pyfastnoisesimd Documentation

Release 0.2.2

Robert A. McLeod

Dec 30, 2018

Contents:

1 Overview of PyFastNoiseSIMD **1**
- 1.1 Installation . . . 1
- 1.2 Benchmarks . . . 2

2 Tutorial on *pyfastnoisesimd* usage **3**
- 2.1 Workflow . . . 3
- 2.2 Example with source . . . 3

3 PyFastNoiseSIMD Reference **7**

4 PyFastNoiseSIMD C-Extension **13**

5 Indices and tables **15**

Python Module Index **17**

CHAPTER 1

Overview of PyFastNoiseSIMD

PyFastNoiseSIMD (*pyfastnoisesimd*) is a Python wrapper around Jordan Peck's *FastNoiseSIMD* (https://github.com/Auburns/FastNoise-SIMD) synthetic noise generation library, *FastNoiseSIMD*.

pyfastnoisesimd can generate noise in a 1-3D grid, via the *Noise.genAsGrid()* or the user can provide arbitrary coordinates in 3D Cartesian space with *Noise.genFromCoords()*

FastNoiseSIMD is also extremely fast due to its use of advanced x64 SIMD vectorized instruction sets, including SSE4.1, AVX2, and AVX512, depending on your CPU capabilities and the compiler used.

Parallelism in *pyfastnoisesimd* is further enhanced by the use of `concurrent.futures` to multi-thread the generation of noise for large arrays. Thread scaling is generally in the range of 50-90 %, depending largely on the vectorized instruction set used. The number of threads, defaults to the number of virtual cores on the system. The ideal number of threads is typically the number of physical cores, irrespective of Intel Hyperthreading®.

1.1 Installation

pyfastnoisesimd is available on PyPI, and may be installed via *pip*:

```
pip install --upgrade pip
pip install --upgrade setuptools
pip install -v pyfastnoisesimd
```

On Windows, a wheel is provided for Python 3.6 only. Building from source or compiling the extension for 3.5 will require either MS Visual Studio 2015 or MSVC2015 Build Tools:

http://landinghub.visualstudio.com/visual-cpp-build-tools

No Python versions compile with MSVC2017 yet, which is the newest version to support AVX512. Only Python 3.5/3.6 support AVX2 on Windows.

On Linux or OSX, only a source distribution is provided and installation requires *gcc* or *clang*. For AVX512 support with GCC, GCC7.2+ is required, lower versions will compile with AVX2/SSE4.1/SSE2 support only. GCC earlier than 4.7 disables AVX2 as well. Note that *pip* does not respect the *$CC* environment variable, so to clone and build from source with *gcc-7*:

git clone https://github.com/robbmcleod/pyfastnoisesimd.git alias gcc=gcc-7; alias g++=g++-7 pip install -v ./pyfastnoisesimd

Installing GCC7.2 on Ubuntu (with *sudo* or as root):

```
add-apt-repository ppa:ubuntu-toolchain-r/test
apt update
apt install gcc-7 g++-7
```

1.2 Benchmarks

The combination of the optimized, SIMD-instruction level C library, and multi-threading, means that *pyfastnoisesimd* is very, very fast. Generally speaking thread scaling is higher on machines with SSE4 support only, as most CPUs throttle clock speed down to limit heat generation with AVX2. As such, AVX2 is only about 1.5x faster than SSE4 whereas on a pure SIMD instruction length basis (4 versus 8) you would expect it to be x2 faster.

1.2.1 Configuration

- **CPU**: Intel i7-7820X Skylake-X (8 cores, 3.6 GHz), Windows 7
- **SIMD level supported**: AVX2 & FMA3

1.2.2 With `Noise.genAsGrid()`

The first test is used the default mode, a cubic grid, `Noise.genAsGrid()`, from `examples\gridded_noise.py`:

- **Array shape**: [8,1024,1024]

Single-threaded mode

Computed 8388608 voxels cellular noise in 0.298 s 35.5 ns/voxel

Computed 8388608 voxels Perlin noise in 0.054 s 6.4 ns/voxel

Multi-threaded (8 threads) mode

Computed 8388608 voxels cellular noise in 0.044 s 5.2 ns/voxel 685.0 % thread scaling

Computed 8388608 voxels Perlin noise in 0.013 s 1.5 ns/voxel 431.3 % thread scaling

