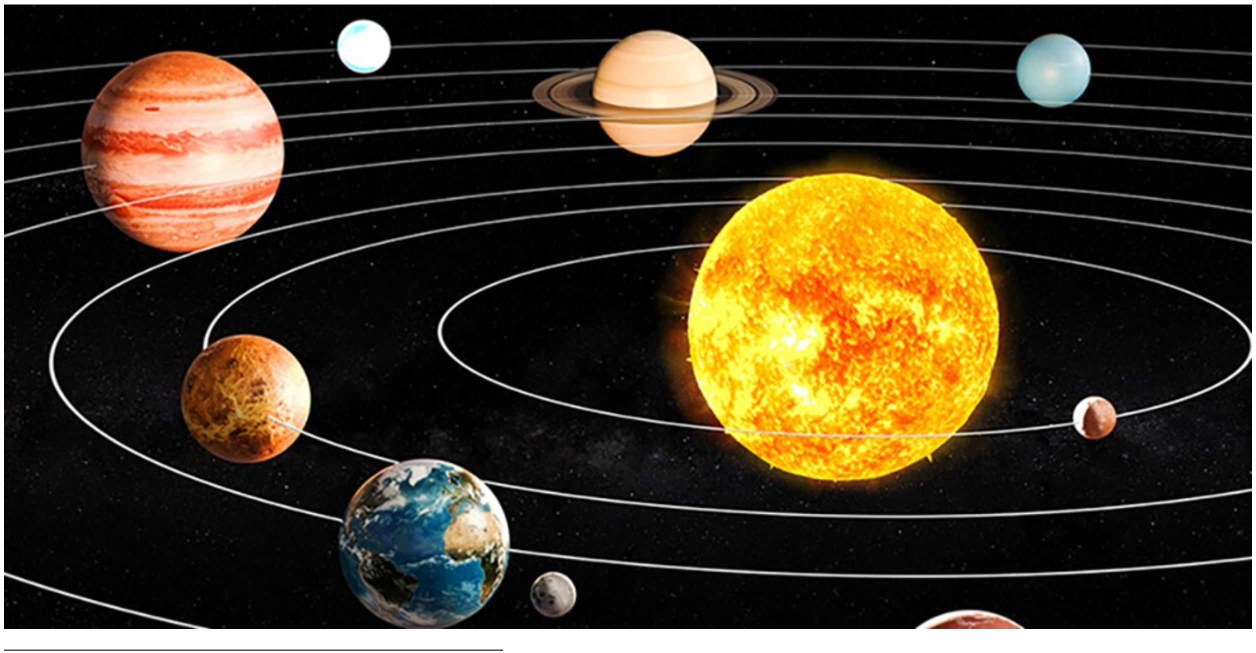
CPSC 453 Assignment 4 Virtual Orrery and Real Time Rendering *

Fall 2021, University of Calgary



* Assignment specification modelled after previous offerings of CPSC 453 by Sonny Chan and Faramarz Samavati. Please notify drlarsen@ucalgary.ca of typos or mistakes.

1 Overview and Objectives

An orrery is a model of the solar system that mechanically replicates the orbital motions of the planets around the sun. Detailed orreries also replicate the orbits of planets' satellites around their primaries. As with many other wonderful mechanical trinkets, orreries have mostly lost their significance with the advent of computer simulations and graphics.

Your job in this fourth assignment of CPSC 453 is to develop a virtual orrery that animates and renders the relative motions of the Sun, Earth and Moon against a starry backdrop. This task allows you to assemble many of the things you've learned in the course into a final rasterization assignment using OpenGL (note that the last assignment in the course will use ray tracing for rendering and not OpenGL). You will need to write C++ and OpenGL shader code to implement many of the concepts learned in class, including geometry specification, coordinate systems, reference frames, interactive input, composition of 3D transformations, model and view transforms, texture mapping, shading, and real-time animation.

Source code for a boilerplate C++/OpenGL application is provided for you to use as a starting point, though you may create your own template if you prefer. The boilerplate provides you with a spherical camera implementation and basic shading for starting your orrery. The camera can be zoomed using the scroll wheel and rotated by holding the right mouse button and dragging the mouse.

