Modeling the Human Skeleton

An exercise in Hierarchical Modeling

Written by: Michael Sault - 8459820
Supervised by: Dr. Wonsook Lee

CSI4103 - Winter 2020
19 April 2020

Table of Contents

1.0 Abstract

Human modeling is a complex project that relies on a number of other concepts including 3D modeling and hierarchical modeling. Human modeling is used in a number of fields, including animation, game design, visual arts and even the medical field. The goal of this course was to explore human modeling and it's implementation as well as the concepts behind it and how it is used in fields outside of computer science. It is important to understand key concepts such as 3D processing, hierarchical modeling and transformation matrices in order to develop not only human modeling projects but many other 3D modeling and animation projects as these concepts are core concepts for so many aspects of these fields.

2.0 Introduction

This semester I started work on a research course with the purpose of learning the basics of human modeling. The end goal of this course was to complete a final project in which I would create a humanoid skeleton that accepts and visualizes motion data. In order to accomplish this goal, it was necessary to learn a number of core concepts including drawing in 3D, transforming and animating 3D objects and hierarchical modeling. Over the course of this semester I researched and worked with a large number of concepts ranging from basic drawing in 3D, to animating 3D objects, to hierarchical modeling and even parsing and working with motion capture data. This report will outline the process I underwent while learning these concepts and how I applied them to the final project while building a human skeleton that responds accurately to motion capture data.

3.0 Concepts, Features and Implementations

3.1 Drawing in 3D

While I had some experience working with Processing in the form of the p5.js library, I had not worked directly with the java version of Processing, nor did I have experience working in a 3D environment. Therefore, the first step in the learning process was simply to attempt to become familiar and get used to working objects in a 3D environment. This includes everything from drawing, translating and rotating objects in 3D space, to more complex concepts like ray casting.

These concepts were obviously taken and used later in the development of the final skeleton as each bone must be drawn on the canvas in the desired starting position, to be later manipulated. While ray casting wasn't specifically used in the final project, was an invaluable testing and debugging tool as it allowed for the ability to pick which object should be manipulated, or which should return values for use in debugging.

3.2 Interaction

The next step in the learning process was to attempt to implement motion concepts and build upon the drawing concepts from 3.1. In order to accomplish this, a number of examples were considered from the interaction category of examples on Processing.org. These examples were provided in 2D space, and during this part of the learning process, they were re-implemented to work in 3D space.

The first example that was considered was called Follow1. It was designed to demonstrate to a new developer how to have an object follow the end user's mouse cursor. The 3D implementation still had the object follow the user's cursor, but also allowed for the object to rotate and move along the z-axis.

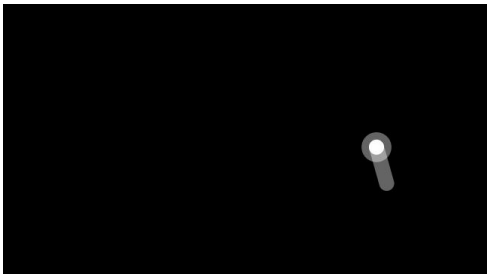

Figure 1*: 2D Implementation of Follow1 from Processing.org*

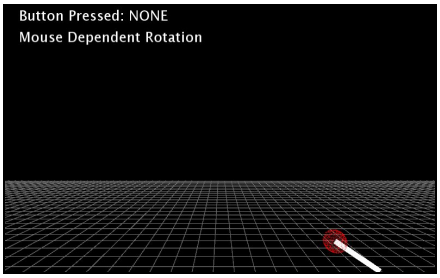

Figure 2*: 3D Implementation of Follow1*

The second example that was considered was called Follow3. It was similar to Follow1 in that the lead object follows the user's mouse cursor, but in this example there is also a trail of similar objects following the path of the first object. The 3D implementation once again had the lead object follow the user's cursor, only this time, the scroll wheel could be used to move the lead object along the z-axis.

