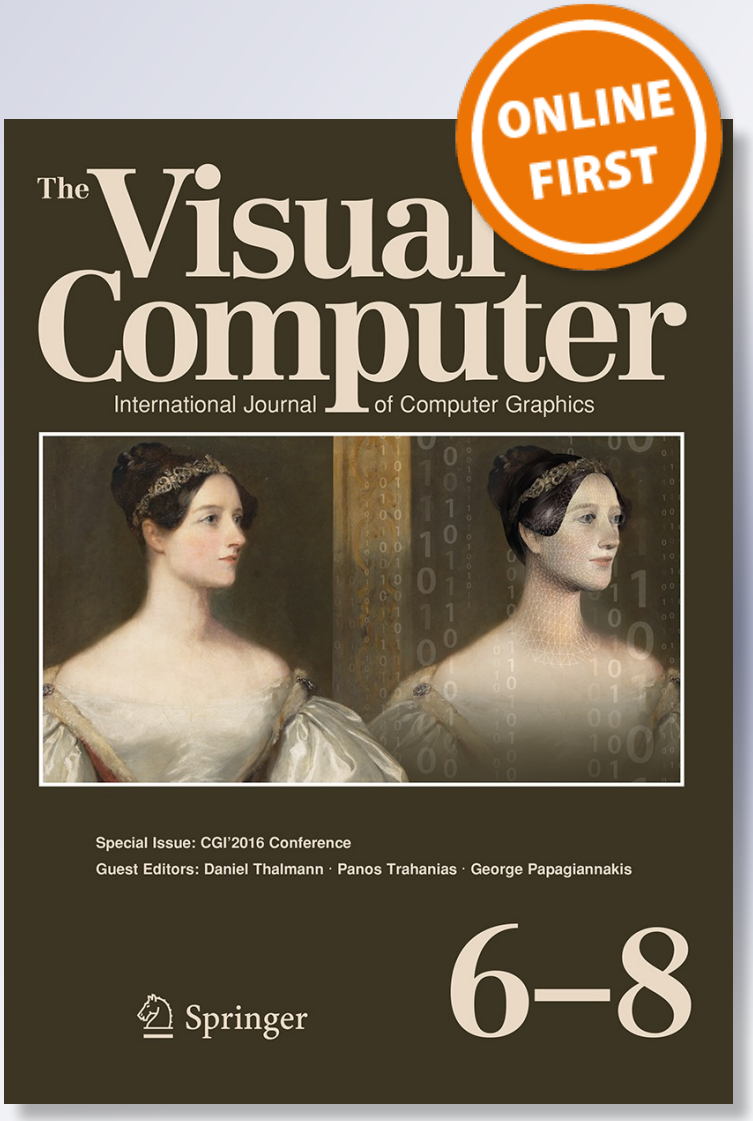
Inverse kinematics using dynamic joint parameters: inverse kinematics animation synthesis learnt from sub-divided motion micro-segments

J. Huang, M. Fratarcangeli, Y. Ding & C. Pelachaud

The Visual Computer
International Journal of Computer Graphics

ISSN 0178-2789

Vis Comput
DOI 10.1007/s00371-016-1297-x



Springer

Vis Comput
DOI 10.1007/s00371-016-1297-x

ORIGINAL ARTICLE

Inverse kinematics using dynamic joint parameters: inverse kinematics animation synthesis learnt from sub-divided motion micro-segments

J. Huang[1] · **M. Fratarcangeli**[2] · **Y. Ding**[1] · **C. Pelachaud**[1]

Abstract In this paper, we describe a novel parallelizable method for the fast computation of inverse kinematics (IK) animation. The existing IK techniques are based on sequential algorithms as they compute a skeletal pose relying only on the previous one; However, for a given trajectory, both the previous posture and the following posture are desired to compute a natural posture of the current frame. Moreover, they do not take into account that the skeletal joint limits vary with temporal spatial skeleton configurations. In this paper, we describe a novel extension of IK model using dynamic joint parameters to overcome the major drawbacks of traditional approaches of IK. Our constraint model relies on motion capture data of human motion. The dynamic joint motion parameters are learned automatically, embedding dynamic joint limit values and feasible poses. The joint information is stored in an octree which clusters and provides fast access to the data. Where the trajectory of the end-effector is provided in the input or the target positions data are sent by data stream, all the computed poses are assembled into a smooth animation sequence using parallel filtering and retargeting passes. The main benefits of our approach are dual: first, the joint constraints are dynamic (as in a real human body), and automatically learnt from real data; second, the processing time is reduced significantly due to the parallel algorithm. After describing our model, we demonstrate its efficiency and robustness, and show that it can generate high visual quality motion and natural looking postures with a significant performance improvement.

Keywords Animation · Inverse kinematics · Octree · Parallel processing

1 Introduction

Inverse kinematics (IK) uses kinematics equations to determine joints' configuration that satisfies a desired position of an end-effector. The joints can be constrained or non-constrained. IK is widely used in robotics and computer graphics, such as motion planning and character animation. Compared to approaches reproducing animation by motion capture (mocap) data, IK offers several advantages, including free of recording sessions, less memory space, etc. For real-time interaction and motion planning, IK is much more flexible than mocap methods (Fig. 1).

Different IK methods [4,10,22,29,34,37] have been proposed over the last 30 years. Speed performance, precision and stability of algorithms have been largely improved by various solutions [3,16,25,28,30,36].

In the past years, several algorithms have been developed to solve more specific problems as described below. Some approaches were proposed to define IK constraints and solve them with different priority levels [6,8,20,26,33,35,35]; other works are geared towards efficient modeling of natural

Electronic supplementary material The online version of this article (doi:10.1007/s00371-016-1297-x) contains supplementary material, which is available to authorized users.

✉ J. Huang
jing.huang@telecom-paristech.fr;
gabriel.jing.huang@gmail.com

M. Fratarcangeli
marcof@chalmers.se

Y. Ding
yu.ding@telecom-paristech.fr

C. Pelachaud
catherine.pelachaud@telecom-paristech.fr

[1] CNRS LTCI and Telecom ParisTech, Paris, France

[2] Chalmers University of Technology, Göteborg, Sweden

Published online: 04 August 2016

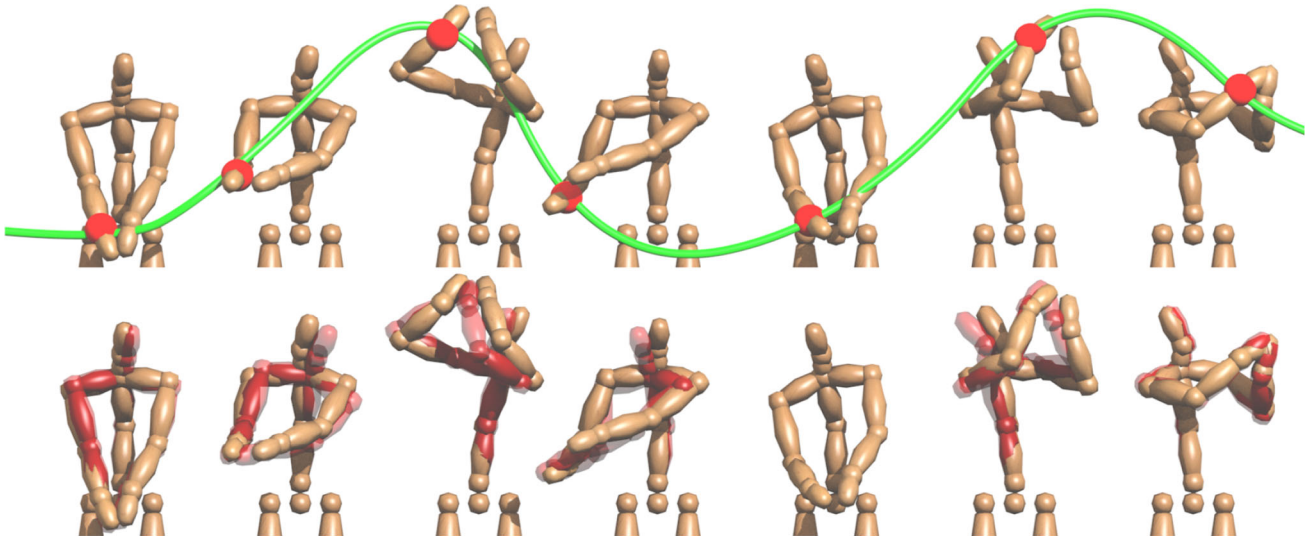


Fig. 1 Golf swing sequence. *Top* From the position of the end-effector placed on the *right* hand (*red sphere*), we compute the character's pose in four parallel steps. *Bottom* Comparison between the resulting poses (wood texture) and reference mocap data (transparent *red*)

