GRAPHICS SYSTEMS AND INTERACTION

Lesson 4

Abstract

Project “Thumb Raiser”

João Paulo Pereira

jjp@isep.ipp.pt

Table of Contents

Table of Figures

Table of Equations

Project “Thumb Raiser”

The aim of this three.js [1] project is to guide a robot through a maze in order to find the exit (Figure 1 and Figure 2).

Figure 1 – Project “Thumb Raiser” (single-view mode)

Figure 2 – Project “Thumb Raiser” (multiple-views mode)

Download the folder “Thumb_Raiser_template”. The project is composed by several subfolders and files:

- “mazes/Loquitas.json”
- “models/gltf/RobotExpressive/RobotExpressive.glb”
- “textures/ground.jpg” and “textures/wall.jpg”
- “Thumb_Raiser.html”
- “orientation.js”
- “default_data.js”
- “merge.js”
- “thumb_raiser_template.js”
- “maze_template.js”
- “ground_template.js”
- “wall_template.js”
- “player_template.js”
- “lights_template.js”
- “fog_template.js”
- “camera_template.js”
- “animations_template.js”
- “user_interface_template.js”

File “Loquitas.json” is located in folder “mazes” and contains the maze’s ground and wall texture URLs, the maze’s size and map, the character’s initial position and orientation, and the exit location.

File “RobotExpressive.glb” is placed in folder “models/gltf/RobotExpressive” and contains the character’s model and animations.

Files “ground.jpg” and “wall.jpg” are in folder “textures” and contain the ground and wall texture images, respectively.

The instantiation of the game and the definition of an animation function are made in file “Thumb_Raiser.html”.

The class Thumb_Raiser is declared in file “thumb_raiser_template.js”. It includes the constructor, which creates a 2D scene (the viewports frames) and a camera, plus a 3D scene (the game itself) and ten cameras, a renderer, the maze (composed by a ground plane and walls), the player (visually represented by an animated robot), the lights (including shadows), the fog, the statistics, and callbacks to process several events. The class also includes the update method, which waits until the maze and player have been loaded. It then adds the maze, the player and the lights to the scene, creates the clock, the model animations, the graphical user interface (GUI) [2], sets the player’s initial position and direction, and starts the game. Once the game is running, the update method is responsible for regularly update the game state, detect collisions, check whether the game is over, render the scenes, etc.

The maze is asynchronously loaded and modeled in class Maze (file “maze_template.js”). Its ground and each of its walls are modeled in classes Ground and Wall, which are described in files “ground_template.js” and “wall_template.js”, respectively. Class Maze also includes methods to convert cell coordinates to cartesian coordinates and vice versa, compute distances to walls and verify if the exit has been found.

The class Player is defined in file “player_template.js” and asynchronously loads a Graphics Language Transmission Format (glTF) [3] [4] 3D model (the animated robot).

Lights and fog are defined in classes Lights and Fog (files “lights_template.js” and “fog_template.js”, respectively).

Class Camera, in file “camera_template.js”, creates two cameras (one that uses perspective projection [5] and another one using orthographic projection [6]) for each view. There are four views – fixed, first-person, third-person and top view – plus the mini-map. the class includes several methods to control parameters such as getting / setting viewport dimensions, setting / updating window size, camera’s target, orientation, distance to target and zoom factor.

Class Animations in file “animations_template.js” holds the character’s animations: states (idle, walking, running, dance, death, sitting and standing) and emotes (jump, yes, no, wave, punch and thumbs up).

Part of the user interface uses dat.GUI, a lightweight controller library for JavaScript [7]. It is defined in class UserInteraction, which is described in file “user_interface_template.js”. It gives the user control of the lights, shadows, fog, and character emotes and expressions (angry, surprised and sad). The remaining parts of the user interface, namely view selecting and camera control, and panels visibility, use HTML elements [8] [9] and are specified in files “Thumb_Raiser.html” and “thumb_raiser_template.js”.

