ECOLE NATIONALE D'INGENIEURS DE BREST

Virtual Museum Visit with BabylonJs

Final Report

Asma NAIFAR
17/12/2021

Table of contents

Figures

1. Introduction

The objective of this project was to try and build a 3D virtual visit of a museum holding differently themed pieces of art work from around the world.

Using Babylon.js, an open source Javascript library for displaying 3D graphics in a web browser, the goal was to create an immersive word where the user can interact with paintings and discover them whilst moving inside the actual building of the museum.

My main focus on this project was to build a cohesive experience and make sure that the user will get a taste of historical paintings around the world.

2. File Structure and technologies used in the project

The most important detail that I focused on while building this 3D visit is that the processes of loading the Javascript files in the browsers was not slow by any means and did not lag the slightest. For reason practically, I chose to work with **node.js** and **webpack** so I can render all these files on the backend and don't have to worry about importing any files and avoiding the problem of *circular dependenc*y[1] completely.

To explain more in detail the architecture of the project, I will first of all present all the technologies I used and afterward I will detail the file structure that I chose to work with.

Technologies:

Node.js:

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser (wiki)

npm:

npm is a package manager for the JavaScript programming language maintained by npm, Inc. npm is the default package manager for the JavaScript runtime environment Node.js.

Webpack:

Webpack is an open-source JavaScript module bundler. It is made primarily for JavaScript, but it can transform front-end assets such as HTML, CSS, and images if the corresponding loaders are included.

Why use Webpack?

Webpack is a great tool to manage dependencies and imports in ES6 and Node.js. It has the mechanism to build a dependency graph of all the files and their dependencies, and bundle all the files.

[1] In software engineering, a **circular dependency** is a relation between two or more modules which either directly or indirectly depend on each other to function properly. Such modules are also known as mutually recursive.

Since this project is based mainly on Javascript files, and to avoid any decrease of performance, I've used a simple, yet effective, configuration of webpack in the project so I can firstly, and as I just mentioned above, don't have to manage the different dependencies that will occurs as the source code increases in complexity and in quantity (although I made sure that the project file structure is as modular as it could be). Secondly, so I can simply generate one compact bundle Javascript file and use it in the main html page. I put in the figure just below the code for the webpack config file which was very important that it'll be set up before any actual code was written.

webpack.config.js > ...

```
const path = require("path")


module.exports = {
  target: "node",
  mode: "development",
  entry: path.resolve(__dirname, "src/app.js"),
  output: {
    path: path.resolve(__dirname, "dist"),
    filename: "app.bundle.js",
    library: 'app',
  },
  resolve: {
    extensions: [".js", ".mjs", "cjs"],
  },
  node: {
    __dirname: false,
  },
  module: {
    rules: [
      {
        test: /\.node$/,
        loader: "node-loader",
      },
    ],
  },
  devtool: "eval-source-map"
}

```

Figure 1 webpack configuration file

I've also used webpack to serve static assets used throughout the project such as the images of paintings, textures used for materials, skybox material and various audio files.

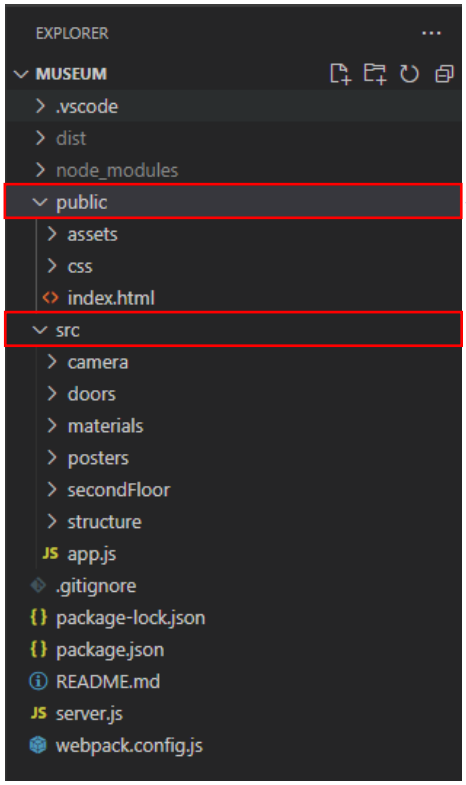
In this project, Webpack allowed me to focus more on the actual code development and offered me the freedom to perform much more complex tasks in a short amount of time.

