Praktikum Echtzeit-Computergrafik
Assignment 2 – Procedural Modelling

Technische Universität München
Institut für Informatik
Lehrstuhl für Computergrafik & Visualisierung

Christoph Neuhauser, Simon Niedermayr – SS 24

"Procedural modeling is an umbrella term for a number of techniques in computer graphics to" - automatically - "create 3D models and textures from sets of rules."[1]

[1] https://en.wikipedia.org/wiki/Procedural_modeling

Introduction

Description

In this assignment, your task will be to procedurally generate a terrain model, i.e., a height field, that has the shape of a realistic terrain. This model then needs to be converted into a triangle mesh, stored in a specific format, and finally loaded into ShaderLabWeb (https://vmwestermann10.in.tum.de/) to display it.

Technically, you need to compute a 2D field (an array) storing values in (0,1), These values are interpreted as height values over a 2D domain. The values are not computed randomly, but via an algorithm that generates a distribution that looks as realistic as possible.

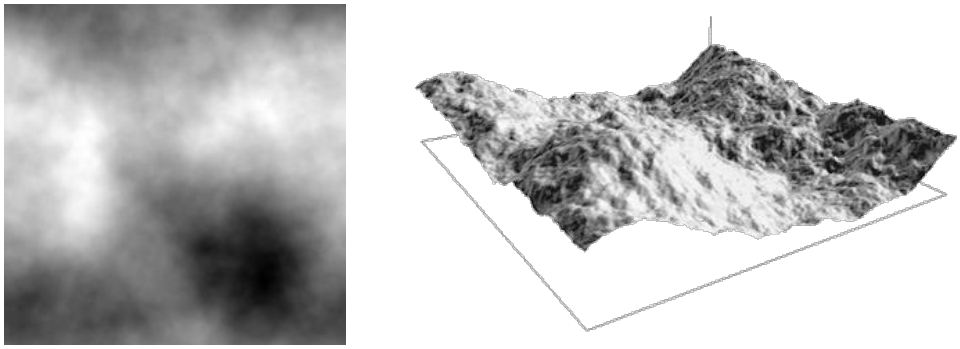

Figure 1: Left: 2D array/image with height values. Right: Triangle mesh generated from height values.

How to create a field that looks like a natural terrain field?

- In nature, we often see structures at different scales overlaid on each other.
- While coarse scale gives basic shape, the finer scales add more and more detail.
- Looking at the displayed height field above, it seems to follow roughly a cosine curve but has some fine details added on top of it.

Midpoint Displacement

Below, a description of terrain modelling via **random midpoint displacements** (first explained in 1D, see Figure 2) is given.

- Generate 2 random height values at the beginning and end of the 1D domain. Let the domain be represented as a 1D array of size N, then the first and last entry are initialized with a height value.
- **Interpolation**: Linear interpolation between the 2 initially given values is used to obtain a new value in-between.
- **Displacement**: In each iteration i, modulate the interpolated values by adding random displacements D_i to each value. Displacement values become smaller and smaller in each iteration.
- Repeat **interpolation** and **displacement** now using the initial and new values.

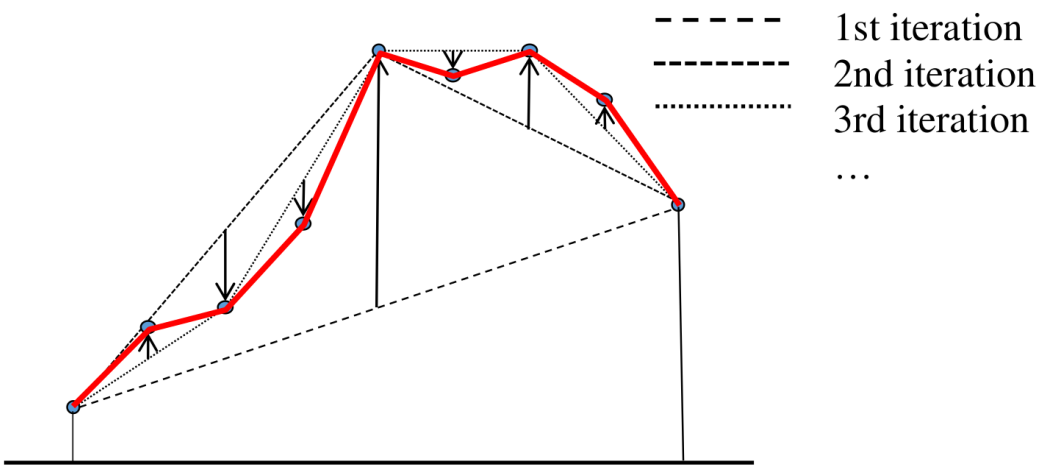

Figure 2: Left and right vertical lines indicate the two initial height values. Vertical arrows indicate the displacements that are added to the linearly interpolated values. Red curve indicates the final height field.

