

AHNoise Documentation v1.1

This document outlines the use of the AHNoise procedural noise library for Apple platforms. Procedural noise is useful for generating textures resembling natural phenomena such as clouds or wood or to provide height maps for infinite worlds.

Copyright Andrew Heard 2016

Contents

Introduction.....	3
Library Overview	3
How to Use.....	4
Example	4
Class References	6
Protocols	6
Classes.....	6
AHNTextureProvider.....	9
AHNContext.....	11
AHNGenerator	12
AHNGeneratorCoherent.....	16
AHNGeneratorBillow	21
AHNGeneratorRidgedMulti	22
AHNGeneratorSimplex	23
AHNGeneratorVoronoi	24
AHNGeneratorChecker	25
AHNGeneratorConstant	27
AHNGeneratorCylinder	29
AHNGeneratorGradientBox	32
AHNGeneratorGradientLinear	34
AHNGeneratorGradientRadial	36
AHNGeneratorSphere	38
AHNGeneratorWave	41
AHNModifier	42
AHNModifierAbsolute.....	44
AHNModifierBlur	45
AHNModifierClamp	46
AHNModifierColour.....	48
AHNModifierInvert	50
AHNModifierLoop	51
AHNModifierPerspective	53
AHNModifierRotate.....	55
AHNModifierRound.....	57
AHNModifierScaleBias.....	58
AHNModifierStep.....	59
AHNModifierStretch.....	61
AHNModifierSwirl	63
AHNModifierMapNormal	65

AHNModifierScaleCanvas	67
AHNCombiner	69
AHNCombinerAdd	71
AHNCombinerDivide	72
AHNCombinerMax	73
AHNCombinerMin	74
AHNCombinerMultiply	75
AHNCombinerPower	76
AHNCombinerSubtract	77
AHNSelector	78
AHNSelectorBlend	80
AHNSelectorSelect	81
Custom Modules	83

Introduction

Cohesive, procedural noise can be used to add natural looking unevenness to a texture. This library provides the tools required to generate procedural noise and modify it all on the GPU using Metal and Swift. It is similar in functionality to Apple's own GameplayKit noise framework released in iOS 10, as both have taken inspiration from the libnoise framework that has not been supported for some time now. There are some discrepancies between this framework and GameplayKit, most notably the extremely different outputs from the respective "Billow" generators and the lack of curve remapping. The only other major difference in capability is in the speed that a noise texture is generated. The GameplayKit noise generation can take a few seconds for a simple Perlin/Simplex texture at 512x512px, whereas this framework can calculate multiple textures a second as demonstrated in the Noise Studio App, built to demonstrate and act as a tool for using this noise library.

Library Overview

Most of the classes in AHNNoise will output a texture that can either be used as the input to another class, or converted into a UIImage for use. The only class that does not output a texture is AHNContext, which is a wrapper for the Metal classes used to communicate with the GPU. All other classes will output a texture by calling the texture() function. Most of the other classes will also allow an input texture to modify in some way. It is only AHNGenerator subclasses as well as the AHNContext class that have no input. AHNModifiers have one texture input and one texture output, AHNCombiners have two texture inputs and one texture output and AHNSelectors have three texture inputs and one texture output. Each subclass has various tools and properties to modify behaviour too.

All classes in this module are prefixed with "AHN". This is then split into categories such as AHNGenerator, AHNModifier, AHNCombiner and AHNSelector. These are superclasses designed to be subclassed into usable modules such as AHNGeneratorSimplex or AHNCombinerMultiply etc. The idea being that by starting to type "AHN" you are presented with a list of all classes which can then be narrowed down as you continue to type.

All texture generation and modification is carried out on the GPU. A module will populate its texture() property which can then be handed to another module which will then use it in its own GPU based calculations. This provides a very simple modular approach, though the downside is that when chaining together a large number of modules, texture memory is being copied to and from the GPU multiple times, which is not the most efficient means of creating a texture though it is a sacrifice made to enable the versatility of the framework.

The noise calculations output a value in the range -1.0 to 1.0. This is mapped to the RGB range 0.0 to 1.0 with 0.0 being black and 1.0 white. The only exception to this is the AHNModifierAbsolute class that performs its calculations before the range is remapped to enable the abs() function to have an effect. Where colour is used, it is sometimes necessary to average the RGB components to greyscale in order to perform calculations, so it is advisable to carry out any colourising of textures at the end of module chains.

How to Use

Chains must start with an AHNGenerator subclass to create the initial texture. These classes require input but output a texture, such as simplex noise.

```
let simplex = AHNGeneratorSimplex()
```

The details of class properties are discussed in detail later in the documentation.

With the first noise module ready it can either be directly turned into a UIImage:

```
let image = simplex.uiImage()
```

Otherwise it can be used as the input to another AHNoise module. For example it can have its values clamped:

```
let clamp = AHNModifierClamp()
clamp.input = simplex
clamp.min = 0.3
clamp.max = 0.8
```

This will clamp the noise values to 0.3 and 0.8.

There are also modules that accept two or three inputs. These are called AHNCombiners and AHNSelectors respectively. All AHNGenerator, AHNModifier, AHNCombiner and AHNSelector classes conform to the AHNTextureProvider protocol, which is the type used to provide input to a module. It is also the type that provides the uiImage() function to convert the underlying noise texture into a usable UIImage. A UIImage can then be converted into an SKTexture if necessary for use in gaming.

Example

The below example will produce a dark wood texture with deep browns in the grain and output it to a UIImage.

```
// Create a simplex noise generator to displace the rings in the x direction.
let xDisplace = AHNGeneratorSimplex()
xDisplace.textureWidth = 512
xDisplace.textureHeight = 512

// Create a simplex noise generator to displace the rings in the y direction and change
the z value to differentiate it from the x displacement.
let yDisplace = AHNGeneratorSimplex()
yDisplace.textureWidth = 512
yDisplace.textureHeight = 512
yDisplace.zValue = 2

// Create some concentric rings and displace them.
let rings = AHNGeneratorCylinder()
rings.xDisplacementInput = xDisplace
rings.yDisplacementInput = yDisplace
rings.textureWidth = 512
rings.textureHeight = 512
rings.frequency = 4.3
rings.displacementStrength = 0.22

// Clamp the values of the rings to create a more realistic effect.
let ringsClamp = AHNModifierClamp()
ringsClamp.provider = rings
ringsClamp.minimum = 0.3
ringsClamp.maximum = 0.75

// Stretch the rings slightly to elongate them.
let ringsStretch = AHNModifierStretch()
ringsStretch.provider = ringsClamp
ringsStretch.yFactor = 1.8

// Create a simplex noise generator for some high frequency noise to act as wood grain.
let grain = AHNGeneratorSimplex()
grain.textureWidth = 512
grain.textureHeight = 512
```

```
grain.frequency = 80
```

```
// Clamp the grain values to create a more even effect.
```

```
let grainClamp = AHNModifierClamp()  
grainClamp.provider = grain  
grainClamp.minimum = 0.45
```

```
// Stretch the grain to produce the long grain effect.
```

```
let grainStretch = AHNModifierStretch()  
grainStretch.provider = grainClamp  
grainStretch.yFactor = 20
```

```
// Combine the rings and the grain.
```

```
let multiply = AHNCombinerMultiply()  
multiply.provider = ringsStretch  
multiply.provider2 = grainStretch
```

```
// Add a brown colour to the lower noise values, a lighter brown for higher values and  
white for very light highlighting.
```

```
let colour = AHNModifierColour()  
colour.provider = multiply  
colour.colours = [  
  (colour: UIColor(red: 0.30, green: 0.20, blue: 0.0, alpha: 1.0), position: 0.11,  
    intensity: 1.0),  
  (colour: UIColor(red: 0.51, green: 0.34, blue: 0.0, alpha: 1.0), position: 0.53,  
    intensity: 1.0),  
  (colour: UIColor(red: 1.0, green: 1.0, blue: 1.0, alpha: 1.0), position: 0.95,  
    intensity: 1.0)  
]
```

Class References

Protocols

The `AHNTextureProvider` is adopted by all `AHNoise` classes with the exception of `AHNContext`.

`AHNTextureProvider`

Defines the interface shared by all `AHNoise` modules to allow them to be chained together. This should be adopted by custom modules.

Classes

The `AHNoise` classes come under four main categories with a couple of outliers:

`AHNModiferColour` and `AHNModifierScaleCanvas` are not subclassed from `AHNModifer` for inheritance reasons, though they operate in a similar manner to them. This should not be noticeable when using either class.

`AHNContext`

A wrapper for `Metal` classes used to communicate with the GPU.

`AHNGenerator`

An abstract superclass for noise generator objects that require no inputs, but output a noise texture.

`AHNGeneratorCoherent`

Generates billowing noise that is useful for representing clouds.

`AHNGeneratorBillow`

Generates billowing noise that is useful for representing clouds.

`AHNGeneratorRidgedMulti`

Generates multifractal noise with sharp, defined edges useful for representing mountain ranges.

`AHNGeneratorSimplex`

Generates simplex noise, the successor to Perlin noise, useful for representing natural phenomena such as terrain.

`AHNGeneratorVoronoi`

Generates a 3D field of cells.

`AHNGeneratorChecker`

Generates a black and white checkerboard pattern.

`AHNGeneratorConstant`

Generates a constant colour based on selected RGB values.

`AHNGeneratorCylinder`

Generates a 3D field of concentric cylinders.

`AHNGeneratorGradientBox`

Generates a box or diamond gradient from the four edges of the texture to the centre.

`AHNGeneratorGradientLinear`

Generates a linear gradient between two control points.

`AHNGeneratorGradientRadial`

Generates a radial gradient centred on a control point.

`AHNGeneratorSphere`

Generates a 3D field of concentric spheres.

`AHNGeneratorWaves`

Generates sinusoidal waves.

`AHNModifier`

An abstract superclass for noise modifier objects that accept one input, modify it, and outputs the modified texture.

`AHNModifierAbsolute`

Replaces all negative values of the noise input with their positive, absolute values.

`AHNModifierBlur`

Blurs the input using a Gaussian filter with a specified radius.

AHNModifierClamp

Replaces the input values outside of the specified range with the extreme values of that range.

AHNModifierColour

Maps input values to a specified colour and interpolates colours between points.

AHNModifierInvert

Replaces the input values with their opposite values, essentially multiplying all values by -1.

AHNModifierLoop

Causes the input values above a specified value to be remapped back to the range 0 - value.

AHNModifierPerspective

Applies a perspective transform to the input noise texture.

AHNModifierRotate

Rotates the input noise texture by a specified angle.

AHNModifierRound

Rounds input values to the nearest multiple of a specified value.

AHNModifierScaleBias

Applies a scale multiplier to all input values with the option of an additive bias applied after the multiplication.

AHNModifierStep

Compares the input values to a specified boundary value, and outputs an upper value or lower value depending on the result.

AHNModifierStretch

Stretches the input texture in the x and y axis by specified amounts.

AHNModifierSwirl

Applies a swirl transform to the input texture.

AHNModifierMapNormal

Produces a normal map of the input texture by analysing the colour gradient across pixels.

AHNModifierScaleCanvas

Resizes and repositions the input texture to a new output texture.

AHNCombiner

An abstract superclass for noise modifier objects that accept two inputs, combines them on a pixel for pixel basis, and outputs the combined texture.

AHNCombinerAdd

Adds the two input values together.

AHNCombinerDivide

Divides the first value input by the second input value.

AHNCombinerMax

Outputs the maximum of the two input values.

AHNCombinerMin

Outputs the minimum of the two input values.

AHNCombinerMultiply

Multiplies the two input values together.

AHNCombinerPower

Raises the first input value to the power of the second input value.

AHNCombinerSubtract

Subtracts the second input value from the first input value.

AHNSelector

An abstract superclass for noise modifier objects that accept two inputs and a selector. The output is a combination of the two inputs based on the corresponding selector value.

AHNSelectorBlend

The two input values are mixed together with a mix weighting provided by the selector value.

AHNSelectorSelect

The input values are compared to the selector value, and one or the other inputs is output depending on the results of this comparison.

AHNTextureProvider

Overview

The AHNTextureProvider protocol defines the interface for modules that output a noise texture. Every class in this framework adopts this protocol with the exception of the AHNContext class. This protocol handles the

Symbols

Initialiser

```
init()
```

Returns a new AHNTextureProvider object.

Texture Generation

```
func canUpdate() -> Bool
```

Returns true if the object has enough inputs to provide an output.

```
var context: AHNContext
```

The AHNContext that is being used by the AHNTextureProvider to communicate with the GPU. This is recovered from the first AHNGenerator class that is encountered in the chain of classes.

```
var dirty: Bool
```

Returns true if the texture needs to be updated to represent the current properties of this AHNTextureProvider object only. This does not take into account changes in other AHNTextureProvider objects this object is dependent on.

```
func isDirty() -> Bool
```

Returns true if the texture needs to be updated to represent the current properties of this AHNTextureProvider object or the updated properties of any other object this one is dependent on for creating a texture. For example if an input object has a property changed this would return true, whereas the dirty property would return false.

```
func texture() -> MTLTexture?
```

Returns the updated output texture for this module representing the current property values of the calling object and all dependents. This method checks whether any updates are needed by calling isDirty() and if it returns true, updateTexture() is called. If the texture cannot be calculated due to insufficient inputs, this function returns nil.

```
func updateTexture()
```

Updates the output MTLTexture. This should not need to be called manually as it is called by the texture() method automatically if the texture does not represent the current properties of the calling object or its dependents.

```
func textureSize() -> MTLSize
```

Returns the size of the texture in pixels. The z component of this value is always 1 to correspond to a 1 pixel depth.

```
func textureProvider() -> AHNTTextureProvider?
```

Returns the first input AHNTTextureProvider object that is used to produce an output. If there is no input this will return nil.

Image Conversion

```
func uiImage() -> UIImage?
```

Returns A UIImage created from the output MTLTexture provided by the texture() function. If there are insufficient inputs to calculate the texture() function, this returns nil.

Value Outputs

```
func greyscaleValuesAtPositions(positions: [CGPoint]) -> [Float]
```

Returns the greyscale values in the texture for specified positions, useful for using the texture as a heightmap. The values for each channel in a coloured image are averaged to produce an output.

Parameter: positions-The 2D pixel positions in the texture for which to return greyscale values between 0.0-1.0.

Though CGPoint is used for the input coordinates to locate pixels, it should be noted that floating point values will be rounded to integer values before accessing a pixel as there is no 4.5th pixel.

```
func colourValuesAtPositions(positions: [CGPoint]) -> [(red: Float, green: Float, blue: Float, alpha: Float)]
```

Returns the colour values in the texture for specified positions.

Parameter: positions-The 2D positions in the texture for which to return colour values for red, green, blue and alpha between 0.0-1.0.

Though CGPoint is used for the input coordinates to locate pixels, it should be noted that floating point values will be rounded to integer values before accessing a pixel as there is no 4.5th pixel.

```
func allTexturePositions() -> [CGPoint]
```

Returns all the texture positions for the calling texture for use with the greyscaleValuesAtPositions() and colourValuesAtPositions() functions.

AHNContext

Overview

The AHNContext class is a wrapper for the Metal objects used to communicate with the GPU. The SharedContext static property is used by any AHNGenerator object to encode commands for the GPU. Other objects such as those in the the AHNModifier, AHNCombiner or AHNSelector class groups request an AHNContext object from their first input for communication with the GPU, this results in all modules in a chain using the same context as the first generator in the chain.

Most of the time you won't have to use this class as it is managed automatically. The only reason you would want to directly use this class is to change which MTLDevice is being used by the context, which effectively lets you choose which graphics hardware to use. For iPhones there is only one device available, but for Macs there can be more than one available to use if for example, there is a separate graphics card installed. To set the device used by the SharedContext object, call the SetContextDevice static function.

Symbols

Access

```
static var SharedContext: AHNContext!
```

The shared AHNContext object that is used by all AHNTextureProvider objects to communicate with the GPU.

```
static func SetContextDevice(device: MTLDevice)
```

Set the MTLDevice of the SharedContext object to a specific object. An MTLDevice is a representation of a GPU, so apps for macOS (OSX) will want to set the device to the most powerful graphics hardware available, and not automatically default to onboard graphics.

Persistent Metal Objects

```
let commandQueue: MTLCommandQueue
```

The MTLCommandQueue that is used to create MTLCommandEncoders for each kernel.

```
let device: MTLDevice
```

The MTLDevice used by the various noise classes to create buffers, pipelines and command encoders.

```
let library: MTLLibrary
```

The MTLLibrary that stores the Metal kernel functions used to create and manipulate noise.

Relationships

Subclass of NSObject.

AHNGenerator

Overview

The AHNGenerator class provides an abstract superclass for all modules that generate an output texture while requiring no inputs. It handles the binding of textures and buffers to the Metal command encoders leaving subclasses to handle which compute kernel to use and which properties to bind to the command encoder. It also handles the creation and updating of textures to represent the input arguments.