File Structure:

Building the file structure, I've made sure that the Javascript files (in src folder) and the static files (in public folder) such as css, html and image and audio assets were separate. The main file called "app.js" is handled by the "server.js" file and served to the browser on the localhost URL

using port 8000 (Please make sure that this port is not currently used on your machine prior to loading the project).

Down below is the actual file structure of the project.



Public folder containing static assets

SRC Folder containing all Javascript files such as file setting up the word camera and the structure of the museum, etc.

3. Detailed description of the museum

Building the museum outer and inner structure was not an obvious task. I had to make sure throughout the entirety of this process that there was no collision between meshes and between the user (First Person shooter camera) and the meshed surrounding them.

In the following sections, I will detail the different steps that I did to make the base of the museum.

3.1. Grounding the world

The first step in building the museum model was to make sure that world is grounded and that the scene had the gravity activated so that all the structures inside the building will be on the floor.

3.2. Outer Structure

Building the main outer structure of the museum was very important to set up from the beginning the actual size of the building 30x30. To do this, I've used the function called "buildFomPlan" that the exist in the official documentation of babylonjs.

The goal from using the function is to quite literally build the actual structure from a floor plan by setting up the values of the outer corners of the building. This function also allows to define the windows and the doors of the structure simply by setting up the actual width and height of the door/window and then specifying the index of the wall it's will be integrated to.

For this project, I've defined the structure down below as well as one main entrance door and four windows in the main hall (two on each side).

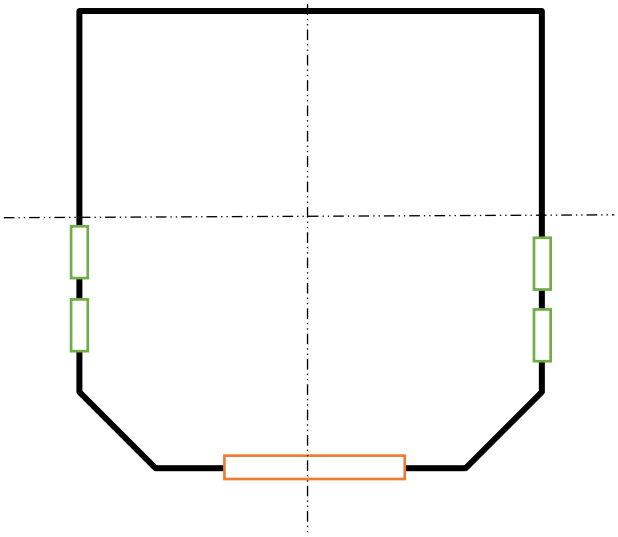


Figure 2 Outer building structure

Down below is a code snippet of the outer building definition mainly defining the values of the outer corner of the building.

src > structure > JS building.js > ...

```
1   import * as BABYLON from "@babylonjs/core";
2   import {corner,wall,win,buildFromPlan,windowSpace,door,doorSpace,} from "./customMesh/buildMeshFomPlan.js
3   import { createRoof } from "./roof.js";
4
5   export function createBuilding(scene) {
6     var exteriorBaseData = [-13, -15, 13, -15, 15, -13, 15, 15, -15, 15, -15, -13];
7     var baseData = [-12.9, -14.9, 12.9, -14.9, 14.9, -12.9, 14.9, 14.9, -14.9, 14.9, -14.9, -12.9];
8
9     var corners = [];
10    var exteriorCorners = [];
11
12    for (let b = 0; b < baseData.length / 2; b++) {
13      corners.push(new corner(baseData[2 * b], baseData[2 * b + 1]));
14    }
15
16    for (let b = 0; b < exteriorBaseData.length / 2; b++) {
17      exteriorCorners.push(new corner(exteriorBaseData[2 * b], exteriorBaseData[2 * b + 1]));
18    }
19
20    var d = new door(6, 7);
21    var ds = new doorSpace(d, 10);
22
23    var windowHall = new win(2, 6);
24
25    var windowSpace20 = new windowSpace(windowHall, 2, 2);
26    var windowSpace21 = new windowSpace(windowHall, 5, 2);
27
28    var windowSpace40 = new windowSpace(windowHall, 25, 2);
29    var windowSpace41 = new windowSpace(windowHall, 22, 2);
30
```

