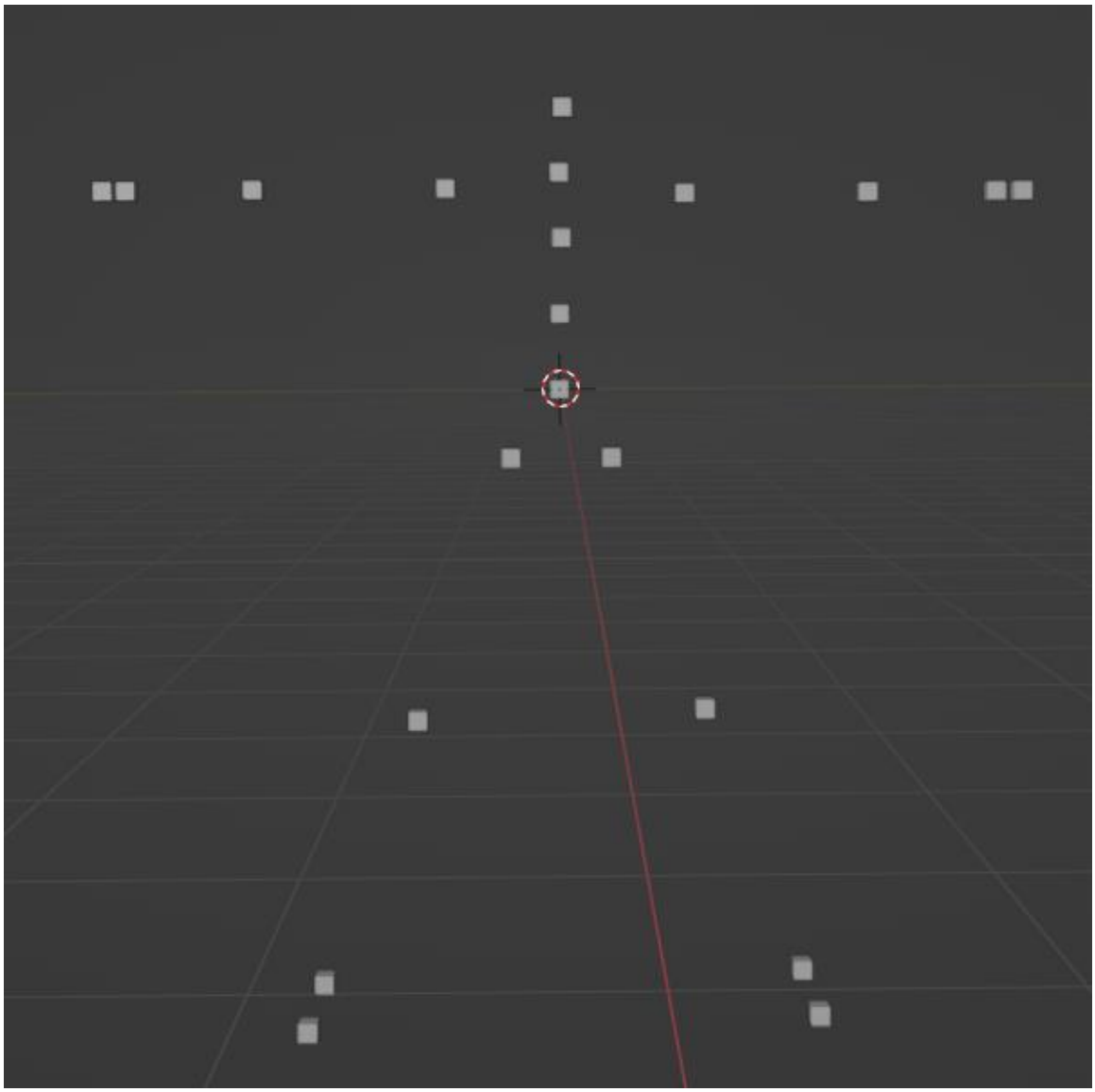
Name: Kwok Ka Yan UID: 3035705336

Introduction

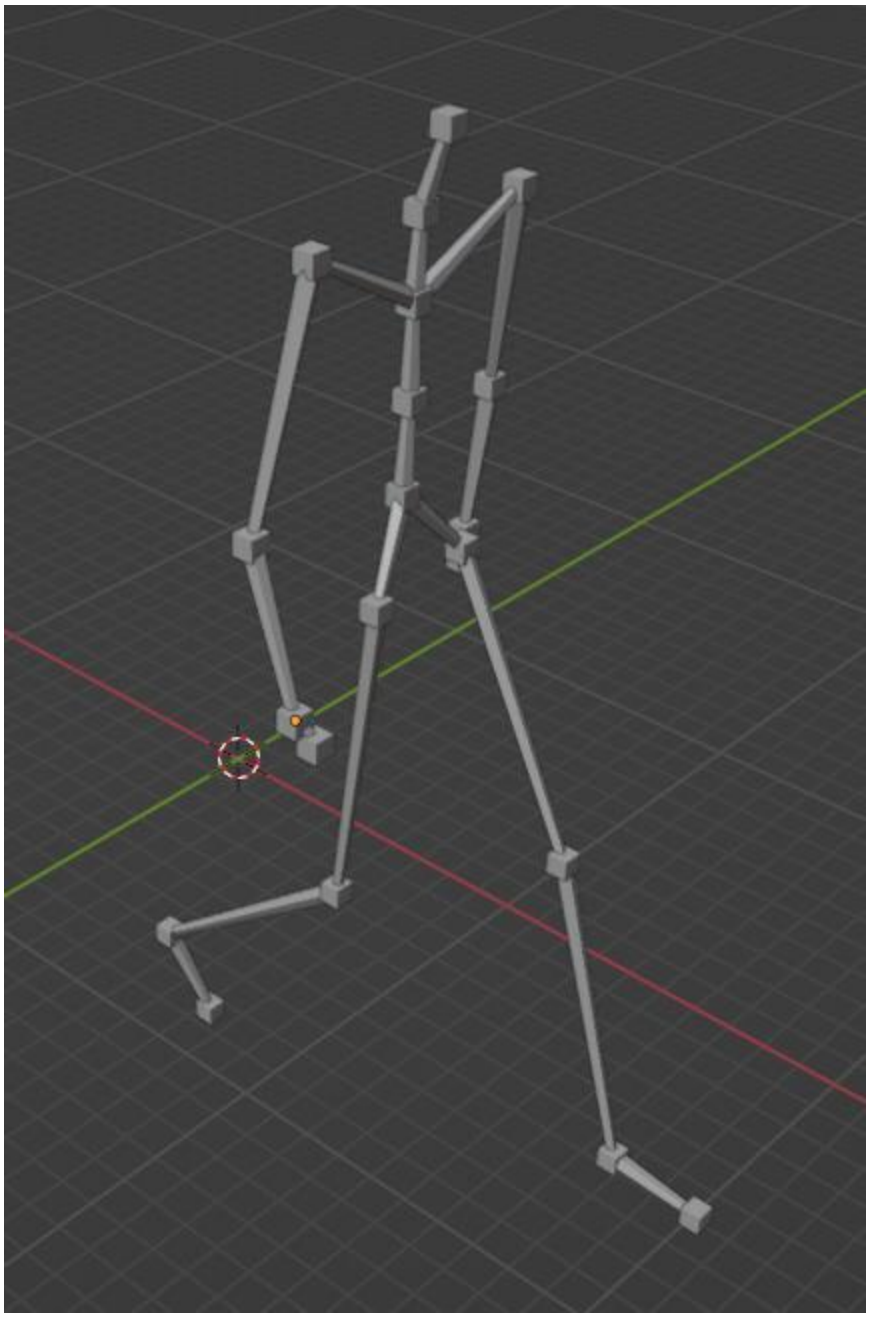
In this report, the implementation of the BVH visualizer and CCD IK solver will be discussed. Then, the open questions will be covered.

