Advanced Computer Graphics Spring 2019 (RPI)

PhysIK: Physics based Inverse Kinematics for Character Posing and Animation

Frederick Choi

April 27, 2019

1 Introduction

Posing and animating rigs can frustrating for animators. Among the tools they use to make things easier, inverse kinematics solvers calculate the relative rotations of rigid bodies connected by different types of joints which is useful for both robotics and animation. However, when posing a rig for 3D modeling and animation, some inverse kinematics solvers can produce unpredictable and unintuitive results, especially for large displacements.

Along with the issues above, IK solvers are also limited in their ability to create dynamic poses. It can take a large number of target point specifications to create a pose or animation that has feasible secondary effects. For example, when a person walks, the legs are not the only parts of the body that moves, but the arms, torso, and head all move in reaction.

PhysIK is a physics-based character posing and inverse kinematics algorithm inspired by how artists manually pose wooden mannequins. To pose a wooden mannequin, the artist just has to move it into place. Part of what makes it so easy to use is that we have an intuitive sense of how physical objects work. Objects in the real world do not jump around or pass through themselves. In order to capture this intuition, PhysIK simulates physics on a rig to move the rig into the desired configuration. By taking into account the form and mass of the rig, PhysIK can accurately simulate plausible secondary motions. For this reason, PhysIK is useful for animation as well as static posing.

In addition, with a physics-based simulation, there is no need to worry about whether a solution exists. Even if the target pose is impossible, the simulation will not break down. Instead, it will find a good compromise between what the user specified and what is physically possible.

2 Background & Related Works

Inverse Kinematics is a topic that has been researched over and over, with a myriad new improvements and new algorithms for different applications being published every year. Popular algorithms tend to be of two types: minimizers, and predictor-corrector type algorithms.

Among the minimizer type algorithms, CCD (Cyclic Coordinate Descent) is the most popular method of inverse kinematics today. One of the first papers to use it for inverse kinematics is by Wang and Chen [1]. They cite advantages over contemporary methods such as gradient descent, where CCD has a much faster convergence rate and does not have issues with singular Jacobians. iTasC published by Smith et al. [2] provides a specification for movement in robotics, which can be used to solve IK problems. An implementation of a solver that works on iTasC specification that uses a DLS (damped least squares) method is available in Blender [3]. A performance comparison of PhysIK to Blender's iTasC solver is included in the results section.

In the predictor-corrector vein of IK solvers, Kulpa and Multon [4] have a hierarchical algorithm for IK on humanoid rigs that is both quick but also calculates realistic poses for humans. FABRIK by Artistidou and Lasenby [5] have an inverse-kinematics solver more geared toward robotics. These are both predictor-corrector methods that approximate the solution without transforming the problem into finding a minimum on a parameter space.

These algorithms are good finding accurate solutions quickly, but when discontinuities are present in the parameter space (e.g. angular constraints), the minimizers can fail to find a minimum, or they might end up finding a local minimum that is far from what the user intended. They might also find a minimum that is infeasible to reach from the current configuration, which is particularly bad in animation, as this can lead to unnatural-looking poses. The predictor-corrector algorithms provide fast solutions to special cases of rigs that tend to avoid the above issues, but they do not take human abilities into account and may produce poses that have unnatural secondary movements, that is, the rest of the rig may not react in the way the user wants.

A physics based solution can take into account the human form. There is a substantial body of research on physics-based marker following, with a sizeable amount motivated by the need for tools to translate motion capture data into computer animation. Zordan and van der Horst [6] published a method of marker following by simulating spring forces on the surface of a rig and using the ODE (Open Dynamics Engine [7]). They do not include explicit constraints on the rig, such as joint angles, and instead, rely on the input data and damping forces to ensure a natural pose is computed. They also offer contact forces which allow for more accurate interaction with the environment. Along similar lines, Cooper and Ballard [8] published a marker-based following algorithm that similarly constructs a physical model of the human body, but instead of using forward simulation, they compute the forces each joint would have to produce to reach that marker. This makes for a more realistic solution since the source of motion would be from the rig itself, like a human with muscles. The secondary motions it produces would also be more in line with what a real human would look like. This algorithm uses ODE to do both forward simulation and inverse physics.

PhysIK uses its own physics engine to simulate forward physics. Using Pickl's master thesis [9] as a reference for rigid body mechanics, and heuristics on the transfer of forces between bones connected by joints, a simplified version of a dynamics engine was developed that automatically calucates inter-joint forces to propagate motion from bone to bone. It is specifically designed to work in the framework of Blender, but it can be extended to any software with a consistent system of local and global coordinates. It is also optimized to be fast enough to be interactive despite the overhead of going through the Blender API.

3 Algorithm

3.1 Overview

The following terms will be used throughout this paper, and are borrowed from Blender's terminology. A rig, also commonly referred to as a skeleton or armature, is a set of bones, which in the case of 3D effects, is used to control the overall shape of a mesh. Each bone has at most 1 parent and any number of children. The local transformations of child bones are defined relative to their parent.

PhysIK solves for the transformation of each bone relative to its parent such that specified parts of the rig are in their target position while satisfying angular constraints. Angular constraints are defined per joint and restrict the rotation of the bone into an interval per axis (x, y, z) (relative to the parent).