human poses [8,33]. To the best of our knowledge, there is no existing work that quantify the constraints acting on joints for any motion types. In 3D modeling tools (e.g., Blender [2] and Maya [1]), the IK constraints are defined by specifying the joint angular limits along the x, y and z axes. To create animations, artists need to define these values before applying IK. In a real human body, however, these constraints change according to the particular pose. Hence, artists need to modify the angular values for specific poses [5,24,31]. This can be a long and tedious process considering that the human skeleton is a complex system and the joint limits have hierarchical dependencies. Furthermore, for a same end-effector position, several solutions for the skeleton are possible (Fig. 2), even though, not all of them are acceptable. To achieve a desired posture, one solution is to fix joint limits, although this may impair some solutions for other target positions [24].

Eventhough IK solutions are very efficient for simple tasks, more complex structured tasks may significantly impact the computational cost. Performance boosts can be achieved by parallelizing algorithms on multi-core CPUs or GPUs. Such a process is not trivial because IK is inherently a sequential process, i.e., each pose depends upon the previous one (Fig. 3). However, if a complete trajectory of target positions is known, a trivial parallel IK solver would compute the skeletal pose for each target position independently. The resulting generated sequence would likely be discontinuous and would possibly not satisfy the predefined joint constraints.

Contributions This paper proposes a novel constrained IK animation technique. Our method automatically captures joint motion parameters from input mocap data. The parameters are stored in an octree and represent a model for the dynamic nature of human joint behavior. During the animation, parameters are fetched from the octree, and are used to solve the IK problem as well as align the end-effector to the target position. We implemented our method in a parallel pipeline to speed up the performance when the full trajectory of an end-effector is provided, or the target positions are in the format of streaming data, such as real-time video games, 3D modeling software and SAIBA-like Embodied Conversational Agent platform [19]. All computed poses are assembled into a smooth animation sequence using parallel filtering and retargeting processes. Combining these two approaches, we show that our solution is capable of generating high visual quality motion with a significant performance improvement.

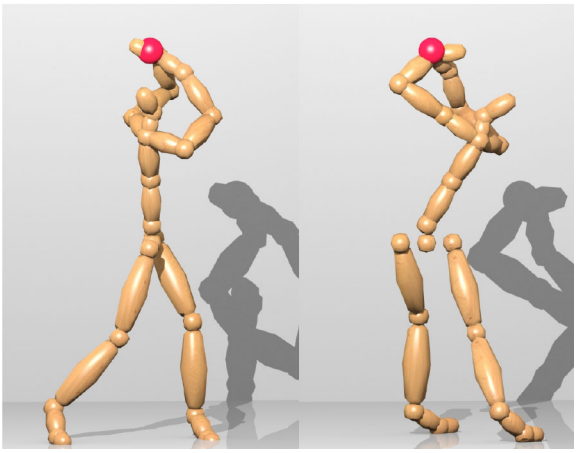


Fig. 2 Different poses are obtained from the same end-effector position

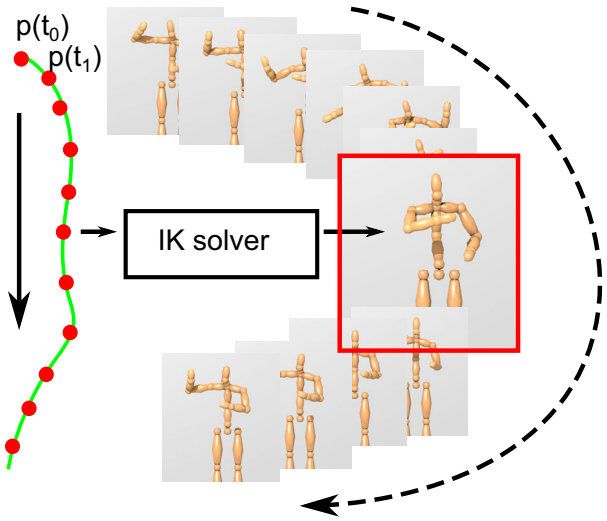


Fig. 3 Each pose is computed considering just the position of the end-effector (*red sphere*). Traditional IK is inherently a sequential process: each pose depends on the previous one

2 Background and motivation

A model of a character skeleton is usually a complex and articulated system composed by kinematic chains of constrained rigid bodies. These rigid bodies are connected to each other with geometric joints, which can be expressed as holonomic constraints, each with independent parameters, the so-called *degrees of freedom (DOFs)*. In this work, we consider each joint having a maximum of three rotational DOFs: a connected Z-Y-X order Euler angles model. In this case, we model each local joint DOF in its individual axis. Through Inverse Kinematics, the animator controls the character skeleton by just positioning an end-effector at the end of the kinematic chain; then, the IK system solves for the joint rotations which place the end-effector at the desired location. Due to the flexibility and performance, IK methods are used to generate animation sequences for reaching and gesture motions and are implemented in a sequential pipeline as shown in Fig. 3.

Given a complete joint configuration of a kinematic chain, its end-effector positions e_i and end-effector targets positions t_i, we need to find the scalars $\theta_1, \ldots, \theta_n$, which represent rotation angles of the joints, satisfying $t_i = e_i$. In forward kinematics (FK), the end-effector position can be parameterized by these scalars. This can be expressed as:

$$\Delta e = f(\Delta\theta) \tag{1}$$

The goal of IK is to find the $\Delta\theta$ such that Δe can make e_i equal to t_i:

$$\Delta\theta = f^{-1}(\Delta e) \tag{2}$$

where f is a nonlinear operator for which f^{-1} may not exist.

The most common approach involves a numerical iterative solver using the Jacobian matrix [18]. The Jacobian J is a partial derivatives matrix that is relative to the current configuration of IK chain; it is defined as:

$$J(\theta)_{ij} = \left(\frac{\partial e_i}{\partial \theta_j}\right)_{ij} \tag{3}$$

where i is the end-effector dimension index, and j is the joints dimension index. The Jacobian inverse solution is:

$$\Delta\theta = J(\theta)^{-1}\Delta e \tag{4}$$

The joint angle is updated by $\theta = \theta + \Delta\theta$.

The IK equation is solvable when the Jacobian matrix is invertible but usually the matrix is not squared and it is not invertible. Several alternatives have also been proposed. One solution [7] uses Pseudo-inverse which works in null space. The Pseudo-inverse is also known as the Moore–Penrose inverse:

$$\Delta\theta = J^{+}\Delta e + (I - J^{+}J)\varphi \tag{5}$$

where $J^{+} = J^{T}(JJ^{T})^{-1}$, φ is any of the vector for the same Δe which can minimize $J\Delta\theta - \Delta e$ in the null space $J(I - J^{+}J)\varphi = 0$. However, the Pseudo-inverse method is often unstable near singularities.

The damped least squares (DLS) solution introduced a regularization term [9,29] to overcome the singularities. Instead of solving $J\Delta\theta - \Delta e$, this method minimizes $\|J\Delta\theta - \Delta e\|^2 + \lambda^2\|\Delta\theta\|^2$, where λ is non-zero damping value. The equation can be solved as follows:

$$(J^{T}J + \lambda^2 I)\Delta\theta = J^{T}\Delta e$$
$$\Delta\theta = (J^{T}J + \lambda^2 I)^{-1}J^{T}\Delta e = J^{T}(JJ^{T} + \lambda^2 I)^{-1}\Delta e$$

Using the λ term, the difference between consequent frames needs to be as small as possible. This keeps the generated frames as a continuous sequence for non-constrained cases, and singularities can be avoided with a suitable λ value.

