Interactive Graphics - Final Project

Damiano Ludovici 1697348

July 2020

Contents

Chapter 1

Introduction: Platfrom

1.1 Idea

The idea of this project is to create a platform game where the player is able to move a man though a corridor full of dangers, survive until the end and beat the final boss with at least one life left.

Figure 1.1: Game screenshot

1.2 User Guide

W: Move forward

S: Move backward

A: Move left

D: Move right

Space bar: jump

C: change camera setting

M: Turn on/off music

There is a tutorial at the beginning of the game, that appears in the advanced gui, and it's developed to not let the player continue if he doesn't know the principal commands (move and jump).
Clicking C is possible to switch to two different camera configurations:
- arcade mode that is the default one (and suggested). Very high, looking to the ground with distance visual limitations;
- immersive mode that is directed to the back of the player and it allows to see until the end of the corridor.
Clicking M is possible to switch the soundtrack to on/off. Effect sounds cannot be disabled because they are relevant component of the game experience.

Before the game starts, it is possible to change the difficulty in the main menu.
There are three difficulties:

easy: 10 lives;

medium: 5 lives

hard: 3 lives

If the difficulty is not chosen, the player starts the game in easy mode (10 lives).
Lives are displayed in the top right corner in game.

Chapter 2

Environment

2.1 Libraries

babylon.max.js:
A real time 3D engine using JavaScript library for displaying 3d graphics in a web browser via HTML5. It's a powerful open rendering engine that gives the possibilities to build meshes, models, lights, cameras and so on. "max" term is referred to the fact that it's a bigger library containing all the newest features.

babylonjs.loaders.js:
Library used to import external objects and models.

babylon.gui.js:
Library used to develop and display advanced user interfaces(gui).

babylon.lavaMaterial.js:
Library to import the lava material to apply to some meshes.

cannon.js:
Library used for physic contents.

2.2 Scene

A scene is a container where meshes, lights and cameras are placed and, thanks to the engine, is made to work. In Platfrom there are three scenes:
- Main menu scene: Basic scene with a camera and a simple environment(light, ground and skybox) that contains the imported main character that dances with music. Gui displays game title, difficulty options(radio buttons) and start button. Clicking on the button the application switches to Game scene.
It's possible to move the camera using the mouse. It's easy to notice that the main menu of the

game is not a static html page but part of the game itself.
- Game over/victory scene: Another basic scene with simple camera and environment with an applied gui. In the gui there is a textbox that changes dynamically if the player wins or loses. There is also a "play again" button that refresh the page if clicked.
- Game scene: The most important scene that contains all the game. This scene is analyzed in this section.

2.2.1 Models

dude

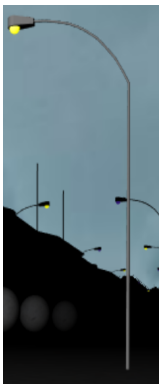
dude.babylon is an imported hierarchical model with a complex structure and texture used in the game as main character.
The function that permits the import is "BABYLON.SceneLoader.ImportMesh(id name, path, filename, scene, function (Meshes, particleSystems, skeletons) { ...} "
Meshes is an array of meshes that composes the model, particleSystem is a variable to handle particles and skeletons contains all the bones as shown in the following figure.
Inside ImportMesh function all the commands to handle Meshes are inserted like scaling, rotation, translation, shadows and physics.

NOTE: dude.babylon had imported animations in the code, so i edited the file setting all animations to null. There is no imported animation in the project as asked in requirements.
The following imported meshes are developed in the same way.

other imported models

(a) Apartment

(b) Mansion



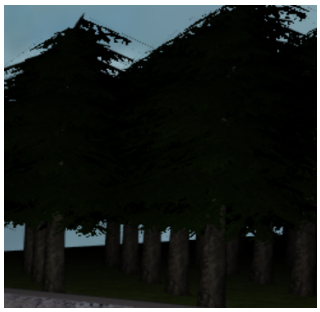
(c) Street Lamp

(d) Street bars

(e) Dog

(f) Dog sign

(g) Police car



(h) Tree

NOTE: cars, bars, trees, dogs and dog sign have their own textures, apartments and mansions have texture applied by myself.

