Organic Shader: Stone Generation

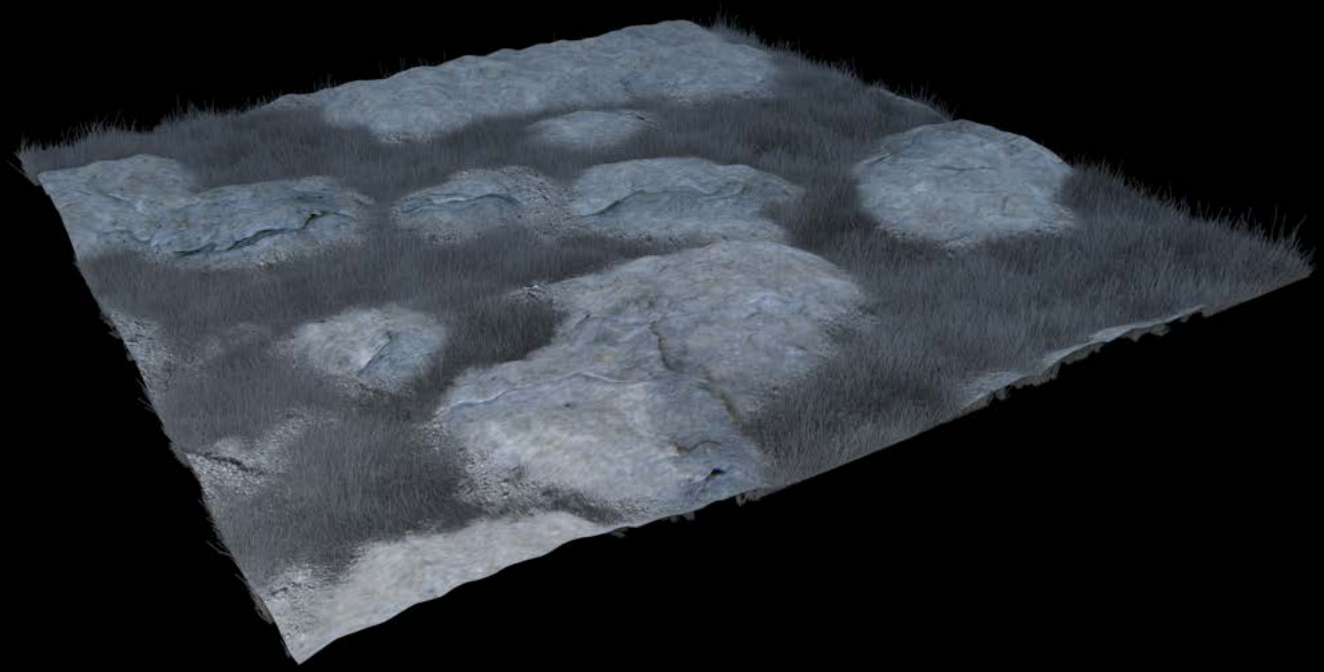
Cobbled.osl

George Arnold

2020

Contents

Description:
This is a full documentation of a shader I developed for creating organic looking cobbled stone surfaces. The concept is to create a field of circular cells that can be affected by noise along their radius as well as in their surface displacement. With almost every aspect of the pattern and displacement user controllable, this shader can be used for a massive variety of surfaces involving rocks or stone patterns. This shader can be used as is for a soft toon-like look, but gives the user sufficient information to create hyper-realistic surfaces as well.



Cobbled.osl

```
//=============================================
// Cobbled.osl
// Creating an Organic Noise Based Shader - Cobblestone
// George Arnold
// SCAD-Atlanta
// VSFX 319
// 5/15/2020
//
// This file creates noisey color patterns and displacement values and masks in order to simulate
// cobblestone or stone brick surfaces. It also can // manipulated to create various landscape textures,
// with masks provided for adding grass or other instances/simulations with ease.
//=============================================
float createCells
(
        point p,
        string spacename,
        float freq,
        float jitter
)
{
        //Creates circular cells based on the specified frequency
        point pp = transform(spacename, p) * freq;
        point thiscell = point(floor(pp[0]) + 0.5,
        floor(pp[1]) + 0.5,
        floor(pp[2]) + 0.5);
        float outCells = 1000;
        int i,j,k;

        for(i = -1; i <= 1; i += 1)
                for(j = -1; j <= 1; j += 1)
                        for(k = -1; k <= 1; k += 1) {
                        point testcell = thiscell + vector(i,j,k);
                        point pos = testcell + (noise("cell", testcell) * jitter) - 0.5;
                        float dist = distance(pos, pp);
                        if(dist < outCells)
                                outCells = dist;
        }
        return outCells;
}
color cellColorationStone
(
        point pIN,
```