BVH Visualizer



build_joint_cubes()

For build_joint_cubes(), the cubes are created using the offset of each joint + the position of the parent joints. Using the provided parent list, the joint hierarchy is traversed in a preorder fashion, similar to build_reset_skeleton(). Each cube is then appended to a list for further processing.



set_cube_animation() for frame 203 in motion_walk.bvh

For set_cube_animation(), to achieve the correct position, each joint's offset (except for the root) is rotated with respect to all parents, ordering from the direct parent to the root, and is achieved by traversing up the hierarchy using the parent list. After rotating each joint, the parent's location is added to reach the final position for the key frame.

CCD IK solver

Iterations effect on performance

In the Unity Engine, update() is called every frame, with the time between frames being variable and affected by many factors, such as the rendering and processing of other scripts and game objects. Therefore, in the original SolveByCCD(), which is called by update(), the iterations will be completed every frame.

In terms of the visual performance, when the iteration value is low, one update() might not be enough to move the end effector to the IK target, with the next (few) update() continuing that rotation. In this case, the character would move to the IK target at a slower pace as more frames are required to finish the rotations. For a higher iteration value, the character would move quicker and almost immediately as fewer frames are needed in processing.

In terms of runtime performance, the higher the iteration value, the longer SolveByCCD() runs. Below is a table showing the time taken to update the CCDIK script from the Unity profiler on my machine (CPU: i5-9400f). With 5000 iterations, there are some stuttering of the character's movement (1ms = 1000fps, 32.8ms = ~30fps).

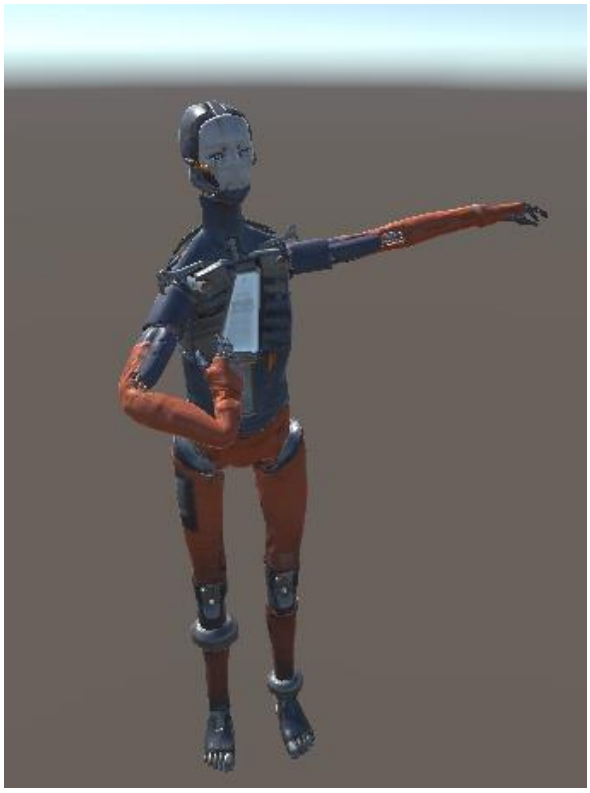
CCDIK.Update() processing times (with IK target being idle)

Iterations	Time (ms)
5	0.04
50	0.34
500	3.28
5000	~32.8

In terms of real application design, the iteration value should be chosen with respect to the type of application. For example, for real-time applications like video games, as many game objects are updated and rendered each frame, the iteration value should be set at a lower value (maybe around 50 - 100) to minimize the processing time while keeping the motion speed and quality acceptable. For 3D animation software, where the motion quality and control are more important, higher iteration values could be justified, especially for professional software as higher computer specs are expected.

