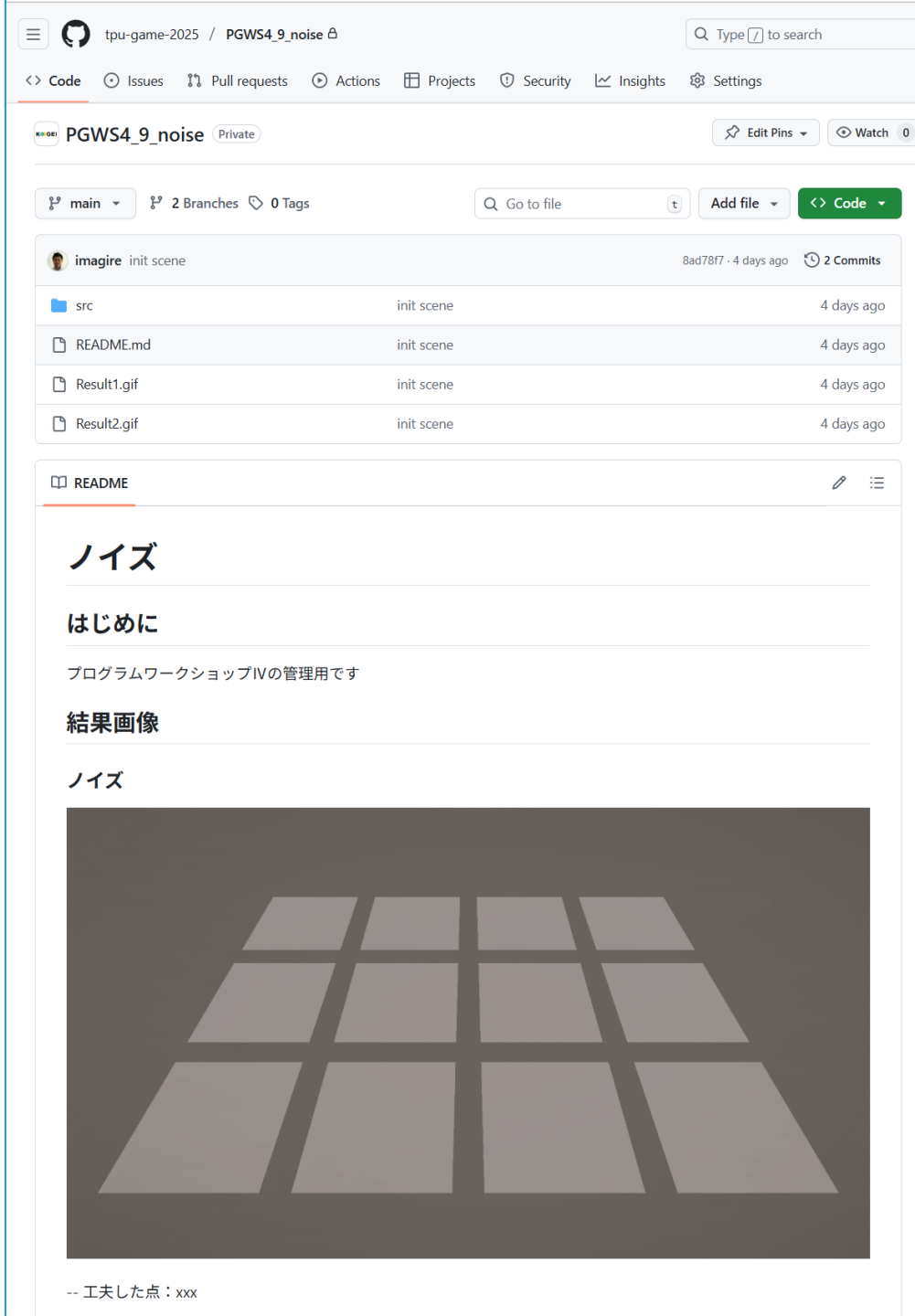
ノイズ

2025年度 プログラムワークショップIV（9）

今回のリポジトリ

- https://github.com/tpu-game-2025/PGWS4_9_noise



本日の内容

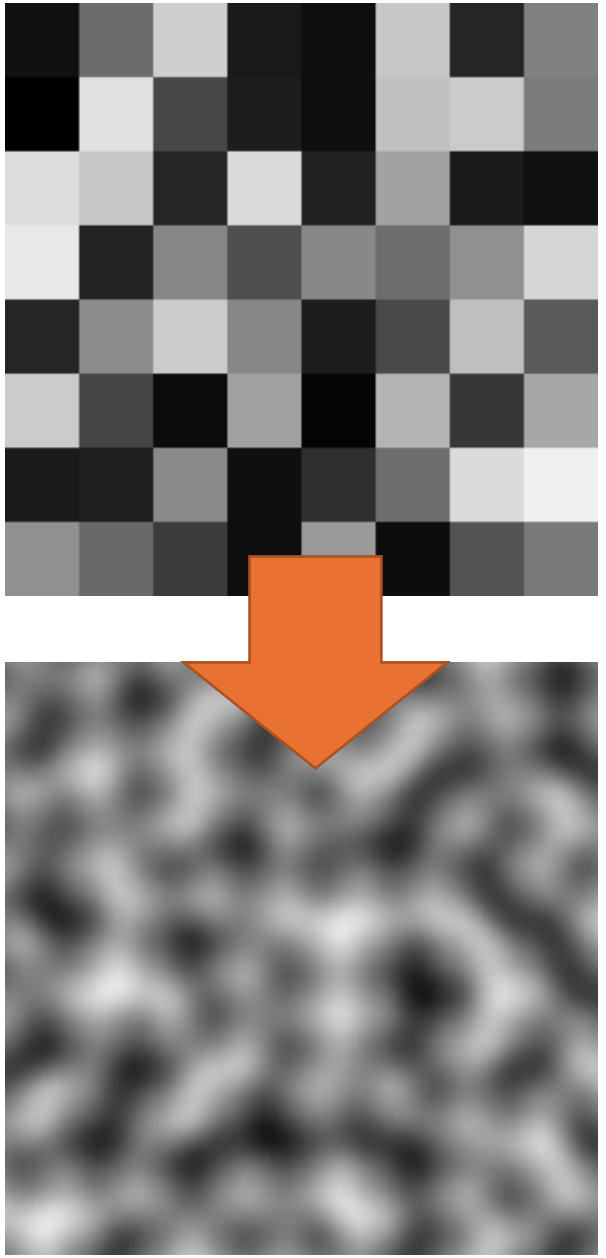
- ノイズの概要
- 基本的なノイズ
- FBM
- 加工したFBM
- ボロノイ図
- 炎
- ディゾルブ

本日の内容

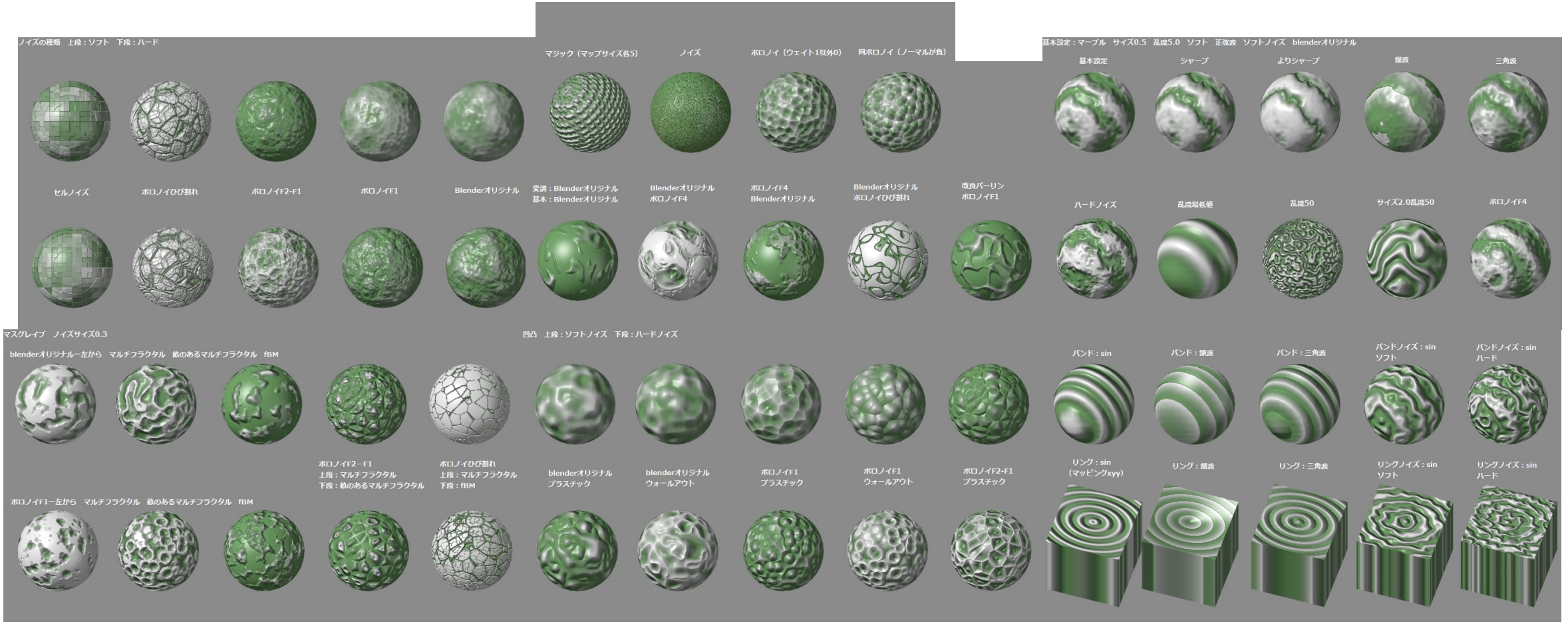
- ノイズの概要
- 基本的なノイズ
- FBM
- 加工したFBM
- ボロノイ図
- 炎
- ディゾルブ

ノイズ

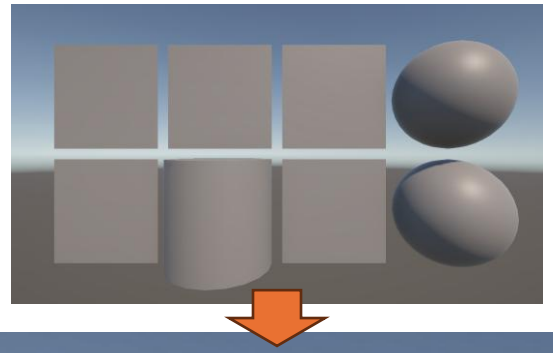
- ランダム性を入れると単調でなくなるので大いに使いたい
- しかし、ただのランダムではざらざらするだけ
- 次第に変化してく乱雑さを使おう