1.2.3 With `Noise.getFromCoords()`

The alternative mode is `Noise.getFromCoords()` where the user provides the coordinates in Cartesian-space, from `examples\GallPeters_projection.py: - noiseType = Simplex - peturbType = GradientFractal`

Single-threaded mode Generated noise from 2666000 coordinates with 1 workers in 1.766e-02 s

6.6 ns/pixel

Multi-threaded (4 threads) mode Generated noise from 2666000 coordinates with 4 workers in 6.161e-03 s

2.3 ns/pixel 286.6 % thread scaling

CHAPTER 2

Tutorial on *pyfastnoisesimd* usage

2.1 Workflow

The basic workflow of using *pyfastnoiseimd* (which we will refer to as *fns*) is:

1. Instantiate a noise object as:

```
noiseObj = fns.Noise()
```

2. Set the desired properties of the *noiseObj* using some recipe of values. The precise recipe used is where noise generation morphs from mathematics to an artistic endeavor. One big advantage of using Python in this instance is that it becomes easy to quickly prototype many different styles of noise and adjust parameters to taste.

3. Use either *fns.Noise.genAsCoords()* to generate 1-3D rectilinear noise or *fns.Noise.genFromCoords()* to generate noise at user-generated coordinates. The *fns.Noise* class is a mapping of Python properties to the *set<...>* functions of the underlying *FastNoiseSIMD* library. Please see the API reference for the properties that may be set.

   The *fns.Noise* class contains three sub-objects:

   - *Noise.cell*: contains properties related to cellular (cubic Voronoi) noise.
   - *Noise.perturb*: contains properties related to perturbation of noise, typically related to applying gradients to noise.
   - *Noise.fractal*: contains properties related fractal noise. Fractals layer noise of similar properties but log-scaled frequencies (octaves).

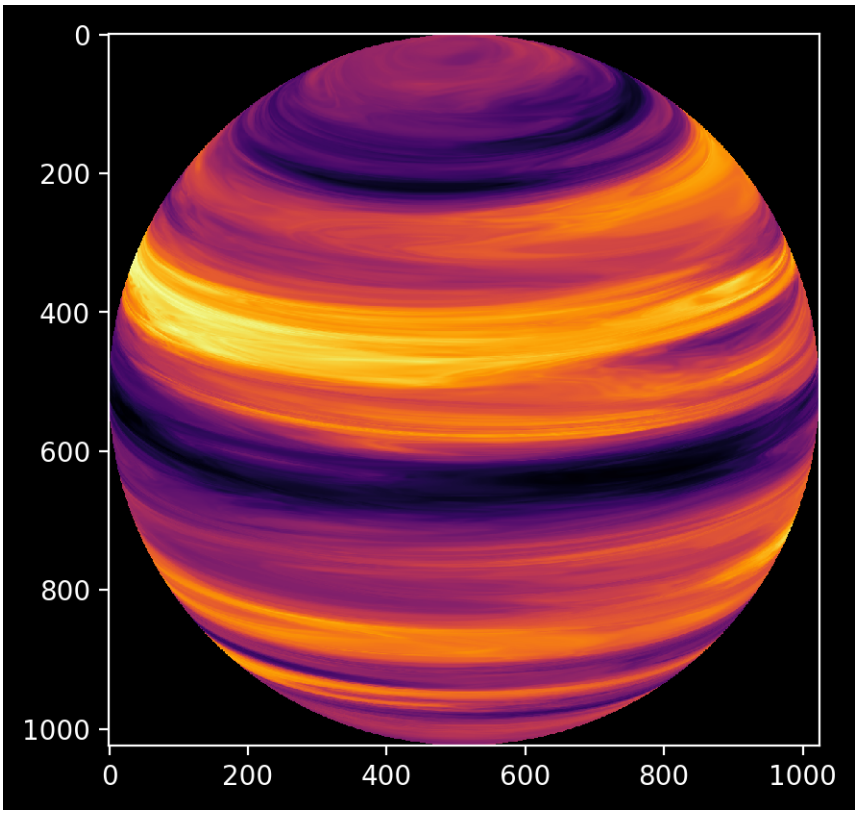
2.2 Example with source

In this case, we want to simulate a height-map on the surface of a sphere, and project coordinates from a 2D orthographic projection (i.e. a sprite) to the 3D Cartesian coordinates that *FastNoiseSIMD* requires, generate the noise, and then return back to the coordinates of our computer monitor.