Weight effect on visual performance

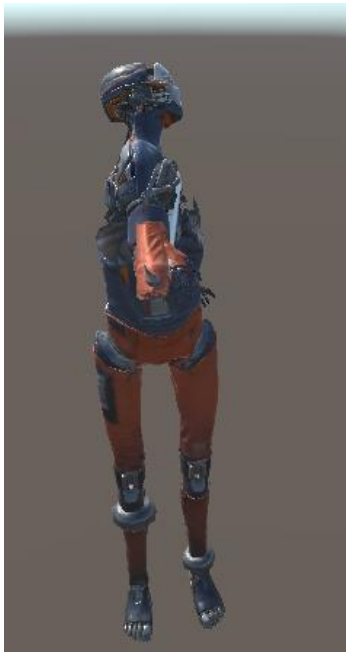
In the original SolveByCCD(), each iteration processes the joints in the order from root to the end effector. Using the Quaternion.Slerp(), the weight should be in the range of [0, 1], with 0 equals original rotation and 1 equals the new rotation. Below are the results of different weights with end effector = LeftWristSite, root = Chest4:



weight = (float)(j+1) / (float)chain.Length

weight = Mathf.Pow((float)(j+1) / (float)chain.Length, 2f)

weight = Mathf.Sqrt((float)(j + 1) / (float)chain.Length)



weight = 1

weight = 0

For the first 3 weights, the equation (float)(j+1) / (float)chain.Length provides a linear progression from almost 0 to 1, with the root joint rotating very slightly and the end joint rotating more directly to the IK target. For the first two, the resulting pose feels more natural as the main rotations are more emphasized on the elbows and wrists, rather than the body itself. The second pose emphasizes even more on the last joints by raising the equation to a power.

For the third one, however, the body moves too much towards the IK target due to the square root, which causes the elbow and wrists to rotate even further to compensate, resulting in an unnatural pose overall.

For the weight calculations, it should be biased against rotations for joints closer to the root while biased for rotations closer to the IK target, so that the end pose feels more natural. The first two weights achieve this goal well enough, and while higher powers can be used, more iterations would be needed when the IK target moves further from the root since it requires more iterations to rotate the upper joints. Therefore, a generalized Mathf.Pow((float)(j+1) / (float)chain.Length, power)) can be used.

Improvement of original CCD IK algorithm

(Implemented) One runtime improvement for the algorithm is the distance checking between the end effector and the IK target. If the end effector is very close to the IK target, it is unnecessary to start new iterations, thus reducing the computation time for cases when the IK target is reachable. As the algorithm keeps running every frame even when the IK target is not moving while the end effector has reached it, this checking can also improve the runtime performance when idling.

(Didn't Implement) One visual improvement is the constraining of the joint angles and degrees of freedom. The current algorithm only uses the weights to control how much each joint can turn towards the IK target at a time, which does not encompass the true rotational constraint for each joint. As such, the constraints can be defined for each joint and used in clamping the rotations. With more realistic constraints, the resulting pose would be more realistic.

Open Questions

Why rotation is more common? What's the problem if we only use joint position to record motion?

One reason why rotation is more common is that it can record information that joint positions cannot. For example, a root's orientation must be recorded in terms of rotation, which cannot be calculated from its position. Also, in a hierarchical body system, it is easier to find the position of the joints using rotations as they can be compounded, which is useful as a joint's position depends on all rotations from its parents. By only recording the joint position, the rotational information of a joint will have to be computed before applying it to its child joints, which would be quite time-consuming when multiple rotations are applied.

How about using IK in real-time gaming? Do you have any idea how to improve it?

In real-time gaming, it is important for IK to be quick rather than being high quality because there are other intensive computations executing concurrently like rendering. As such, the runtime performance can be improved by simplifying the computation. For example, the error values can be increased such that fewer computations will occur each frame, while the number of joints to be processed can be reduced by setting the root, end effectors, and other constraints.