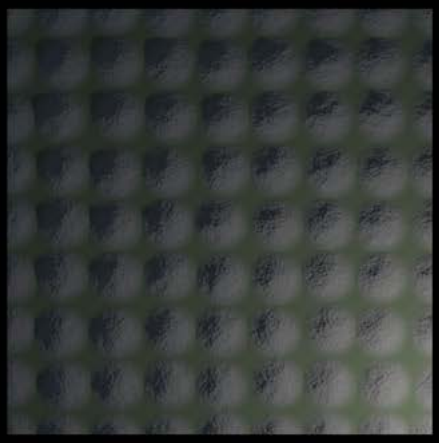
```
)
{
	//Colors the stones by mixing layers of noise with varying but similar color values
	color intHSV3 = color("hsv", 0.1472222222222222, 0.09, noise("usimplex", pIN * 8));
	color intHSV2 = color("hsv", 0.1472222222222222, 0.1, noise("usimplex", pIN * 1.5));
	color intHSV1 = color("hsv", 0.13166666666666667, 0.05, (noise("usimplex", pIN * 0.7)*0.6));
	color intHSVm = mix(intHSV1, intHSV2, 0.5);
	color intHSV = mix(intHSVm, intHSV3, 0.3);
	return intHSV;
}
color cellColorationBack
(
	point pIN,
)
{
	//Colors the background by mixing layers of noise with varying but similar color values
	color intHSV1 = color("hsv", 0.1083333333333333, noise("usimplex", pIN*2), 0.33);
	color intHSV2 = color("hsv", 0.4583333333333333, noise("usimplex", pIN*0.5), 0.37);
	color intHSV3 = color("hsv", 0.1083333333333333, noise("usimplex", pIN*10), 0.18);
	color intHSV4 = color("hsv", 0.117, 0.47, (noise("usimplex", pIN * 1.3)*0.6));
	color intHSV5 = color("hsv", 0.2861111111111111, noise("usimplex", pIN*6), 0.43);
	color intHSVm1 = mix(intHSV1, intHSV2, 0.5);
	color intHSVm2 = mix(intHSVm1, intHSV3, 0.3);
	color intHSVm3 = mix(intHSVm2, intHSV4, 0.3);
	color intHSV = mix(intHSVm3, intHSV5, 0.3);
	return intHSV;
}
float displacementValues
(
	point pIN,
	int multiplier,
	float rF
)
{
	//Create the values for stone displacement based on noise and the user-specified frequency
	float dispNum = noise("usimplex", pIN * multiplier);
	color noDisp = color(0,0,0);
	color yesDisp = color(dispNum, dispNum, dispNum);
	color disp = mix(noDisp, yesDisp, rF);
	return disp[0];
}

shader
Cobbled
```

```
(
        float cellfreq = 5,
        float radius = 0.15,
        float blur = 0.0,
        float dispBlur = 0.3,
        float jitter = 0.15,
        int Medium1_Freq = 20,
        int Medium2_Freq = 25,
        int Small_Freq = 80,
        string spacename = "shader",
        output float resultSmallDisp = 0,
        output float resultMediumDisp1 = 0,
        output float resultMediumDisp2 = 0,
        output color resultRGB = 0,
        output float resultMask = 0,
        output float resultBigDisp = 0,
        output float resultF = 0
)
{
        //Create the circular cells
        float d = createCells(P, spacename, cellfreq, jitter);

        //Create a mask for where the cells are, including their blur factors
        resultF = 1 - smoothstep(radius, radius + dispBlur + blur, d);

        //Create the base displacement value that rises the stones from the background
        resultBigDisp = 1 - smoothstep(radius - dispBlur, radius + dispBlur, d);

        //Create the coloration for the stones and background
        color intcolor = cellColorationStone(P);
        color bakcolor = cellColorationBack(P);

        //Create the noisey displacement values for the stones, to be blended together in software
        resultSmallDisp = displacementValues(P, Small_Freq, resultF);
        resultMediumDisp1 = displacementValues(P, Medium1_Freq, resultF);
        resultMediumDisp2 = displacementValues(P, Medium2_Freq, resultF);

        //Create the final color output and mask for where the stones are
        resultRGB = mix(bakcolor, intcolor, resultF);
        resultMask = (d <= radius) ? 1 : 0;
}
```

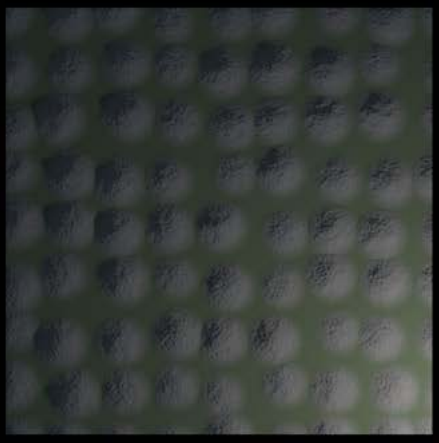
Parameters

Parameter	Type	Default Value	Description
cellfreq	float	5.0	Determines the number of cells spread across the surface
radius	float	0.15	Determines the radius of the individual cells (this should be controlled by an external noise or fractal value ex: PxrFractal)
blur	float	0.0	Blurs the edges of the cells, negative values sharpen
dispBlur	float	0.3	Blurs the edges of the displacement of the cell, values closer to 0 make the stones sharper where values closer to 1 make them more rounded
jitter	float	0.15	Offsets the cells from their center point at random, higher values == more variance
Medium1_Freq	int	20	Controls the frequency of a layer of displacement on the stones
Medium2_Freq	int	25	Controls the frequency of a layer of displacement on the stones
Small_Freq	int	80	Controls the frequency of a layer of displacement on the stones
stoneColor	color	HSV(53°, 10%, 50%)	Color of the stone, values influence the shade rather than color
backColor	color	HSV(53°, 10%, 50%)	Color of the background, values directly influence the color
spacename	string	“shader"	Assigns a name to the area of points where cells can be created, this does not need to be changed and does not have an effect on the user space

Jitter – Low values produce uniform results, higher values more random. Above 1.0 stones will become scarce and blend together and values higher than 1.5 tend to cause clipping issues.



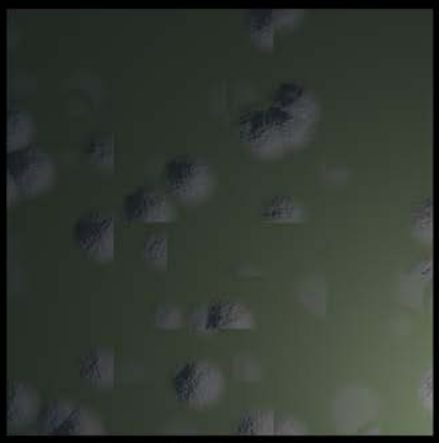
Jitter: 0.1



Jitter: 0.3

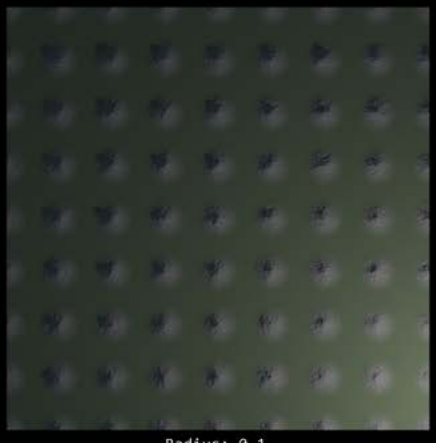
Jitter: 0.6

Jitter: 1.5

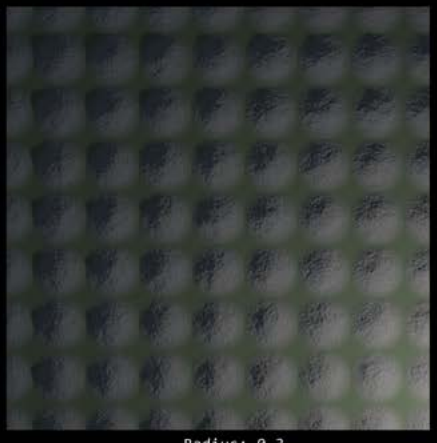


Jitter: 2.5

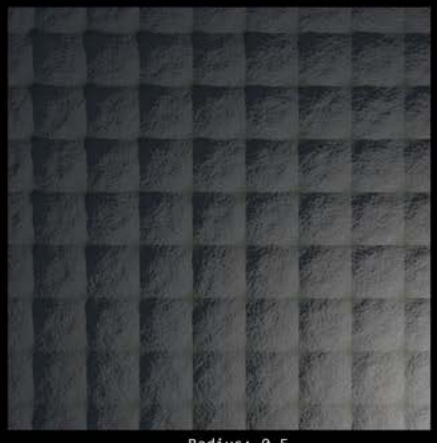
Radius – Adjusting the radius of the cells from 0.0-0.5, this parameter should in most cases be controlled using a clamped fractal value as seen on Pg. 16.