The mathematics for the projection can be found at:

https://en.wikipedia.org/wiki/Orthographic_projection_in_cartography

Here is an example output using a recipe



Here is the complete source code:

```
import numpy as np
from time import perf_counter
import pyfastnoisesimd as fns

import matplotlib.pyplot as plt
plt.style.use('dark_background')
from mpl_toolkits.mplot3d import Axes3D

def orthoProject(noise:fns.Noise, tile2: int=512, p0: float=0., l0: float=0.) -> np.
↪ndarray:
    '''
    Render noise onto a spherical surface with an Orthographic projection.

    Args:
        noise: a `pyfastnoisesimd.Noise` object.
        tile2: the half-width of the returned array, i.e. return will have shape␣
↪``(2*tile2, 2*tile2)``.
        p0: the central parallel (i.e. latitude)
        l0: the central meridian (i.e. longitude)

    See also:

        https://en.wikipedia.org/wiki/Orthographic_projection_in_cartography
```

(continues on next page)

(continued from previous page)

```
    '''
    # We use angular coordinates as there's a lot of trig identities and we
    # can make some simplifications this way.
    xVect = np.linspace(-0.5*np.pi, 0.5*np.pi, 2*tile2, endpoint=True).astype('float32
↪')
    xMesh, yMesh = np.meshgrid(xVect, xVect)
    p0 = np.float32(p0)
    l0 = np.float32(l0)

    # Our edges are a little sharp, one could make an edge filter from the mask
    # mask = xMesh*xMesh + yMesh*yMesh <= 0.25*np.pi*np.pi
    # plt.figure()
    # plt.imshow(mask)
    # plt.title('Mask')

    # Check an array of coordinates that are inside the disk-mask
    valids = np.argwhere(xMesh*xMesh + yMesh*yMesh <= 0.25*np.pi*np.pi)
    xMasked = xMesh[valids[:,0], valids[:,1]]
    yMasked = yMesh[valids[:,0], valids[:,1]]
    maskLen = xMasked.size

    # These are simplified equations from the linked Wikipedia article.  We
    # have to back project from 2D map coordinates [Y,X] to Cartesian 3D
    # noise coordinates [W,V,U]
    # TODO: one could accelerate these calculations with `numexpr` or `numba`
    one = np.float32(0.25*np.pi*np.pi)
    rhoStar = np.sqrt(one - xMasked*xMasked - yMasked*yMasked)
    muStar =  yMasked*np.cos(p0) + rhoStar*np.sin(p0)
    conjMuStar = rhoStar*np.cos(p0) - yMasked*np.sin(p0)
    alphaStar = l0 + np.arctan2(xMasked, conjMuStar)
    sqrtM1MuStar2 = np.sqrt(one - muStar*muStar)

    # Ask fastnoisesimd for a properly-shaped array
    # Failure to use a coords array that is evenly divisible by the
    # SIMD vector length can cause a general protection fault.
    coords = fns.empty_coords(maskLen)
    coords[0,:maskLen] = muStar                              # W
    coords[1,:maskLen] = sqrtM1MuStar2 * np.sin(alphaStar)   # V
    coords[2,:maskLen] = sqrtM1MuStar2 * np.cos(alphaStar)   # U

    # Check our coordinates in 3-D to make sure the shape is correct:
    # fig = plt.figure()
    # ax = fig.add_subplot(111, projection='3d')
    # ax.scatter( coords[2,:maskLen], coords[1,:maskLen], coords[0,:maskLen], 'k.' )
    # ax.set_xlabel('U')
    # ax.set_ylabel('V')
    # ax.set_zlabel('W')
    # ax.set_title('3D coordinate sampling')

    pmap = np.full( (2*tile2, 2*tile2), -np.inf, dtype='float32')
    pmap[valids[:,0], valids[:,1]] = noise.genFromCoords(coords)[:maskLen]
    return pmap

# Let's set the view-parallel so we can see the top of the sphere
p0 = np.pi-0.3
# the view-meridian isn't so important, but if you wanted to rotate the
```

(continues on next page)