2 Due Date

November 28th at 11:59 PM

3 Programming Assignment

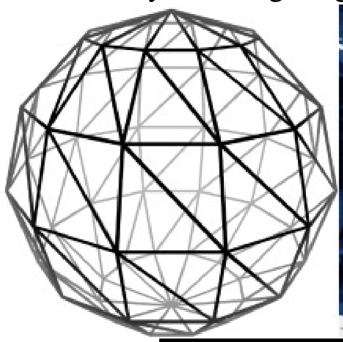
There are a total of four parts to this programming assignment with an additional requirement for integration and controls. This assignment is worth a total of 20 points, with the point distribution as indicated in each part, and an additional possibility of earning two "bonus" designations. Note that the bonuses may require going beyond what is presented in the lecture and tutorial components of the course and may require student initiative to find online resources to complete successfully.

3.1 Part I: Sphere and Texture Coordinates (4 Points)

Starting from the initially provided boilerplate code, or any template that you may have created during this course, write a program that will generate triangle geometry to approximate a three-dimensional unit sphere. There may be several good ways to do this. One possibility is to write a C++ function that generates triangles from a parametric or other representation of a sphere (or an ellipsoid created as a surface of revolution) and fills a vertex array with their vertex positions. For your sphere, you must also generate normals for each vertex to be used in shading calculations later and texture coordinates, to apply textures to your celestial bodies. Create four different spheres in your scene, one for the Sun, Earth, Moon and an additional one to encompass the entire scene for the starry backdrop. Texture these spheres using code similar to what was provided in assignment 2 (Note that you may simply copy and paste code from Assignment 2 if you like). To find the textures for your celestial bodies feel free to download the images from one of the following sources:

1. http://www.solarsystemscope.com/nexus/textures/
2. http://planetpixelemporium.com/index.php
3. http://www.shadedrelief.com/natural3/pages/textures.html

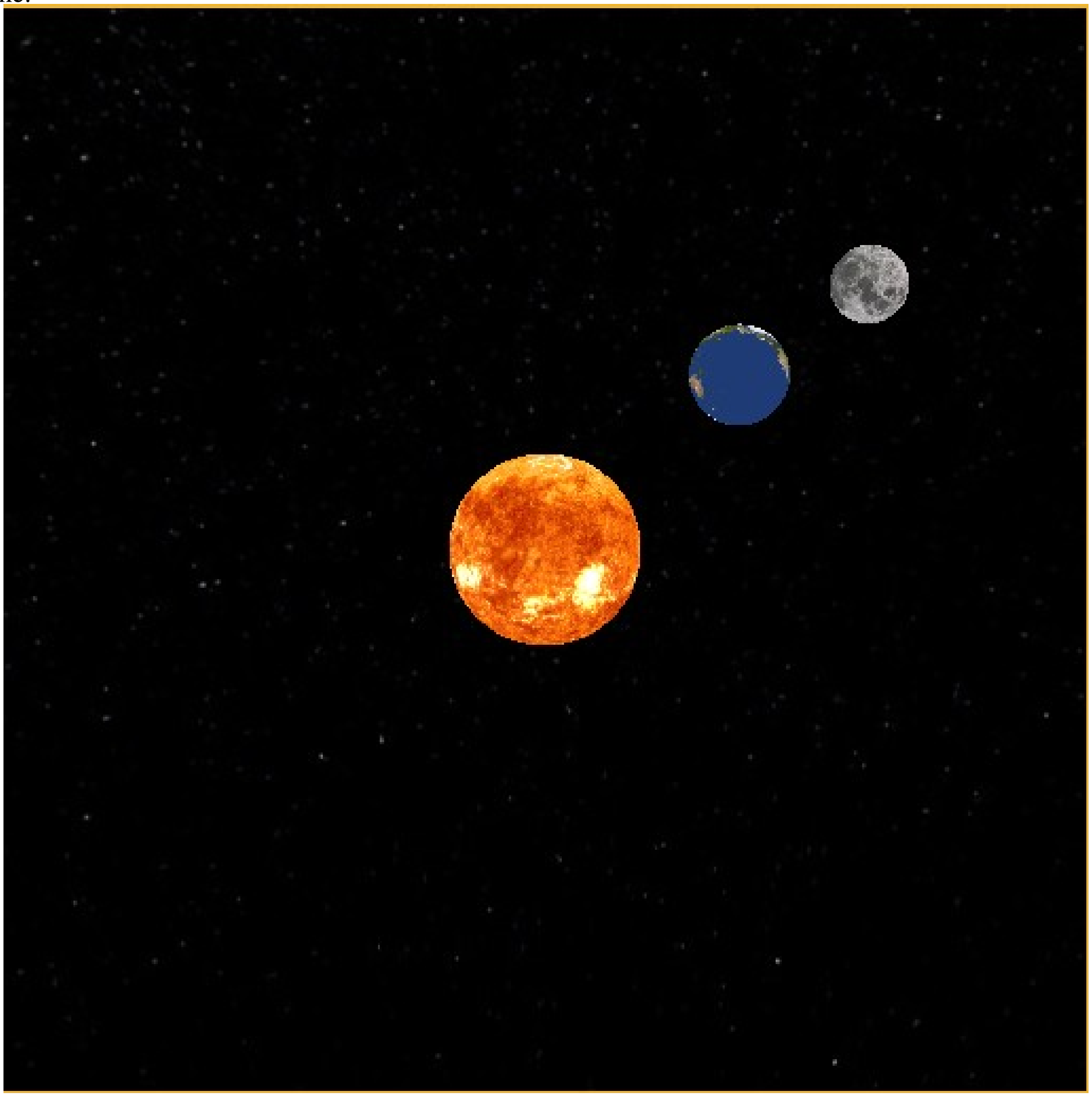
Don’t worry about lighting or shading at this point - those will come later.