The transformations are computed using a physical simulation. Spring forces are used to coerce the rig into the desired configuration, while drag and other forces are used to stabilize the simulation. The propagation of motion between bones are computed after all forces are applied, and this creates the natural motions that PhysIK aims to simulate.

Objectives. The objectives of PhysIK are to be:

- Accurate. For static poses, the rig should converge to a configuration that is close to what the user specifies.
- Interactive. The algorithm should converge and should converge fast enough to be interactive.
- Intuitive. The whole rig should react in an appropriate, physically intuitive manner without explicitly specifying secondary movement.

Accuracy can be evaluated quantitatively. Intuitiveness will be evaluated qualitatively by comparing it with other IK solvers in Blender. While speed is also a consideration, the only requirement is that it should be interactive, since the limitations of the Blender plugin API makes it difficult to measure the true performance of the algorithm.

Physics. A custom physics engine is implemented where the objectives are to model rigid body physics to simulate a rig. Spring forces (linear and non-linear), inter-joint forces, and torsion springs are simulated. Spring forces are used to coerce parts of the rig to their target position. Inter-joint forces are used to propagate the motion of one bone to other bones. Torsion springs are used to enforce angular constraints.

Control Points. Control points are how the user specifies the final pose of the rig. A control point consists of two parts: the attachment point and the target point. The attachment point is on the rig, while the target point lies somewhere in the space. In the physical simulation stage, the attachment point is moved toward the target point by simulating a spring (linear or non-linear).

Movement Interpolation. When the target point of a control point is moved far away from a control point (i.e. the angular displacement exceeds some threshold) then the movement of the target point will be interpolated over each iteration to improve the convergence of the algorithm. Some interpolation schemes that were investigated include a circular arc around the center of mass, a circular arc around the head of the bone, and linear interpolation.

Interface. Physics on the rig is simulated in real time as the control points are dragged around. This allows for live feedback to the user as they pose the rig. This also allows the simulation to be recorded as an animation for easy secondary movement.

3.2 Physical Simulation

A physical simulation is used to pose the rig into a configuration that approximately satisfies all constraints, but in a way that is predictable. When an artist poses a wooden model, the limbs do not jump around. Likewise, by simulating the bones of the rig, they move into position in a temporally and spatially coherent manner so they end up in a predictable position.

In order to run a physical simulation the following forces were simulated: inter-joint forces to ensure joints stay connected, spring forces to move parts of the rig into place, and torsion springs to ensure angular constraints are satisfied. Damping was simulated in order to ensure convergence. Stiffness and mass are simulated to better match physical intuition.

To clarify terminology, let the local fixed frame of a bone be a coordinate frame whose origin coincides with the center of mass of the bone, and is fixed with respect to the bone.

Inter Joint Forces. In the simulation, bones are treated as independent rigid rods. Below is the equation for the dynamics of a single bone (simplified from [9], derived from Newton-Euler equations). To simplify calculations, the origin of the reference frame is the center of mass of the bone.

$$\begin{aligned} \boldsymbol{a}_{cm} &= m^{-1}\boldsymbol{F}_{net} \\ \boldsymbol{\alpha} &= I_{cm}^{-1}(\boldsymbol{\tau}_{net} - \boldsymbol{\omega} \times I_{cm}\boldsymbol{\omega}) \end{aligned} \tag{1}$$

where

$\boldsymbol{a}_{cm}$ = acceleration of the center of mass of bone
m = mass of bone
$\boldsymbol{F}_{net}$ = net force acting on bone
$\boldsymbol{\alpha}$ = angular acceleration of bone
I_{cm} = moment of inertia about center of mass
$\boldsymbol{\tau}_{net}$ = net moment on bone
$\boldsymbol{\omega}$ = angular velocity of bone

To ensure that the bones stay connected at their joints, the acceleration at the point of contact should be the same for each bone. Consider a point $\boldsymbol{r}$ in the local fixed frame of the bone. The acceleration of the point $\boldsymbol{r}$ can be found with the following formula:

$$\begin{aligned} \boldsymbol{a}_{\boldsymbol{r}} &= \boldsymbol{a}_{cm} + \boldsymbol{\alpha} \times \boldsymbol{r} \\ &= m^{-1}\boldsymbol{F}_{net} + I_{cm}^{-1}(\boldsymbol{\tau}_{net} - \boldsymbol{\omega} \times I_{cm}\boldsymbol{\omega}) \times \boldsymbol{r} \\ &= m^{-1}\boldsymbol{F}_{net} + [\boldsymbol{r}]_\times^T I_{cm}^{-1}(\boldsymbol{\tau}_{net} - [\boldsymbol{\omega}]_\times I_{cm}\boldsymbol{\omega}) \end{aligned} \tag{2}$$

where $[\cdot]_\times$ is the cross product matrix.