(continued from previous page)

```
# view, this is how you do it.
l0 = 0.0

# Now create a Noise object and populate it with intelligent values. How to
# come up with 'intelligent' values is left as an exercise for the reader.
gasy = fns.Noise()
gasy.frequency = 1.8
gasy.axesScales = (1.0,0.06,0.06)

gasy.fractal.octaves = 5
gasy.fractal.lacunarity = 1.0
gasy.fractal.gain = 0.33

gasy.perturb.perturbType = fns.PerturbType.GradientFractal
gasy.perturb.amp = 0.5
gasy.perturb.frequency = 1.2
gasy.perturb.octaves = 5
gasy.perturb.lacunarity = 2.5
gasy.perturb.gain = 0.5

gasy_map = orthoProject(gasy, tile2=512, p0=p0, l0=l0)

fig = plt.figure()
fig.patch.set_facecolor('black')
plt.imshow(gasy_map, cmap='inferno')
# plt.savefig('gasy_map.png', bbox_inches='tight', dpi=200)
plt.show()
```

CHAPTER 3

PyFastNoiseSIMD Reference

class pyfastnoisesimd.**CellularDistanceFunction**
Enum: The distance function for cellular noise.

Values: `{Euclidean, Manhattan, Natural}`

Euclidean = 0

Manhattan = 1

Natural = 2

class pyfastnoisesimd.**CellularReturnType**
Enum: The functional filter to apply to the distance function to generate the return from cellular noise.

Values: `{CellValue, Distance, Distance2, Distance2Add, Distance2Sub, Distance2Mul, Distance2Div, NoiseLookup, Distance2Cave}`

CellValue = 0

Distance = 1

Distance2 = 2

Distance2Add = 3

Distance2Cave = 8

Distance2Div = 6

Distance2Mul = 5

Distance2Sub = 4

NoiseLookup = 7

class pyfastnoisesimd.**FractalType**
Enum: Fractal noise types also have an additional fractal type.

Values: `{FBM, Billow, RigidMulti}`

Billow = 1

FBM = 0

RigidMulti = 2

class pyfastnoisesimd.**Noise**(*seed: int = None*, *numWorkers: int = None*)

Noise encapsulates the C++ SIMD class FNSObject and enables get/set of all relative properties via Python properties.

Parameters

- **seed** – The random number (int32) that seeds the random-number generator If seed == None a random integer is generated as the seed.
- **numWorkers** – The number of threads used for parallel noise generation. If numWorkers == None, the default applied by *concurrent.futures.ThreadPoolExecutor* is used.

axesScales

Sets the FastNoiseSIMD axes scales, which allows for non-square voxels. Indirectly affects *frequency* by changing the voxel pitch.

Default: (1.0, 1.0, 1.0)

frequency

The frequency of the noise, lower values result in larger noise features.

Default: 0.01

genAsGrid(*shape=[1, 1024, 1024]*, *start=[0, 0, 0]*) → numpy.ndarray

Generates noise according to the set properties along a rectilinear (evenly-spaced) grid.

Parameters

- **shape** – Tuple[int] the shape of the output noise volume.
- **start** – Tuple[int] the starting coordinates for generation of the grid. I.e. the coordinates are essentially *start: start + shape*

Example:

```
import numpy as np
import pyfastnoisesimd as fns
noise = fns.Noise()
result = noise.genFromGrid(shape=[256,256,256], start=[0,0,0])
nextResult = noise.genFromGrid(shape=[256,256,256], start=[256,0,0])
```

genFromCoords(*coords: numpy.ndarray*) → numpy.ndarray

Generate noise from supplied coordinates, rather than a rectilinear grid. Useful for complicated shapes, such as tesselated surfaces.

Parameters **coords** – 3-D coords as generated by fns.empty_coords() and filled with relevant values by the user.

Returns a shape (N,) array of the generated noise values.

Return type noise

Example:

```
import numpy as np
import pyfastnoisesimd as fns
numCoords = 256
coords = fns.empty_coords(3,numCoords)
# <Set the coordinate values, it is a (3, numCoords) array
```

(continues on next page)

(continued from previous page)

```
coords[0,:] = np.linspace(-np.pi, np.pi, numCoords)
coords[1,:] = np.linspace(-1.0, 1.0, numCoords)
coords[2,:] = np.zeros(numCoords)
noise = fns.Noise()
result = noise.genFromCoords(coords)
```

noiseType
The class of noise.

Default: `NoiseType.Simplex`

numWorkers
Sets the maximum number of thread workers that will be used for generating noise. Generally should be the number of physical CPU cores on the machine.

Default: Number of virtual cores on machine.

seed
The random-number seed used for generation of noise.

Default: `numpy.random.randint()`

class pyfastnoisesimd.**NoiseType**
The class of noise generated.