Class Orientation extends three.js class Vector2 and is described in file “orientation.js”. It defines two additional properties: .h, which stands for “horizontal” and is an alias for property .x; and .v, which stands for “vertical” and is an alias for property .y. It is used to set cameras horizontal and vertical orientation angles.

Default data is defined in file “default_data.js”. It can be redefined in file “Thumb_Raiser.html” when instantiating the game.

The function merge is defined in file “merge.js”. It performs shallow / deep object merging and is responsible for merging default and post-defined data.

Your assignment is to write the code that handles some of the game’s functions, namely: model the maze, load and configure the ground and wall textures, import and animate the player, detect collisions, set the lights and shadows, create the fog, load the user interface, check if the player found the exit and trigger the final sequence.

Maze description

File “Loquitas.json” describes the maze. Its content is as follows:

```
{
    "groundTextureUrl": "./textures/ground.jpg",
    "wallTextureUrl": "./textures/wall.jpg",
    "size": { "width": 10, "height": 10 },
    "map": [
        [3, 3, 2, 2, 2, 2, 1, 2, 2, 3, 1],
        [1, 1, 3, 2, 1, 1, 2, 2, 1, 1, 1],
        [1, 1, 2, 1, 1, 2, 2, 1, 1, 0, 1],
        [1, 2, 3, 1, 2, 2, 1, 1, 2, 0, 1],
```

```
            [3, 0, 0, 3, 1, 2, 0, 3, 1, 2, 1],
            [1, 3, 0, 0, 0, 3, 1, 0, 1, 1, 1],
            [1, 2, 1, 3, 2, 0, 2, 2, 1, 1, 1],
            [3, 2, 0, 2, 1, 1, 3, 1, 2, 1, 1],
            [1, 3, 0, 1, 2, 2, 0, 1, 2, 0, 1],
            [1, 2, 2, 1, 1, 2, 3, 0, 0, 2, 1],
            [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0]
        ],
        "initialPosition": [7, 6],
        "initialDirection": 0.0,
        "exitLocation": [-0.5, 6]
    }
```

Where:

- `groundTextureURL` declares the URL of the texture image assigned to the ground
- `wallTextureURL` declares the URL of the texture image assigned to each wall
- `size` defines the maze's width (X-dimension) and height (Z-dimension) in number of cells
- `map` describes the contents of each cell
- `initialPosition` defines the character's initial position
- `initialOrientation` defines the character's initial orientation
- exitLocation identifies the maze exit

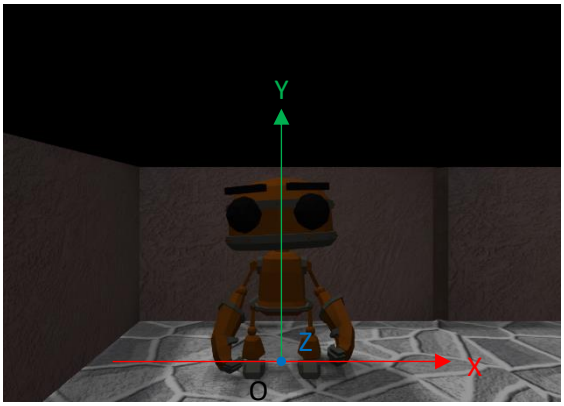
The meaning of the numbers in `map` is the following:

- 0: the cell has no west or north wall
- 1: the cell has a west wall but no north wall
- 2: the cell has a north wall but no west wall
- 3: the cell has a west and a north wall

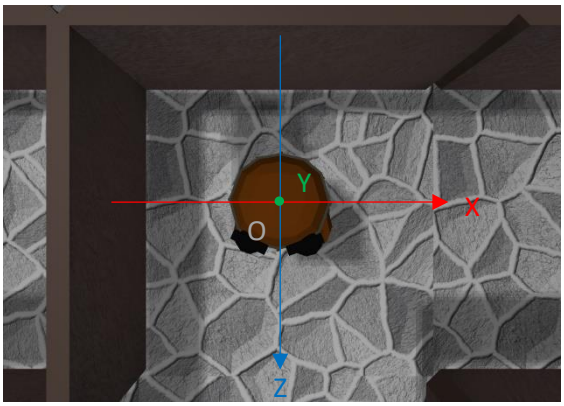
Note that although the maze size is 10 x 10, the map dimension is 11 x 11, in order to represent the maze's eastmost and southmost walls.