Now suppose another force $\boldsymbol{F}$ is applied at the point $\boldsymbol{s}$ in the local frame of the bone. Then the new acceleration $\bar{\boldsymbol{a}}_r$ at the point $\boldsymbol{r}$ can be calculated like so:

$$\begin{aligned} \bar{\boldsymbol{a}}_r &= m^{-1}(\boldsymbol{F}_{net} + \boldsymbol{F}) \\ &\quad + [\boldsymbol{r}]_\times^T I_{cm}^{-1}(\boldsymbol{\tau}_{net} + \boldsymbol{s} \times \boldsymbol{F} - [\boldsymbol{\omega}]_\times I_{cm}\boldsymbol{\omega}) \\ &= m^{-1}\boldsymbol{F}_{net} + [\boldsymbol{r}]_\times^T I_{cm}^{-1}(\boldsymbol{\tau}_{net} - [\boldsymbol{\omega}]_\times I_{cm}\boldsymbol{\omega}) \\ &\quad + m^{-1}\boldsymbol{F} + [\boldsymbol{r}]_\times^T I_{cm}^{-1}(\boldsymbol{s} \times \boldsymbol{F}) \\ &= \boldsymbol{a}_r + m^{-1}\boldsymbol{F} + [\boldsymbol{r}]_\times^T I_{cm}^{-1}(\boldsymbol{s} \times \boldsymbol{F}) \\ &= \boldsymbol{a}_r + m^{-1}\boldsymbol{F} + [\boldsymbol{r}]_\times^T I_{cm}^{-1}[\boldsymbol{s}]_\times \boldsymbol{F} \\ &= \boldsymbol{a}_r + (m^{-1}I_3 + [\boldsymbol{r}]_\times^T I_{cm}^{-1}[\boldsymbol{s}]_\times)\boldsymbol{F} \end{aligned} \tag{3}$$

where I_3 is the 3 by 3 identity matrix.

Let $M(\boldsymbol{r}, \boldsymbol{s}) = (m^{-1}I_3 + [\boldsymbol{r}]_\times^T I_{cm}^{-1}[\boldsymbol{s}]_\times)$. Then

$$\bar{\boldsymbol{a}}_r = M(\boldsymbol{r}, \boldsymbol{s})\boldsymbol{F} + \boldsymbol{a}_r \tag{4}$$

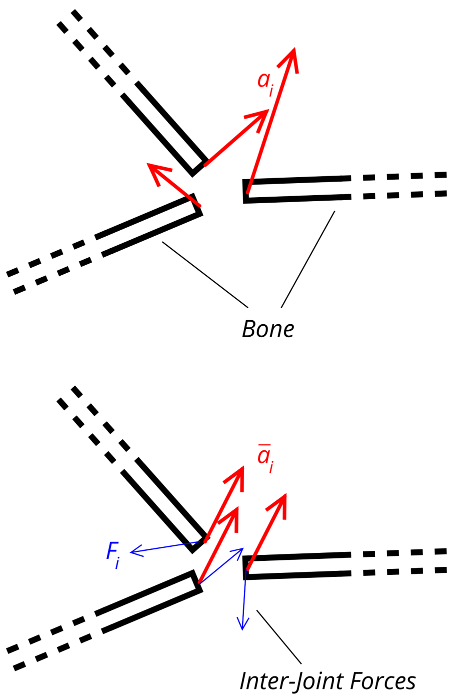


Figure 1: Annotated diagram illustrating the application of inter-joint forces

The above can be represented as an affine transformation to directly get the acceleration at a point $\boldsymbol{r}$ when a force is applied at the same point:

$$A(r) = T(\boldsymbol{a}_r)M(\boldsymbol{r}, \boldsymbol{r}) \tag{5}$$

Call this the force-acceleration matrix.

Suppose k bones $B_1, B_2, ..., B_k$ are connected at a joint. The following heuristic is used to ensure that they remain connected. Inter-joint forces $F_1, F_2, ..., F_k$ are applied the corresponding bone such that:

$$F_1 + F_2 + ... + F_k = 0 \tag{6}$$

and

$$\bar{\boldsymbol{a}}_1 = \bar{\boldsymbol{a}}_2 = ... = \bar{\boldsymbol{a}}_k \tag{7}$$

Where $\bar{\boldsymbol{a}}_1$ is the acceleration of B_i at the location of the joint. Let A_i be the force-acceleration matrix for B_i where $\boldsymbol{r}$ is the position of the joint. Then the acceleration constraint can be expressed as follows:

$$A_1F_1 = A_2F_2 = ... = A_kF_k = \boldsymbol{v} \tag{8}$$

Where $\boldsymbol{v}$ is some constant vector. Inverting each A_i, the term for the forces can be isolated:

$$F_i = A_i^{-1}\boldsymbol{v} \tag{9}$$

Then since the forces should sum to $\boldsymbol{0}$, the following relation is obtained.

$$\begin{aligned} &A_1^{-1}\boldsymbol{v} + A_2^{-1}\boldsymbol{v} + ... + A_k^{-1}\boldsymbol{v} \\ &\qquad = (A_1^{-1} + A_2^{-1} + ... + A_k^{-1})\boldsymbol{v} = \boldsymbol{0} \end{aligned} \tag{10}$$

Then $\boldsymbol{v}$ can be computed directly with the following:

$$\boldsymbol{v} = (A_1^{-1} + A_2^{-1} + ... + A_k^{-1})^{-1}\boldsymbol{0} \tag{11}$$

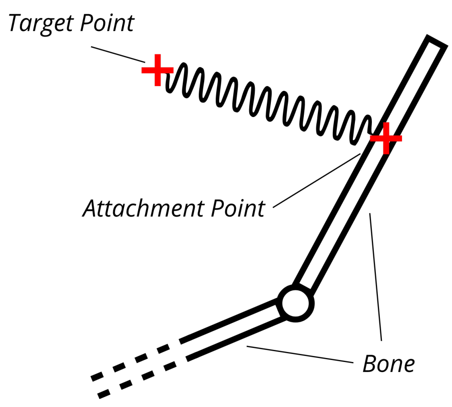


Figure 2: An illustration of the spring forces in a control point system.

