Applied Inverse Kinematics for Bipedal Characters Moving on the Diverse Terrain

Łukasz Burdka, Paweł Rohleder
Techland Sp. z o.o.
ul. Żółkiewskiego 3, 63-400 Ostrów Wlkp, Poland
http://techland.pl

Abstract.

A solution to the problem of adjusting the pose of an animated video game character to the diverse terrain and surroundings is proposed. It is an important task in every modern video game where there is a focus on animated characters. Not addressing this issue leads to major visual glitches such as legs hovering above the ground surface, or penetrating the obstacles while moving. As presented in this work, the described problem can be effectively solved by examining the surroundings in real-time and applying Inverse Kinematics (IK) as a procedural post process to the currently used animation.

Key words: inverse kinematics, animations, visual improvements, physics, game systems

1. Introduction

The reception of modern video games relies highly on their visual quality. Such quality is achieved not only by graphics rendering techniques, but also by making sure that the elements of a virtual world fit each other and form a believable whole [1]. In most of the First Person Perspective (FPP) and Third Person Perspective (TPP) games there is a strong focus on animated characters inhabiting the world. Such characters traverse the world guided by human players or Artificial Intelligence (AI). They interact with the game environment, such as terrain or obstacles, using physics systems dedicated for games.

Physics prevents characters from colliding with the world geometry, but in order to operate efficiently they provide only a very rough approximation of the character's shape. Traditionally it is a capsule or an ellipsoid. While the entire character is moved by the physics system, the motion of its body parts is realized by the Forward Kinematics (FK) animations. FK animations provide a time varying position and orientation of each of the character's virtual bones in its local coordinate system (Fig. 1).

Since the bones form a hierarchy and the position of the hierarchy root is controlled by another system (physics or any other locomotion controller) the absolute position of each bone can be determined. However the main issue with this standard approach is that animations do not adapt to the character's surroundings. Although different animations can be played while walking on a pavement and on the stairs, it is virtually

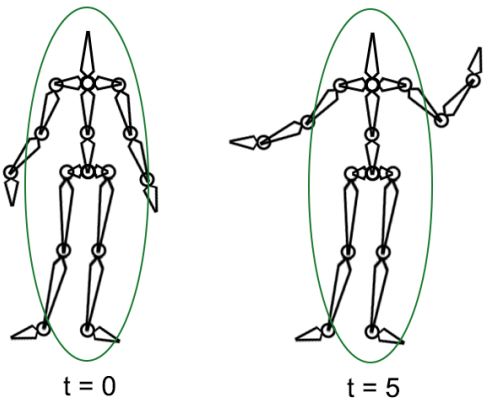

Fig. 1. Animated skeleton and its physical approximation.

impossible to prepare different versions of all animations for every possible terrain type. Obstacles lying on the ground pose an additional challenge to overcome. As a result, legs either hover above the terrain when they are above the bottom of the physical capsule, or penetrate obstacles when they move outside of it as a result of playing an animation (Fig. 2).

In order to tackle this issue, the character's surrounding would need to be examined and the animation would need to be adjusted. Further sections of this paper propose a solution of this problem by applying a trace-based and IK-based system. The overview of our method is presented in Section 2. Section 3 describes, in detail, how Inverse Kinematics works and how it is applied to an animated character's skeleton. In section 4 our work is summarized and further research directions are outlined.

2. Method overview

In video games, there is often a need to modify existing FK animations in real-time to fit the virtual environment where a character operates. Exemplary applications are adjusting the hand position while climbing, positioning hands on the steering wheel while driving, or adjusting feet to the terrain while standing and moving. In this paper we focus on the last case, since it is a very common problem in TPP and FPP games that utilize animated characters, regardless of the subject of the game. The general solution to all these issues can be divided into three steps. First, a position where the limb should be located needs to be found. Then IK can be applied to determine the correct configuration of the skeleton to match this requirement. Finally a blending between

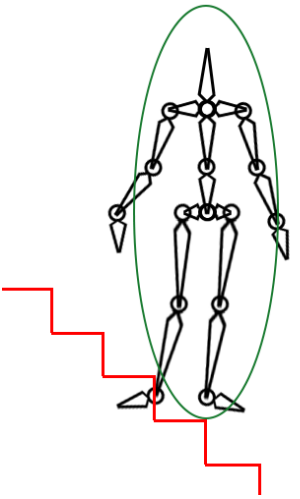

Fig. 2. One foot of the skeleton penetrates the stairs while the other hovers above them.

the FK and IK pose can be applied to make the animation look natural. The hardest problem in most real-life applications is how to find the desired position of the limbs?

