COMPUTER GRAPHICS 1: Assignment #2

Due on 22/09/2020

IE500217

Nicolò Pinciroli

In this assignment, I have realized a smart city simulator using Unity3D. This simulator is based on a randomly generated city. The reason why I have chosen this solution, rather than building a city manually is that in this way it is easier to expand the city, adding building and changing their shapes, also considering possible future developments of the project.
For this reason, the initial scene is completely empty, with the exception of a camera. The scripts, then, populate the scene and allow the user to interact with it using the keyboard keys, as I will describe briefly. This simulator supports the visualization of spatiotemporal data, since it allows to easily visualize spatial information (the buildings, which are tridimensional, and are seen from a camera), and also the change of some elements of the scene over time. In particular, in this version of the simulator, it is possible to move the sun (using the keyboard keys A and D), and this movement causes a movement in the shadows. It would be easy to have an automatic simulation, by just pressing the keys until the sun has completed a round around the scene. This process is not automatic to allow the user to decide when the sun can be moved. However, the simulation of the shadow variation is performed, both when the user moves the sun using the keyboard and when the calculation of the shadows is performed.
The buildings, as previously mentioned, have been generated randomly. They are all cubes which have been dilated. It would also be possible to create other kind of shapes (for example cylinders), but those are not as common as the ones I have inserted in a typical city. However, this would not affect the shadow calculation or other calculations in any way. The park has been modelled as a rectangle, and a grass texture has been selected, always using a script. The texture has been chosen for two reasons: make the park more realistic and recognizable and show the presence of the shadows, which could not be displayed in absence of a texture. The same applies for the city pavement. However, I plan to improve the scene by adding other elements and more realistic textures in the future. A drawback of having realistic and detailed textures would be the complexity of the scene and the number of calculations that should be performed in order to properly generate it, which would make the user experience not very effective, since on older devices the simulator would be extremely slow and unresponsive.
The buildings don't have a texture. The reason why I have chosen to avoid a texture on them is that in this way it is easier to visualize colors related to the requested computations, among which it is very important to consider the landmark visibility. The landmark is created as the other buildings, but differently from them it has a different color and a different name. The buildings, on the other hand, all have a name starting by "building" and followed by their coordinates in the scene, in such a way that they are uniquely identifiable.
The landmark visibility is calculated as presented on the slides attached to this assignment. From the landmark there are several rays, which depart from the landmark levels and that go around the building radially. If the ray cast from the landmark encounters another object,

that object becomes red.
I have verified this algorithm both in a scene without obstacles other than the buildings and in a scene which also had a fence that separated some buildings from the rest of the scene. The latter buildings were not colored in red, which is a confirm that the algorithm has worked correctly in that situation. This has been very useful in order to understand how to cast the rays properly and for debug purposes. However, such a fence does not exist in the scene.
As also commented in the code, the view from the landmark is assumed to be planar, which means that the buildings below the current level are not considered. However, it would be easy to include this computation as well, by just adding other for loops to iterate through the remaining angles.
One of the most challenging parts of the project has been the calculation of the shadow map and the Change Frequency Heatmap. However, it has been easier than the implementation in JavaScript and THREE.js, since Unity3D includes many high-level functionalities, which result very handy in the scripting phase. In this way, it is much easier to perform calculations using vectors, positions and angles.
As for the computation of the shadow heat map, the strategy I have followed is the following:

- Divide the park area in smaller areas, which right now have the dimension of 1 x 1 units (assuming those units are meters)
- From each position of the sun, and from each small area in the park, cast a ray that has a direction directed towards the sun, which is a directional light. It is very important to avoid having those rays directed towards the center of the sun, since it is assumed to be at an infinite distance, even if it is not for practical reasons.
- For each ray, the intersections are checked up to a certain value (which is 1000 units in the code). If no intersections are found, this means that there is no shadow, otherwise there is a shadow.
- There exist a map (from a data structure perspective), which has positions as keys and number of rays which have intersected a building as a value. When a point is in shadow for the first time, a new entry is created in the map, with a value of one, otherwise the existing one is updated by adding 1.
- When this loop ends, the map contains the points that have been in shadow at least once and the number of times they have been in shadow. Of course, the maximum number is the same as the number of movements of the sun.
- Later, the map is converted into a bidimensional matrix of integers, which covers the park area. This matrix has a value of zero where there is no shadow during the day,

and a value coming from the map otherwise.

