Assignment #3 – Planet rotates around sun

Yoke Kai Wen, 2020280598

April 18, 2021

1 Introduction

In this assignment, I am required to build a solar system, by (1) setting the orbits, sizes and textures of planets, (2) include keyboard control for movement and speed, and (3) add text besides planets that move with the planets. Figure 1 shows the a frame of the output rendering.

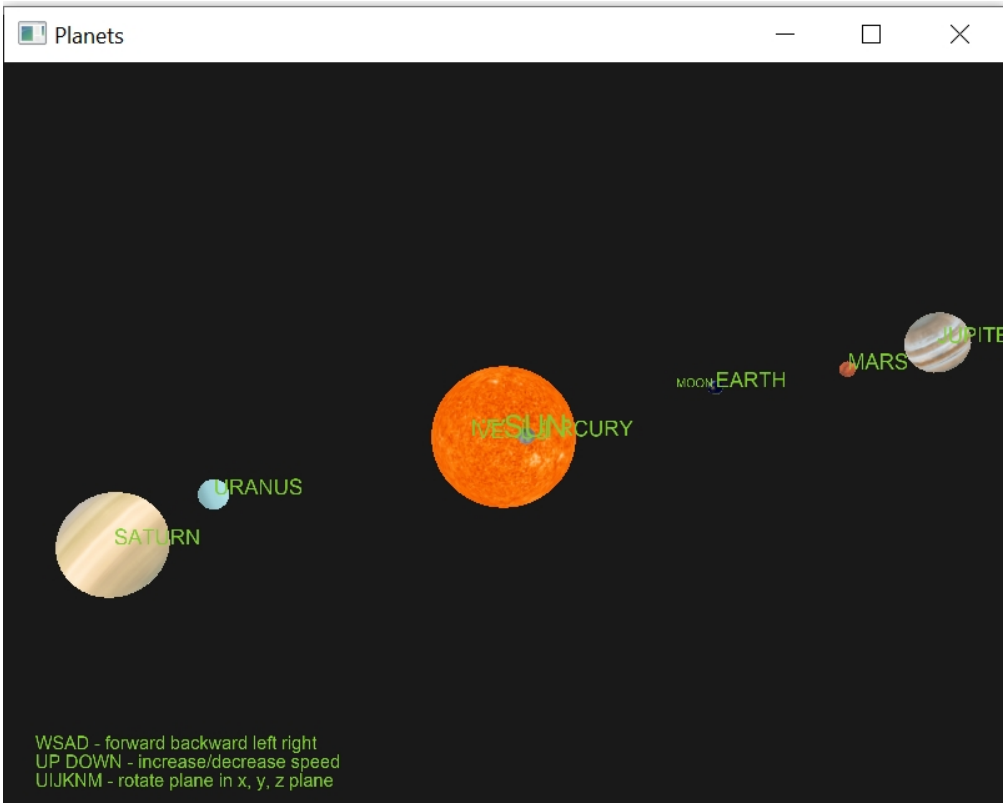

Figure 1: Rendering of planets rotating around sun

2 Files and directory structure

1. Main script: main.cpp
2. Header file for sphere: Sphere.h (remember to set root dir as include directory in VS)

3. Shader files in shaders/ folder: main.vert.glsl, main.frag.glsl, text.vert.glsl, text.frag.glsl
4. Other header files: include/tut_headers/camera.h, include/tut_headers/shader.h
5. Set include/ and lib/ folders as include and library directories respectively in VS.
6. Texture bitmaps in Textures/: 10 bitmaps
7. Font file in font/: arial.ttf

3 Usage

1. Use 'UP' 'DOWN' to increase and decrease planet orbiting speed.
2. Use 'W', 'S', 'A', 'D' to move forward, backward,left right.
3. Use 'U', 'I', 'J', 'K', 'N', 'M' to rotate around X, Y, Z planes.

4 Design

4.1 Generating 3d sphere position and texture vertices

In the Sphere.h file, the sphere's vertices are computed with the input of three parameters: `radius, sectorCount, stackCount`. The sphere is divided into horizontal strips (stacks), and each stack is further divided into quadrilaterals (sectors). The surface of each quad sector is then rendered by drawing two triangles. The index of each sector and stack corresponds to the texture coordinates (the unrolled horizontal strips correspond to the 2D texture bitmaps). The indices are also stored so that the triangles can be rendered with `glDrawEleements`. Figures 3 and 3 gives a visual illustration.

