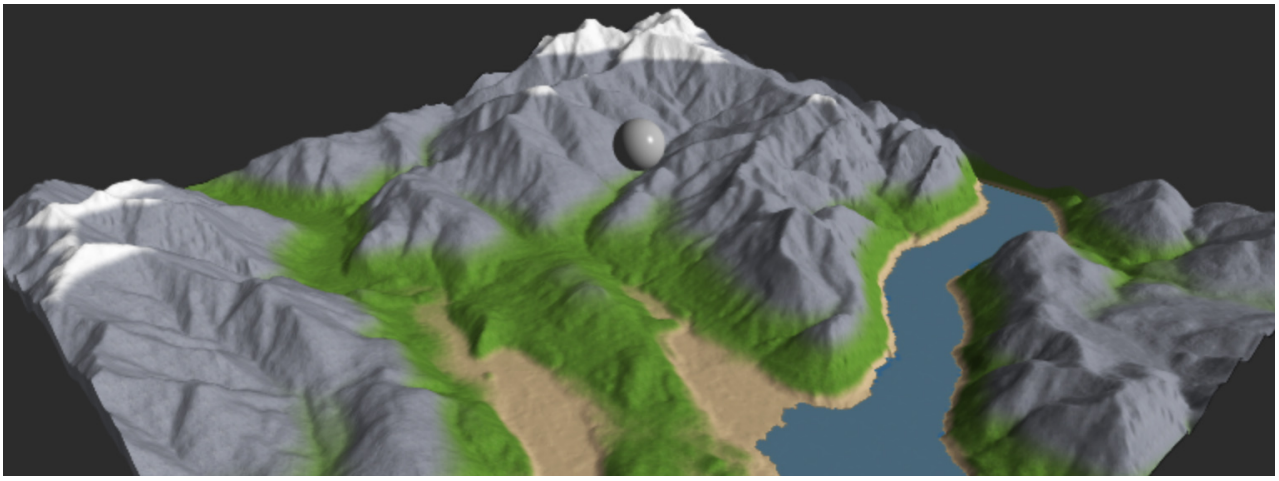
Università
della
Svizzera
italiana

Faculty
of Informatics

Computer Graphics

Assignment 6: Texturing



In this assignment, you will use texturing techniques to add more details to the scene. We consider the task of rendering a terrain like the one above. The assignment is split into there exercises and one bonus exercise. You will start from the flat land to which you will first add height using displacement mapping. Next, you will texture the terrain using images of, for example, grass, sand, stones, etc. In the third exercise, your task is to estimate normal directions for the terrain from the heightmap and add simple diffuse shading to the scene. The last (bonus) exercise is about improving normal estimation to get a more accurate shading. The image above presents a sample rendering after completing all the tasks.
Important: You have an option to opt out from these tasks and choose to implement texturing in our raytracer. The option might be especially appealing for people taking part in the rendering competition. Please find more details on that at the end of the document.

Template Update

The current template provides a solution to the previous assignment. It was also modified to accommodate new functionalities you will add in this assignment. Below, find the most significant changes to the previous template:

1. We added a new file ***tools.js***. It is a container for helper functions to clean a bit the main JavaScript file.
2. The ***geometry.js*** file includes now a function for creating terrain geometry, which is a subdivided quad (Figure 1). A variable inside the function can control the amount of subdivision. Look for the comments in the file.
3. The main file includes now additional functionality related to the terrain rendering and textures setup. For example, there is a definition of additional shader program for terrain rendering, the code for creating a VAO for terrain, as well as reading textures, uploading them to GPU, and setting them up during the rendering. Some of the code has to be still uncommented. Look for the comments in the file. To correctly initialize the textures you will need to feel two arrays ***textureFiles*** and ***textureVariables***. They should include the paths to the images and corresponding names in the shader program.

