Prof. Dr. Michael Krone
Marco Schäfer
Marcel Bok
Big Data Visual Analytics in Life Sciences
Department of Computer Science & IBMI
Faculty of Science

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Scientific Visualization **SS/2021**
Assignment 8 **Due: July 8th, 2021, 9:00 am**

Code skeleton:

We supply you with a basic code template to use for this assignment but you are free to use the code you wrote for assignment one as a basis for this task. Our supplied code will contain code snippets assisting you in solving the tasks. So be sure to look at the supplied code if you choose to use your own code as a basis for this assignment.

1 Scalar Field Visualization (20 pts)

In this assignment you are asked to write a vertex shader implementing terrain rendering. The basic idea is to apply a vertical displacement to a mesh of vertices depending on the values read from a texture (i.e., modify the height/y value). This texture is thus called *heightmap* and contain only grayscale values (Figure 1).

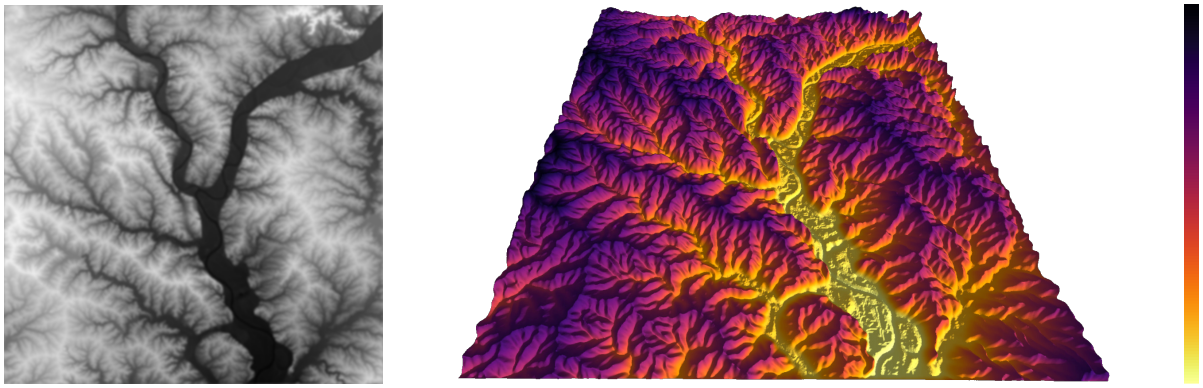

Figure 1: Left: Heightmap of terrain, Right: final render of terrain with the height mapped to the 1D color map shown to the right.

In the left image, the lowest point is colored black (0.0,0.0,0.0), while the highest point is colored white (1.0,1.0,1.0), with intermediate points being linearly interpolated. In the shader, this information can then be used to vertically displace vertices according to this values to recreate the topography of the heightmap on the final terrain.

The basis for this is to have a regular geometry with triangular faces (Figure 2).

In a next step, this geometry has to be supplied with texture coordinates – so-called uv-values – for each vertex such that the furthermost vertex in each direction is supplied with 1.0 (Figure 3). This is crucial as the texture is mapped via uv-mapping, this means that information in the texture (accessed by these texture coordinates) is mapped to the vertices with the corresponding

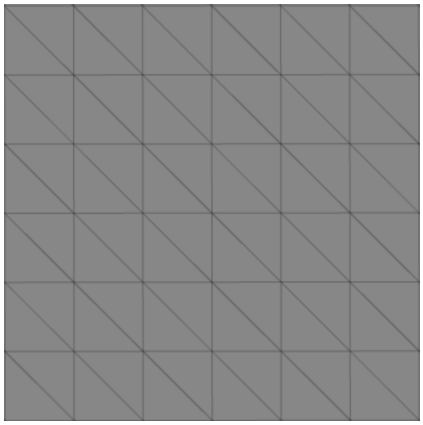

Figure 2: Uniform grid with triangular faces.

