Computer Animation & Gaming

Programming Assignment 2

Write a program that loads a character skeleton from a .skel file (described below) and display it in 3D. All joints in the skeleton will be 3-DOF rotational that rotate in x first, then y, then z. The program should perform the forward kinematics computations to generate world space matrices for the joints. The program should be able to load any skel file given to it. You can design your own skel file, and you may want to be able to load/display different files while you are doing your demo.

Skel File Description

The .skel file is a simple indented hierarchy of joints. Each joint also lists some relevant data about its configuration. You can use test.skel as well as dragon.skel which is a complex skel.

The .skel data file has 8 keyword tokens which are then followed by data (usually floats). These specify properties of a particular joint. Not all joints will specify all properties, so reasonable default values should be used. Here is a list of the keywords and their associated data.

offset	x y z	(joint offset vector)
boxmin	x y z	(min corner of box to draw)
boxmax	x y z	(max corner of box to draw)
rotxlimit	min max	(x rotation DOF limits)
rotylimit	min max	(y rotation DOF limits)
rotzlimit	min max	(z rotation DOF limits)
pose	x y z	(values to pose DOFs)
balljoint	name { }	(child joint)

The ‘offset’ is the constant positional offset to add to the local joint transformation. It represents the location of a joint’s pivot point described relative to the parent joint’s

space. This will almost always be specified for a joint, but if not, it should default to (0,0,0).

The ‘boxmin’ and ‘boxmax’ parameters describe the min and max corners of a box representing the bone to render for the particular joint. If these are not specified for a joint, they should still have a reasonable default, say (-0.1,-0.1,-0.1) and (0.1,0.1,0.1).

The ‘rotlimit’ tokens describe the rotational joint DOF limits. Note that all angles are in radians. These should default to an essentially unlimited state if they are not specified in the file (i.e., -100000 to 100000 would be fine).

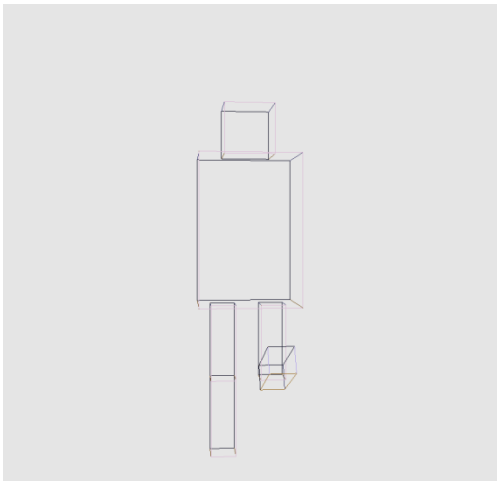
The ‘pose’ token specifies values to pose the DOFs at. Normally, data like this would not be needed in a skeleton file, as the skeleton is usually posed by a higher level animation system. For the purposes of this project, however, we will include optional pose data in the .skel file. By default, these should be 0. If the pose specifies values outside of the range of the DOF limits, then it should get properly clamped before displaying. Again, remember that these values are in radians.

The ‘balljoint’ token specifies a child balljoint which will have its own data and possibly its own children. There should not be any limit to the number of children allowed. Every balljoint has a name in the file, but you don’t have to do anything with this string if you don’t want to. Optionally, you could draw bone names as labels in the 3D view though…

In theory, one could extend this file format to include other joint types (hingejoint, prismaticjoint, freejoint, etc.). Each one of these joint types could have additional data tokens if needed.

Samples

Here is a sample rendering of how test.skel should look:



This is from a more or less straight view with the x-axis pointing to the right, y-axis pointing up, and z-axis coming out of the screen towards the viewer. Also notice that by default, the character is aligned with the viewer, therefore looking ahead, and facing away from us.

Note that the right knee is bent. This is because the right knee has values specified in the 'pose' field within the .skel file. Also note that these values are being clamped to within the DOF limits for the knee. The knee is able to rotate about the x-axis from –2 radians to 0 radians, but y and z are constrained to be locked at 0, effectively deactivating those DOFs, and making the knee a 1-DOF hinge joint. This is all controlled in the .skel file by specifying the DOF limits.

Here are sample renderings of how dragon.skel should look (viewed from top and side):