Gabe Aron

9/29/2015

Final Report

**Wham wearables – Skeletal Animation Framework**

**Purpose**

The Skeletal Animation Framework is designed to take positional data in the form of (X,Y,Z, body-part index) coordinates, which it uses to sequentially animate a human-form mesh on the user’s screen. The X, Y and Z coordinates correspond to movement in the x, y and z directions in three-dimensional space, respectively. The body-part index corresponds to one of fifty-five unique body parts of the human mesh. The combination of (X,Y,Z, body-part index) coordinates moves the body part specified by the index in the x, y and z directions according to the double floating-point values of X, Y, Z and body-part index.

The insertion of (X,Y,Z, body-part index) coordinates into one or more FIFO structures enables the figure to move one or more body parts in sequence over time and/or concurrently, depending upon the needs of the user. Each FIFO object parses through twenty sets of coordinates at a time, rendering each X,Y,Z change on the mesh’s correct body part, according to the body-part index, in sequence over time. Similarly, multiple FIFO objects render the X,Y,Z changes of each index in parallel sequence to one another.

Say, for example, that the user adds values to two FIFO structures. Say again that the user specifies a right arm movement in the first index of the first FIFO structure and a left arm movement on the first index of the second FIFO structure. This in turn will move the right and left arms of the animated mesh simultaneously. Say that the user specifies a right arm movement of the first index of the first FIFO and a left arm movement of the second index of the FIFO. This will move the right arm and then the left arm of the mesh.

The user can create as many linear sequences of movements that he or she wants. The user can also create a total of fifty-five (the number of unique body parts) parallel sequences of movements. Parallel and linear sequences of movements can be run together (i.e. the mesh can move its right arm and then its left arm, while also moving its right leg and then its left leg, all the while moving its head, etc).

Once all twenty FIFO indices have been used, the structure “wraps around”, allowing for an indefinite number of possible insertions for each FIFO structure.

This system is designed to interface with the positional data presented it by a team of developers within the same project who will convert wearable device data into positional data. This data will in turn be fed into the system in a continuous and automated way.

The intention of this system is to work as an interface for the positional data, which, combined, will enable a user with wearable devices attached to his or her person to render the animation properties of the mesh according to his or her own movements.

**Methods**

The Skeletal Animation Framework utilizes three components: a Collada mesh file created using Blender, Objective C, and CoreAnimation (a library of Objective C).

The Collada mesh was designed and rigged in Blender, with a skeletal structure enabling the body to move each part of its bone structure programmatically.

This system itself was written in Objective C – a programming language common for creating iOS applications. Within the Objective C framework, I created a simple structure of two-dimensional arrays. This array is accessed and manipulated by the class MovementQuards, which treats the array like a stack data structure. This means that each element is accessed one-at-a-time by two index markers, one of which is incremented with insertion, and the other of which is incremented when the current element is read.

There are 55 invoked objects of MovementQuards, corresponding to the 55 unique body parts of the mesh. As values are added and read from each object, the body moves in linear sequence over time according to the double floating-point data of each element (X,Y,Z, body-part index). Objects that are executed in parallel sequence will run simultaneously to one another (each element of each separate object with the same index being executed at the same time).

The scene for the mesh was set up and rendered in SceneKit, a library within Objective C, which can work in tandem with CoreAnimation to render and animate Collada meshes.

The program begins by allowing the user to insert (X,Y,Z, body-part index) data sets into the current writing element of any or some combination of MovementQuards objects. The mesh is then rendered in the AnimationController class, which also animates the mesh according to the values of each data set. The method of the AnimationController class, animate, reads the values of each MovementQuards object at a time, and, using CABasicAnimation, places sequential movements of like body parts (right arm and right arm, say) into unique CABasicAnimation groups. These groups, in turn, are allocated as two dimensional arrays , with the fist index slot being for the current MovementQuards object to be divvied up into groups, and the second index slot being for the number of sequences for each unique body part (a number which goes up to twenty). The number of sequences for each unique body part specifies how many times one body part may move per twenty (X,Y,Z, body-part index) elements. Setting up groups in this fashion is necessary because CABasicAnimation does not enable the same mesh layer to move without inserting in into a CABasicAnimation group.