Enums: {Value, ValueFractal, Perlin, PerlinFractal, Simplex, SimplexFractal, WhiteNoise, Cellular, Cubic, CubicFractal}

Cellular = 7

Cubic = 8

CubicFractal = 9

Perlin = 2

PerlinFractal = 3

Simplex = 4

SimplexFractal = 5

Value = 0

ValueFractal = 1

WhiteNoise = 6

class pyfastnoisesimd.**PerturbType**
Enum: The enumerator for the class of Perturbation.

Values: {NoPeturb, Gradient, GradientFractal, Normalise, Gradient_Normalise, GradientFractal_Normalise}

Gradient = 1

GradientFractal = 2

GradientFractal_Normalise = 5

Gradient_Normalise = 4

NoPerturb = 0

Normalise = 3

pyfastnoisesimd.**aligned_chunks**(*array*, *n_chunks*, *axis=0*)
An generator that divides an array into chunks that have memory addresses compatible with SIMD vector length.

Parameters

- **array** – numpy.ndarray the array to chunk
- **n_chunks** – int the desired number of chunks, the returned number _may_ be less.
- **axis** – int the axis to chunk on, similar to *numpy* axis commanes.

Returns chunk: numpy.ndarray start: Tuple[int]

the start indices of the chunk, in the *array*.

pyfastnoisesimd.**check_alignment**(*array*)
Verifies that an array is aligned correctly for the supported SIMD level.

Parameters **array** – numpy.ndarray

Returns bool

Return type truth

pyfastnoisesimd.**empty_aligned**(*shape*, *dtype=<class 'numpy.float32'>*, *n_byte=32*)
Provides an memory-aligned array for use with SIMD accelerated instructions. Should be used to build

Adapted from: https://github.com/hgomersall/pyFFTW/blob/master/pyfftw/utils.pxi

Parameters

- **shape** – a sequence (typically a tuple) of array axes.
- **dtype** – NumPy data type of the underlying array. Note FastNoiseSIMD supports only *np.float32*. Seg faults may occur if this is changed.
- **n_byte** – byte alignment. Should always use the *pyfastnoisesimd.extension.SIMD_ALIGNMENT* value or seg faults may occur.

pyfastnoisesimd.**empty_coords**(*length*, *dtype=<class 'numpy.float32'>*, *n_byte=32*)

pyfastnoisesimd.**full_aligned**(*shape*, *fill*, *dtype=<class 'numpy.float32'>*, *n_byte=32*)
As per *empty_aligned*, but returns an array initialized to a constant value.

Parameters

- **shape** – a sequence (typically a tuple) of array axes.
- **fill** – the value to fill each array element with.
- **dtype** – NumPy data type of the underlying array. Note FastNoiseSIMD supports only *np.float32*. Seg faults may occur if this is changed.
- **n_byte** – byte alignment. Should always use the *pyfastnoisesimd.extension.SIMD_ALIGNMENT* value or seg faults may occur.

pyfastnoisesimd.**num_virtual_cores**()
Detects the number of virtual cores on a system without importing `multiprocessing`. Borrowed from NumExpr 2.6.

pyfastnoisesimd.**test**(*verbosity=2*)
Run `unittest` suite for `pyfastnoisesimd` package.

class pyfastnoisesimd.helpers.**FractalClass**(*fns*)
Holds properties related to noise types that include fractal octaves.

Do not instantiate this class separately from *Noise*.

fractalType
The type of fractal for fractal NoiseTypes.

Default: `FractalType.FBM`

gain
Octave gain for all fractal noise types. Reflects the ratio of the underlying noise to that of the fractal. Values > 0.5 up-weight the fractal.

Default: `0.5`

lacunarity
Octave lacunarity for all fractal noise types.

Default: `2.0`

octaves
Octave count for all fractal noise types, i.e. the number of log-scaled frequency levels of noise to apply. Generally `3` is sufficient for small textures/sprites (256x256 pixels), use larger values for larger textures/sprites.

Default: `3`

class pyfastnoisesimd.helpers.**CellularClass**(*fns*)
Holds properties related to *NoiseType.Cellular*.

Do not instantiate this class separately from `Noise`.

distanceFunc

distanceIndices
Sets the two distance indices used for `distance2X` return types Default: `(0, 1)`

jitter
The maximum distance a cellular point can move from it's grid position. The value is relative to the cubic cell spacing of `1.0`. Setting `jitter > 0.5` can generate wrapping artifacts.

Default: `0.45`

lookupFrequency
Relative frequency on the cellular noise lookup return type.