Once $\boldsymbol{v}$ is computed, the inter joint forces can be computed with eq. 9.

This enables the computation of the inter-joint forces at a single joint after all other forces have been computed. This can be extended to simultaneously solve for all joints at once, but this would require the inversion of a large matrix. Iteratively computing the inter-joint forces at each joint approaches an approximate solution, sacrificing physical accuracy for speed and simplicity. This loss of accuracy can lead to divergent solutions, which are mitigated by control point movement interpolation in the following section.

The inter-joint forces are thus computed for each joint in multiple passes until the maximum difference in acceleration between any two bones at any joint has a magnitude less than some ϵ.

Control point springs. As stated before, a control point consists of a target point and an attachment point. The attachment point, which is on the rig, is brought to its target position (target point) by simulating connecting the two with rest length 0. Two types of springs were considered: linear, and capped-linear. The forces are computed as follows.

Suppose a is the position of the attachment point, B is the specific bone on which a is attached, and t is the target position. A force F is applied on B, where F is defined below for different types of springs:

- Linear: $F = -k(a - t)$
- Capped-linear: $F = -k\min(\|a - t\|, c)\left[\frac{a-t}{\|a-t\|}\right]$

Where k is the spring constant and c is proportional to the maximum force of a capped-linear spring. Linear springs have the advantage that large displacements cause larger forces spring forces that move the rig into place faster. However, because the forces are unbounded, there is a potential for large forces to cause the simulation to diverge. Capped-linear springs solve this issue by placing a maximum on the force that a spring can apply. Specific examples of simulations with both types of springs are in section 4 below.

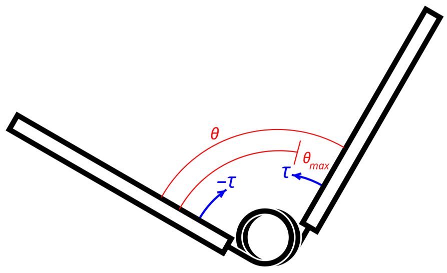


Figure 3: An illustration of a joint that exceeds the maximum angle defined by an angular constraint, and the simulated corrective torque.

Torsion springs. Torsion springs are used to enforce angular constraints. Define y as the axis pointing down the bone, and let x, z be the axes perpendicular to y and to each other. Note that these axes are local to the bone, and change as the parent bone moves or rotates. An angular constraint may be defined for any combination of these axes and it constrains the angle at the joint where the bone meets its parent to a certain interval around each axis.

Suppose the rotation at the joint some axis $\boldsymbol{v}$ is constrained to $[\theta_{min}, \theta_{max}]$, and the rotation about the $\boldsymbol{v}$ axis is currently θ. Then a torque $\boldsymbol{\tau}_v$, proportional to $\boldsymbol{v}$, is applied to the bone and a torque of $-\boldsymbol{\tau}_v$ is applied to the parent bone, $\boldsymbol{\tau}_v$ is defined below for different types of springs.

- Linear: $\boldsymbol{\tau} = L(\theta)\boldsymbol{v}$

  $$\text{where } L(\theta) = \begin{cases} -k(\theta - \theta_{max}) & \text{if } \theta > \theta_{max} \\ -k(\theta - \theta_{min}) & \text{if } \theta < \theta_{min} \\ 0 & \text{otherwise} \end{cases}$$

- Capped-linear: $\boldsymbol{\tau} = \min(L(\theta), c)\boldsymbol{v}$

Where k is the spring constant and c is proportional to the maximum force of a capped-linear spring. This $\boldsymbol{\tau}_v$ is calculated for each axis x, y, z to compute the total constraint enforcing torque of $\boldsymbol{\tau} = \boldsymbol{\tau}_x + \boldsymbol{\tau}_y + \boldsymbol{\tau}_z$, which is then applied to the bone, while $-\boldsymbol{\tau}$ is applied to the parent bone.

While capped-linear springs are more stable, they are not as good at enforcing angular constraints because they have a limit on how much torque they can output. Linear springs turn out to be stable enough while effectively enforcing angular constraints.

Damping. Damping was applied at the ends of each bone according to the usual $F_{damping} = -k\boldsymbol{v}$ where $\boldsymbol{v}$ is the velocity of the end of the bone. Angular velocity was damped by halving the angular velocity at each iteration.

3.3 Control Point Movement

When the user moves the target point, the actual position of the target point used in the computation of spring forces above is interpolated over a certain number of iterations, with which the rest of the simulation is run with the target point in its final position. Three schemes of interpolation were examined: linear interpolation, polar interpolation around the head of the bone, and polar interpolation around the center of mass of the rig.

For the following, suppose the target point was at a position $\boldsymbol{t}_{old}$ and was moved to the position $\boldsymbol{t}_{new}$. Also, suppose the number of iterations to run is i_{max}. Let $\boldsymbol{t}_i$ be the position of the target point used in spring force computation at the i-th iteration.