The textures provided by an AHNGenerator subclass represent a 2D slice through a 3D noise field. This slice can be rotated about the X, Y and Z axes to distort the resulting texture.

Symbols

Initialisers

```
init(functionName: String)
```

Initialises an AHNGenerator object that will use a Metal function with the specified functionName to create the output texture. This is called by subclasses to create a generator class that implements a specific Metal function.

```
init()
```

Initialises an AHNGenerator object with a dummy function.

Persistent Metal Objects

```
var context: AHNContext
```

The AHNContext that is being used by the AHNGenerator to communicate with the GPU. This is taken from the SharedContext class property of AHNContext.

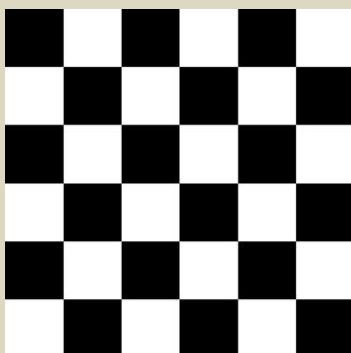
```
var uniformBuffer: MTLBuffer?
```

The MTLBuffer used to communicate the constant values used by the compute kernel to the GPU. This is encoded into the command buffer at the index 0.

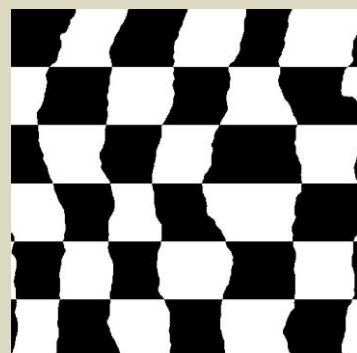
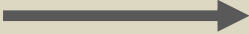
Texture Modification

```
var xoffsetInput: AHNTextureProvider?
```

The texture of another AHNTextureProvider to offset pixels by in the x axis. Pixel values less than 0.5 offset to the left, and above 0.5 offset to the right.



Simplex set as
xoffsetInput



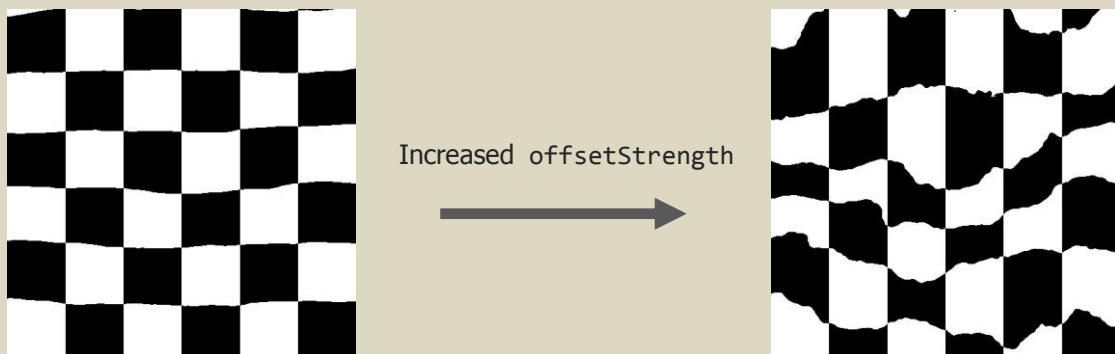
`var yoffsetInput: AHNTextureProvider?`

The texture of another [AHNTextureProvider](#) to offset pixels by in the y axis. Pixel values less than 0.5 offset downwards, and above 0.5 offset upwards.



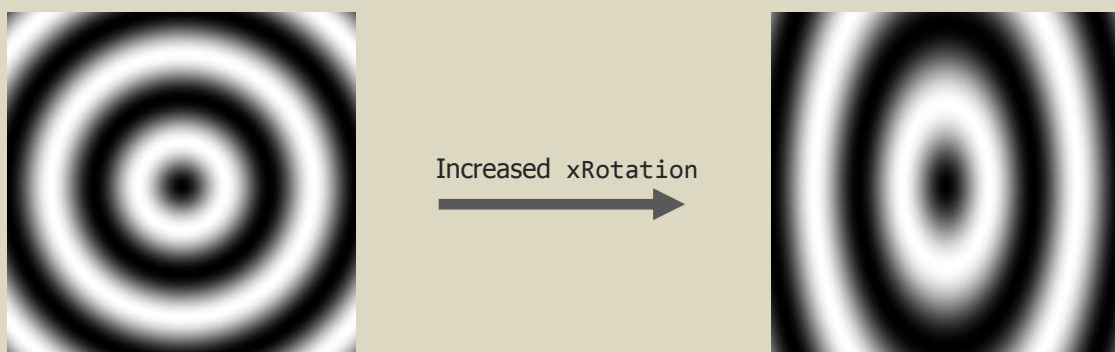
`var offsetStrength: Float`

The intensity of the effects of the `xoffsetInput` and `yoffsetInput`. A value of 0.0 results in no displacement. The default value is 0.2.



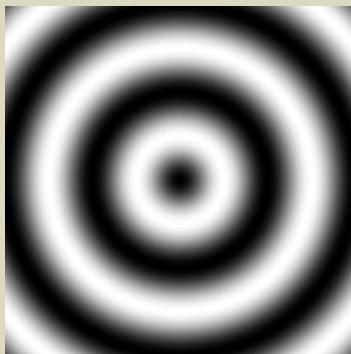
`var xRotation: Float`

The angle (in radians) by which to rotate the 2D slice of the texture about the x axis of the 3D space of the geometric or noise `kernelFunction`. The default value is 0.0.

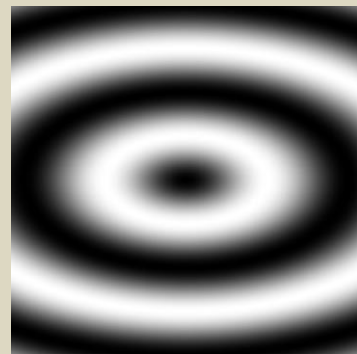


```
var yRotation: Float
```

The angle (in radians) by which to rotate the 2D slice of the texture about the y axis of the 3D space of the geometric or noise kernelFunction. The default value is 0.0.

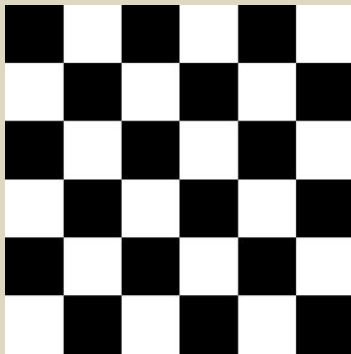


Increased yRotation
→

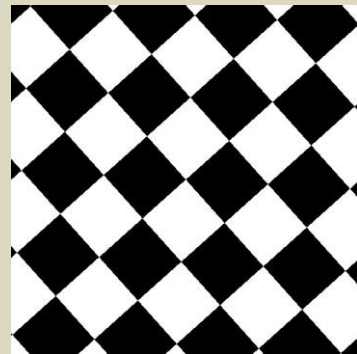


```
var zRotation: Float
```

The angle (in radians) by which to rotate the 2D slice of the texture about the z axis of the 3D space of the geometric or noise kernelFunction. The default value is 0.0.



Increased zRotation
→



```
var textureWidth: Int
```

The width of the output MTLTexture in pixels. The default value is 128.

```
var textureHeight: Int
```

The height of the output MTLTexture in pixels. The default value is 128.



Increased textureWidth
and textureHeight from
64 to 512
→



```
var dirty: Bool
```

Indicates whether or not the output texture needs updating.

Texture Creation and Updating

```
func configureArgumentTableWithCommandencoder(commandEncoder: MTLComputeCommandEncoder)
```

This function is overridden by subclasses to write class specific variables to the uniformBuffer.

Parameter: commandEncoder – The MTLComputeCommandEncoder used to encode the kernel. This can be used to lazily create a buffer of data and add it to the argument table. Any buffer index can be used without affecting the rest of this class.

Relationships

Subclass of NSObject.

Conforms to the AHNTextureProvider protocol.

AHNGeneratorCoherent

Overview

The `AHNGeneratorCoherent` class provides an abstract superclass for all modules that generate an output texture representing coherent, procedural noise. This class provides the necessary properties to create coherent noise such as octaves and frequency and handles the binding of these properties to the Metal command encoder for execution on the GPU.

The effects of each of the properties is slightly different for each different type of noise.

Symbols

Initialisers

```
init(functionName: String)
```

Initialises an `AHNGeneratorCoherent` object that will use a Metal function with the specified `functionName` to create the output texture. This is called by subclasses to create a generator class that implements a specific Metal function.

```
init()
```

Initialises an `AHNGeneratorCoherent` object with a dummy function.

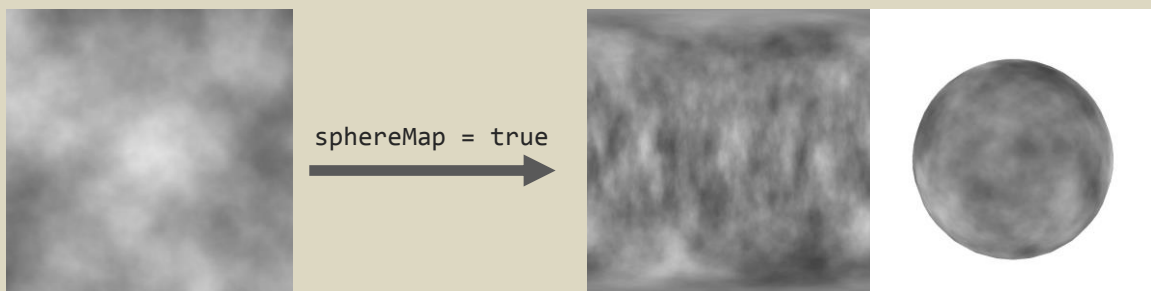
Texture Modification

```
var use4D: Bool
```

When `true`, the simplex kernel used has four degrees of freedom, which allows for interesting seamless patterns but has extra computational cost. This has no effect for the `AHNGeneratorVoronoi`, which is always calculated in 4 dimensions. The default value is `false`.

```
var sphereMap: Bool
```

When `true`, the output texture is warped towards the top and bottom to seamless map to a UV sphere. When `true` the `zValue` and `wValue` properties have no effect as they are overridden in the shader to produce the sphere mapped effect. The default value is `false`.

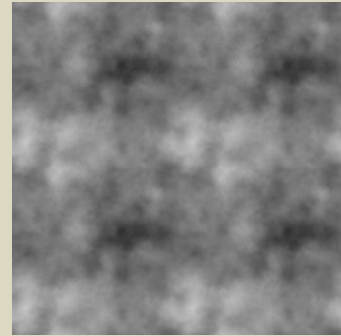


```
var seamless: Bool
```

When true, the output texture can be seamlessly tiled without apparent edges showing. When true the `zValue` and `wValue` properties have no effect as they are overridden in the shader to produce the seamless effect. The default value is false.



seamless = true
→



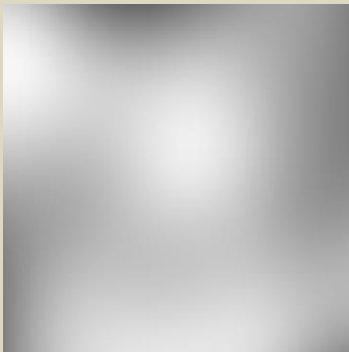
```
var octaves: Int
```

The number of octaves to use in the texture. Each octave is calculated with a different amplitude (altered by the `persistence` property) and frequency (altered by the `lacunarity` property). The amplitude starts with a value of 1.0, the first octave is calculated using this value, the amplitude is then multiplied by the `persistence` and the next octave is calculated using this new amplitude before multiplying it again by the `persistence` and so on. The frequency follows a similar pattern with the `lacunarity` property.

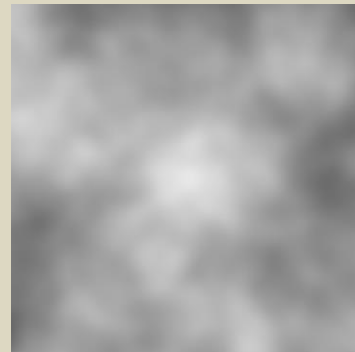
Each octave is calculated and then combined to produce the final value.

Higher values (12) produce more detailed noise, where as lower values produce smoother noise. Higher values have a performance impact.

The default value is 6.



octaves = 1 to
octaves = 4
→



`var persistence: Float`

Varies the amplitude every octave. The amplitude is multiplied by the persistence for each octave. Generally values less than 1.0 are used.

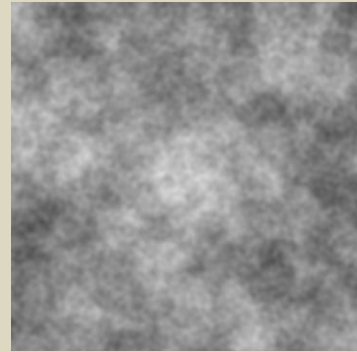
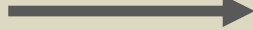
For example an initial amplitude of 1.0 (fixed) and a persistence of 0.5 for 4 octaves would produce an amplitude of 1.0, 0.5, 0.25 and 0.125 respectively for each octave.

Higher values result in a rougher texture.

The default value is 0.5.



Increased persistence



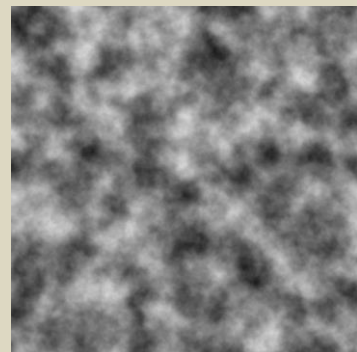
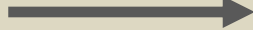
`var frequency: Float`

The frequency used when calculating the noise. Higher values produce more dense noise with more closely packed features. The frequency is multiplied by the lacunarity property each octave.

The default value is 1.0.



Increased frequency



```
var lacunarity: Float
```

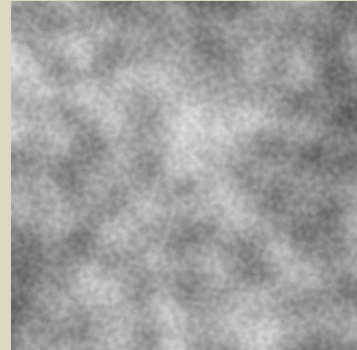
Varies the frequency every octave. The frequency is multiplied by the lacunarity for each octave. Generally values greater than 1.0 are used.

For example an initial frequency of 1.0 and a lacunarity of 2.0 for 4 octaves would produce a frequency of 1.0, 2.0, 4.0 and 8.0 respectively for each octave.

The default value is 2.0.



Increased lacunarity
→



```
var xValue: Float
```

The origin of the noise along the x axis in noise space. Changing this slightly will make the noise texture appear to move.

The default value is 1.0.

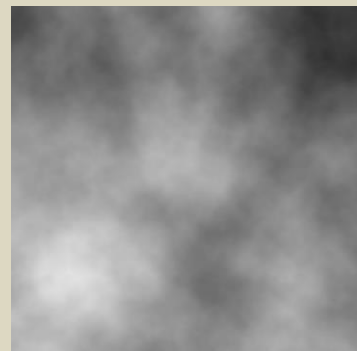
```
var yValue: Float
```

The origin of the noise along the y axis in noise space. Changing this slightly will make the noise texture appear to move.

The default value is 1.0.



Increased xValue and
yValue
→



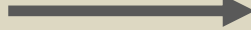
```
var zValue: Float
```

The value for the third dimension when calculating the noise. Changing this slightly will make the noise texture appear to animate.

Default is 1.0.



Increased zValue



```
var wValue: Float
```

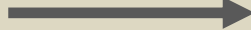
The value for the fourth dimension when calculating the noise. Changing this slightly will make the noise texture appear to animate.

This essentially has the same effect as zValue, but is another variable to change when the zValue is already being used. For example when calculating a volumetric noise body, this can be used to change the whole volume while leaving the zValue alone.

Default is 1.0.



Increased wValue



Relationships

Subclass of AHNGenerator.

Conforms to the AHNTtextureProvider protocol.

AHNGeneratorBillow

Overview

The AHNGeneratorBillow class outputs a coherent, procedural noise texture that represents billowing clouds. This has a smoother appearance than other coherent generators.

Symbols

Initialisers

`init()`

Initialises an AHNGeneratorBillow object with default property values.



Relationships

Subclass of AHNGeneratorCoherent.

Conforms to the AHNTextureProvider protocol.

AHNGeneratorRidgedMulti

Overview

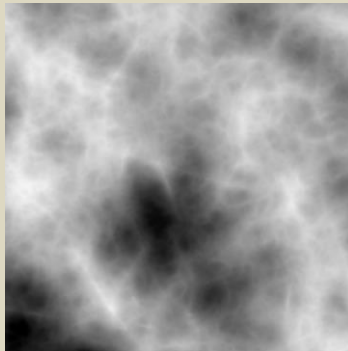
The AHNGeneratorRidgedMulti class outputs a coherent, procedural noise texture that represents jagged terrain. This has a sharper, more defined appearance than other coherent generators.