Radius: 0.1

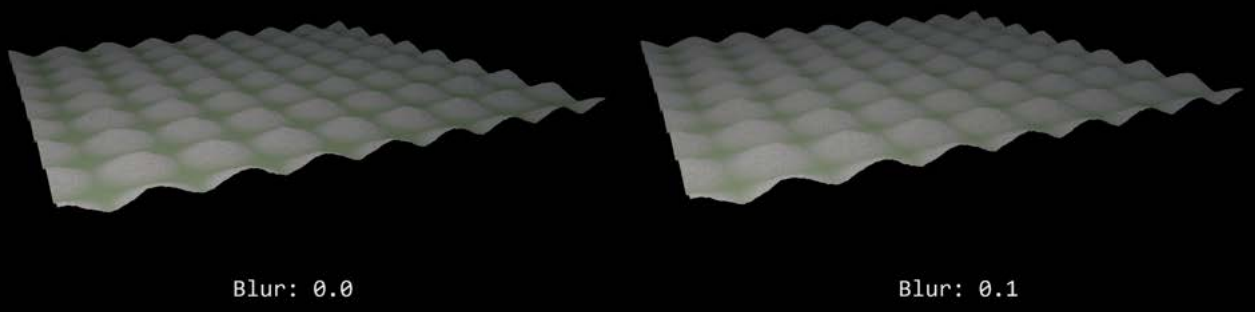


Radius: 0.3

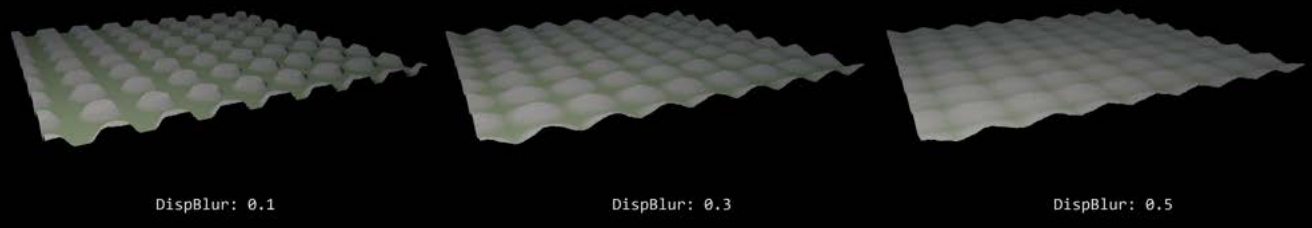


Radius: 0.5

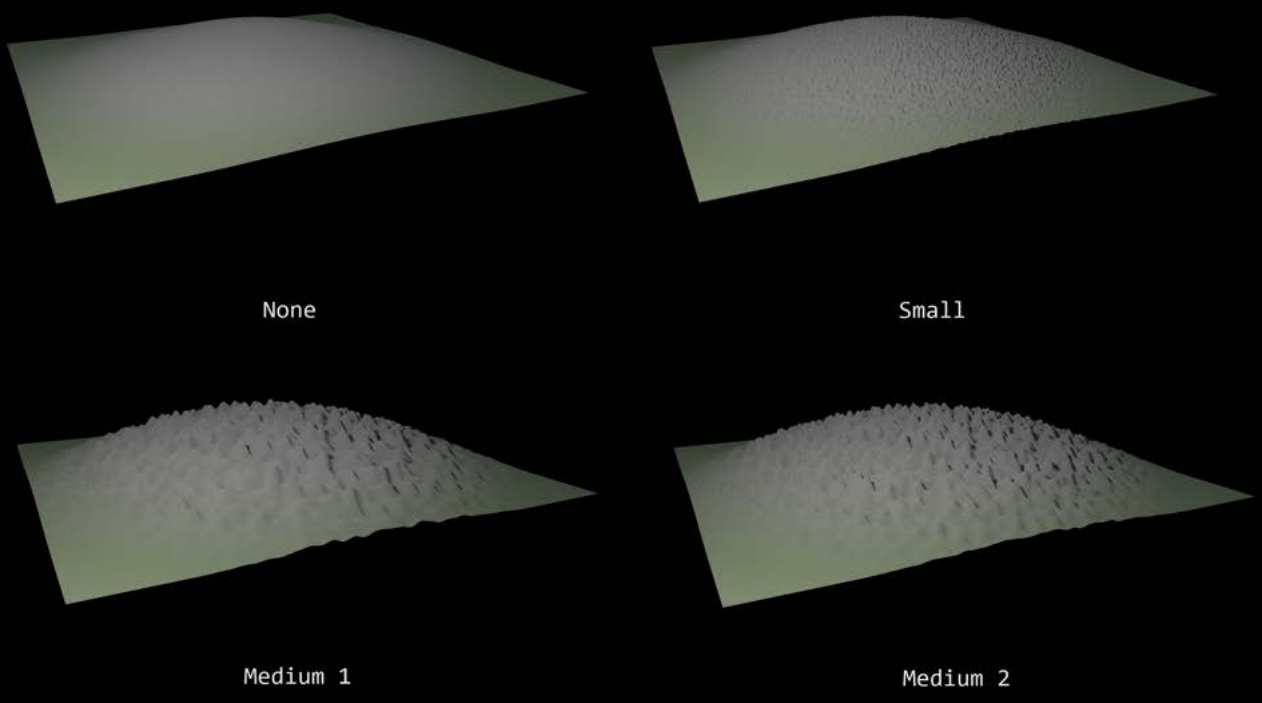
Blur – Coloration is blurred on the edge of the cells from 0.0-0.5, low values are best.



DispBlur – The main displacement is blurred on the edge of the cells from 0.0-0.5 creating a sharper look with lower values and a more rounded look with higher values.



