

CSCI 491 Project 2

IK-Rigging in three.js

Emily Palmieri

December 8, 2014

1 Introduction

Rigging is the process of preparing a model for animation by creating a skeleton, pivot points, and controls, collectively referred to as a rig. There are two common types of rigs: Forward Kinematic and Inverse Kinematic. Which type of rig is created for a 3D model or object hierarchy depends on the needs of the animation. With IK-rigs, an animator can set the position a child object, and the position and rotation of one or more parent objects is calculated automatically. This is opposed to FK-rigs where the animator sets the child's position by positioning and rotating its parents. IK-rigs are commonly used to animate the arms and legs of 3D-animated characters. It's often more convenient for an animator to move a character's hand to a specific position than to determine what combination of rotations in the arm joints will get the hand to that position.

Because of their usefulness, IK-rigs and tools for creating them are standard in 3D modeling and animation software such as Maya and Blender, but implementations of IK-rigs in three.js are near non-existent. In this research, an IK-rig for a simple human arm model is created in a three.js application. The hand of the arm can be positioned by the user with a GUI and the position of the elbow is determined using a specially designed object hierarchy, trigonometry, and matrix and vector operations.

2 Methods

The object hierarchy (shown in Figure 1) consists of two Mesh objects representing the upper and lower arm and three Object3D pivots. The arm parts are referred to as the Upper Limb and Lower Limb. One end of these two Meshes is placed at the calculated position of the elbow. The other end of the Lower Limb is placed at the position of the hand, defined by the user. The other end of the Upper Limb is placed at the shoulder. These are parented to an Object3D called Socket. The Socket is attached to the Object3D Limb, which is attached to another Object3D Root. These three pivots, located at the same position and initialized with the same rotation, represent the shoulder of the arm. The Limb and Root define the orientation and position of the Pointer, an axis defined by the vector from the pivots to the hand. The Socket rotates the Limb around the Pointer axis, the Limb rotates the Pointer axis around the Root, and the Root controls the x-, y-, and z-position of the object hierarchy.

Users position the hand with respect to the Root, but the position of the Upper and Lower Limbs is set with respect to the Socket. To translate from the Root coordinate system to the Socket coordinate system, the Limb is first rotated until the Pointer intersects the position of the hand with respect to the Root. Then, the elbow's position is calculated in the coordinate system defined by Socket. Figures 2, 3, and 4 show diagrams of the following calculations. The code can be found in `javascript/iklimb.js`

First, the orientation of the Pointer axis, defined by the angles between the Pointer and the x-axis on the xy-plane θ_y and the xz-plane θ_z , is calculated. The position of the hand is known (x_c, y_c, z_c) , so θ_z can be calculated directly:

$$\theta_z = \arctan \frac{z_c}{x_c}$$

Calculating θ_y isn't as simple. The total distance or the distance along the xz-plane from the Root to the hand will be needed. The result of

$$\theta_y = \arccos \frac{y_c}{x_c}$$

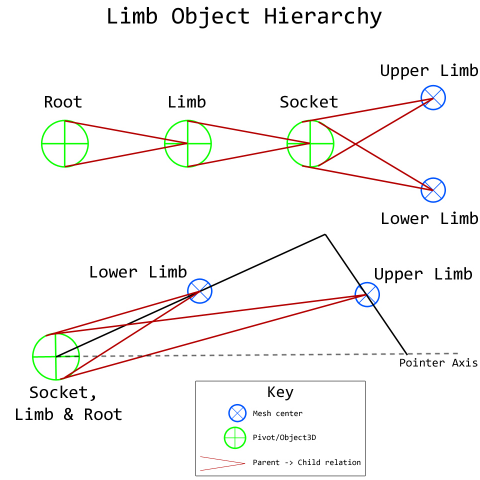


Figure 1: The hierarchy of objects used to define the position of the arm.

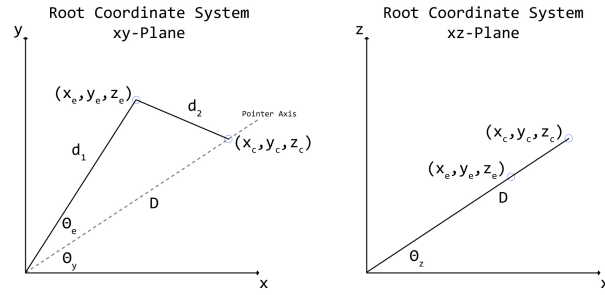


Figure 2: The position of the hand (x_c, y_c, z_c) and elbow (x_e, y_e, z_e) in the Root coordinate system.