The computation of the Jacobian inverse is usually very high, however it can be avoided by using proper approximation techniques. In the Cyclic Coordinate Descent (CCD) method [30], one joint is aligned after another to bring the end-effector closer to the target. This method is efficient because it does not require any matrix computation, but it may suffer from unnatural looking results for simulation of the human body. FABRIK [3] is another iterative method in which one end of the joint is translated to reduce the distance between the end-effector and the target. This method is efficient and addresses the positional constraints problem, but it is unsuitable for synthesizing the natural motion of virtual

characters. IKAN [26] is an analytic inverse kinematics system for anthropomorphic limbs; the system solves generic reaching tasks with a higher performance than conventional Jacobian systems, even though it is limited to processing a single kinematic chain. Unfortunately, when applied to complete characters the results are rigid and less natural.

2.1 Related work

As early as the 1970, Liegeois et al. [21] used kinematics equations to solve the poses of a constrained robotic arm sequentially. We use a similar idea: our algorithm learns all the parameters from input mocap data, and defines the joint limits for each degree of freedom.

For simulating virtual characters, the result generated using IK solvers can be quite far from being satisfactory.

Marcelo has proposed a whole-body analytical inverse kinematics (IK) method [17] which integrates collision avoidance and customizable body control for reaching tasks. The computation is based on the interpolation of predesigned key body postures combining with analytical IK solution.

Mocap-based solutions have increasingly gained popularity because the high visual quality. Feng et al. [11] select a number of mocap data samples that are close to the desired constraints, and interpolate them to obtain a new animation sequence. The range of movements achievable in their method is limited by the number of samples given as input.

Grochow et al. [13] use a Scaled Gaussian Process Latent Variable Model to train low dimensional IK search space with high-dimensional data. Their method provides an optimized interpolation kernel for natural poses constrained by positional trajectories. It builds internally an interpolation plane in its motion space (or mapping space) using its variance kernel, however their spatial constraints are not always guaranteed to be reached. The synthesized motion varies according to the choice of samples, and the performance depends on the number of sample candidates. We use a linear model for small segments; the parameters are learnt by clustering similar small motion segments and this removes the redundancy from the data.

Harish et al. [14] propose a parallel IK architecture for high degree of freedom models. They also use the Damped Least Squares Inverse Kinematics method. The parallel computation is used through forward integration and posture reference factor evaluation for different DOFs. Their parallel solution is well defined for high DOFs models. Their solution is applied for each time stamp and their scalability depends on the number of DOFs, while we propose our parallel process for a total sequence through time and our scalability depends on how long the input trajectory is.

Our solution is not a mapping technique, but an extension of a basic IK solver. It is able to generate natural animations by using the learnt parameters. Our solution captures the motion properties of the joints from input mocap data, and stores them in an octree. This octree is accessed during the animation to fetch the motion properties which are more suitable for the IK solver to align the end-effector to a given target point. This allows us to produce natural looking poses similar to the mocap data sequence. Furthermore, we provide a pipeline to compute all the poses in parallel, improving significantly the overall performance of the process.

3 Automatic learning of joint motion parameters

In our system, we employ a variation of Selective Damped Least Squares (SDLS) IK [9,29], and we choose it as our general IK solver because of its stability near singularities. Differently from the original formulation of SDLS, we use a vector $\Lambda = \{\lambda_1, \ldots, \lambda_n\}$, where n is the total number of DOFs and λ_i is a damping factor for the angular velocity of the ith DOF. We modify the objective function for our damping vector:

$$\underset{\theta \in (\theta_{\min}, \theta_{\max})}{\arg\min} \; \|J\Delta\theta - \Delta e\|^2 + \|\mathrm{diag}[\Lambda]\Delta\theta\|^2$$

where $\theta_{\min}, \theta_{\max}$ are the boundaries of DOFs which are learnt and stored in an octree as explained in Sect. 3.1.

The corresponding normal equation is:

$$\begin{pmatrix} J \\ \mathrm{diag}[\Lambda] \end{pmatrix}^T \begin{pmatrix} J \\ \mathrm{diag}[\Lambda] \end{pmatrix} \Delta\theta = \begin{pmatrix} J \\ \mathrm{diag}[\Lambda] \end{pmatrix}^T \begin{pmatrix} \Delta e \\ 0 \end{pmatrix}$$

where $\mathrm{diag}[\Lambda] = \Lambda^T I$ is a diagonal matrix. Hence:

$$\Delta\theta = \left(J^T J + \mathrm{diag}\left[\Lambda^T I \Lambda\right]\right)^{-1} J^T \Delta e \tag{6}$$

It is important to find a suitable assignment of values to the minimal and maximal allowed angles for each degree of freedom, and Λ that model joint rotation velocities.

Some complex joints, such as the human shoulder, are difficult to model manually because joint limits vary due to muscle contraction and joint dependency [5,24,31]. This phenomenon inspired us to model constraints in a flexible dynamic way. Our work on constraints modeling has similar goals, but takes a different approach. We extract such values from mocap data, store them in an octree data structure and use them during the animation.

Feng et al. [11] collect mocap data examples in a k-D tree, using K-nearest points queries to select the best candidate sequences, and blending them into the final animation. Our work on constraints modeling has similar goals, but takes a different approach. We use an *octree* data structure as a container and to cluster similar motions in the same cell.

Each cell in the octree collects similar postures data in spatial nearby region. The data in the same cell are analyzed to define the adaptive joint constraints in this region.

3.1 Octree description

A general octree is used to cluster, compute and save our parameter values. An octree is defined by several initial representations:

The insert points Each tree point in our case represents one pose frame. All frames are sorted by the spatial 3D coordinates of the wrist relative to the root joint computed by their frame poses which are the arm reaching targets (arm end-effectors) in the local space of root joint $\mathbf{p_e}$. A 3D position of the wrist is actually the 3D value of an insert point.

The spatial volume of octree The volume is defined by minimum and maximum values of x, y, and z axes. Given a dataset from an animation sequence, the minimum and maximum values are decided by iterating among all insert points.

The depth of octree In our tests, we use a maximum of 6 levels for the octree. Mostly, the traversal level is between level 3 and level 5. The result shows the defined depth is deep enough to get high quality motions.

The tree cell (octant) An octree cell has its space subdivided in eight sub-cells. It is also a cluster of points inside of its volume. We keep all points (frames) in one vector. Each cell contains the point indices corresponding to the vector. A tree cell contains parameter values computed from clustered points.

The insertion When a cell contains more than 20 points, we split the cell and put these points into the underlying level. We do it recursively till the maximum level. However, we do not remove the point indices in the original level. All levels have a copy of the containing points. This is used to compute the IK parameter values for all levels and all clusters as described in the following.

The octree is filled with mocap data in the initialization phase. We insert animation data (bvh file) by sequence. More specifically, each frame represents a chain pose related to the wrist position. It is also an entry point in the octree. This entry $\mathbf{v_i}$ is defined as

$$\mathbf{v_i} = \left\{\mathbf{p_e}; r_1^i, \ldots, r_m^i; i-1\right\}$$

where m is the number of joints degrees of freedom, $\mathbf{p_e}$ is the end-effector position which is also the traversal index in the tree, r_j^i is the jth degree of freedom of the ith pose stored in the octree cell, i is the frame number and $i-1$ is the corresponding previous frame index (we can easily access the neighbor frame's information in the octree). If the frame is the first frame of one sequence, we put -1 into $i-1$ which indicates that the frame doesn't have a previous frame.