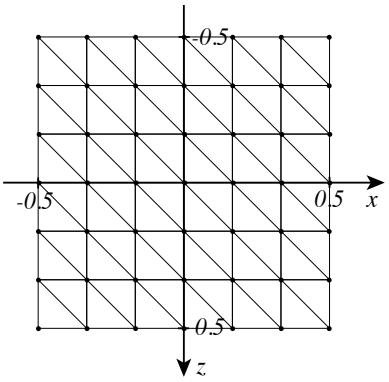


Figure 1: The definition of the terrain geometry; view from the top, down the y-axis. The z-values for all vertices are equal to 0.

Feel free to adapt the template, rewrite it in your way, or copy parts of it to your solution of the previous assignment.

Exercise 1 [3 points]

To complete this exercise, you will need to add displacement mapping to the terrain. You should start by getting a displacement map corresponding to terrain altitude. You have two options for that:

1. On iCorsi you can find preselected fragments of the altitude map of Switzerland.
2. There is also a link to the altitude map of the entire Switzerland from which you can crop a region you would like to work on.

In general, we recommend looking for places with nice height variation as well as some water. Your next step should be adding the heightmaps as a texture to the WebGL code. You should also define a new shader program (both vertex and fragment shader, see the template) for rendering the terrain. In this assignment, the shader program should: (1) displace the vertices along the y-axis according to the information in the heightmap; (2) render the terrain using one color, e.g., grey. Optionally, you can color the surface of the terrain with the height. This will allow you to investigate better whether the displacement mapping works correctly.

Exercise 2 [3 points]

The next step is adding the textures to color the terrain. You should use at least three different textures. In the example from the beginning of the document, we used the textures for grass, sand, and rocks. Additionally, the water and the snow have single colors, i.e., they are not textured. For the textures, you will find many resources online. In particular a very useful one is 3DTextures[1]. On iCorsi you can also find a video demonstrating how to download textures from this website.
We suggest applying textures to the terrain according to the height. In other words, the final color of the fragment should get the value of the texture depending on which height it corresponds to. To make the scene nicer, you can blend the textures on the transitions. This can be done, for example, by using GLSL ***smoothstep*** function.
Alternatively to the above texturing strategy, you can prepare additional texture/textures, which encode the masks for each material. You can get very nice results with such a technique, but it requires a bit more work. For the inspiration on this you can check the following article[2]

Exercise 3 [2 points]

In the last step, you should add the shading to your terrain. Please assume that the terrain is a Lambertian surface. For this exercise, you will need to expand the fragment shader for the terrain. In particular, for each

[1] `https://3dtextures.me/`
[2] `http://www.forgottenplanet.com/creating-height-and-material-masking-maps-for-iclone-5-5-using-photoshop/`

fragment (in the fragment shader), you need to estimate the corresponding normal vector using heightmap information. To this end, please recall that the normal vector can be estimated as a cross-product between two vectors, which are tangential to the surface. While you can compute tangential vectors along any direction, the most natural way is to choose the vectors along u and v-axes. You can estimate the vectors by fetching the neighboring texels from the heightmap.

Tips:

- Leave the Phong-shaded sphere from the previous assignment and set it such that it appears above the terrain. This will allow you to easily identify the direction of the light and check whether the shading is correct.
- You may want to scale the terrain non-uniformly using a model matrix to flatten it a bit. Remember to use $(M^{-1})^T$ matrix for normal transformation, where M is the model matrix.
- To get a nice surface of water, you can change the normal estimation for the water to be $[0.0, 1.0, 0.0]^T$.

Bonus [2 points]

To get extra bonus points, you can improve the normal estimation in your renderer. Currently, the problem is that your Web browser most likely does not support 16-bit images while the downloaded hightmaps should, in fact, be 16-bit images. This means that your heightmap in the GLSL program is represented using only 8-bit. This introduces visible errors in the estimated normals (see the illustrations in Figure 2). But there is an old trick in computer graphics to overcome this limitation. You already use a texture that has at least three channels (red, green, blue), and each of them has 8-bit precision. You can, therefore, encode your 16-bit heightmap in two channels, for example, by encoding the first 8-bits in the red channel and the others in the green channel. This way, you can store in the texture all 16 bits even though the texture has only 8-bit precision. You can use any means to do the conversion. To generate the sample solution, we used a short Matlab script. Feel free to use any programing language. To do the improved normal computation in your fragment shader you will need to decode the height from the two channels before computing the normal vector.

