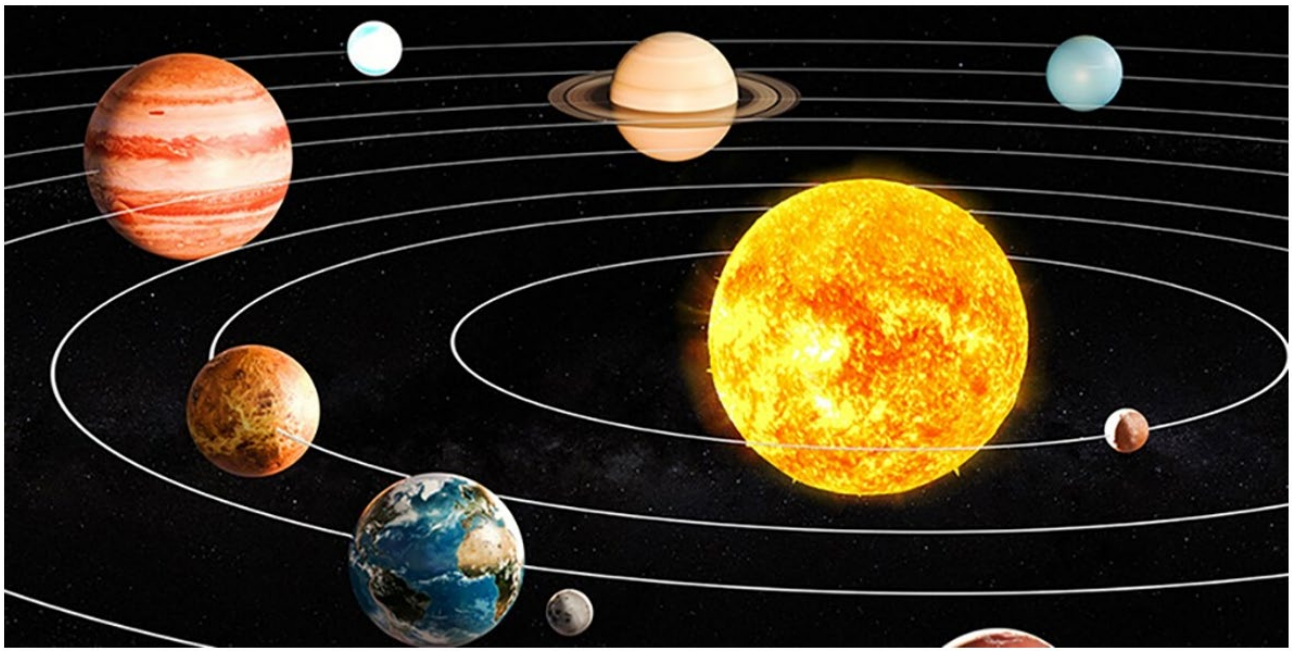
CPSC 453 Assignment 4 Virtual Orrery and Real Time Rendering [1]

Fall 2024, University of Calgary



[11]Assignment specification taken from the previous offering of CPSC 453 prepared by Sonny Chan and Faramarz Samavati and updated by John Hall. The Fall 2024 revision was updated by Riley James. Please notify riley.james1@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 (and optional) 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.

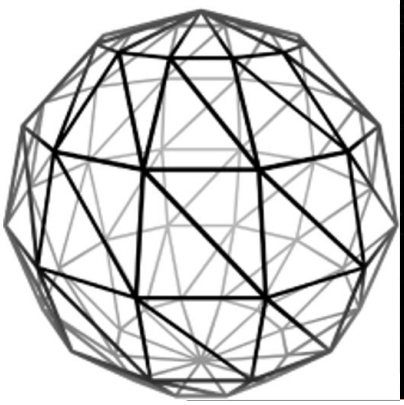
Source code for a boilerplate C++/OpenGL application is provided for you to use as a starting point. The boilerplate provides you with a spherical camera implementation with a cube in the center. The camera can be zoomed using the scroll wheel and rotated by holding the right mouse button and dragging the mouse.

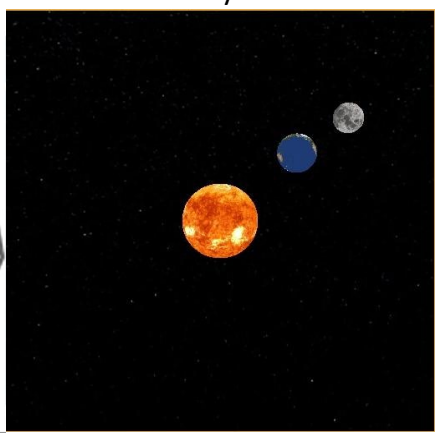
2 Base Assignment

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.