After all fifty-five sets of twenty CABasicAnimation groups have been created, each group is run in sequence. This creates the effect of allowing for linear sequences per MovementQuards index and parallel sequences per MovementQuards object.

**Challenges**

The greatest challenge that faced me was learning how to enable like body-part indices to animate using CABasicAnimation. CABasicAnimation does not allow for individual animations to work of the same layer (body part) by use of individual animations. Animations of like body-part indices must, therefore, be arranged into CABasicAnimation groups. Doing so was challenging because each sequence of (X,Y,Z, body-part index) elements may contain like body-part indices in any part of the sequence. Handling this difficulty entailed that a control-flow structure be implemented that assigns each element with a like body-part index into a unique group. This challenge was accentuated by the need to assign begin times and durations of each assignment in each group to run at the point of the sequence in which it was intended in the initial MovementQuards object.

This problem was further complicated by the fact that each unique group had to be uniquely named in order to animate properly. This was solved by created a two-dimensional array of CABasicAnimation groups with the number of movements per like body-part index sequence in one index slot, and the number of total MovementQuards objects in the other.

Ultimately, although with much effort, I was able to create a sequence of CABasicAnimation groups that runs animations in both linear and parallel sequences on one or more body parts.

The second most challenging problem related to providing a “wrap around” capability for each MovementQuards object. This was challenging because animations cannot render a mesh in CoreAnimation until every animation and group is set. This means that, while I could revert both index markers of the MovementQuards objects to zero once they reach the twentieth element, I had to somehow rerun the animation immediately before the “wrap around”.

Ultimately, I leave the solution of this problem to the imagination of the user. The idea is to reset the stack of the AnimationController object, and then run the animation after each twenty insertions. As far as the MovemntQuards object is concerned, it “wraps around” after the stack has been completely animated. This means that it clears itself after one to twenty of its elements have been animated.

**Outcome**

The current Skeletal Animation Framework is capable of:

1. Accepting (X,Y,Z, body-part index) positional data from outside sources via MovemtnQuards FIFO objects.
2. Reading these objects to the effect of animating a pre-defined, rendered Collada mesh.
3. Invoking up to fifty-five instances of MovementQuards, enabling up to fifty-five parallel sequences of movements.
4. Adding an indefinite number of (X,Y,Z, body-part index) positional data elements, enabling an indefinite number of linear sequences of movements over time.

The current Skeletal Animation Framework is not capable of:

1. Automatically feeding positional data into the MovementQuards objects from an outside source (this algorithm must be implemented by the developers who intend to use this system as an interface).
2. Constrain the Collada mesh to specified ranges of motion. (Since the primary data will come from the positional data provided by the wearable devices, this will not matter).
3. Necessarily translate positional data from any form into meaningful or sensible animations.

**Recommendations – Going forward**

The Skeletal Animation Framework is primarily intended for use by other software. It does not function well as a “stand-alone” system. It acts as a convenient interface between programs that continuously feed it data and visual effects on a user’s screen.

For its current application, this system is meant for use by the acceleration-to-position software that is currently under development. The data feed of this software will have to adhere to the convention of the (X,Y,Z, body-part index) format to communicate with this system properly. Also, the developers of the acceleration-to-position software will have to implement an algorithm for feeding data into the correct element of the correct MovementQuards object.

The Skeletal Animation Framework is a system with many prospects. Applied to the acceleration-to-position software, it can be used for animatronic technology, and behavioral/athletic applications. Applied to videogames, it can provide an easy interface for making a rendered character move based on control inputs. One can also create ones own sequence of animations to experiment with different forms.

Applied to a mesh other than that of human form, one could use input controls to create physics simulations and crash-tests. Of course, the system would have to be modified to adhere to the new mesh, but the movement algorithm would remain the same.

Overall, the possibilities are vast, provided that the correct tools are created to interact with the system.

**Resources**

Download xCode: <https://developer.apple.com/xcode/download/>

Download Blender: <https://www.blender.org/download/>

Learn about SceneKit: <http://code.tutsplus.com/tutorials/an-introduction-to-scenekit-user-interaction-animations-physics--cms-23877>

Learn about Core Animation: http://www1.in.tum.de/lehrstuhl\_1/home/98-teaching/tutorials/509-sgd-ws13-tutorial-core-animation