- The matrix is then parsed into JSON and a webpage is generated to show the heatmap. Such webpage uses Plotly and JS, and the function which generates it can be used to display data from any bidimensional matrix of integers. Please note that Plotly automatically normalizes the colors according to the minimum and maximum values of the color themselves.

As for the CFH, the approach is quite similar once the matrix is obtained, indeed the function being used for the webpage generation is exactly the same. However, calculating the matrix is more challenging in this case. First, the sequence that has to be checked is inserted in the code. Right now it is hardcoded, but it might be taken as an input from the user in the future. The sequence is basically a sequence of 0's and 1's, which encode, respectively, the absence of shadow and the presence of shadow. In this case, for each sun movement and point in the park area, the algorithm computes if the current status (in shadow or not) is compatible with the sequence read so far. In case it is compatible (this is calculated using regular expressions), the current sequence associated to a point is kept, otherwise it is discarded. The number of times in which the input sequence and the observed behaviour is registered is then stored in a map, whose structure is the same as the one for the shadows computations. In this way, the next steps are the same as in the shadow computation case. It is possible to note that if the sequence is "1" the CFH is the same as the shadow map, and if the sequence is "0" it is the complementary map. However, those cases are not sufficient to guarantee the complete accuracy of the algorithm, per se.
Here I list the keys that can be used to interact with the scene:

- C to compute the shadow map and the CFH (those computations use, in part, the same loops, so it is convenient to calculate them together)
- V to compute the landmark visibility
- A and D to move the Sun around the scene
- R and T to rotate the main camera
- X to stabilize the main camera
- The four arrows to move the camera using translations

The following images represent some examples of output.