How to generate the **random displacements**:

- Random displacements are modelled via random numbers with a certain distribution; the distribution defines the probabilities of the occurrence of values of a random variable.
- E.g., uniform [0,1] distribution: all values in [0,1] occur equally likely.
- E.g., normal distribution with expectation value μ and standard deviation σ, i.e., $N(\mu, \sigma^2)$: the values are more likely to occur close to μ, 68.2% of values fall between $\mu \pm \sigma$. σ is the variation (spread) from the expectation value.

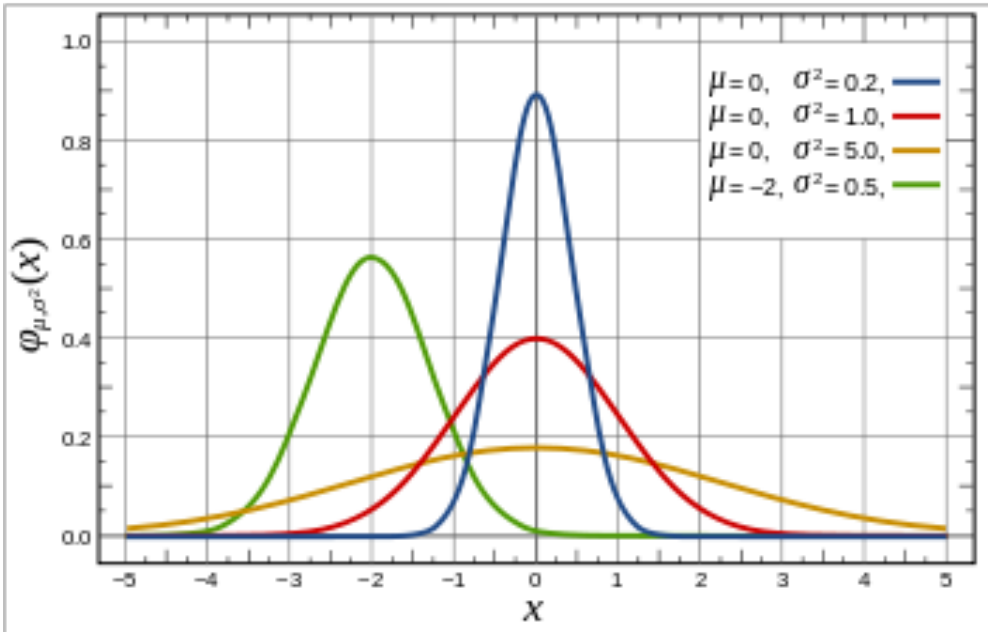

Figure 3: Graphs of the probability density function of of different normal distributions.

In Java, you can use the builtin function nextGaussian() (from java.util.Random) to compute normally distributed random values.

```
Random rand = new Random();
float stdNormVal = rand.nextGaussian();
```

In each iteration i of the midpoint displacement algorithm, random displacements D_i are computed. The D_i are modelled via normal distributed random numbers with expectation value $\mu[D_i] = 0$ and standard deviation $\sigma[D_i] = \sigma_i$. Starting with σ_0 at the first level, we want to generate ever smaller random displacements at subsequent iterations. This is modelled via $\sigma_i = \sigma[D_i] = \frac{\sigma_{i-1}}{2^H}$. H is a constant that can be set to control the roughness of the terrain. Use different values for H and generate your favoured look. Because the random number generator of Java only gives you N(0,1) distributed random numbers, you can use $N(0, \sigma^2) = \sigma N(0, 1)$ to generate your required distribution.

Diamond-Square-Algorithm

The Diamond-Square-Algorithm can be used for terrain generation over a 2D domain, i.e. using a 2D array of height values. Below, a description of the different steps of the algorithm can be found.

