Inverse Kinematics in Computer Graphics using FABRIK

Christian A. Melendez Gallegos

University of Manitoba

Course: Computer Graphics 2

Course Instructor: John Braico

April 13th, 2019

Animation is a very important topic in Computer Graphics for the development of video games, simulations and even movies. There has been a lot of research to make it more manageable, easy and accurate, and because of that, there exist a lot of techniques involving animation. In this paper and its corresponding implementation, one of those is explored.

Before starting with the main topic, it is important to understand the topics of **Forward Kinematics** and **Inverse Kinematics**. Let's say we want to animate a robotic arm to grab a glass of water. To accomplish this task, we can simply hard-code or calculate the angle values for each part of the arm until we get a decent animation; easy but a lot of work. This process is called ***Forward Kinematics,*** the sum of all the transformation, rotation and translation equations for one object. This is similar to what we have already seen in class. However, if the glass of water is moving around, or the object is very complex, it is very hard to calculate the angles. That is where using **Inverse Kinematics** can be used. The concept is, give it the goal position and calculate all the angles from the last joint to the root. This helps a lot with accuracy and constant movement of goal position, and it is well used in other areas like robotics and medicine.

The problem for this Inverse Kinematics is that it is very hard to implement an algorithm to satisfy every problem like dealing with complex figures with a large number of joints and degrees of freedom (Aristidou, 2011). There exist a lot of methods to approach this problem, the most popular is using something called "the Jacobian inverse technique". But there exists another simpler and effective method called **"Forward and Backward Reaching Inverse Kinematics" (FABRIK)**

FABRIK method involves both inverse and forward kinematics. A hierarchy of the joints is used to facilitate iteration between parent-child joint angles. This helps a lot with knowing which parts of the body will be affected by the movement. After organizing all the joints, we are ready to apply our algorithm.

First, it is needed to check whether the goal position can be reached or not, given the limits of the joints. If the goal can be reached, the algorithm iterates first through all the joints, applying inverse kinematics from last joint (also called the end-effector) to the root, using the

goal position. After all the calculations are done, the algorithm will return new positions for the whole hierarchy of joints involved. The problem with this is that the original position will also be moved, which is something we don't want because if it was a robot, it will mean detaching the arm from the body. Therefore, to fix that, forward kinematics is applied from the first joint (using the original position as a goal). This process will be repeated until the end-effector is within an area very close to the goal position, this can be a constant (also called error) that can be set depending on how accurate we want the result. It is important to say that the final result is always an approximation, so this may not work if we require an exact calculation; the iteration could go forever, that is why a depth limit is also needed.

To have a more realistic movement, there can be added orientation constraints, which are the limits of rotations. This can be used for example when designing a human-like character and limit the angles to don't go more than 180 degrees of rotation, which is something that will look more natural.

In Aristidou's research, they did a comparison between FABRIK and CCD, Jacobian Transpose, FTL and more inverse kinematics techniques. With a single end-effector, FABRIK was able to find the result faster than the other methods. It had the lowest computational cost and produced the smoothest and most natural movements. FABRIK can also be used with multiple end-effector and goals, if for example, having a hand with different goals for each finger. This method also performed well against others.

In the code provided, there are two implementations, a simple arm reaching a point and a version of the running robot from the first assignment using FABRIK. There are some glitches because of the rotation matrices, but the algorithms work well enough. To move the goal position, simply use the numpad as the readme says. As you can see, the calculations are very reliable. For the running robot, I was planning to do a walking engine, but it resulted harder than expected because of other calculations. Like the goal positions inside a step. To see my implementation, go to https://github.com/chris-mega/FABRIK-project

Other stuff which is not implemented, but it would be nice to add to the walking engine, is to have some ray tracing intersections between the bottom of the foot and the floor to get

the goal point for one step. This will help a lot when dealing with an uneven terrain. In the GDC video in the references page, it shows some of the things developed by Ubisoft using not only joint position and angles, but also the details of the object to animate like weight, volume, and task to perform. Using artificial intelligence techniques, it will help to find more realistic goal positions for every movement in every scenario.

In conclusion, FABRIK is an effective and easy approach to do animation with inverse and forward kinematics. With only providing the goal position, all the angles can be calculated to reach that desired point.

References

Aristidou, A., Lasenby, J. (2011). FABRIK: A fast, iterative solver for the Inverse Kinematics problem. *Graphical Models, 73,* 243-260.

GDC. (2018, January 19). *IK Rig: Procedural Pose Animation*. [Video File] Retrieved from https://www.youtube.com/watch?v=KLjTU0yKS00