3D Representation of a topographic surface[1]

Donato D'Ambrosio
Department of Mathematics and Computer Science
University of Calabria

November 15, 2018

[1]Course of Computer Graphics and GPGPU Programming, Department of Mathematics and Computer Science, University of Calabria, Italy, Planet Earth

Assignment

Build an application displaying a topographic surface based on the following dataset (file `DEM_test_small.dat` into the data folder):

```
ncols         4
nrows         3
xllcorner     0.0
yllcorner     0.0
cellsize      0.1
NODATA_value  -9999
0.9 0.8 0.7 0.6
0.8 0.7 0.6 0.5
0.7 0.6 0.5 0.4
```

As you can see, data is composed by a header (first 6 rows) and a matrix of values representing topographic altitudes. Specifically, the lowest altitude is `z_min=0.4`, while the highest one is `z_max=0.9`.

The header defines the number of rows and columns of the actual data (`ncols` and `nrows`), the coordinates of the lower left corner (`xllcorner` and `yllcorner`), the distance between two adjacent points into the matrix along the horizontal and vertical directions (`cellsize`) and a special value representing non-sampled points (`NODATA_value`), this latter not considered in this example.

The matrix of values that follows represent a regular grid of altitudes.

What you have to do is to give a representation of the above dataset by means of triangles.

For this purpose, you can define a buffer containing the sequence of vertex positions and colors, to be sent to the GPU by means of a vertex buffer object (`VBO`), and a buffer of indices representing how the vertices have to be connected, to be sent to the GPU by means of a element buffer object (`EBO`). If you arrange the vertices starting from the lower-left corner up to the upper-right one, by evaluating the color as a linear function that maps the $[z_{min}, z_{max}]$ range into $[0, 1]$, you sound obtain a buffer like one shown below

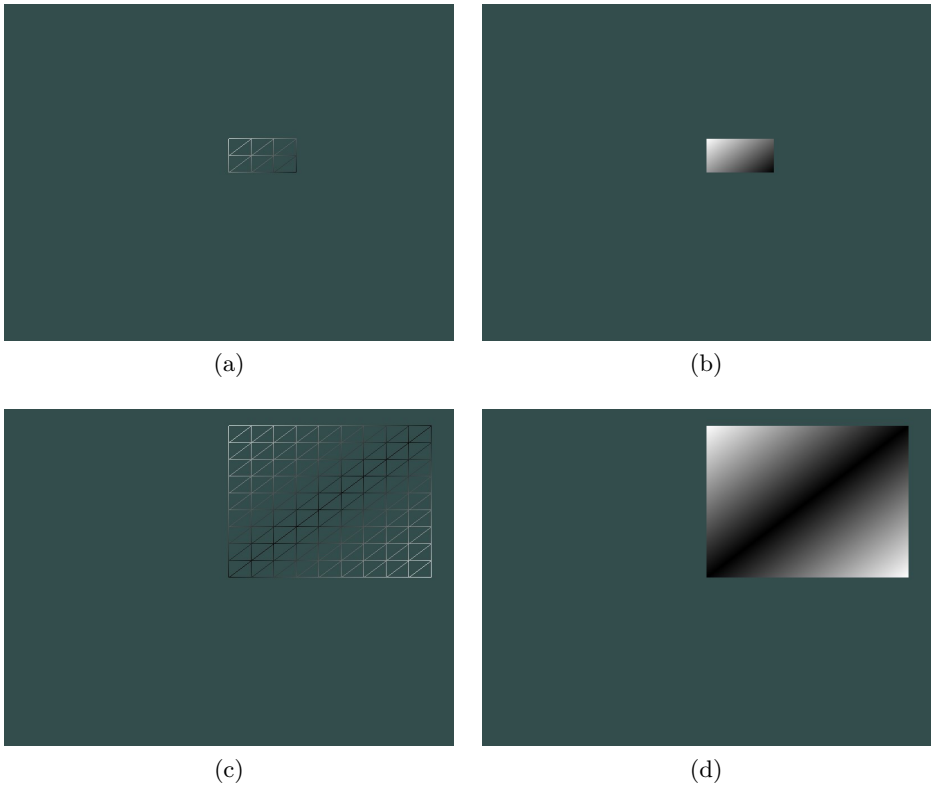
```
Positions    Colors
0.0 0.0 0.7 0.60 0.60 0.60
0.1 0.0 0.6 0.40 0.40 0.40
0.2 0.0 0.5 0.20 0.20 0.20
0.3 0.0 0.4 0.00 0.00 0.00
0.0 0.1 0.8 0.80 0.80 0.80
0.1 0.1 0.7 0.60 0.60 0.60
0.2 0.1 0.6 0.40 0.40 0.40
...
```

while the indices to be considered should be defined as:

```
0 1 5
5 4 0
1 2 6
6 5 1
...
```

Step 1

The first step consists in rendering the `DEM_test_small.dat` and then `DEM_test.dat` dataset, which are properly contained into the NDCS, as it is. You should obtain a result similar to the one shown in Figure 1.