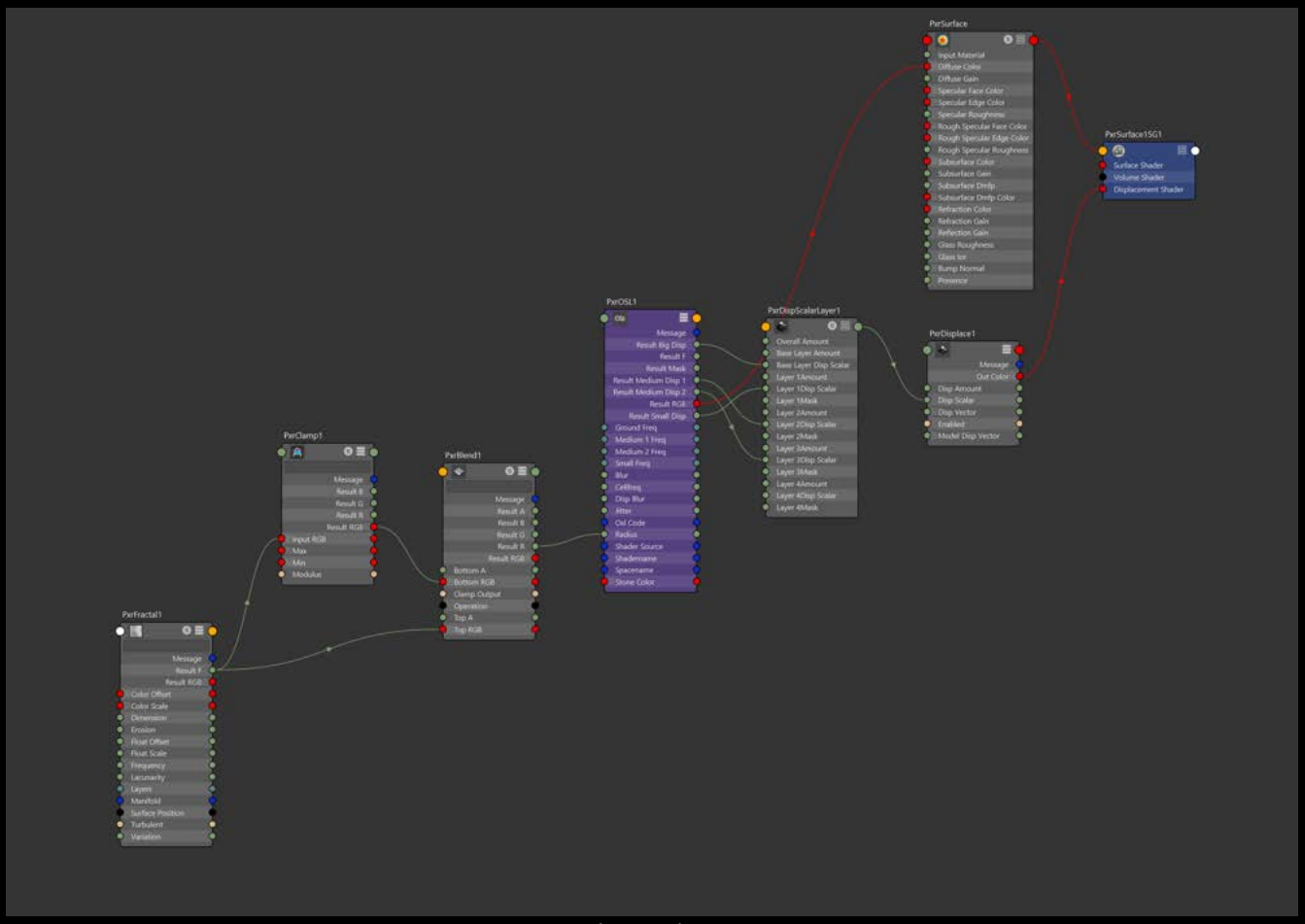
Frequencies – There are three additional layers of displacement that are added to each cell, their frequencies can be individually controlled to create nuance and organic surface texture. Here each layer is isolated with its default frequency and an exagerated amplitude.