When a human player looks at the game character, it is obvious for them where the legs should be, because they can see and interpret the whole rendered scene. This data is not as easily accessible and interpretable for game systems. Instead, the systems can use other means to examine the game environment in a certain location. Tracing is one of the most common tools used to analyze the geometry at the points of interest. Tracing is explained in Algorithm 2.1.

By tracing the surroundings of the character, the information how far the limbs can move, and where exactly they should be, can be precisely obtained. There are two sub-scenarios in which the correct configuration of legs needs to be determined. One of them

Algorithm 2.1 Level geometry tracing.

```
function trace(From, To, Geometry):
for all Piece in Geometry: do
    if Piece intersects Segment(From, To): then
        return Pair(IntersectionPoint, SurfaceNormal)
    end if
end for
return NoIntersection
```

is when the character stands still with both feet placed on the ground (static case) and the other is when the character is moving and at least one foot at a time moves above the ground (dynamic case). The static case is the simpler one. The solution for this problem is presented in Algorithm 2.2.

The terrain is traced vertically where each foot is located. When it is determined if each foot should be positioned above or below its FK pose, the Inverse Kinematics Equation (see Section 3) is applied for the bones of the leg to find the correct bone configuration. Thus IK poses for legs are obtained. If one of the legs was supposed to be positioned below the possible range of the leg, pelvis can be lowered to enable the pose. The dynamic case is more complex because it is not obvious where the tracing should be performed. The leg, which is on the ground, can be handled the same way as in the static case. However when the foot is in the air, its whole trajectory needs to be modified to provide believable movement without rapid changes of leg position. The modified trajectory should respect the altitude of the place where the foot is going to land on the ground. This position can be calculated using movement speed and the data about pacing extracted from the animation. Additionally the foot should be able to avoid obstacles crossing its trajectory so that the path it is going to traverse needs to be analyzed to find obstacles and modify the trajectory accordingly. The animations used with our system were manually annotated to mark moments when the foot lands on the

Algorithm 2.2 Determining leg configuration for the static case.

```
function positionLegsInStaticCase(FKSkeletonPose, Geometry):
LeftFootFK, RightFootFK, LeftCalfFK, RightCalfFK, LeftTighFK, RightTighFK,
        PelvisFK = getBonesFK(FKSkeletonPose)
LeftPosition, LeftOrientation = trace(over(LeftFootFK), under(LeftFootFK),
        Geometry)
RightPosition, RightOrientation = trace(over(RightFootFK), under(RightFootFK),
        Geometry)
LeftFootIK, LeftCalfIK, LeftTightIK = getConfigurationIK(LeftPosition,
        LeftFootFK, LeftCalfFK, LeftTightFK)

setOrientation(LeftFootIK, LeftOrientation)
RightFootIK, RightCalfIK, RightTightIK=getConfigurationIK(RightPosition,
        RightFootFK, RightCalfFK, RightTightFK)
setOrientation(RightFootIK, RightOrientation)
PelvisIK = PelvisFK
lowerPelvisIfNeeded(PelvisIK, LeftTightIK, RightTightIK)
blendPoses(FKSkeletonPose, LeftFootIK, RightFootIK, LeftCalfIK, RightCalfIK,
        LeftTighIK, RightTighIK, PelvisIK)
```

ground and is lifted into the air. The attempts to annotate the animations automatically gave poor results and led to many visual glitches. The simplified version of the algorithm used in the dynamic case is presented in Algorithm 2.3.