ノイズを使ったテクスチャの例（Blender）

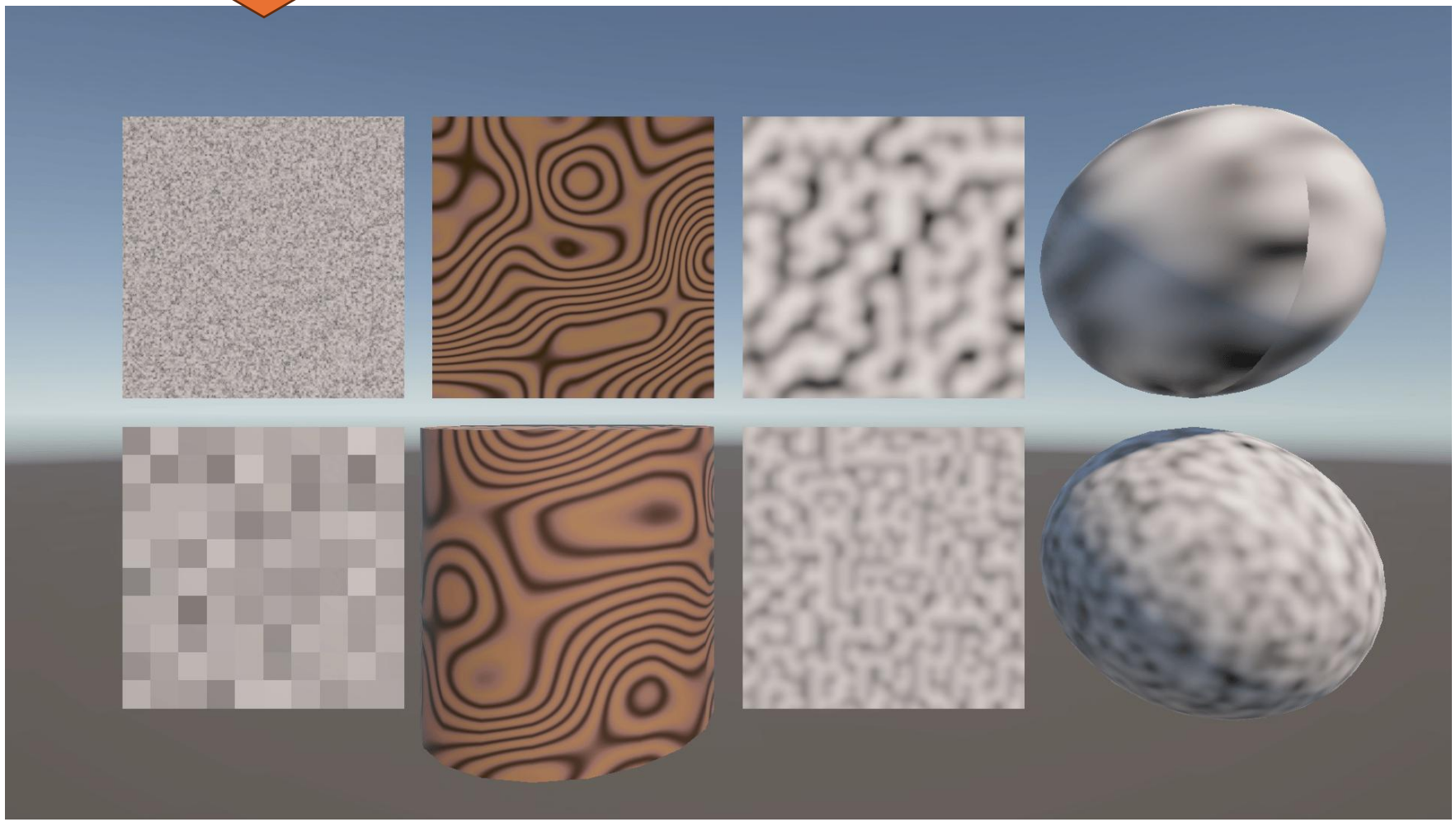


本日の内容



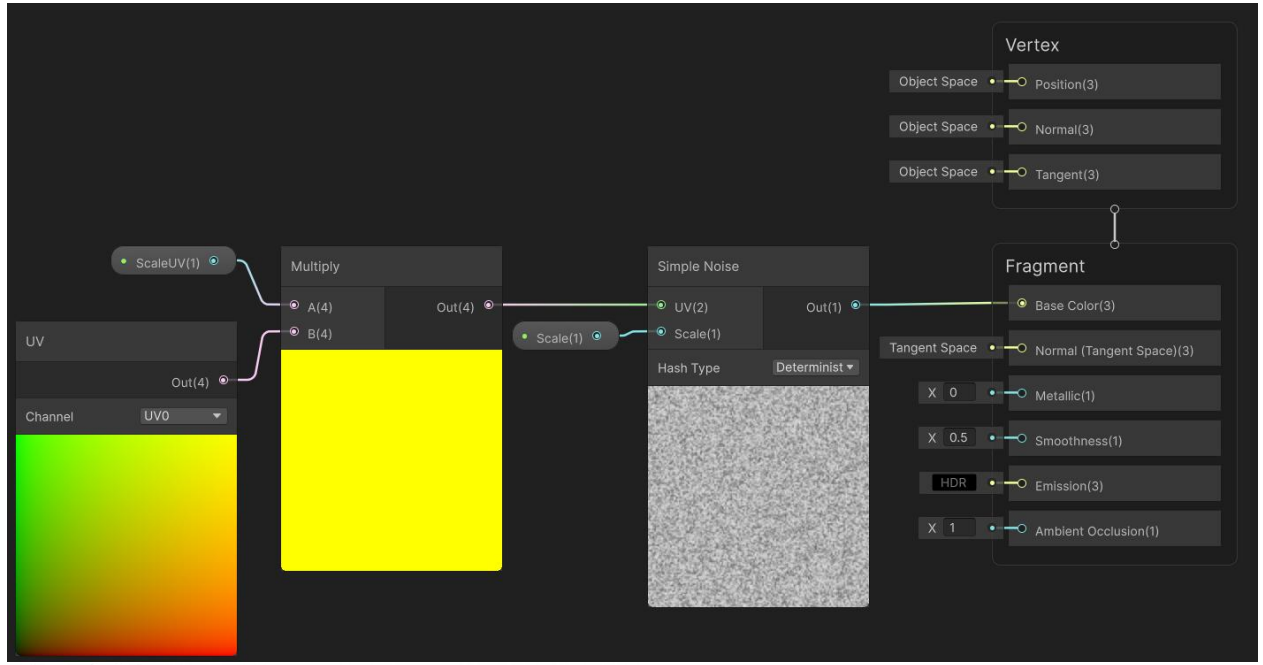
シーン：1 Noise Scene

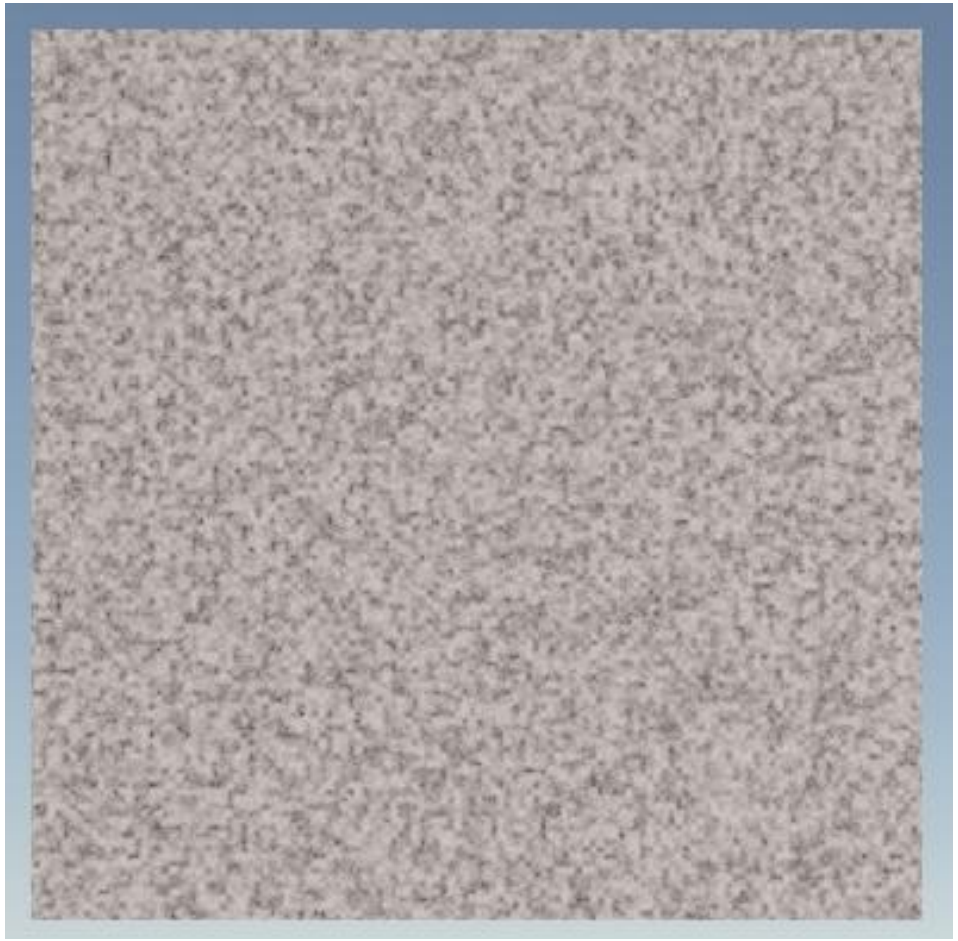
- ノイズの概要
- 基本的なノイズ
  - 一様ノイズ
  - ブロックノイズ
  - 年輪模様
  - 周期的ノイズ
  - Perlinノイズ
  - ボリュームノイズ
- FBM
- 加工したFBM
- ボロノイ図
- 炎
- ディゾルブ

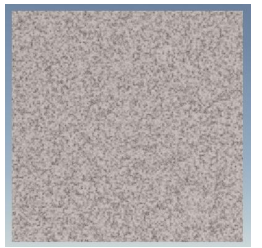


一様ノイズ

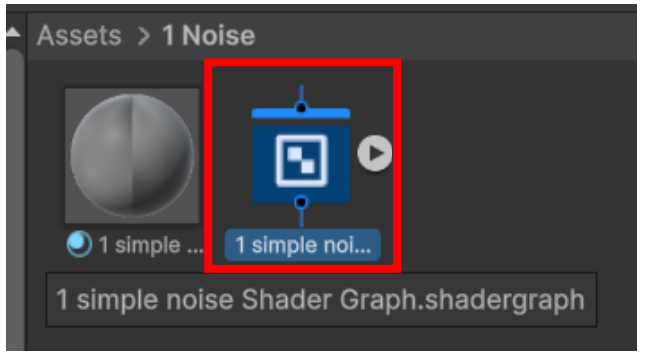
- [0, 1]の範囲で入力毎に乱数を返す
- Unityでは、Simpleノイズ
  - テクスチャ座標なので異なる入力を与える



