BELLEY B., BERNARD F., CLIFFORD B.: Media-rich programming with openml. SIGGRAPH 2001.

[Val00] VALVE SOFTWARE: *Modeling and Animating for Half-Life*, half-life sdk pro 2.0 ed., 2000.

[VBO02] Using vertex buffer objects (vbos). *NVIDIA Corporation* (2002).

[Wel93] WELMAN C.: *Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation*. Master's thesis, Simon Fraser University, September, 1993.

[WP02] WATT A., POLICARPO F.: *3D Games: Animation and Advanced Real-time Rendering*, vol. 2. Harlow: Addison-Wesley, 2002.