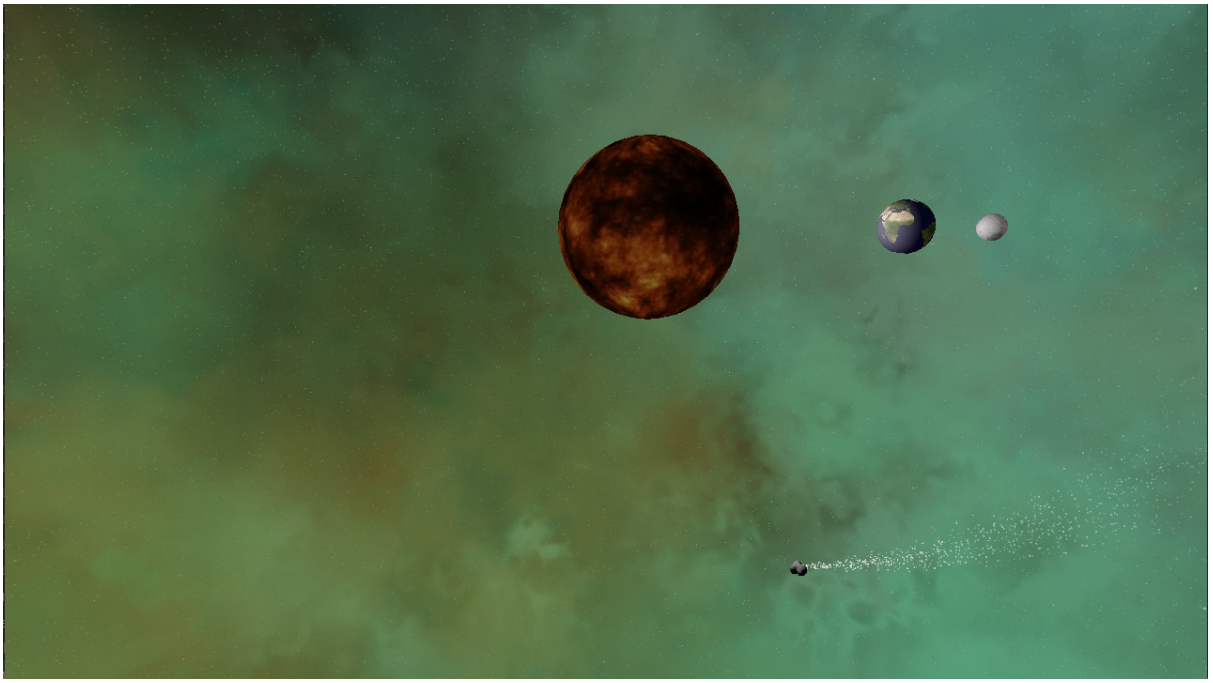
# INFO-H502 Project Report

2022-2023

VAN BEGIN Nathan, KIEFFER Axel



## Introduction

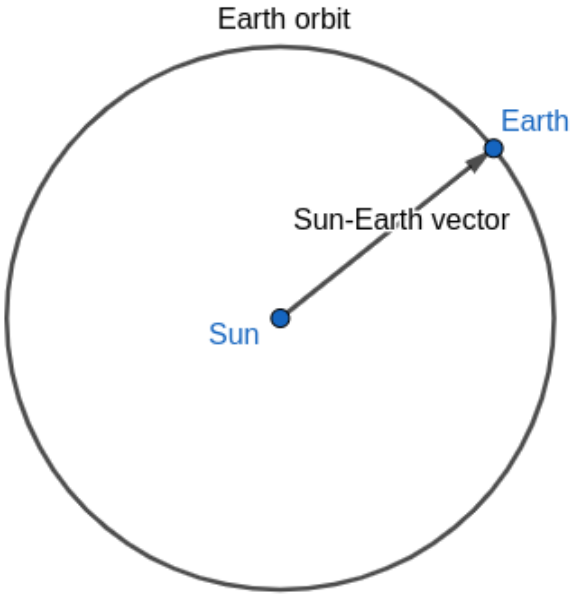
For the VR project, we choose to make the implement to following idea :
A simplified solar system in which we can move around using the Z,Q,S,D keys and the mouse, with a sun, an earth in orbit around the sun, a moon in orbit around the earth, and a particle emitting asteroid roaming around.

## The Sun

The sun is a sphere emitting light located at the origin. We found a video of flowing lava on the internet. We took 20 frames from that video, converted them to jpegs, and loaded them as textures that we put in a sunTextures[] array. Then we go through that array back and forth, (in one direction then the other) and apply the current indexed texture to the sun. In this way, we obtain an animated texture on the sun, without a continuity jump between the first and the last frame of the animation.

