Isaacs Noise Library

The purpose of this system is to create 2D noise in the form of an array of pixels. To utilise the program enter the create the type of noise you want (IsaacsNoiseLibrary::PerlinNoise perlinNoise(width, height)) for example then reference the pixels via perlinNoise.pixels, this variable exists for all noise classes. Then the pixels contain a Vector2 and a Colour (0-255 range, RGB)

Although no additional libraries are required to run the system, I am using Easy BMP to save the array to an image for the purposes of visualization.

The system will be distributed as 2 .lib files, one for release configuration and one for debug configuration and a header file.

The system utilises the following mathematical operations

Linear interpolation

* minValue + transformValue \* (maxValue - minValue)

and a fade function to get a smooth transition for the linear interpolation

* ((6 \* a - 15) \* a + 10) \* a \* a \* a

where a is the transform value for linear interpolation

The algorithms used in the system are:

* Perlin Noise

A type of gradient noise commonly used for texture generation and height maps

* Fractal Brownian Motion

Perlin Noise layered multiple times to create a sharper image, good for land mass generation

* Cellular Noise

Noise that resembles a clump of cells, used for texture generation.

* Voronoi Tessellation

Noise where a pixel is coloured based on its nearest point

* Poisson Disc Distribution

Creates an even non lattice distribution of points

To integrate the system into an application, first download the distributable package and move the files into your applications directory. Next open the applications properties and add [Filepath]IsaacsNoiseLibrary.lib, or IsaacsNoiseLibraryd.lib for debug configuration, to the following fields. VC++ Directories -> Additional Directories; C/C++ -> Additional Include Directories; Linker -> Additional Library Directories. After that add Noisy.h to the project and include it in the application. The system should now be ready to use.

When generating noise larger than 1000 x 1000 the performance drops, especially where Voronoi noise and fractal noise are concerned, in spite of this the system still runs as intended. Future updates to the system will include, Delaunay Triangulation and performance boosts through profiling and potentially multithreading so larger noise can be generated. Also noise in more dimensions is a possibility in the future as well as more customisation, array size for Voronoi and cellular noise for example, or change details about Poisson Disc distribution and the ability to generate a distribution independent of Voronoi Tessellation.