Implementation Example

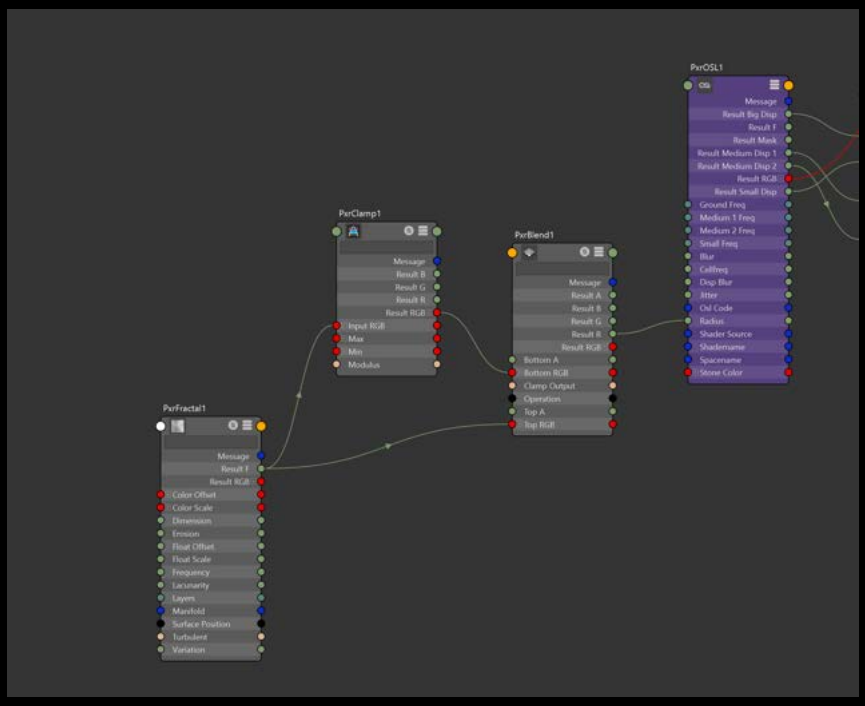
Small: 50 Medium1: 17 Medium2: 3

Implementation



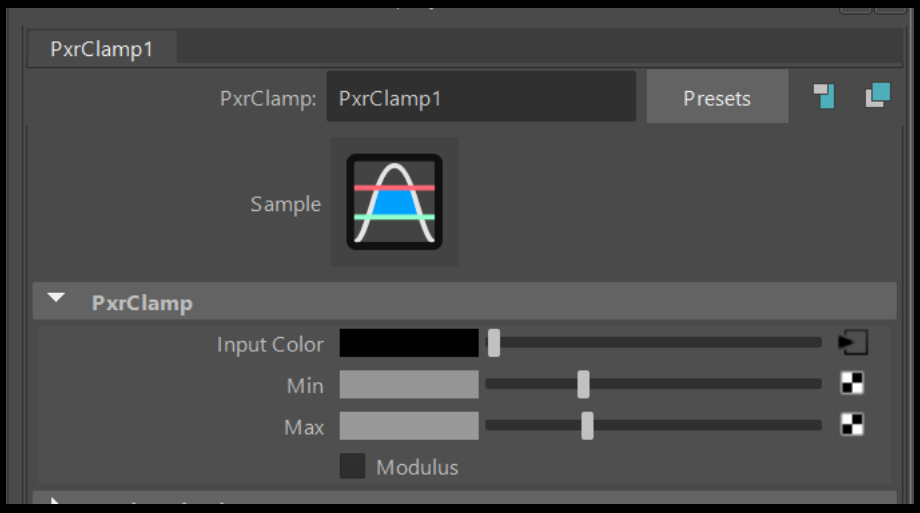
Example Node System

Inputs

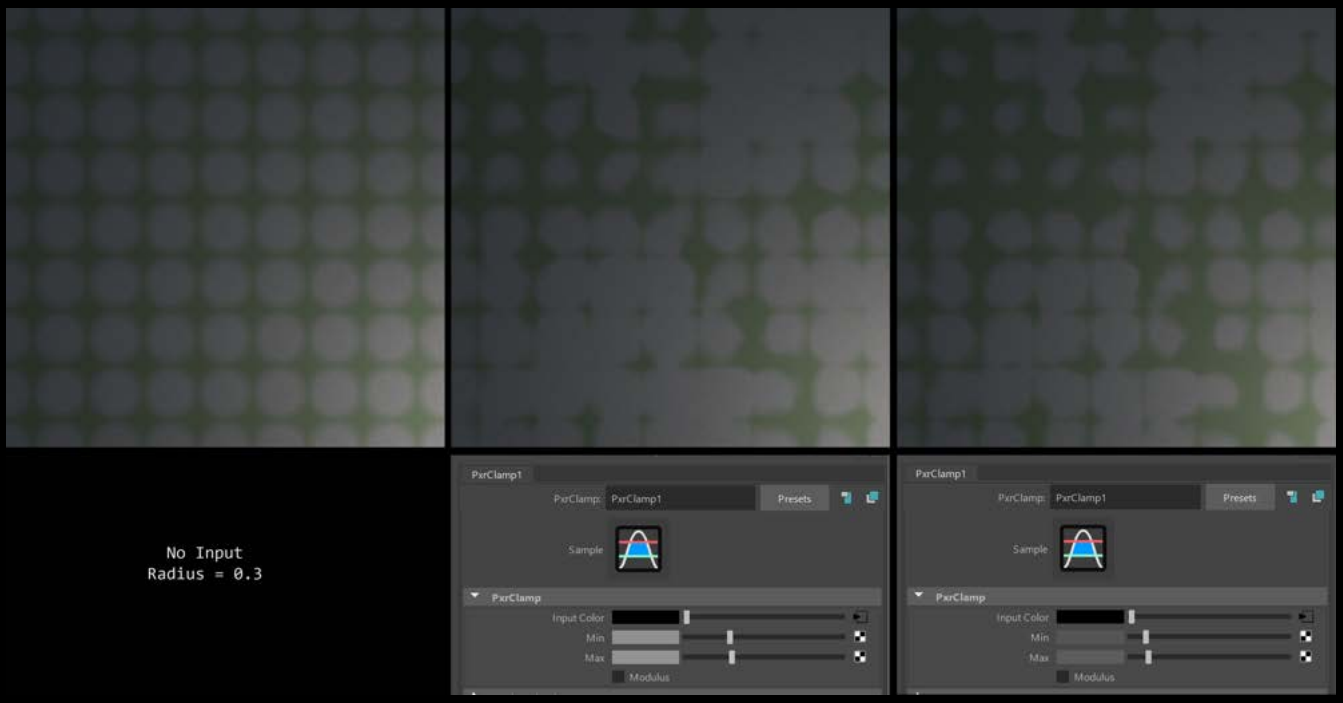


The radius of each cell is driven by a PxrFractal node in order to create a more organic profile for the stone.

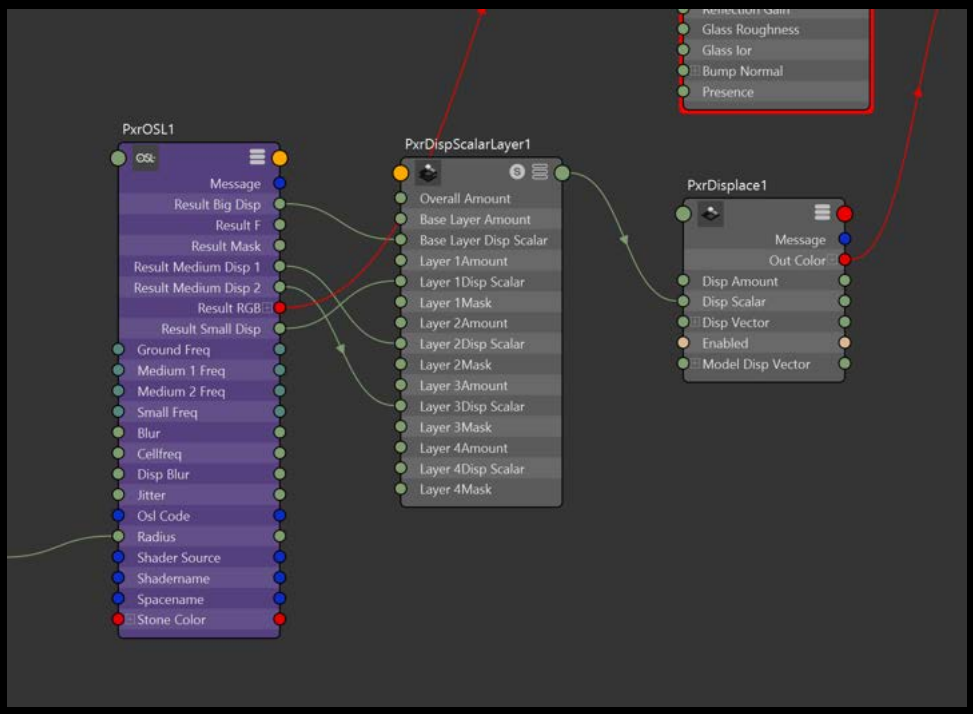
Since each cell has a maximum radius of 0.5, the values from PxrFractal need to be clamped in order to conform and also in order to more easily control the general radius of the resulting shape.