Symbols

Initialisers

`init()`

Initialises an AHNGeneratorRidgedMulti object with default property values.



Relationships

Subclass of AHNGeneratorCoherent.

Conforms to the AHNTexureProvider protocol.

AHNGeneratorSimplex

Overview

The AHNGeneratorSimplex class outputs a coherent, procedural noise texture that represents pure simplex noise. Simplex noise is similar to Perlin noise but uses a more efficient algorithm. Simplex noise is useful for resembling natural unevenness or phenomena.

Symbols

Initialisers

`init()`

Initialises an AHNGeneratorSimplex object with default property values.



Relationships

Subclass of AHNGeneratorCoherent.

Conforms to the AHNTextureProvider protocol.

AHNGeneratorVoronoi

Overview

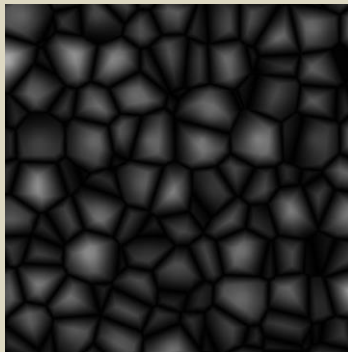
The `AHNGeneratorVoronoi` class outputs a texture of cells surrounding randomly seeded points. This is useful for representing natural phenomena such as plateaus or crystals. It is also useful for dividing an area into discrete sections.

Symbols

Initialisers

`init()`

Initialises an `AHNGeneratorVoronoi` object with default property values.



Relationships

Subclass of `AHNGeneratorCoherent`.

Conforms to the `AHNTextureProvider` protocol.

AHNGeneratorChecker

Overview

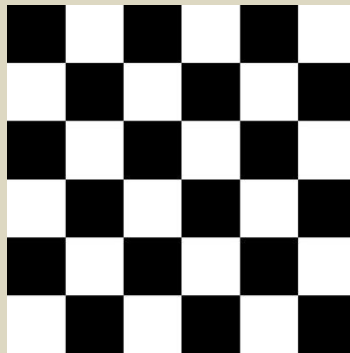
The AHNGeneratorChecker class outputs a geometric noise texture representing a 3D checker grid or alternating cube pattern.

Symbols

Initialisers

`init()`

Initialises an AHNGeneratorChecker object with default property values.

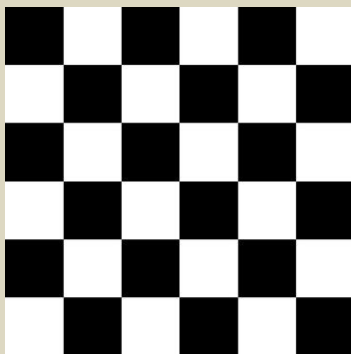


Texture Modification

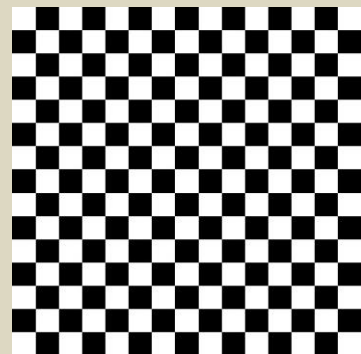
`var frequency: Float`

The frequency of the cubes, higher values result in more, closer packed cubes.

The default value is 1.0.

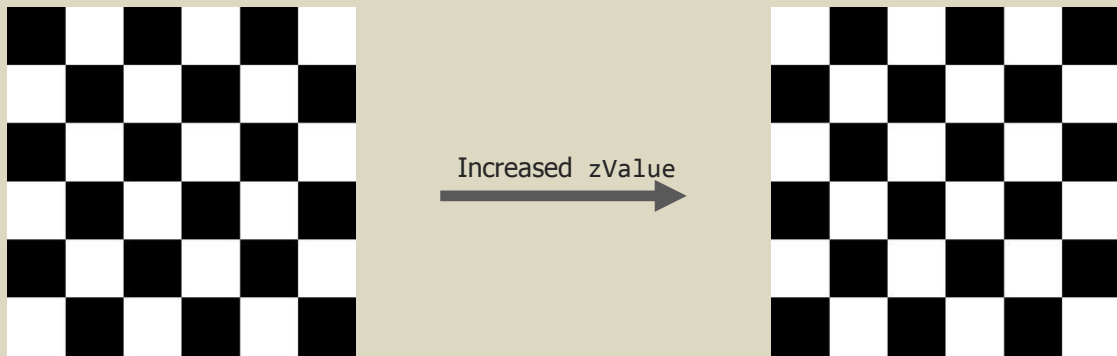


Increased frequency
→



```
var zValue: Float
```

The value along the z axis that the texture slice is taken. This moves the slice into the next layer of bocks of alternating colour and can be combined with inherited slice transforming properties to produce interesting results. The default value is 0.0.



Relationships

Subclass of `AHNGenerator`.

Conforms to the `AHNTextureProvider` protocol.

AHNGeneratorConstant

Overview

The AHNGeneratorConstant class outputs a texture of constant RGB value. This can be useful when combining or displacing noise values.

Symbols

Initialisers

```
init()
```

Initialises an AHNGeneratorConstant object with default property values.



Texture Modification

```
var red: Float
```

The red component of the colour to be output in the range 0.0-1.0.

The default value is 0.5.

```
var green: Float
```

The green component of the colour to be output in the range 0.0-1.0.

The default value is 0.5.

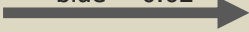
```
var blue: Float
```

The blue component of the colour to be output in the range 0.0-1.0.

The default value is 0.5.



red = 0.28
green = 0.82
blue = 0.62



Relationships

Subclass of `AHNGenerator`.

Conforms to the `AHNTextureProvider` protocol.

AHNGeneratorCylinder

Overview

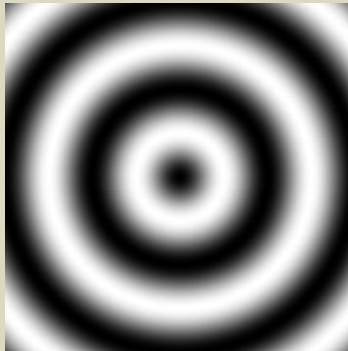
The `AHNGeneratorCylinder` class outputs a texture representing a field of concentric cylinders. By rotating the texture slice through this field it is possible to create elongated concentric rings useful in wood effects.

Symbols

Initialisers

`init()`

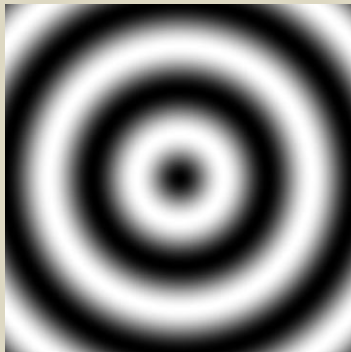
Initialises an `AHNGeneratorCylinder` object with default property values.



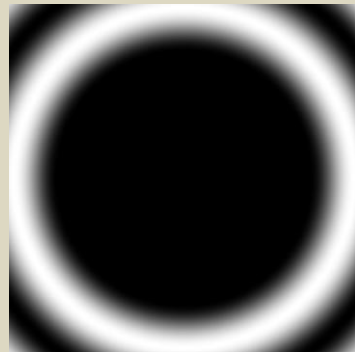
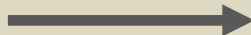
Texture Modification

`var offset: Float`

The distance to offset the first cylinder from the centre by. The default value is `0.0`.

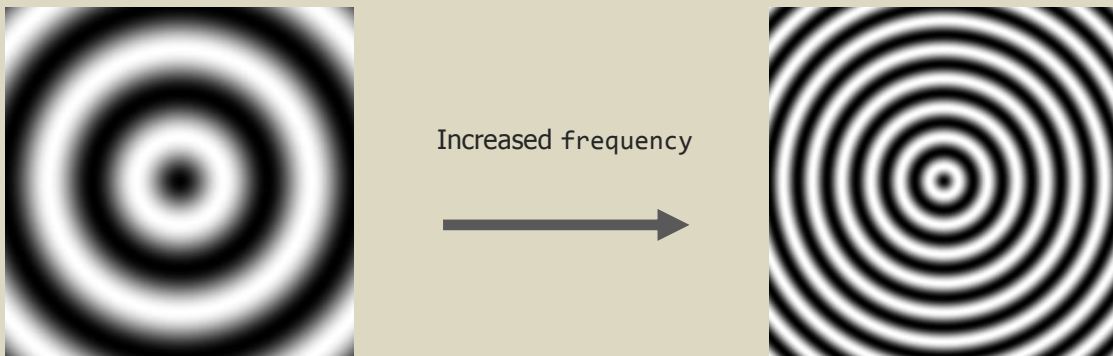


Increased offset



```
var frequency: Float
```

The frequency of the cylinders, higher values result in more, closer packed cylinders. The default value is 1.0.

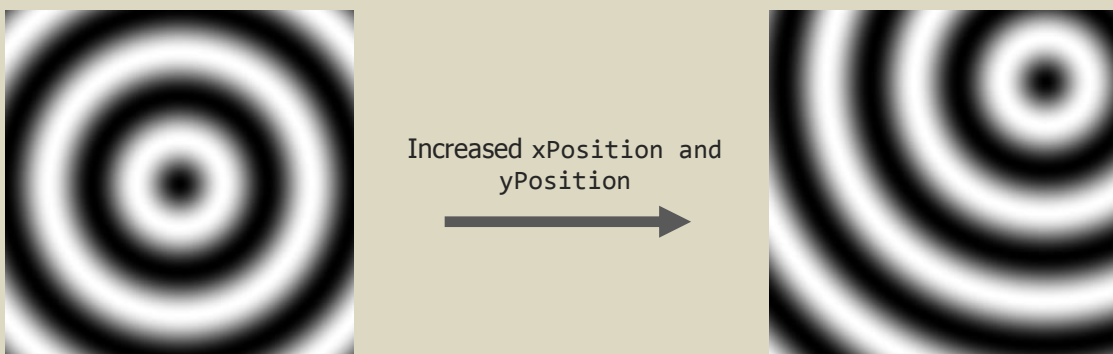


```
var xPosition: Float
```

The position along the x axis that the cylinders are centred on. A value of 0.0 corresponds to the left texture edge, and a value of 1.0 corresponds to the right texture edge. The default value is 0.5.

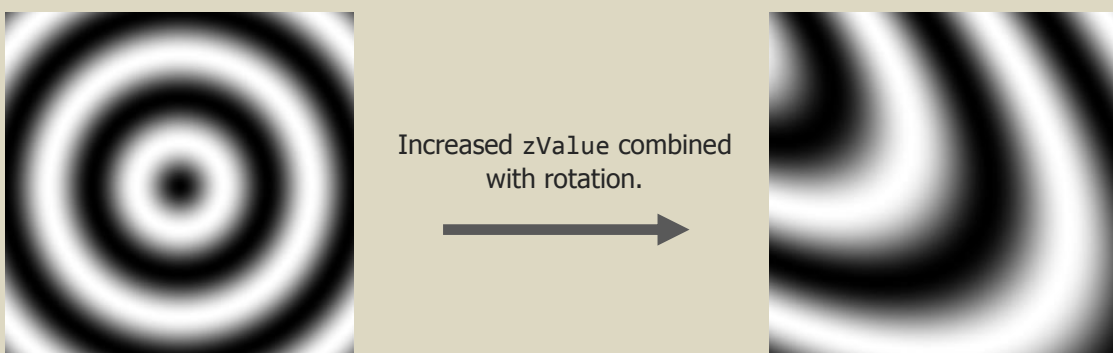
```
var yPosition: Float
```

The position along the y axis that the cylinders are centred on. A value of 0.0 corresponds to the bottom texture edge, and a value of 1.0 corresponds to the top texture edge. The default value is 0.5.



```
var zValue: Float
```

The value along the z axis that the texture slice is taken. This essentially slides the slice up the height of the cylinders, so can appear to have no effect on cylinders that have not been rotated using inherited properties. The default value is 0.0.



Relationships

Subclass of `AHNGenerator`.

Conforms to the `AHNTextureProvider` protocol.

AHNGeneratorGradientBox

Overview

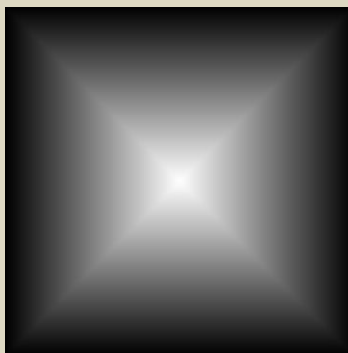
The `AHNGeneratorGradientBox` class outputs a texture representing a gradient originating from the four edges of the texture with specified a falloff in the X and Y directions.

Symbols

Initialisers

```
init()
```

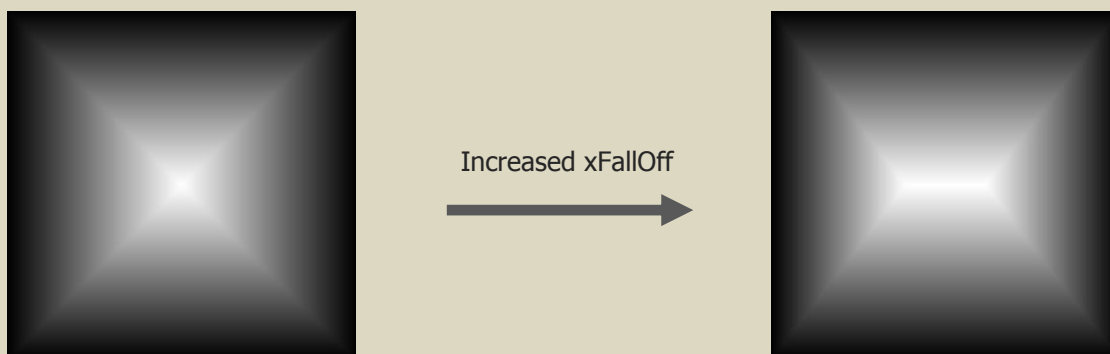
Initialises an `AHNGeneratorGradientBox` object with default property values.



Texture Modification

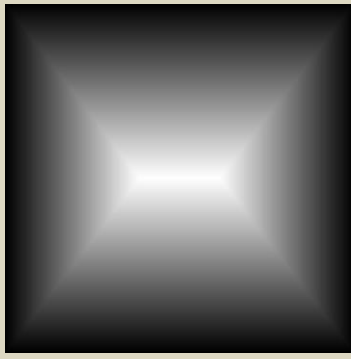
```
var xFalloff: Float
```

The falloff of the gradients originating from the left and right hand texture edges. The default value of `0.0` results in the horizontal gradients terminating at the centre of the texture, higher values cause the gradient to terminate closer to its originating edge.

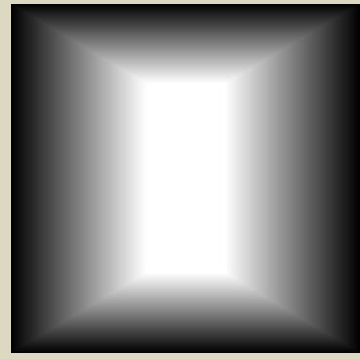


```
var yFalloff: Float
```

The falloff of the gradients originating from the top and bottom texture edges. The default value of `0.0` results in the vertical gradients terminating at the centre of the texture, higher values cause the gradient to terminate closer to its originating edge.



Increased yFallOff with
xFallOff > 0



Relationships

Subclass of AHNGenerator.

Conforms to the AHNTextureProvider protocol.

AHNGeneratorGradientLinear

Overview

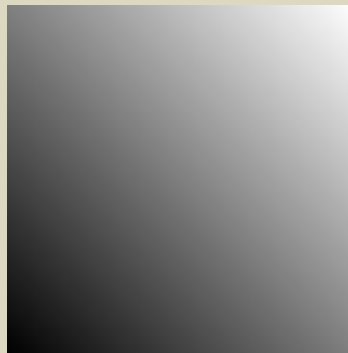
The `AHNGeneratorGradientLinear` class outputs a texture representing a linear gradient ranging from 0 to 1 between two control points.

Symbols

Initialisers

`init()`

Initialises an `AHNGeneratorGradientLinear` object with default property values.