Figure 3 Code of outer structure of the building

From the same coordinates defined for the outer structure, I've used them to also build the roof of the museum with the help of a similar function as the "buildFromPlan" function and that was also taken from the Babylon.js official documentation.

Down below is a figure of the final result.

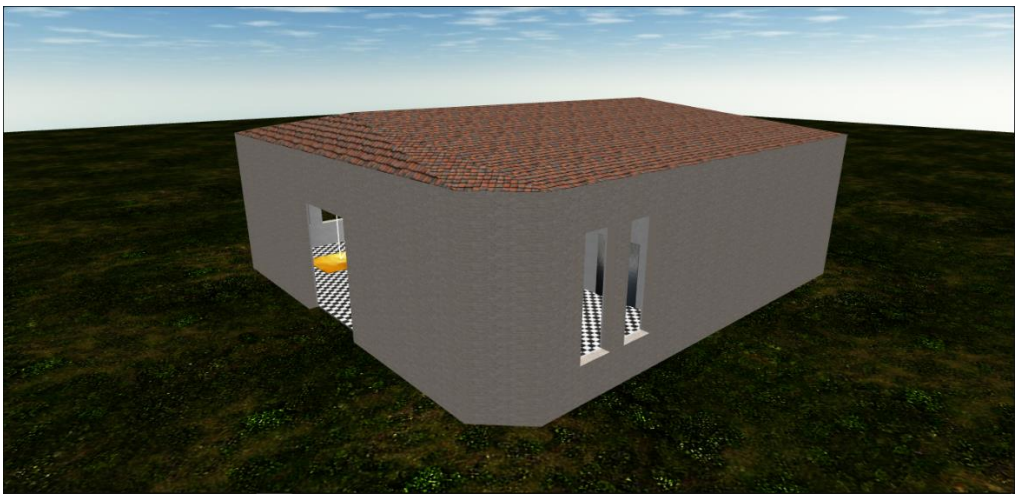


Figure 4 Outer structure and roof

3.3. Inner Structure

Before coding actual building and defining the interior structure inside the museum, I had to make sure that the actual interior building was not the same thing as the outer structure since because and the main reason for this is to be able to apply after word different material for the inner wall of the building and the outer walls.

To achieve this, I had to build another structure identical to that of the outer structure but it had to be less thick and it should not collide with the outer one.

The figure down below shows the end result of this process.

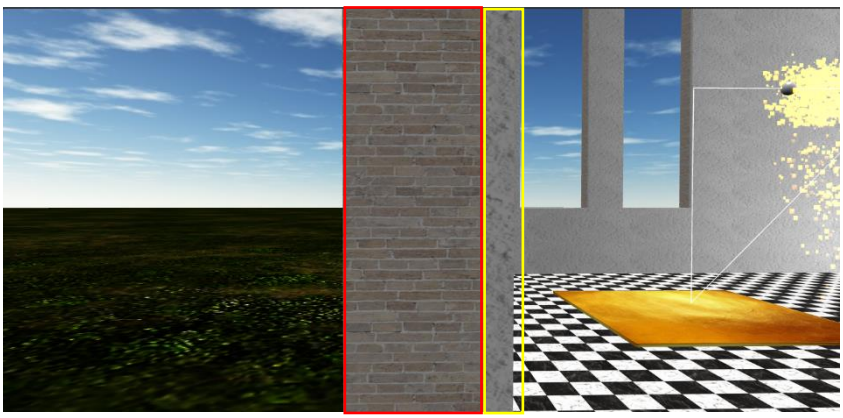


Figure 5 outer stucture (RED) Inner stucture (YELLOW)

3.4. The First Floor

The first floor was trickier to build since it had to fit three rooms and also the main hall. Alongside building the separation and the walls between the rooms, it was necessary to build a separate floor, inner walls and ceiling for each single room so that the customization of each one would be possible.

