# Assignment 1: Introduction to the Framework & the base rendering pipeline

Real-time Graphics 2024-2025

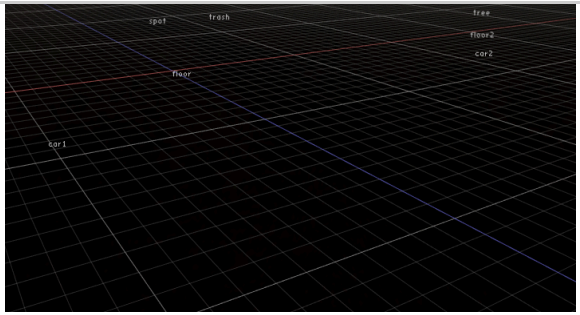
## 1 Introduction

The goal of this assignment is to be familiar with the framework, and implement a basic scene scene representation system that we are going to use in the following classes.

In this lab you are going to work on the files *src/pipeline/renderer.h* and *src/pipeline/renderer.cpp*, so try to read and understand what is going on at the files.

## 2 Objectives

Going from rendering an empty scene, to ending on a scene like the following, while also fixing the visual bugs from the rendering issues.





## 3 Assignment

In this assignment, you are asked to implement some basic infrastructure for rendering a scene, familiarizing with the framework.

Nodes that we are going to render as is, since this is going to bring problems on the following practices.
Since we want to render, we will need a structure that contains **AT LEAST** (for other exercises and assignments you might need to add more parameters):

- Reference to a *mesh*.
- A *model matrix*.
- A reference to the *material* we are going to use.

Take a closer look at the *src/pipeline/renderer.h* for a suitable place to implement.

## 3.2 Parsing the scene in order to generate Render Calls

The next step is to fill an array of render calls, based on the scene. For this process you will need to iterate through all the entities in the scene and if you find a Prefab Entity, add a new render call to the list with the data of each one of its children.

Take a closer look at the *src/pipeline/renderer.cpp* for a suitable function in order to implement.

is defined on the render call. For this, you can use the *Renderer::renderMeshWithMaterial* function.

## 3.4 Ordering Render Calls

One issue of rendering transparent surfaces with the depth buffer activated, is that ordering is really important. In order to avoid the issue that is shown in slide 25 of the presentation, it is really important to order the render of the meshes according to its render mode.


We should see the car from the other window.

If the render mode is transparent, they need to be rendered from further to closer to the camera, in order to prematurely avoid filling the zBuffer.
In the case the render mode is opaque, order needs to be from closest to furthest from the camera, since the zBuffer works fine for this use case, and by doing it in this order we minimize overdraw.

## 3.5 EXTRA: Frustum culling

One of the easiest optimizations that we can do on a scene of any kind is to **only render the objects that the camera is seeing**. There are multiple

On the *Camera* class you can find the functions for generating the frustum, and for testing some **shapes** with the frustum of the camera instance.

The only other thing that you need is a way to generate this shape from the Mesh or the Nodes in your scene.

## 4 Where to start

1. Read the slides and the **assignment** carefully. Understand well the tasks that are being requested.
2. Try to follow the steps of the assignment and stop to read the slides if you have any issue with understanding the mentioned concepts.
3. Steps are more detailed than normal so it should be easy to pick up some speed!