2.1 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 are 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 vertex normals 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 use as a skybox 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. https://www.solarsystemscope.com/textures/
2. http://planetpixelemporium.com/index.php
3. http://www.shadedrelief.com/natural3/index.html

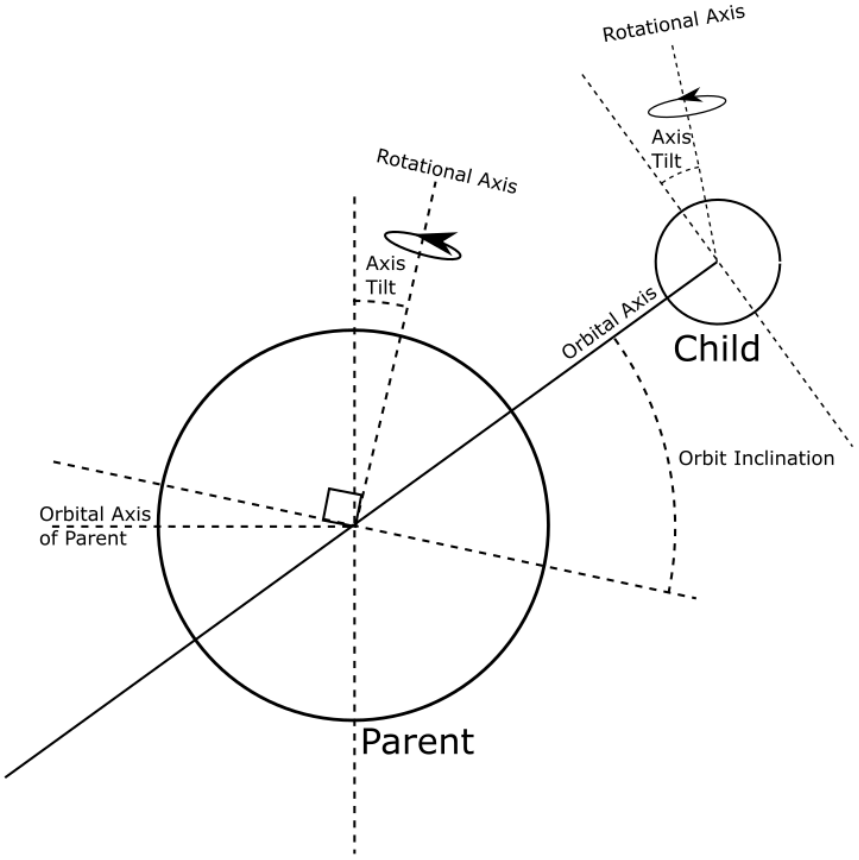
In summary, you must:

- Generate a unit sphere to use as the basis of your geometry, including vertex positions, normals, and texture coordinates.
- Texture four spheres to resemble the Sun, Earth, Moon and starry background.
- Place the 3 planets in the scene (for now, anywhere visible) with the starry background (scaled very large and placed at the center so the entire scene is inside).

Don't worry about lighting or shading at this point as those will come later. To improve performance in preparation for the bonuses, it might be a good idea to implement instanced rendering (bonus from assignment 2) and/or indexed geometry, though this is totally optional.

2.2 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 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 to see shading details later on. 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. To better define these terms, please see the image below:



In summary, you must:

- Define the transformation hierarchy for circular orbits with Orbital inclination and Axis tilt for each celestial body to its parent.

Before implementing this part, be sure to read through the requirements of part 4 and plan out, in detail, how you are going to implement this hierarchy of transformations (some transformations will apply to the children, some will not).

2.3 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 and using your normal from part 1, shade the planet(s). 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 ambient components to make your scene look as nice as you can.

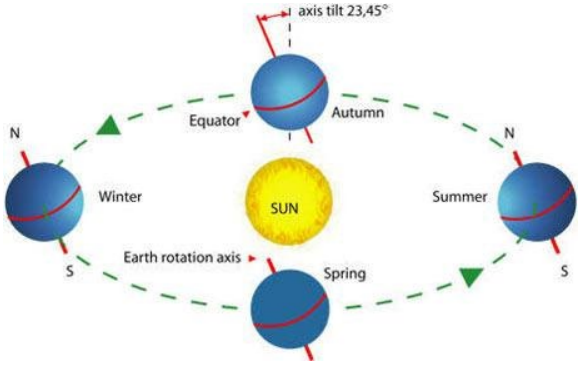
In summary, you must:

- Modify the shaders to apply the Phong Shading model (per fragment) with diffuse, specular, and ambient lighting.
- Experiment with appropriate parameters for each body to get to make your scene look as nice as you can.

