CSCE 470 Project Write-Up: 3D N-body Simulation in WebGL

Davis Rempe
University of Nebraska-Lincoln

1. Introduction

My final project was implementing a three-dimensional n-body simulation using WebGL. The final product is a fully functional simulation of the gravitational interactions between multiple celestial bodies and allows the user to interactively parameterize the simulation system. A link to the project has been posted on http://cse.unl.edu/~drempe/.

2. Project Overview

The simulation has two parts to its appearance: the canvas showing the movement and gravitational interaction of the planets and stars, and the user interface to set the constraints on the system. On loading, the system is set to default parameters and a simulation begins to run immediately. There are only two different types of celestial bodies displayed as spheres: stars and planets. Stars are textured with the yellow and orange, while planets all appear identical to Earth. The diameter of the spheres is correlated to its mass (see section 3.1). When two bodies collide they are meshed together into a single body.

Users can rotate the simulation using left-mouse drags, zoom in on the simulation by using middle-mouse button drags, and can pan over the simulation using CTRL+left-mouse drags. The user interface to the right of the simulation provides a "Default View" button that resets the view of the simulation to what it is on the page load. The user interface also allows users to change the parameters used to run the simulation as well as view information about it. The simulation is carried out with bodies placed in random positions, with random mass, and random initial velocities, but parameters that the user provides constrains the range of these random values. For both stars and planets users can set the number of bodies, the mass range, and the velocity range. There is also a button to reset these to the default values. To run the simulation with the parameters that the user changes, they must hit the "Restart Simulation" button. Users can also pause the simulation at any time. The simulation is interactive in the sense that users can change the settings and immediately run and view a simulation with these parameters without reloading the page. The extra information displayed is the years passed since the start of the simulation, the number of bodies currently in the system, and the refresh rate that the simulation is being displayed with.

3. Implementation Approach

The project implementation was heavily based on step 4 of assignment 4. This assignment had already implemented being able to display a seemingly 3D sphere with only a quad made of 2 triangles through proper lighting and texturing. It also used ideas from assignment 2 to deal with updating the body positions and velocities. Some changes were made to the original ideas laid out in the proposal that basically came down to design decisions. In the end it was decided to keep the system more geared toward celestial mechanics, or interactions within a solar system, resulting in more accurate force calculations and making collisions possible, but allowing less bodies to be simulated at once thanks to less computational efficiency. The most noticeable consequence of this is that the calculation of forces on a single body took the brute force approach, rather than the Barnes-Hut algorithm laid out in the proposal. Not having to implement this rather intricate algorithm also left more room to focus on the visualization side of the project, since after all it is a computer graphics project.

3.1 Sphere Models and Data Transfer

As previously mentioned the spheres were modeled using 4 vertices that make up 1 quad of 2 triangles. Since this was already set up for 1 sphere in assignment 4, the challenge in this project was generalizing it to any number of spheres. To accomplish this, much more data needed to be passed from the application program to the shaders. Buffers were set up to pass the quad vertices and centers after quaternion rotation as well as the vertices and centers before any transformations. The former is for lighting, and the latter for texturing (see 3.2). For each vertex, the radius of the body that it makes up and a float representing whether that body is a planet are also passed through buffers. The body radius is used for both lighting and texturing, while the float indicating planet or star is used to texture the body properly. The "isPlanet" float is basically used as a boolean value. Uniform values are also set for shading parameters and model-view, projection, and inverse rotation matrices (for texturing).

All bodies in the simulation are internally stored as objects that have real physical values: mass (in kg), position (in meters), and velocity (in m/s). Since they are stored like this, within the render loop it is necessary to translate all these bodies into the camera coordinate system and give them proper radii. So on each render, after the bodies are updated from the forces of other bodies, the "createBodySpheres" function goes through each body to create its sphere.

The radius of the sphere that represents the mass is calculated differently for stars and planets. For planets, the proportion of the current body's mass to the maximum starting planet mass is calculated and then scaled by 0.25 before being added to 0.25. This ensures that stars will have at least a radius of 0.25 and at most 0.5. Note that this max radius of 0.5 is only for initial planets; after a planet has collided with many others its mass may be greater than the initial maximum mass causing it to render with a radius greater than 0.5. For stars, the mass is also normalized by the maximum initial star mass, but this value is scaled by 1.5 and then added to 0.5. This ensures that all stars will initially be of greater radius than any planets, but not exceed a radius of 2 (again, this is just initially, it may collide and gain more mass). These functions were chosen after it was found that a logarithmic scale did not differentiate enough between stars and planets. Though they are not necessarily physically accurate, they were chosen for good visualization purposes.

After finding the proper radius, the position of the body also needs to be transformed. For each component of the position vector, the program finds the proportion of the position to the initial system radius and scales this by 100. This ensures that, at the beginning of the simulation, all bodies start within a 100 side-length cube centered at the origin (in camera coordinates before applying the model-view matrix). It's worth noting that the initial system radius previously mentioned is about the radius of our solar system; so the system is initially constrained to the diameter of our solar system. This is consistent with the idea of keeping the simulation focused on celestial mechanics.

After finding the radius and position of the body, vertices are found to construct the quad that will appear as the sphere. A quad that represents the sphere under no rotation, translation, and scaling is found. This is used for texturing in the fragment shader. Next, the positon of the body under quaternion rotation is found, and then a quad is constructed around it such that it faces the viewer. This is the quad that will actually be displayed and used to calculate lighting. Before actually pushing all this quad data to their respective buffers, it was necessary to sort the data by the z value of the rotated quad. This is to ensure that the quads will be drawn from back to front to avoid issues with blending.