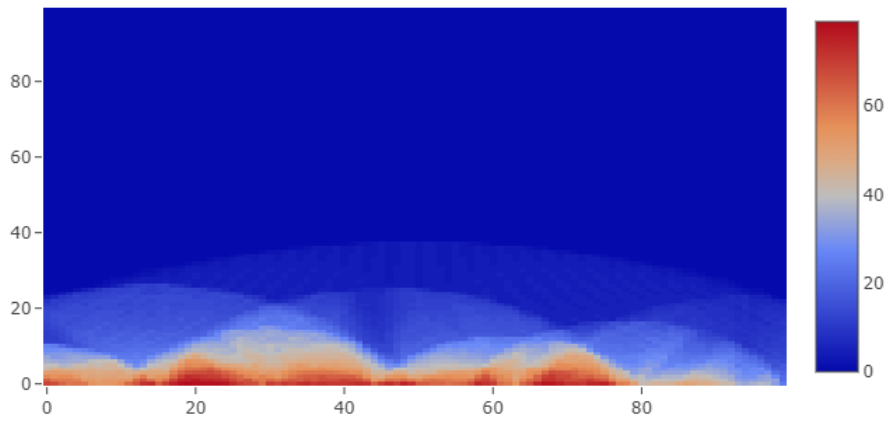

Figure 1: Shadow heat map

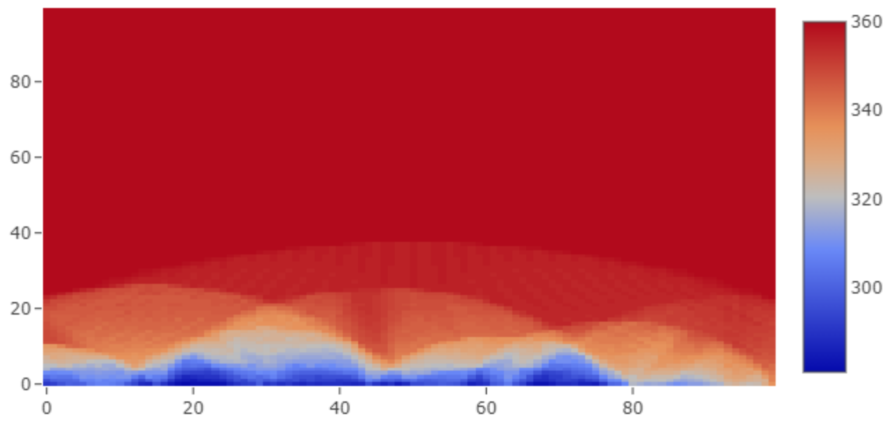

Figure 2: CFH for the sequence ”0”

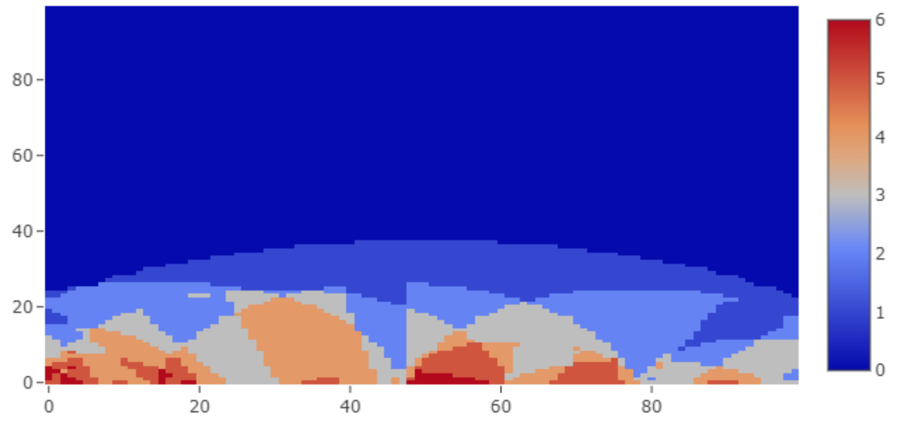

Figure 3: CFH for the sequence ”10”

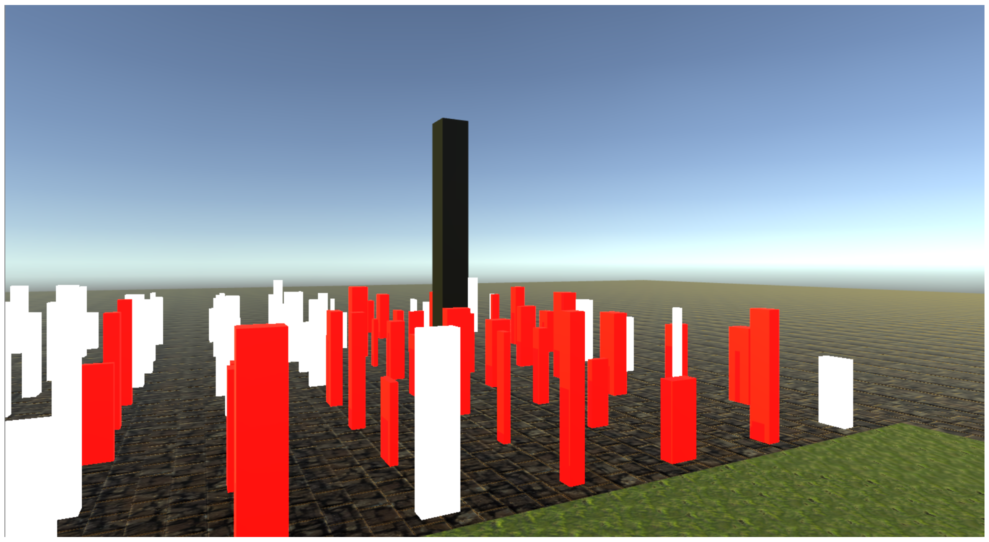

Figure 4: Landmark visibility