

Assignment 2: Planets Rotation

Student Name: Kevin Pruvost (潘凱文)

Student ID: 2021400603

Table of Contents

[Table of Contents](#)

[Introduction](#)

[Compilation](#)

[Demonstration](#)

[Content](#)

[How to use it](#)

[Controls](#)

[Additional Interactions](#)

[Code Architecture](#)

[Documentation](#)

[DebugInfo](#)

[Log](#)

[Shaders](#)

[Shader_Base](#)

[Shader](#)

[Meshes](#)

[Face](#)

[Vertex](#)

[VertexNormalTexture](#)

[Mesh_Base](#)

[Mesh_Sphere](#)

[Mesh_Obj](#)

[Mesh](#)

[Light](#)

[LightRendering](#)

[PointLight](#)

[PointLight_Shader](#)

[Entities](#)

[Entity_Skeleton](#)

[Entity](#)

Text

Character

Font_Base

Font

Text2D

Text3D

Rendering

Rendering

General

Camera

GUI

Input

Obj

Window

OpenGL_Timer

Texture

Quaternion

Important Points/Personal Reminders

Introduction

For this assignment, the main theme was **Planets Rotation**.

The objectives here were:

1. Set the orbits, size and textures of the planets
2. Add keyboard control to increase/decrease the rotation speed
3. Add fonts. Each planet moves with its name

Compilation

This project and all of my CG projects will be compiled with CMake, if you open the project directly with Visual Studio, you should be able to directly compile it.

Though, as CMake permits it, you will be easily able to compile on other platforms.

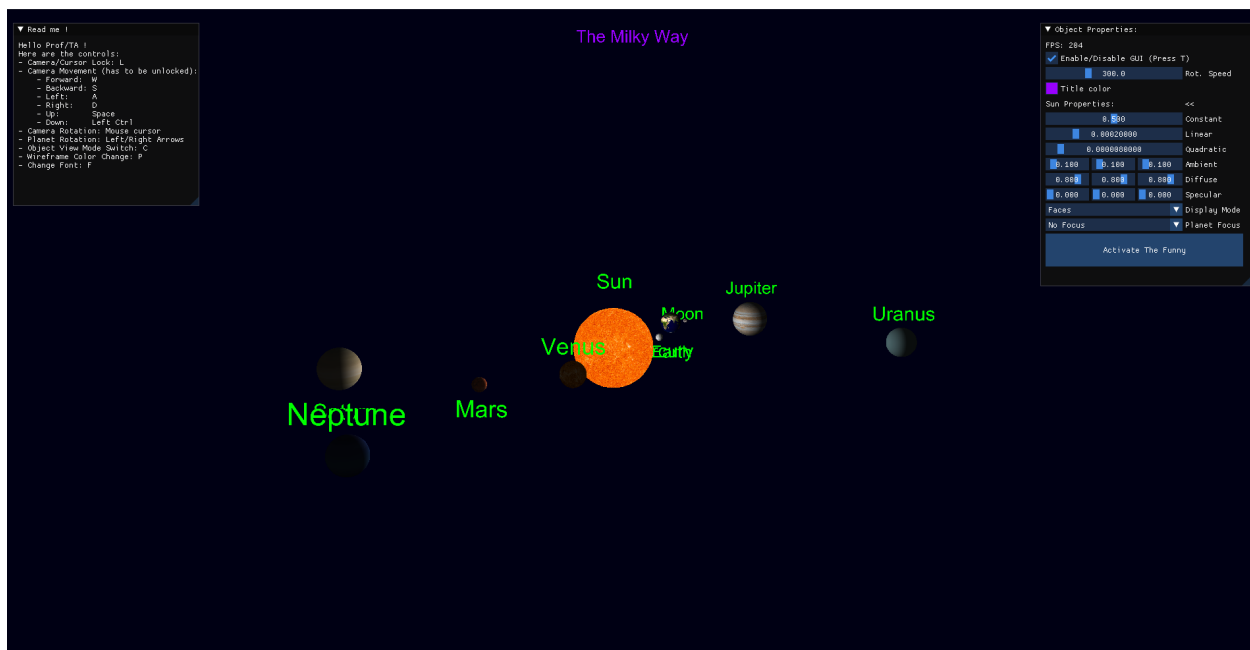
Demonstration

Content

For my Assignment, I got all these points covered:

- Textured planets
- Font change
- Rotation Speed Control
- Bonus:
 - Camera Movement + Rotation (can be enabled/disabled)
 - Simple Lighting (with the Sun being a PointLight)
 - Focus View System
 - Display Mode (Vertices, Wireframe, Faces)
 - Luminosity parameters Control
 - Enable/Disable GUI

How to use it





You can launch the `Assignment2.exe` directly, if you already have Visual C++ Redistributable.