Linear Interpolation. The linear interpolation method simply moves the effective position of the target point in a straight line from $\boldsymbol{t}_{old}$ to $\boldsymbol{t}_{new}$ at constant speed. Then t_i be computed by the following formula:

$$t_i = t_{old}\left(1 - \frac{i}{i_{max}}\right) + t_{new}\left(\frac{i}{i_{max}}\right)$$

Polar Interpolation. This method moves the effective position of a target point in an arc around the origin of rotation. The origin of rotation, which could be the head of the bone of the center of mass of the rig, may change at each iteration. The angle between the interpolated target position and the new target position (with the origin as the pivot) is varied at a constant rate, while the distance from the interpolated target position is also varied at a constant rate until it matches the distance from the new target position and the origin.

This is formulated mathematically as follows. Let $\boldsymbol{o}_i$ be the origin of rotation at iteration i. Then $\boldsymbol{t}_i$ can be described by the following recurrence relation:

$$\begin{aligned} \boldsymbol{t}_0 &= \boldsymbol{t}_{old} \\ \boldsymbol{t}_{i+1} &= r_i R(\boldsymbol{n}_i, \alpha_i\theta_i)\frac{\boldsymbol{t}_i - \boldsymbol{o}_i}{\|\boldsymbol{t}_i - \boldsymbol{o}_i\|} + \boldsymbol{o}_i \end{aligned} \qquad (12)$$

where

$$\begin{aligned} \boldsymbol{n}_i &= \text{the vector normal to } \boldsymbol{t}_i - \boldsymbol{o}_i \text{ and } \boldsymbol{t}_{new} - \boldsymbol{o}_i \\ \theta_i &= \text{the angle between } \boldsymbol{t}_i - \boldsymbol{o}_i \text{ and } \boldsymbol{t}_{new} - \boldsymbol{o}_i \\ R(\boldsymbol{n}_i, \theta_i) &= \text{rotation by an angle } \theta_i \text{ about the axis } \boldsymbol{n}_i \end{aligned}$$

and

$$\begin{aligned} \alpha_i &= \frac{1}{i_{max} - i} \\ r_i &= (1 - \alpha_i)\|\boldsymbol{t}_i - \boldsymbol{o}_i\| + \alpha_i\|\boldsymbol{t}_{new} - \boldsymbol{o}_i\| \end{aligned}$$

To avoid dividing by 0, if either $\boldsymbol{t}_i$ or $\boldsymbol{t}_{new}$ is too close to $\boldsymbol{o}_i$, then the algorithm falls back on linear interpolation.

4 Results

The rig in figure 4 was the primary test case used to evaluate the performance of PhysIK on the objectives outlined above. It consists of 18 bones and 14 joints,

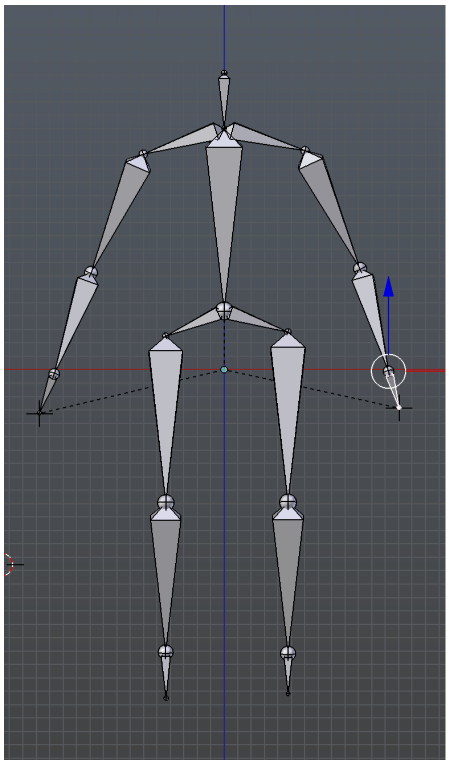


Figure 4: The rig used to evaluate the performance of PhysIK.

with a maximum of 4 bones around a single joint. The angle between the torso, the collarbone and hips are fixed with 0 degrees of freedom. The elbow and the knees have one degree of freedom and are constrained between 0 (straight out) and 120 degrees on the axis perpendicular to the length of the bone, and fixed on the axis parallel. The head, hip, and shoulder joints have 3 degrees of freedom and are unconstrained.

In each of the following examples, the Δt used to step the simulation was 0.05, with a damping coefficient of 20, and a joint acceleration epsilon of 0.001 (in arbitrary Blender units).

Accuracy. Figure 5 is a screenshot of a static pose created using PhysIK. The 8 control points used to

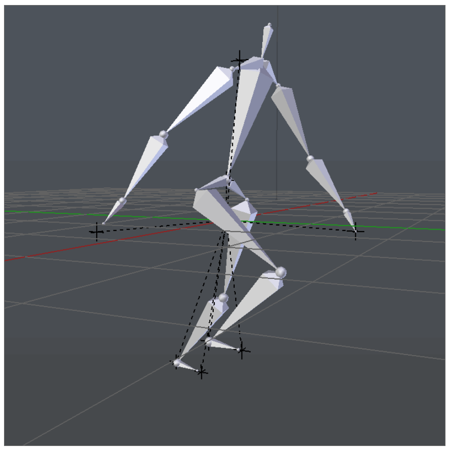


Figure 5: A static running pose.