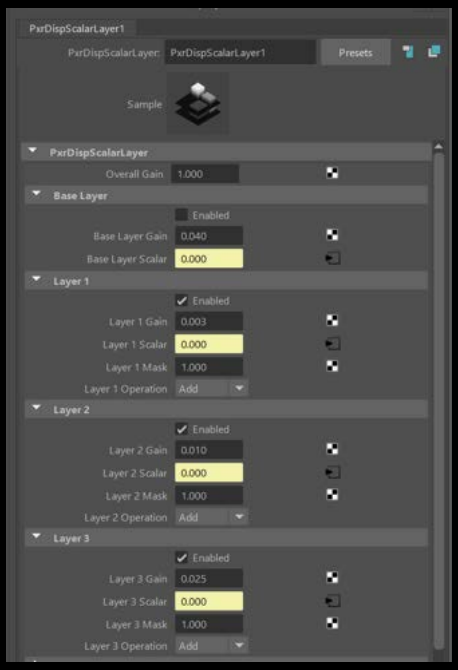
It is necessary to clamp these values to a very small range to promote uniformity, however you can translate the minimum and maximum together in order to change the resulting radius. The range can be widened if you desire more size variation in the stones.



Displacement



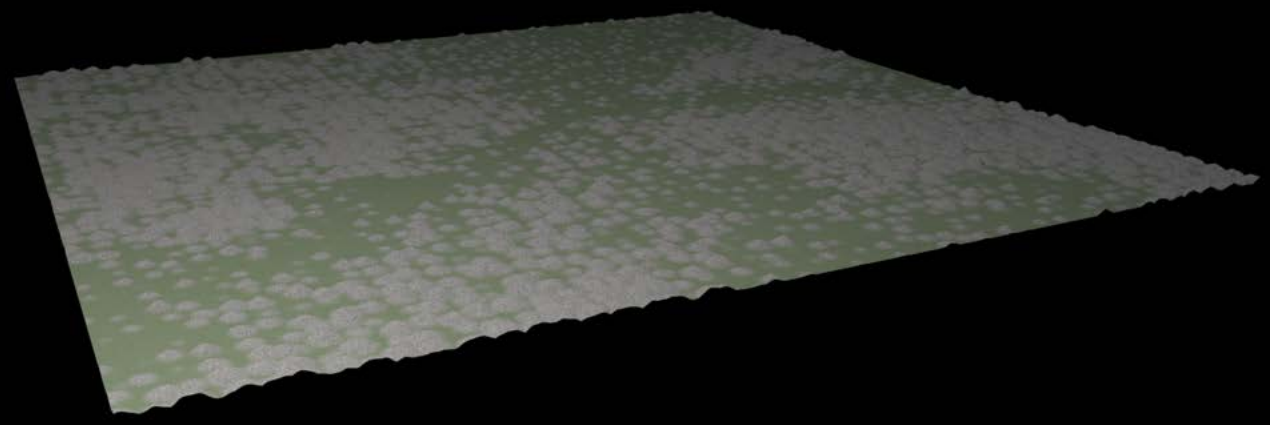
This shader exports four float values that are intended to be layered to create an organic stone displacement.



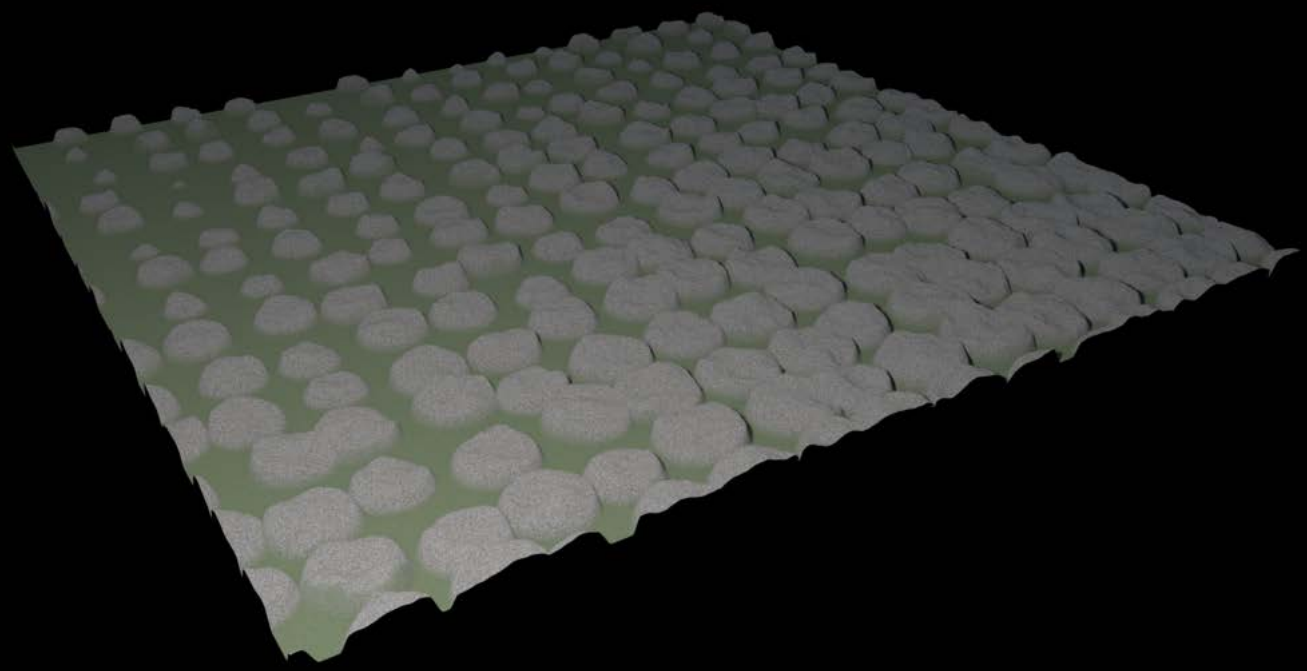
These four values can be easily combined in a PxrDispScalarLayer node where the displacement amplitude of each layer can be controlled by the Gain value.

For easy look development adjust the individual values first with a low cell frequency, then scale to your needed cell frequency and matching that by using the Overall Gain.