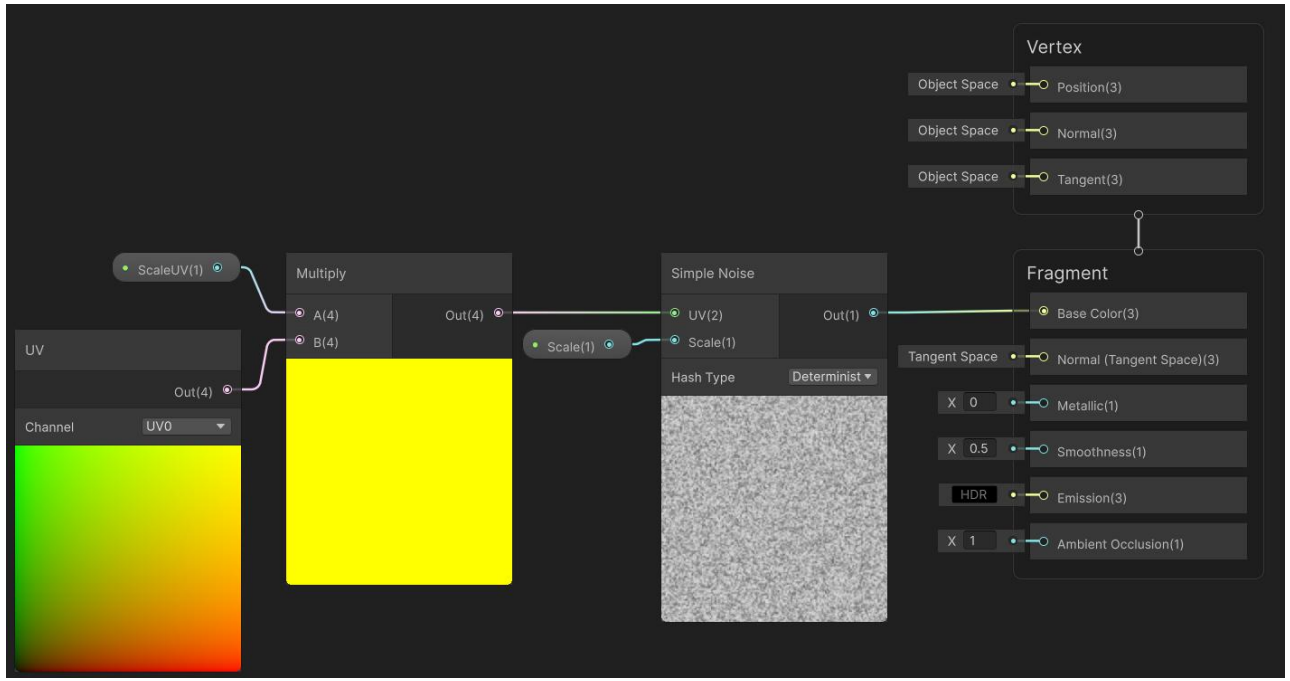




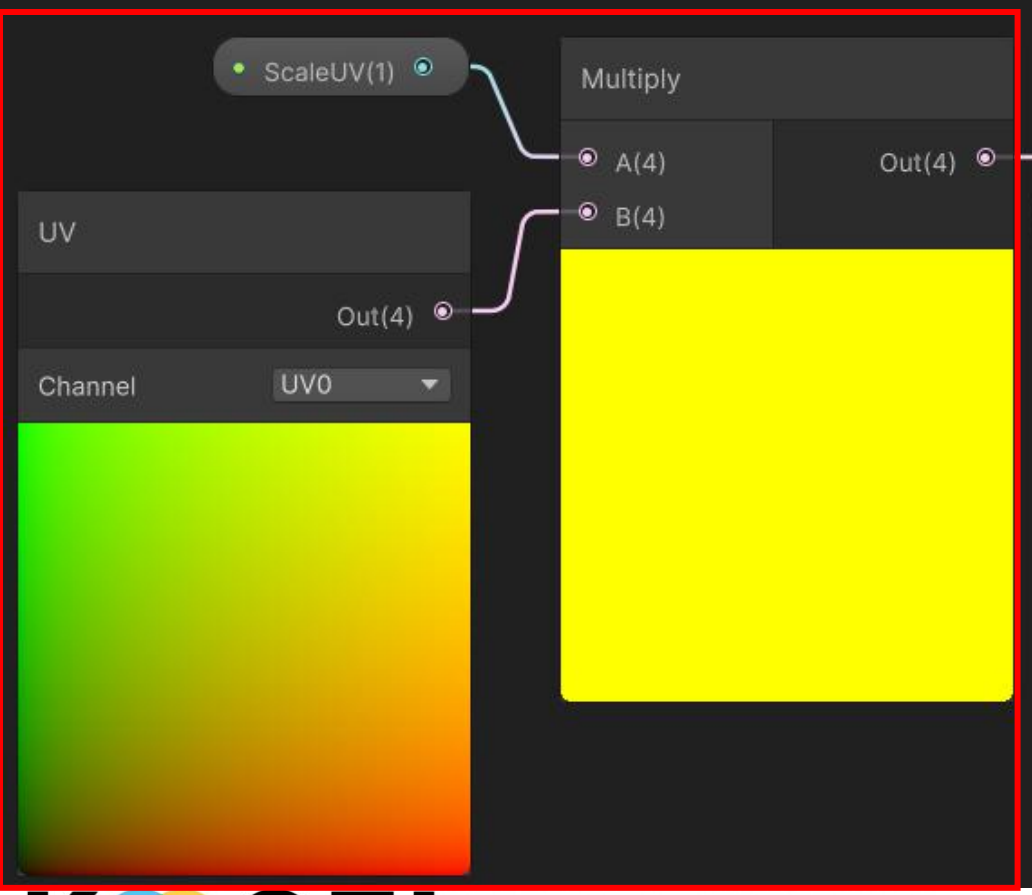
やってみよう



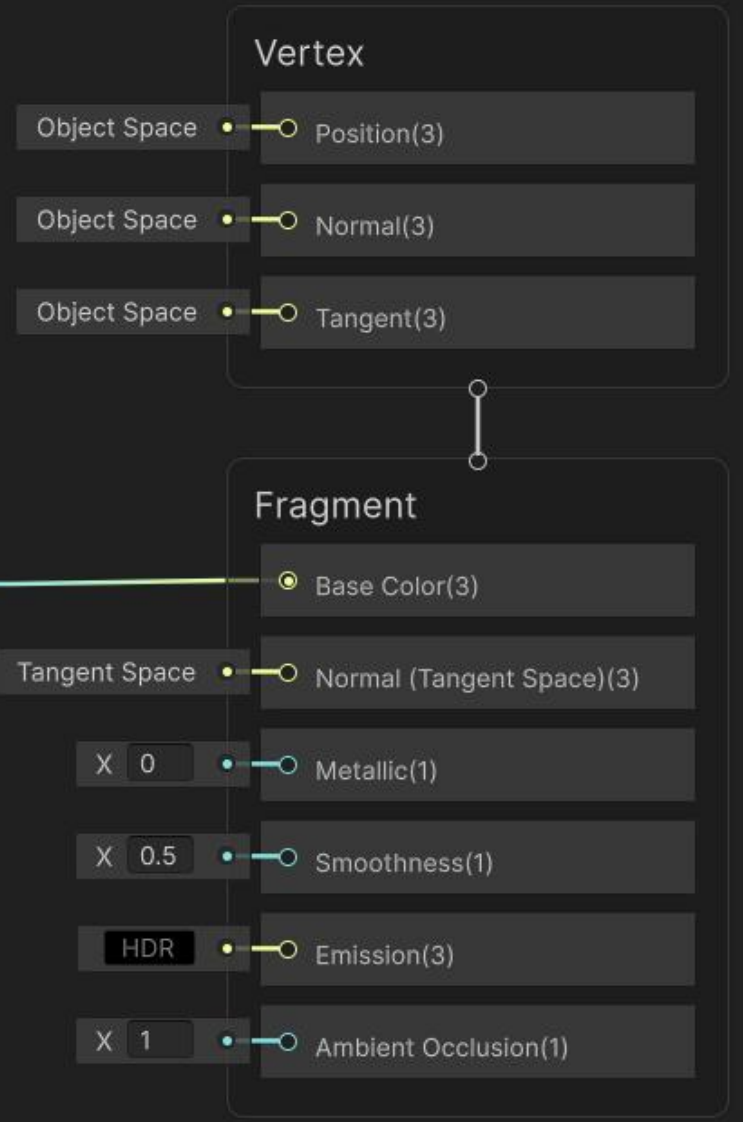
- Shader Graphを作成
  - 「1 Noise/1 simple noise Material」に対して設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を2つ追加(次は設定例)
    - Scale
      - Float型 (Mode: Slider)
      - 範囲: [0, 10]
      - 初期値: 1
    - Scale UV
      - Float型(Mode: Slider)
      - 範囲: [0, 1000]
      - 初期値: 500



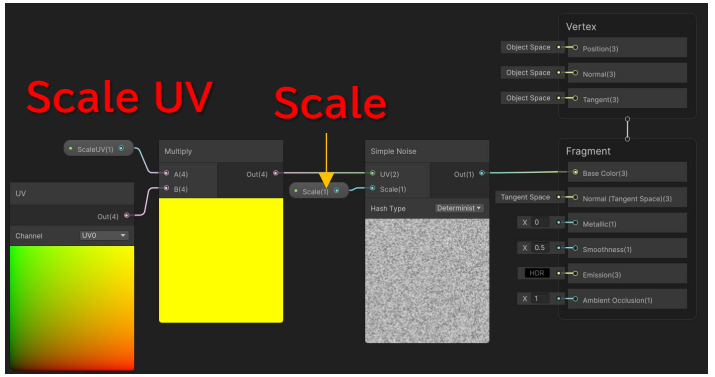
テクスチャ座標の拡大
（確認用）



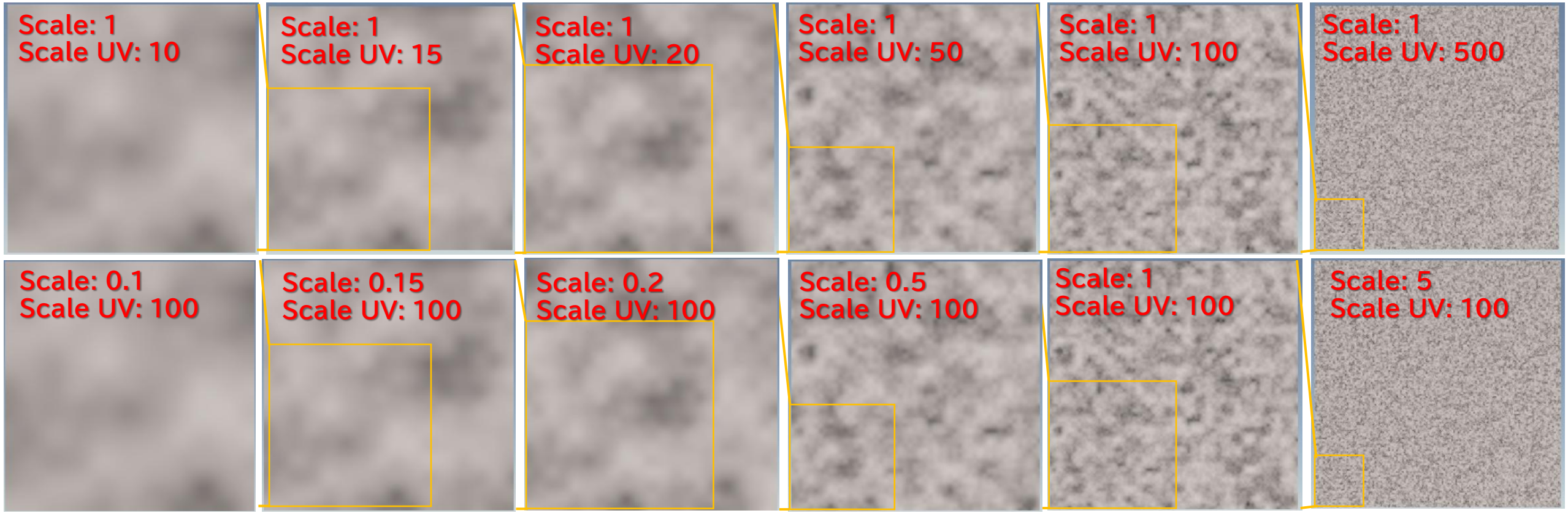
パラメータ
確認用



パラメータの変化



- Simple NoiseノードのScaleは入力の拡大と同じ



注意

- 通常使われる乱数は疑似乱数
  - アルゴリズム的に適正した十分乱雑に感じられる数
    - 入力の範囲が不適切だと、繰り返しや乱雑に見えない値が生成される
    - 常に同じ値が返ってくる
      - 再現性は高い
      - 毎回同じ結果となる
- 実装例

```
float random (float2 uv) {
    return frac(sin(dot(uv.xy, float2(12.9898f, 78.233f))) * 43758.5453123f);
}
```

本日の内容

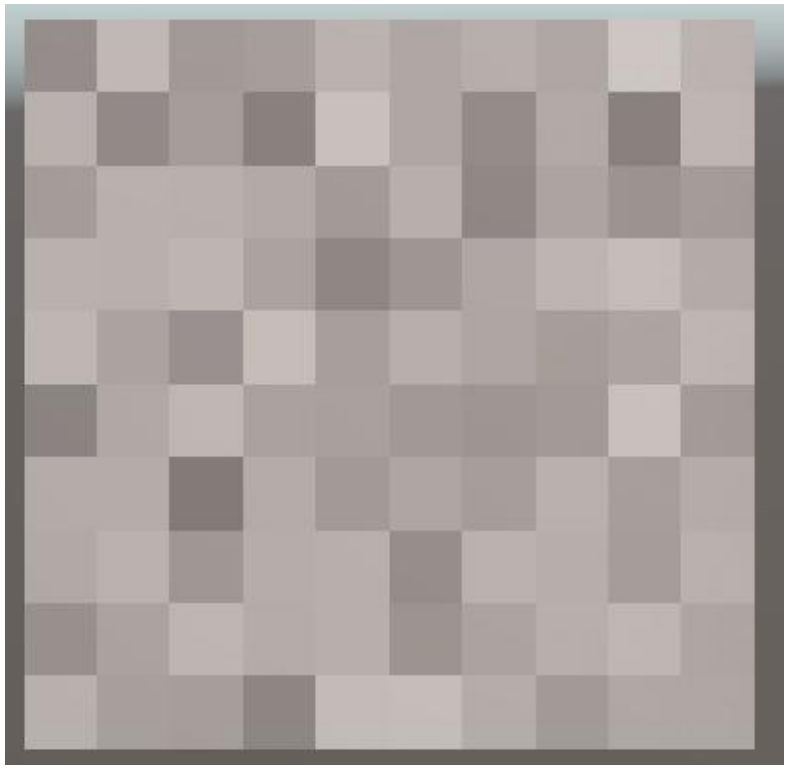
- ノイズの概要
- 基本的なノイズ
  - 一様ノイズ
  - ブロックノイズ
  - 年輪模様
  - 周期的ノイズ
  - Perlinノイズ
  - ボリュームノイズ
- FBM
- 加工したFBM
- ボロノイ図
- 炎
- ディゾルブ

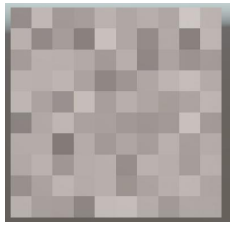
ブロックノイズ

- ここでは、大きさが目に見える四角のノイズ
- ある範囲について同じ値でノイズをサンプリング
  - 小数部を0に切り落としてノイズをサンプリング

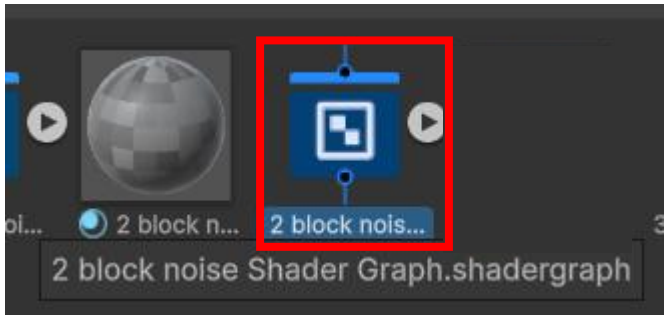
$$block\ noise = noise([uv \times scale])$$
$$= noise(floor(uv \times scale))$$