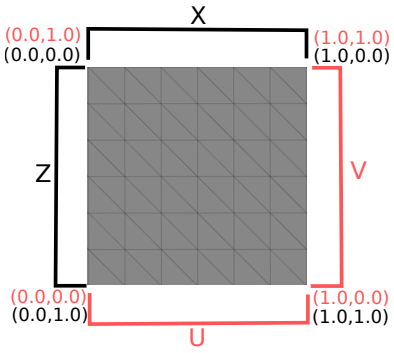

Figure 3: For texture mapping (in this case onto the XZ-Plane, uv values need to be assigned to each vertex: the top-left vertex should be assigned (0.0,0.0), while to bottom-right vertex should be assigned (1.0,1.0). The uv coordinates of the vertices between have to be interpolated linearly.

uv-coordindates. A given texture acts like a kind of lookup table, in which for a given uv-coordinate a specific value is extracted from the texture (see lecture chapter 2 "Computer Graphics – Part 2", slides 26/27).

While textures are usually used to apply color images to geometry, it can be used for any kind of information stored in an image file, like in our case, mapping the color value to a vertical displacement (Figure 4).

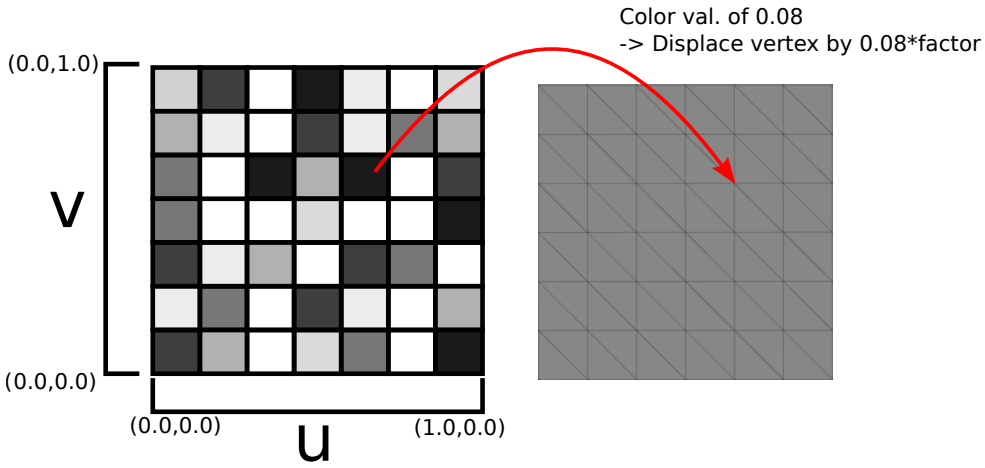

Figure 4: Vertical vertex displacement using a heightmap and uv-mapping.

1.1 Uniform Grid With Triangular Faces and UV Coordinates (7 pts)

In a first step, write a function that generates the vertices for a uniform (equidistant rectangular) grid in the XZ-plane. The center of the grid should be placed at the origin of the coordinate system. Furthermore, your function should allow to adapt the amount of vertices generated to change the level of detail of the resulting visualization. Your function also has to create the triangles that connect the vertices as shown in Figure 2, that is, you have to compute the index lists to render these triangles. Irrespective of the amount of vertices, the total extent of the grid should stay the same (20.0 in each X and Z direction, that is, -10.0...10.0). This grid

should then be used in a `THREE.BufferGeometry` to generate and render a geometry with same sized triangular faces, facing upwards, and correct UVs ranging from 0.0 to 1.0 in both X and Z direction (note that the Y values of all vertices can be set to 0.0).

Hint: You can use `THREE.MeshBasicMaterial` with the `wireframe` option set to `true` to debug your code.

1.2 Use the Heightmap Texture to Vertically Displace Vertices in the Vertex Shader (9 pts)

1.2.1 Texture the Grid using the Heightmap Texture (2 pts)

In order to use provided heightmap in the vertex shader, you first have to load the heightmap image file (`assets/heightmap_flat.png`). Use this texture as `map` in `THREE.MeshBasicMaterial` to apply the image to your grid to make sure that the texture is mapped correctly (see Figure 5).

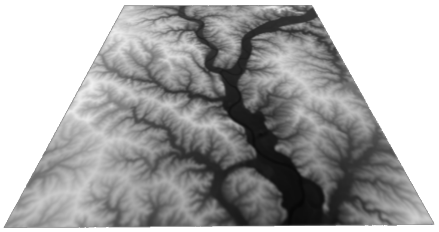

Figure 5: Textured grid.