(a) (b) (c) (d)

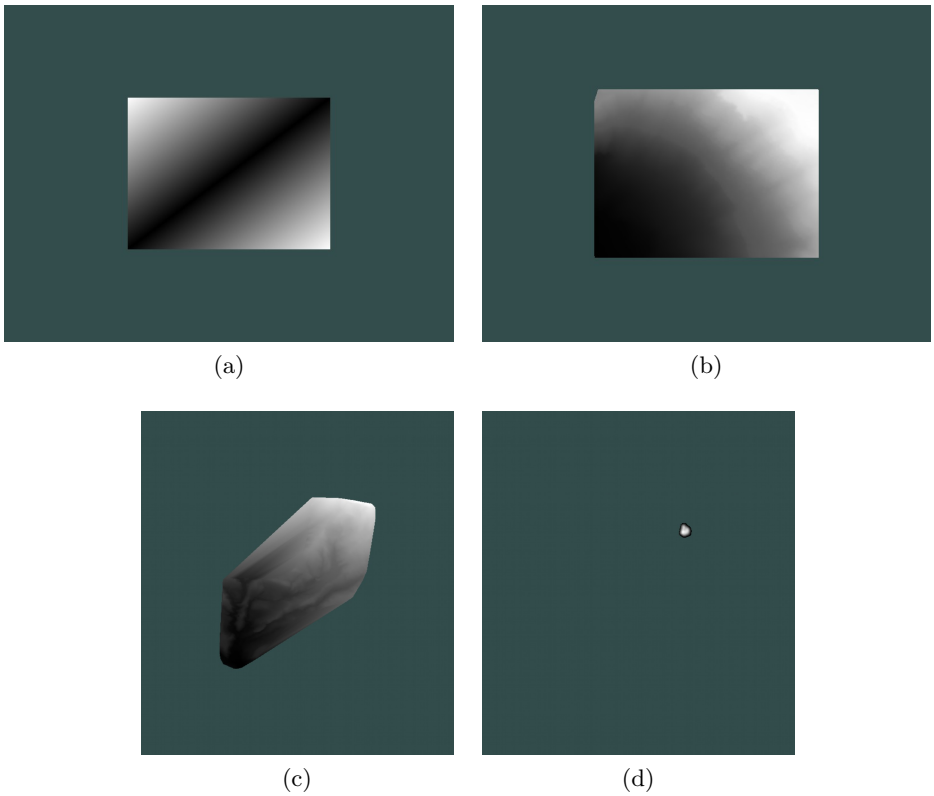
FIGURE 1: (a) Rendering of `DEM_test_small.dat` in wireframe. (b) Rendering of `DEM_test_small.dat` in fill mode. (c) Rendering of `DEM_test.dat` in wireframe. (d) Rendering of `DEM_test.dat` in fill mode.

Step 2

The next step consists in centering the dataset into the graphic window, scaling the dataset in case it is not contained into the NDCS, and to properly manage NODATA values (that must not be represented).

Note that the vertices transformations (translations and scaling) can be performed both host and device side. In the second case, however, vertices can be translated in parallel by vertex shader, which is certainly the best choice when the number of vertices is high. For this purpose, the host application can evaluate the displacements and scaling factors along each direction and send them to the vertex shades by means of uniforms.

Figure 2 shows what it should be obtained when different datasets are considered. In particular, those in Figure 2b-d do not are contained into the NDCS. Nevertheless, they are properly rendered thanks to the considered transformations.



(a) (b)

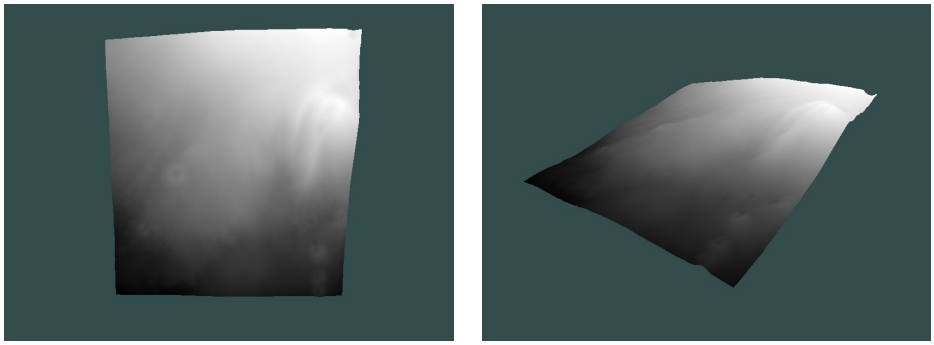
(c) (d)

FIGURE 2: (a) Updated rendering of `DEM_test.dat` in fill mode. (b) Rendering of `DEM_Albano.dat` in fill mode. (c) Rendering of the `tessina_ndem.txt` dataset. (d) Rendering of the `tessina_nsources.txt` dataset.