Each cell of the octree stores all the indices $\{i_1, \ldots, i_n\}$ of the entries contained in it. The motion properties of each cell are computed by using these indices to access the DOFs of the corresponding entries, and we can compute the minimal, maximal and weighted average values of each DOF for the cell (Fig. 4). We suppose that all the motion happening in the cell region won't violate these DOF boundaries learnt from data. Taking one cell, we compute the vectors of the minimal and maximal values for each DOF, respectively $\mathbf{r^{min}} = \{r_1^{\min}, r_2^{\min}, \ldots, r_m^{\min}\}$, $\mathbf{r^{max}} = \{r_1^{\max}, r_2^{\max}, \ldots, r_m^{\max}\}$, and the weighted average value $\bar{\mathbf{r}} = \{\bar{r}_1, \bar{r}_2, \ldots, \bar{r}_m\}$. The angular limits for the jth DOF in one cell are computed as the following (see Fig. 5):

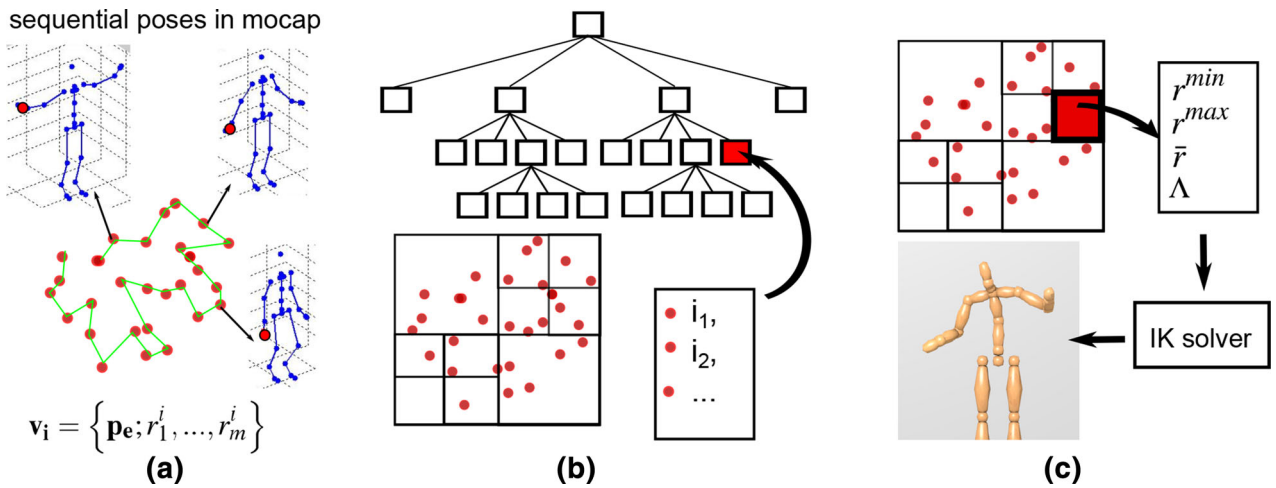


Fig. 4 **a** One vertex is built by the right hand end-effector position and its skeleton posture frame; **b** each tree cell is filled by the data indices which are used to compute its joint constraints set; **c** when doing animation generation, IK solver needs to traverse the octree to fetch the IK constraints in the corresponding cell

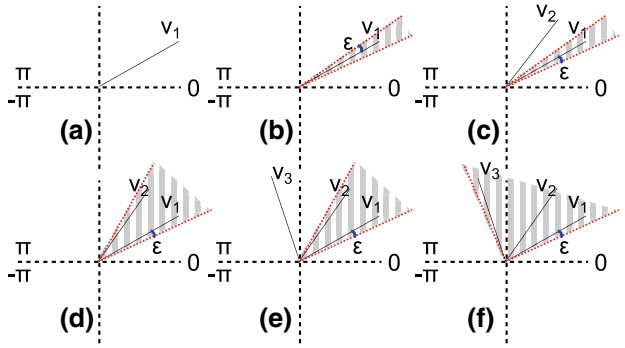


Fig. 5 Computation of $r^{\min}$, $r^{\max}$ (*red* boundaries) by adding entries in the octree cell. The *shadow* part is the covered range

$$r_j^{\min} = \min\left(r_j^{i_1}, \ldots, r_j^{i_n}\right) - \epsilon \tag{7}$$

$$r_j^{\max} = \max\left(r_j^{i_1}, \ldots, r_j^{i_n}\right) + \epsilon \tag{8}$$

$$\bar{r}_j = \frac{\sum_{i_1}^{i_n} w_j^i r_j^i}{\sum_{i_1}^{i_n} w_j^i} \tag{9}$$

$$w_j^i = 2\pi - \left|\left(\mathrm{E}[r_j] - r_j^i\right)\right| \tag{10}$$

where ϵ is a manually set-up small arbitrary positive quantity which is used to control the relaxation of DOF limits which means that when the original learnt joint motion range is too small for reaching certain position, ϵ is used to trade off the reaching target constraint and joint range constraint. By default, we use 0 or 0.001 these small values to keep the motion ranges close to the original data. $r^{\min}$ and $r^{\max}$ define the range of possible values that the DOF will have when the target position will traverse the cell. $\mathrm{E}[r_j]$ is the expectation value of r_j which equals to the average of $r_j^{i_1}, \ldots, r_j^{i_n}$. The $\bar{r}$ represents the most likely configuration in the cell among the observed ones.

For each entry $\mathbf{v}_i$ in the cell, we compute also the Λ_i vector. Rearranging Eq. 6:

$$\mathrm{diag}\left[\Lambda_i^T I \Lambda_i\right] \Delta\theta = J^T(\Delta e - J\Delta\theta) \tag{11}$$

where $\Delta e = \mathbf{p_e}(t) - \mathbf{p_e}(t-1)$, $\mathbf{p_e}(t-1)$ is the position of the end-effector at the previous time step in the trajectory of the mocap data by fetching the $i-1$ indexed frame entry, and $\Delta\theta = \mathbf{r}(t) - \mathbf{r}(t-1)$. We define that each couple frames $i-1$ and i during $\Delta\theta$ is a micro-motion segment. We define the vector $\mathbf{g}$:

$$\mathbf{g} = J^T(\Delta e - J\Delta\theta)$$

and compute λ_j^2:

$$\lambda_{ij}^2 = \left|\frac{g_j}{\Delta\theta_j}\right|$$

Intuitively, λ_{ij}^2 represents the damping factor for the rotation of the jth DOF when the target point is in the cell. We suppose that each micro-motion segment (both start pose and the target pose) is in the same cell. Then, we compute the vector $\bar{\Lambda}$ taking the average of all the Λ_i in the cell. Basically, we use a linear operator to find the damping factor which is adaptable in our situation when the motion segments are small.

Finally, each cell in the octree contains one vector of properties $\mathbf{r^{min}}$, $\mathbf{r^{max}}$, $\bar{\mathbf{r}}$, Λ. This vector represents how fast the movement will be like and how narrow the joint motion range is in this cell region. Mostly, deeper level cells will have a smaller range for joint motion and more specific motion character. For the highest level, we use a set of manual set-up values (such as, for the elbow x axis rotation limits, $r^{\max} = 0$, $r^{\min} = -3.14$). Given a target point (arm wrist position), a pop-up traversal process is used to find the corresponding joint parameters in the octree. The parameters are in the cell containing the target point position. If an empty cell is found, we will do a pop-up, taking a higher level cell instead, including up to the highest level. If the point is not in the tree, we use the highest level parameters. In this case, our solution becomes a general SDL IK solution.

Over all, the Eq. 6 is used to solve for each frame independently. Combined with filtering kernels, we can achieve a natural and smooth motion sequence. The whole process can be computed in parallel. We explain the algorithm using multi-threads more clearly in the following parallel process section.

4 Parallel computation of inverse kinematics animation

Given a set of target points defining a trajectory and the constraints octree, we compute the final animation in four parallel passes. The whole parallel pipeline is depicted in Fig. 6.

Intuitively, each target point is initially assigned to a different thread, which computes the corresponding skeletal pose using the constraint motion parameters stored in the octree. This provides a first crude approximation of the animation which is then refined in a smooth natural sequence in the following steps, as explained in the next sections (see Fig. 7 and the accompanying video for reference).

4.1 Crude IK parallel pass

In this first pass, we aim to obtain natural looking poses without being necessarily temporally coherent. Smoothly changing poses in time will be obtained in the following refinement steps.