3.2 Part II: Transformation Hierarchy (4 Points)

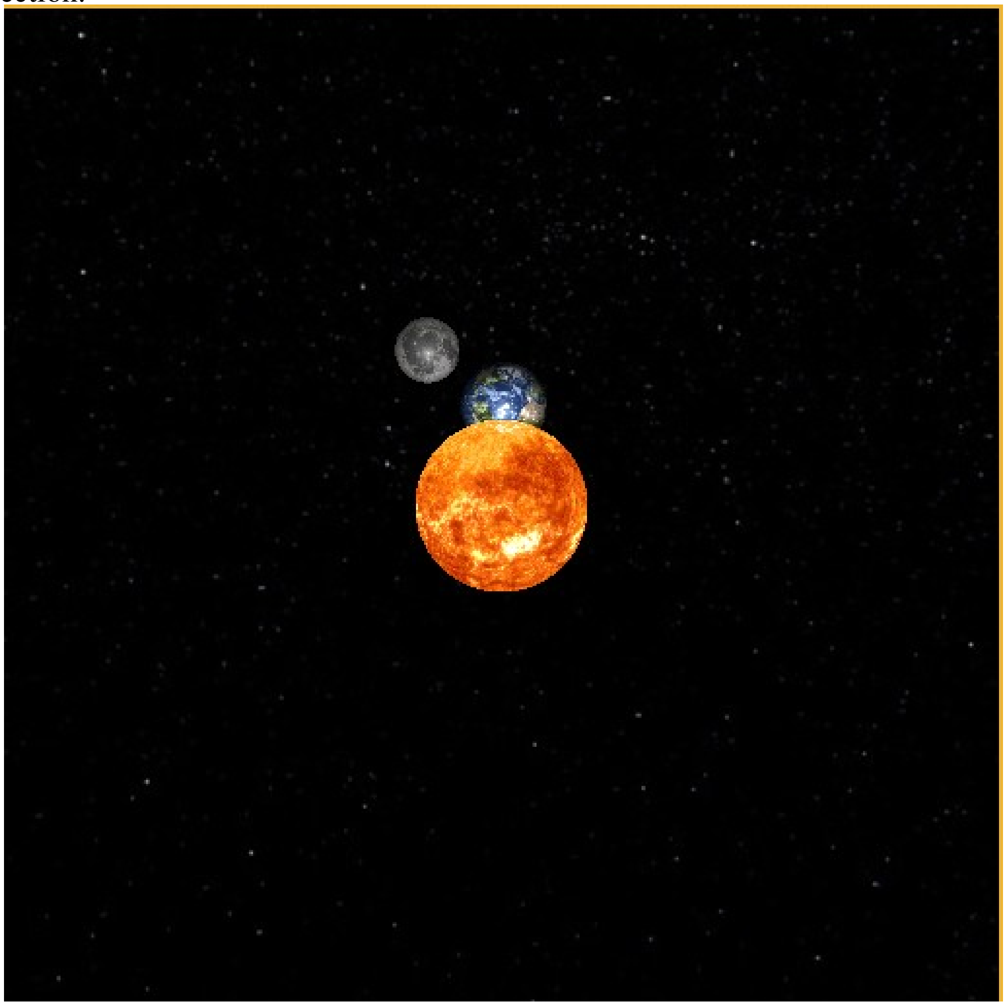
Define your "world" reference frame in terms of the Solar System, with an origin located at the centre of the Sun. Organize your objects explicitly or implicitly so that the Earth's position can be described relative to the Sun, and the Moon's relative to the Earth.