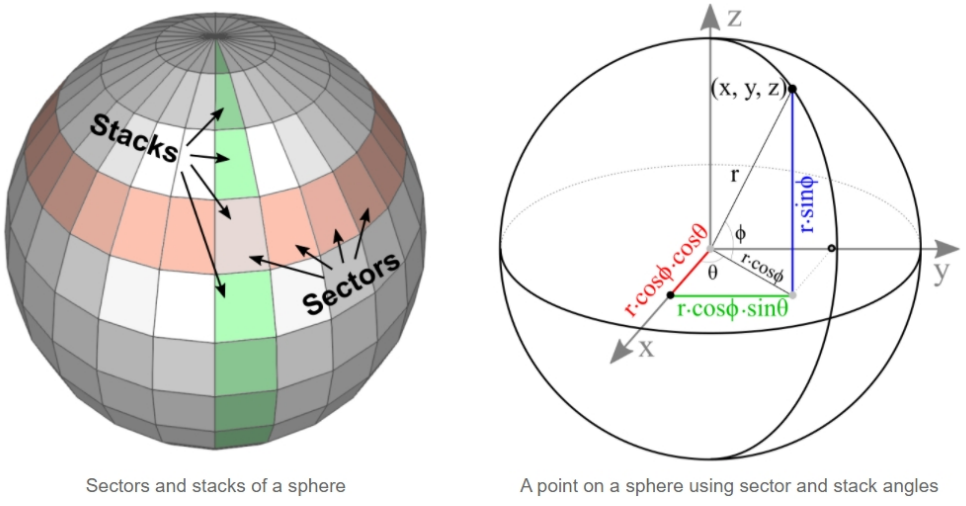

Figure 2: Sphere divided into stacks and sectors, with stack and sector angles shown

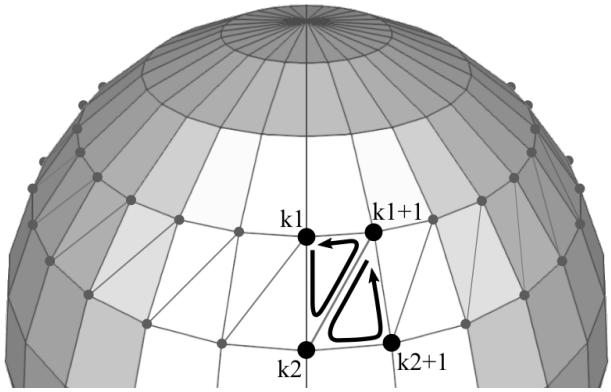

Figure 3: Sphere vertex indices for drawing triangles

The vertices of each sector in each stack are calculated based on the formula: $x^2 + y^2 + z^2 = r^2$. Then, with $-90 \leq \phi \leq 90$ being the stack angle, and $-180 \leq \theta \leq 180$ being the sector angle, we have:

$$x = (rcos\phi)cos\theta$$

$$y = (rcos\phi)sin\theta$$

$$z = rsin\theta$$

4.2 Creating planets with different textures and sizes

Using the `Sphere` constructor from Sphere.h, I create the 10 planets with the example command `Sphere Planet(radius, stackCount, sectorCount)`. I set the radius of each planet differently to roughly show the relative ratio of the planet sizes in real life. For most of the planets, I set the number of stacks to 18, and number of counts to 36, but since the sun is much bigger than the other planets, I set the stackCount and sectorCount to 5 times of other planets so that the Sun sphere will still look smooth.

```
// SPHERE GENERATION (sizes determined by radius)
float factor = 500.0f; //arbitrarily determined so that all planets visible in
    window
float shrink = 0.5; // arbitrarily determined to scale planet names to visibility
Sphere Sun(100.0f/factor, 36 * 5, 18 * 5);
Sphere Mercury(10.0f / factor, 36, 18);
Sphere Venus(12.0f / factor, 36, 18);
Sphere Earth(11.8f / factor, 36, 18);
Sphere Mars(8.0f / factor, 36, 18);
Sphere Jupiter(40.0f / factor, 36, 18);
Sphere Saturn(37.0f / factor, 36, 18);
Sphere Uranus(30.0f / factor, 36, 18);
Sphere Neptune(30.0f / factor, 36, 19);
Sphere Moon(5.5f / factor, 36, 18);
/* SPHERE GENERATION */
```

I then load the ten texture bitmaps with the `loadTexture` function from Tutorial 6_3 and bind them accordingly to the correct planet sphere.