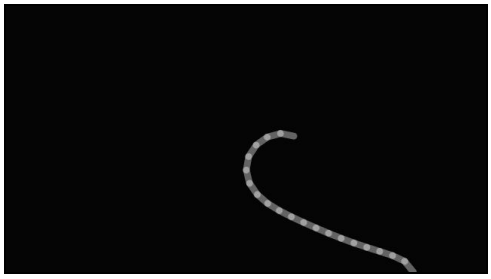

Figure 3*: 2D Implementation of Follow3 from Processing.org*

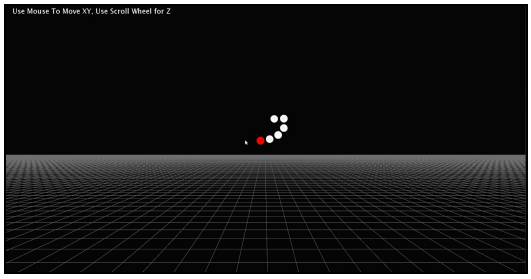

Figure 4*: 3D Implementation of Follow3*

The final example that was considered was called Reach2. It was functionally the same as Follow3, only with one end of the path anchored at a base point. The 3D implementation was a slight alteration of the 3D Follow3 implementation, but again, the end not following the mouse cursor was anchored at the center of the screen.

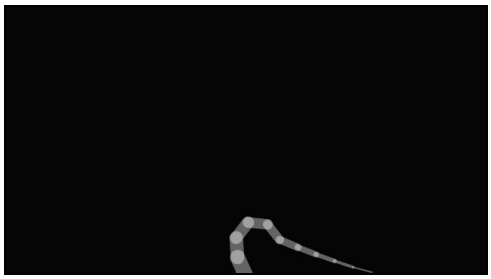

Figure 5*: 2D Implementation of Reach2 from Processing.org*

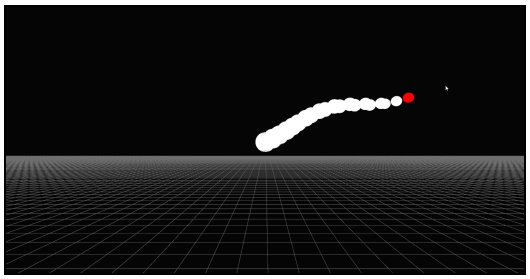

Figure 6*: 3D Implementation of Reach2*

3.3 Motion

Having the interaction examples successfully implemented in 3D space, the next step was to consider the examples in the motion category. While the interaction examples provided us with experience directly manipulating objects based on the mouse cursor, re-implementing these examples in 3D was an exercise in having objects move independently of the user and interact with other objects that they may collide into on their trajectory. Like the interaction examples from the Processing.org site, these examples were provided in 2D space, and during this part of the learning process, they were re-implemented to work in 3D space.

The first example that was considered was the Bounce example. Basically, it had an object move and bounce off of the edge of the screen in 2D space. In the 3D implementation, there was a 3D sphere positioned inside a wireframe cube. It was then set on a random trajectory and the program detected when it collided with one of the walls, at which point the trajectory was reversed and the sphere changed direction.

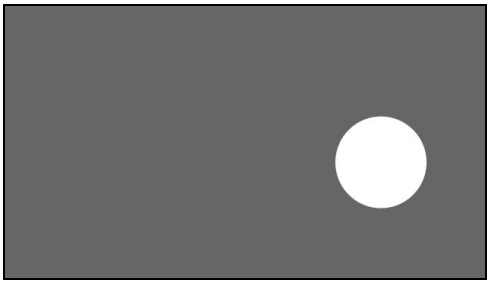

Figure 7*: 2D Implementation of Bounce from Processing.org*

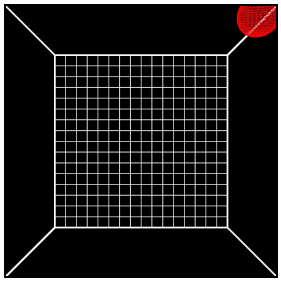

Figure 8*: 3D Implementation of Bounce*

The second example was called Circle Collision. It was an improvement on the bounce example, only with multiple objects that detected when two or more collided and changed direction accordingly. In the 3D implementation, there were three spheres, and the program detected when two or more spheres collided and their trajectory would then be reversed in the direction corresponding to the side of the sphere the majority of the collision took place on. Meaning if the spheres collided on the x-axis, the velocity on the x-axis would reverse, while the y-axis and z-axis would remain consistent.