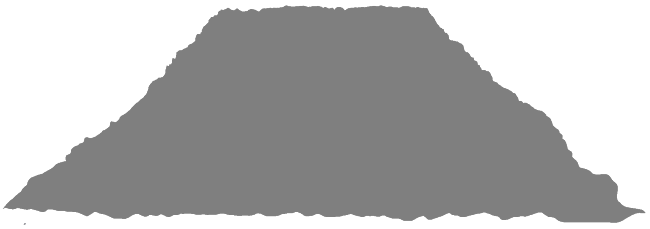

Figure 6: Displaced grid.

1.2.2 Displace Grid using the Heightmap Texture (3 pts)

Now we will us the heightmap texture to vertically displace our grid vertices and create an actual terrain rendering. For this, we will replace the MeshBasicMaterial with our own material that uses custom shaders.

Use the incomplete vertex shader code in `vertexShader_heightmap.vert.js` to write a vertex shader that uses the heightmap texture to vertically displace the vertices of our grid according to its uv-mapping and a scaling factor (for this use a new `uniform float scaleFactor;`). Keep in mind that you have to upload the previously loaded heightmap texture to the already available texture variable (`uniform sampler2D heightmap;`) in the shader. You can experiment with the scaling factor and its influence on the resulting topography. Your result should look similar to Figure 6. As you can see, the shading is not correct, since we are still using the normals of the original, undisplaced grid (0.0, 1.0, 0.0).

1.2.3 Terrain Normals using Central Differences (4 pts)

In order to get a proper shading, correct normals must be calculated for the displaced mesh. For this, we will use the finite difference method with central differences to obtain the derivatives of the heightfield in x and z direction. Using these derivatives, we can then calculate the normals. To choose/calculate a proper offset value (see Figure 7) you can use `ivec2 size = textureSize(heightmap, 0);` in your vertex shader, which will give you the size of your texture (that is, the number of pixels in each dimension). You can see the result in Figure 8.

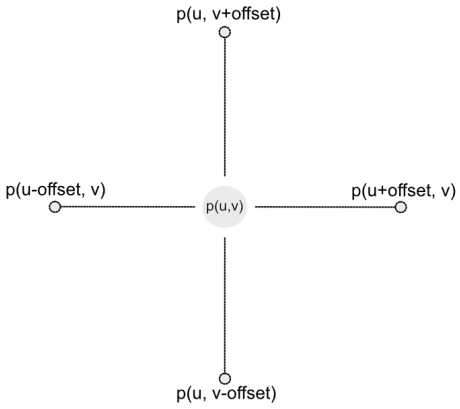

Figure 7: Basic idea of central difference finit-difference derivative generation.

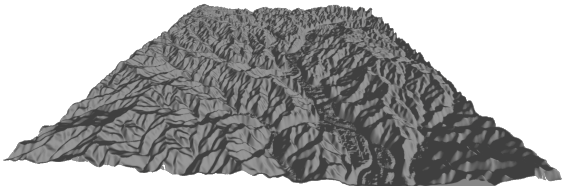

Figure 8: Displaced geometry with Phong shading without specular term and gray base color.

1.3 Use a Colormap as Transfer Function to Generate Vertex Colors (4 pts)

Currently, the supplied code uses Phong shading with gray as base color (Figure 8). Use one of the supplied color maps (Figure 9) from the assests folder as a transfer function to map the height read from the heightmap to the vertex color (Figure 10). That is, you have to load the image as another texture and use it in vertex shader (`uniform sampler2D colormap;`) to get the correct color for each vertex. Note that the v coordinate can always be set to 0.0 when accessing the texture, since only the u value is relevant.

Figure 9: Simple color scale as a 100px by 1px image.