The coordinate system

The 3D Cartesian coordinate system is represented in Figure 3. The maze ground lies on plane OZX (plane *y* = 0.0) and is centered on the origin. The scene's up direction is parallel to the positive Y-semiaxis (0.0, 1.0, 0.0). The model's default orientation is parallel to the positive Z-semiaxis (0.0 degrees).



Front View



Top View

Figure 3 – The coordinate system

To-do #1 – Load the ground texture image

Open the file "ground_template.js" and look for comment "To-do #1". Follow the instructions.

To-do #2 – Set the texture wrapping modes

Look for comment "To-do #2" and follow the instructions.

Code example and class to consider: Texture [10].

To-do #3 – Configure the magnification and minification filters

Look for comment "To-do #3" and follow the instructions.

Class to consider: Texture [10].

To-do #4 – Assign the texture to the material's color map

Look for comment "To-do #4" and follow the instructions. You should now see a textured ground.

Class to consider: MeshPhongMaterial [11].

To-do #5 – Create the north walls

Open the file "maze_template.js" and look for comment "To-do #5". Follow the instructions. When placing the wall, assume the following:

- The maze is centered on the origin
- The size of each cell is 1 x 1 (X-dimension, Z-dimension)
- The newly created wall is centered on the origin
- The size of each wall is 1 x 1 (X-dimension, Y-dimension); the wall's Z-dimension is irrelevant

To-do #6 – Create the west walls

Look for comment "To-do #6" and follow the instructions. Keep in mind that unlike north walls, west walls must be reoriented.

To-do #7 – Create the top face of each wall

Open the file "wall_template.js" and look for comment "To-do #7". Follow the instructions. The top face of the walls is represented in Figure 4. Given that it is facing up, the normal vectors must be parallel to the Y-semiaxis (0.0, 1.0, 0.0).

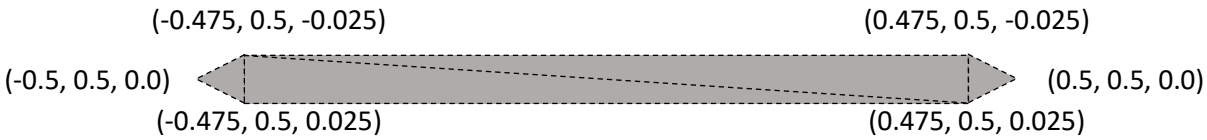


Figure 4 – Top face of the maze walls

To-do #8 – Load the wall texture image

Open the file "wall_template.js" and look for comment "To-do #8". Follow the instructions.

To-do #9 – Configure the magnification and minification filters

Look for comment "To-do #9" and follow the instructions.

Class to consider: Texture [10].

To-do #10 – Assign the texture to the material's color map

Look for comment "To-do #10" and follow the instructions. You should now see a textured ground.

Class to consider: MeshPhongMaterial [11].

To-do #11 – Add the player to the scene

Open the file "thumb_raiser_template.js" and look for comment "To-do #11". Follow the instructions.

To-do #12 – Compute the distance covered by the player (assuming that the player is walking)

Look for comment "To-do #12" and follow the instructions.

To-do #13 – Compute the player's direction increment (assuming that the player is turning left or right while walking)

Look for comment "To-do #13" and follow the instructions.

To-do #14 – Adjust the distance covered by the player (assuming that the player is running)

Look for comment "To-do #14" and follow the instructions.

