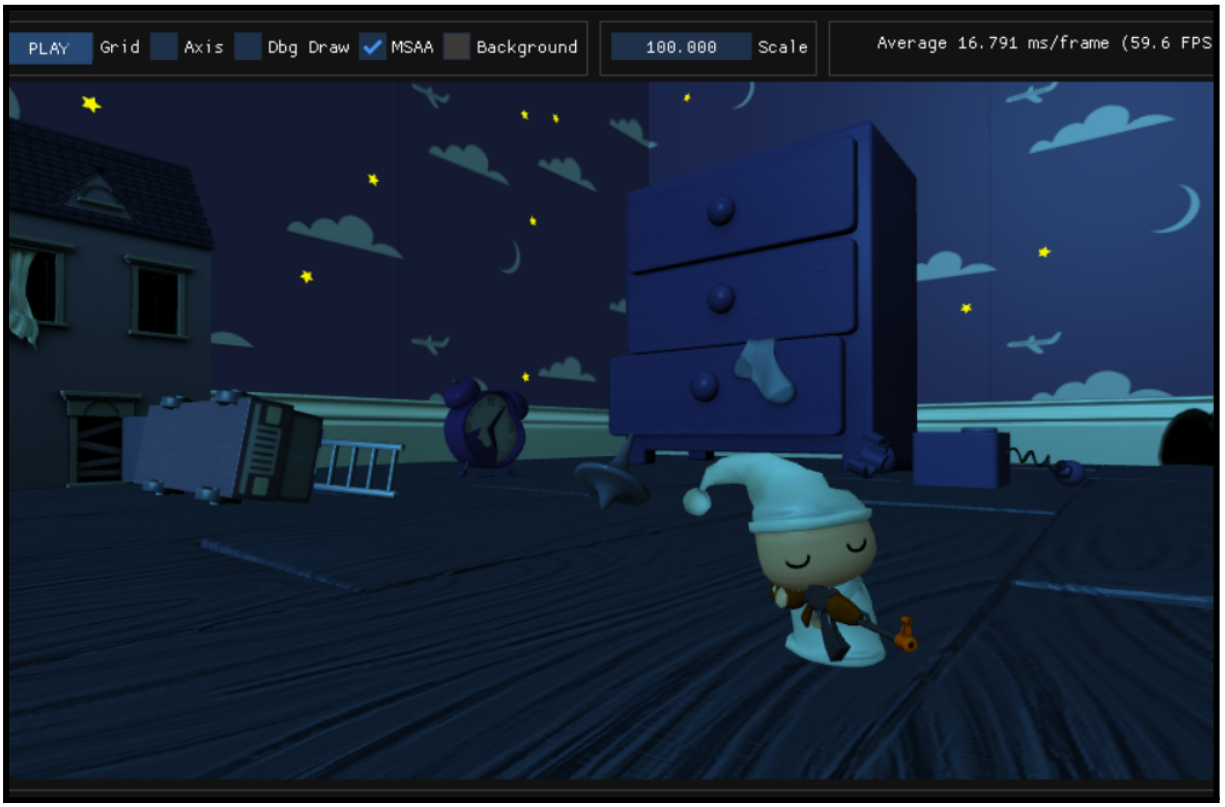
Assignment 2 - Game Engine

Overview

For our second assignment, we aim to deliver a complex scene edited from scratch and it will be serialised in your format.
To validate our purpose, we want to use Survival Shooter Unity tutorial assets to generate the scene. These assets are provided along with this statement. We also recommend that you search Survival Shooter Unity tutorial screenshots using Google to see the kind of scene we are expecting.



Creating a scene editor and rendering high-quality assets with good performance is a complex goal. For this reason, we will do this assignment altogether as a single team (with a single repository) but divided into 5 groups:

- Scene Management
- FileSystem
- Spatial Partitioning and Frustum Culling
- PBR shading and Materials
- Analytic lights and Normal map

The members of each group will be provided at the end of the first assignment. Keep in mind that, even divided into groups, you are a whole team, with a single repository and dependencies between groups. For example, the FileSystem team can't serialise the whole

scene until Scene Management has not coded the Scene Hierarchy. On the other hand, the File System group should also be efficient which means that they shouldn’t be stopped until Scene Management finishes coding the Scene Hierarchy. In this case, they can start, for example, serialising textures or meshes into binary files.

Below will be detailed the requirements of the assignment for each group.

Scene Management Group

We aim to deliver a complex scene structure using hierarchical GameObjects. The scene must be created from scratch using ImGui to:

- Add and remove GameObjects
- Add and remove Components to GameObjects
- Edit Component Properties
- Link GameObjects to other GameObjects

All the GameObjects must have a Local and a Global Transform that must also be edited using ImGui widgets for translations, rotations and scales.
There should be a MeshRenderer Component that can be added to any GameObject. The MeshRenderer Component should contain a mesh and its corresponding diffuse texture that will be enhanced later (by another group) with a full Material definition.

FileSystem Group

The whole scene must be serialised and unserialized. This means serialising the GameObjects hierarchy with its components and also serialising assets like meshes or textures into binary Files.
This group must work coordinated with the Scene Management group to properly serialise the scene hierarchy but also with all the other groups to serialise its assets (textures, cubemaps, lights data, normal maps, etc.).
The final delivery should start automatically unserializing the generated Survival Shooter scene. Any scene should be loaded or saved using an ImGui menu option.