Hierarchical model - The Final Boss

In order to satisfy the request for a more complex hierarchical model than the bear of the last homework, i decided to develop an end-of-level boss in which each component was animated and which showed its own hierarchical structure.
The model is composed of cubes, cones, spheres and a torus.
Each of them is linked to another with a parent attribute that every mesh has.
The bigger cube in the center (body) is the root parent of the boss.

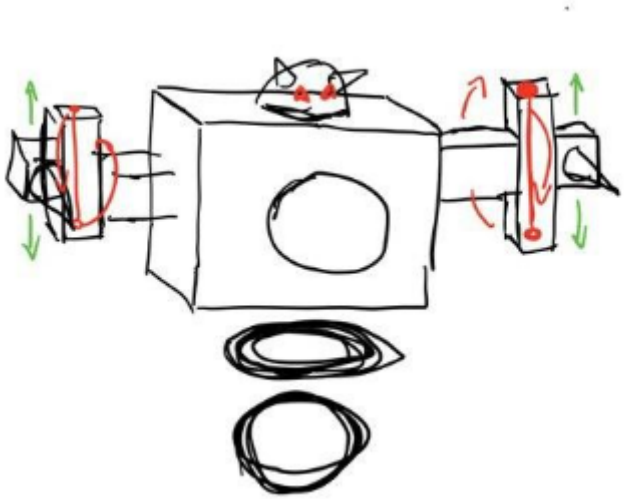


Figure 2.2: sketch of the model

In order to make the model more realistic and less fake, i rounded the corners of the body by intersecting a cube and a sphere.
It was possible applying Constructive Solid Geometries functions(CSG) that permits, giving in input two meshes, to have a new mesh that is the intersection of the input meshes.
The sphere in the center of the body is the only one that has an emissive color texture that lights the mesh also in the dark(without spotlights).
This sphere is the battery indicator of the final boss and since it is possible to observe him in the dark he gives a fear effect when he presents himself to the player's eyes in the final part of the level.

2.2.2 Camera

In the game scene there is a follow camera. "Follow" means that it's possible to define a mesh as lockedTarget and the camera will center the vision to it.
The chosen mesh is the imported "dude", so every time the player moves it, camera will translate with it.
Camera has common attributes(radius, heightOffset, rotationOffset) but i decided to limit them in order for example to not permit to see under the ground or to evade from the skybox with a big radius value.
It's possible to move the camera clicking and dragging with the mouse or using directional keys.

2.2.3 Lights and shadows

Lights

There are eight Point-lights. A point-light is a light source positioned in a point and it emits light in all directions.
Lights are enabled one by one based on player position and they are located in the center of a sphere inside the street lamp.

For every pair of street lamps one is working and the other is broken. Working and broken lamps are alternated in the corridor.
I set range to 400 so each light ends where a new one start.

Shadow

Shadows are entrusted to Shadow Generator function that takes in input a light source.
There is a shadow generator for each light. Every mesh that the generator have to map must be added to the shadow caster list. Only dude shadow is mapped for optimization purpose, details are explained in Optimization section.
Shadows are handled with a blur exponential mapping.
In conclusion it's needed to set receiveShadows value to true in every mesh material that you want they catch shadows(street, lava, grass).

2.2.4 Texture

In Babylonjs textures are applied to a material. There are different types of materials(lavaMaterial for example) and different type of texture.
Every object, model or mesh in the application has textures. The most used material is StandardMaterial and texture are images or colors applied to it.
To define what material is linked to a mesh is needed to fill the attribute "material" of the mesh.
The world is closed within a skybox to which sky images are applied as texture. The cube is bigger than the ground and it's possible to see it switching in immersive camera mode.

2.2.5 GUI

Babylon.GUI uses a DynamicTexture to generate a fully functional user interface which is flexible and GPU accelerated. In this application it is fullscreen so it will cover the entire screen and will rescale to always adapt to your rendering resolution.
Gui is composed of Rectangles and Textblocks that displays fps, life number and an hint section. All these components are above the screen and will follow you in the game.
There are also two different textblocks that are attached to a mesh(bars and wall). They will move based on mesh position and they are visible only if you are looking at them.