  - テクスチャ空間で[0,0]-[1,1]の範囲を縦横N個のブロックに区切るのであれば $scale = N$

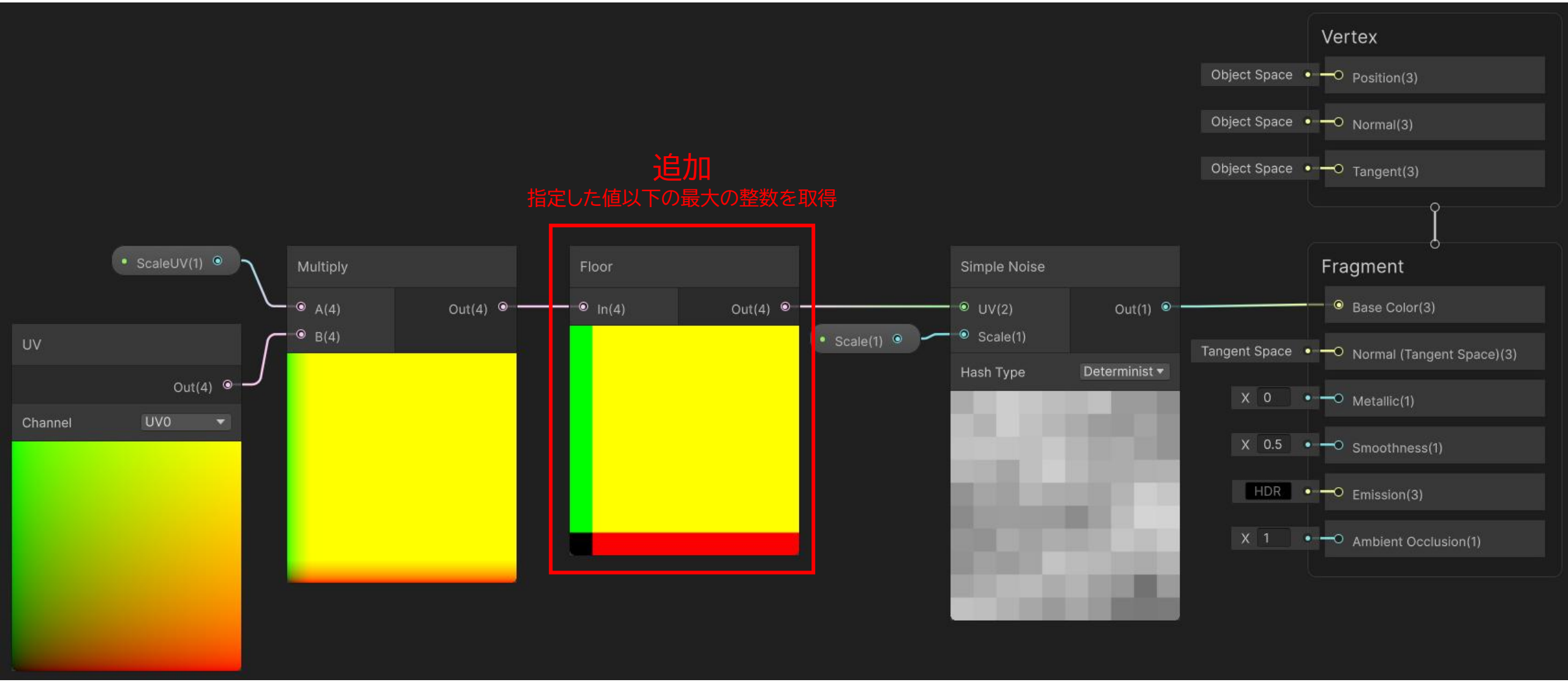




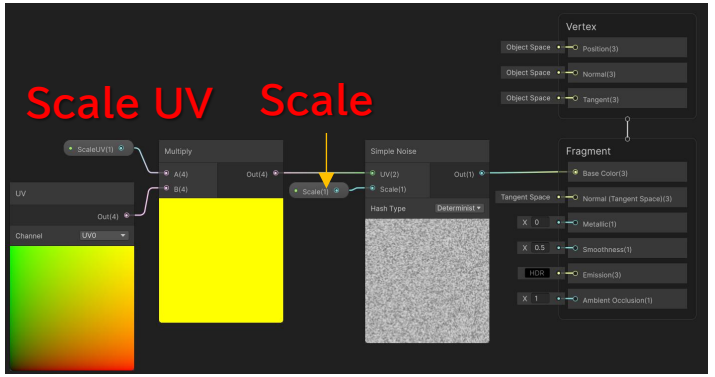
やってみよう



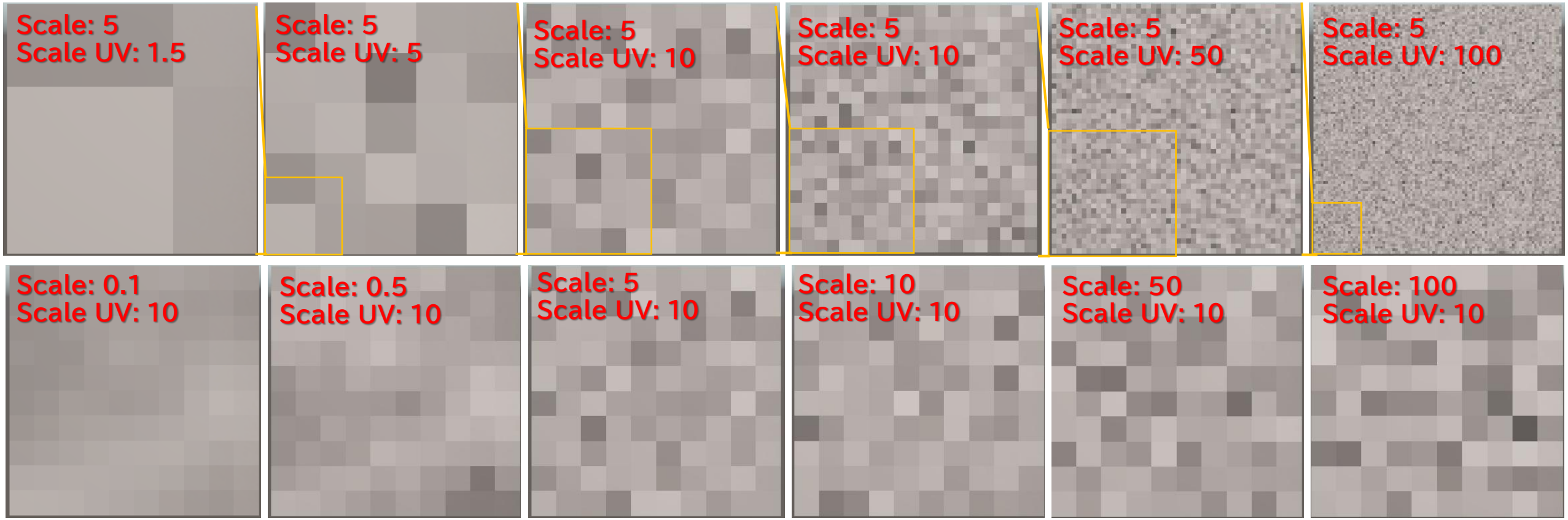
- Shader Graphを作成
  - 「1 Noise/2 block noise Material」に対して設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を2つ追加(次は設定例)
    - Scale
      - Float型 (Mode: Slider)
      - 範囲: [0, 10]
      - 初期値: 5
    - Scale UV
      - Float型 (Mode: Slider)
      - 範囲: [0, 100]
      - 初期値: 10


追加
指定した値以下の最大の整数を取得
Vertex
Object Space
Position(3)
Object Space
Normal(3)
Object Space
Tangent(3)
ScaleUV(1)
Multiply
A(4)
B(4)
Out(4)
Floor
In(4)
Out(4)
Scale(1)
Simple Noise
UV(2)
Scale(1)
Out(1)
Hash Type
Deterministic
Fragment
Base Color(3)
Tangent Space
Normal (Tangent Space)(3)
X 0
Metallic(1)
X 0.5
Smoothness(1)
HDR
Emission(3)
X 1
Ambient Occlusion(1)
UV
Out(4)
Channel
UV0

パラメータに伴う変化



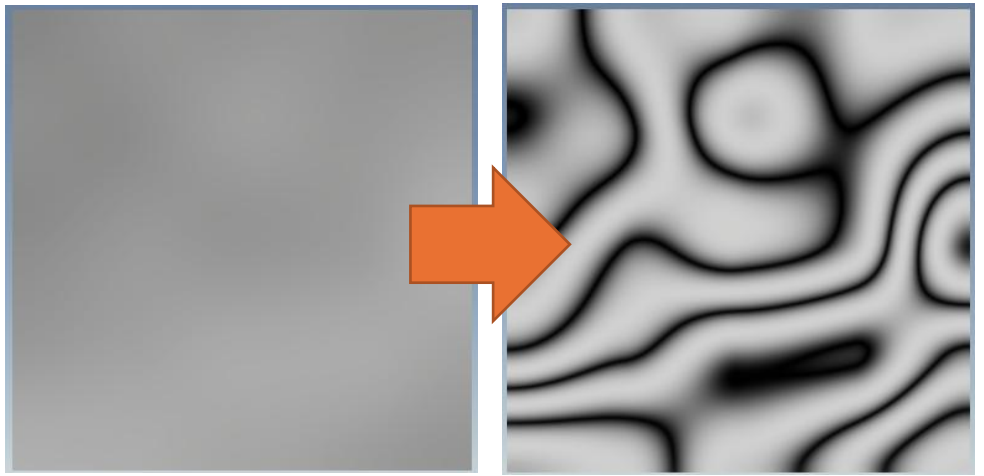
- Scale: ブロックパターン、Scale UV: ブロック間隔



本日の内容

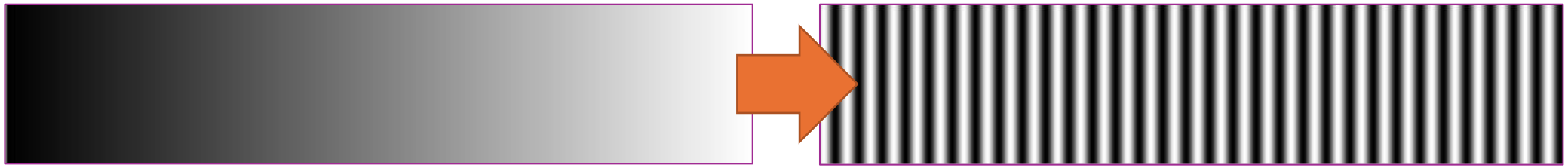
- ノイズの概要
- 基本的なノイズ
  - 一様ノイズ
  - ブロックノイズ
  - 年輪模様
  - 周期的ノイズ
  - Perlinノイズ
  - ボリュームノイズ
- FBM
- 加工したFBM
- ボロノイ図
- 炎
- ディゾルブ

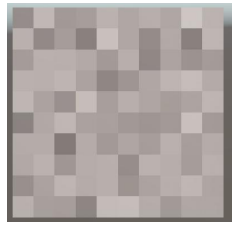
ノイズ変化



- ノイズの出力を変化させる
- 単にノイズを表示するのとは違う
  雰囲気を出すことができる
- どのような例がある？
  - 感性的な物なので、いろいろと試そう

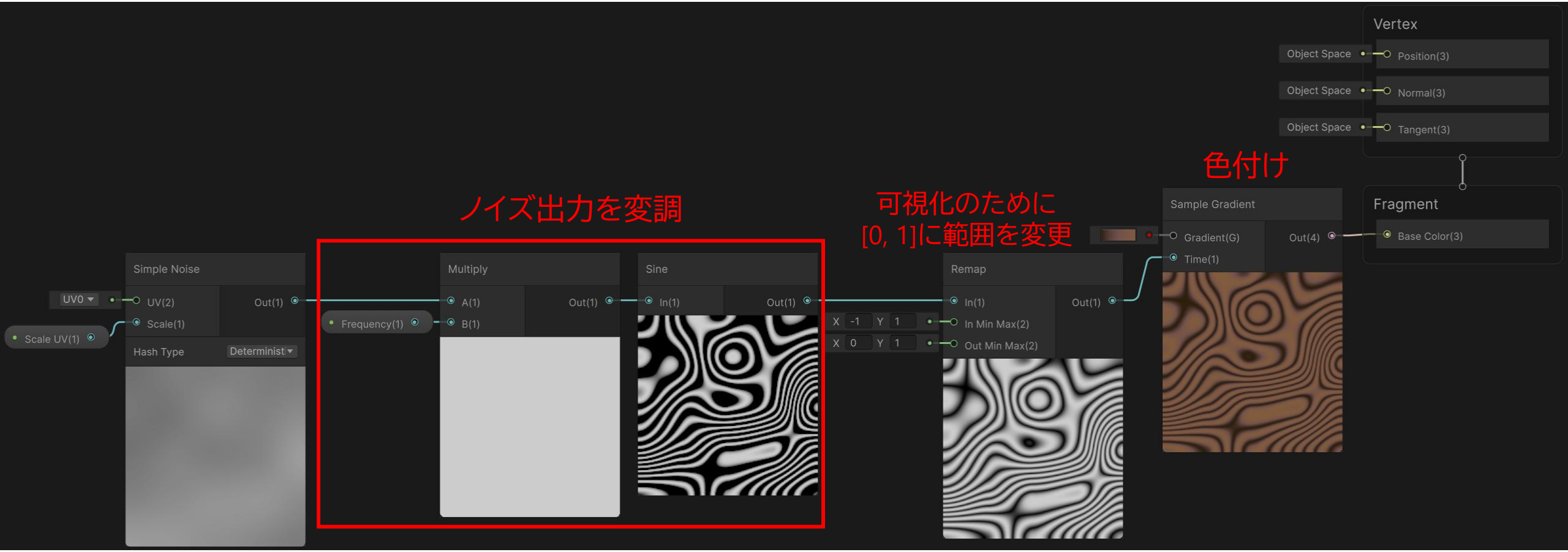
$$t \rightarrow \sin(100t)$$



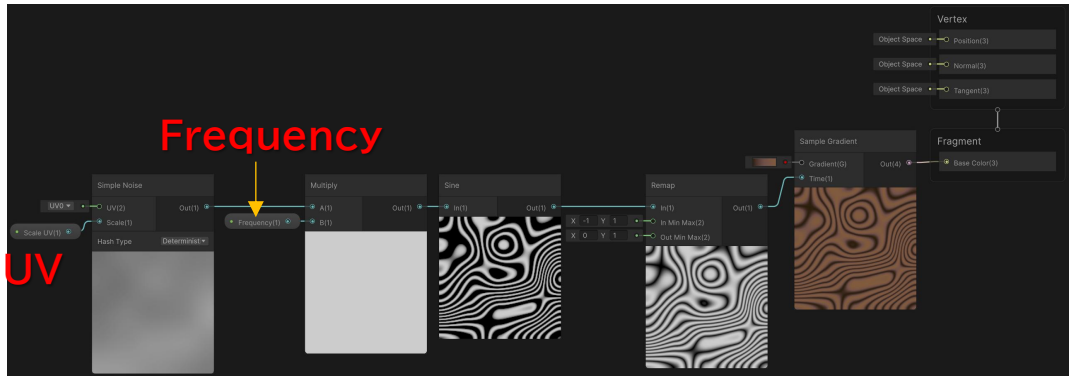


やってみよう

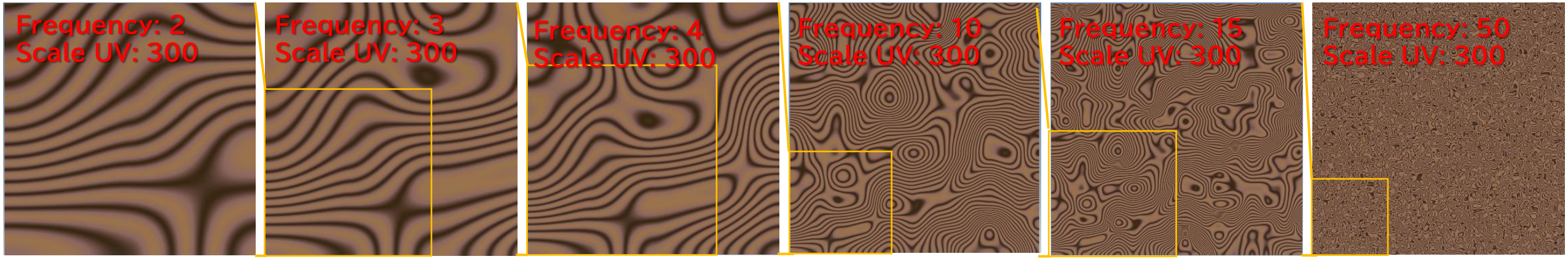
- Shader Graphを作成
  - 「1 Noise/3 ring Material」に対して設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を2つ追加(次は設定例)
    - Frequency
      - Float型 (Mode: Slider)
      - 範囲: [0, 1000]
      - 初期値: 300
    - Scale UV
      - Float型 (Mode: Slider)
      - 範囲: [0, 100]
      - 初期値: 4

