Interactive Graphics

A.Y. 2017/2018

Project Report

Train in a village

DIAG

Sapienza

July 2018

Lorenzo Germano

Matricola: 1771129

Abstract

For the realization of the Interactive Graphics Project i chose to develop a simple village made up of houses and trees crossed by a train created as a hierarchical model.
The main aspect on which i focused was the realization of the game of light and shadows generated by the lights of the train on the surrounding environment and a crystal of fire in the center of the village that simulates the light generated by a simple real fire.
All the implementation of the environment has been realized through the use of the ThreeJS library and the formatting of the commands in the browser window using CSS.

Contents

Part 1

Tools and Libraries

1.1 ThreeJS

ThreeJS is a JavaScript library used to create animated objects both in 2D and 3D for computer graphics in a web browser. It is based on WebGL, a lower-level library that allows computational calculation through the use of the GPU for the creation of animations and the management of graphics. In this way the author can use the browser without adding external applications or plug-ins. An important feature of the ThreeJS library is the use of object-oriented programming that allows the user, in a simpler way, to create complex scenarios. The use of the library is available by importing into the HTML file a single library in JavaScript format called "*Three.js*". Thanks to it, the user can access most of the main functions offered such as: Scene management and camera view, objects, geometric transformations, lights and shadows. In the package offered by ThreeJS are then present other files, always in JavaScript format, which allows the use of other features related to advanced management of external models, features related to the management of the camera, etc.

1.2 CSS

CSS stands for Cascading Style Sheets and is a language used to define the formatting of HTML documents on web sites and related web pages. Thanks to, it is possible to intervene on the format of the text, on the positioning of the graphic elements and on the arrangement that these elements will have with respect to different devices or browsers. CSS instructs a browser or other user program on how the document should be presented to the user, for example by defining its fonts, colors, background images, layout, placement of columns or other elements on the page, etc.

The use of CSS has been chosen for a comfortable and simple management of the interface with the user in the project, in order to obtain a better graphic rendering and the use of simple interactions with the commands.

1.3 Blender

Blender is free software and multiplatform modeling, rigging, animation, compositing and rendering of three-dimensional images. It also has features for UV mapping, fluid simulations, coatings, particle, other non-linear simulations and creation of 3D applications/games. The potentiality of the software is given by its characteristics that allow it to support a wide variety of geometric primitives, the conversion from and to numerous formats for 3D applications, animation management, interactive features and a versatile internal rendering engine.
The use for the project carried out with Blender was to create models to be imported into the scenario or the use of free models available and obtainable on the web created by the community of developers in the open source project. Thanks to Blender, scenario models have been created and modified, materials managed and finally converted into *.json* files for import, through the *TreeJs ObjectLoader* library and used in the project.

Part 2

Technical aspect of the Project

2.1 Skybox

The first step in the realization of the project was to create an external scenario that would host the main theme composed by the village. For the realization of the scenario I chose to implement a Skybox. Technically it can be defined as a mapping of a very large cube. In my project, it consists of six cubes placed as faces of a larger cube and inside the cube is placed the starting frame of the camera view. An image has been applied to each cube, which is the portion of a larger one composed of them. Each cube has as a texture a portion of the complete image and has been inserted in such a way that once the edges have been connected, a 360-degree view will result from inside the cube and from the outside. In this way the result is a very wide scenario that completely surrounds the village giving the idea of an infinite world.
The chosen images represent a night sky so that the realization of the village and the environment were nocturnal to make better the vision of the lights and shadows used.
To each cube the parameters of refraction and creation of the shadows emitted by the objects hit by the ray of the lights present in the village were not managed, so as to render the effect of a scenario tending towards infinity and not near the center of the scene composed by the village.

2.2 Camera View

The scenario that I wanted to build consists of a village inside a skybox and I opted for the use of a camera view that allowed the user to go around the village and to fully view every aspect and detail of the scenario.
For the realization of this type of view I imported and used the library offered in the ThreeJS package called *OrbitControls.js*
The library has been loaded externally from the main one, namely *Three.js* and included in the HTML file.