- Assign (random) values to the corners.
- **(Diamond)** Assign the average of four corners plus a random displacement to the midpoint.
- **(Square)** For each diamond, average four corners and add a random displacement.
- Repeat step 2 and 3 for a given number of iterations.

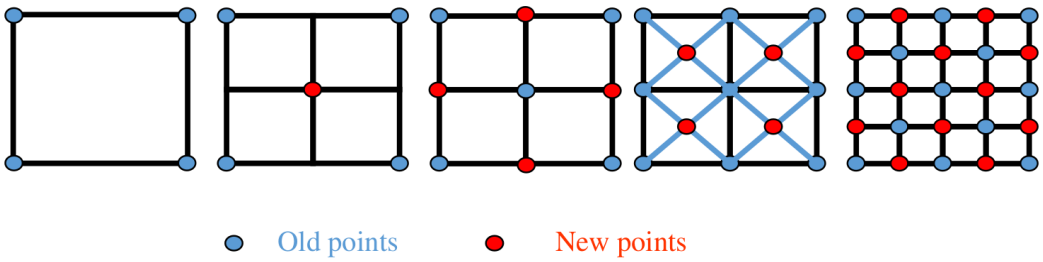

Figure 4: Diamond-square steps.

To generate the triangle mesh from the 2D array with height values, 2 triangles are formed from each 2x2 block of adjacent array elements (see Figure 5). The triangle mesh needs to be written to a file as a shared vertex representation, i.e. .obj file, as explained in the lecture and specified here: `https://en.wikipedia.org/wiki/Wavefront_.obj_file`. Please note that you can omit writing material information.

Therefore, you can first loop over all array elements, and create and store one vertex for each element. Then, you loop over all adjacent blocks of 2x2 vertices to form 2 triangles, which are then each stored as 3 indices into the vertex list.

The final generated mesh file can be loaded in ShaderLabWeb (via `Model` → `Mesh` → `.obj File`) and rendered as a wireframe (via `Model` → `Show Wireframe`) or with each triangle having constant color.

Tasks

- Write a Java program that uses the Diamond-Square-Algorithm to generate a terrain mesh in the .obj format. Please keep in mind that a) the Diamond-Square-Algorithm needs a domain size of $2^N + 1$ for $N \in \mathbb{N}$ and b) you may need to write special code for handling the steps at the borders of the domain.
- Import the mesh in ShaderLabWeb via `Model` → `Mesh` → `.obj File`. Play around with, e.g., the wireframe mode (`Model` → `Show Wireframe`). Hint: You can use Shift + Click to move the camera.

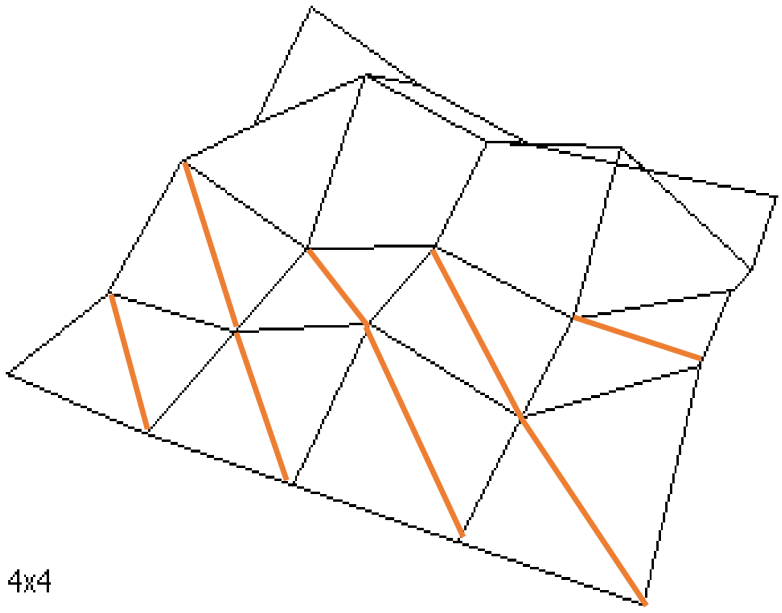

Figure 5: Sketch of a terrain subdivided into individual triangles.