In the sections below, I will explain the processes towards achieving this, the themed of each room as well as the second floor.

3.4.1. The main hall

Although, it was the simplest part of the museum to do from a structural standpoint, since all the actual building is already in place, the main hall had to have something that attracts the user and be aesthetically appealing.

So, I've decided to go with a classic checked flooring that I scaled down so that the hall looked big from the inside as well as a central stone fountain that was particle emitter and emitted water like particles on a loop. These particles have various size and fall with gravity.

This fountain was also brought to life by the audio background music attached to the mesh of the fountain that had running water in its track so that the whole scene went together.

In the figure down below is a screen shot of the fountain.

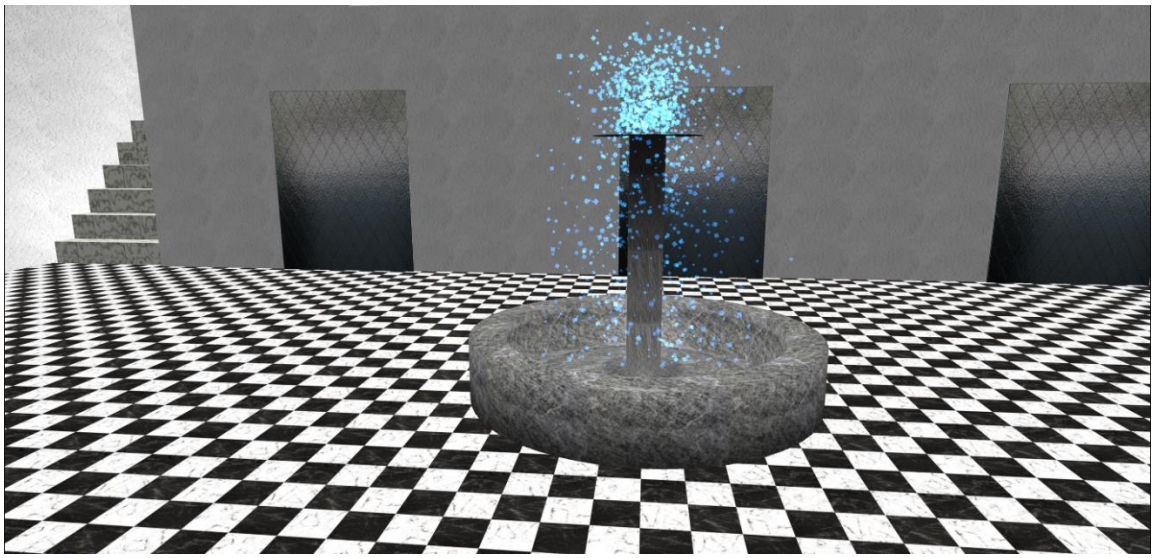


Figure 6 Fountain with a water-like particle system

Besides the central fountain, the main hall had two animated structures of a sphere continuously moving along a triangular track. I've attached to this sphere a golden particle system that falls with gravity as the sphere moves along its track. These structures rest on a golden platform to emphasise the gold of the particle system.