At first glance, the solution for the dynamic case looks as simple as for the static case, once the trajectory is obtained. The target of the trajectory is determined by tracing the predicted step location. Additionally, obstacles modify the trajectory if they intersect the predicted path (Fig. 3). The current IK position of the foot can be easily read from the trajectory. In both the static and dynamic case the foot position should be modified only vertically to avoid any modification of the animation's character. The problem, however, arises when the character changes the walking direction when the foot is in mid-air or modifies the walking speed. As a result, the trajectory needs to be reevaluated repeatedly when the foot is in mid-air. The new evaluated trajectory and the old one need to be blended smoothly to avoid rapid foot movement. In general, the ability to predict the position where the foot is going to land as soon as it takes

Algorithm 2.3 Determining leg configuration in the dynamic case.

```
function getTrajectory(CurrentStepPositionFK, Animation, WalkSpeed, Geometry):
NextStepFrame = getNextStepFrame(Animation)
NextStepPositionFK = getLegPosition(Animation, NextStepFrame,WalkSpeed)
RightPosition, RightOrientation = trace(over(RightFootFK),under(RightFootFK),
    Geometry)
LeftFootIK, LeftCalfIK, LeftTightIK = getConfigurationIK(LeftPosition,
    LeftFootFK, LeftCalfFK, LeftTightFK)
Trajectory = getAnimationTrajectory(Animation)
FinalPointIK = trace( over(Trajectory.end),under(Trajectory.end), Geometry)
modifyTrajectory(Trajectory.end, Trajectory, FinalPointIK)
for all substep in Range( CurrentStepPositionFK, NextStepPositionFK) do
   TracedPosition = trace(over(substep), under(substep), Geometry)
end for
AnimationPosition = getTrajectoryPoint(substep, Trajectory)
if isAbove(TracedPosition, AnimationPosition): then
   modifyTrajectory(substep, Trajectory, TracedPosition)
   return Trajectory
end if

function positionLegInDynamicCase( FKSkeletonPose, NewTrajectory, StepProgress):
PositionIK = getTrajectoryPosition( NewTrajectory, StepProgress)
//Apply analogically to the static case
getConfigurationIK(PositionIK, FKSkeletonPose)
```

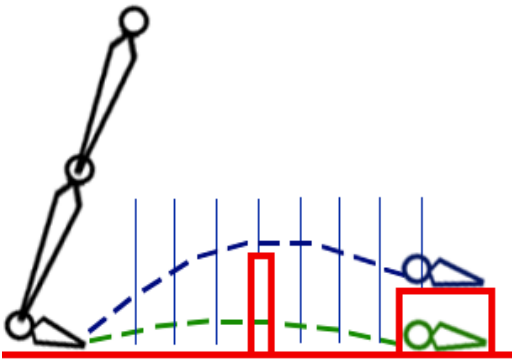

Fig. 3. The green dashed line is the FK trajectory. The blue dashed line is the IK trajectory modified by the vertical tracing of the FK trajectory.

off from the ground greatly improves the visual quality of applying Inverse Kinematics. Additionally, tracing the path only once can greatly reduce the computational power needed to obtain the correct trajectory. It is because in complex scenes the cost of precise trajectory tracing is much higher than the cost of calculating the IK equation itself and applying it to the skeleton.

3. Applying Inverse Kinematics

One of the most common algorithms for solving the IK problem is Cyclic Coordinate Descent (CCD) [2]. The main idea behind this method is to iteratively rotate each joint to move the effector closer to the target position. In case of CCD application for legs, the foot acts as an effector while the knee and hip form a two degree of freedom system [3]. The pseudo code for CCD algorithm is presented in Algorithm 3.1.

The algorithm converges fast for short bone chains but can be used for longer chains as well. It can be very effectively used for the case of finding the correct leg configuration of a bipedal character. The only challenge in this rather non-complex algorithm is to find the rotation angle for each joint. The sought angle and auxiliary vectors are shown in Fig. 4. To compute the angle, the following formulas must be solved:

$$\alpha = \arccos\left(\frac{d_T}{||d_T||} \bullet \frac{d_E}{||d_E||}\right), \tag{1}$$

$$r = \frac{d_T}{||d_T||} \times \frac{d_E}{||d_E||}, \tag{2}$$

where d_T is the vector to the target and d_E is the vector to the effector, and α is the angle, as shown in Fig. 4. The vector r is needed to determine the orientation of the angle α. Repeating these calculations until the effector's position error is below some arbitrary tolerance threshold leads to reaching the target location. The maximum number of iterations must be set to protect the algorithm in case the target is beyond the effector's reach. Additionally, in the unconstrained case, it would be possible to rotate the knee or hip beyond its anatomical movement range. Thus, IK constraints need to be applied to joints in order to limit the motion and allow the rotation only between certain angles. The only modification that needs to be done is to apply the clamp function when the angle is computed in each iteration for each joint.