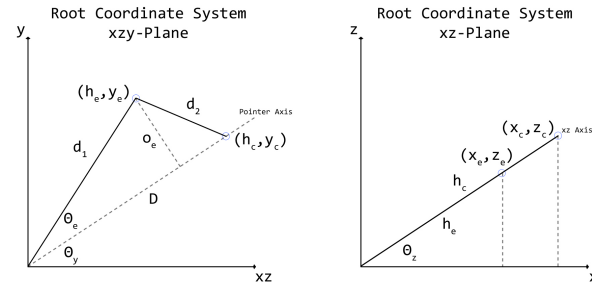


Figure 3: The position of the hand in the Root coordinate system translated into 2D.

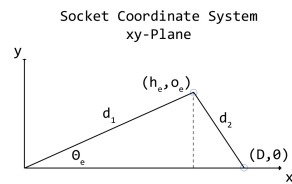


Figure 4: The hand and elbow positions in the Socket coordinate system.

won't be correct as the x component of the hand's position doesn't represent the distance along the xz-plane if the z-position is non-zero. This xz-distance can be calculated as:

$$h_c = \frac{x_c}{\cos \theta_z}$$

This result can be used to calculate

$$\theta_y = \arccos \frac{y_c}{h_c}$$

Next, the location of the elbow is calculated. To do this, the law of cosines is used to find the angle between the Pointer and the Upper Limb θ_e . Using this angle, the x- and y-position of the elbow with respect to the Socket can then be calculated. Before the law of cosines can be used though, d_1 , the length of the Upper Limb; d_2 , the length of the Lower Limb; and D , the total distance from the Root to the hand must be known. The lengths of the two limbs are known, and D is calculated as

$$D = \frac{h_c}{\cos \theta_y}$$

Now θ_e can be calculated:

$$d_2^2 = d_1^2 + D^2 - 2d_1D \cos \theta_e$$

$$\theta_e = \arccos \frac{d_2^2 - d_1^2 - D^2}{-2d_1D}$$

Using θ_e , the x- and y-coordinates of the elbow are calculated as:

$$h_e = d_1 \cos \theta_e$$

$$o_e = d_1 \sin \theta_e$$

Finally, the Upper and Lower Limb cylinder meshes are positioned based on the calculated position of the elbow and hand. This is done with a method created by Eric Haines called `makeLengthAngleAxisTransform`. Using matrix and vector operations, this method positions a given cylinder so that its top and bottom are at endpoints also passed to the method. The cylinder's axis is defined as the vector from the bottom to top position. A rotation axis perpendicular to this axis is calculated with the cross product of the cylinder axis and the y-axis. The cylinder mesh is positioned so that its center is an equal distance from the end points and rotated about the rotation axis until it reaches the desired position. This angle is calculated with a dot product of the cylinder axis and the y-axis. See [javascript/iklimbbuilder.js](https://github.com/iklimbbuilder.js) for the complete implementation.

3 Testing and Verification

Testing of this application was done visually. For troubleshooting and demonstration purposes, several guide lines and objects were included in the application. Grid lines are drawn on the xy-, xz-, and yz- planes. If the hand is placed on a grid line and moved parallel to it, the hand's distance from the line shouldn't change. This shows that when the user modifies, say, the x-position of the hand, the y- and z-positions remain static. This is correct behavior.

There are four guide objects that can be toggled on and off. The Cursor is a small, black sphere that shows the position the user has the controls set to with respect to the Root. The hand should always be at the position of the Cursor, or if the Cursor is outside the limit of the arm, the arm should point at the Cursor. The Pointer, a thin, black cylinder, is a visible representation of the Pointer axis. If the algorithm is implemented correctly, the hand should always be on the Pointer; when the arm is straight, the Pointer should be at its center; and the Pointer should always point to or intersect the Cursor.

The last two objects are the Inner Limit and the Outer Limit. The Inner Limit is a slightly transparent, black sphere positioned around the Root object. With a radius equal to the absolute value of the difference between the Upper and Lower Limb, it represents the area that the hand can't reach. The Outer Limit is a similar object that shows the maximum distance that the hand can reach, the arm's total length. If the implementation of the IK-rig is correct, the hand should be able to reach any point within the area between the two spheres but shouldn't be able to move outside the Outer Limit or inside the Inner Limit.

The IK-rig in this demonstration passes all of these tests and additional visual ones. The hand moves only in the dimension the user is currently manipulating. The hand and Pointer always point to or intersect the Cursor when it is

inside the Inner Limit, between the Inner and Outer Limits, or outside the Outer Limit. The hand never goes outside the bounds of the Inner and Outer Limits. In addition, there isn't a visual separation between the upper and lower arm at the elbow, a product of incorrect calculations. A jarring, but expected, jump in the arm's position occurs when the Cursor is placed within the Inner Limit or crosses the x-axis when its y- and z-positions are zero, but the arm's movement is fluid in all other cases. When the Socket is rotated, the elbow rotates around the Pivot axis as expected.