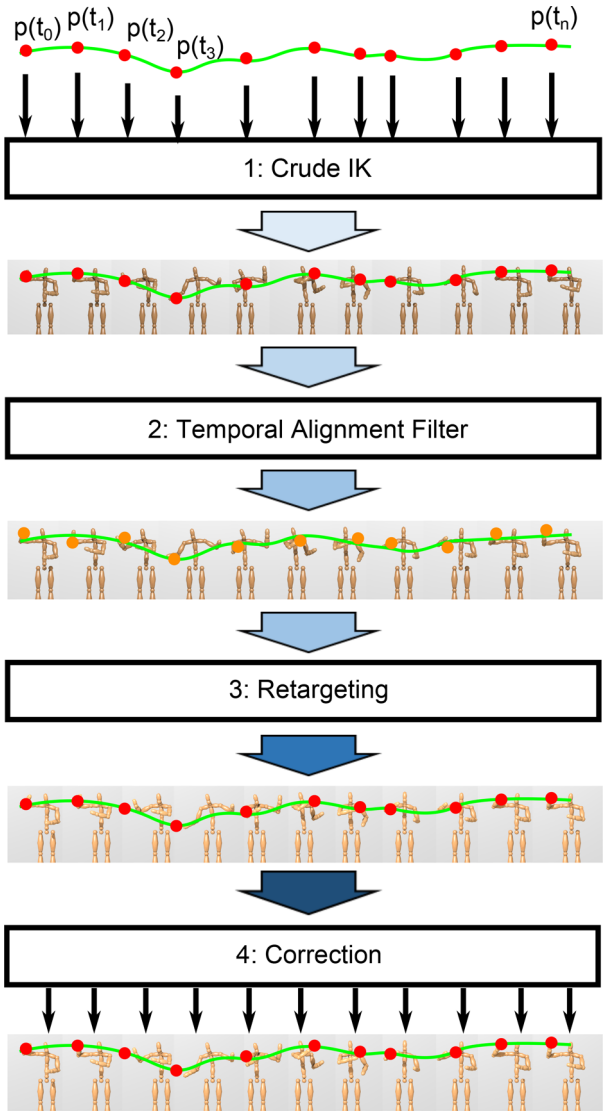


Fig. 6 Parallel pipeline of our IK framework

For each target position, the corresponding thread fetches the joint motion parameters from the octree and feed the IK solver. By using the joint parameters stored in the octree, the resulting poses look natural and not synthetic, because such parameters have been captured from the motion of a real human being.

For each target point, the following operations are performed in the corresponding thread:

1. The octree is traversed with a pop-up, finding the cell c corresponding to the target point t_i;
2. The initial configuration of the kinematic chain is instanced, considering $\theta = \bar{\mathbf{r}}$, $\Lambda = \bar{\Lambda}$, $\theta_{\min} = \mathbf{r}^{\min}$, $\theta_{\max} = \mathbf{r}^{\max}$ (Fig. 7a).
3. The SDLS IK solver Eq. 6 is instanced to compute the skeletal configuration of the joints. A hard clamp kernel is used to cut the resulting θ into the boundary:

$$\theta = \begin{cases} \theta_{\min} & \text{if } \theta < \theta_{\min} \\ \theta_{\max} & \text{if } \theta > \theta_{\max} \\ \theta \end{cases} \tag{12}$$

Even though all the constraints are satisfied and the obtained poses are natural looking, the obtained animation may result as a discontinuous sequence in time, as shown in Fig. 7b and in the accompanying video.

The reason for the discontinuity is that for each small segment, the learnt parameters are from a small cell region. The $\theta_{\min}$, $\theta_{\max}$ are much smaller than the physical joint limits that other referenced methods used. The resulting angle in the small range may not fit for both the previous frame and the following frame when each frame is computed individually and independently in Fig. 8. The skipped frame may happen when two close-frames have too much difference and their solutions are not sequential dependent.

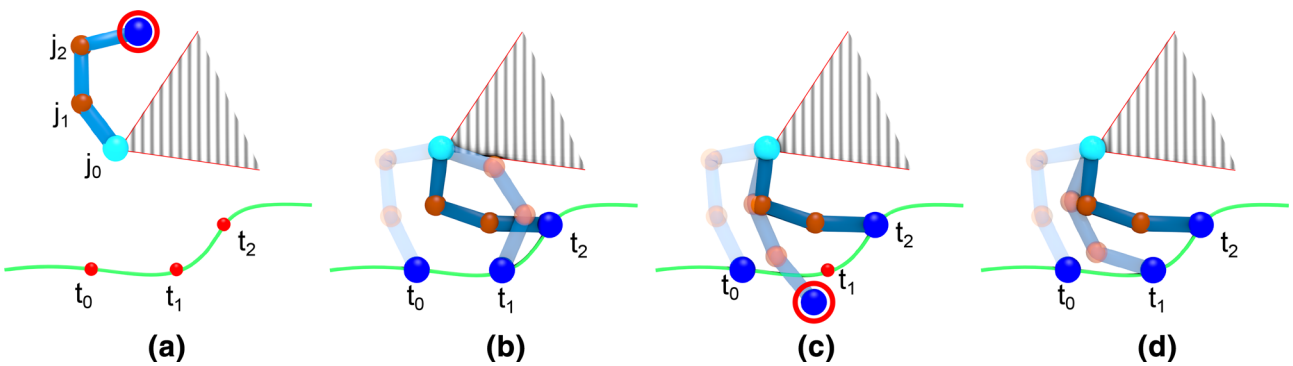


Fig. 7 **a** Initial configuration of a kinematic chain. t_0, t_1 and t_2 are the target positions along the trajectory. **b** Crude IK solver pass. The resulting poses are not temporally coherent. **c** Temporal Alignment Filtering. The solution is now smooth but some end-effectors may be not aligned anymore to the target position. **d** Retargeting and final correction. Using the previous solutions as the starting configuration, the IK solver finds a smooth solution with end-effectors aligned correctly to the target positions. The *white* part is the free space for the joint's rotation

4.2 Temporal alignment filtering pass

In the second step, all the computed poses are aligned in time in a smooth, continuous sequence. We use a simple box filter [15,23,27], with a large half window size w to convolve the crude skeletal poses:

$$r^{\text{filtered}} = \frac{1}{2w} \sum_{i \in \Omega} r_i \tag{13}$$

where r is the joint angles configuration, Ω is the set of r in the time window, the index i is included in $-w$ to w, and w is the size of the half window (for all our test cases, we used $w = 10$). At the end of this pass, the angular difference between corresponding joints in sequential frames is highly reduced. As a result, the animation is temporally coherent but the end-effectors are not guaranteed to reach the targets (Fig. 7c). In this step, we only use large window box filter to get smooth sequence which is different from the fourth step.

4.3 IK retargeting and correction passes

The remaining two parallel steps are used to further refine the animation sequence.

IK retargeting pass In this pass, we basically repeat the first step: we instance a thread for each target position and run the Ik solver separately for each target position. The main difference with the first step is that the solver is fed with the results of the previous step, in particular r and Λ. As a result, the end-effector reaches the target positions and, differently from the first step, the skeletal poses are temporally coherent (Fig. 7d).

Final correction pass Even though the end-effector position and the temporal coherency are satisfied, there may be cases in which the joint angular acceleration produces abrupt changes in the motion (e.g., see Figs. 9, 10).

To prevent the effect and obtain a smoother sequence, we add another filtering pass to correct this issue. The skipped-frame problem due to the independent frame computation is solved with such a filtering process. We use a bilateral filter [15,23,27] with a small half window size w (in this case $w = 2$ or $w = 3$): the weights are parameterized by both a) the Euclidean distance between end-effector and its target, and b) the angular difference between frames.

$$r^{\text{filtered}} = \frac{1}{W} \sum_{i \in \Omega} r_i f(\|t_i - e_i\|) g(\|i\|), \tag{14}$$

where g is a temporal distance kernel, f is a spatial distance kernel, t_i, e_i are the target position and end-effector position, W is the sum of all the weights in one window. In this step, we choose to use an edge preserving filter. By using such filter, we can keep the sequence temporally smooth in joint space while reducing the error in target space. We are able to deal with two signals.