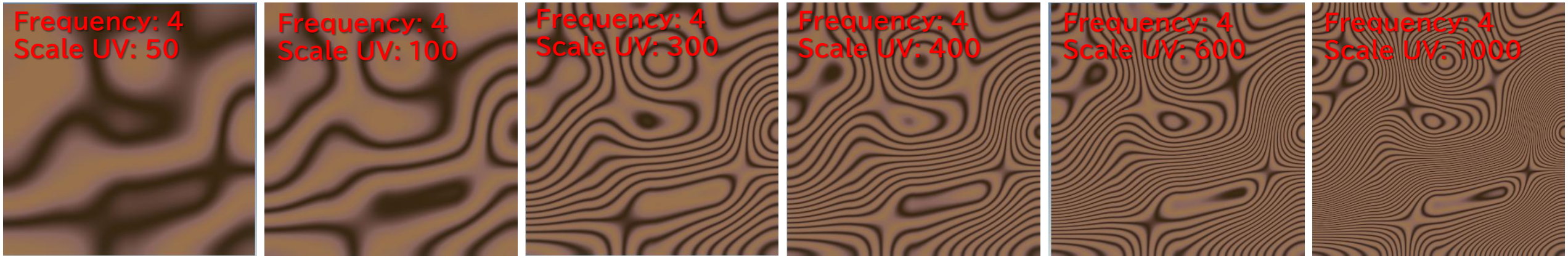

Vertex
Object Space
Position(3)
Object Space
Normal(3)
Object Space
Tangent(3)
色付け
ノイズ出力を変調
可視化のために
[0, 1]に範囲を変更
Sample Gradient
Gradient(G)
Out(4)
Time(1)
Fragment
Base Color(3)
Simple Noise
UV0
UV(2)
Out(1)
Scale UV(1)
Scale(1)
Hash Type
Deterministi
Frequency(1)
Multiply
A(1)
B(1)
Out(1)
Sine
In(1)
Out(1)
Remap
In(1)
Out(1)
X -1 Y 1
In Min Max(2)
X 0 Y 1
Out Min Max(2)

パラメータに伴う変化



- Frequency: 年輪の密度、Scale UV: 模様の拡大縮小



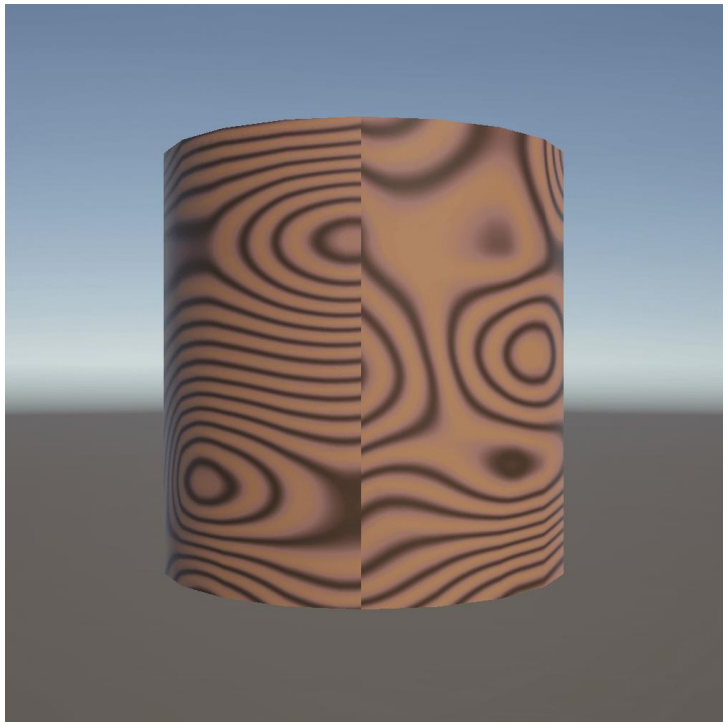


本日の内容

- ノイズの概要
- 基本的なノイズ
  - 一様ノイズ
  - ブロックノイズ
  - 年輪模様
  - 周期的ノイズ
  - Perlinノイズ
  - ボリュームノイズ
- FBM
- 加工したFBM
- ボロノイ図
- 炎
- ディゾルブ

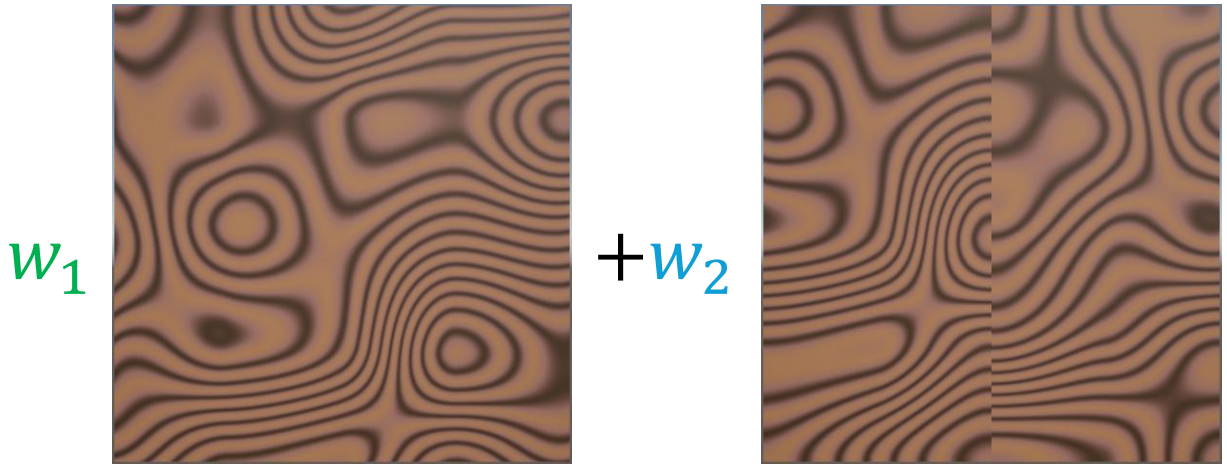
ノイズの欠点

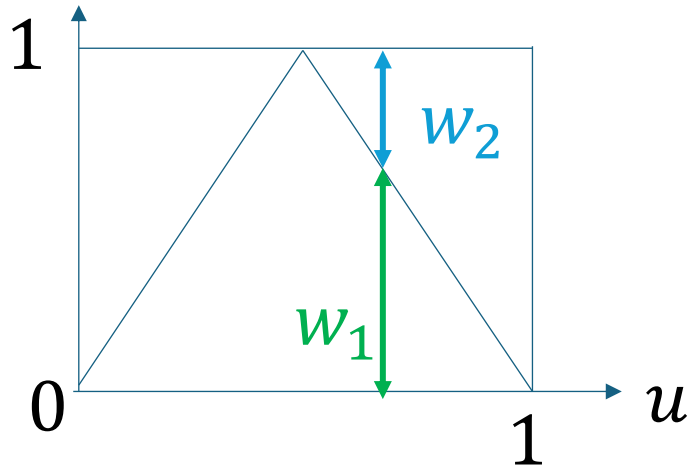
- 円柱に張った際にきれいにつながらない
  - テクスチャであれば、左右がつながっているテクスチャを用意すればよいが、繰り返すノイズを用意するのは難しい
    - 自分で実装すれば可能



ノイズのくり返し

- つながらない場所を使用せず、他のノイズを割り当てる
  - つながらない場所を半分だけずらしたノイズを用意し、つながらない場所の重みを0として合成する





$$w_1 + w_2 = 1$$

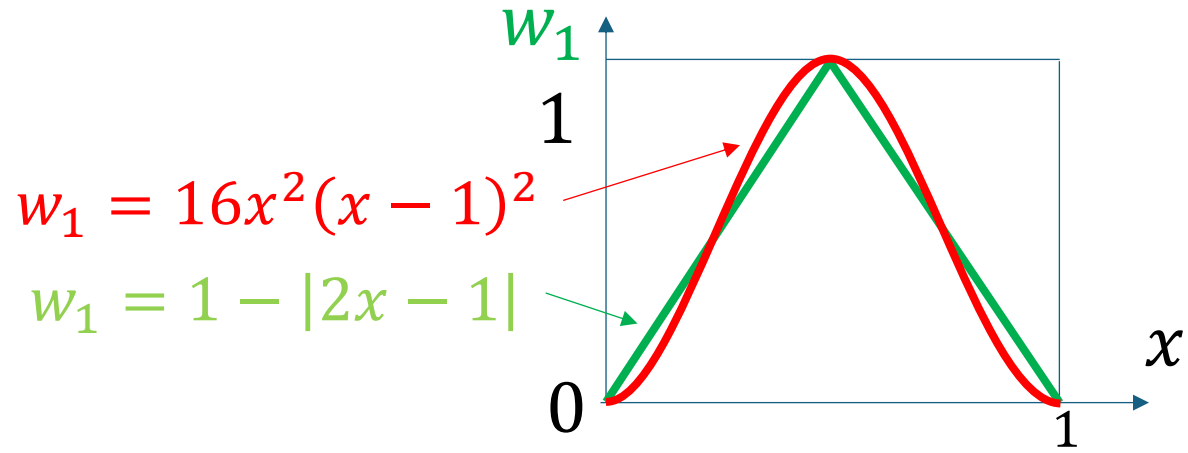
重み関数

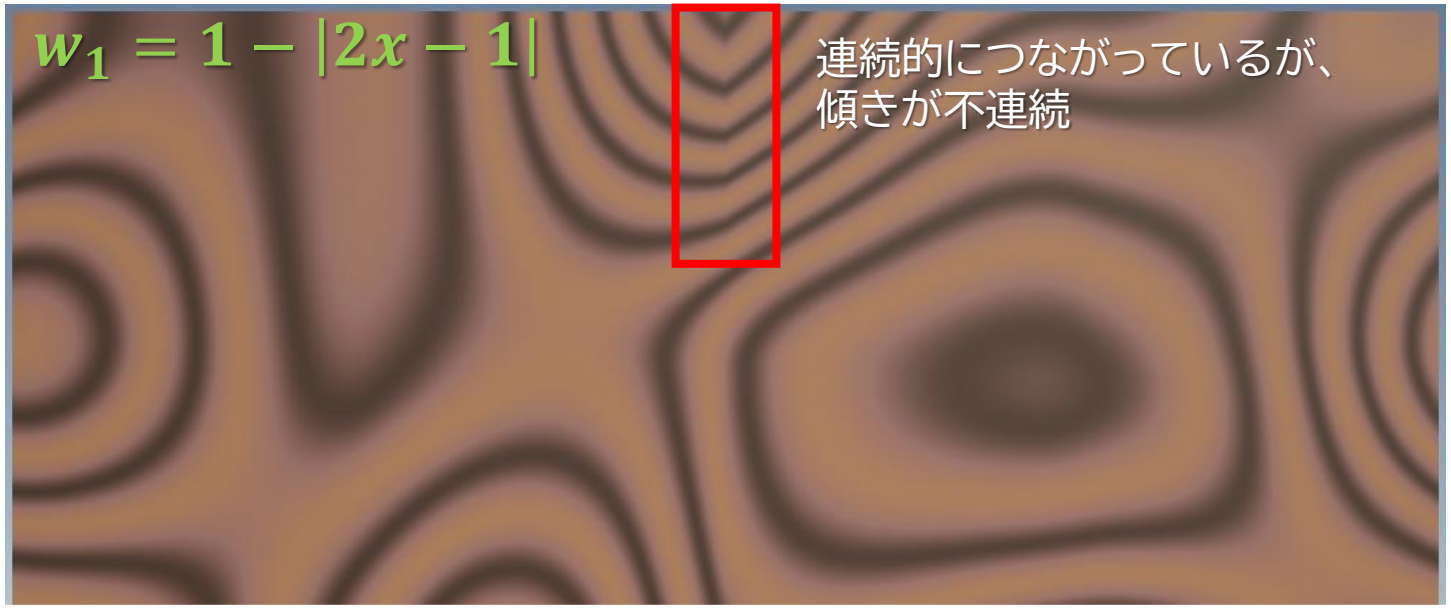
- ただし、重み関数として一次関数を使うと、結果の勾配が不連続になる
- 0,1で0となり、0.5で1となり、それぞれの点で勾配が0となる関数を用いる