Default: `0.2`

noiseLookupType
Sets the type of noise used if cellular return type.

Default: *NoiseType.Simplex*

returnType
The return type for cellular (cubic Voronoi) noise.

Default: `CellularReturnType.Distance`

class pyfastnoisesimd.helpers.**PerturbClass**(*fns*)
Holds properties related to the perturbation applied to noise.

Do not instantiate this class separately from `Noise`.

amp
The maximum distance the input position can be perturbed. The reasonable values of `amp` before artifacts are apparent increase with decreased `frequency`. The default value of `1.0` is quite high.

Default: `1.0`

frequency
The relative frequency for the perturbation gradient.

Default: `0.5`

gain
The octave gain for fractal perturbation types. Reflects the ratio of the underlying noise to that of the fractal. Values > 0.5 up-weight the fractal.

Default: `0.5`

lacunarity
The octave lacunarity (gap-fill) for fractal perturbation types. Lacunarity increases the fineness of fractals. The appearance of graininess in fractal noise occurs when lacunarity is too high for the given frequency.

Default: `2.0`

normaliseLength
The length for vectors after perturb normalising

Default: `1.0`

octaves
The octave count for fractal perturbation types, i.e. the number of log-scaled frequency levels of noise to apply. Generally `3` is sufficient for small textures/sprites (256x256 pixels), use larger values for larger textures/sprites.

Default: `3`

perturbType
The class of perturbation.

Default: `PerturbType.NoPeturb`

CHAPTER 4

PyFastNoiseSIMD C-Extension

pyfastnoisesimd.extension.**AlignedSize**()
AlignedSize(int size) – Rounds the size up to the nearest aligned size for the current SIMD level.

class pyfastnoisesimd.extension.**FNS**
FastNoiseSIMD factory

FillNoiseSet()
FillNoiseSet(float* noiseSet, int xStart, int yStart, int zStart, int xSize, int ySize, int zSize, float scaleModifier)– Fill a noise set.

GetSeed()
int GetSeed() – Returns seed used for all noise types.

NoiseFromCoords()
NoiseFromCoords(numpy.ndarray noise, numpy.ndarray coords)– Fill a noise set from arbitrary coordinates. Must be a shape (3,N) array of dtype 'float32'.

SetAxesScales()
SetAxesScales(float zScale, float yScale, float xScale) – Sets scaling factor for individual axis. Defaults: 1.0.

SetCellularDistance2Indices()
SetCellularDistance2Indices(int cellularDistanceIndex0, int cellularDistanceIndex1) – Sets the 2 distance indicies used for distance2 return types. Default: 0, 1. Note: index0 should be lower than index1, index1 must be < 4.

SetCellularDistanceFunction()
SetCellularDistanceFunction(int cellularDistanceFunction) – Sets distance function used in cellular noise calculations. Default: Euclidean. Use the dict _ext.cellularDistanceFunction to convert names to enums.

SetCellularJitter()
SetCellularJitter(float cellularJitter) – Sets relative frequency on the cellular noise lookup return type. Default: 0.2

SetCellularNoiseLookupFrequency()
SetCellularNoiseLookupFrequency(float cellularNoiseLookupFrequency) – Sets relative frequency on the cellular noise lookup return type. Default: 0.2

SetCellularNoiseLookupType()
SetCellularNoiseLookupType(int cellularNoiseLookupType)– Sets the type of noise used if cellular return type is set to NoiseLookup. Default: Simplex. Use the dict _ext.noiseType to convert names to enums.

SetCellularReturnType()
SetCellularReturnType(int cellularReturnType) – Sets return type from cellular noise calculations. Default: Distance. Use the dict _ext.cellularReturnType to convert names to enums.

SetFractalGain()
SetFractalGain(float gain) – Sets octave gain for all fractal noise types. Default: 0.5.

SetFractalLacunarity()
SetFractalLacunarity(float lacunarity) – Sets octave lacunarity for all fractal noise types. Default: 2.0.

SetFractalOctaves()
SetFractalOctaves(int octaves) – Sets octave count for all fractal noise types. Default: 3.

SetFractalType()
SetFractalType(int fractalType) – Sets method for combining octaves in all fractal noise types. Default: FBM. Use the dict _ext.fractalType to convert names to enums.

SetFrequency()
SetFrequency(float frequency) – Sets frequency for all noise types. Default: 0.01

SetNoiseType()
SetNoiseType(NoiseType noiseType) – Sets noise return type of (Get/Fill)NoiseSet(). Default: Simplex. Use the dict _ext.noiseType to convert names to enums.