The program was compiled in Release mode.

As it is displayed in the program, here are the controls by order of priority for the assignment:

Controls

- Change Font: F
- Enable/Disable GUI: T
- Change Rotation Speed:  
- Object View Mode Switch: C
- Wireframe Color Change: P
- Camera/Cursor Lock: L
- Camera Movement:
 - Forward: W (or Z on AZERTY layout)
 - Backward: S
 - Left: A (or Q on AZERTY layout)
 - Right: D
 - Up: Space
 - Down: Left CTRL

Additional Interactions

- You can change the `Title color`
- You can change the `Planets Rotation speed` with a slider
- You can change every parameters related to the sun's emitted light
- You can change the Display Mode on the `Display Mode` menu.

- You can select a Planet to focus and that will lock your camera on, though you can move and the camera will only rotate towards the focused planet.
- You can click on the `Activate the Funny` button to replace every planet with a giant rat

Code Architecture

The Complete Documentation is available in the project, in the Code Architecture part will only be explained a small description of what classes do.

Documentation

There is a documentation available [here](#) or in `docs/index.html` if you want to have a better view on the classes I made.

DebugInfo

Contains classes and code about Debugging & Logging information about the studied and tested processes.

Log

Centralizes the whole logging process.

Shaders

Shader_Base

Contains every variables used for OpenGL to manage every shader operations, through construction/deletion to writing variables to the GPU...

Shader

Contains an ID to a Shader_Base object so that Entities can carry shaders with a lighter structure.

Meshes

Face

Face structure, contains indices to vertex positions, normals and texture coordinates.

Vertex

Simple Vertex structure (x y z).

VertexNormalTexture

Vertex structure also containing normals and texture coordinates (x y z, nx ny nz, s t).

Mesh_Base

Base class of all meshes.

Mesh_Sphere

Sphere Mesh constructed from sphere parameters.

Mesh_Obj

Mesh constructed from an Obj object, to load .obj files into meshes.

Mesh

Contains an ID to a Mesh_Base object so that Entities can carry meshes with a lighter structure.

Light

LightRendering

Static class taking care of Light Rendering and UBO data related processes.

PointLight

Point Light class, treated as an Entity, it also contains parameters for light rendering.

PointLight_Shader

Structure containing every needed information for shaders to render light on objects. The most important feature of this structure is that it is adapted to the std140 layout.

Entities

Entity_Skeleton

Base class of Entities.

Entity

Manages key components of an entity such as position, rotation, shaders, meshes, calculating needed matrices, ...

Text

Character

Contains and manages information about characters loaded from fonts.

Font_Base

Carries information about loaded fonts, characters and all needed metrics.

Font

Only carries an ID to a Font_Base object to make fonts usage lighter.

Text2D

2D text only rendered directly on the Camera's screen, not considering the 3D Point of View.

Text3D

Treated as an Entity, has the particularity to carry information about rendering a 3D text, billboard is active on these instances.

Rendering

Rendering

Handles everything about rendering all types of Entities.

General

Camera

Manages everything about the point of view we need for the view matrix.

GUI

Manages the overall GUI, mostly written with <https://github.com/ocornut/imgui>.

Input

Manages everything related to input from keyboard and mouse.

Obj

.obj files Parsing/Loading class, for now, only vertices & triangle faces are handled.

Window

Inherits Input and manages everything about the OpenGL window.

OpenGL_Timer

Timer based on GPU time metrics.

Texture

Contains and loads everything about textures.

Quaternion

Serves as a basis for rotation purposes.

Important Points/Personal Reminders

- Uniform Buffer Object tend to be very useful within contexts where uniform variables do not change through every loop. We can just send these variables to Uniform Buffer Objects to make less calls to the GPU.
- Getting Uniform Locations is a very heavy process, it is better to store the return values CPU side to limit GPU calls.

1 (0.37%)	55	const Shader & shader = entity.GetFaceshader();
40 (14.87%)	56	const glm::mat4 & model = entity.GetModelMatrix();
	57	
	58	shader.Use();
	59	
	60	// Heaviest line (~40% time passed here in the function)
113 (42.01%)	61	auto id = glGetUniformLocation(shader.Program(), "model");
1 (0.37%)	62	glUniformMatrix4fv(id, 1, GL_FALSE, glm::value_ptr(model));
	63	
12 (4.46%)	64	glUniform1i(glGetUniformLocation(shader.Program(), "_texture"), 0);
	65	glActiveTexture(GL_TEXTURE0);