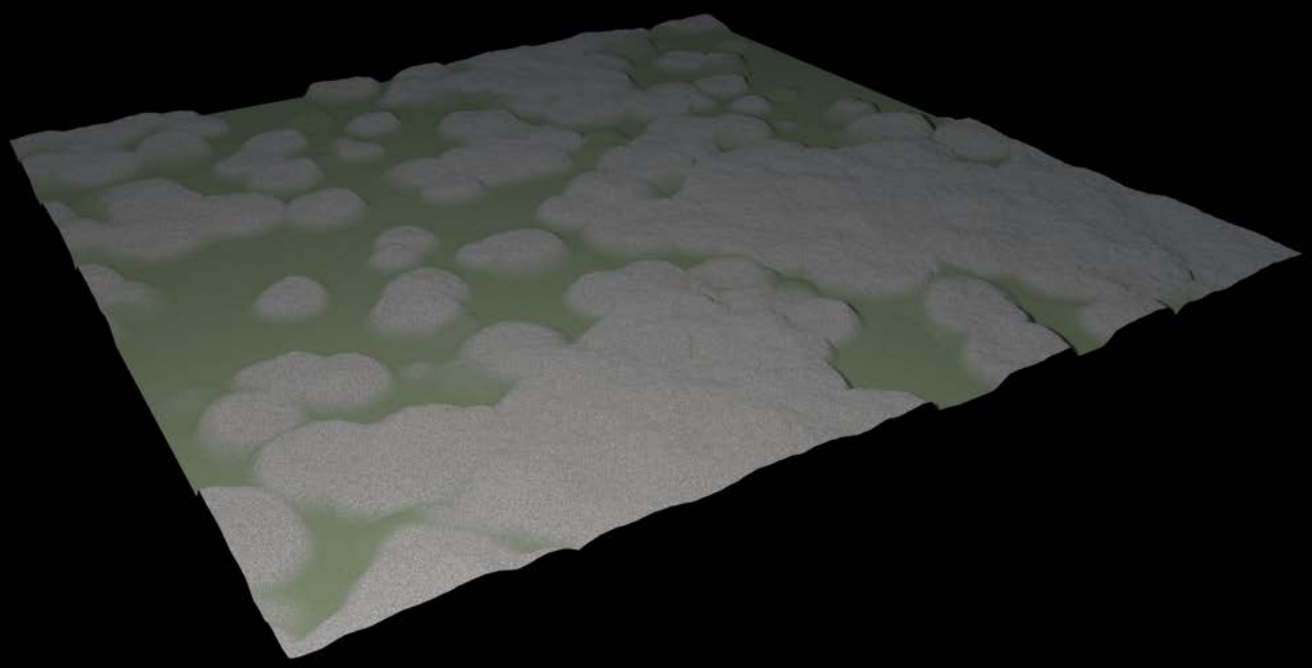
Example Applications



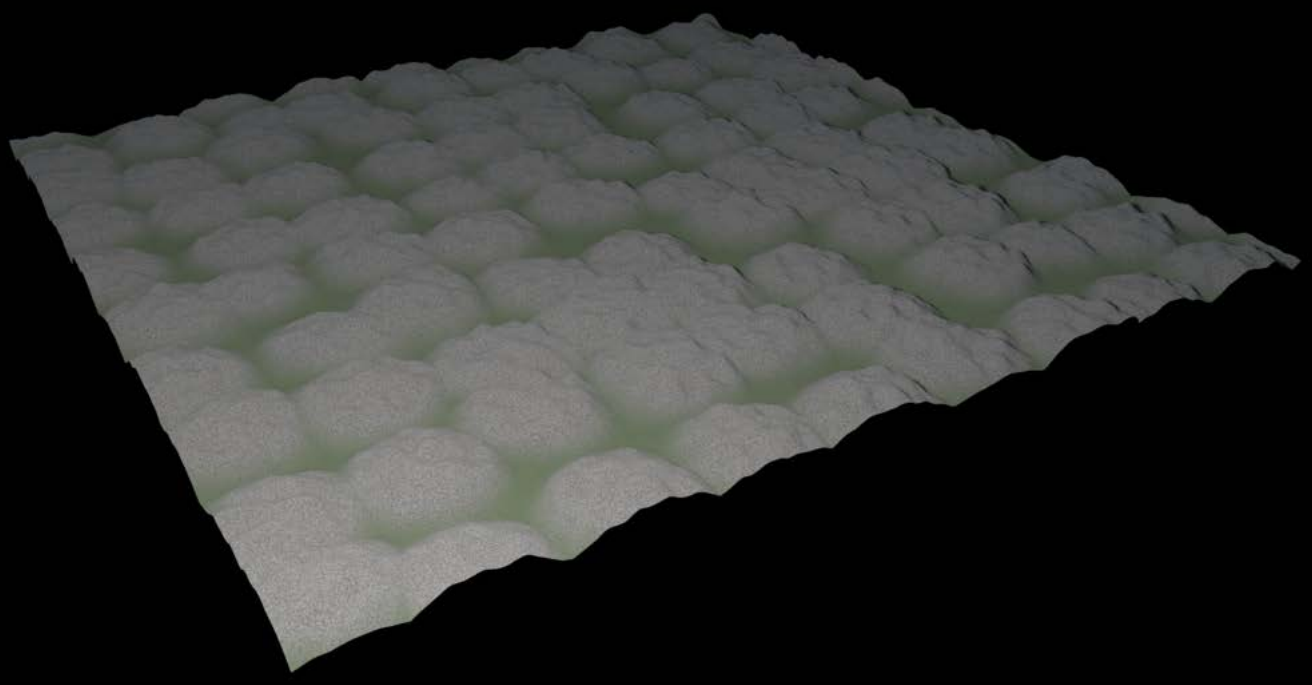
CellFreq	Radius	Blur	DispBlur	Jitter	Medium1	Medium2	Small
45	PxrFractal	0	0.3	0.57	20	25	80



CellFreq	Radius	Blur	DispBlur	Jitter	Medium1	Medium2	Small
14	PxrFractal	0	0.1	0.33	17	20	75



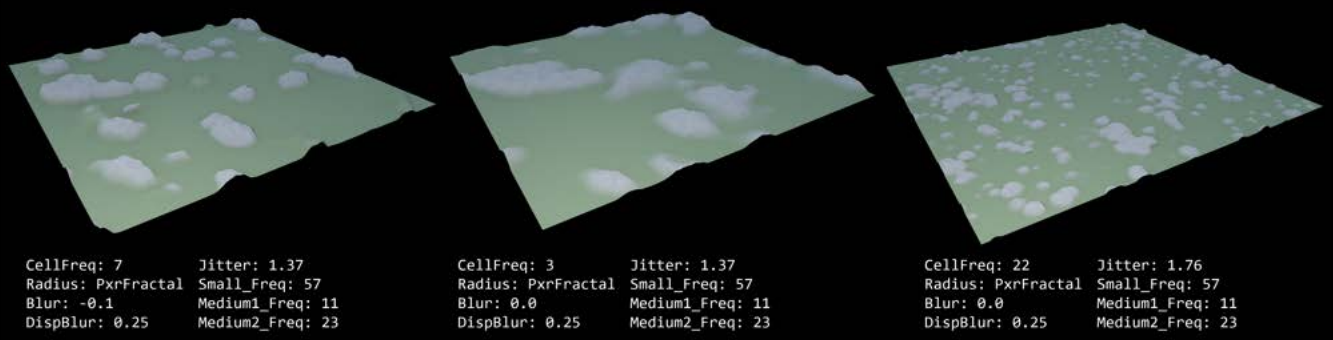
CellFreq	Radius	Blur	DispBlur	Jitter	Medium1	Medium2	Small
12	PxrFractal	0	0.2	1.07	18	9	57



CellFreq	Radius	Blur	DispBlur	Jitter	Medium1	Medium2	Small
9	PxrFractal	-0.1	0.4	0.34	17	29	77

Extended Application

This shader was originally built to simulate cobbled stone walkways and surfaces. During development it was discovered that this shader is very versatile in creating basic displacements and visualization useful for landscape look development.