Finally, the VBOs are updated with all the new quad data.

3.2 Body Texturing and Lighting

The implementation of per-fragment Phong shading with true normals done in the fragment shader was minimally changed from assignment 4. The only difference is that the position of the light in the scene is no longer affected by the model-view matrix. It is simply held in one place so that no matter what transformations the user may apply to the simulation system, they will still be able to see it lit.

To further visually differentiate between stars and planets, it was decided to apply a unique texture to spheres depending on whether it was a planet or star. On application initialization, two different cube map textures are created and passed to the fragment shader. This first, for planets, is the texture of the Earth. The second, for stars, is a texture you might typically see for the sun. Since it was not possible to tell whether a body is a planet or not solely from its radius (thanks to collisions), it was necessary to pass a boolean value in to the shader to specify this attribute. Since it is not possible to pass booleans, a float value of 1.01 was passed in for planets, and 0.01 for stars. The fragment was then textured based on this value. The calculation for the texture normal was done identically to assignment 4. This is simply the normal of the sphere with the inverse rotation quaternion applied. This is because if a user rotates the sphere, the texture we want to see is the one on the side of the sphere facing the viewer, not the back side, which would be the side of the normal if just the rotation matrix was applied.

The idea of texturing the background of the simulation using a skyboxing technique came up during development, but was ultimately decided against. This is because I thought it would be very difficult for users to visually distinguish small bodies in the simulation from a busy texture in the background.

3.3 Simulation Physics

The main physics calculation done in the simulation is the gravitational force vector to apply to a body at each time step. A time step of 1 day was chosen for the simulation. This was experimentally derived and seemed short enough to get mostly accurate results while staying long enough to keep the simulation fast enough for the user to its effects. For each time step the "updateBodies" function is called and handles calculating and updating bodies based on the force applied to it.

The force on each body was calculated using Newton's law of gravitation. The equation was laid out in the proposal as

$$\vec{F}_i = -\sum_{j \neq i} \frac{G m_i m_j (\vec{r}_i - \vec{r}_j)}{\left(\left|\vec{r}_i - \vec{r}_j\right|^2 + \epsilon^2\right)^{3/2}} ,$$

where G is the gravitational constant, and ε the softening length. The softening length of 3E10 was chosen experimentally. The calculation of force took a brute-force approach. The force for a body is the summation of all the forces on that body from every other body in the system. This is contrary to the Barnes-Hut algorithm presented in the proposal, since after further research it was discovered that this algorithm was meant for very large, collisionless systems where close interactions were not important [1]. Since my project took collisions into account, and since its scope is about the diameter of our solar system, it was decided that this algorithm was not appropriate. Besides enabling collisions, the brute-force approach also provides perfectly accurate force calculations since no approximations are made. The drawbacks are that the algorithm is $O(n^2)$ which is not as efficient as the $O(n \log n)$ Barnes-Hut. Therefore, the system can handle only a few hundred bodies before a noticeable slowdown is seen. This is appropriate, though, since solar system interactions don't usually go over this many bodies anyway. After calculating the total force on a body, the acceleration was derived with Newton's second law and then the solution of the differential equation to get velocity was approximated using Euler's method.

While looping through the bodies to calculate forces, it was also necessary to detect any collisions between bodies. Two bodies were determined to be colliding if the distance between them was less than the softening length. Since the bodies didn't inherently have a radius, this was the next best option. As the softening length is basically the distance that the force between two bodies goes to infinity, it seemed appropriate to determine that bodies at this distance were colliding or very close to it. The only problem with this approach is that visually it can look like two bodies are colliding, but they really aren't, so the bodies appear to pass through each other in the simulation. To avoid complications with force calculations, collisions were not immediately resolved in the force loop, this came after all forces were done being calculated. To resolve a collision, the two bodies were combined into one. The position of the new body was set to be that of the body with the larger mass. The new mass was simply calculated as the sum of the two colliding bodies. The new velocity of the combined body was calculated by treating the collision as inelastic. Finally, if the two bodies were planets, then the resulting body was also a planet, otherwise it was deemed a star.

Lastly, the new bodies array is populated with the newly combined, collided bodies as well as those that didn't collide with their updated positions and velocities.

3.4 User Interface

The user interface was implemented using dat.gui, an open-source, lightweight, JavaScript library for user interfaces. It was very easy to use and provided a good looking UI for users to parameterize the system.

4. Future Work

There are still things that I think can be added on to or improve the application created. The main one is to generalize the system a bit so that it could be used to create large-scale, collisionless n-body simulations. These simulations would implement the Barnes-Hut algorithm and be able to show several thousand bodies rather than just a few hundred. This would be a matter of adding a new mode where collisions are turned off, the viewer is zoomed out to show a much larger system, and implementing the algorithm to approximate forces. But this could ideally all be done by simply expanding the user interface and modifying some implementation details.

Some other smaller ideas could be added too. Individual rotation for each body around its own axis would make the simulation a little more realistic and not terribly difficult to implement. An option to show the path that a body has traveled would also be informative.

References

1. Trenti and Hut, *N-Body Simulations (gravitational)*, http://www.scholarpedia.org/article/N-body_simulations_(gravitational), 2008.