Position and scale your celestial bodies so that their relative sizes are reasonable in your virtual scene. Encode the orbital inclination and axial tilt of the bodies as rotations in your transformation hierarchy. Feel free to exaggerate the orbital inclination of the Earth so that it's obvious in your scene.



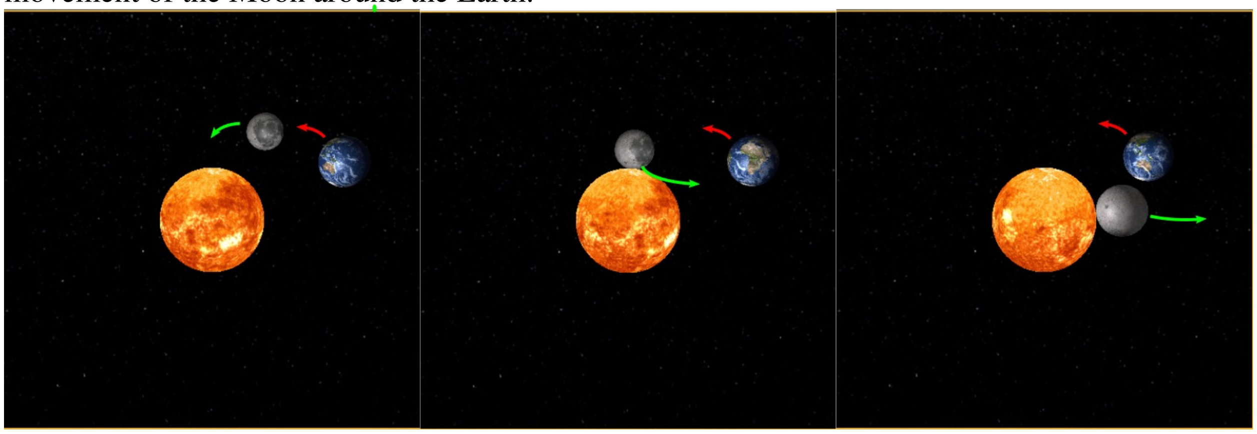
3.3 Part III: Shading (4 Points)

Introduce a point light source into your scene, positioned at the centre of the sun. In OpenGL, this usually just takes the form of a uniform vector variable in your shader. This will allow you to calculate a light direction vector for your lighting equation. You will also need surface normals which, like texture coordinates, can be stored and retrieved as vertex attributes or computed in the vertex shader. Program the fragment shader to apply a shading model (Phong Shading) to your objects. The Sun itself naturally needs no shading: it emits the light that illuminates your other objects. Apply a diffuse reflection model for the Earth, and the Moon so that the side facing away from the Sun is dark. Add specular and/or ambient components as you desire to make your scene look as nice as you can. Do not shade the Sun or the starry backdrop with diffuse or specular reflection.