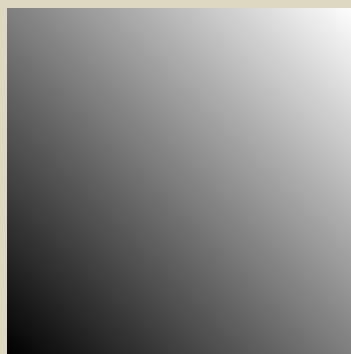
Texture Modification

`var startPositionX: Float`

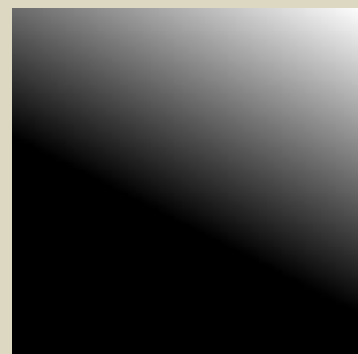
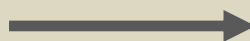
The location along the x axis of the first control point. A value of `0.0` corresponds to the left hand edge and a value of `1.0` corresponds to the right hand edge. The first control point dictates where the black section starts to fade to white. The default value is `0.0`.

`var startPositionY: Float`

The location along the y axis of the first control point. A value of `0.0` corresponds to the bottom edge and a value of `1.0` corresponds to the top edge. The first control point dictates where the black section starts to fade to white. The default value is `0.0`.



Increased `startPositionX`
and `startPositionY`

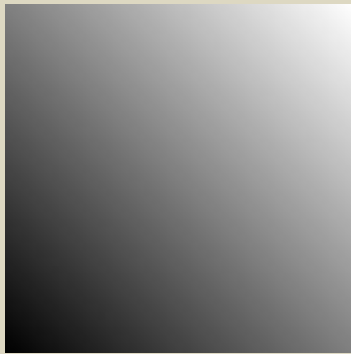


```
var endPositionX: Float
```

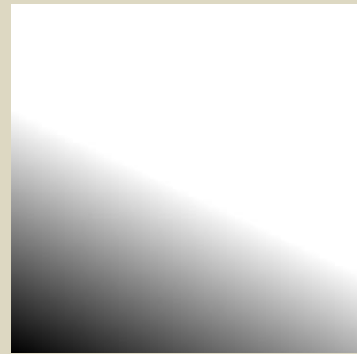
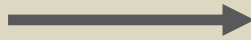
The location along the x axis of the second control point. A value of 0.0 corresponds to the left hand edge and a value of 1.0 corresponds to the right hand edge. The second control point dictates where the black section has completely faded to white. The default value is 1.0.

```
var endPositionY: Float
```

The location along the y axis of the second control point. A value of 0.0 corresponds to the bottom edge and a value of 1.0 corresponds to the top edge. The second control point dictates where the black section has completely faded to white. The default value is 1.0.



Decreased endPositionX
and endPositionY



Relationships

Subclass of AHNGenerator.

Conforms to the AHNTextureProvider protocol.

AHNGeneratorGradientRadial

Overview

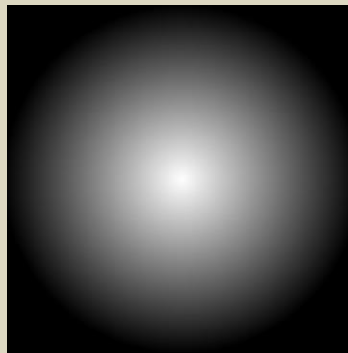
The `AHNGeneratorGradientRadial` class outputs a texture representing a gradient originating from a control point with specified a falloff in the X and Y directions.

Symbols

Initialisers

```
init()
```

Initialises an `AHNGeneratorGradientRadial` object with default property values.



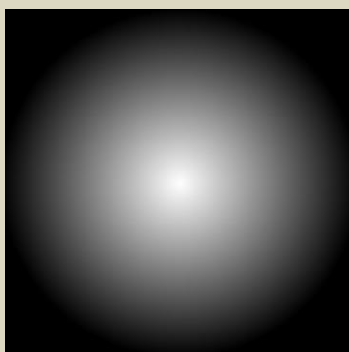
Texture Modification

```
var xPosition: Float
```

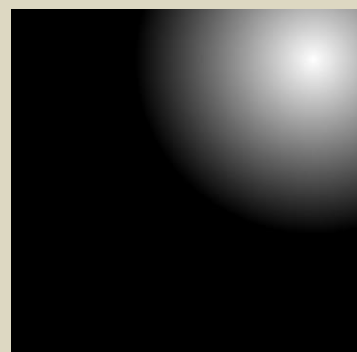
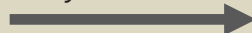
The location along the x axis of the control point that the gradient is centred on. A value of `0.0` corresponds to the left hand edge and a value of `1.0` corresponds to the right hand edge. The default value is `0.5`.

```
var yPosition: Float
```

The location along the y axis of the control point that the gradient is centred on. A value of `0.0` corresponds to the bottom edge and a value of `1.0` corresponds to the top edge. The default value is `0.5`.



Increased `xPosition` and
`yPosition`



```
var xFalloff: Float
```

The horizontal falloff of the radial gradient. A value of 1.0 results in the gradient terminating at the edges of the texture, lower values cause the gradient to extend beyond the edge of the texture and vice versa.



```
var yFalloff: Float
```

The vertical falloff of the radial gradient. A value of 1.0 results in the gradient terminating at the edges of the texture, lower values cause the gradient to extend beyond the edge of the texture and vice versa.



Relationships

Subclass of AHNGenerator.

Conforms to the AHNTextureProvider protocol.

AHNGeneratorSphere

Overview

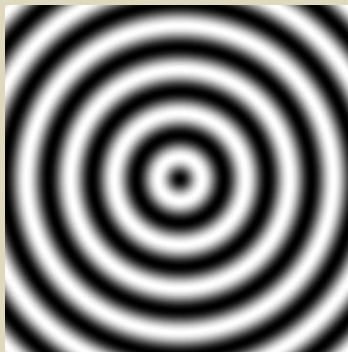
The `AHNGeneratorSphere` class outputs a texture representing a field of concentric spheres.

Symbols

Initialisers

`init()`

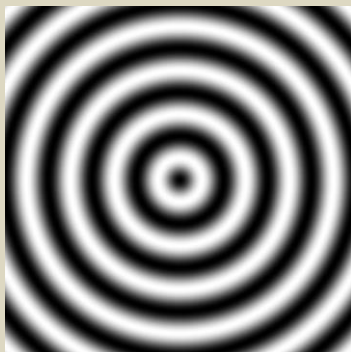
Initialises an `AHNGeneratorSphere` object with default property values.



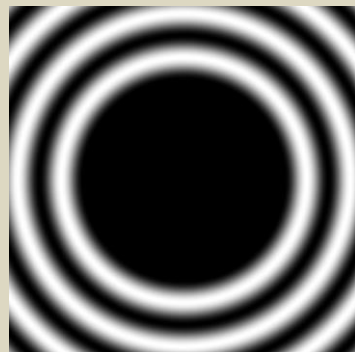
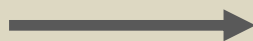
Texture Modification

`var offset: Float`

The distance to offset the first sphere from the centre by. The default value is `0.0`.

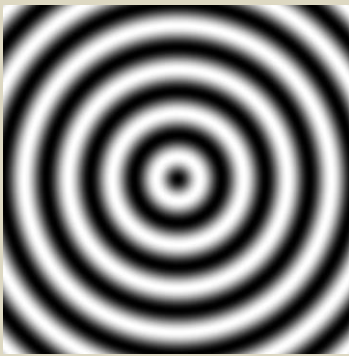


Increased `yFa110ff`

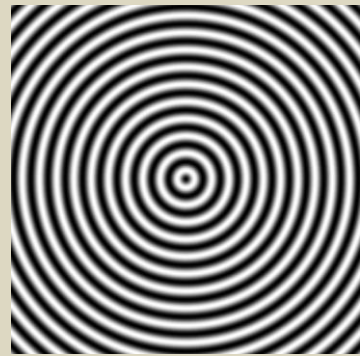


```
var frequency: Float
```

The frequency of the spheres, higher values result in more, closer packed spheres. The default value is 1.0.



Increased frequency
→

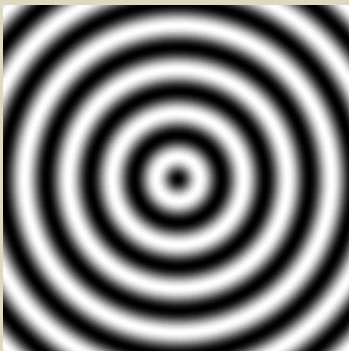


```
var xPosition: Float
```

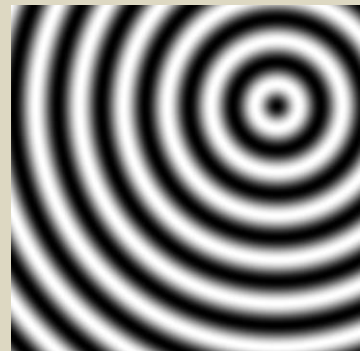
The position along the x axis that the spheres are centred on. A value of 0.0 corresponds to the left texture edge, and a value of 1.0 corresponds to the right texture edge. The default value is 0.5.

```
var yPosition: Float
```

The position along the y axis that the spheres are centred on. A value of 0.0 corresponds to the bottom texture edge, and a value of 1.0 corresponds to the top texture edge. The default value is 0.5.

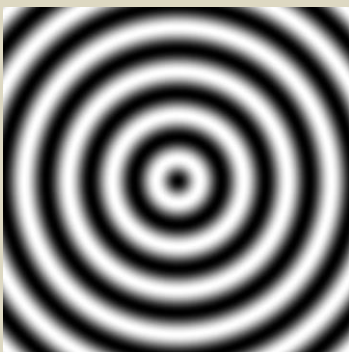


Increased xPosition and
yPosition
→

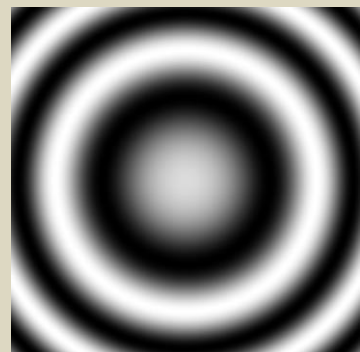


```
var zValue: Float
```

The value along the z axis that the texture slice is taken, cutting through differing sections of different spheres unlike with AHNGeneratorCylinder. The default value is 0.0.



Increased zValue
→



Relationships

Subclass of `AHNGenerator`.

Conforms to the `AHNTextureProvider` protocol.

AHNGeneratorWave

Overview

The `AHNGeneratorSphere` class outputs a texture representing a field of sinusoidal waves.

Symbols

Initialisers

```
init()
```

Initialises an `AHNGeneratorWave` object with default property values.



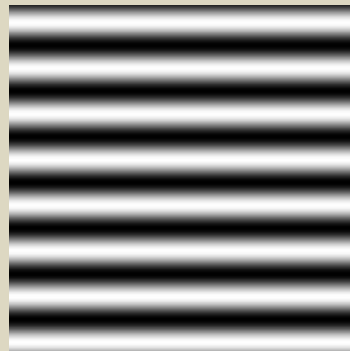
Texture Modification

```
var frequency: Float
```

Increases the number and compactness of waves visible in the texture. The default value is `1.0`.



Increased frequency
→



Relationships

Subclass of `AHNGenerator`.

Conforms to the `AHNTextureProvider` protocol.

AHNModifier

Overview

The `AHNModifier` class provides an abstract superclass for all modules that modify the output texture of another module. It handles the binding of textures and buffers to the `Metal` command encoders leaving subclasses to handle which compute kernel to use and which properties to bind to the command encoder. It also handles the creation and updating of textures to represent the input arguments.

Symbols

Initialisers

```
init(functionName: String)
```

Initialises an `AHNModifier` object that will use a `Metal` function with the specified `functionName` to create the output texture. This is called by subclasses to create a modifier class that implements a specific `Metal` function.

```
init()
```

Initialises an `AHNModifier` object with a dummy function.

Persistent Metal Objects

```
var context: AHNContext
```

The `AHNContext` that is being used by the `AHNModifier` to communicate with the GPU. This is taken from the first `AHNGenerator` in the input chain which in turn is taken from the `SharedContext` class property of `AHNContext`.

```
var uniformBuffer: MTLBuffer?
```

The `MTLBuffer` used to communicate the constant values used by the compute kernel to the GPU. This is encoded into the command buffer at the index 0.

Texture Properties

```
var textureWidth: Int
```

The width of the output `MTLTexture` in pixels. This is taken from the provider property's `textureWidth` value.

```
var textureHeight: Int
```

The height of the output `MTLTexture` in pixels. This is taken from the provider property's `textureHeight` value.

```
var dirty: Bool
```

Indicates whether or not the output texture needs updating.

Metal Performance Shader Incorporation

```
var usesMPS: Bool
```

Indicates whether this modifier makes use of a `Metal Performance Shader`. If set to `true`, a compute kernel is not used and instead a `Metal Performance Shader` is used to create the texture. When using MPS be sure to create a dummy function that is loaded on initialisation but not used.

Texture Creation and Updating

```
func configureArgumentTableWithCommandencoder(commandEncoder: MTLComputeCommandEncoder)
```

This function is overridden by subclasses to write class specific variables to the uniformBuffer.

Parameter: commandEncoder – The MTLComputeCommandEncoder used to encode the kernel. This can be used to lazily create a buffer of data and add it to the argument table. Any buffer index can be used without affecting the rest of this class.

```
func addMetalPerformanceShaderToBuffer(commandBuffer: MTLCommandBuffer)
```

Override this method in subclasses to configure a Metal Performance Shader to be used instead of a custom kernel. This is called when usesMPS is set to true, and is called instead of the above function. When overriding the function, update the Metal Performance Shader object and encode it to the provided commandBuffer.

Parameter: commandBuffer – The MTLCommandBuffer used to run the Metal Performance Shader.

Relationships

Subclass of NSObject.

Conforms to the AHNTextureProvider protocol.

AHNModifierAbsolute

Overview

The `AHNModifierAbsolute` class applies the `abs()` function to each pixels noise value. Pixel values are in the range `0.0` to `1.0`, and applying the `abs()` function to this range would have no effect. This means the pixel values must be converted back into the original `-1.0` to `1.0` noise range, then perform the `abs()` function and finally convert back into the colour range of `0.0` to `1.0`.

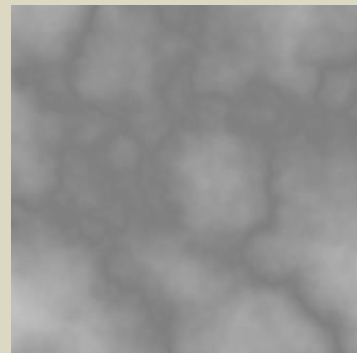
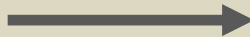
This results in a natural output texture with pixels in the range of `0.5` to `1.0`. This can be remapped to the `0.0` to `1.0` range by setting the `normalise` property to `true`.

Symbols

Initialisers

`init()`

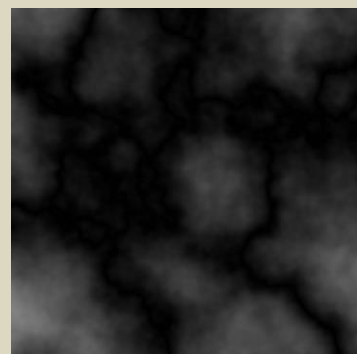
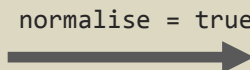
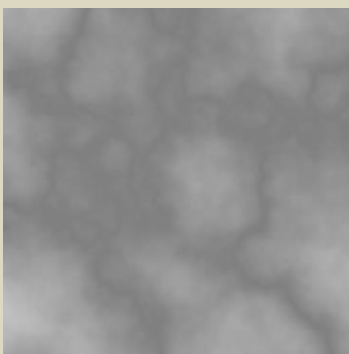
Initialises an `AHNModifierAbsolute` object with default property values.



Controlling Output

`var normalise: Bool`

Set this to `true` to remap the range of the output values back to `0.0` to `1.0` after applying the `abs()` function. When set to `false` the range of the output values is `0.5` to `1.0`. The default value is `false`.



Relationships

Subclass of `AHNModifier`.

Conforms to the `AHNTextureProvider` protocol.

AHNModifierBlur

Overview

The `AHNModifierBlur` class performs a Gaussian blur on the input texture using a specified blur radius.

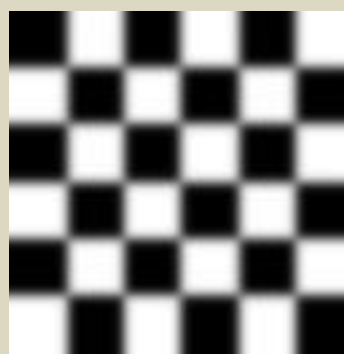
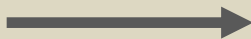
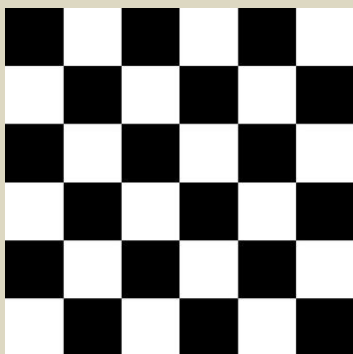
This modifier differs slightly from others as it uses the Metal Performance Shader implementation of the Gaussian blur function instead of a custom function.