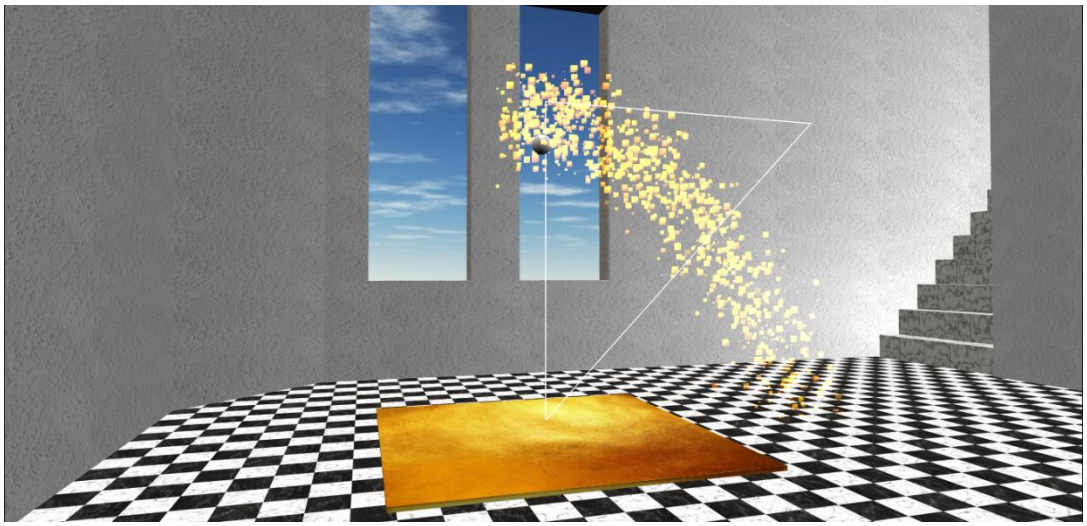


Figure 7 One of two main structures in the main hall

3.4.2. Inner Separation

As a first step of building the inner separation, I had to specify the plan of each room and its size so the whole inner structure would look as symmetrical as possible. I've went with a similar strategy as when building the inner and outer structure with the help of the "buildFromPlan" function as well. So, I defined the outer corners of the separation, the needed space for each moving door of each room, two staircases to go up to the second floor and four windows through which the main hall is visible.

The figure down below is the floor plan for the inner separation.

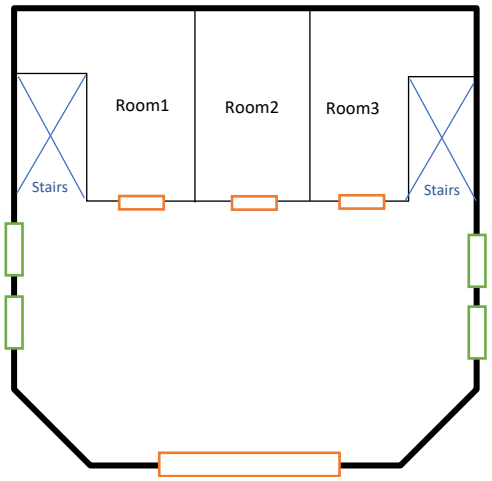


Figure 8 First floor *inner separation*

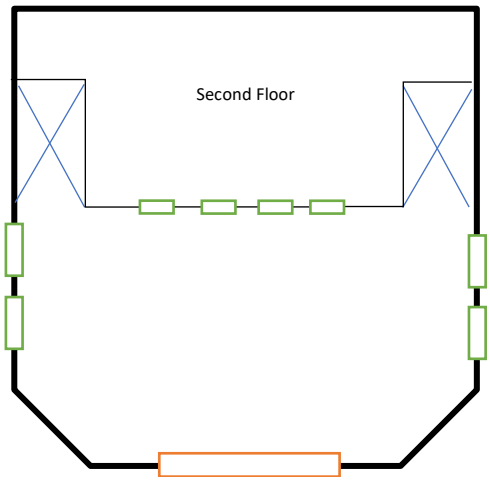


Figure 9 second Floor separation

For the material of the stairs, I've chose to go with a marble material that I imported from the module "babylonjs-procedural-textures" and went really well with the aesthetic the museum.

Below is a figure of one of the stairs showcasing the marble material.

