**Solar System Simulator**

**by Daniel Young**

Introduction

The Fall 2015 Computer Graphics course offered at Purdue University Calumet challenged participating students to further their understanding of computer graphics through a semester programming project. Students were to use their knowledge of OpenGL functions and Visual C++ to create some program which demonstrated what was learned throughout the course and had the option to choose their project's topic. This report is written for a Solar System Simulation project created by Daniel Young.

This Solar System Project was intended to be a more mature version of the solar system models that were typical of grade school science projects, with less Styrofoam and more binary. While many solar system models exist (Bayer 2015, Seal 2015), these all have their costs and benefits in terms of functionality. Where one simulation may have flexible camera controls, another simulation may have better proportionality between its planets and distances, where as another simulation may have more information on each planet. This project was meant to reach a middle ground, bringing in a variety of different simulations' benefits into one. This simulation, therefore, would benefit those whom wanted yet another, more to-scale representation of the solar system in which we live.

Methodology

To create the simulation using OpenGL and Visual C++, several aspects of the simulator had to be individually constructed apart from the rest. For instance, the flexible camera controls were their own set of functions, relying heavily on camera controls created in previous assignments in the graphics course. The camera's position (eye position), the point at which the camera was facing (center of interest), and controls to ensure the camera never flipped upside down or rolled improperly were integrated. To handle camera movement, mouse movement within the screen was measured relative to the center point of the simulator window. When the mouse moved from its original position, the eye position would in turn point at a new center of interest. Center of interest locations were limited to directly above and directly below the eye position (+/- 90o) to limit the camera from flipping upside down and disorienting the users. When the mouse spins beyond +/-180o on the X plane, it flips 360o in the opposite direction to ensure the angle never jumps outside of the range of -180o to 180o. A "calculate camera movement" function was used to determine how large a change was translated to the camera and center of interest positions, similar to (but not identical to) those mentioned in other OpenGL tutorials (Jouvie 2004). A check was added to ensure that, any time the mouse pointer was moved from the center of the simulator's window, it was warped back to the center of the screen until the simulation was ended. This allowed the user to stay centered in the model's world without worrying about leaving the functional simulation space. The cursor proved a nuisance, so a GLUT function to hide the cursor icon was added soon after camera controls were functional.

Upon finishing the camera controls to view a simple teapot, a yellow sphere was introduced to replace the teapot and act as a "sun" for the simulator. This sphere was rendered flat, and there was no lighting introduced just yet. Its rotation seemed irrelevant, so a planet was added to the simulation immediately. This planet was supposed to simulate "Earth," and it was given an orbit distance from the sun. This orbit distance was simply a translation distance to follow its rotation around the y axis. So this planet was rotated, translated, and drawn in the proper location. Originally, this set of function calls was performed directly in the "display" method, but was eventually separated into a "render planet" function, which would be overloaded to accommodate planets with several moons and various arguments. Before moons were drawn around planets, the current matrices were pushed onto the stack. The moons would then be given a rotation and translation before being drawn, and the matrix would be popped back off the stack. This procedure was extended to the planets after multiple planets were drawn (after the introduction of what was considered "Mercury").

The point source for light was still missing, and the planets were pretty dull at this point. The lighting settings were enabled following the rendering of the sun, which was believed to need the highest-possible brightness and was uninfluenced by its own light. The point source was centered wherever the sun was rendered (the location of which was given a variable at this point, in case the user wanted to one day render the Solar System at a different location than the center of the three dimensional space). The lighting effects were enabled after the sun was rendered, giving each planet an illumination source. While in nature, the level of ambient light is extremely low (to the point where the back sides of planets are not visible), the ambient light level for the simulation was set relatively high so users could at least located planets while located on the outskirts of the Solar System.

Multiple planets were redesigned to no longer be hard-coded in the rendering functions themselves. A planet class was created to hold information like distance from the sun, name, size, and current rotation around the sun. Sizes were set to mimic those in reality, with radii of 10,000km being equivalent to a glut solid sphere with an argument of size 1.0f. Distances were set to mimic those in reality to a different proportion, with 108,000,000km being equivalent to a distance of 2.5\*108.0f from the sun in the simulator. This is effectively quartering the distances from the sun relative to the sizes of the planets in order to allow the user to still "find" the planets they are trying to search for.