Symbols

Initialisers

```
init()
```

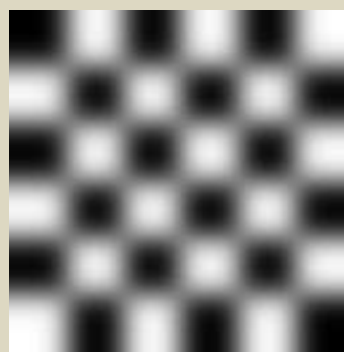
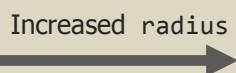
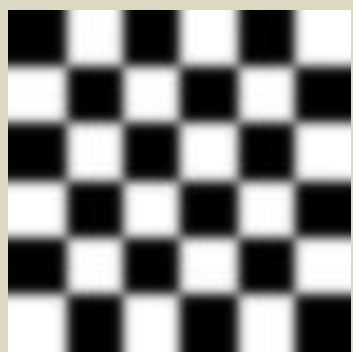
Initialises an `AHNModifierBlur` object with default property values.



Controlling Output

```
var radius: Float
```

The radius of the blur. Higher values result in a more blurred image but impacts performance. The default value is 3.0.



Relationships

Subclass of `AHNModifier`.

Conforms to the `AHNTextureProvider` protocol.

AHNModifierClamp

Overview

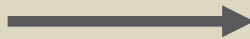
The `AHNModifierClamp` class replaces pixel values outside of a specified range with the values at the extremes of that range. The clamped range can be remapped to occupy the full `0.0` to `1.0` range again by setting the `normalise` property to `true`.

Symbols

Initialisers

```
init()
```

Initialises an `AHNModifierClamp` object with default property values.



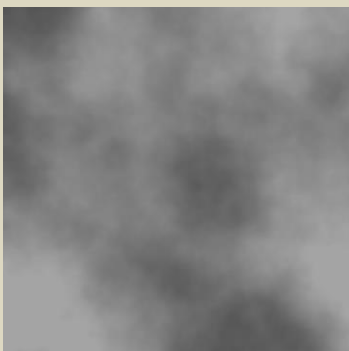
Controlling Output

```
var maximum: Float
```

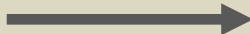
The maximum value of the range to clamp to. Values larger than this will be written to the output as this value. The default value is `1.0`.

```
var minimum: Float
```

The minimum value of the range to clamp to. Values smaller than this will be written to the output as this value. The default value is `0.0`.



Reducing range

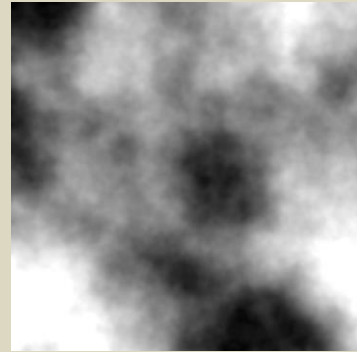
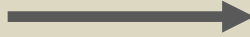


```
var normalise: Bool
```

Set this to `true` to remap the range of the output values back to `0.0` to `1.0` after clamping values. When set to `false` the range of the output values is minimum to maximum. The default value is `false`.



normalise = true



Relationships

Subclass of `AHNModifier`.

Conforms to the `AHNTextureProvider` protocol.

AHNModifierColour

Overview

The `AHNModifierColour` class maps a colour gradient to the noise values. A series of colours are positioned along the range 0.0 to 1.0, the colours are linearly interpolated between these positions. Each colour has an associated intensity that is also linearly interpolated between points, this intensity determines the mix ratio between the colour to be applied and the input pixel colour. The pixel value is then replaced with the corresponding colour from the colour gradient.

Symbols

Initialisers

```
init()
```

Initialises an `AHNModifierColour` object with default property values.

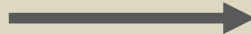
Controlling Output

```
var colours: [(colour: UIColor, position: Float, intensity: Float)]
```

The colours and associates positions and intensities that will be applied to the input texture. This is stored as an array of tuples. The default value is a single white colour with a position of 0.5 and an intensity of 1.0.

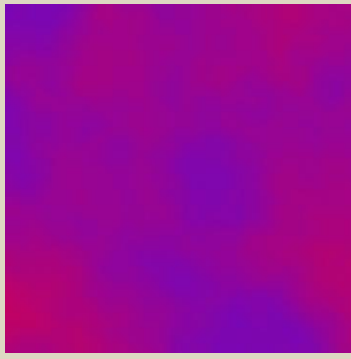


A red colour with a position of 0.0 and a blue colour with a position of 1.0. Both with intensities of 1.0

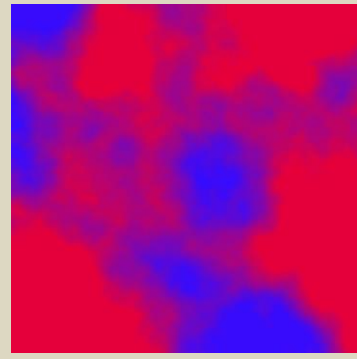


As above but with intensities of 0.5





As above but with intensities
of 1.0 again and positions of
0.4 and 0.5



```
func addColour(colour: UIColor, position: Float, intensity: Float)
```

Adds a new colour with an associated position and intensity.

```
func removeAllColours()
```

Removes all colours and replaces them with the default white value.

Relationships

Subclass of NSObject.

Conforms to the AHNTextureProvider protocol.

AHNModifierInvert

Overview

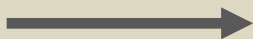
The `AHNModifierInvert` class replaces pixel values with their inverted counterpart. Pixel values in the range of 0.0 to 1.0 are remapped to noise values in the range of -1.0 to 1.0 this means values are multiplied by -1 , resulting in -0.3 becoming 0.3 , or 0.6 becoming -0.6 . These values are then returned to the 0.0 to 1.0 range.

Symbols

Initialisers

`init()`

Initialises an `AHNModifierInvert` object.



Relationships

Subclass of `AHNModifier`.

Conforms to the `AHNTextureProvider` protocol.

AHNModifierLoop

Overview

The `AHNModifierLoop` class replaces pixel values above a specified boundary with the value minus the boundary. For example a value of 0.3 with a boundary of 0.2 will result in an output value of 0.1, or a value of 0.5 with a boundary of 0.2 will result in 0.3, but as 0.3 is still above the boundary the process is repeated and results in an output value of 0.1 again. This is essentially a modulo function on the pixel value using the boundary.

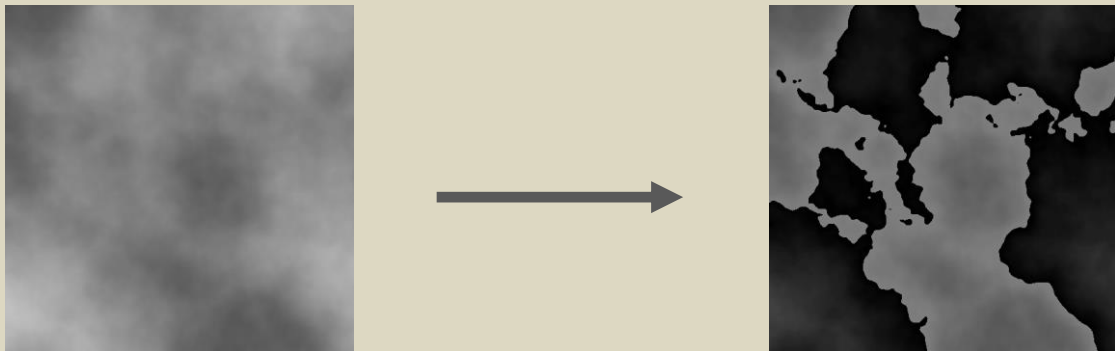
This output can be very dark for low boundary values, so it can be normalised to fill the range 0.0 to 1.0 again by setting the `normalise` property to `true`.

Symbols

Initialisers

`init()`

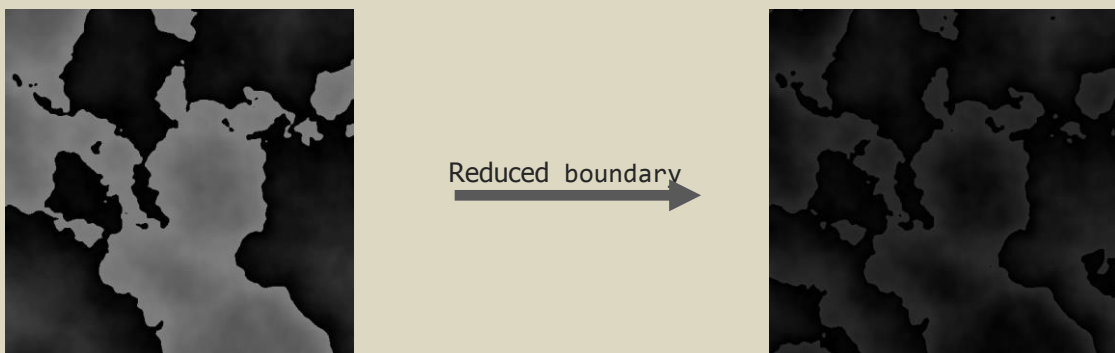
Initialises an `AHNModifierLoop` object with default property values.



Controlling Output

`var boundary: Float`

The upper limit of noise values before looping back to 0.0. The default value is 1.0.

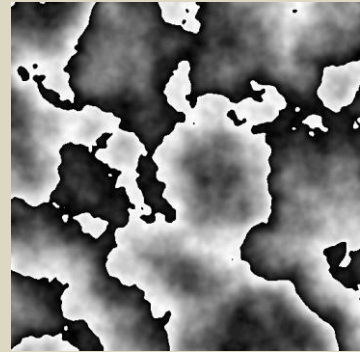


```
var normalise: Bool
```

Set this to true to remap the range of the output values back to 0.0 to 1.0 after looping values. When set to false the range of the output values is 0.0 to boundary. The default value is false.



normalise = true
→



Relationships

Subclass of `AHNModifier`.

Conforms to the `AHNTextureProvider` protocol.

AHNModifierPerspective

Overview

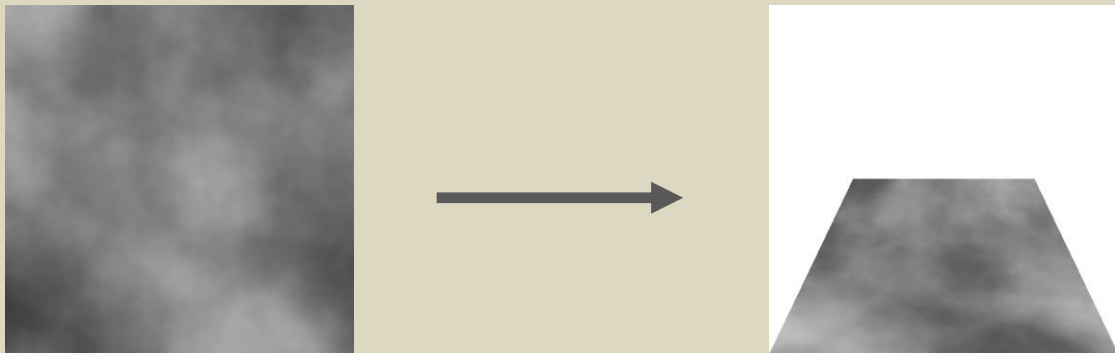
The `AHNModifierPerspective` class distorts the input texture to give the appearance of it receding away from the point of view. This can be useful when combining textures to give the illusion of depth in the texture.

Symbols

Initialisers

```
init()
```

Initialises an `AHNModifierPerspective` object with default property values.



Controlling Output

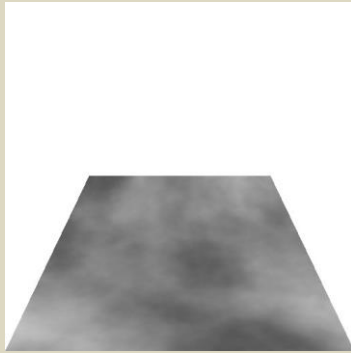
```
var xCompression: Float
```

Compresses the upper edge of the texture, converting the texture from a square into an isosceles trapezoid. The default value is `0.5`.

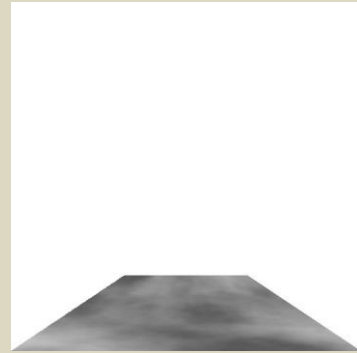


```
var yScale: Float
```

Compresses the texture vertically to simulate the angle of incidence. The default value is 0.5.

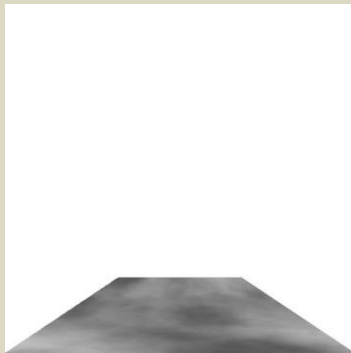


Decreased yScale
→

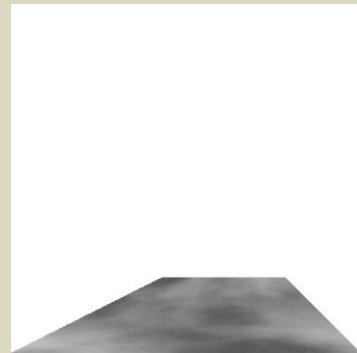


```
var direction: Float
```

Shears the texture left for negative values and right for positive values to simulate the position relative to the point of view. The default value is 0.0.



Increased direction
→



Relationships

Subclass of `AHNModifier`.

Conforms to the `AHNTextureProvider` protocol.

AHNModifierRotate

Overview

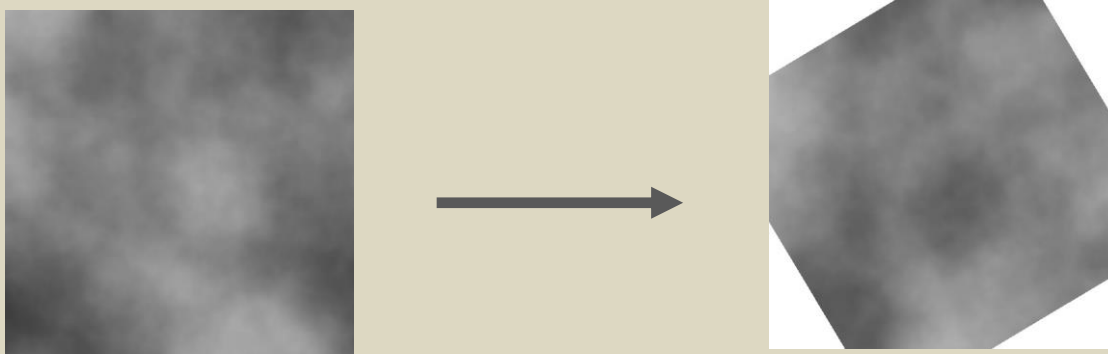
The `AHNModifierRotate` class rotates the input texture about a given centre. Pixels beyond the bounds of the original texture can be optionally clamped to edge values or cut away.

Symbols

Initialisers

```
init()
```

Initialises an `AHNModifierRotate` object with default property values.



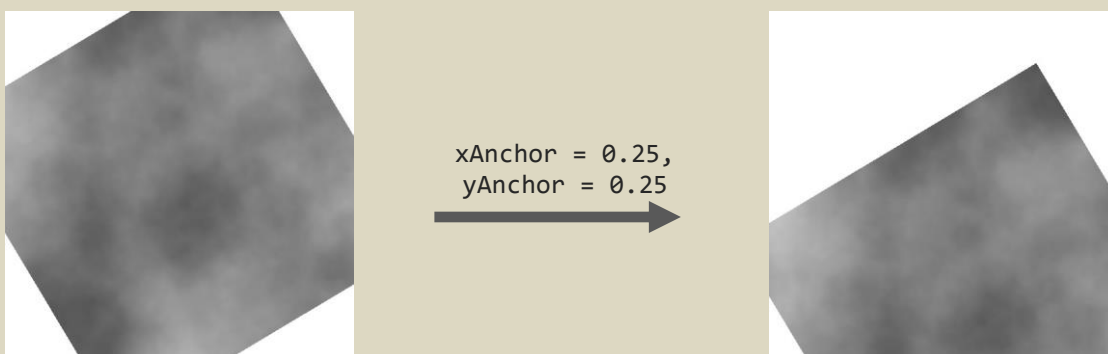
Controlling Output

```
var xAnchor: Float
```

The position along the X axis of the point about which the texture is rotated. `0.0` is the left edge and `1.0` is the right edge, the default value is `0.5`.