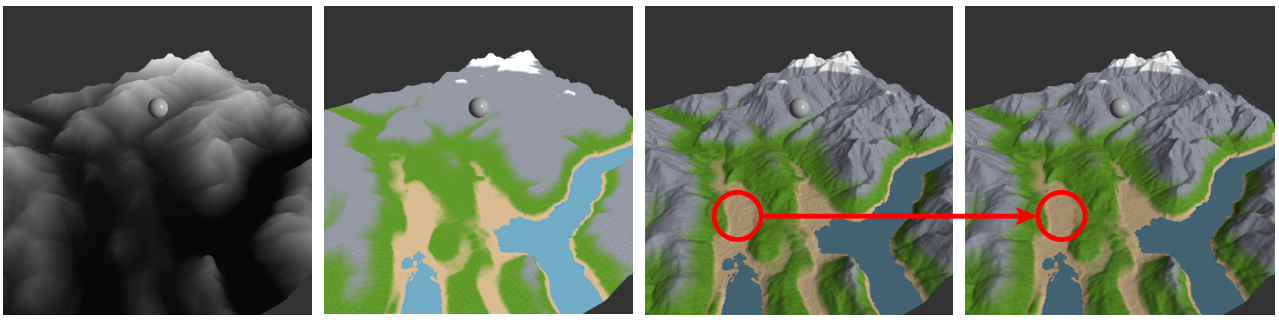


Figure 2: The illustrations of the sample solutions for the individual exercises. Please, note that in the first image, the color of each fragment is set according to the input height. The red circles demonstrate the problem when using only 8-bit precision and the improvement proposed in the bonus exercise.

Texturing in the raytracer [10 points[3]]

If you like the raytracer we were developing in the first part of the course, now it is the time to extend it by nice texture mapping. Instead of solving the above exercises you can implement texturing in your raytracer. You should use at least one object to be textured. It can be either a sphere or a cone. Solving this version of the assignment will enable you to add interesting materials to the scene. This is the alternative list of tasks which at the end will sum up to the same number of points as the regular assignment:

[3]The points from the rasterization and the raytracing parts cannot be combined. You have to choose one option.

- **Images as textures [4 point]** You will need to start by making your raytracer application read images. For this you will need to use an external library. One option is to use a library for reading JPEG files. Reading these files is, however, complicated and so are the usually available libraries. Alternatively, you can convert images representing textures to PNG, BMP, or PPM files using available tools for image manipulation. Libraries for reading these formats are usually much simpler. One possibility is to use a one-file bmpmini library[4] for reading BMP files. However, you are free to use any solution and library you like. Once you are able to read image files, implement computation of UV coordinates for ray-object intersection points and use them to texture one object in the scene. For the source of textures see Exercise 2.
- **Normal mapping [4 points]** While the previous tasks is about getting simple texturing working, now you are asked to also read normal map and implement normal mapping. For this, you will need to compute tangent and bitangent vectors to properly transform normals read from the files to the object space.
- **Ambient occlusion and roughness textures [2 points]** Make use of ambient occlusion and roughness textures. Ambient occlusion texture represent the amount of indirect light and can be used as a multiplier for the ambient term in the Phong reflectance model. Roughness can be used to calculate shininess parameter in model. Note that roughness is opposite to shininess, and you can map the roughness value to the shininess parameter in whatever way you want. One possibility is to use something like:

$$shininess = \frac{0.5}{roughness^4} - 0.5.$$

Submission

Your submission **must** contain one ZIP-file with:

- a readme file or a PDF document file with information about which exercises you solved, the authors of the solutions, and the explanation of encountered problems, if any,
- a directory named *code* containing all the source code required for running your renderer with all additional files, e.g., textures.

The ZIP file name must be of a form *surname1_surname2.zip* for a team of two, and *surname.zip* for a single submission. Only one person from the team should submit the solution.

Solutions must be submitted via iCorsi by the indicated there deadline.

[4] `https://github.com/yangcha/bmpmini`