Figure 9*: 2D Implementation of Collision from Processing.org*

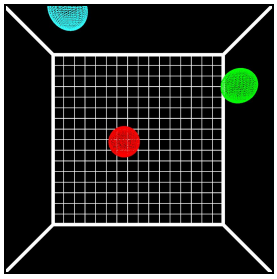

Figure 10*: 3D Implementation of Collision*

The final example was called Brownian. This example drew a line in 2D space, based on a randomly generated length and angle. The 3D implementation made a simple change and allowed the line to move in 3D space by randomly generating an additional value to define the angle the line will move on the z-axis. The peasycam library was used for this example in order to allow the end user to move the camera in 3D space to see the line's movements along the z-axis, as otherwise it may look very similar to the 2D example.

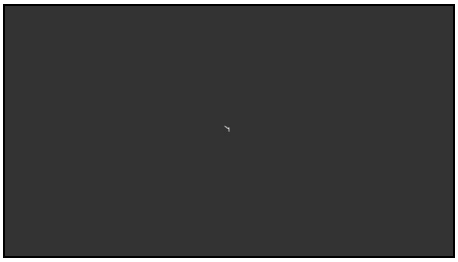

Figure 11*: 2D Implementation of Brownian from Processing.org*

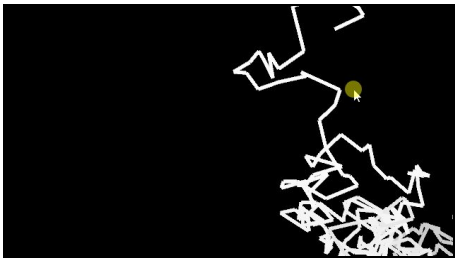

Figure 12*: 3D Implementation of Brownian*

3.4 External Forces

The next step in the learning processes was to consider external forces. Previously, all the examples focused mainly on motion in a vacuum, however to add a sense of realism we need to consider external forces like gravity.

For this implementation the bounce example was used as a starting point. The only difference is that we don't just want to reverse the velocity of the sphere when it collides with something, we also want to decrease the velocity of the sphere in the opposite direction of the external force. This should eventually cause it's trajectory to arc and begin moving in the opposite direction, similar to what would happen if you were to throw a baseball straight up in the air.

The option to have an external force apply on multiple axis was also implemented, as well as the option to change the direction of the force at will. With multiple forces acting on an object, the change in velocity and trajectory works almost identically to when there is only one force, only in this case it would affect multiple variables and be applied to the velocity in multiple directions at once.

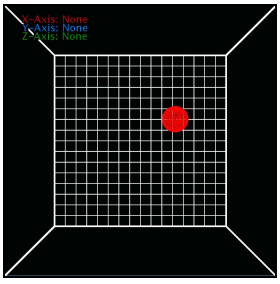

Figure 13*: 3D Implementation of Bounce with external forces acting on the spheres*

3.5 Hierarchical Modeling

The final step of the learning processes was to learn the basics of hierarchical modeling. Hierarchical modeling is when the model is designed in a tree-like structure, in that the child nodes of the tree are dependent on their parent nodes. A good example of this is how your hand is dependent on the movement of your arm, however your hand can also move independently of your arm, in that it can bend at the wrist. If you look at this example as a hierarchical model, the hand would be the child node and the arm would be the parent.

The example that was used as a reference in my first project was from a computer graphics lecture created by the Computer Science Department at the Worcester Polytechnic Institute. This lecture outlines the concepts of hierarchical modeling, specifically how and when to push

and pop matrices. The concept behind pushing and popping matrices, put simply, is that when you want to apply a transition to only a specific part of a 3D model you must push the matrix before drawing this part and pop it afterwards. By nesting these matrices it is possible to draw different pieces of the overall model and apply transitions to only the specified parts.
For example, say you want to move an arm with a hand on the end. You would first need to push a matrix for the entire canvas. Any changes to this matrix, between when it is first pushed to the stack and when it is popped from the stack, would apply to both the arm and the hand, meaning the hand would move depending on the arm. If you want the hand to move independent of the arm, you would need to push another matrix to the stack after the arm is drawn, but before the hand is drawn to the canvas. By nesting this call to pushMatrix, and placing the hand within the nested call, one can animate the hand independently of the arm until they decide to pop the nested matrix. By nesting calls to pushMatrix and popMatrix, it is possible to achieve a hierarchical model, and define which objects move dependently and independently of each other and under what conditions they do so. Another important thing to note, is that there are two types of transformations, absolute and relative transformations. Absolute transformations and relative transformations. Absolute transformations are simply a transformation applied relative to the origin whereas relative transformations are applied relative to an object's parent. These are often much easier to program and are what I used