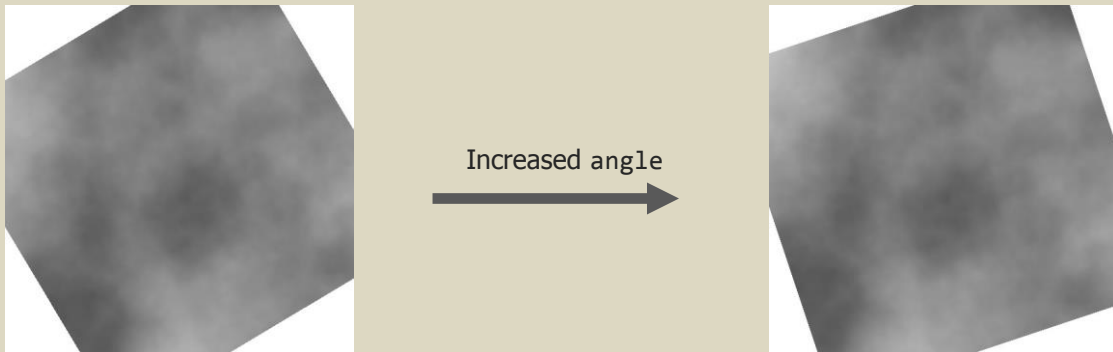
```
var yAnchor: Float
```

The position along the Y axis of the point about which the texture is rotated. `0.0` is the bottom edge and `1.0` is the top edge, the default value is `0.5`.



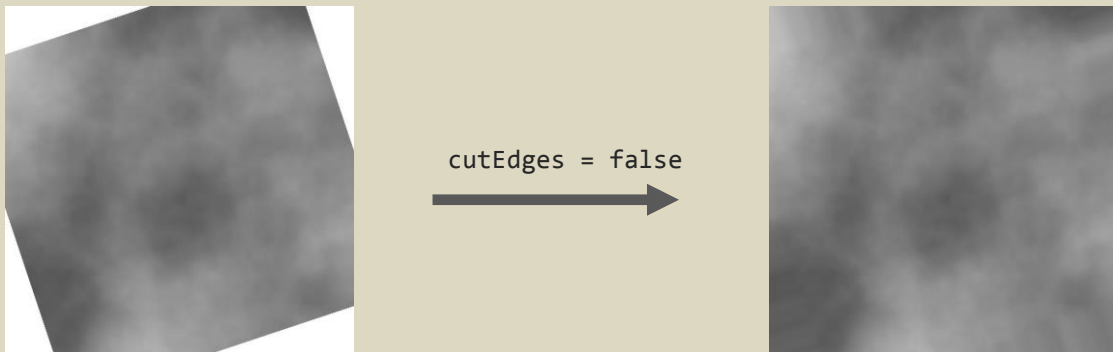

```
var angle: Float
```

The angle in radians to rotate the input texture clockwise around the point of rotation. The default value is 0.0 .



```
var cutEdges: Bool
```

When `true`, pixels beyond the extents of the input texture are left empty, if set to `false` these pixels take the value of the closest input texture pixel. The default value is `true`.



Relationships

Subclass of `AHNModifier`.

Conforms to the `AHNTextureProvider` protocol.

AHNModifierRound

Overview

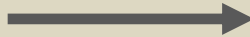
The `AHNModifierRound` class replaces pixel values with the nearest integer multiple of a specific value. For example pixel values of 0.2 and 0.5 with a `roundValue` of 0.3 would output 0.3 and 0.6 respectively.

Symbols

Initialisers

```
init()
```

Initialises an `AHNModifierRound` object with default property values.



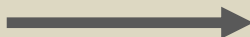
Controlling Output

```
var roundValue: Float
```

The value that pixel values are rounded to integer multiples of. The default value is 1.0.



Decreased `roundValue`



Relationships

Subclass of `AHNModifier`.

Conforms to the `AHNTextureProvider` protocol.

AHNModifierScaleBias

Overview

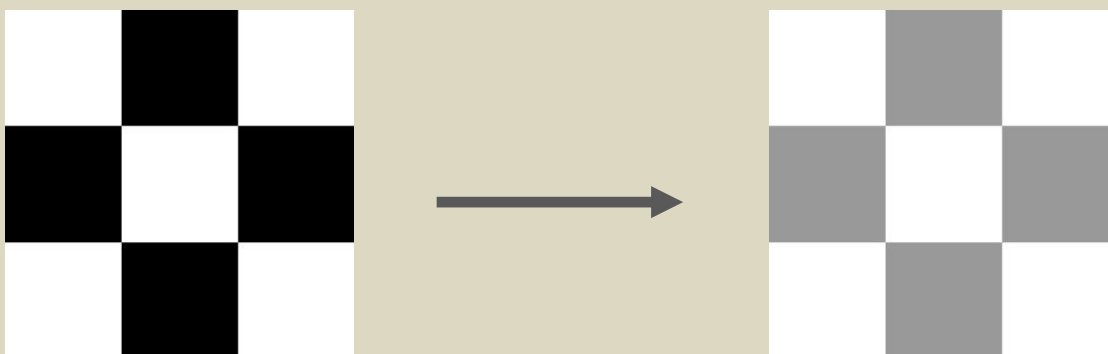
The `AHNModifierScaleBias` scales each pixel by a constant factor and adds a constant to it after the multiplication. This can be used to alter the range of pixel values while retaining the features of the texture. For example to remap a texture from the default 0.0 to 1.0 range to 0.6 to 1.0 you would apply a scale of 0.4 with a bias of 0.6.

Symbols

Initialisers

```
init()
```

Initialises an `AHNModifierScaleBias` object with default property values.



Controlling Output

```
var scale: Float
```

The value each pixel value is multiplied by prior to adding the bias value. The default value is 1.0.

```
var bias: Float
```

The value added to each pixel value after being multiplied by the `scale` value. The default value is 0.0.



Relationships

Subclass of `AHNModifier`.

Conforms to the `AHNTextureProvider` protocol.

AHNModifierStep

Overview

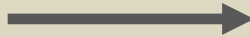
The `AHNModifierStep` class replaces pixel values higher than a specified boundary with a specified `highValue`, and values below the boundary with a specified `lowValue`.

Symbols

Initialisers

`init()`

Initialises an `AHNModifierStep` object with default property values.

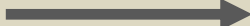


Controlling Output

`var` boundary: `Float`

Pixel values above this value are replaced with the `highValue`, those below this value are replaced with `lowValue`. the default value is 0.5.



Increased boundary


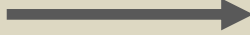


```
var lowValue: Float
```

The value to output when the input pixel value is below the boundary value. The default value is 0.0.



Increased lowValue

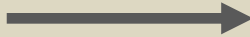


```
var highValue: Float
```

The value to output when the input pixel value is above the boundary value. The default value is 1.0.



Decreased highValue



Relationships

Subclass of AHNModifier.

Conforms to the AHNTexureProvider protocol.

AHNModifierStretch

Overview

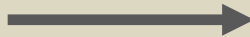
The `AHNModifierStretch` class stretches or compresses the input texture by user defined factors in the X and Y axes about a central point.

Symbols

Initialisers

```
init()
```

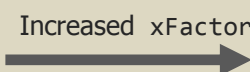
Initialises an `AHNModifierStretch` object with default property values.



Controlling Output

```
var xFactor: Float
```

The factor to stretch the texture by in the X axis. The default value of `1.0` results in no stretching, values less than `1.0` result in compression and larger than `1.0` result in stretching.



```
var yFactor: Float
```

The factor to stretch the texture by in the Y axis. The default value of 1.0 results in no stretching, values less than 1.0 result in compression and larger than 1.0 result in stretching.

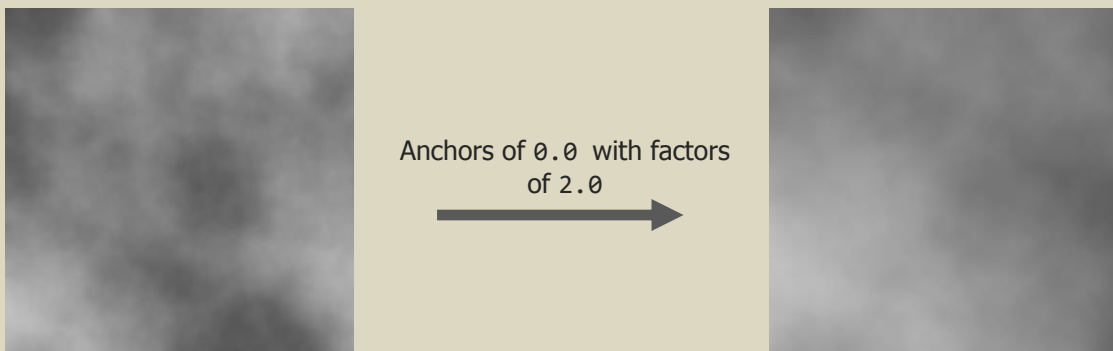


```
var xAnchor: Float
```

The position on the X axis about which the stretch takes place. Pixels to the left of this point are stretched to the left and vice versa. A value of 0.0 corresponds to the left hand edge and 1.0 corresponds to the right hand edge. The default value is 0.5.

```
var yAnchor: Float
```

The position on the Y axis about which the stretch takes place. Pixels above this point are stretched upwards and vice versa. A value of 0.0 corresponds to the bottom edge and 1.0 corresponds to the top edge. The default value is 0.5.



Relationships

Subclass of AHNModifier.

Conforms to the AHNTextureProvider protocol.

AHNModifierSwirl

Overview

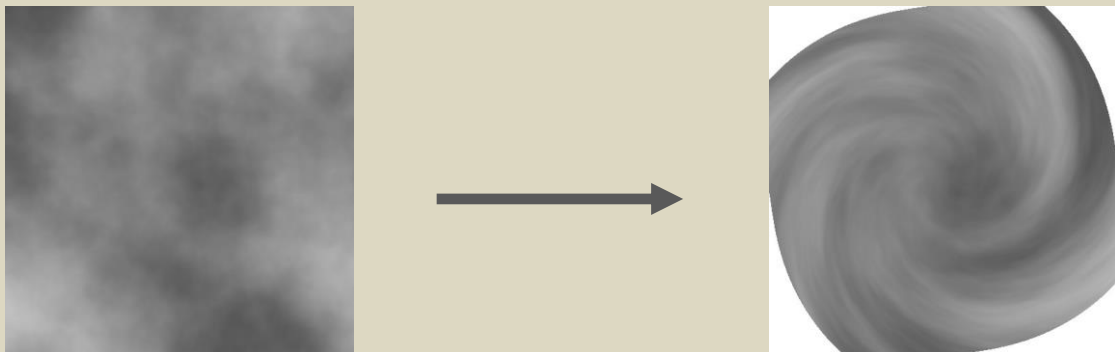
The `AHNModifierSwirl` class deforms the input texture by rotating each pixel about a central point by an angle proportional to the pixels distance from that point. This results in a vortex distortion of the input texture which can be used to model procedural galaxies or tornadoes.

Symbols

Initialisers

```
init()
```

Initialises an `AHNModifierSwirl` object with default property values.



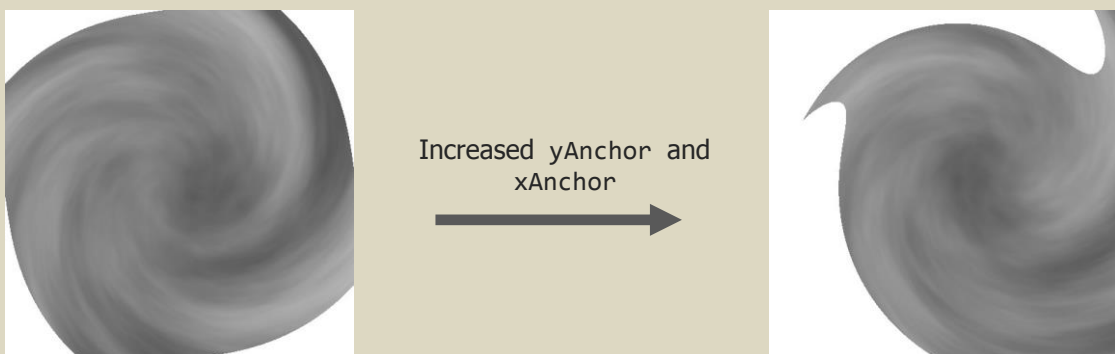
Controlling Output

```
var xAnchor: Float
```

The position on the X axis about which the swirl takes place. A value of `0.0` corresponds to the left hand edge and `1.0` corresponds to the right hand edge. The default value is `0.5`.

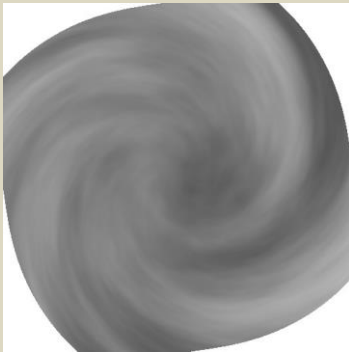
```
var yAnchor: Float
```

The position on the Y axis about which the swirl takes place. A value of `0.0` corresponds to the bottom edge and `1.0` corresponds to the top edge. The default value is `0.5`.

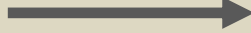



```
var intensity: Float
```

The intensity of the swirl, higher values result in more distortion. Positive values swirl the texture in a clockwise direction. The default value is 0.5.



Increased intensity

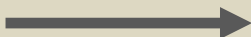


```
var cutEdges: Bool
```

When true, pixels beyond the extents of the input texture are left empty, if set to false these pixels take the value of the closest input texture pixel. The default value is true.



cutEdges = false



Relationships

Subclass of AHNModifier.

Conforms to the AHNTextureProvider protocol.

AHNModifierMapNormal

Overview

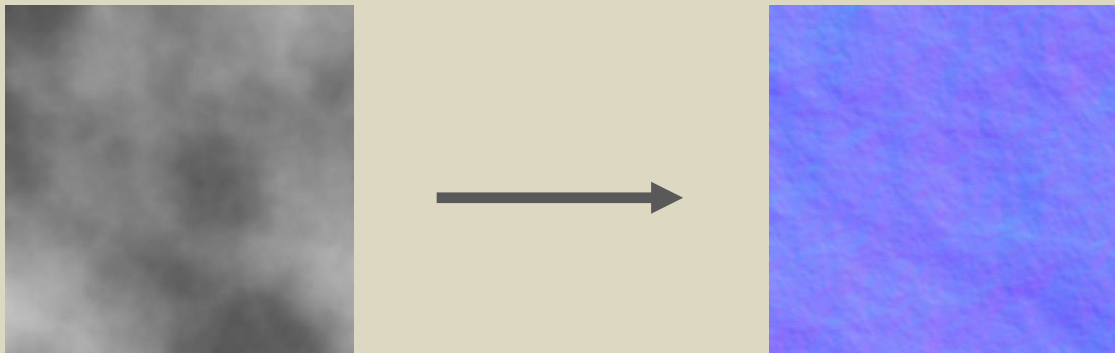
The `AHNModifierMapNormal` class generates a normal map for the input texture by analysing each pixels surroundings and generating a gradient from the data.

Symbols

Initialisers

```
init()
```

Initialises an `AHNModifierMapNormal` object with default property values.



Persistent Metal Objects

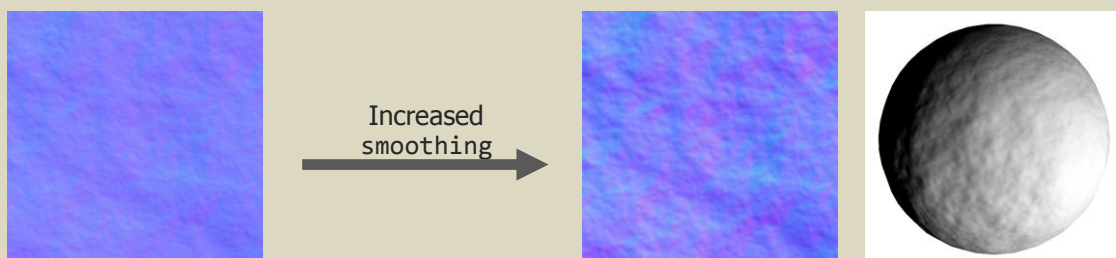
```
var context: AHNContext
```

The `AHNContext` that is being used by the `AHNModifierMapNormal` to communicate with the GPU. This is taken from the first `AHNGenerator` in the input chain which in turn is taken from the `SharedContext` class property of `AHNContext`.