It was decided that the user would need details of planets in the simulation as the user was free-roaming. A "heads up display" (HUD) was the next logical item to be implemented, and the OpenGL rasterization level of code was used to display text to the user in the bottom left-hand corner of the simulator window. Originally, this text would just display "Wandering the Solar System...", and was enhanced to read which planet the user approached. This enhancement, however, came with a bug that did not trace planetary location correctly as the simulator ran. This bug remains, and if the user wants details on the planets as they are passed, the user needs to pause the simulation at its start using the "slower" hotkey ("-") and find the planets afterwards.

Planetary orbits were eventually assigned using simple "glPoints" with size of 2. This allows the orbits to be visible from great distances, while being nearly invisible when close to planets. This was an unexpected but incredibly useful effect, which the designer decided to keep. Planetary texture mapping was desired, which would wrap cylindrical textures around the planetary spheres and give the simulation a greater sense of depth. This feature was not added due to time constraint and unexpected difficulty in implementation. A texture-loading class was left commented out of the code, along with some other texture-binding OpenGL calls, as a testament to the attempts made.

Results

The simulation contains the "Sun"; "Mercury"; "Venus"; "Earth"; "Luna" (Earth's moon); "Mars"; "Phobos" and "Deimos" (Mars' moons); "Jupiter"; and "Io", "Europa", "Ganymede", and "Callisto" (Jupiter's 4 well-known moons). The sun acts as a point-source of light, reflecting off of the planets according to their position relative to the sun. The user of the simulation may free-roam the system using the mouse to control the camera direction and the "W", "A", "S", and "D" keys to control camera location. With the "+" and "-" hotkeys, the user may control the simulation's speed, with the "+" key speeding planetary rotations and the "-" key slowing (or even stopping) the rotations. Approaching planetary orbits from a certain angle does indeed change the HUD to display planetary names, but this feature is currently incomplete, as is the texture mapping of the planetary spheres. Overall, the project is functional with plenty of room for improvements.

Discussion

A few difficulties were encountered in creating this simulator. While planetary sizes and orbits were to be proportional to real-world measurements, these measurements would have either made the simulation impossible to navigate within reasonable time, or they would have made the planets impossible to find for untrained users. The simulator was set to two separate scales to accommodate these issues, so planetary sizes were dictated by glSolidSphere functionality, whereas orbit distances were a quarter of the sizes they relatively would have been based on planet size. Looking at the simulation as it is, planets are still difficult to find, but their lighting functions operate correctly, as do the relative rotational speeds. Planetary rotational speeds were based off of earth days. Camera speed was multiplied by a constant determined at runtime for convenience. The simulator is inaccurate in relative positions of planets in that their rotations all start at the same position (they are all aligned at the start of the simulator), and this may be easily remedied by picking a start date (and planetary rotation) for each of the planets in the simulation. However, finding these rotations will take additional time not available to the developer, and the accuracy of this aspect in the simulation is not relevant to the project's current scope.

Success was measured solely in progress made on the simulation. Every step closer to what could be considered a "complete" model was considered an individual success. Each step taken in development was tested to ensure the outcome was as expected; for instance, cameras were verified to point up or down, or horizontally around, with any upside-down rendering considered a failure. The results from the last several runs of this program show that the next step in development is ready to be taken.

Conclusion

The project for this course is a work-in-progress that is currently functional. While the simulator integrates several of the original goals, as well as some added at a later date, it still has many areas for improvement, including texture mapping and proximity detection for the camera and planets. The simulation's current functionality is a good demonstration of several OpenGL concepts and should show how the designer has made progress in the topic of Computer Graphics.

References / Works Cited

Bayar, M. Solar System Scope. [Online]. Available: http://www.solarsystemscope.com/. [2015, December 13].

Jouvie, J. [2004]. OpenGl Tutorial 26: Moving in a 3D world - Camera. [Online]. Available: http://jerome.jouvie.free.fr/opengl-tutorials/Tutorial26.php. [2015, December 13].

Seal, D. NASA/JPL Solar System Simulator. [Online]. Available: http://space.jpl.nasa.gov/. [2015, December 13].