Step 3

The third step consists in introducing a perspective clip space (frustum). Modeling and viewing transformations can be applied to center the dataset into the frustum by allowing the data to be observable from different points of view.

Figure 3 shows an example of how the date should be rendered.



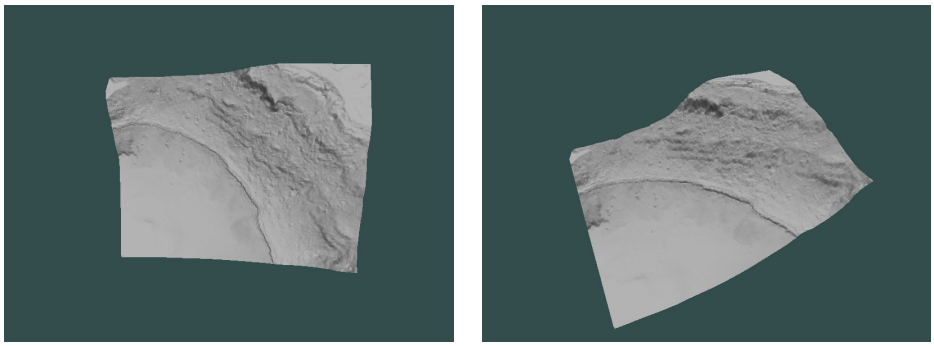
(a) (b)

FIGURE 3: (a) Rendering of the `altitudes.dat` from the `dataset_lava`. (b) Rendering of the `altitudes.dat` from the `dataset_lava`.

Step 4

The fourth step consists in defining a normal vector for each triangle of the mesh (by assigning the same normal vector to each vertex composing the triangle) and to consider the ambient and diffuse component of the Phong model to lit the surface. A source light acting as the Sun can be simply defined into the fragment shader by considering a source light direction $L = (0.0, 0.0, 1.0)$ with a color $L_c = (0.8, 0.8, 0.8)$. A flat color (e.g. $C = (0.8, 0.8, 0.8)$) can be assigned to the vertices.

Figure 4 provides an example of what should be rendered.



(a) (b)

FIGURE 4: (a) Rendering of the `DEM_Albano.dat` dataset. (b) Rendering of the `DEM_Albano.dat` dataset.

Step 5

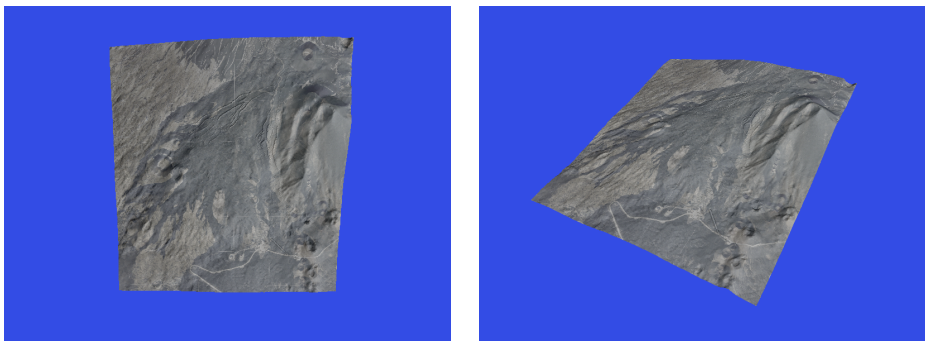
In this step we consider the `altitudes.dat` file from `dataset_lava` since it comes with a texture, namely `texture.png`.

In order to apply the provided texture to the topographic surface, it is necessary to properly change the host application, evaluate the texture coordinates, store them into the VBO used to send the vertex data to the GPU and update the fragment shader so that it can sample the texture to fetch the color to be used.

As regard the evaluation of the texture coordinates, consider that the texture must perfectly fit the topographic data. In other words, the texture coordinates $(0, 0)$ and $(1, 1)$ must be assigned to the lower-left and upper-right vertices, respectively.

Eventually, since the VBO now contains additional information regarding the texture, also the vertex shader must be updated in order to fetch this data and pass it to the fragment shader. For this purpose, an additional vertex shader attribute can be defined.

Figure 5 shows what should be obtained. In the example, the color sampled from the texture has been mixed with the color of the fragment evaluated by applying the Phong lighting model.



(a) (b)

FIGURE 5: (a) Rendering of `altitudes.dat` from `dataset_lava` with the `texture.png` texture applied. (b) Another view of `altitudes.dat` from `dataset_lava` with the `texture.png` texture applyied.