Controlling Output

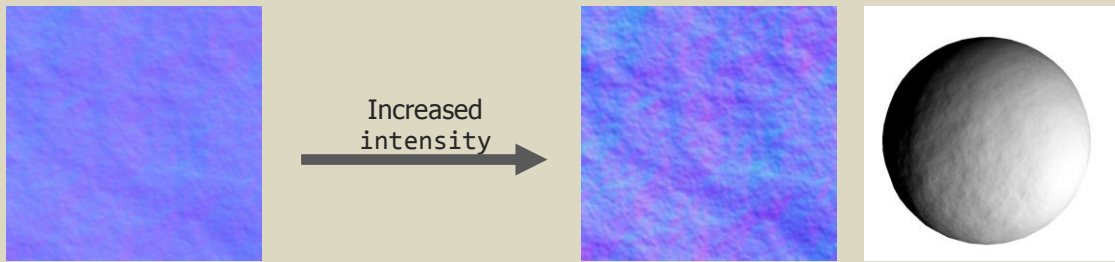
```
var smoothing: Float
```

A value in the range `0.0` to `1.0` indicating how much the input texture should be smoothed before the normal map is generated. A value of `0.0` indicates no smoothing. The default value is `0.0`.



```
var intensity: Float
```

A value that magnifies the effect of the generated normal map. The default value of 1.0 indicates no magnification.



```
var textureWidth: Int
```

The width of the output MTLTexture in pixels. This is taken from the provider property's textureWidth value.

```
var textureHeight: Int
```

The height of the output MTLTexture in pixels. This is taken from the provider property's textureHeight value.

```
var dirty: Bool
```

Indicates whether or not the output texture needs updating.

Relationships

Subclass of AHNModifier.

Conforms to the AHNTextureProvider protocol.

AHNModifierScaleCanvas

Overview

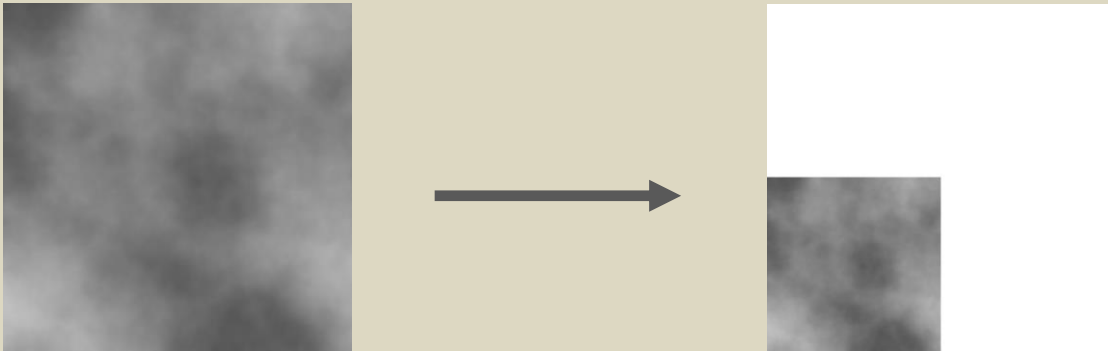
The `AHNModifierScaleCanvas` class places the input texture inside a new resized texture with the option of scaling the input texture and repositioning it within the new texture.

Symbols

Initialisers

```
init()
```

Initialises an `AHNModifierScaleCanvas` object with default property values.



Persistent Metal Objects

```
var context: AHNContext
```

The `AHNContext` that is being used by the `AHNModifierScaleCanvas` to communicate with the GPU. This is taken from the first `AHNGenerator` in the input chain, which in turn is taken from the `SharedContext` class property of `AHNContext`.

```
var uniformBuffer: MTLBuffer?
```

The `MTLBuffer` used to communicate the constant values used by the compute kernel to the GPU. This is encoded into the command buffer at the index 0.

Texture Properties

```
var textureWidth: Int
```

The width of the new output `MTLTexture` in pixels. This is taken from the provider property's `textureWidth` value.

```
var textureHeight: Int
```

The height of the new output `MTLTexture` in pixels. This is taken from the provider property's `textureHeight` value.

```
var dirty: Bool
```

Indicates whether or not the output texture needs updating.

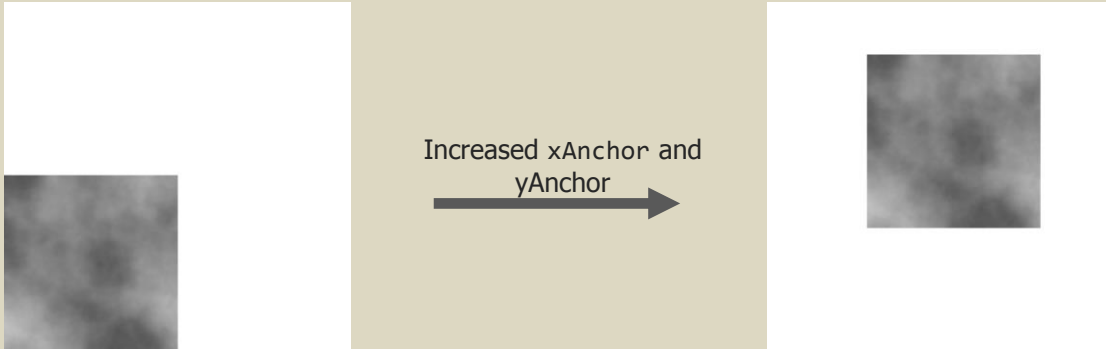
Controlling Output

`var xAnchor: Float`

The position along the horizontal axis of the bottom left corner of the input in the new canvas. Ranges from 0.0 for far left to 1.0 for far right, though values beyond this can be used. Default value is 0.0.

`var yAnchor: Float`

The position along the vertical axis of the bottom left corner of the input in the new canvas. Ranges from 0.0 for the bottom to 1.0 for the top, though values beyond this can be used. Default value is 0.0.

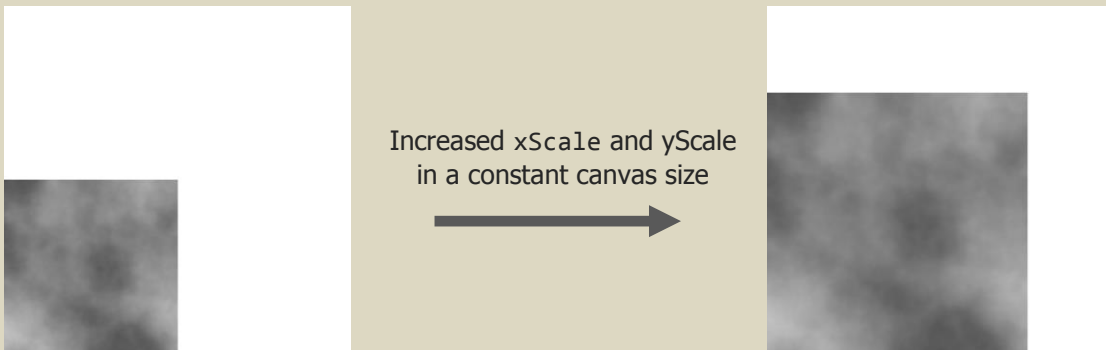


`var xScale: Float`

The scale of the input when inserted into the canvas. If an input had a width of 256, which is being resized to 512 with a scale of 0.5, the width of the input would be 128 in the canvas of 512. Default value is 1.0.

`var yScale: Float`

The scale of the input when inserted into the canvas. If an input had a height of 256, which is being resized to 512 with a scale of 0.5, the height of the input would be 128 in the canvas of 512. Default value is 1.0.



Relationships

Subclass of `AHNModifier`.

Conforms to the `AHNTextureProvider` protocol.

AHNCombiner

The AHNCombiner class provides an abstract superclass for all modules that combine the output textures of two other modules. It handles the binding of textures and buffers to the Metal command encoders leaving subclasses to handle which compute kernel to use and which properties to bind to the command encoder. It also handles the creation and updating of textures to represent the input arguments.

Symbols

Initialisers

```
init(functionName: String)
```

Initialises an AHNCombiner object that will use a Metal function with the specified functionName to create the output texture. This is called by subclasses to create a combiner class that implements a specific Metal function.

```
init()
```

Initialises an AHNCombiner object with a dummy function.

Persistent Metal Objects

```
var context: AHNContext
```

The AHNContext that is being used by the AHNCombiner to communicate with the GPU. This is taken from the first AHNGenerator in the input chain which in turn is taken from the SharedContext class property of AHNContext.

```
var uniformBuffer: MTLBuffer?
```

The MTLBuffer used to communicate the constant values used by the compute kernel to the GPU. This is encoded into the command buffer at the index 0. Most AHNCombiner modules perform mathematical functions and do not make use of this buffer.

Additional Inputs

```
var provider2: AHNTextureProvider?
```

In addition to the standard textureProvider() function adopted by all modules, combiners accept a second provider module to combine with the first. Switching round the inputs such that provider becomes provider2 and vice versa may produce alternative results.

Texture Properties

```
var textureWidth: Int
```

The width of the output MTLTexture in pixels. This is taken as the maximum of provider or provider2 properties textureWidth value.

```
var textureHeight: Int
```

The height of the output MTLTexture in pixels. This is taken from the maximum of provider or provider2 properties textureHeight value.

```
var dirty: Bool
```

Indicates whether or not the output texture needs updating.

Texture Creation and Updating

```
func configureArgumentTableWithCommandencoder(commandEncoder: MTLComputeCommandEncoder)
```

This function is overridden by subclasses to write class specific variables to the uniformBuffer.

Parameter: commandEncoder – The MTLComputeCommandEncoder used to encode the kernel. This can be used to lazily create a buffer of data and add it to the argument table. Any buffer index can be used without affecting the rest of this class.

Relationships

Subclass of NSObject.

Conforms to the AHNTextureProvider protocol.

AHNCombinerAdd

Overview

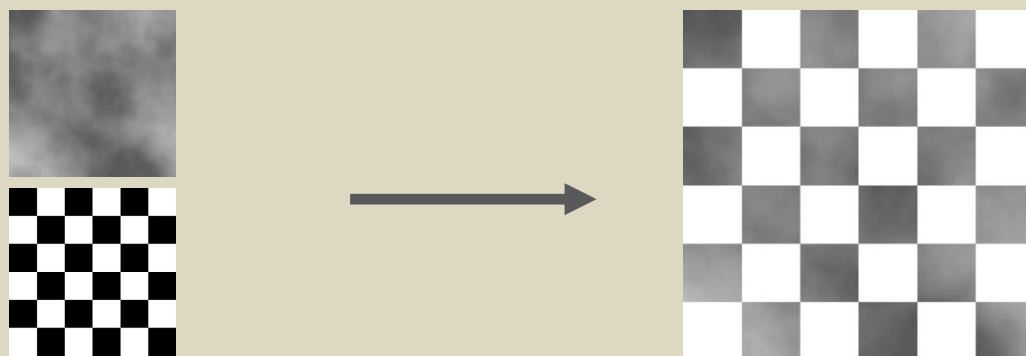
The AHNCombinerAdd mathematically sums each pixel value like for like. As this can result in pixel values above 1.0 which will always show as white, there is the option to normalise the addition, dividing the result by 2.0 to ensure the output is within the range 0.0-1.0.

Symbols

Initialisers

```
init()
```

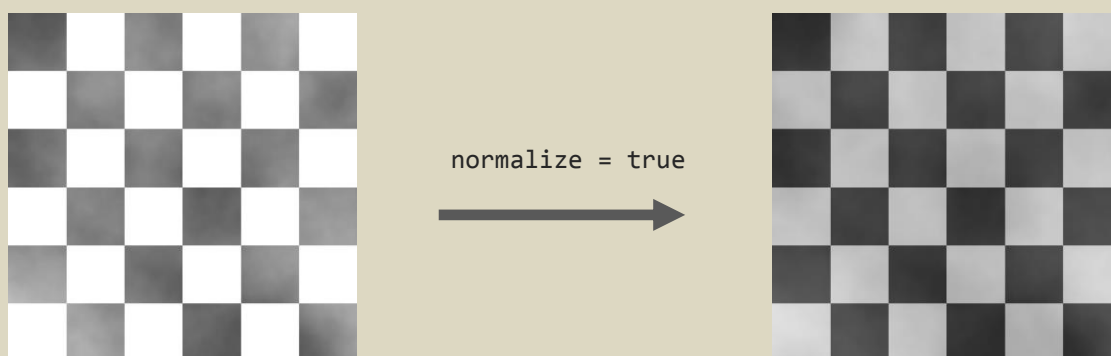
Initialises an AHNCombinerAdd object with default property values.



Controlling Output

```
var normalise: Bool
```

When set to true (false by default) the output value range is remapped back to 0.0 - 1.0 to prevent overly bright areas where the combination of inputs has exceeded 1.0. Setting this to true results in the output being the average of the two inputs.



Relationships

Subclass of AHNCombiner.

Conforms to the AHNTextureProvider protocol.

AHNCombinerDivide

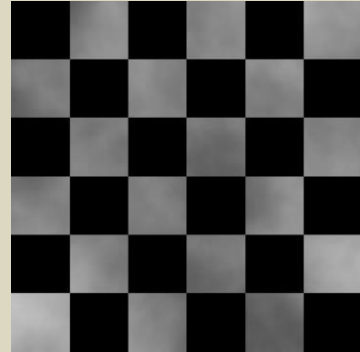
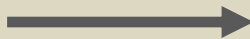
The AHNCombinerDivide mathematically divides the pixel value of the provider by the corresponding pixel value from provider2.

Symbols

Initialisers

`init()`

Initialises an AHNCombinerDivide object.



Relationships

Subclass of AHNCombiner.

Conforms to the AHNTextureProvider protocol.

AHNCombinerMax

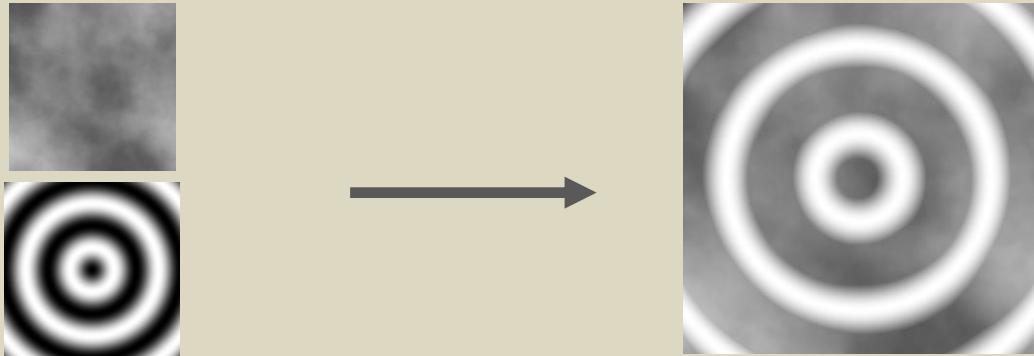
The AHNCombinerMax outputs the maximum pixel value from either the provider or provider2.

Symbols

Initialisers

`init()`

Initialises an AHNCombinerMax object.



Relationships

Subclass of AHNCombiner.

Conforms to the AHNTextureProvider protocol.

AHNCombinerMin

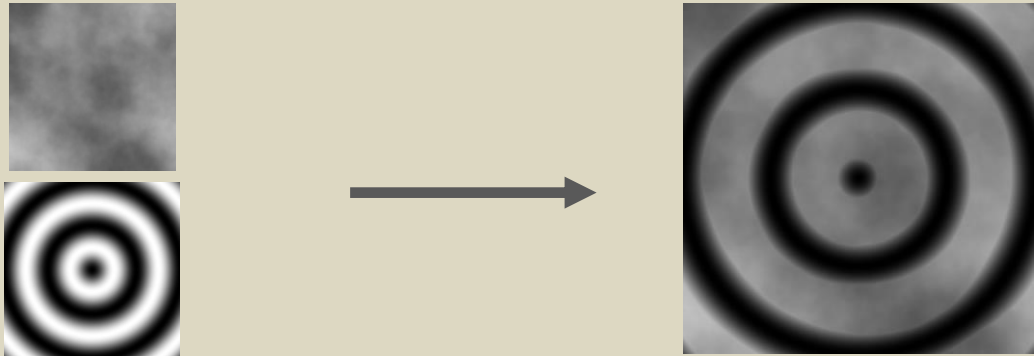
The AHNCombinerMin outputs the minimum pixel value from either the provider or provider2.

Symbols

Initialisers

`init()`

Initialises an AHNCombinerMin object.



Relationships

Subclass of AHNCombiner.

Conforms to the AHNTexureProvider protocol.

AHNCombinerMultiply

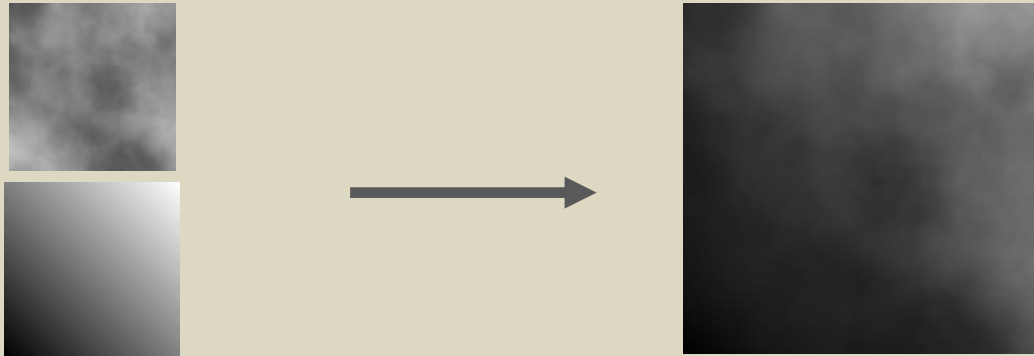
The AHNCombinerMultiply outputs the multiple of the pixel values from provider and provider2.