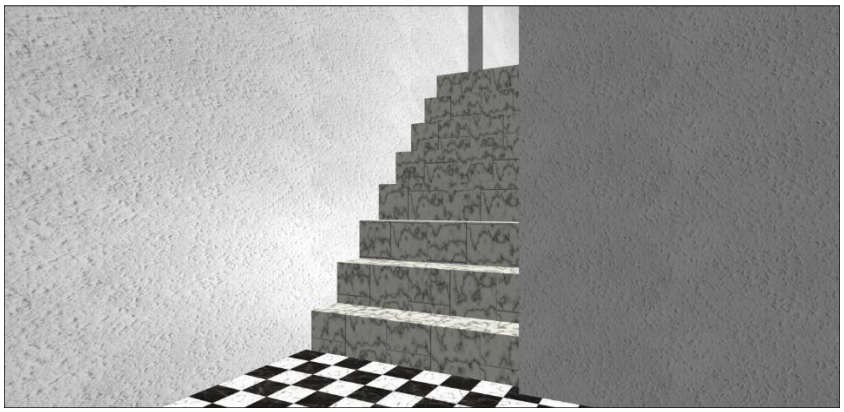


Figure 10 Marble stairs

Down below is a figure showcasing the final interior separation with a sliding door for each room.

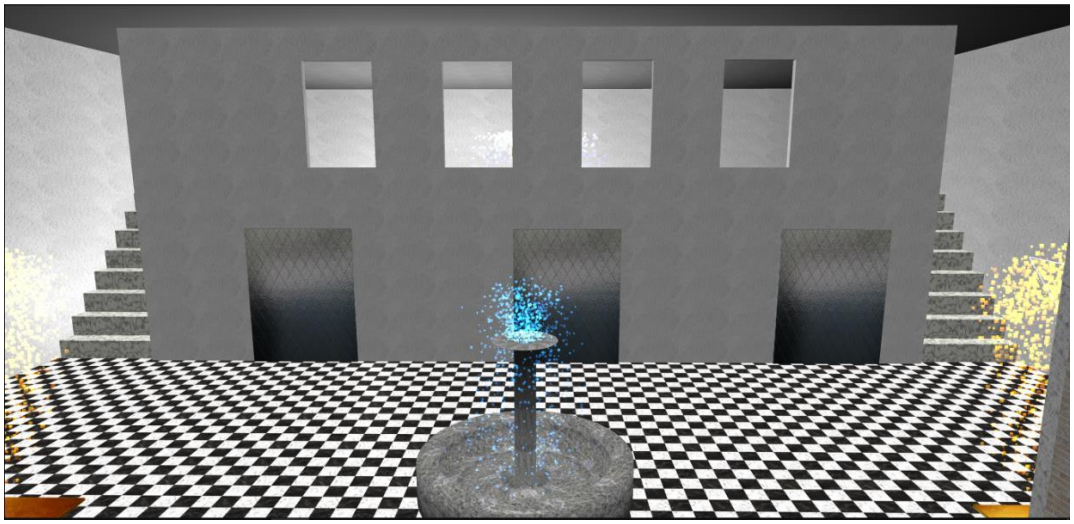


Figure 11 final interior separation

3.4.3. The Galleries

For each gallery entrance, I've went with a sliding door that slide to the left when the user approaches the door and automatically closes. I achieved this by attaching a collider to the camera and checking for collision with this collider. If the user gets near the door, this will trigger the animation of each door.

Below is a code snippet for this behaviour.

```
let actionDoor3 = new BABYLON.ExecuteCodeAction(
  {
    trigger: BABYLON.ActionManager.OnIntersectionEnterTrigger,
    parameter: {
      mesh: door3,
    },
  },
  (evt) => {
    const anim3 = scene.beginAnimation(door3, 5.7, 2 * frameRate, false);
  }
);

collider.actionManager.registerAction(actionDoor1);
```

Figure 12 code snippet of sliding door

On the inside of each gallery there is three painting that go with three themes mentioned down below. The paintings have wooded frames and a marble plaque just below it that has its description on it.

Below is an example of a painting.