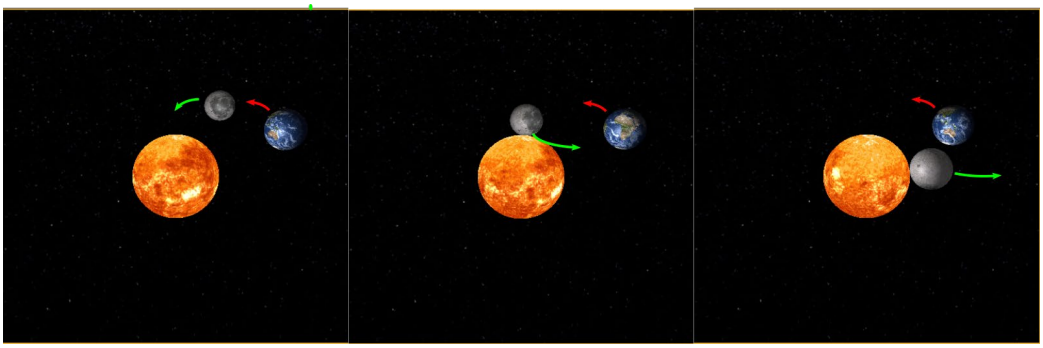
To apply the shading model to only some bodies, you can create two different shader programs and switch between them or create a uniform to distinguish between bodies you need to shade and not shade.

2.4 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 (including the sun) and the orbital rotation of the Earth and Moon about their respective primaries (parent objects). The axial rotation of the earth and moon is independent of its position around its orbit and all the orbits up the hierarchy (Ex: only the position of the earth is rotating around the sun) as seen below:



Similarly, the orbital rotation of the moon should not be influenced by the orbital position or rotation of the earth. Depending on how you set up part 2, the transformation hierarchy might already support this. 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 a means for pausing and restarting your animation. Add the ability to adjust the animation speed. 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.



To receive all 4 marks (rather than ¾), the animation should be scaled to real-time (not dependent on framerate). This can be done by tracking the current system time using $glfwGetTime()$ and measuring the time difference between the last update and the current update.

In summary, you must:

- Animate the orbital and axis rotations for the moon and earth, as well as the axis rotation of the sun.
- Allow the user to pause, play, and restart the animation as well as scaling the animation speed.
- Make the simulation independent of framerate.

2.5 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 Additional Bonus Requirements

For this assignment we have several bonus options. In total, you can receive up to 8 bonus marks, in any combination. However, feel free to implement more than 8 points worth if you'd like. The marks of the bonuses will be assessed not only on completion, but also quality of result and implementation. Be sure to cite any sources of textures, lessons, or details used for these bonuses (even if the link is provided).

3.1 Bonus Category 1: Gouraud shading (2 points)

Implement a form of Gouraud shading, where the relevant intensity coefficient(s) from the Phong reflection model are calculated per-vertex and interpolated for each pixel. Allow the user to switch between Gouraud and Phong shading.

3.2 Bonus Category 2: Improved Rendering (Up to 4 points total)

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. Textures can be found using the links provided in part 1.

1. Apply normal mapping to the Earth and/or Moon to add additional detail. *(Up to 3 points)*
   https://learnopengl.com/Advanced-Lighting/Normal-Mapping
2. Apply specular mapping to improve the specular highlight of the earth's oceans. *(Up to 1 point)*
   https://learnopengl.com/Lighting/Lighting-maps
3. Animate a cloud texture rotating independently around the earth. *(Up to 2 point)*
   https://learnopengl.com/Advanced-OpenGL/Blending
4. Blend a night texture with city lights with the day texture so the side facing away from the sun is properly lit up with lights. *(Up to 2 point)*
   https://learnopengl.com/Advanced-OpenGL/Blending
5. Implement a cube map skybox texture for rendering the stars. *(Up to 3 point)*
   https://learnopengl.com/Advanced-OpenGL/Cubemaps

6. Generate shadows for simulating eclipses, both lunar and solar. (Ex: ray-sphere intersection or distance between line and point) *(Up to 4 points)*
   https://en.wikipedia.org/wiki/Line%E2%80%93sphere_intersection
   https://en.wikipedia.org/wiki/Distance_from_a_point_to_a_line

3.3 Bonus Category 3: Control and Completeness (Up to 4 points total)

The second possibility of earning a bonus is to flesh out your virtual orrery.

1. Include all the planets of the solar system. Add the moons of planets (if a planet has more than 3 moons, only include the 3 largest). Textures for the moons don't have to be correct. For Saturn, add its planetary rings by generating ring geometry. Ensure the rings are rendered appropriately with Phong shading *(Up to 4 point)*
   https://solarsystem.nasa.gov/solar-system/our-solar-system/overview/
   https://www.solarsystemscope.com/textures/
2. Implement a way to center the turntable camera on the earth and moon (and any other planets/moon implemented). *(Up to 2 point)*
3. Implement elliptic orbits (orbital elasticity) and take into account the change in speed as planets progress through their orbits. To illustrate this bonus, the elasticity of the orbits can be exaggerated. Just like the circular orbits in the base assignment, elliptic orbits should work with orbital inclination. *(Up to 3 points)*
   https://courses.lumenlearning.com/waymakercollegealgebra/chapter/equations-of-ellipses/
   https://space.fandom.com/wiki/Elliptic_orbit

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/.