Symbols

Initialisers

`init()`

Initialises an AHNCombinerMultiply object.



Relationships

Subclass of AHNCombiner.

Conforms to the AHNTextureProvider protocol.

AHNCombinerPower

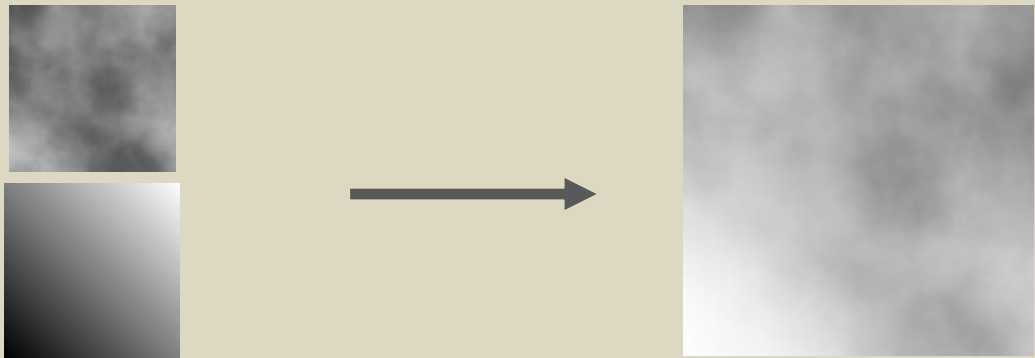
The AHNCombinerPower outputs the result of raising the pixel values of provider to the power of the corresponding pixel values of the provider2.

Symbols

Initialisers

`init()`

Initialises an AHNCombinerPower object.



Relationships

Subclass of AHNCombiner.

Conforms to the AHNTextureProvider protocol.

AHNCombinerSubtract

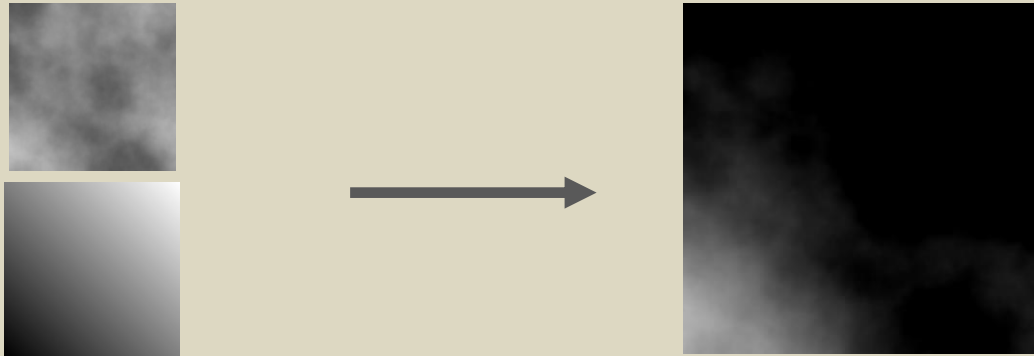
The AHNCombinerSubtract outputs the result of subtracting the pixel values of provider2 from the provider.

Symbols

Initialisers

`init()`

Initialises an AHNCombinerSubtract object.



Relationships

Subclass of AHNCombiner.

Conforms to the AHNTexureProvider protocol.

AHNSelector

The AHNSelector class provides an abstract superclass for all modules that combine the output textures of two other modules using a third selector module. A blend of the pixel values from the provider and provider2 are output using a weighting provided by the selector pixel values. It handles the binding of textures and buffers to the Metal command encoders leaving subclasses to handle which compute kernel to use and which properties to bind to the command encoder. It also handles the creation and updating of textures to represent the input arguments.

Symbols

Initialisers

```
init(functionName: String)
```

Initialises an AHNSelector object that will use a Metal function with the specified functionName to create the output texture. This is called by subclasses to create a selector class that implements a specific Metal function.

```
init()
```

Initialises an AHNSelector object with a dummy function.

Persistent Metal Objects

```
var context: AHNContext
```

The AHNContext that is being used by the AHNSelector to communicate with the GPU. This is taken from the first AHNGenerator in the input chain which in turn is taken from the SharedContext class property of AHNContext.

```
var uniformBuffer: MTLBuffer?
```

The MTLBuffer used to communicate the constant values used by the compute kernel to the GPU. This is encoded into the command buffer at the index 0.

Additional Inputs

```
var provider2: AHNTextureProvider?
```

In addition to the standard textureProvider() function adopted by all modules, selectors accept a second provider module to combine with the first.

```
var selector: AHNTextureProvider?
```

In addition to the standard textureProvider() function adopted by all modules and the provider2 property, selectors accept a third provider module to use a weighting when blending the provider and provider2.

Texture Properties

```
var textureWidth: Int
```

The width of the output MTLTexture in pixels. This is taken as the maximum of the provider, provider2 or selector properties' textureWidth value.

```
var textureHeight: Int
```

The height of the output MTLTexture in pixels. This is taken from the maximum of the provider, provider2 or selector properties' textureHeight value.

```
var dirty: Bool
```

Indicates whether or not the output texture needs updating.

Texture Creation and Updating

```
func configureArgumentTableWithCommandencoder(commandEncoder: MTLComputeCommandEncoder)
```

This function is overridden by subclasses to write class specific variables to the uniformBuffer.

Parameter: commandEncoder – The MTLComputeCommandEncoder used to encode the kernel. This can be used to lazily create a buffer of data and add it to the argument table. Any buffer index can be used without affecting the rest of this class.

Relationships

Subclass of NSObject.

Conforms to the AHNTextureProvider protocol.

AHNSelectorBlend

Overview

The AHNSelecorBlend blends two inputs together using a weight from a third input used as the selector.

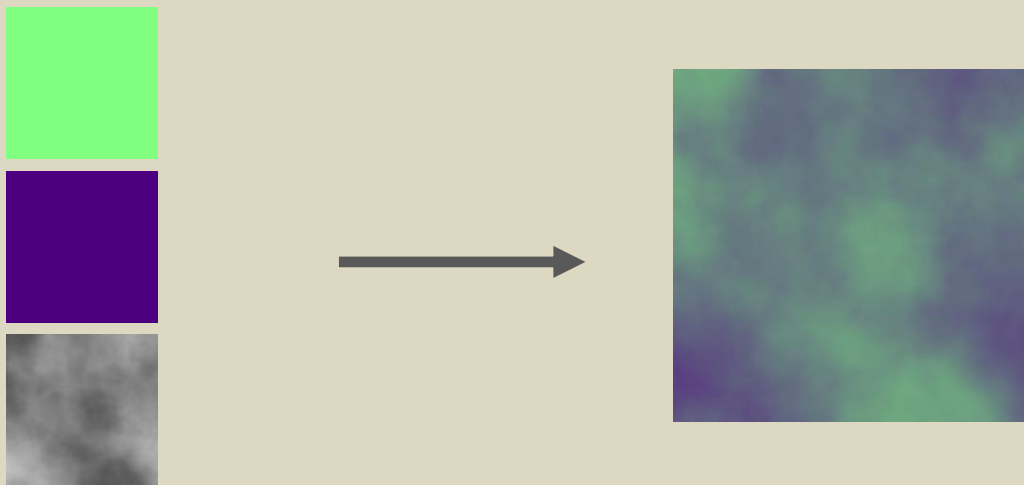
The input AHNTTextureProviders may range from a value of 0.0 - 1.0. This value is taken from the selector for each pixel to provide a mixing weight for the two providers. A value of 0.0 will output 100% provider and 0% provider2, while a value of 1.0 will output 100% provider2 and 0% provider. A value of 0.25 will output a mixture of 75% provider and 25% provider2.

Symbols

Initialisers

`init()`

Initialises an AHNSelectorBlend object with default property values.



Relationships

Subclass of AHNSelector.

Conforms to the AHNTTextureProvider protocol.

AHNSelectorSelect

The `AHNSelectorSelect` to write to the output using a weight from a third input used as the selector.

The input providers may range from a value of `0.0-1.0`. This value is taken from the selector for each pixel to select which input to write to the output `MTLTexture`. A selector value between `0.0-boundary` will result in provider being written to the output, whereas a selector value between `boundary-1.0` will result in provider2 being written to the output.

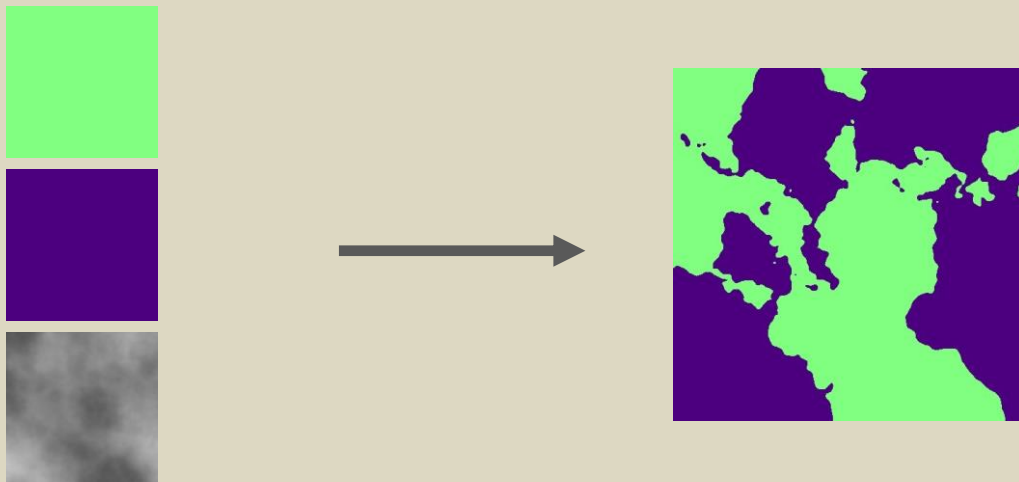
The `edgeTransition` property is used to define how abruptly the transition occurs between the two inputs. A value of `0.0` will result in no transition or a step. Higher values cause the transition to be softened by interpolating between the two inputs at the border between them. A maximum value of `1.0` results in the edge transition covering the whole of the two inputs.

Symbols

Initialisers

`init()`

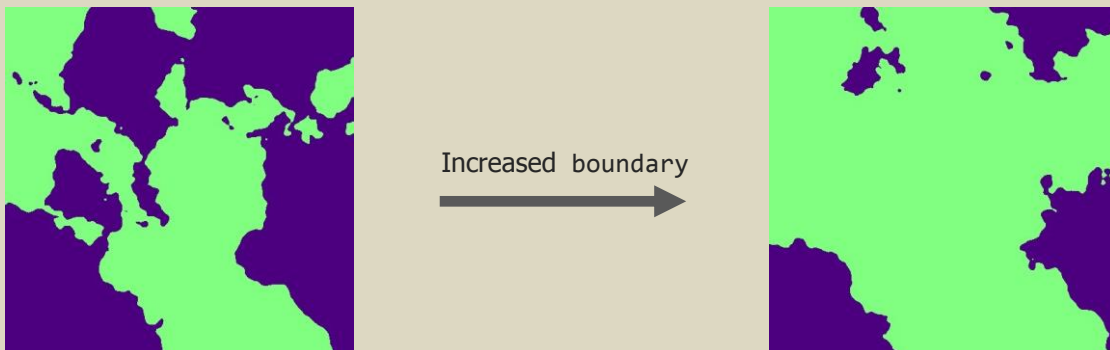
Initialises an `AHNSelectorSelect` object with default property values.



Controlling Output

`var` boundary: `Float`

The boundary that the selector value is compared to. Selector values larger than this boundary will output provider2, and less than this will output provider. The default value is `0.5`.



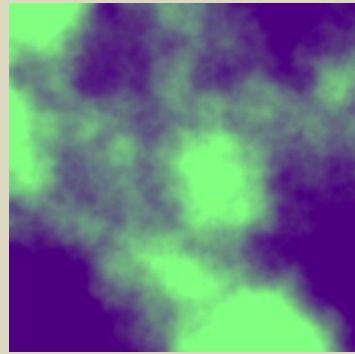
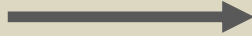
```
var transition: Float
```

The amount the transition between the two inputs should be softened `(0.0-1.0)`. Values outside the range `(0.0-1.0)` may result in undesired behavior.

Default value is `0.0`.



Increased transition



Relationships

Subclass of `AHNSelector`.

Conforms to the `AHNTextureProvider` protocol.

Custom Modules

You can create custom modules easily by subclassing one of the following classes:

- AHNGenerator / AHNGeneratorCoherent
- AHNModifier
- AHNCombiner
- AHNSelector

The configuration required to execute a module is relatively simple. Each of the above classes has a uniform buffer into which class properties and variables can be bound. This can be done using a simple Struct of those variables and then writing them into the uniform buffer provided by the superclass. This is done by overriding the `configureArgumentTableWithCommandencoder` function that is present in all of the above classes..

As well as this you will need to write a Metal compute function for execution on the GPU. This function MUST have the following basic declaration in order to successfully bind the various buffers/textures for use in the kernel.

In the below declarations `functionName` must match the name given when calling `super.init(functionName)`. `FunctionName` is a type (primitive or struct) that must be declared both in the metal file in the Metal Shader Language and the class implementation. The values bound to the `uniformBuffer` must be of this type.

AHNGenerator

```
kernel void functionName(texture2d<float, access::write> outTexture [[texture(0)]],
                        texture2d<float, access::read> xoffset [[texture(1)]],
                        texture2d<float, access::read> yoffset [[texture(2)]],
                        constant FunctionInputs &uniforms [[buffer(0)]],
                        uint2 gid [[thread_position_in_grid]],
{};
```

AHNModifier

```
kernel void functionName(texture2d<float, access::read> inTexture [[texture(0)]],
                        texture2d<float, access::write> outTexture [[texture(1)]],
                        constant FunctionInputs &uniforms [[buffer(0)]],
                        uint2 gid [[thread_position_in_grid]],
{};
```

AHNCombiner

```
kernel void functionName(texture2d<float, access::read> inTexture1 [[texture(0)]],
                        texture2d<float, access::read> inTexture2 [[texture(1)]],
                        texture2d<float, access::write> outTexture [[texture(2)]],
                        constant FunctionInputs &uniforms [[buffer(0)]],
                        uint2 gid [[thread_position_in_grid]])
{};
```

AHNSelector

```
kernel void functionName(texture2d<float, access::read> inTexture1 [[texture(0)]],
                        texture2d<float, access::read> inTexture2 [[texture(1)]],
                        texture2d<float, access::read> selector [[texture(2)]],
                        texture2d<float, access::write> outTexture [[texture(3)]],
                        constant FunctionInputs &uniforms [[buffer(0)]],
                        uint2 gid [[thread_position_in_grid]])
{};
```

If necessary it is possible to bind extra textures and buffers up to the maximum allowed by Apple. All of the classes included in AHNnoise have the above declarations. When binding custom textures and buffers, be careful not to overwrite one written by the superclass.

Below is the declaration of AHNCombinerAdd as an example.

Swift Implementation

```
class AHNCombinerAdd: AHNCombiner {
    var normalise: Bool = false{
        didSet{
            dirty = true
        }
    }

    required init(){
        super.init(functionName: "addCombiner")
    }

    override func configureArgumentTableWithCommandencoder(commandEncoder: MTLComputeCommandEncoder) {
        var uniforms = normalise

        if uniformBuffer == nil{
            uniformBuffer = context.device.newBufferWithLength(strideof(Bool), options: .CPUCacheModeDefaultCache)
        }

        memcpy(uniformBuffer!.contents(), &uniforms, strideof(Bool))

        commandEncoder.setBuffer(uniformBuffer, offset: 0, atIndex: 0)
    }
}
```

Metal Shader Language Function

```
kernel void addCombiner(texture2d<float, access::read> inTexture1 [[texture(0)]],
                        texture2d<float, access::read> inTexture2 [[texture(1)]],
                        texture2d<float, access::write> outTexture [[texture(2)]],
                        constant bool &uniforms [[buffer(0)]],
                        uint2 gid [[thread_position_in_grid]])
{
    float4 in1 = inTexture1.read(gid);
    float4 in2 = inTexture2.read(gid);
    float3 out = in1.rgb+in2.rgb;
    if (uniforms == true){
        out /= 2;
    }

    outTexture.write(float4(out,1), gid);
}
```