SetPerturbAmp()
SetPerturbAmp(float perturbAmp) – Sets the maximum distance the input position can be perturbed. Default: 1.0.

SetPerturbFractalGain()
SetPerturbFractalGain(float perturbGain) – Sets octave gain for perturb fractal types. Default: 0.5.

SetPerturbFractalLacunarity()
SetPerturbFractalLacunarity(float perturbLacunarity) – Sets octave lacunarity for perturb fractal types. Default: 2.0.

SetPerturbFractalOctaves()
SetPerturbFractalOctaves(int perturbOctaves)– Sets octave count for perturb fractal types. Default: 3.

SetPerturbFrequency()
SetPerturbFrequency(float perturbFrequency) – Set the relative frequency for the perturb gradient. Default: 0.5.

SetPerturbNormaliseLength()
SetPerturbNormaliseLength(float perturbGain) – Sets the length for vectors after perturb normalising . Default: 1.0.

SetPerturbType()
SetPerturbType(int perturbType) – Enables position perturbing for all noise types. Default: None. Use the dict _ext.perturbType to convert names to enums.

SetSeed()
SetSeed(int seed) – Sets seed used for all noise types. Default is 42.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

pyfastnoisesimd, 7
pyfastnoisesimd.extension, 13

Index

A

aligned_chunks() (in module pyfastnoisesimd), 9
AlignedSize() (in module pyfastnoisesimd.extension), 13
amp (pyfastnoisesimd.helpers.PerturbClass attribute), 11
axesScales (pyfastnoisesimd.Noise attribute), 8

B

Billow (pyfastnoisesimd.FractalType attribute), 7

C

Cellular (pyfastnoisesimd.NoiseType attribute), 9
CellularClass (class in pyfastnoisesimd.helpers), 11
CellularDistanceFunction (class in pyfastnoisesimd), 7
CellularReturnType (class in pyfastnoisesimd), 7
CellValue (pyfastnoisesimd.CellularReturnType attribute), 7
check_alignment() (in module pyfastnoisesimd), 10
Cubic (pyfastnoisesimd.NoiseType attribute), 9
CubicFractal (pyfastnoisesimd.NoiseType attribute), 9

D

Distance (pyfastnoisesimd.CellularReturnType attribute), 7
Distance2 (pyfastnoisesimd.CellularReturnType attribute), 7
Distance2Add (pyfastnoisesimd.CellularReturnType attribute), 7
Distance2Cave (pyfastnoisesimd.CellularReturnType attribute), 7
Distance2Div (pyfastnoisesimd.CellularReturnType attribute), 7
Distance2Mul (pyfastnoisesimd.CellularReturnType attribute), 7
Distance2Sub (pyfastnoisesimd.CellularReturnType attribute), 7
distanceFunc (pyfastnoisesimd.helpers.CellularClass attribute), 11
distanceIndices (pyfastnoisesimd.helpers.CellularClass attribute), 11

E

empty_aligned() (in module pyfastnoisesimd), 10
empty_coords() (in module pyfastnoisesimd), 10
Euclidean (pyfastnoisesimd.CellularDistanceFunction attribute), 7

F

FBM (pyfastnoisesimd.FractalType attribute), 7
FillNoiseSet() (pyfastnoisesimd.extension.FNS method), 13
FNS (class in pyfastnoisesimd.extension), 13
FractalClass (class in pyfastnoisesimd.helpers), 10
FractalType (class in pyfastnoisesimd), 7
fractalType (pyfastnoisesimd.helpers.FractalClass attribute), 11
frequency (pyfastnoisesimd.helpers.PerturbClass attribute), 11
frequency (pyfastnoisesimd.Noise attribute), 8
full_aligned() (in module pyfastnoisesimd), 10

G

gain (pyfastnoisesimd.helpers.FractalClass attribute), 11
gain (pyfastnoisesimd.helpers.PerturbClass attribute), 12
genAsGrid() (pyfastnoisesimd.Noise method), 8
genFromCoords() (pyfastnoisesimd.Noise method), 8
GetSeed() (pyfastnoisesimd.extension.FNS method), 13
Gradient (pyfastnoisesimd.PerturbType attribute), 9
Gradient_Normalise (pyfastnoisesimd.PerturbType attribute), 9
GradientFractal (pyfastnoisesimd.PerturbType attribute), 9
GradientFractal_Normalise (pyfastnoisesimd.PerturbType attribute), 9