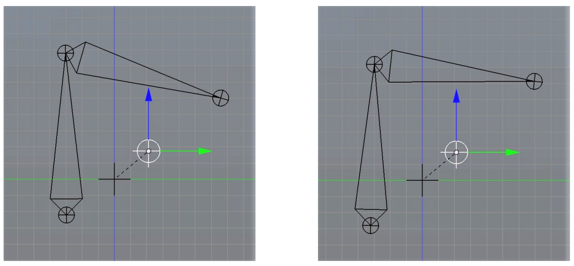


Figure 6: A comparison of linear (left) and capped-linear (right) springs on an elbow constrained to 90°.

create this pose all have spring coefficients of 30.

Observe that rig coincides with the control points on the right hand and on the feet. The control points on the left hand and shoulder are visibly out of place, but notice that they are too far apart to make a feasible pose. This highlights the fact that PhysIK can still produce good results even when the constraints are impossible to satisfy.

For a comparison of accuracy on linear and capped-linear springs, two examples were used. First, to evaluate the accuracy of angular constraints, consider figure 6. The rig in the figure is a simple elbow rig, where the angle between the bones is constrained between 180° and 90°. Note that the configuration defined by the control points is impossible to reach without breaking angular angular constraints. What is of significance here is the degree to which the angular constraints are broken. A linear spring puts more stress on the torsion spring enforcing the angular constraint than the capped linear spring would, resulting in a joint that is clearly overbent.

Another aspect of accuracy can be seen in the next example with linear and capped-linear springs. When the control point is instantly moved to a point far away, the simulation initially diverges for the linear spring, and converges to a strange pose, especially near the elbow and hips (see figure 7). When capped linear springs are used, the algorithm converges to stably, though it takes a few more iterations.

There are cases, however, where the algorithm fails to converge entirely. When angular constraints are stressed and one of the bones in the chain have enough degrees of freedom, then oscillation can occur around the control point. For example, in figure 8, the tip of the hand will oscillate around the target point indefinitely. This is most likely caused by the inaccuracies of Euler integration combined with the extra degrees of freedom that allows for such behavior.

Interactivity. The poses above were created starting with the pose in figure 4, then adding and manipulating control points. The pose in figure 5 was completed in under a minute, with the simulation converging than 3 seconds whenever a control point was moved to its target position.

When using the profile module in Python[10] to evaluate the speed of the algorithm, it reported an average of $\sim$ 98ms per frame for the humanoid rig, with $\sim$ 70ms spent on computing inter-joint forces, with around $\sim$ 20 passes per step.

The rig is simulated live while posing the rig, which makes it easier to see how the rig will move into place. It also leaves the overhead of deciding how many iterations to run to the user (i.e. it runs until the user deems that the system has converged enough). While this may not be ideal (see discussion), it also helps to hide the relatively long running time per step. When run at 100ms per frame, the tool still feels smooth enough and fast enough to use practically.

Intuitiveness. Intuitiveness is difficult to measure quantitatively and is up to the preference of the user. Presented below figure 9 are comparisons of Blender's iTasC solver with the PhysIK algorithm on a pose with a single outstretched hand. The angular constraints are the same in both cases, and both have 3 points to specify the position of the rig. Note how the shoulder in the PhysIK pose is turned toward the viewer, and the character is leaning slightly forward. As a bonus, the head also has a slight tilt. Compare this to the stiff pose computed by Blender's iTasC solver.

To see how control point interpolation affects the results, consider the following set of frames. Figure 10 compares polar interpolation about the center of mass and linear interpolation. The control point is interpolated for the first 50 steps (250ms) and the rest is simulated normally. Note how the linear interpolation causes more secondary motion while the circular arc, which would be a more natural path for the arm, results in a more natural looking pose.

As a final example, consider the following frames of a sitting animation created with PhysIK with only a single moving control point and 4 static control points figure 11.

5 Discussion & Future Work

PhysIK makes it easier to create poses from scratch, with interactive simulations and force-based posing making the who process of posing and animating more intuitive. By simulating external forces, PhysIK mimics the action of posing a wooden mannequin, resulting in natural poses with plausible secondary motion, even with a small number of control points. It does, of course, have its limitations.

The physical simulation in PhysIK can be sensitive to a change in the damping coefficient. If the damping coefficient is too large, then with the Euler's method integration used in this paper, any slight movement is over-corrected, and the resulting oscillations diverge, rendering the tool useless. Too small, and the simulation takes too long to converge. The ideal range of values for the damping coefficient depends on the scale of the model and the mass of each bone. A way to automatically compute the ideal damping coefficient or a different method of damping altogether would help to make the tool even more intuitive.