For both of the f and g functions, we use a Wendland [32] kernel which has derived piecewise polynomial functions with positive definite and compact support.

$$wl(d) = \begin{cases} \left(1 - \frac{d}{\epsilon}\right)^4 \left(\frac{4d}{\epsilon} + 1\right) & \text{if } 0 \le d \le \epsilon \\ 0 & \text{if } d > \epsilon \end{cases} \tag{15}$$

where d is the current distance, ϵ is the maximum or threshold distance. We setup the $\epsilon = w$ for g and $\epsilon = 0.05$ for f.

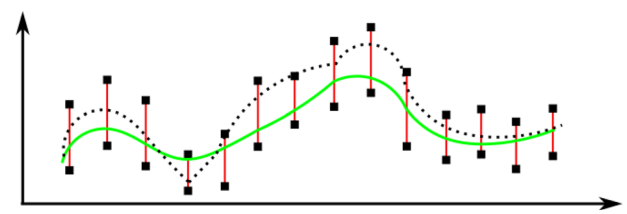


Fig. 8 The possible IK results in our experience: *red vertical lines* are DOF ranges for each time instance learnt from the data; *dot line* is the sequence that each frame is generated independently; *green line* is the sequence with temporal alignment

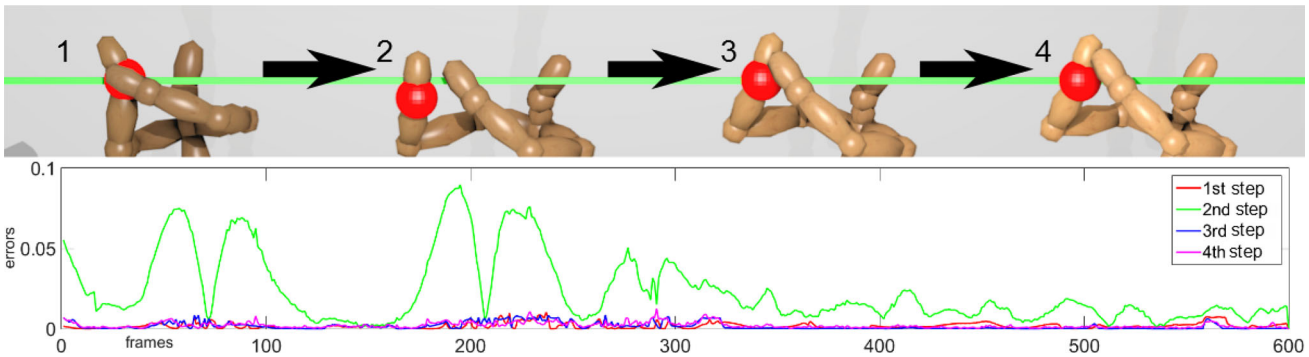


Fig. 9 An example of trajectory errors in meter introduced by our method with four parallel steps. Only the second pass—aligning filter pass makes the targets unreachable (with large errors)

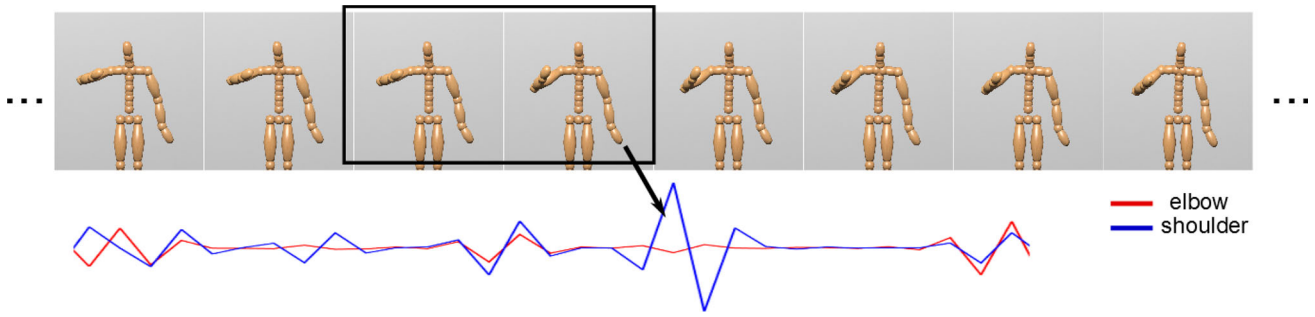


Fig. 10 *Top* One example of skipped frames in the retargeting pass, and this can be minimized by filtering frames in correction pass; *bottom* the joints accelerations, *red-line* is elbow, *blue-line* is shoulder, the shoulder's velocity varies too quickly

Table 1 Time performance comparison in milliseconds of different solutions when giving different trajectory frames (fs) from motion capture data: **sq** (our sequential solution), **pl** (our parallelized implementation), **dls1** (dls solution taking original pose as initial state), **dls2** (dls solution taking previous frame pose as initial state)

Motion pass	sq/pl	CIP 1st	TAFP 2nd	IKRP 3rd	CFP 4th	Sum
Angry	sq	578.8	3.1	454.7	0.6	1037.3
1500fs	pl	78.8	1.2	62.6	0.3	143.0
	dls1					753.2
	dls2					461.3
Throw	sq	16.9	0.3	23.2	0.1	40.4
150fs	pl	2.8	0.2	3.6	0.1	6.7
	dls1					42.3
	dls2					15.5
Move	sq	110.3	1.0	50.1	0.3	161.8
600fs	pl	19.9	0.6	6.1	0.3	27.0
	dls1					191.3
	dls2					109.4
Golf	sq	265.1	3.7	200.2	0.7	469.7
2500fs	pl	36.6	1.7	38.5	0.3	77.1
	dls1					847.1
	dls2					304.5
Tennis	sq	171.0	3.7	153.1	0.6	328.4
2000fs	pl	24.1	1.4	23.4	0.4	49.3
	dls1					673.3
	dls2					211.7

5 Results

We implemented and tested the performance of our method on an Intel Xeon CPU @2.93GHz, 2 processors with 4 cores each. The implementation uses Intel Threading Building Blocks (TBB) library to parallelize the code. We tested several examples and report the performance in Table 1, where we compare the computational time with the sequential implementation of our code. Note that the first step of the sequential implementation is equivalent to the performance of the SDLS IK method (similar to dls2 in Table 1). The computation time is not linear with the number of frames but depends on the number of iterations needed to reach a target position, similarly to SDLS IK and any iterative IK method.

Table 2 The error metric of comparing different solutions with original mocap in both joint space and target distance

Methods	Joint mse	Distance mse
Ours (sq/pl)	0.0015	0.00783
dls1	0.261	0.00861
dls2	0.326	0.00761

We also compute the average errors using both error metric in joint space and target distance space to evaluate different methods in Table 2. Our solution can generate good postures that are closer to the referenced mocap data compared with original dls solutions.

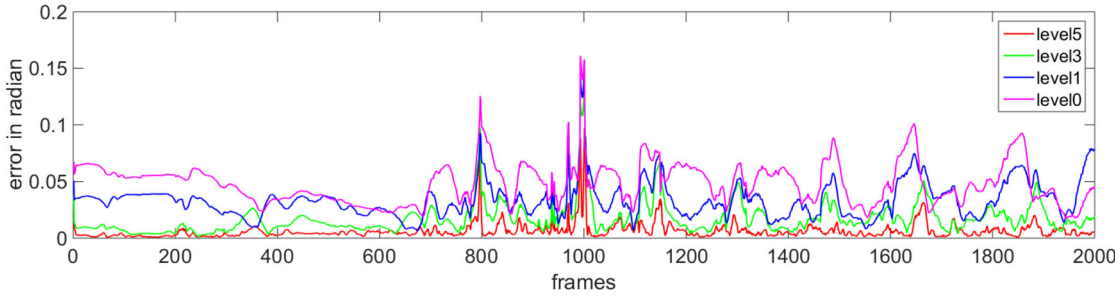


Fig. 11 Comparison of normalized mean squared errors (rotations over all joints) considering our animation at different levels compared to the original mocap data in the example of tennis. The errors summed up for the whole sequence are: 97.42 (level 0), 67.13 (level 1), 46.73 (level 2), 34.33 (level 3), 22.22 (level 4), 12.70 (level 5)