Orbit controls allow the camera to orbit around to target. Once the camera view variable was created using the library *THREE.PerspectiveCamera()* and initialized the parameters relative to the initial position and the reference with the size according to the browser window, it was possible to use a single function offered by *OrbitControls.js*. In this way the controls of the camera view were managed by the mouse by the user who can freely move around in the scenario. The controls constantly update the camera view so recalculate the rendering process from the current position.
To follow the movements of the hierarchical model of the train, a direct view on it was also implemented. The view, managed through the interaction with a button, was positioned above the train in such a way as to visualize its position and follow it during the animation. Also in this case, once the view is positioned, the user can move freely, having the train as a visual objective. Through the use of the button presented in the GUI you can then return to a free view and vice versa

2.3 Models and hierarchical model

A fundamental part of the project were the models that were imported and created, constituting the scenario presented. In the scenario there are some models that make up the village like: Trees, houses, rocks and fire. These models were created with Blender and imported into the scenario. With the use of Blender I could import models found on the web, modify them and make them usable in the project. In particular, to use the models and manage them in the scenario, we used the library called *ObjectLoader.js* in the ThreeJS package. As with the *OrbitControls.js* external library, the ObjectLoader.js library has also been declared externally in the HTML file. For each model present, a *.json* file has been exported containing all the information concerning the geometry structure and the materials that make it up. In the template management JavaScript, the geometry and materials contained in the Json file were loaded for each individual model representing an object to be included in the scenario. At this point the position, the creation and refraction of the shadows was managed and for the objects present in greater number as the trees were managed the variables related to the position on the x and z axis in a random way through an external function that allowed to position them in random interval in the scenario. The same method was used for the creation and placement of some crystal like rocks in the scenario.

The model related to the fire is made through the model of a crystal whose materials have been modified with Blender, to make it similar to a red crystal that emits a light similar to fire.
For the houses that represent the village the same scheme used for the other objects was used, but being objects present in finite number in the scenario their positioning was decided in such a way as to compose a small village with the fire crystal in the middle.
On the contrary, the main object of the scenario was created differently: the train. It is a hierarchical model composed of a group of basic geometries created thanks to the use of the *Three.js* library. The train consists of objects that use geometries such as cylinders, cubes and torus. For each individual geometry that makes up the train, the geometry of the object and the material was first initialized and finally the final object created using the *THREE.Mesh()* function. For each one of them, dimensions and positions have been decided to make up the outline of a simple train. In the hierarchical model of the train the lights were also inserted and a pointer object used to direct the lights of the train, this to facilitate the creation of animation.
Each part of the train was made part of the train object by adding it to the *THREE.Group()* function.
Finally, the rails on which the train rests are two simple Torus geometries.

2.4 Lights and shadows

One of the most interesting aspects of the project is the use of lights and shadows not for the complexity of the code used but for the graphic rendering and impact in the scenario. Regarding the lights used to recreate a setting, I chose the use of ambient lights, pointlight and spotlight.
The ambient light has been used to have a minimum visibility over the whole scenario and to allow in particular the visibility on the Skybox being the only part of the scenario that is not influenced by other lights and shadows. The ambient light was recalled from the *Three.js* library using the *THREE.AmbientLight()* function, managing color and intensity.
Being the ambient light very low to make the effect of a night scenery, I inserted a light of type PointLight at the height of the Moon present in the scenario on the Skybox. Managing an intensity even here minimal and a cold color i could recreate what is the effect of night lighting rendered by the light of the Moon. The PointLight has a 360 degree emission from the point of origin and for this reason it has also been used to make the light effect