The main example I re-created was based on page 9 of the WPI computer graphics lecture. It depicts a simple robot hammer. The base(root), lower arm(child), upper arm(grand-child) and the hammer(great-grand-child) each move dependent on themselves and their parent node, which in turn are dependent on their own parent nodes. By pushing a new matrix to the stack before applying the transitions to each child node, we are able to re-create this example in processing, as seen below.

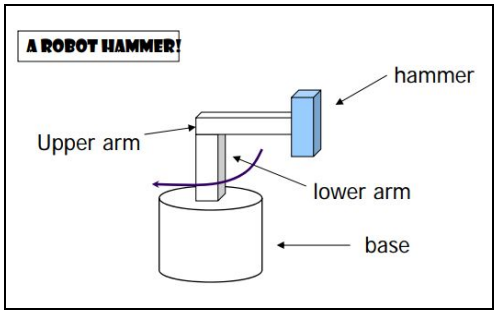

Figure 14: *The Simple Robot Hammer from The WPI computer graphics lecture*

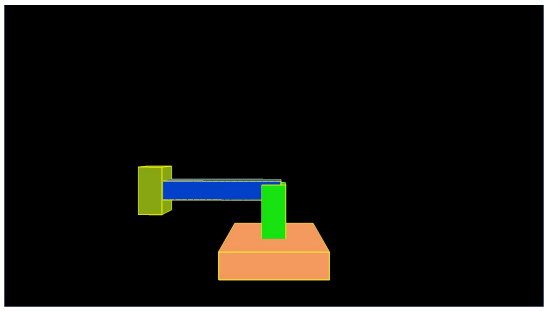

Figure 15: *Implementation of the WPI Simple Robot Hammer*

3.6 Joints and Constraints

Taking the same simple hammer example from the previous section, it is important to consider how the joints work and what constraints should be put on them. In practice this can be done by setting limits on the variables controlling the joints in question. More difficult perhaps was deciding what constraints to limit joints to. In the simple hammer example, the base was constrained to transitioning 250px in any directions, the lower arm was allowed to rotate 180 degrees in either direction, the upper arm was restricted to moving up and down 45 degrees from the starting point and the hammer was not constrained and could move freely.

3.7 Poseable Skeleton

The first step in the development of the final project was to create a skeleton that could be posed and manipulated at will. The scene graph for this hierarchical model can be found in the figure below. This example was developed much like the simple hammer machine from sections 3.5 and 3.6, only it was much more complex with many more matrices being pushed to the stack, both nested and unnested.

Using the hips as the root node, the scene graph then branches to 5 unique hierarchies, one for each arm, each leg and one for the spine up to the head. Each bone on the skeleton was set to be able to be moved dependent on its parent nodes and independently according to constraints imposed upon it relative to its parent node. In order to determine which bone was to be moved, I made use of the shapes3D library and its picking function so I could simply click on a bone and manually transform the bone and pose the skeleton.

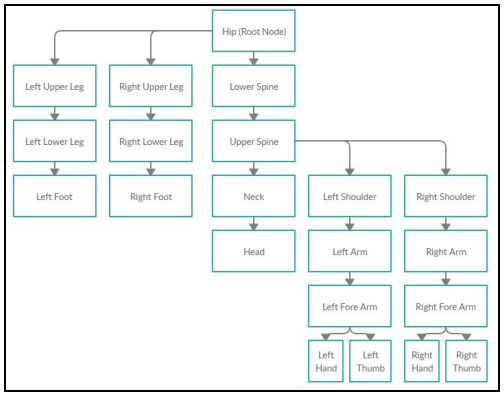

Figure 14*: Scene graph for a human skeleton*

Figure 15*: Implementation of a poseable human skeleton*

3.8 Applying Motion Capture Data

The final step in the development process of this skeleton was to consider motion capture data and how it can be applied to the skeleton. By using motion capture data provided by the National University of Singapore, I was able to parse the motion capture data and apply its transformation data to the corresponding joints in a slightly modified version of the skeleton created for part 3.7.