Table 3 Time to compute an animation using octrees with different maximal depth

Level	0	1	2	3	4	5
Golf	227.6	92.0	75.9	78.6	77.7	76.2
Tennis	115.7	72.5	65.6	51.1	58.1	51.4
Throw	12.1	8.2	7.6	6.8	7.8	8.3

As input, we use the mocap animation sequences in the Emilya data set [12]. It turns out that the parallel version is ∼7 times faster than the sequential version, and ∼ 4 times faster than non-constrained SDLS when a whole animation sequence is generated. It is worth mentioning that our method is scalable, meaning that better performances can be achieved if more cores are available. In all the test cases, we use the right hand-wrist joint as the end-effector. To validate the synthesized animations, we used the trajectory of the hand in the original data from mocap sequences. Then, we compared the original ground truth (the mocap data), with our resulting animation. This comparison is visually represented in the accompanying video. In Fig. 11, we show the normalized errors considering octrees with different depths. We also measured the performance of our method using octrees with different maximal depths. Table 3 shows that the performance does not change significantly when the maximal depth number is bigger than 2.

The effect of the smoothing process is represented in Fig. 13: the smoothness of the motion is already largely improved after temporal alignment and retargeting. The final filtering step generates a continuous sequence. In extreme case, the target positions of trajectory are not in the space of learned motion, our solution can still generate smooth animation (Fig. 12) with given learned joint limits (Fig. 14).

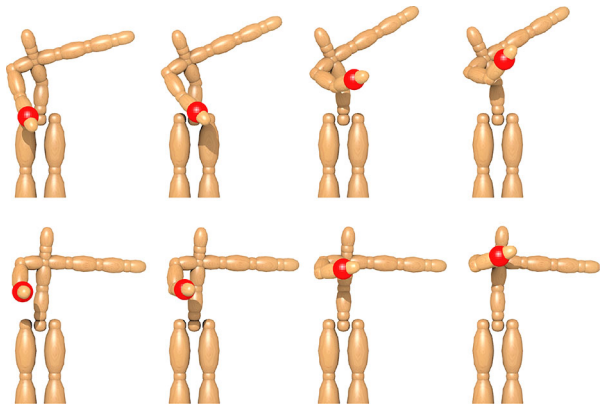


Fig. 12 Given a golf swing trajectory and tennis joint constraints, we generate tennis-like golf swing: *top* original golf, *bottom* tennis-like golf

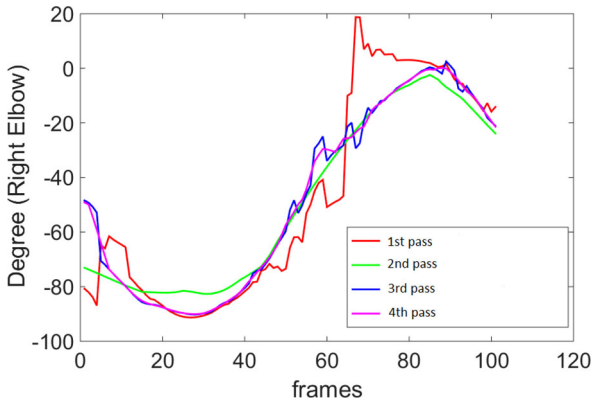


Fig. 13 Elbow joint rotation in sequential frames

5.1 Limitations

Our solution can generate smooth motion. However, there exists several drawbacks. First, the input of our generation is a trajectory. Without the temporal information, it is hard to apply our filtering process. For our solution, it is required to stream in whole trajectories one by one to generate continuous sequence. The smoothness is only guaranteed when using our edge preserving filters. Further, during the animation the joint motion parameters are computed according to the position of the target point: if this is not included in the octree, the resulting animation will likely appear incoherent and unnatural as illustrated in the accompanying video.

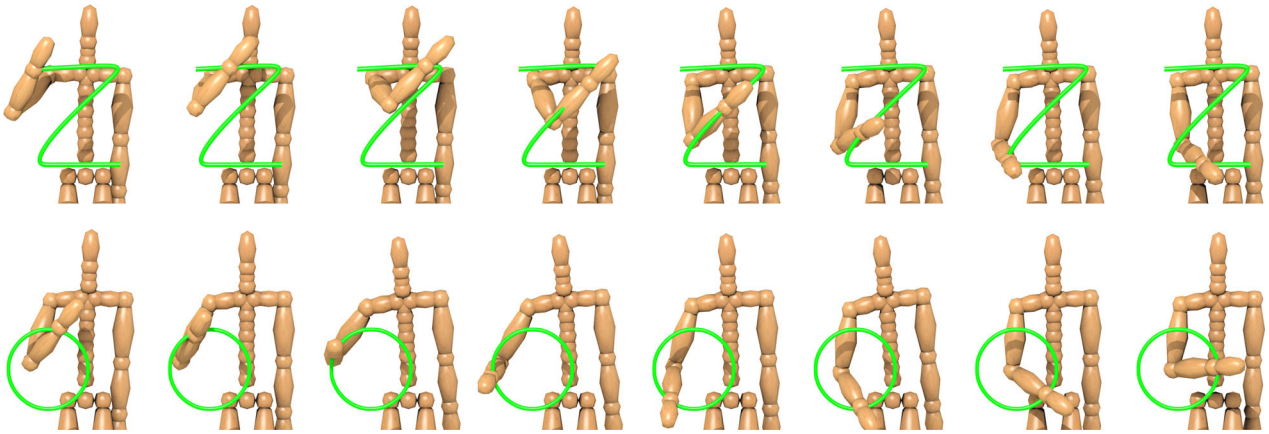


Fig. 14 We use the octree to synthesize novel motion not present in the input mocap data

For these extreme cases, certain frames may not satisfy both target constraint and DOFs limits, the trade-off parameterization between constraints is needed. Moreover the filtering steps could cut away quick joint rotation though this is a problem shared by many different IK approaches. Our method does not include the dimension reduction process, thus it is not adaptable for high dimensional targets; our solution is useful for chain IK that can generate natural postures.

6 Conclusions and future work

We propose a novel parallelizable constrained IK technique which allows us to model and use dynamic joint motion parameters. These values do not need to be set manually but are automatically learned from input mocap data. They are stored in an octree and accessed and used during the animation according to the current target position.

We implemented our method in a computational pipeline which can process in parallel all the target positions of the input trajectory, generating smooth joint rotations and eliminating potential discontinuities. We used SDLS IK as the building block in our pipeline. However, it is noted that any constrained IK method can be used in our approach. Our technique also allows us to parallelize the Inverse Kinematics solver for temporally dependent targets in one trajectory. The parallel passes strategy can well solve the incoherent skipped-frames problem while doing multi-processing. The obtained results show how our approach largely improves the performance of procedural IK animation generation. Note that this parallel strategy can be applied for different procedural IK methods.

As a future work, we would like to investigate the GPU-based implementation of our pipeline to speed up the performances. We also aim to explore the idea of using a forest of octrees to characterize different types of stylistic motion. We would also like to improve character IKs for high-dimensional mapping with full body performance.