Chapter 3

Physic

In a platform game, gravity and collisions are the most important part.
To enable physic in Babylonjs it's only needed to apply to scene the following function:
scene.enablePhysics(gravityVector, physicsPlugin)
where gravityVector is a 3-dimensional vector that define the direction and the intensity of the simulated gravity acceleration and physicPlugin is Cannonjs.
To allow interaction between objects, the physics engine use an impostor, which is a simpler representation of a complex object. The imposter can be assigned physical attributes such as mass, friction and a coefficient of restitution. Cannon has a lot of different impostor shapes(box, sphere, ground, particle, plane, mesh) but to make simple computations i used only boxes, spheres and plane.
For example wrecking balls are spheres with spereImpostor with mass, friction, and restitution value.

3.1 Joint

Building a real wrecking ball was pretty hard taking the fact that it has to swing maintaining a fixed position.
I added a smaller sphere in top of the first one and i applied a sphereImpostor with mass 0. Mass 0 means it not suffers gravity so it remains in levitation. In the end i used the function addJoint to keep the two spheres at a fixed distance and i create a line which has them for extremes.

3.2 Hitbox

When we consider more complex meshes like dude, cars, dogs i decided to incorporate them in a invisible box and apply the impostor to it. Cannon gives the possibility to apply MeshImpostor but it's heavier(in terms of memory) than a common boxImpostor. After that box is set as parent of the mesh so if there is an interaction or an animation, mesh and box move in the same way.

3.3 Walls

The entire corridor is surrounded by invisible walls of mass 0 that impose the player to follow the predetermined path.

3.4 Interactions

Interactions between impostors are handled with registerOnPhysicsCollide that add a callback function that will be called when an impostor collides with another impostor. Every time there is a collision between two meshes it does what is written in the function.
registerOnPhysicCollide is inserted in scene.registerBeforeRender(function(){ ...}); that execute the function before the scene is rendered.
Since the rendering is started in a loop, collisions are monitored in every moment.
Interaction are all between a mesh and dude hitbox.
Wrecking balls, dogs, boxes, lava and bullets interaction with the player activates the callback function that removes a life and plays a scream sound.
If life value becomes equal to 0 the application switches to Game over scene.
Street, cars and grass interaction with the player enables jump command.

Chapter 4

Animation

Babylon.js has its own method to animate properly objects. It use frames, keyframes and framerate to generate packaged animations.
It doesn't support tweens.js library suggested in requirements so i decided to make animations by myself translating and rotating meshes in registerBeforeRender function and using physic features. Every time i discuss about axis in this section i am referring to local axis (axis relative to single mesh).

4.1 Wrecking ball

Knowing the structure explained above, the animation is done applying an impulse to the sphereImpostor.
"sphere.physicsImpostor.applyImpulse(new BABYLON.Vector3(0, 5000, 0), sphere.getAbsolutePosition());" where the first vector define impulse direction and intensity and the second one define the position(sphere location).
It gives to the sphere a linear speed that decrease in time. The commuter of the sphere is given by the joint with the other sphere that, imposing a fixed distance between the two, does not make it move in a straight line. Line joining the two spheres is updated in the registerAfterRendering so it follows the wrecking ball movement.

4.2 Dog

Player x position triggers dog animation that just translate the object in X axis.
After he reach a certain position dog is dispose. Dispose function delete the element from the scene. Position control and translation are inserted in RegisterBeforeRender function.

4.3 Explosion

Using physicsHelper (babylon feature) it's possible to apply a radial explosion impulse that, giving a 3-dimensional vector for position and radius, strength, falloff parameters as input, it applies impulses in every direction in that point.
In the application is used to simulate boxes and final boss explosion.

4.4 Jump

Jump uses the same mechanincs of wrecking balls. Every time spacebar is pressed an impulse is applied to dude hitbox that makes it jump.
Since i want the character to jump only once, the jump command is enabled only when dude is standing on a surface.