4.3 Controlling orbit and rotations of planets

The variable `PlanetSpeed` controls the speed of planet orbit (angular velocity). For each planet, the angular velocity is set differently - the closer the planet is to the sun, the higher the angular velocity. Since we assume the orbit to be on the x-z plane (y-coordinate = 0), we can compute the new x and z coordinates at the next time instant using $x = sin(\omega t)$, $z = cos(\omega t)$, multiplied by a distance factor that controls the size of the orbit. Again, for different planets, the distance factor is different. We also set each planet to rotate about its own axis to simulate reality: each planet rotates 90deg around the x axis, and γ degree around the y axis, and $\gamma\omega t$ degrees around the z axis, with γ, ω determined by the degree of rotation of each planet in reality. Finally, we also set each planet model to rotate according to scene rotation. The sample code below shows how the orbit and rotation of the Venus planet is controlled.

```
/* VENUS */
glm::mat4 model_venus;
xx = sin(glfwGetTime() * PlanetSpeed * 0.75f) * 100.0f * 3.0f /factor;
zz = cos(glfwGetTime() * PlanetSpeed * 0.75f) * 100.0f * 3.0f / factor;
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture_venus);
model_venus = glm::translate(model_venus, point);
model_venus = glm::rotate(model_venus, glm::radians(SceneRotateY), glm::vec3(1.0f,
    0.0f, 0.0f));
model_venus = glm::rotate(model_venus, glm::radians(SceneRotateX), glm::vec3(0.0f,
    0.0f, 1.0f));
model_venus = glm::rotate(model_venus, glm::radians(SceneRotateZ), glm::vec3(0.0f,
    1.0f, 1.0f));
model_venus = glm::translate(model_venus, glm::vec3(xx, 0.0f, zz));
model_venus = glm::rotate(model_venus, glm::radians(-90.0f), glm::vec3(1.0f, 0.0f,
    0.f));
model_venus = glm::rotate(model_venus, glm::radians(-132.5f), glm::vec3(0.0f, 1.0f
    , 0.f));
model_venus = glm::rotate(model_venus, (GLfloat)glfwGetTime() * glm::radians
    (-132.5f) * 0.012f, glm::vec3(0.0f, 0.0f, 1.f));
glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm
    ::value_ptr(model_venus));
Venus.Draw();
/* VENUS */
centrepos = projection * view * model_venus * glm::vec4(tmpPoint, 1.0f);
PlanetsPositions[2] = glm::vec3(centrepos.x, centrepos.y, centrepos.z) / centrepos
    .w;
```

4.4 Rendering planet names next to moving planets

I used the `RenderText` function and separate text shaders from Tutorial 6_2 to render the planet names in Arial font. When computing the orbit and rotation of each planet, I also store the planet position coordinates in an array by multiplying perspective projection matrix with camera view matrix and the planet model movement matrix (see code fragment below).

```
centrepos = projection * view * model_sun * glm::vec4(tmpPoint, 1.0f);
PlanetsPositions[0] = glm::vec3(centrepos.x, centrepos.y, centrepos.z) / centrepos
    .w;
```

I then render the planet names with the respective x and y planet position coordinates using othogonal projection, so that the planet names are rendered in 2D onto the screen at the planet position, appearing as if the name moves with the planet. Note that I had to map the position coordinates (range -1 to 1) to the size of the window, as seen in the code fragment below.

```
//Render planet names
RenderText(textShader, "SUN", (PlanetsPositions[0].x + 1) / 2 * WIDTH, (
    PlanetsPositions[0].y + 1) / 2 *HEIGHT, 1.0f * shrink, glm::vec3(0.5, 0.8f, 0.2
    f));
```

5 References

Below are references that I heavily relied on to complete this assignment.

1. Sphere construction: http://www.songho.ca/opengl/gl_sphere.html
2. Sample solar system with OpenGL: https://github.com/1kar/OpenGL-SolarSystem