To-do #15 – Adjust the player's direction increment (assuming that the player is turning left or right while running)

Look for comment "To-do #15" and follow the instructions.

To-do #16 – Check if the player is turning left or right and update the player direction accordingly

Look for comment "To-do #16" and follow the instructions.

To-do #17 – Check if the player is moving backward or forward and update the player position accordingly

Use the parametric form of the circle equation to compute the player's new position [12]. Given that the player is moving on the OZX plane (the ground), and that the player's default orientation is parallel to the positive Z-semiaxis (0.0 degrees), the circle equation is slightly different from the one used in projects "Watch" and "Pong" (Figure 5 and Equation 1).

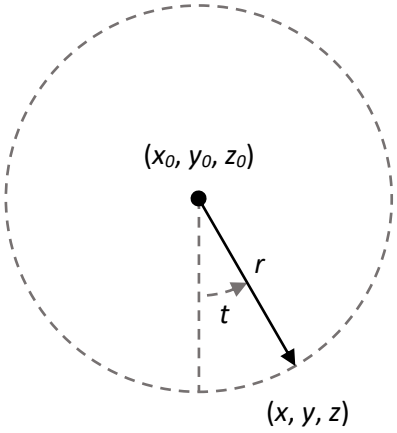


Figure 5 – Updating the player position

$$\begin{cases} x = r * \sin(t) + x_0 \\ y = y_0 \\ z = r * \cos(t) + z_0 \end{cases}$$

Equation 1 – Parametric form of the circle equation (OZX plane)

Where:

- (x, y, z) are the player's new coordinates
- (x_0, y_0, z_0) are the player's current coordinates
- r is the distance covered by the player
- t is the player direction (expressed in radians)

Look for comment "To-do #17" and follow the instructions.

To-dos #18 and #19 – If the player collided with a wall, then trigger the death action; else, trigger either the walking or the running action

Look for comments "To-do #18?" and "To-do #19" and follow the instructions (for the moment no collision will be detected).

To-do #20 – Check the player emotes (jump, yes, no, wave, punch and thumbs up)

Look for comment "To-do #20" and follow the instructions.

To-do #21 – If the player is not moving nor emoting, then trigger the idle action

Look for comment "To-do #21" and follow the instructions.

To-do #22 – Set the player's new position and orientation

Look for comment "To-do #22" and follow the instructions.

To-do #23 – Measure the player's distance to the walls

Open the file "maze_template.js" and look for comment "To-do #23". Follow the instructions.

To-do #24 – Check if the player collided with a wall

Assume that a collision is detected if the distance between the player position and any of the walls is less than the player radius.

Open the file "thumb_raiser_template.js" and look for comment "To-do #24". Follow the instructions.

To-do #25 – Set the default camera projection

Open the file "camera_template.js" and look for comment "To-do #25". Follow the instructions.

To-dos #26 and #27 – Create both point lights and set their position in the scene

Open the file "lights_template.js" and look for comments "To-do #26" and "To-do #27". Follow the instructions.

Class to consider: PointLight [13].

To-dos #28 and #29 – Add both point lights to the scene

Look for comments "To-do #28" and "To-do #29". Follow the instructions.

To-do #30 – Turn on shadows in the renderer and filter shadow maps using the Percentage-Closer Filtering (PCF) algorithm [14]

Open the file "thumb_raiser_template.js" and look for comment "To-do #30". Follow the instructions.

Code example to consider: PointLightShadow [15].

To-dos #31 and #32 – Turn on shadows for both point lights and set their properties

Open the file "lights_template.js" and look for comments "To-do #31" and "To-do #32". Follow the instructions.

Code example to consider: PointLightShadow [15].

To-do #33 – Set the ground plane to receive shadows but not cast them

Open the file "ground_template.js" and look for comment "To-do #33". Follow the instructions.

Code example to consider: PointLightShadow [15].

To-dos #34 to #36 – Set the walls to cast and receive shadows

Open the file "wall_template.js" and look for comments "To-do #34" to "To-do #36". Follow the instructions.