4 Analysis

The IK-rig in this demonstration is limited. Users move the hand, and by extension the arm, with respect to the Root. When the Root is moved, the arm moves with it. It is possible, and perhaps more common, to create an IK-rig where the hand remains at its set position even if the Root is moved. Suppose a character placed her hand on a table top. If she is animated realistically, her shoulder will likely not remain perfectly still while her hand is on the table. When her shoulder shifts, it is undesirable for her hand slide freely around the table top with it. The IK-rig implemented here requires that the Root remain perfectly still if the hand is to remain at its set position.

Implementing an IK-rig for an arm is also not unique. IK-rigging is a well studied area for its uses in 3D animation as well as robotics [7]. The algorithm implemented here is based on the 2D version of the problem and solution as explained by Ryan Juckett [3]. This method is simple as compared to more general and complex IK algorithms that can handle a chain of any number of objects.

Still, even for its limitations, this application demonstrates how to create an IK-rig in three.js, a feature rarely found in three.js examples. In researching this topic, only two WebGL-based implementations could be found. The first demonstrates tadpoles that have tails animated with IK-rigs, but it appears to be implemented in a custom-built WebGL application, not three.js [6]. The other demonstrates a three.js application where a skinned, humanoid model can be posed with an IK-rig [1]. The code, however, is undocumented and unreadable. There are a couple unanswered StackOverflow questions on the topic [2, 8] as well as a suggestion to add IK-rigging to an example [9], which also suggests that the problem is unsolved even for simple two-part chains.

5 Interpretation

Even though the rig created is more limited than desired, the goal of this research still was reached. An arm that is animated through Inverse Kinematics was created in three.js. Users can position the hand of the arm, and the position of the upper and lower arm responds as desired. It is hoped that the source code for this implementation will be useful to other three.js programmers.

In addition to creating an IK-rig where the hand's position remains static if the Root is moved, future work in this area could include implementing an IK-rig in three.js for animating a larger chain of objects. An IK-rig for a snake-like creature with many bones or meshes composing its body is an example of how such a rig would be used. If an animator moves the head of the creature in an S-shaped pattern, the rest of its body could follow the same or a similar pattern as the animation proceeded. Calculating the position of multiple objects in a chain like this could be done with algorithms such as Constraint Relaxation IK [4] or Cyclic Coordinate Descent [5].

An FK/IK-rig, an object hierarchy that can be animated using Forward or Inverse Kinematics, could also be created. Suppose an animator is given a humanoid character to animate. An IK-rig for this character would be useful for placing the hand at a specific point such as on an object. An FK-rig, however, might be more helpful for creating a walking animation. The character's hands don't need to be at specific points, but they need to rotate around the shoulder and elbow in a convincing manner. When this type of rig is implemented in applications such as Maya, the animator is given a control to toggle between the two types of rigs. A similar control would likely be included in a three.js implementation.

References

- [1] BVH Pose Editor. (n.d.). Retrieved from <http://www.akjava.com/demo/poseeditor/>
- [2] Hilton, W. (Oct 26, 2013). How can I ground a child object in Three.js?. Message posted to <http://stackoverflow.com/questions/19583689/how-can-i-ground-a-child-object-in-three-js>

- [3] Juckett, R. (2008). *Analytic Two-Bone IK in 2D*. Retrieved from <http://www.ryanjuckett.com/programming/analytic-two-bone-ik-in-2d/>
- [4] Juckett, R. (2009). *Constraint Relaxation IK in 2D*. Retrieved from <http://www.ryanjuckett.com/programming/constraint-relaxation-ik-in-2d/>
- [5] Juckett, R. (2009). *Cyclic Coordinate Descent in 2D*. Retrieved from <http://www.ryanjuckett.com/programming/cyclic-coordinate-descent-in-2d/>
- [6] Neuro Productions. (2011). *Tadpoles*. Retrieved from <http://www.neuroproductions.be/webgl/tadPoles/>
- [7] Oscar. (2012). *Inverse Kinematics for Hexapod and Quadruped Robots*. Retrieved from <http://blog.oscarliang.net/inverse-kinematics-implementation-hexapod-robots/>
- [8] pyrotechnick. (Aug 30, 2011). Robot Arm Example - Rotation about an Axis [Msg 5]. Message posted to <https://github.com/mrdoob/three.js/issues/472>
- [9] yaku. (Jan 20, 2013). Inverse kinematic animation. Message posted to <http://stackoverflow.com/questions/14421031/inverse-kinematic-animation>