Algorithm 3.1 Cyclic Coordinate Descent for series of joints.

```
function solveCCD(Joints, Effector, Target):
while isTooFar(Effector, Target): do
    //Start next to the effector and move to the last joint
    for all Joint in Joints: do
        ToTargetVector = Target - Joint
        ToEffectorVector = Effector - Joint
        RotationAngle = GetAngleToAlignEffectorAndVector(ToEffectorVector, ToTargetVector)
    end for
    GetAngleToAlignEffectorAndVector(ToEffectorVector, ToTargetVector)
    RotateJoint(Joint, RotationAngle)
end while
```

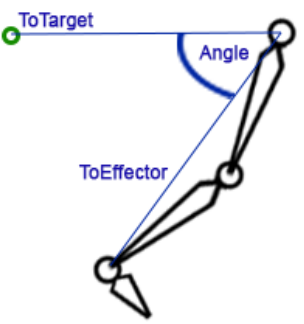

Fig. 4. Calculating the rotation of the hip joint, to move the foot effector closer to the target.

Once the correct pose is determined, it needs to blended with the FK animation pose to obtain a natural result. Parametric motion blending [4] is mostly applicable in this case, since there is often a need to control how much IK and how much FK should be used or to find a smooth transition between previous and current leg trajectory. In case of skeletal animations, where the pose is given by rotations of subsequent joints, the most effective way of blending poses is using Spherical Linear Interpolation (Slerp) [5]. For each blended joint the source and target quaternion must be known as well as the weight of blending. Given these three arguments, the sought angle of the joint can be calculated by solving the formula for Spherical Linear Interpolation between two joints with rotation given by quaternions q_1 and q_2:

$$\mathrm{Slerp}(q_1, q_2, w) = \frac{\sin\left((1-w)\cdot\alpha\right)}{\sin(\alpha)} q_1 + \frac{\sin(w\cdot\alpha)}{\sin(\alpha)} q_2 \tag{3}$$

where w is the blending weight and α is the angle between effector and target as introduced before.

4. Conclusion

Adjusting the legs of a bipedal animated character to diverse terrain is an important issue in modern video games. Failure to address this problem leads to glitches that decrease the overall visual quality of a game. Inverse Kinematics can be effectively applied to solve the leg configuration problem. Determining the correct feet positions for a moving character is not a trivial task. Tracing the scene's geometry allows to examine the character's path. The higher the speed of movement the more difficult it gets to produce a smooth and believable leg motion using the procedural post processing of IK. Frequent changes of movement direction make the step prediction inefficient and introduce the need of repeated trajectory reevaluation. At high speed, e.g. while running, it is recommended to gradually blend from the IK controlled pose towards the FK animated pose. It is because when legs move fast enough it is difficult to see if steps hit the ground precisely or hover above it. On the other hand repeated plan changes may lead to sharp and unnatural motion. While walking or standing, however, it is crucial to align the feet to the terrain and to make sure the legs do not penetrate obstacles while moving.

Acknowledgement

The paper was created as the result of the project "Industrial Research on the Technology of Scaled City for the Needs of Computer Games" (original title in Polish: „Badania przemysłowe nad technologią skalowanego miasta na potrzeby gier komputerowych").

The project is co-financed by the European Regional Development Fund under the Innovative Economy Operational Programme.

References

[1] C. C. Bracken and P. Skalski. Telepresence and video games: The impact of image quality. *PsychNology Journal*, 7(1):101–112, 2009. Retrieved Jan 10, 2017, from www.psychnology.org. Online: http://www.psychnology.org/File/PNJ7(1)/PSYCHNOLOGY_JOURNAL_7_1_BRACKEN.pdf].

[2] L. C. 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*, 7(4):489–499, Aug 1991. doi:10.1109/70.86079.

[3] R. H. Cannon. *Dynamics of Physical Systems.* McGraw-Hill, New York, 1967.

[4] A. Feng, Y. Huang, M. Kallmann, and A. Shapiro. An analysis of motion blending techniques. In M. Kallmann and K. Bekris, editors, *Motion in Games. Proc. 5th Int. Conf. on Motion in Games MIG 2012*, pages 232–243, Rennes, France, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-34710-8_22.

[5] K. Shoemake. Animating rotation with quaternion curves. *SIGGRAPH Computer Graphics*, 19(3):245–254, July 1985. doi:10.1145/325165.325242.