$$w_1 = 16x^2(x-1)^2$$

$$w'_1 = 64x\left(x-\frac{1}{2}\right)(x-1)$$

（$x = 0, 0.5, 1$で不連続）

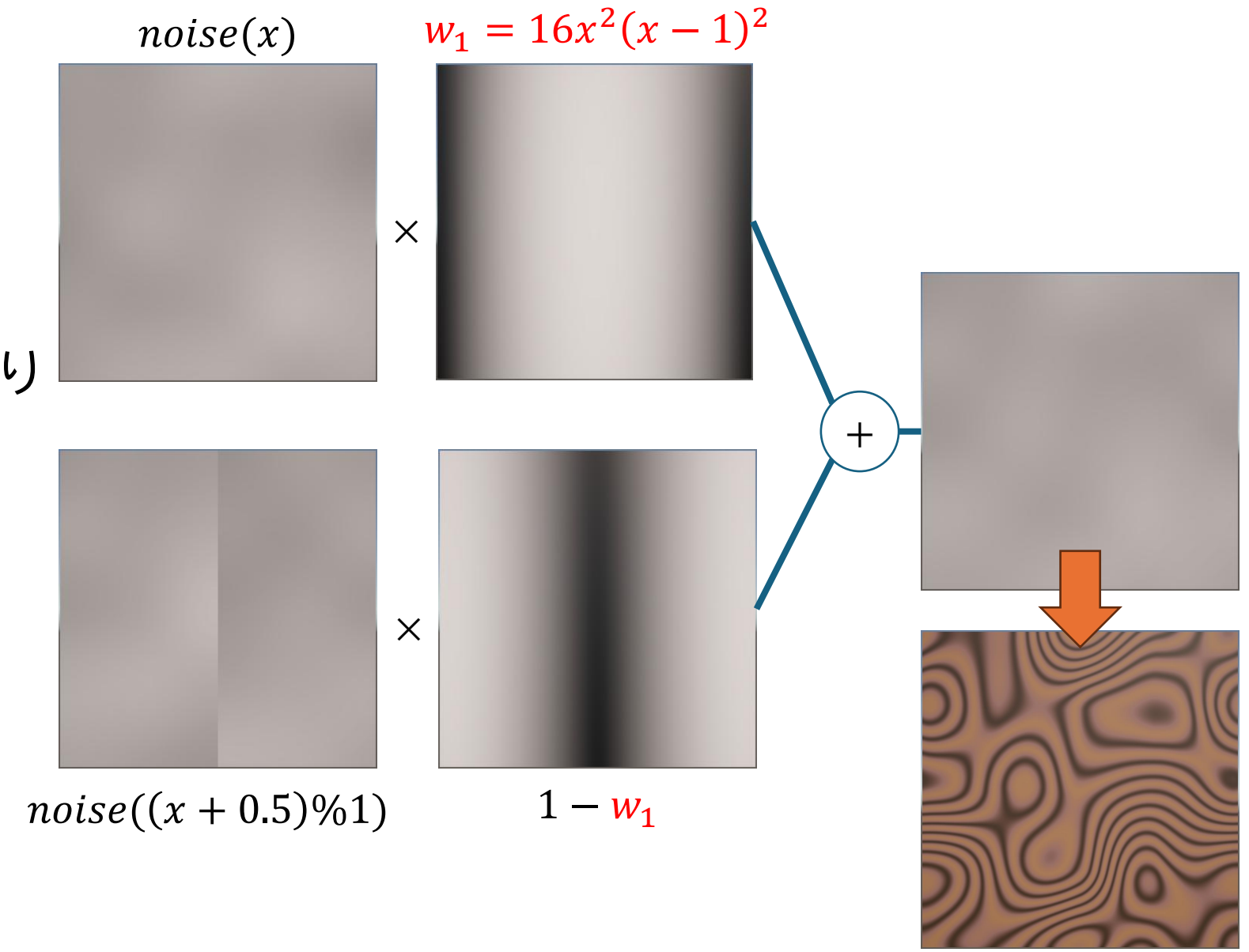




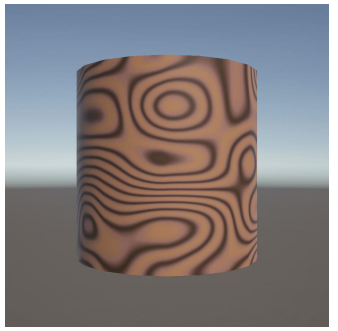
KOGEI TOKYO POLYTECHNIC UNIVERSITY

ノイズの合成

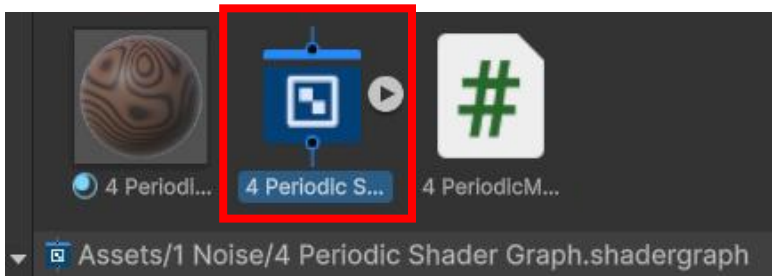
- 一様ノイズの段階で合成を行う
  - 年輪のように変調したり色付けは後で行う
    - 処理をまとめて軽量にする

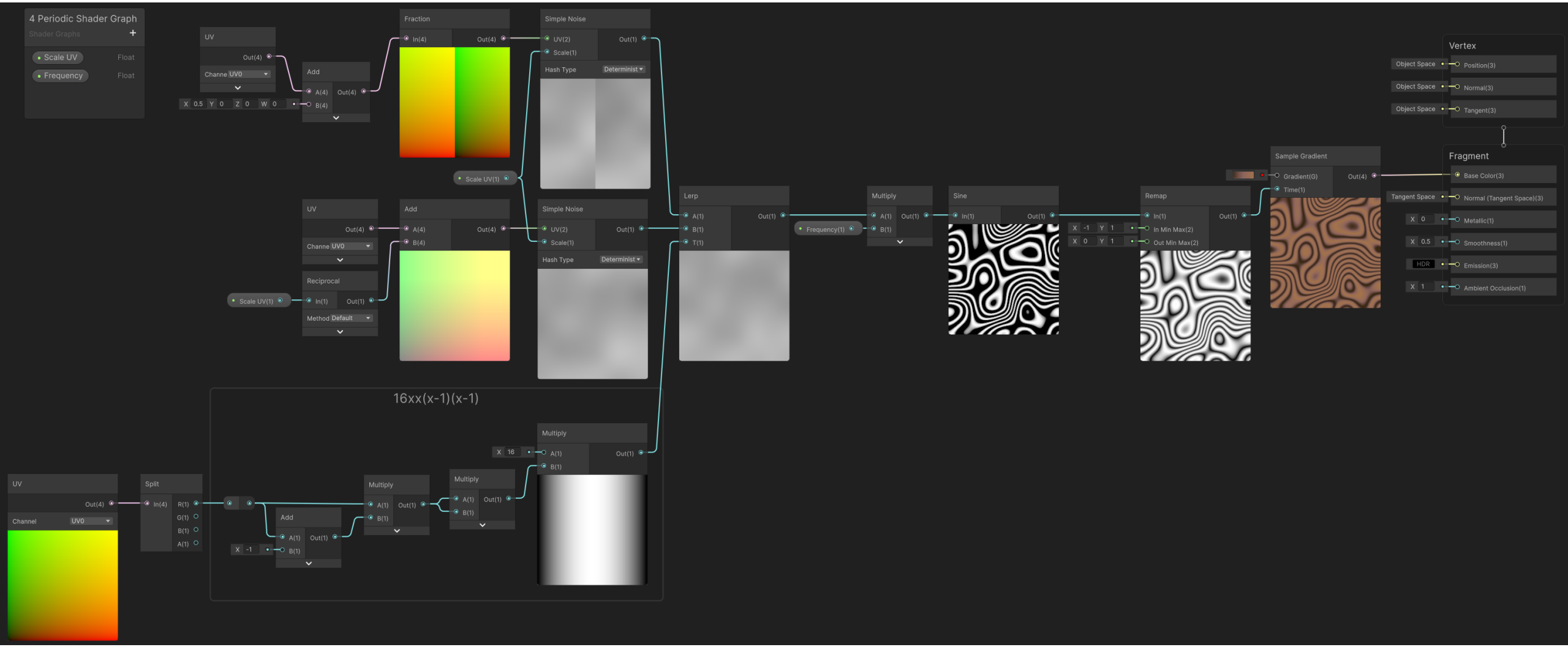


やってみよう



- Shader Graphを作成
  - 「1 Noise/4 Periodic Material」に対して設定
    - オブジェクトは、「4 Periodic Material」で回転するように設定済み
- Shader Graphを実装(ノード構成は次頁)
  - 変数を2つ追加(次は設定例)
    - Frequency
      - Float型 (Mode: Slider)
      - 範囲: [0, 1000]
      - 初期値: 300
    - Scale UV
      - Float型 (Mode: Slider)
      - 範囲: [0, 100]
      - 初期値: 4

