Procedutex

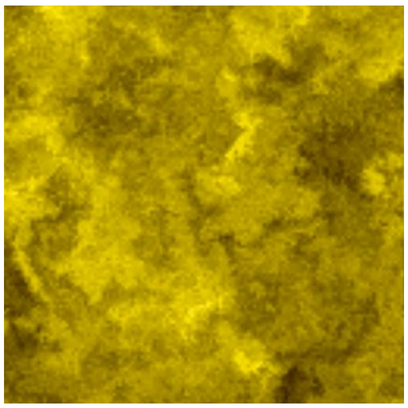
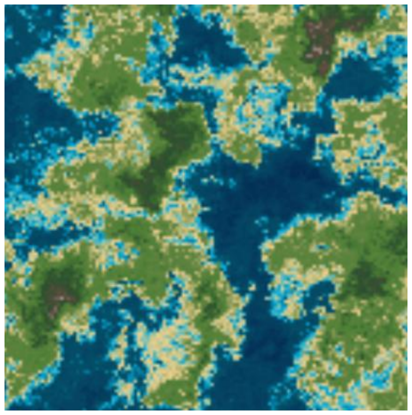
A 2D Procedural Texture Generation Library

Jibran Syed

Introduction

Procedutex is a library that facilitates the use of procedural noise in order to make random textures. The noise generation components can actually be used without the need to make textures, but easy texture generation was a major goal for this library.

Another goal of the library was to wrap the mathematical and mundane complexities of procedural noise generation into objected-oriented "building blocks" that are easy to use, yet facilitate advanced noise generation techniques. The following images demonstrate some of the textures that can be possible made with this library:

Data Pipeline

In order to make a texture, the user of the Procedutex library first start with a TextureBuffer2d, which defines a power-of-2, rectangular grid of pixels on the CPU, in which the final color values will be written. These pixels could be written in directly, but Procedutex offers some facilities to streamline the process without the user needing to make his/her own loops.

The main procedural logic that will define the relative value of a pixel is controlled by a collection of objects called Generators. Generators take in a position in "sampling space", and output a normalized value. Perlin noise would be an example of a Generator.

Sampling space is an arbitrary real number space where a position can be extracted and processed by a Generator. The sampling space is sampled by the texture dimensions defined for the image buffer.

A “normalized value” is a value that exists in an arbitrary range that can be easily converted into any specific range. In this library, the arbitrary range is between -1 and +1, with 0 being the middle of this arbitrary range. These normalized values are outputted by a Generator and can be used in anyway the user desires, including defining a pixel.

But in order to define the pixel, a normalized value must first be converted into a color value. That is the purpose of another class of objects called Converters. Given a normalized value from a Generator, a Converter determines how to turn that value into a color, and thus defining one pixel on the texture buffer.

The texture buffer has a special method called Generate() that will fill the texture image with a procedural texture value, but it must be given a Generator and a Converter as parameters in order to work. From there, the data from the texture buffer can be loaded into the GPU by a custom graphics engine.

Generators also can have their input and output modified using objects called Modifiers. An object that takes in a position and outputs a position is called an Input Modifier, and an object that takes in a normalized value and outputs a normalized value is called an Output Modifier. Generators have a list of both types of Modifiers each, which it will use to alter the values. Modifiers are optional and are not needed for Generators to work.

Sometimes, the user will want to use a special kind of Generator called a Composite Generator that takes two Generators and merges the output values somehow. The way that the output values are merged together is defines by a class of objects known as Combiners, which are required when using a Composite Generator.

Classes

Generators

In order to make most Generators, the user will need to define a seed (for randomization), and if desired, a sample-space definition. To define sample space, the user provides an origin, width, and height (basically, a Rect). The origin is defined at the top left corner, in order to be consistent with the texture buffer orientation.

There are 3 major types of Generators: Single Generators, Fractal Generators, and Composite Generators.

A Single Generator is a general definition for any Generator that does *not* need another Generator in order to create a new normalized value. The most famous example of a Single Generator would be Perlin Noise, which in Procedutex, inherits from Single Generator.

Currently, in this version of Procedutex, the other Single Generator is called a Constant Generator, which outputs the same value regardless of the input position. This constant value is defined in the Constant Generator's constructor.

The Fractal Generator is a special Generator (not single) that can take in any Generator (including another Fractal Generator, if you have the computing power to do that), and will call this "base" Generator multiple times with different frequencies and amplitudes. Every call into this base Generator is called an "octave". In every octave, the amplitude of the noise decreases, while the frequency increases. In the end, these octaves are added together to get the final noise.

The Composite Generator is a Generator that can take in two Generators (including other Composite Generators) and combine the output values in the way defined by its Combiner. The two base Generators and the Combiner are defined in the constructor of a Composite Generator. Composite Generators allow for advanced procedural generation by creating a "tree" hierarchy of Generators, allowing more than two Generators to be combined.

Combiners

In the current version of Procedutex, there are 4 kinds of Combiners (for Composite Generators), and are rather simple in implementation: Add, Multiply, Min, and Max.

The Add Combiner tells the Composite Generator to add the two base outputs together. Any values exceeding the normalized range will be clamped.

The Multiply Combiner tells the Composite Generator to multiply the two base outputs together.

The Min Combiner tells the Composite Generator to find the minimum of the two base outputs. Pixel-by-pixel, the lesser value can vary due to the nature of procedural noise.

The Max Combiner tells the Composite Generator to find the maximum of the two base outputs. Pixel-by-pixel, the greater value can vary due to the nature of procedural noise.

Input Modifiers

If a Generator has any Input Modifiers in its internal list, it will modify the input position in the order of when the Modifiers were added to that Generator. Procedutex currently has 2: Translate and Domain Distortion.

Translate simple takes the input position and translates it by a delta defined in the Modifier's constructor.

Domain Distortion is a special Input Modifier because it requires its own Generator in order to modify the input position. Domain Distortion takes the original position and uses its own noise to create a “distortion vector”, which is basically a random translation vector, which is applied onto the original position before being passed into the actual Generator.

Output Modifiers

There are currently 5 Output Modifiers in Procedutex: Invert, Abs, Binarize, Gain, and Custom Function.

Invert simply takes the normalized output value and multiplies it by -1, thus “inverting” the value.

Abs takes the absolute value of the normalized output, since the output can be negative, making everything positive. It can be used alongside Invert to force the output to be all negative.

Binarize checks a user-defined threshold with the current output. If the output is less than the threshold, then the output becomes -1, otherwise, the output becomes +1.

Gain shifts positive values up by a user-defined amount, and shifts down negative values by that same amount. It is used to “sharpen” the noise value.

The last Output Modifier is arguably the most advanced. This Modifier will alter the output with a user-provided functional object that takes in 1 float and returns 1 float. This feature requires C++11 lambdas in order to provide convenient custom output modification.

Converters

Converts take the final output of all the used Generators combined, and converts that output into a color. Procedutex currently has 2 kinds: Grayscale Converter and Gradient Converter.

The Grayscale Converter is very simple. It takes the normalized output and converts it to any shade between black and white. -1 returns black, and +1 returns white. 0 returns 50% gray.

The Gradient Converter is a little more complicated, and much more powerful. First, this Converter needs a Gradient defined in its constructor. A Gradient a variable amount of colors located at normalized 1D “gradient space” coordinates. The Gradient Converter takes the final normalized output and looks for two gradient colors nearest to the output. From there, the two colors are linearly interpolated to a final color.