J

jitter (pyfastnoisesimd.helpers.CellularClass attribute), 11

L

lacunarity (pyfastnoisesimd.helpers.FractalClass attribute), 11

lacunarity (pyfastnoisesimd.helpers.PerturbClass attribute), 12
lookupFrequency (pyfastnoisesimd.helpers.CellularClass attribute), 11

M

Manhattan (pyfastnoisesimd.CellularDistanceFunction attribute), 7

N

Natural (pyfastnoisesimd.CellularDistanceFunction attribute), 7
Noise (class in pyfastnoisesimd), 8
NoiseFromCoords() (pyfastnoisesimd.extension.FNS method), 13
NoiseLookup (pyfastnoisesimd.CellularReturnType attribute), 7
noiseLookupType (pyfastnoisesimd.helpers.CellularClass attribute), 11
NoiseType (class in pyfastnoisesimd), 9
noiseType (pyfastnoisesimd.Noise attribute), 9
NoPerturb (pyfastnoisesimd.PerturbType attribute), 9
Normalise (pyfastnoisesimd.PerturbType attribute), 9
normaliseLength (pyfastnoisesimd.helpers.PerturbClass attribute), 12
num_virtual_cores() (in module pyfastnoisesimd), 10
numWorkers (pyfastnoisesimd.Noise attribute), 9

O

octaves (pyfastnoisesimd.helpers.FractalClass attribute), 11
octaves (pyfastnoisesimd.helpers.PerturbClass attribute), 12

P

Perlin (pyfastnoisesimd.NoiseType attribute), 9
PerlinFractal (pyfastnoisesimd.NoiseType attribute), 9
PerturbClass (class in pyfastnoisesimd.helpers), 11
PerturbType (class in pyfastnoisesimd), 9
perturbType (pyfastnoisesimd.helpers.PerturbClass attribute), 12
pyfastnoisesimd (module), 7
pyfastnoisesimd.extension (module), 13

R

returnType (pyfastnoisesimd.helpers.CellularClass attribute), 11
RigidMulti (pyfastnoisesimd.FractalType attribute), 8

S

seed (pyfastnoisesimd.Noise attribute), 9
SetAxesScales() (pyfastnoisesimd.extension.FNS method), 13
SetCellularDistance2Indices() (pyfastnoisesimd.extension.FNS method), 13
SetCellularDistanceFunction() (pyfastnoisesimd.extension.FNS method), 13
SetCellularJitter() (pyfastnoisesimd.extension.FNS method), 13
SetCellularNoiseLookupFrequency() (pyfastnoisesimd.extension.FNS method), 13
SetCellularNoiseLookupType() (pyfastnoisesimd.extension.FNS method), 14
SetCellularReturnType() (pyfastnoisesimd.extension.FNS method), 14
SetFractalGain() (pyfastnoisesimd.extension.FNS method), 14
SetFractalLacunarity() (pyfastnoisesimd.extension.FNS method), 14
SetFractalOctaves() (pyfastnoisesimd.extension.FNS method), 14
SetFractalType() (pyfastnoisesimd.extension.FNS method), 14
SetFrequency() (pyfastnoisesimd.extension.FNS method), 14
SetNoiseType() (pyfastnoisesimd.extension.FNS method), 14
SetPerturbAmp() (pyfastnoisesimd.extension.FNS method), 14
SetPerturbFractalGain() (pyfastnoisesimd.extension.FNS method), 14
SetPerturbFractalLacunarity() (pyfastnoisesimd.extension.FNS method), 14
SetPerturbFractalOctaves() (pyfastnoisesimd.extension.FNS method), 14
SetPerturbFrequency() (pyfastnoisesimd.extension.FNS method), 14
SetPerturbNormaliseLength() (pyfastnoisesimd.extension.FNS method), 14
SetPerturbType() (pyfastnoisesimd.extension.FNS method), 14
SetSeed() (pyfastnoisesimd.extension.FNS method), 14
Simplex (pyfastnoisesimd.NoiseType attribute), 9
SimplexFractal (pyfastnoisesimd.NoiseType attribute), 9

T

test() (in module pyfastnoisesimd), 10

V

Value (pyfastnoisesimd.NoiseType attribute), 9
ValueFractal (pyfastnoisesimd.NoiseType attribute), 9

W

WhiteNoise (pyfastnoisesimd.NoiseType attribute), 9