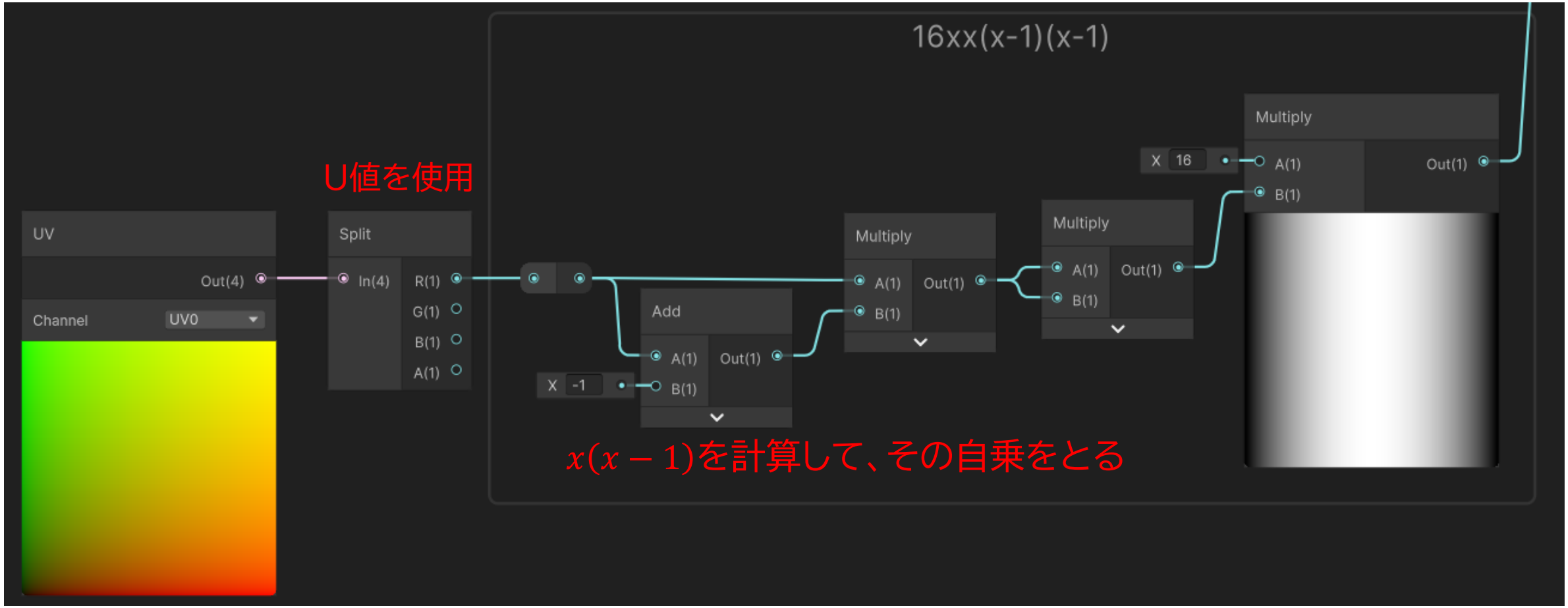




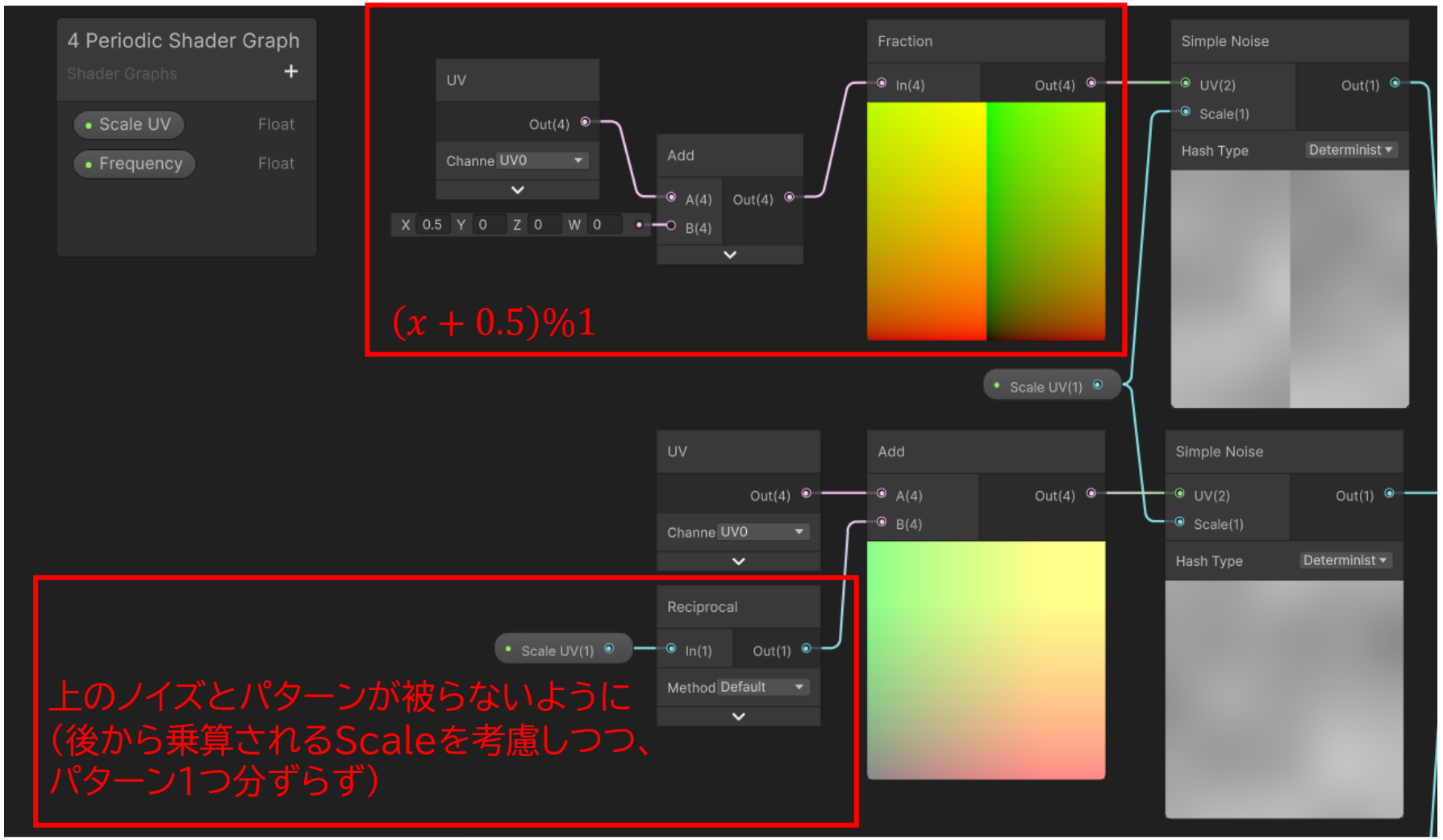
KOGEI TOKYO POLYTECHNIC UNIVERSITY

重みの計算

$$w_1 = 16u^2(u-1)^2$$

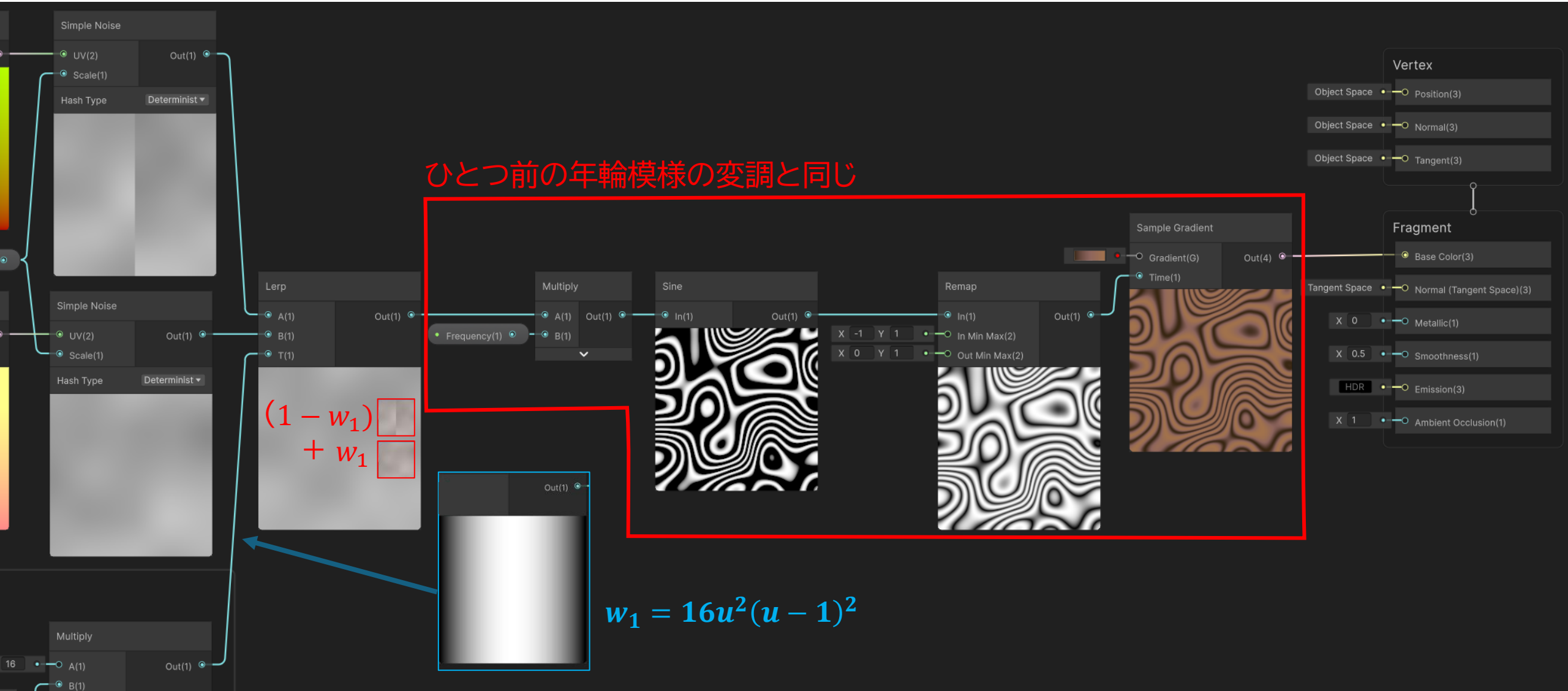


ノイズの生成



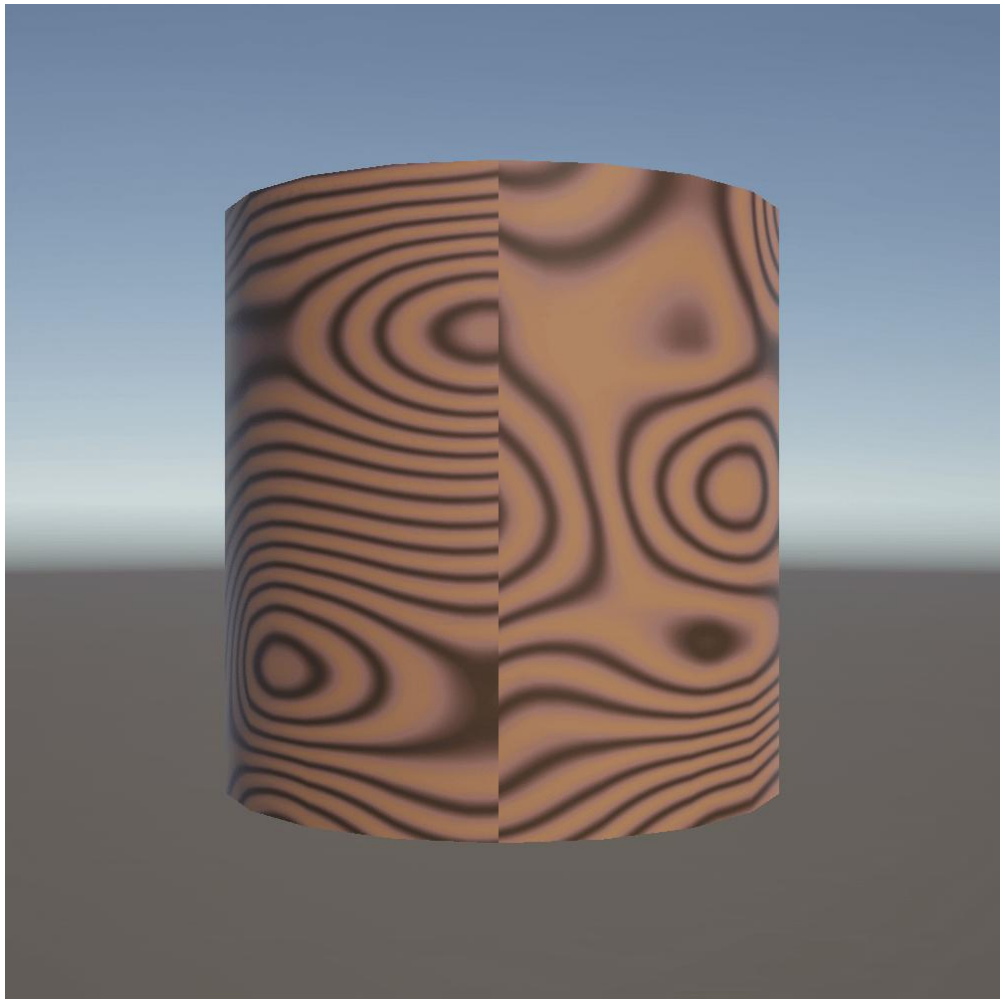
KOGEI

ノイズの合成と加工

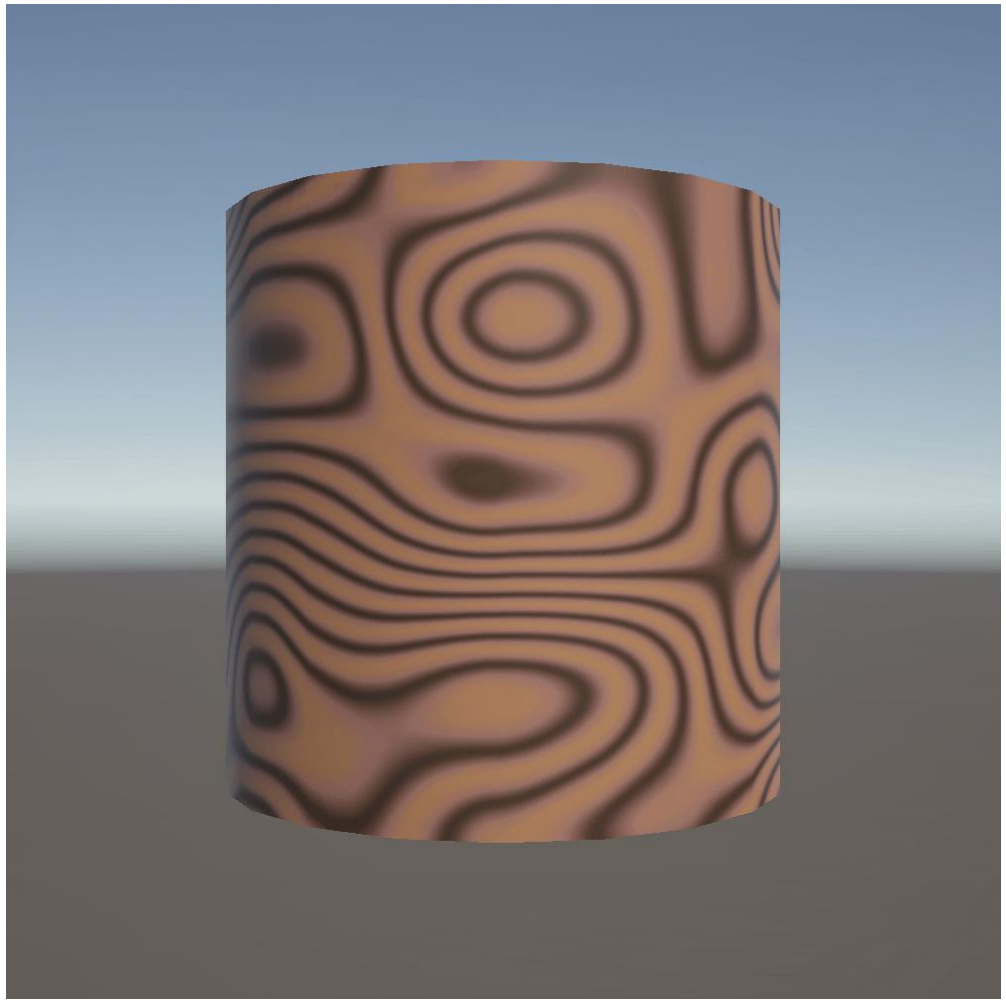

Simple Noise
UV(2)
Scale(1)
Out(1)
Hash Type
Deterministic
Simple Noise
UV(2)
Scale(1)
Out(1)
Hash Type
Deterministic
Lerp
A(1)
B(1)
T(1)
Out(1)
$(1 - w_1)$
$+ w_1$
Out(1)
ひとつ前の年輪模様の変調と同じ
Multiply
A(1)
B(1)
Out(1)
Frequency(1)
Sine
In(1)
Out(1)
X -1 Y 1
X 0 Y 1
Remap
In(1)
In Min Max(2)
Out Min Max(2)
Out(1)
Sample Gradient
Gradient(G)
Time(1)
Out(4)
Vertex
Object Space
Position(3)
Object Space
Normal(3)
Object Space
Tangent(3)
Fragment
Base Color(3)
Tangent Space
Normal (Tangent Space)(3)
X 0
Metallic(1)
X 0.5
Smoothness(1)
HDR
Emission(3)
X 1
Ambient Occlusion(1)
Multiply
16
A(1)
B(1)
Out(1)
$w_1 = 16u^2(u-1)^2$

