The TerrainLib project aims at providing simple to use, and fast algorithms that can not only be re-used in other projects, but also be used by end users as the library is easy to use.

TerrainLib Documentation

Release 0.0.1

Nathan Graule

Jul 05, 2018

Contents:

TerrainLib is a library of algorithms used for landscape generation. They include generation, filtering and export functions to allow to do everything with only one dependency: this library.

TerrainLib aims to be useful at both other projects wanting to include terrain generation, and end-user that don't mind using scripting as a mean to generate landscape. The API is easy to use by striping away the complicated stuff while staying customizable.

In a few lines you'll be able to create realistic looking terrain of any size.

CHAPTER 1

Coverage

Test coverage data is available under the coverage folder.

CHAPTER 2

License

TerrainLib is licensed under LGPL v3. Non open-source, commercial projects are welcomed, but should ask for explicit licensing agreement.

2.1 Getting Started

2.1.1 Installation

TerrainLib is available on PyPi, therefore the easiest way to install it is the following:: pip install TerrainLib

You can also install the library by downloading a version over the GitLab project home.

2.1.2 Introduction to the library

Once the library installed, you can import the algorithms from those three places:

- `terrainlib.generator` holds all generators, in sub-categories
- `terrainlib.filters` holds all filters, also in sub-categories
- `terrainlib.readers` holds all readers, in sub-categories

Definitions

What do we mean by those words? It may sound weird, as I, a non-native English speaker chose those words, but it allows everyone to know what they're talking about:

Generator An algorithm that only has parameter inputs, and outputs a terrain. This include image import, Perlin noise, etc.

Filter An algorithm that both inputs and outputs a terrain, while also accepting other parameter inputs. Those are erosion algorithms, manipulations on the terrain, masking, etc.

Reader An algorithm that inputs a terrain (and parameters) and output something completely different. These include image export, mesh export, texture output, 'playability' scores, etc.

By feeding each the output of the previous algorithm we are able to define *pipelines* that define our terrain. Let's create one right now.

2.1.3 A first pipeline

Start off by importing the Diamond Square generator. This is a fast noise generation alrogithm that can output detailed terrain fast - at the cost of not being tile-able.

```
1 from terrainlib.generators.procedural import DiamondSquareGenerator
```

The output of that generator will be either too flat or too rough for the terrain to be interesting or even somewhat realistic. To smooth things out, we need to simulate the process by which rocks and dirt fall downhill. We need some thermal erosion:

```
2 from terrainlib.filters.erosion import ThermalErosionFilter
```

Last, but not least, we need to export that terrain somewhere. The best way to do that is through an image - it's easily visualisable, and can be imported anywhere really. Load up the Image exporter:

```
3 from terrainlib.readers.image import PILImageReader
```

Now, we need to initialize those. We're going to create a 1025 pixel wide terrain, with a decent amount of roughness, then apply 150 iterations of erosion at standard rates, and then export it to 'terrain_out.png' as a 16-bit image.

```
4 generator = DiamondSquareGenerator(10, 0.1)
5 erosion = ThermalErosionFilter(150)
6 img_reader = PILImageReader(PILImageReader.BITDEPTH_16)
```

Couple of things to notice:

- Due to the way the Diamond Square algorithm works, we do not enter the desired size directly, but the power of two that will result in the desired size. Here, we're taking the 10th power of two (2^10 = 1024), and the algorithm adds one to that number (this is a technical restriction, the algorithm needs a center pixel to work with, needing an odd number of pixels on the side). Thus, we get a 1025 pixel wide terrain.
- We aren't providing a filename to the reader *directly*, because the reader outputs a Pillow image. The actual file will be saved from the *PIL.Image* instance.

Now, let's setup our pipeline:

```
9  terrain = erosion(generator())
10 img = img_reader(terrain)
11 img.save('terrain_out.png')
```

If you run your script, and after some processing time, it will have created a file named 'terrain_out.png' with the terrain saved in it. If you open that image as a heightfield, you will see your image in all its glory!

Here is the whole script:

```
1  from terrainlib.generators.procedural import DiamondSquareGenerator
2  from terrainlib.filters.erosion import ThermalErosionFilter
3  from terrainlib.readers.image import PILImageReader
4
5  generator = DiamondSquareGenerator(10, 0.1)
6  erosion = ThermalErosionFilter(150)
7  img_reader = PILImageReader(PILImageReader.BITDEPTH_16)
8
9  terrain = erosion(generator())
10 img = img_reader(terrain)
11 img.save('terrain_out.png')
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search