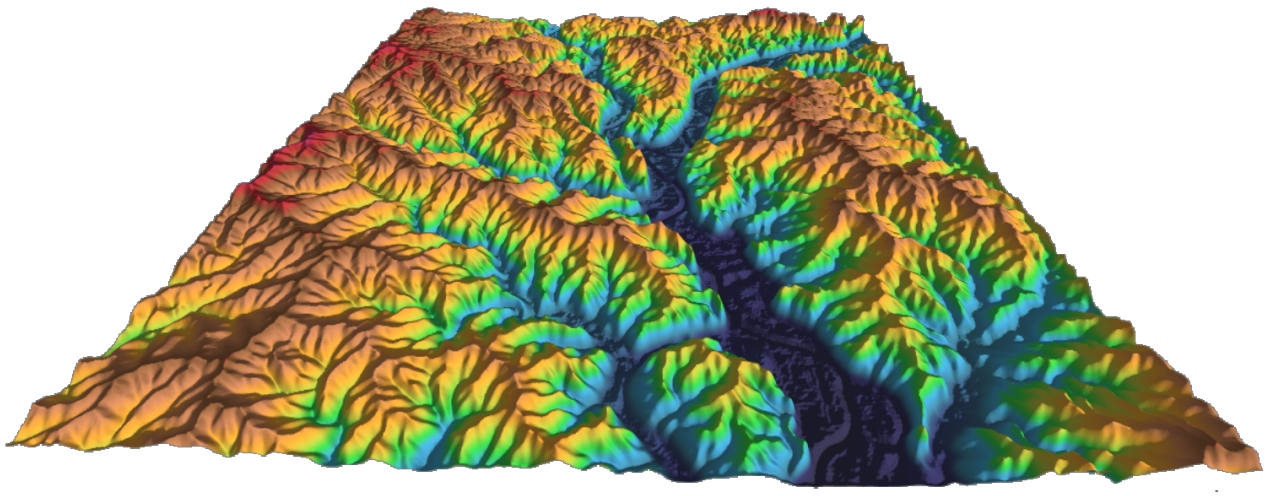

Figure 10: Displaced geometry with vertex coloring obtained from a 1D colormap transfer function

1.4 Bonus: Add Water to the Created Terrain (1 pts)

Add a flat water surface on a certain height to your terrain. Keep in mind that water is transparent, so your water surface has to be transparent as well (Figure 11).

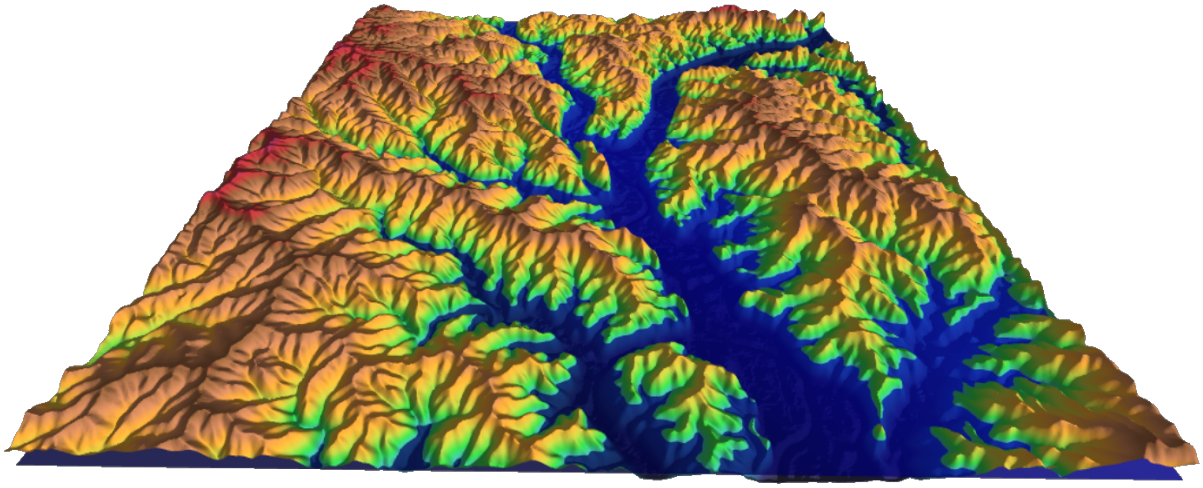

Figure 11: Displaced geometry with vertex coloring and transparent water surface.

Handing in

For the Hand-in, write the names of both group members at the top of all code files and PDF documents. Create a ZIP archive of your project folder and **PLEASE EXCLUDE** the folder **NODE_MODULES** and also the **main.js** file from your archive. Name it with the last and first name of each group member, and the assignment number (e.g., *schaefer_marco___bok_marcel_assignment8.zip*) and upload it to `Assignment_8`.