Spatial Partitioning and Frustum Culling Group

To render a big scene maintaining a good performance we will need to cull the objects that are inside the camera frustum. The list of objects of the scene can be obtained by iterating the GameObjects hierarchy and picking all the MeshRenderer Components.
To efficiently do the culling we want to take the mesh's bounding boxes and convert them to Global coordinates (using GameObjects Global transformation) and finally check whether the bounding boxes are inside or outside of the camera frustum.
As the number of objects in the scene increases, the Frustum Culling cost also does it. For this reason, we want to store all those bounding boxes in a Spatial Partitioning structure that will reduce the linear complexity of the algorithm.

Is up to you to decide which spatial partitioning structure to use: quadtree, octree or kdtree.
The Spatial Partitioning Group must be synchronised with the Scene Management group to access the Scene hierarchy and the GameObjects transformations. For that reason, it's recommended to start by coding Frustum OOBB intersections and building the Spatial Partitioning structure while waiting for that dependency.
The Spatial Partitioning structure, the Camera Frustum and the Mesh's OOBB must be rendered using debug draw primitives demonstrating that both algorithms are working properly.

PBR Shading and Materials Group

We want to render Survival Shooter quality assets using a PBR Shader. This group must implement a Phong BRDF with Lambert Diffuse and Fresnel. The user must be able to tweak the following shader parameters:

- Diffuse Colour
- Specular Colour (Fresnel at 0 degrees)
- Shininess

These parameters are the material parameters and must be edited using ImGui as part of the MeshRenderer Component edition Material parameters can be edited as a single value for the whole mesh or they can be specified as part of a texture. Survival Shooter assets contain diffuse and specular textures. Note that the alpha channel of the specular texture can be interpreted as shininess (scaled to a valid range).
This group has dependencies with the Scene Management group but can start working by prototyping using the PBR Shader using ShaderEd.

Analytic lights and normal map Group

We want to render our scene using different types of lights:

- Ambient
- Directional
- Point
- Spot

Lights must be added and removed and their parameters must be edited. So, they can be a Component of a GameObject.
The PBR Shader from the previous group must be modified so that not one but multiple lights are applied to the whole scene.
Furthermore, this group is responsible for adding the Normal map technique to the rendering shader and adding functionality to select the Normal texture as part of the Material Component editor.
Finally, this group is also responsible for adding a sky CubeMap to the scene and rendering it. The CubeMap textures can be changed in the editor.
This group has dependencies with the Scene Management group and the PBR Shading group. I recommend starting working from the sky CubeMap rendering that has no dependencies.

Completion Test

The delivery must be a **Build** zipped in its folder inside *"Second Assignment"* named after your engine. The release should be a **zip** containing:

1. **README.md**:
    - Short description of the engine and link to the github.com page of the project
    - How to use the engine, detailing the controls and any specific action
    - Additional functionality in the engine outside of the assignment requirements
    - Additional comments for the teachers to understand some part of the engine

2. **LICENCE.md**:
    - Chose a licence that fits your project - https://choosealicense.com/

3. **Project files:**
    - Executable compiled in Release with all necessary DLL files
    - A resource folder with all the media files (fbx, png, dds,...)
    - *No other file must be there!* - Be sure to remove any code and unnecessary files

The folder structure should be the following:

```
> EngineName_v.X
    > EngineName
        - Game/ assets/resource files
        - .dll files
        - EngineName.exe
    > README.md
    > LICENCE.md
    > Licences (folder with all third party libraries' licences)
```

4. **Github:**
    - The repository under github.com must contain a copy of the build under the Release section.

Assignment Content

1. You must keep the Unity-like controls for the camera.
2. Also the configuration windows and the general menu from Assignment 1 must be there.
3. Upon starting the editor the scene must be already loaded containing:
    a. The room using the assets from "Survival Shooter" tutorial from Unity

    b. The setup should feel like the player and the enemies would be able to appear and play
4. The scene must be loaded from your own format (json/…)
5. The active camera should use frustum culling accelerated with a quadtree (or octree/kd-tree/aabbtree).
6. The scene should be editable with gizmos and saved again.
7. Materials should have diffuse and specular maps.
8. Have all types of lights: directional light, point and spot light. And must be shaded using PBR Phong shading.
9. The application must keep stable 30 fps at all time with or without vsync (at the university computers)
10. Have Play/Stop/Pause controls that ignore changes done during play.

The **zip** must be submitted before **January 18th 23:59** (folder closes automatically).

Grading Criteria

To **accept** a submission for grading, it must comply with:

1. The build is not malformed.
2. The code compiles and uses only english.
3. The release did not crash while testing.

Once accepted, the criteria is as follows:

- 10%: Repository & Commit structure (small commits with clear description)
- 60%: Code structure / Good choice of containers / Const-correctness and use of pointers/references
- 30%: Scene setup is ready for gameplay and has good graphical fidelity

We value the code to be simple and readable to achieve the maximum grade in each area of the code.