Code example to consider: PointLightShadow [15].

To-do #37 – Set the character to cast shadows but not receive them

Open the file "player_template.js" and look for comment "To-do #37". Follow the instructions.

Code example to consider: PointLightShadow [15].

To-do #38 – Create the fog

Open the file "fog_template.js" and look for comment "To-do #38". Follow the instructions.

Class to consider: Fog [16].

To-do #39 – If the fog is enabled, then assign it to the scene; else, assign null

Open the file "thumb_raiser_template.js" and look for comment "To-do #39". Follow the instructions.

Class to consider: Fog [16].

To-do #40 – Create the user interface

Look for comment "To-do #40" and follow the instructions.

To-do #41 – Toggle the user interface visibility

Look for comment "To-do #41" and follow the instructions.

To-do #42 – Check if the player found the exit

Assume that the exit is found if the distance between the player position and the exit location is less than (0.5 * maze scale) in both the X- and Z-dimensions.

Open the file "maze_template.js" and look for comment "To-do #42". Follow the instructions.

To-do #43 – Trigger the final sequence

1. Disable the fog
2. Reconfigure the third-person view camera
3. Set it as the active view camera
4. Set single-view mode
5. Set the final action

Open the file "thumb_raiser_template.js" and look for comment "To-do #43". Follow the instructions.

References

[1] Three.js, "Three.js – JavaScript 3D Libray," [Online]. Available: https://threejs.org. [Accessed 25 July 2021].

[2] Wikipedia, "Graphical user interface," [Online]. Available: https://en.wikipedia.org/wiki/Graphical_user_interface. [Accessed 20 October 2021].

[3] Wikipedia, "glTF," [Online]. Available: https://en.wikipedia.org/wiki/GlTF. [Accessed 23 October 2021].

[4] Khronos Group, "glTF Overview," [Online]. Available: https://www.khronos.org/gltf. [Accessed 23 October 2021].

[5] Wikipedia, "Perspective (graphical)," [Online]. Available: https://en.wikipedia.org/wiki/Perspective_(graphical). [Accessed 23 October 2021].

[6] Wikipedia, "Orthographic projection," [Online]. Available: https://en.wikipedia.org/wiki/Orthographic_projection. [Accessed 23 October 2021].

[7] Google Open Source, "dat.GUI," [Online]. Available: https://opensource.google/projects/datgui. [Accessed 23 October 2021].

[8] Mozilla, "HTML elements reference," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML/Element. [Accessed 23 October 2021].

[9] W3Schools, "HTML Element Reference," [Online]. Available: https://www.w3schools.com/tags/default.asp. [Accessed 23 October 2021].

[10] Three.js, "Texture," [Online]. Available: https://threejs.org/docs/api/en/textures/Texture.html. [Accessed 27 October 2021].

[11] Three.js, "MeshPhongMaterial," [Online]. Available: https://threejs.org/docs/api/en/materials/MeshPhongMaterial.html. [Accessed 27 October 2021].

[12] Wikipedia, "Circle," [Online]. Available: https://en.wikipedia.org/wiki/Circle. [Accessed 25 July 2021].

[13] Three.js, "PointLight," [Online]. Available: https://threejs.org/docs/api/en/lights/PointLight.html. [Accessed 30 October 2021].

[14] NVIDIA, "Chapter 11. Shadow Map Antialiasing," [Online]. Available: https://developer.nvidia.com/gpugems/gpugems/part-ii-lighting-and-shadows/chapter-11-shadow-map-antialiasing. [Accessed 27 October 2021].

[15] Three.js, "PointLightShadow," [Online]. Available: https://threejs.org/docs/api/en/lights/shadows/PointLightShadow.html. [Accessed 27 October 2021].

[16] Three.js, "Fog," [Online]. Available: https://threejs.org/docs/api/en/scenes/Fog.html. [Accessed 31 October 2021].