3.4 Part IV: Animation (4 Points)

Now that you have all your celestial bodies rendered beautifully with textures and shading, it is time to make them move. Animate the axial rotation of each body and the orbital rotation of the Earth,and the Moon about their respective primaries. Make both the axial and orbital rotation periods of each body reasonable. A rate of one day per second of animation time, or even faster is appropriate. Provide at minimum a means for pausing and restarting your animation, and if you'd like, add the ability to adjust the animation speed. Note that each body has an axial tilt, meaning that its axis of rotation is not perpendicular to its orbital plane. The Moon has an orbital inclination with respect to the Earth's equator, meaning that its orbital plane is tilted from the Earth's axis of rotation. The Earth also has an orbital inclination with respect to the Sun's equator, and you may choose whether you prefer to tilt the Sun's rotational axis or the Earth's orbital plane for your orrery. You must capture axial tilts and orbital inclinations in your animation to receive full credit for this part. You may ignore orbital eccentricity and have the bodies follow circular orbits. The red arrows in the diagram denote movement of the Earth around the sun. The green arrows denote the movement of the Moon around the Earth.



3.5 Integration and Control (4 Points)

Ensure that your program does not produce rendering anomalies, or crash, when animating your scene. Efficiency is important too! Programs that fail to clean up resources or generate excessive amounts of unnecessary geometry may bog down the system or eventually crash, and will not receive full credit.

3.6 Bonus Part I: Realistic Orrery (4 Points)

The first possibility is to use some advanced texture mapping techniques, combined with the programmability of the fragment shader, to create an Earth that looks as realistic as possible. Here are some suggestions for what you might do:

1.Normal or bump mapping: shade the mountains and valleys to stand out.
2.Specular mapping: the oceans may look better if they were shinier than land masses.
3.Cloud texture(s): animate clouds that move over the Earth's atmosphere.
3.Day/night textures: use a dark texture with city lights for the side away from the Sun.
4.Shadows: simulate the effects of solar and lunar eclipses.

3.7 Bonus Part II: Complete Orrery (4 Points)

The second possibility of earning a bonus is to flesh out your virtual orrery to include all the planets of the solar system and as many of their satellites as you can find information about. Add the ability to:

1.To slow down or speed up the animation to appreciate slow or fast objects.
2.To centre the camera on different planets to appreciate those perspectives.
3.Incorporate orbital eccentricity (elliptic orbits of the celestial bodies).

4 Submission

We encourage you to learn the course material by discussing concepts with your peers or studying other sources of information. However, all work you submit for this assignment must be your own, or explicitly provided to you for this assignment. Submitting source code you did not author yourself is plagiarism! If you wish to use other template or support code for this assignment, please obtain permission from the instructors first. Cite any sources of code you used to a large extent for inspiration, but did not copy, in completing this assignment.

Please upload your source file(s) to the appropriate drop box on the course Desire2Learn site. Include a "readme" text file that briefly explains the keyboard controls for operating your program, the platform and compiler (OS and version) you built your submission on, and specific instructions for compiling your program if needed. In general, the onus is on you to ensure that your submission runs on your TA's grading environment for your platform! It is recommended that you submit a test assignment to ensure it works on the environment used by your TA for grading. Your TAs are happy to work with you to ensure that your submissions run in their environment before the due date of the assignment. Broken submissions may be returned for repair and may not be accepted if the problem is severe. Ensure that you upload any supporting files (e.g. makefiles, project files, shaders, data files) needed to compile and run your program. Your program must also conform to the OpenGL 3.2+ Core Profile, meaning that you should not be using any functions deprecated in the OpenGL API, to receive credit for this part of the assignment. We highly recommend using the official OpenGL 4 reference pages as your definitive guide, located at:`https://www.opengl.org/sdk/docs/man/`.