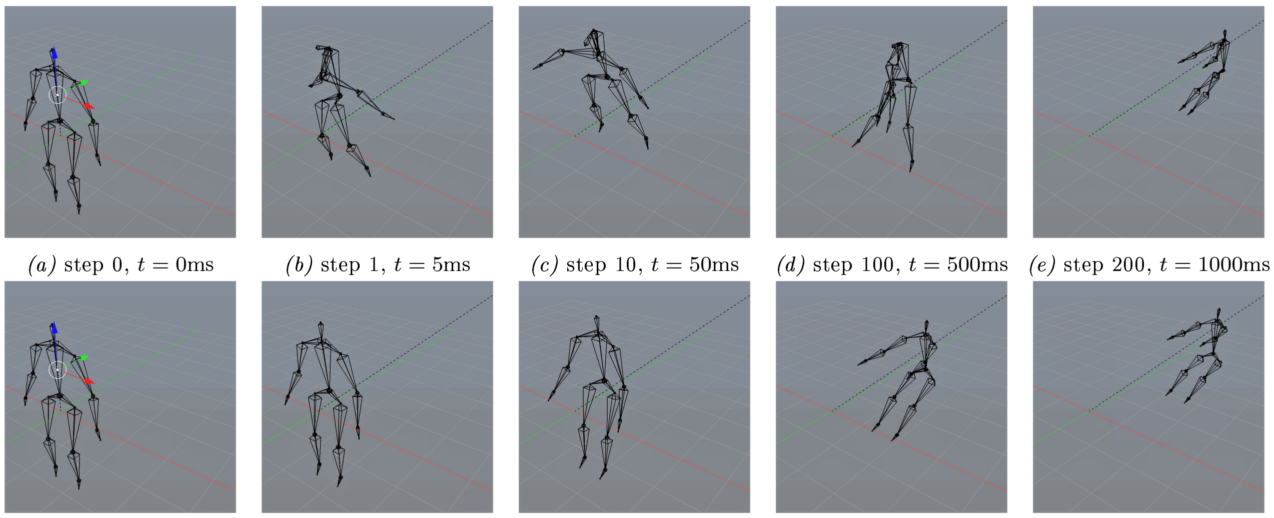

(a) step 0, $t = 0$ms *(b)* step 1, $t = 5$ms *(c)* step 10, $t = 50$ms *(d)* step 100, $t = 500$ms *(e)* step 200, $t = 1000$ms

Figure 7: A comparison of linear (top row) and capped-linear (bottom row) springs with a large, sudden displacement of a control point.

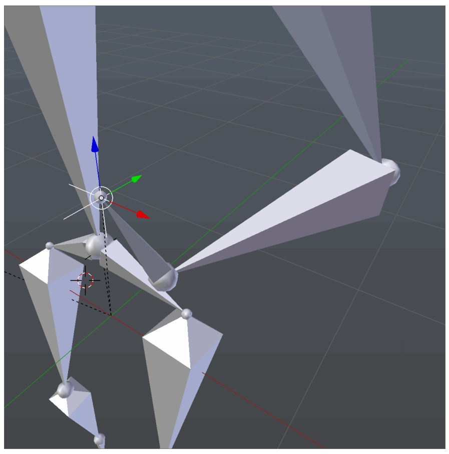


Figure 8: A configuration where the tip of the hand oscillates around the target point.

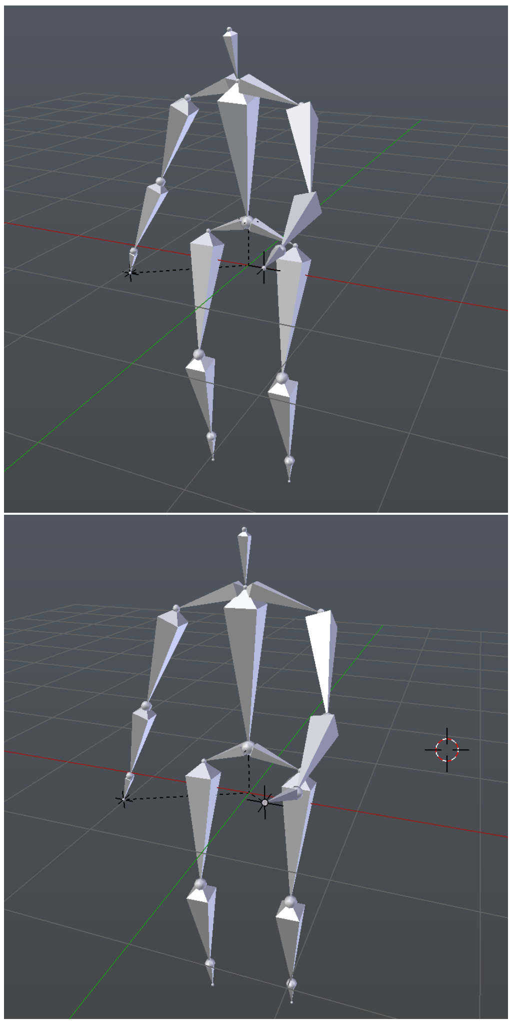
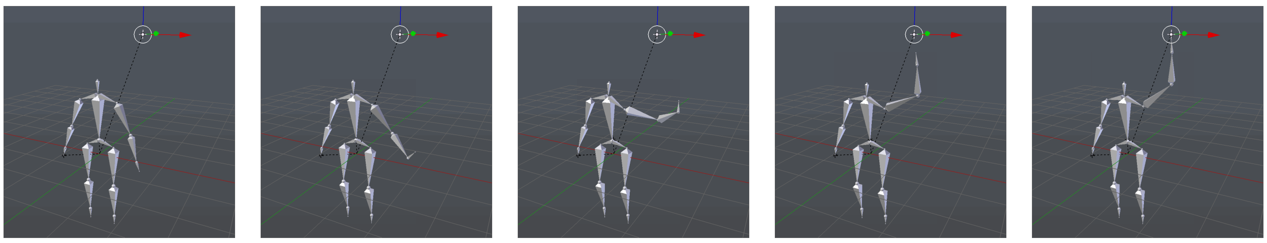


Figure 9: A comparison of PhysIK (top) and iTasC (bottom) on a pose with an outstretched hand.