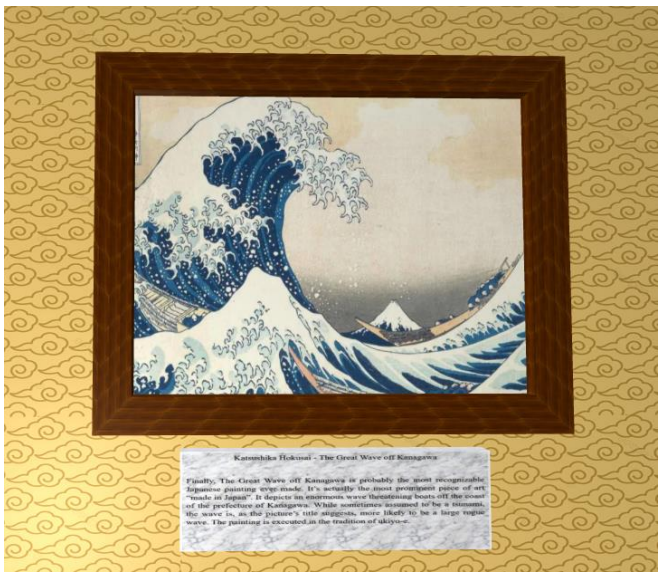


Figure 13 example of a painting

User interaction with paintings:

When the user clicks twice on a painting it will trigger a sound that say its title.

When the user press on the mouse for a long time (hold a click for a longer period of time) it will trigger a sound that describes the painting.

- **The First Gallery (room on the left): Traditional Japanese Paintings**

I dedicated the first room to traditional Japanese paintings. I've included three famous painting such as the wave and tiger.

I chosen also a beautiful traditional Japanese wall paper that go perfectly with this theme.

- **The Second Gallery (room in the middle): Tribute to the Dutch Artist Johannes Vermeer**

The second gallery is dedicated to my favourite painter Johannes Vermeer and a few of his famous painting including the girl with the pearl earring.

Johannes Vermeer was a Dutch Baroque Period painter who specialized in domestic interior scenes of middle-class life.

- **The Third Gallery (room on the right): Famous Arabic Paintings**

The third and final room hold a few of old famous Arabic paintings two of which are from Egypt by the painter Gazzar and Said.

Notice also in this Gallery the floor material which is a traditional red carpet that goes very well with the theme.

3.5. The second Floor

The second floor and the final part of the virtual visit is the home of a simple structure of a big golden sphere that has particle system attached to it.

The figure down below is a screenshot of this structure.

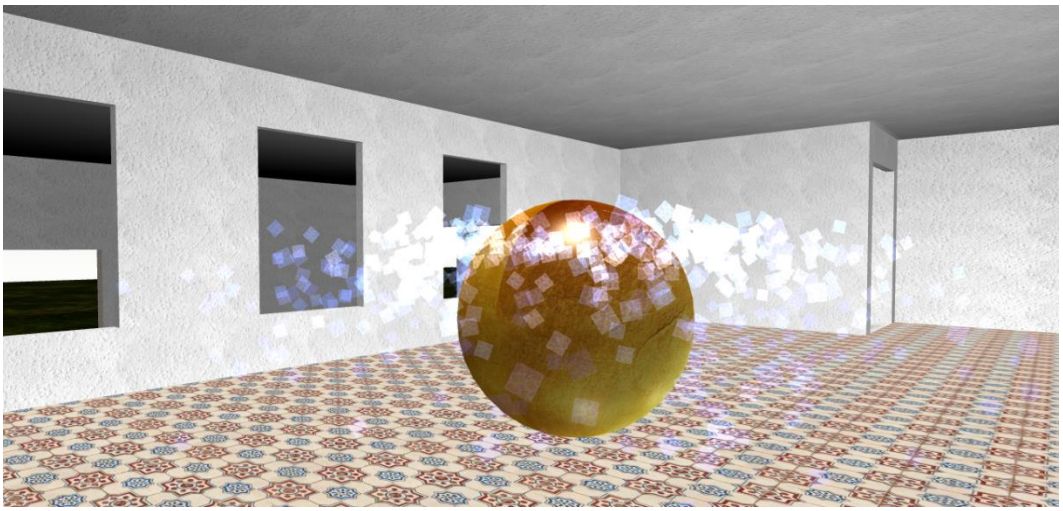


Figure 14 second floor structure

4. Conclusion

Babylonjs is a powerful tool to build very intricate and complex scenes that can render in the browser. There are many impressive scenes developed with babylonjs that recreate user immersive scenes. This project was introduction to the world of 3D development and a great personal creative outlet.