Hierarchical Modeling

Objectives

- Build, maybe import, 3D models
- Do a little more hierarchical modeling
- Incorporate some animation

We studied some hierarchical code in class. The robot arm examples are from the WebGL Programming Guide Chapter 9 examples. I recommend that you study them before starting this assignment. Please check on Moodle the documents related to this hierarchical modeling example.

To complete this homework using only WebGL you will need to update the shaders with consideration to the 3D view projection, modeling matrices and lighting calculations and store the 3D model in a hierarchical data structure. Thus, I strongly recommend you use sample from the online textbook we have been using. In particular, the two following demos contain most of the relevant code you will need to augment.

- Simple Model Viewer with Lighting: Phong lighting
- Simple World Viewer: complex object and tree structure

Orbits motion

For this assignment you are going to write a WebGL program that builds an hierarchical scene, which is at least as complicated than the solar system described.

You have to develop some three dimensional models and then put them together in a rotational system. For example you may start simulating a simple solar system where shapes orbit around one another.

You should have at least three shapes. They do not have to use gradient, the appearance is up to you: they need to look good together, i.e. be well-integrated in their appearance and motion.

Every visible shapes body in the scene should spins in place at a specific controllable rate (some should be slow, other faster). Body can have other body in orbit around it at some fixed radius (start with circular orbits). Every body could presumably be in orbit around some other body or around an axis.

Requirement

- You have to build a system at least as complicated as a simple solar system, made of the earth, the moon and the sun.
- There should be a center fixed in location with an object such as a sun or an axis, which can be invisible.
- There should be at least two more objects rotating around the central object (three if the center one is invisible).
- Objects should move in relation to other.
- Not all rotations or orbits can have the same rotational velocity.
- You should have at least two types of shape/form, and one of them should be something other than the cube or the pyramid shown above. It should be a complex object made of simple shapes.
Figure 1: At least 3 shapes, should be different form or material

Figure 2: Example of possible motion
At least two shapes should appear different with regard to illumination.
The shapes should have material, lighting, size and form so that their spin can be noted.
Not all rotation can be around the y-axis.
Orbits should be independent of the orbited body’s rotation.
The whole system should be animated, can be paused (key P), slow down (<) and speed up (>).
The mouse should permit to change the scene view orientation.
You should use perspective projection (I know we haven’t gotten there, start with perspective(70, 1, .1, 90), and we will get to the explanation of what it does and how to tweak it later).

Implementation

With this assignment, I am taking the training wheel off and we will finally let you have some real creativity. I mean that with respect to what you build and how you build it. That said, I will be looking to see how clean your code is, and I will be particularly looking to see how easy it would be to extend your system. So, the rest of this documentation is in the form of some very strong suggestions.

Object-oriented design

Your system has to be hierarchical. With the exception of the center point or axis of your system, the position of every body is with respect to some other body. Each body also has some internal state (like the rate at which it turns, current angle, and possibly scale). Each body has to know how to draw itself and update its angle.

To me, this sounds like an object-oriented design. We could imagine building a class that had properties for rate, angle, scale and anything else we deem important. It would have methods for draw() and update(), that could be called externally.

Handling hierarchy

To handle hierarchy, we could just have each body responsible for keeping track of all of its dependents in a list. To do this, we could create another method called add(), which allowed us to add an orbiting body and specify the radius of the orbit, the axis of the orbit, and the speed. One of the components of the draw() and update() methods would be to iterate through the list of sub-bodies and call their update and draw() functions. This makes management easy because all the main code would have to do is construct the hierarchy, and then it can just call the update() or draw() function on the sun-like object, and the whole system would update itself.

Within the draw function, a body needs to take care of its rotation, and then it needs to iterate over the list of its dependent (moon-like) object. For each of them, it can do a rotation and translation to set the current transform in the correct location for it to be drawn (which means that all bodies happily think they are just spinning in place at the origin).

Ah, I brought up the current transform... It should be no great surprise that here in this assignment, the matrix stack really comes into its own. Now, being able to push the current transformation onto the stack so you can make some modifications for orbiting bodies becomes essential.
Separation of concerns

In the interest of generality, my approach also makes one further tweak. All of the functionality described above is completely removed from what the body actually looks like. This is as it should be, a square and a pyramid both behave the same in this system. So, the final thing that I add as a property to my orbiting body is something that knows how to draw a particular shape. In my case, I created another object that knows how to draw itself and a further property that sets the scale (an example is provided of what that might look like).

Object-oriented JavaScript

As you know JavaScript is an odd language. It is inherently object-oriented. However, for those of us used to writing object-oriented code in languages like Java, C++, or Python, there are perhaps three big differences that can be a little startling.

1. there are no classes, just objects,
2. the most common type of object is a singleton (i.e., a one off, rather than some kind of new type), and
3. we can add properties and methods to an object on the fly, i.e. at any point (admittedly, Python makes this easy as well).

This is not the place for a full discussion of these topics. Refer to the JS book we used to help you out. The short version, however, is that we are primarily concerned with using a constructor, which is something of a stand in for a Java class description (and this is the keyword you should start with when looking for more information).

To get you started and give you some organizational ideas, you may use this simple example. Don’t forget the examples linked at the start of this document.

Inspiration I would like you to consider: Calder sculpture

Submit

Your submission on Moodle should consist of

- at least one username_hw3.html
- usernames_hw3.js and the required WebGL helper files you used
- a diagram illustrating your system including
  - the center, shapes rotating and orbiting, indications of motion, material and relative velocity
  - types and placements of lights
- in conjunction with a readme that gives explanations for your diagrams and answer to the following questions
  - what was the most challenging part of this homework?
  - what was the most rewarding? i.e. what are you most proud to have achieved?

Bring your diagrams after the homework is due so I can collect them.

Credit This homework has been designed by Christopher Andrews