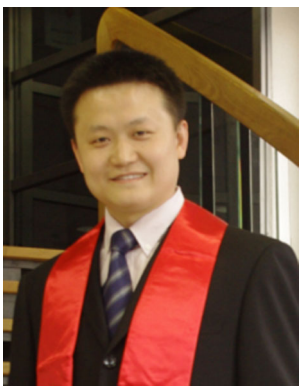
References

1. Autodesk Maya (2015). http://www.autodesk.com/ Accessed 20 Sept 2015
2. Blender (2015). https://www.blender.org/. Accessed 20 Sept 2015
3. Aristidou, A., Lasenby, J.: Fabrik: a fast, iterative solver for the inverse kinematics problem. Graph. Models **73**(5), 243–260 (2011). doi:10.1016/j.gmod.2011.05.003
4. Baerlocher, P., Boulic, R.: Task-priority formulations for the kinematic control of highly redundant articulated structures. In: Proceedings of 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 1, pp. 323–329 (1998). doi:10.1109/IROS.1998.724639
5. Baerlocher, P., Boulic, R.: Parametrization and range of motion of the ball-and-socket joint. DEFORM/AVATARS **196**, 180–190 (2000)
6. Baerlocher, P., Boulic, R.: An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. Vis. Comput. **20**(6), 402–417 (2004)
7. Baillieul, J.: Kinematic programming alternatives for redundant manipulators. In: Proceedings of 1985 IEEE International Conference on Robotics and Automation, vol. 2, pp. 722–728 (1985). doi:10.1109/ROBOT.1985.1087234
8. Blow, J.: Inverse kinematics with quaternion joint limits. Game Developer (2002)
9. Buss, S.R., Kim, J.S.: Selectively damped least squares for inverse kinematics, pp. 37–49 (2004)
10. Chiaverini, S.: Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. IEEE Trans. Robotics Autom. **13**(3), 398–410 (1997)
11. Feng, A.W., Xu, Y., Shapiro, A.: An example-based motion synthesis technique for locomotion and object manipulation. In: Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 95–102. ACM (2012)
12. Fourati, N., Pelachaud, C.: Emilya: Emotional body expression in daily actions database, pp. 3486–3493 (2014)
13. Grochow, K., Martin, S.L., Hertzmann, A., Popović, Z.: Style-based inverse kinematics. In: ACM SIGGRAPH 2004 Papers, SIGGRAPH '04, pp. 522–531. ACM, New York (2004). doi:10.1145/1186562.1015755

14. Harish, P., Mahmudi, M., Callennec, B.L., Boulic, R.: Parallel inverse kinematics for multithreaded architectures. ACM Trans. Graph. **35**(2), 19:1–19:13 (2016)
15. Huang, J., Boubekeur, T., Ritschel, T., Holländer, M., Eisemann, E.: Separable approximation of ambient occlusion. In: Proceedings of Eurographics 2012 Short, pp. 29–32 (2011)
16. Huang, J., Pelachaud, C.: An efficient energy transfer inverse kinematics solution. Proc. Motion Game 2012 **7660**, 278–289 (2012)
17. Kallmann, M.: Analytical inverse kinematics with body posture control. Comput. Anim. Virtual Worlds **19**(2), 79–91 (2008)
18. Klein, C.A., Huang, C.H.: Review of pseudoinverse control for use with kinematically redundant manipulators. IEEE Trans. Syst. Man Cybern. **SMC–13**(2), 245–250 (1983). doi:10.1109/TSMC.1983.6313123
19. Kopp, S., Krenn, B., Marsella, S., Marshall, A.N., Pelachaud, C., Pirker, H., Tharisson, K.R., Vilhjalmsson, H.: Towards a common framework for multimodal generation in ecas: The behavior markup language. In: Proceedings of the 6th International Conference on Intelligent Virtual Agents, pp. 21–23, Marina (2006)
20. Le Callennec, B., Boulic, R.: Interactive motion deformation with prioritized constraints. Graph. Models **68**(2), 175–193 (2006)
21. Liegeois, A.: Automatic supervisory control of the configuration and behavior of multibody mechanisms. IEEE Trans. Syst. Man Cybern. **7**(12), 868–871 (1977)
22. Nakamura, Y., Hanafusa, H.: Inverse kinematic solutions with singularity robustness for robot manipulator control. J. Dyn. Syst. Meas. Control **108**(3), 163–171 (1986)
23. Paris, S., Durand, F.: A fast approximation of the bilateral filter using a signal processing approach. In: Computer Vision—ECCV 2006, pp. 568–580. Springer, Berlin (2006)
24. Shao, W., Ng-Thow-Hing, V.: A general joint component framework for realistic articulation in human characters. In: Proceedings of the 2003 Symposium on Interactive 3D Graphics, pp. 11–18. ACM (2003)
25. Tang, W., Cavazza, M., Mountain, D., Earnshaw, R.A.: Real-time inverse kinematics through constrained dynamics. In: Proceedings of the International Workshop on Modelling and Motion Capture Techniques for Virtual Environments, CAPTECH '98, pp. 159–170. Springer, London (1998)
26. Tolani, D., Goswami, A., Badler, N.I.: Real-time inverse kinematics techniques for anthropomorphic limbs. Graph. Models **62**(5), 353–388 (2000)
27. Tomasi, C., Manduchi, R.: Bilateral filtering for gray and color images. In: Sixth International Conference on Computer Vision, pp. 839–846. IEEE (1998)
28. Unzueta, L., Peinado, M., Boulic, R., Suescun, A.: Full-body performance animation with sequential inverse kinematics. Graph. Models **70**, 87–104 (2008)
29. Wampler II, C.W.: Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods. IEEE Trans. Syst. Man Cybern. **16**(1), 93–101 (1986). doi:10.1109/TSMC.1986.289285
30. Wang, L.C., Chen, C.: A combined optimization method for solving the inverse kinematics problems of mechanical manipulators. IEEE Trans. Robotics Autom. **7**(4), 489–499 (1991). doi:10.1109/70.86079
31. Webber, B.L., Phillips, C.B., Badler, N.I.: Simulating humans: computer graphics, animation, and control, p. 68. Center for Human Modeling and Simulation (1993)
32. Wendland, H.: Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. Adv. Comput. Math. **4**(1), 389–396 (1995)
33. Wilhelms, J., Gelder, A.V.: Fast and easy reach-cone joint limits. J. Graph. Tools **6**(2), 27–41 (2001)
34. Wolovich, W., Elliott, H.: A computational technique for inverse kinematics. In: The 23rd IEEE Conference on Decision and Control, vol. 23, pp. 1359–1363 (1984)
35. Yamane, K., Nakamura, Y.: Natural motion animation through constraining and deconstraining at will. IEEE Trans. Vis. Comput. Graph. **9**, 352–360 (2003). doi:10.1109/TVCG.2003.1207443
36. Yuan, J.: Local svd inverse of robot jacobians. Robotica **19**(1), 79–86 (2001)
37. Zhao, J., Badler, N.I.: Inverse kinematics positioning using nonlinear programming for highly articulated figures. ACM Trans. Graph. **13**, 313–336 (1994)

J. Huang is a postdoctoral researcher at CNRS in the laboratory LTCI, TELECOM ParisTech. His research interests are mainly interactive rendering and animation generation with parallel high performance computing. He is also working on learning and modeling expressive gestures in interactive communicative system. He obtained his Ph.D. degree (2013) in computer graphics at TELECOM ParisTech, and received his master degree (2009) at University of Paris Descartes

M. Fratarcangeli is a Senior Lecturer at Chalmers University of Technology, in Gothenburg (Sweden). His research interests are in the area of interactive animation of physically based objects. Previously, he was an assistant professor (2011–2014) at Sapienza University of Rome (Italy), where he also obtained his Ph.D. degree (2009) in the field of computer graphics. He have been a visiting professor (03–07/2014) at Télécom ParisTech in Paris (France), and from 2004 to 2006 a visiting researcher at the Linköping Institute of Technology (Sweden)



Y. Ding is a postdoctoral researcher at CNRS in the laboratory LTCI, TELECOM ParisTech. He received the BS degree (2006) in automation from Xiamen University, the MS degree (2010) in computer science from Pierre and Marie Curie University and the Ph.D. degree (2014) in computer science from Telecom Paristech. His researches include modeling and understanding human communication and interaction, with application in recognition and synthesis

C. Pelachaud is a Director of Research at CNRS in the laboratory LTCI, TELECOM ParisTech. Her research interest includes embodied conversational agent, nonverbal communication (face, gaze, and gesture), expressive behaviors and socio-emotional agents. She is associate editors of several journals among which *IEEE Transactions on Affective Computing*, *ACM Transactions on Interactive Intelligent Systems* and *Journal on Multimodal User Interfaces*. She has co-edited several books on virtual agents and emotion-oriented systems. She is recipient of the ACM—SIGAI Autonomous Agents Research Award 2015