PhysIK also breaks down when there are too many tight angular constraints which create a lot of torsion forces. For example, consider a hand, where the carpals are highly constrained in their relative rotations with the metacarpals, which are in turn highly constrained in their relative rotations to the wrist, or root node. Small perturbations tend to be amplified by errors in integration, and any attempt to modify the rig using control points causes the rig to explode. Perhaps better methods of integration or enforcing angular constraints can be explored to mitigate this issue.

PhysIK can also be slow. It may not be desirable to have to wait for the rig to converge onto its target points. In this case, it may be beneficial to pair the physical simulation with CCD or some other fast IK solver. The challenge here would be figuring out when to stop the physical simulation and switch over to the other solver.



(a) step 0, $t = 0$ms *(b)* step 20, $t = 100$ms *(c)* step 40, $t = 200$ms *(d)* step 60, $t = 300$ms *(e)* step 80, $t = 400$ms

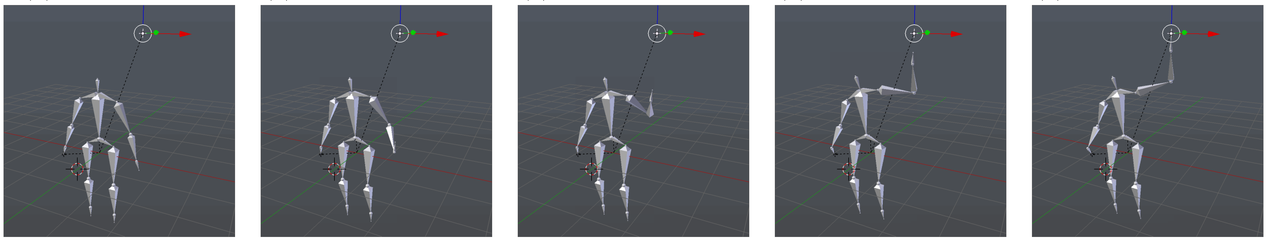
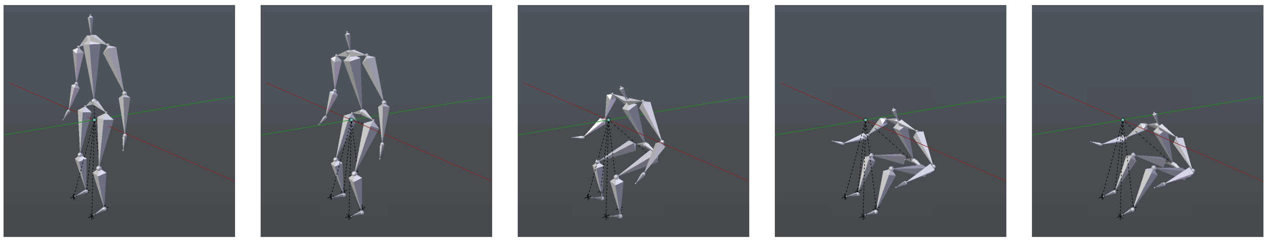


Figure 10: A comparison of polar interpolation about the center of mass (top row) and linear interpolation (bottom row) of the control point.



(a) step 0, $t = 0$s *(b)* step 40, $t = 1$s *(c)* step 80, $t = 2$s *(d)* step 120, $t = 3$s *(e)* step 160, $t = 4$s

Figure 11: Frames of a sitting animation

References

[1] L. . T. Wang and C. C. Chen, "A combined optimization method for solving the inverse kinematics problems of mechanical manipulators," *IEEE Transactions on Robotics and Automation*, vol. 7, pp. 489–499, Aug 1991.

[2] R. Smits, T. De Laet, K. Claes, H. Bruyninckx, and J. De Schutter, *iTASC: A Tool for Multi-Sensor Integration in Robot Manipulation*, vol. 35, pp. 235–254. 03 2009.

[3] "Itasc(IKParam)." Blender 2.77.0 - API documentation, https://docs.blender.org/api/blender_python_api_2_77_0/bpy.types.Itasc.html, 2019-04-02.

[4] R. Kulpa and F. Multon, "Fast inverse kinematics and kinetics solver for human-like figures," in *5th IEEE-RAS International Conference on Humanoid Robots, 2005.*, pp. 38–43, Dec 2005.

[5] A. Aristidou and J. Lasenby, "FABRIK: A fast, iterative solver for the inverse kinematics problem," *Graphical Models*, vol. 73, pp. 243–260, 2011.

[6] V. Zordan and N. van der Horst, "Mapping optical motion capture data to skeletal motion using a physical model," in *SCA 03': Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, (Aire-la-Ville, Switzerland), pp. 245–250, Eurographics Association, 2003. ACM Order No.: 428063.

[7] "Open dynamics engine." http://www.q12.org/, 2003.

[8] J. L. Cooper and D. Ballard, "Realtime, physics-based marker following," in *Motion in Games* (M. Kallmann and K. Bekris, eds.), (Berlin, Heidelberg), pp. 350–361, Springer Berlin Heidelberg, 2012.

[9] K. Pickl, "Rigid body dynamics: Links and joints," Master's thesis, University of Erlangen-Nuremberg, 03-09 2009.

[10] "The Python Profilers." Python 3.7.3 documentation, https://docs.python.org/3.7/library/profile.html, 2019-04-19.