Report

Nicolò Pinciroli

February 12, 2020

Introduction

This project consisted in modeling a city by creating a city simulator. The goal of such a simulator should be the support of the decision-making process in urban development related tasks.

Since no data have been provided, the city has been generated by scratch, and some randomness has been introduced in order to create a slightly different setting every time the simulator is run.

The proposed web application is based on the WebGL technology and on the use of JavaScript, CSS and HTML. It hasn't been necessary to use a server for this purpose, since all the results can be computed on the user machine. This approach presents some advantages (for instance, there is no need to have a server to run the simulator) and some disadvantages (since the computation is performed by the user machine, performance strongly depend on the user machine and the simulation can take some seconds before something can be visualized).

Tools and libraries

It is possible to use pure WebGL, without any additional library, to build the simulator. This approach certainly allows to go in depth in the structure of WebGL and to understand thoroughly the mathematical foundations of computer graphics, but it also presents some disadvantages. Firstly, creating tridimensional objects in WebGL requires to model them starting by creating points and triangles. Then, in order to create several objects with similar features, it can be a good idea to create a library containing some commonly-used functions. In this case, an example can be a function to create a cube, or a pyramid. The implementation of such libraries can require a huge amount of time, especially considering that the topics related to tridimensional modeling in WebGL will be covered after the submission deadline. There exist several libraries that allow to handle tridimensional scenes in WebGL with ease. Such libraries have been developed during several years by many developers, and I expect them to rely on well-established good practices and on an optimized code. Building such libraries in one month would probably not lead to the same result. Two disadvantages of using libraries, however, are that this assignment hasn't allowed

me to learn the foundations of WebGL and that even if I need only for a few functionalities, I have to import the entire library.

The proposed solution, therefore, consists in using three.js, whose code is released under the MIT license and is used by thousands of developers, as shown on its GitHub page[1].

Primitives

Three.js certainly makes use of graphics primitives, and in particular it is based on triangles. However, this level of detail is hidden, therefore a programmer can create tridimensional objects directly, without having to define them using simpler objects.

I have also used, for debug purposes, the line primitive, that allows to model rays appropriately.

As for the other shapes, I have modeled buildings by using a box geometry, since they have the shape of a parallelepiped. On some buildings (the ones that represent the old buildings), there is also a roof, that has been modeled by using a cone geometry. In fact it appears as a pyramid.

Moreover, I have used planes to define the ground and the park (I have only two planes, but another possible implementation may have been to have several small planes and to assign to each plane a texture).

Finally, I have modeled a semi-transparent sculpture by using a torus knot geometry.

Structure of a scene

The scene has been created in a procedural way. Moreover, I have introduced some randomness related to the position and orientation of buildings, in order to create a slightly different scene every time.

The ground of the scene is represented by a plane, on which I have applied a texture to make it easily identifiable. The plane is at the same level as the bottom of the buildings. In a future development, it could be replaced by a terrain or by water (the purpose could be, for instance, to model a partially submerged city as a consequence of climate change).

The park has also been modeled as a plane, that has been put on top of the ground plane. I have decided to position the park in the center of the city.

My idea was to divide the city in two parts: a modern part, characterized by modern concrete buildings, and an old part, characterized by stone buildings with roofs. This is the reason why there are two different kinds of buildings on the two different sides of the park. This can be useful to model a city in which there is an historical center and a modern part, built afterwards, since the houses have different properties, for instance from a seismic point of view.

The entire city is surrounded by a box representing the sky. The idea was to create an environmental map, in order to obtain a parallax effect, that should provide more realism.

[1] https://github.com/mrdoob/three.js/

The sun light has been modeled as a directional light and it rotates following a circular path. In order to improve the performance of the simulator, the sun can only move in ten different positions, that range from an angle of 0 degrees to an angle of 180 degrees (from dawn to sunset). Night hasn't been modeled, since I considered it outside the scope of this simulator.

Graphics pipeline

The graphics pipeline hasn't been implemented explicitly, since it is part of the three.js library.

Shaders and lighting models

Shaders haven't been implemented explicitly, since they have been defined in three.js. However, I had to define different materials in order to obtain different effects when the object surfaces interact with light. It is possible to easily add some objects that reflect the sun light but I haven't created them in order to keep the scene as clean as possible.

Cameras and projections

There is one camera, that allows the user to explore the city. The current view has been implemented by using the orbit controller, that can be easily integrated with the main functionalities provided by three.js. Moreover, since many objects cast shadows, it is necessary to have an orthographic camera to compute the shadow position, since the source of light responsible of the shadows is a directional light (the sun). However, this has been defined inside three.js.

Lights

I have defined the sun light as a directional light and an ambient light. There aren't other lights, but three.js allows to insert them easily. Moreover, the position of the sun can change, therefore the direction of the light changes as well.

Raster and vector graphics, rasterization, textures and color

I have defined a color in the case of the knot sculpture, while in many situations I have decided to define textures. In particular, I have looked for seamless textures, in order to obtain a more realistic and appealing result. The old buildings have a random texture, selected among several predefined textures. As for the other objects, there is only one texture per object. The textures I have used are bidimensional textures and they are not generated procedurally. I don't have to perform any operation related to UV mapping because such operations are defined by three.js. Rasterization has not been defined explicitly, but it is performed every time the scene is generated (that means, every time there is a change in the position and/or orientation of the camera).