By doing this, it was possible to create a simple application that you could load motion capture data into, and so long as it had a similar skeleton associated with it, the application could apply the motion capture and transformation data to the skeleton for visualization purposes.

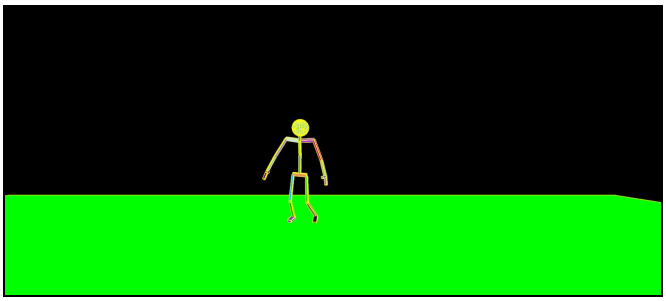

Figure 16*: Implementation of a Human Skeleton used to visualize motion capture data(.bvh files).*

4.0 Tools and Libraries Used

4.1 Processing Java

The Java version of Processing3 was the primary tool used during this course and the development of the final project. The official website defines processing as "a flexible software sketchbook and a language for learning how to code within the context of the visual arts." This tool is a great tool for programming and modeling in 3D space and is the backbone of everything developed for this report and during this course.

4.2 shapes3D Library

The shapes3D library is a processing library developed by Peter Leger, that allows for an easy way to define and draw 3D objects in processing. The feature that I used it for however, was it's picking feature. This feature is invaluable as it allows for an easy way to discern which object was clicked on. This was used heavily in developing the simple hammer machine and the poseable skeleton as it allowed me to select which part of the overall model I wanted to transform.

4.3 NUS Motion Capture Database

The motion data was obtained from a database created by the National University of Singapore. This motion data was integral to the final version of the human skeleton I developed as it relied on this motion data in order to animate the skeleton. The motion data provided by this database was provided in ascii format and because of this it simplified the process of parsing the file in order to apply the motion data to the final version of my skeleton.

5.0 Conclusion

During the course of this project I was able to successfully explore concepts relating to human modeling such as drawing in 3D space, animation concepts such as motion and interaction, external forces, hierarchical modeling, and types of joints and the constraints that apply to them. Not only was I able to develop proof of concept programs for each of these concepts, but I was also able to make use of each of them in the development of my final project while creating a human skeleton that is responsive to motion capture data. Overall I believe the final project was a success as it not only worked as intended but did so by building on each of the concepts explored during the learning and research portion of this course.

5.1 Next Steps

There are a number of ways to continue working on the human skeleton project. These can include anything from animating multiple skeletons at once, adding external forces to interact with the skeletons and potentially even looking into recording new motion capture data for use with the skeleton from the final project.

References

1. Foundation P Examples. Short, prototypical programs exploring the basics of programming with Processing. In: Back to the Processing cover. https://processing.org/examples/. Accessed 27 Feb 2020
2. Foundation P Bounce \ Examples \ Processing.org. In: Back to the Processing cover. https://processing.org/examples/bounce.html. Accessed 27 Feb 2020
3. Foundation P Brownian \ Examples \ Processing.org. In: Back to the Processing cover. https://processing.org/examples/brownian.html. Accessed 27 Feb 2020
4. Foundation P CircleCollision \ Examples \ Processing.org. In: Back to the Processing cover. https://processing.org/examples/circlecollision.html. Accessed 27 Feb 2020
5. Foundation P Follow1 \ Examples \ Processing.org. In: Back to the Processing cover. https://processing.org/examples/follow1.html. Accessed 27 Feb 2020
6. Foundation P Follow2 \ Examples \ Processing.org. In: Back to the Processing cover. https://processing.org/examples/follow2.html. Accessed 27 Feb 2020
7. Foundation P Reach3 \ Examples \ Processing.org. In: Back to the Processing cover. https://processing.org/examples/reach3.html. Accessed 27 Feb 2020
8. Agu, E., 2020. *Lecture 5 (Part 3): Hierarchical 3D Models.* Accessed 12 Mar 2020
9. SFU Motion Capture Database. In: SFU MOCAP. http://mocap.cs.sfu.ca/. Accessed 20 Apr 2020