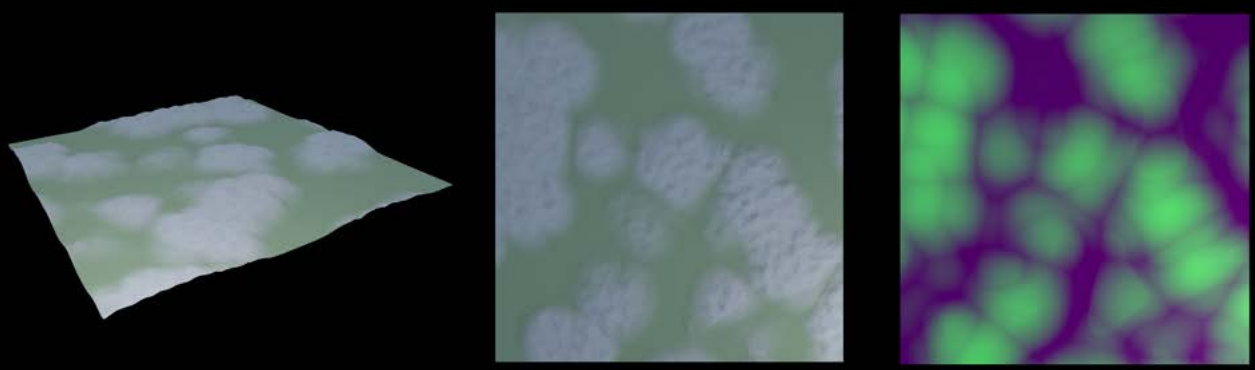
CellFreq: 7 Jitter: 1.37
Radius: PxrFractal Small_Freq: 57
Blur: -0.1 Medium1_Freq: 11
DispBlur: 0.25 Medium2_Freq: 23

CellFreq: 3 Jitter: 1.37
Radius: PxrFractal Small_Freq: 57
Blur: 0.0 Medium1_Freq: 11
DispBlur: 0.25 Medium2_Freq: 23

CellFreq: 22 Jitter: 1.76
Radius: PxrFractal Small_Freq: 57
Blur: 0.0 Medium1_Freq: 11
DispBlur: 0.25 Medium2_Freq: 23

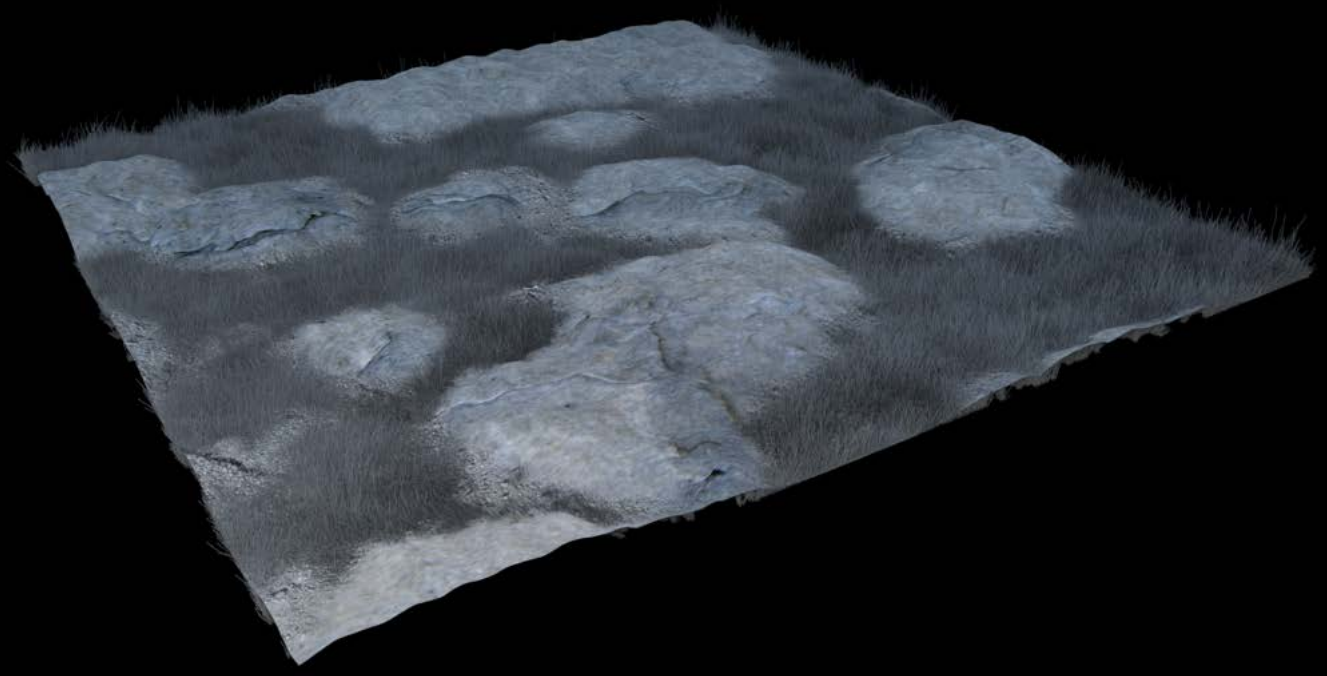
Using the shader in this way provides a jumpstart on creating realistic landscapes because it provides organically placed and formed rock and stone formations, and the displacements needed for them without the hassle of painting maps.

The shader by default exports all of the masks it uses to create the basic coloration and displacements. These masks can be referenced by the user to apply more realistic texture maps to the existing surfaces. The masks can also be referenced from the geometry level and used to control placement of things like grass generation.



This is an example of the map generated by the Base Displacement layer. This layer correlates most closely with the coloration in most cases.

Here is a render of the above mask being used to control procedurally created grass placement within Houdini*. A second mask is also used to procedurally replace the original coloration with more detailed textures.



CellFreq	Radius	Blur	DispBlur	Jitter	Medium1	Medium2	Small
4	PxrFractal	-0.1	0.4	0.87	4	17	45

*The shader networks used in Houdini are identical to those used in Maya.