# The Earth

The movement of the earth around the sun is implemented in the following manner :



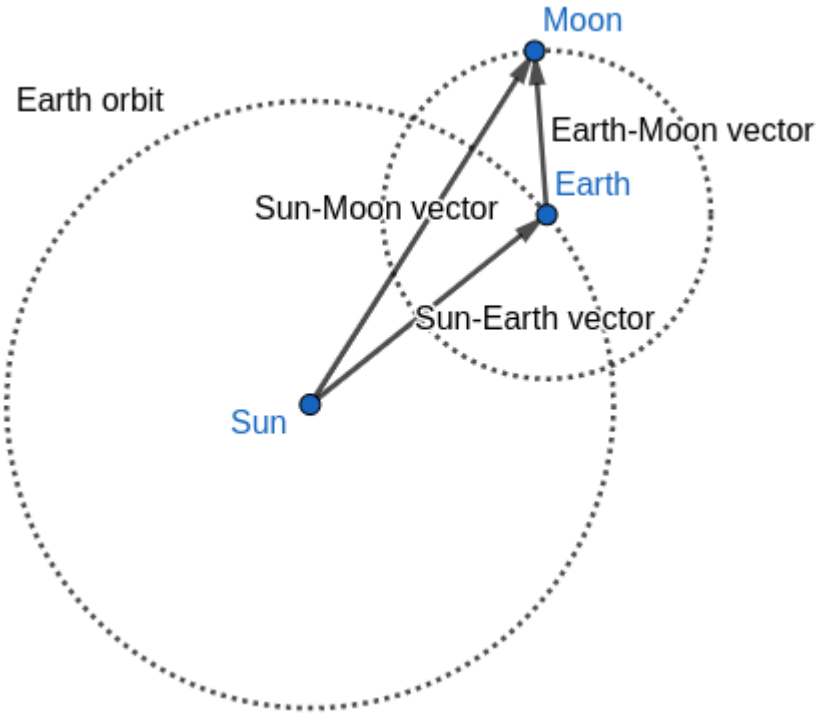

We have a **Sun-Earth** vector of constant magnitude rotating at constant angular speed around the origin (where the sun is located). Then we have to convert this vector from spherical coordinates to cartesian coordinates because the `glm::translate()` function takes a cartesian vector as an argument.

This was done using the following formulas :
$x = r \,.\, cos(\theta)$
$y = r \,.\, sin(\theta)$
$z = 0$
In our case we made the earth orbit in the x,y-plane, thus allowing us to maintain the z component at 0, simplifying the calculations a little.

To draw the earth, at each step of our “gameloop” we followed those next steps:

- We rotate the earth around the $(0.0, 0.2, 1.0)$ axis by an angle $c \,.\, \theta$ using the `glm::rotate()` function. The 0.2 factor is used to tilt the axis of the earth a little bit to simulate the real life tilting of the earth. The c constant is added to make the angular speed of the earth around the sun different from the angular speed of the earth around her own axis. Allowing the earth to have a rotation along her own axis, and a new face of the earth to be exposed to the sun at each moment, simulating the day-night phenomenon as it happens in real life.
- Then we translate the earth to the position $(x, y, z)$ computed previously using the `glm::translate()` function.
- Then we update the M,V,P matrices of the earth in the shaders.
- Then we draw the earth using the `draw()` function
- Then we undo the translation of the earth by doing an opposite translation
- Then we undo the rotation of the earth by doing an opposite rotation.

# The Moon




To make the moon orbit around the earth (itself orbiting around the sun), it was a little bit more complicated. We had to convert both the **Sun-Earth** vector and the **Earth-Moon** vector from spherical coordinates to cartesian coordinates. Then we could add those two vectors and obtain the cartesian **Sun-Moon** vector. From then on it was relatively easy to draw the moon at the correct location in a similar fashion as was done for the earth.

# The comet

Firstly, to create the particle cloud representing the tail of the comet, we created a particle struct holding all the parameters of the particles, like their position and their velocity, and since we thought that we might need to add a color to the particles later we also added a color attribute. We also added floats representing the time to live of the particle and its size.

Now that the struct was made we tried to draw the simplest cloud possible which is a simple particle with an infinite lifetime and no velocity. This example revealed an issue with our attempt, being that the square particle might not always face the camera. To solve this we used the `glm::lookAt()` function that allows a model to face something. Here we wanted the particles to face the camera at any time.

The next step was to add some velocity to our particle and give it a finite lifetime. Since we wanted the comet to move, we needed to add the emitter position to the position of the particle, so when a particle dies, it can reappear at the correct emitter position, synchronized with the displacement of the comet. Now that we had this figured out, it was trivial to add more particles.

To apply our particle system to the future comet we needed to find the velocities of each particle. The particle velocity will be more or less the opposite vector of the velocity of our comet. Since we use cosine and sine functions to describe the position of the comet, the velocity of the particle is the derivative of these functions, which is really easy to find with some basic trigonometry. Then we took the opposite of this vector and added some random numbers to each component of each particle to simulate the aspect of a real cloud. Now all that was left was to find a 3D mesh for the comet and draw it at the position of the particle emitter and we had our nice-looking comet.

# The Skybox

To show the cosmic background, we added a simple skybox textured with textures generated on the following website [https://wwwtyro.net/](https://wwwtyro.net/) (space 3D section of the website)