Game Engine Design

Assignment 2 – "*Fractal Landscape Generation*"

10 Points

In preparation of your 3D game, we will create procedural heighfields which will be used as a terrain lateron. The terrain generator will be a standalone commandline tool and will be extended by texture generation in the next assignment.

Preparation

- In `external\Tools\bin`, you will find `TerrainGenerator\TerrainGenerator.exe` as well as `TerrainViewer\TerrainViewer.exe`. Open a windows command line (fast lane: Windows-Key + R, type "cmd" and hit enter) and execute `TerrainGenerator.exe` with the following parameters:

  ```
  -r 1024 -o_height C:\local\gep\<username>\terrain_height.png
  -o_color C:\local\gep\<username>\terrain_color.png -o_normal
  C:\local\gep\<username>\terrain_normal.png
  ```

  This will create a 2D heightfield of resolution 1024 times 1024 (the `-r` parameter) as well as a texture and normal map and write it to the specified paths.

- Start `TerrainViewer.exe`. In the "Open File" dialog, select the 3 files which were just created (in the order terrain_height, terrain_color, terrain_normal). You should see a spinning landscape.

Project Setup (1P)

We provide you with an existing solution that already contains some required libraries as well as a template for our game. For the time being, we will only add our own project to the solution and ignore the existing game template.

- **Copy** the existing solution from `external\templates\GEDGame` to your local folder (`C:\local\gep\<username>\GEDGame`).
- Open the existing solution (GEDGame.sln).
- Set the following additional library path for the "Game" project (Properties -> Configuration Properties -> Linker -> General -> Additional Library Directories):
  - For Visual Studio 2013:
    ```
    ..\..\..\..\external\Tools\lib\VS2013\$(Platform)\
    ```
  - For Visual Studio 2015:
    ```
    ..\..\..\..\external\Tools\lib\VS2015\$(Platform)\
    ```
  - For Visual Studio 2017:
    ```
    ..\..\..\..\external\Tools\lib\VS2017\$(Platform)\
    ```

- Then build everything (Right-click on the solution -> Build Solution). All projects should compile without errors or warnings.

- Create a new project "`TerrainGenerator`" inside the solution (Right-click the solution -> Add -> New Project). As location, choose the subfolder "projects" inside the solution directory. As before, create a Win32 Console Application and uncheck the "precompiled headers" checkbox in the wizard.

- Make this new project your startup project (Right-click the project -> Set as Startup Project)

- Add the following additional C++ include path (Project Properties -> Configuration Properties -> C/C++ -> General -> Additional Include Directories). Make sure you have "All Configurations" and "All Platforms" selected before making these changes!

```
..\..\..\..\external\Tools\include\
```

  Note that this path is relative to the **project** directory.

- Add the following additional library path (Properties -> Configuration Properties -> Linker -> General -> Additional Library Directories):

  - For Visual Studio 2013:
    ```
    ..\..\..\..\external\Tools\lib\VS2013\$(Platform)\
    ```
  - For Visual Studio 2015:
    ```
    ..\..\..\..\external\Tools\lib\VS2015\$(Platform)\
    ```
  - For Visual Studio 2017:
    ```
    ..\..\..\..\external\Tools\lib\VS2017\$(Platform)\
    ```

- Add the library "GEDUtils.lib" to the dependencies of the Release-Configurations and "GEDUtilsd.lib" to the Debug-Configurations (Properties -> Configuration Properties -> Linker -> Input -> Additional Dependencies).

- Build TerrainGenerator.

Command Line and Test Heightfield (2P)

As a first test, we will create a 2D field of random numbers and write that into a file.

- Your TerrainGenerator should parse the command line to accept arguments of the following form:

```
-r <resolution> -o_height <output heightfield filename> -
o_color <output color filename> -o_normal <output normal
filename>
```

Check and convert the parameters passed to your `main()`-function accordingly (see the slides!). If a parameter is missing or `<resolution>` is <= 0, print an error message to the console and exit the application.

- Create a 2D array of size `<resolution>` times `<resolution>` (as given by the command line parameter). Create the array as a flattened 1D array just as in the last assignment!
- Fill the 2D array with random floating point values in the range of [0, 1]. The values should be **normally distributed** – use the appropriate distribution from the C++11 Standard Library (see slides!) instead of `rand()`.

- Save the heightfield to the file path given by the command line argument `o_height`. This can be done using the included `GEDUtils::SimpleImage` class – include <SimpleImage.h> and check the slides and the comments in the header file (you can right-click the #include in the editor and select "Open Document").

- As a last step, your program needs to output textures for the generated terrain. For now, this is done using GEDUtils::TextureGenerator:generateAndStoreImages() (TextureGenerator.h).

Diamond-Square-Algorithm (6P)

The newly created project should now be extended to perform the Diamond Square Algorithm. Check the slides for details on the algorithm.

- Instead of filling the array from the previous part of the assignment with random numbers, fill it using the Diamond Square Algorithm. It should create floating point values in the range [0,1] for the given resolution and a certain (hard-coded) roughness (try some values between 0 and 1).

    - As in the last part of the assignment, use normally distributed random numbers
    - Copy the macro from the last assignment to convert a 2D index into a 1D index
    - We suggest implementing the algorithm in a separate function or class as we will add some more code to your project next week
    - To create a heightfield of resolution $(2^n)^2$, run the Diamond-Square on a heightfield of resolution $(2^n + 1)^2$. Skip the last row and column lateron when you **save** the heightfield
    - Try to break down the algorithm into multiple functions, e.g. a diamondStep(…) and squareStep(…) function, and some utility functions (e.g. random(min, max), getHeight (x, y), setHeight(x, y) etc…).
    - Use the debugger to test your algorithm step by step; you will need to set the according command line parameters in the Visual Studio project settings ("Debugging"). Use a small resolution (e.g. 4) to investigate your heightfield in the memory window!

- Check if your heightfield is displayed correctly in the TerrainViewer. You can also open the saved .png file directly; you should then see the height values as grey values (the brighter a pixel, the higher the value).

You will get 1 Point for correct random number generation. You receive the remaining 5 Points for a correct implementation of Diamond Square; 3 Points, if you produce minor artifacts and 1 Point, if you produce major artifacts.

Heightfield Smoothing (1P)

- Use your already implemented Mean Value Filter from the last Assignment (just copy & paste the code) to smooth the heightfield after the Diamond Square Algorithm. Check your heightfield in the TerrainViewer again.
    - Take a look at the last assignment for details and special border cases!
    - Apply the filter multiple times (try 1x, 10x, 100x, …) to amplify the smoothing effect. Check the visual result for a view values and just select what you think looks best.

If you face any difficulties, please use the Q&A Forum at https://qage.in.tum.de/ or ask your tutor.