rendered by the fire. To recreate the fire I managed the intensity through a function that would allow to have both positive and negative values so as to make visible also the impact of a cold and dark light on the surrounding objects. The values that manage the intensity of the light of the fire vary so the color from a base, orange similar to fire, to a dark blue one. The use of PointLight was possible thanks to the use of the *THREE.PointLight()* function in the basic *Three.js* library. In addition to the management of color and light intensity, the parameters relating to the emission distance of the light produced were managed, used to differentiate the type of a small fire in the center of the village from the stronger one of the Moon and the decay that manages set to 2 for physically correct lighting as recommended by the three.js documentation.
The last type of lights used were the SpotLight. Available in the three.js base library and usable by the *THREE.SpotLight()* function, they were used to build the train lights. They have been positioned in correspondence with the geometry of the headlights on the train. The color of this light is similar to that of a warm lamp of the trains and the intensity has been managed in order to make the effect on the surrounding environment more probable. In particular, a 3D object was managed to create a target object to which the lights are pointed in such a way as to have the lights in the same direction as the movement of the train.
Finally, the shadows in the scenario were managed. To make it possible to create shadows and overlay the shadows on other objects, i have enabled the mapping of shadows using the *renderer.shadowMap.enable()* function. At this point every object is both important as a model and created as a hierarchical model has been composed of two other functions that allow the creation of the shadow of the model and to receive the shadows of other models. This was possible through the use of two functions: *castShadow()* for creating the shadow of the model and *receiveShadow()* to receive the shadows of other models on the object. The last step for creating shadows was to manage the parameters of each light for creating shadows. As for the models and geometries, each light has enabled the *castShadow()* function. Moreover the properties of the shadows were managed for the light that generated them through the following functions:
light.shadow.mapSize.width = 512; // default
light.shadow.mapSize.height = 512; // default
light.shadow.camera.near = 0.1; // default
light.shadow.camera.far = 500; // default

2.5 Animation

The animations present in the project mainly concern the movement of the train, the variation in intensity of the light of the fire and the shadows produced by the emitted lights.
The animations were made possible by creating time interval management variables that constantly update the status of the scenario. The function used is *setInterval()* which consists of the function to be updated and the update time interval.
With regard to the movement of the train, the rotation of the hierarchical model around the center of the village was managed in order to make the movement effect. The train lights move dynamically with the model and the shadows of the objects that are hit by the light vary depending on the position of the lights.
The intensity of the fire is instead managed by a function that randomly varies the intensity parameter over time so as to give the probable effect of the light produced by the flame of the fire, in this case the fire crystal. As with objects hit by the light of the train, even the objects that are hit by the light of the fire undergo variations on the shadows they generate and the shadows that are projected onto them.
To make the movement of the train more realistic, other animations were implemented separately. The wheels of the train move independently of the rest of the model simulating the vertical movement caused by the changes in the rails. In the same way the rest of the model, independently, has vertical oscillations caused by the wheel jumps.
Finally, the animation of the chimney of the train engine has been implemented, which jumps vertically to make the idea of a division from the model of the train caused by a crest. The chimney after the jump, gradually returns to the original position.

2.6 GUI

The interface with the user consists of two main aspects: The view, which can be managed by mouse input and the buttons at the top of the screen that allow interaction with the train animation, the power on or switching off of

the train lights and viewing the information on the scenario and the commands available.
The buttons at the top of the screen were managed by CSS who manages the graphics and animation by creating a file that handles the style called *style.css*. A simple animation has been implemented that allows the button, once the user places the cursor on the button to be used, to gradually change the color and vice versa to return to the original color when the cursor is no longer positioned on the button.

2.7 Other technical aspect

Some functions were necessary and fundamental for the realization of the project. The display of the text is managed using a button thanks to the *showText()* function which, through the input received from the button present in the graphic interface, inserts or removes the text that will provide the user with information about the project.
Another important function for the management of the position of the models and reused also for the management of the intensity of the light of the fire was *getRandomIntInclusive(min,max)* which takes two values entered manually by the programmer and returns an integer number included between them extremes included. Thanks to this function it was possible to manage the position of the trees within the scenario randomly and with a precise choice of the parameters to avoid overlapping with the houses present in the village. Moreover this function allows to take both negative and positive values that can be used to manage the intensity of the light of the fire.
During the realization phase of the project some functions were used that allowed to have graphic references on the effects of the changes on the elements within the scenario. In particular, the helper libraries within *Three.js* were used to manage the axes by means of the *THREE.AxesHelper()* function, the display of a help grid using the *THREE.GridHelper()* function and with regard to the lights the use of the *THREE.CameraHelper()* function to display the direction of the train lights and adjust them accordingly.
A last useful function used to manage the display window of the project depending on the browser window and its size was *window.addEventListener()* that through the implementation of a function, dynamically based on the size of the browser, performs the resize of the entire screen keeping the ratios of size.