4.5 Player and boss movement

Dude is composed of a skeleton that is a list of bones that are manipulable (translate, rotate, scale). In order to make a smooth animation, i included translations and rotations of arms and legs in registerBeforeRender function.
Every time w,a,s,d are clicked, dude hitbox is translated in the relative axis (X for w and s, Z for a and d), body rotates and animation starts in loop.
If keys are all up dude returns with his bones to the initial position and rotation.

The final boss was created with the intent to completely satisfy the request of the hierarchical model and an animation that fully showed its structure.
The animation is triggered by player position and it starts after five second timeout(lights and music effect waiting).
The root model(body) is translated in z axis so all other components follow it.
Head rotate to y axis and wheel on x axys. Arms are composed of many parts end each of them does translations and rotations.
For two times, every 15 seconds translations and rotations increase in speed to give the impression of a energy overload.
After a total of 45 seconds, animation stops and final boss explode, with a radial explosion impulse as explained above.
To make a realistic explosion, before it happens, arms and head parent value is set to null so they can move away from the body.

4.6 Bullets

When final boss starts moving it fires bullets in x axis direction. Bullets are physic object(cones) that damage the player if they collide with him.
Every bullet is generated after a timeout that decrease with the energy overload (more fire rate in the end).
When it reaches a certain position the bullet is disposed.

Chapter 5

Optimization

During the development of the application i had to face the number one enemy of gameplay: lag.

5.1 Lights and shadows

The initial idea was a corridor with all lights enabled togheter and shadows of all meshes.

Figure 5.1: Example of 4 shadows

Babylon.js has a light limit of 4. To extend this value is needed to apply maxSimultaneusLight attribute to some material and setting it to a number bigger than 4 (for Platfrom was 20).
Receive shadows was applied to all meshes, but in the end fps in a medium/low device was 15.
I removed half of lights and enabled one a time and the only shadow caster in the application now is dude.
Despite this change, light was processed on too many objects simultaneously and still resulted in a decrease in performance. For example the appartments had hundreds of meshes and all of them were enlightened.
I resolved applying inside ImportMesh MergeMeshes function that takes the mesh list and merge all of them in one.

5.2 Mesh, Clone, Instance

Another reason of lag is the amount of meshes loaded in the scene. First I chose objects that weigh less and then it's possible to notice that some objects are repeated several times.
Instead of repeating imports and generating memory capacity problem, i used clones and instances.
An instance is an excellent way to use hardware accelerated rendering to draw a huge number of identical meshes.
Each instance has the same material as the root mesh. They can vary on the following properties: position, rotation, rotationQuaternion, setPivotMatrix, scaling.
Clone has the same characteristics of instance but it shares also the same geometry.
I used clones for meshes when physic is applied(boxes, cars, dogs) and instances when physic is not needed(trees, houses, street lamps).

5.3 Camera

In debugging mode i could notice that more objects camera is looking, less is the framerate.
In order to solve this problem, arcade camera mode is high positioned, looking the ground and few objects and i added maxZ attribute that define a distance of what you can see from the camera.

5.4 gl.clear and material update

By default, Babylon.js automatically clears the color, depth, and stencil buffers before rendering the scene. On systems with poor fill rates, these can add up quickly and have a significant impact on performance. Setting:
scene.autoClear = false;
scene.autoClearDepthAndStencil = false;
background color will be always white and not refreshed.
The game is inside a skybox so when the camera is in immersive mode nothing changes, but arcade mode camera has limitations so if player rotates it he can see a white background color.

By default the scene will keep all materials up to date when you change a property that could potentially impact them (alpha, texture update, etc...). To do so the scene needs to go through all materials and flag them as dirty. This could be a potential bottleneck if you have a lot of material. Setting scene.blockMaterialDirtyMechanism = true; it's possible to resolve the problem.

5.5 Scene Optimizer

The SceneOptimizer tool is designed to reach a specific framerate by gracefully degrading rendering quality at runtime.
In registrAfterRender if current fps are less than 60 it starts making an hardware scaling.
If the device used is not performing well enough, scene optimizer impose lower render quality in order to get 60 fps and have a smooth gameplay.
NOTE: Gui can get bigger when scene optimizer is activated.