結果

2Dノイズ



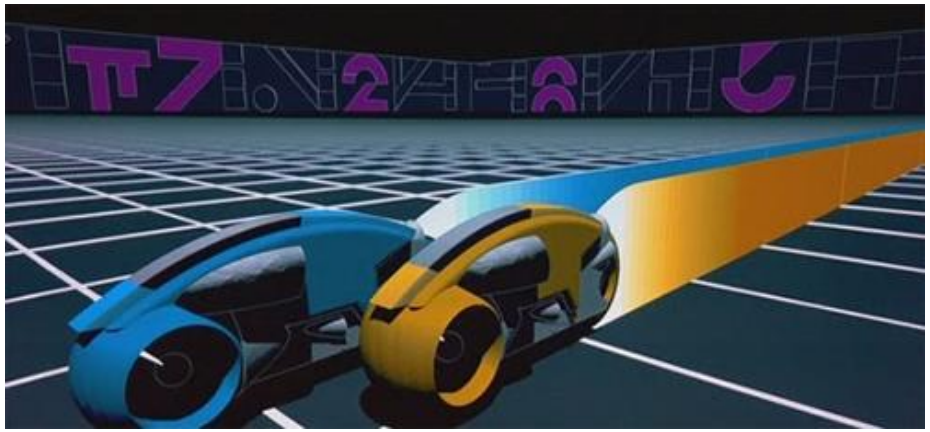
周期的ノイズ



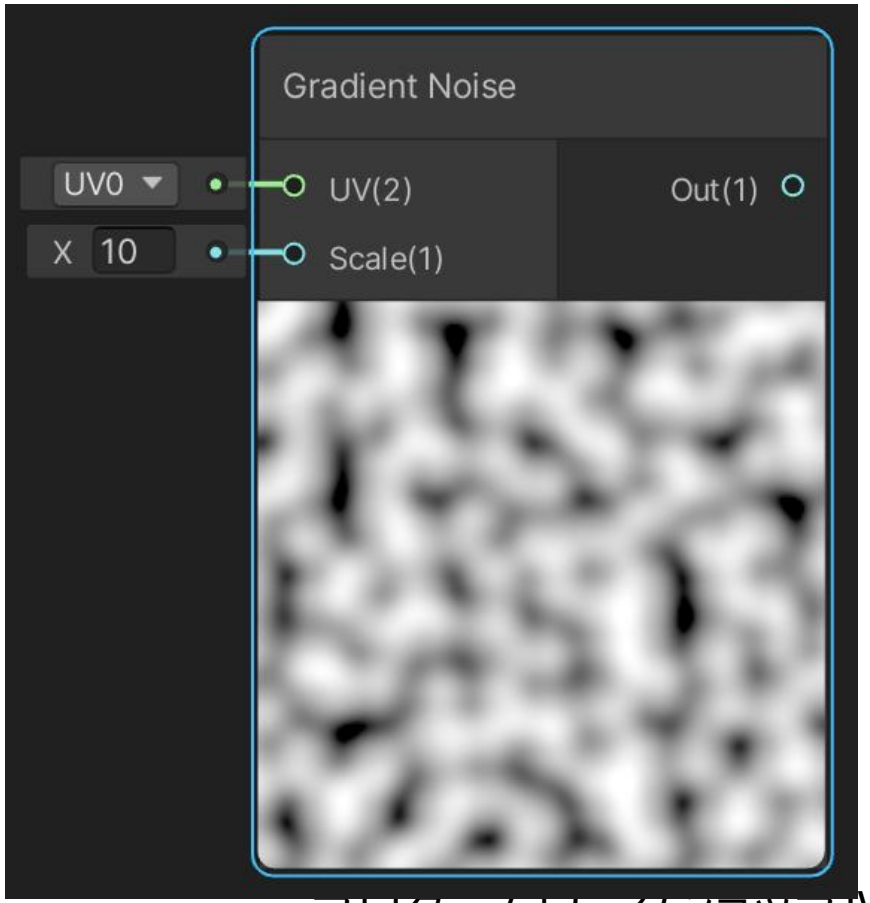
本日の内容

- ノイズの概要
- 基本的なノイズ
  - 一様ノイズ
  - ブロックノイズ
  - 年輪模様
  - 周期的ノイズ
  - Perlinノイズ
  - ボリュームノイズ
- FBM
- 加工したFBM
- ボロノイ図
- 炎
- ディゾルブ

Perlin ノイズ



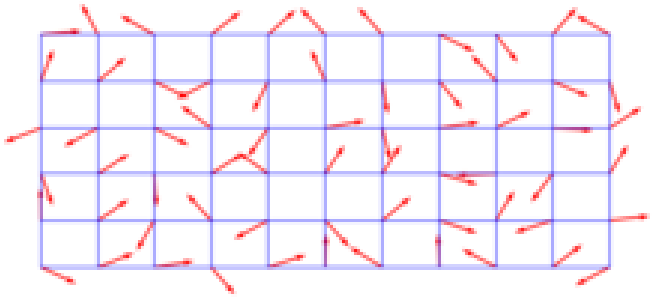
- Ken Perlin
- 映画TRONでの仕事(アカデミー賞)
- UnityではGradientノイズ

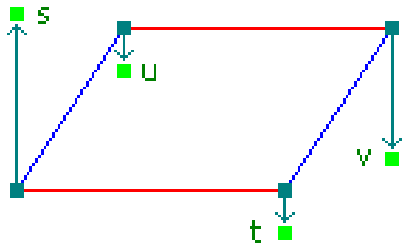


https://blog.form.dev/pixelart/perlin-noise
https://tisch.nyu.edu/itp/itp-people/faculty/adjuncts/ken-perlin.html

生成アルゴリズム

- 空間をグリッドに区切る
- 各頂点に乱数を振る
  - 実際には疑似乱数
- 乱数を空間的に補間





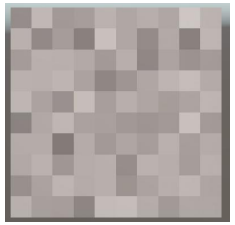
実装例

```
float2 unity_gradientNoise_dir(float2 p)
{
    p = p % 289;
    float x = (34 * p.x + 1) * p.x % 289 + p.y;
    x = (34 * x + 1) * x % 289;
    x = frac(x / 41) * 2 - 1;
    return normalize(float2(x - floor(x + 0.5), abs(x) - 0.5));
}

float unity_gradientNoise(float2 p)
{
    float2 ip = floor(p);
    float2 fp = frac(p);
    float d00 = dot(unity_gradientNoise_dir(ip), fp);
    float d01 = dot(unity_gradientNoise_dir(ip + float2(0, 1)), fp - fl
oat2(0, 1));
    float d10 = dot(unity_gradientNoise_dir(ip + float2(1, 0)), fp - fl
oat2(1, 0));
    float d11 = dot(unity_gradientNoise_dir(ip + float2(1, 1)), fp - fl
oat2(1, 1));
    fp = fp * fp * fp * (fp * (fp * 6 - 15) + 10);
    return lerp(lerp(d00, d01, fp.y), lerp(d10, d11, fp.y), fp.x);
}

void Unity_GradientNoise_float(float2 UV, float Scale, out float Out)
{
    Out = unity_gradientNoise(UV * Scale) + 0.5;
}
```


https://docs.unity3d.com/ja/Packages/com.unity.shadergraph@10.0/manual/Gradient-Noise-Node.html

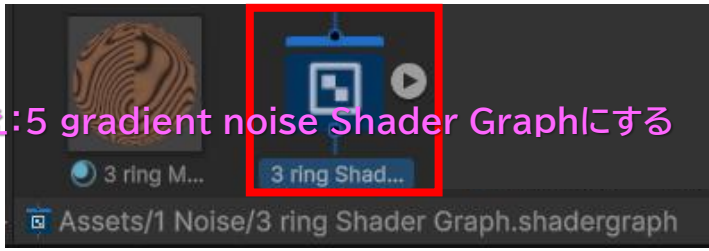
より詳しく知りたい場合

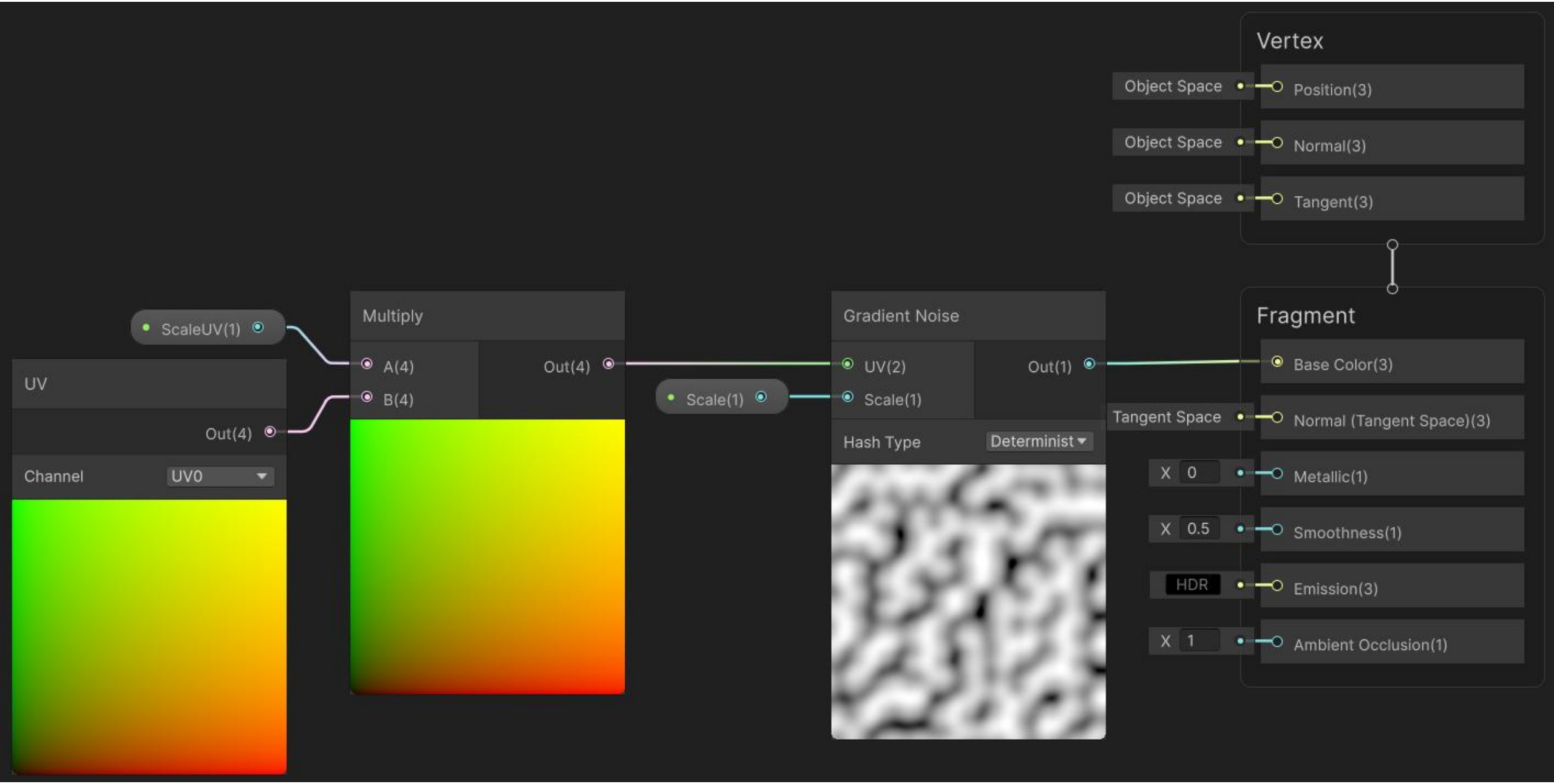
- Ken Perlin, 2002. Improving Noise. ACM Transactions on Graphics (TOG), 21(3), pp. 681–682.
- 詳しい解説記事も存在
  - Adrian Biagioli, 2015. パーリンノイズを理解する. https://postd.cc/understanding-perlin-noise/



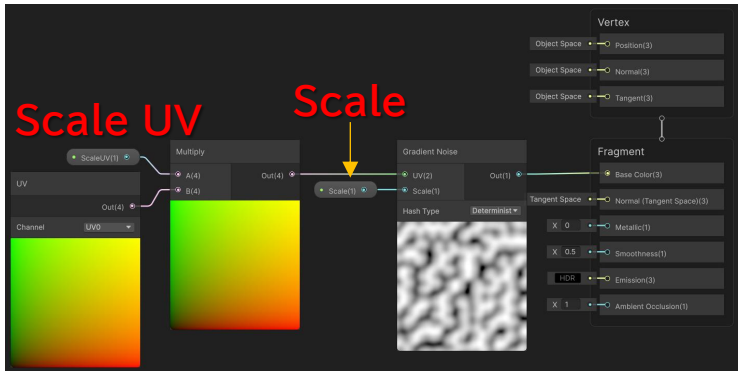
やってみよう

- Shader Graphを作成
  - 「1 Noise/5 gradient noise Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を2つ追加(次は設定例)
    - Scale
      - Float型 (Mode: Slider)
      - 範囲: [0, 100]
      - 初期値: 10
    - Scale UV
      - Float型 (Mode: Slider)
      - 範囲: [0, 10]
      - 初期値: 1

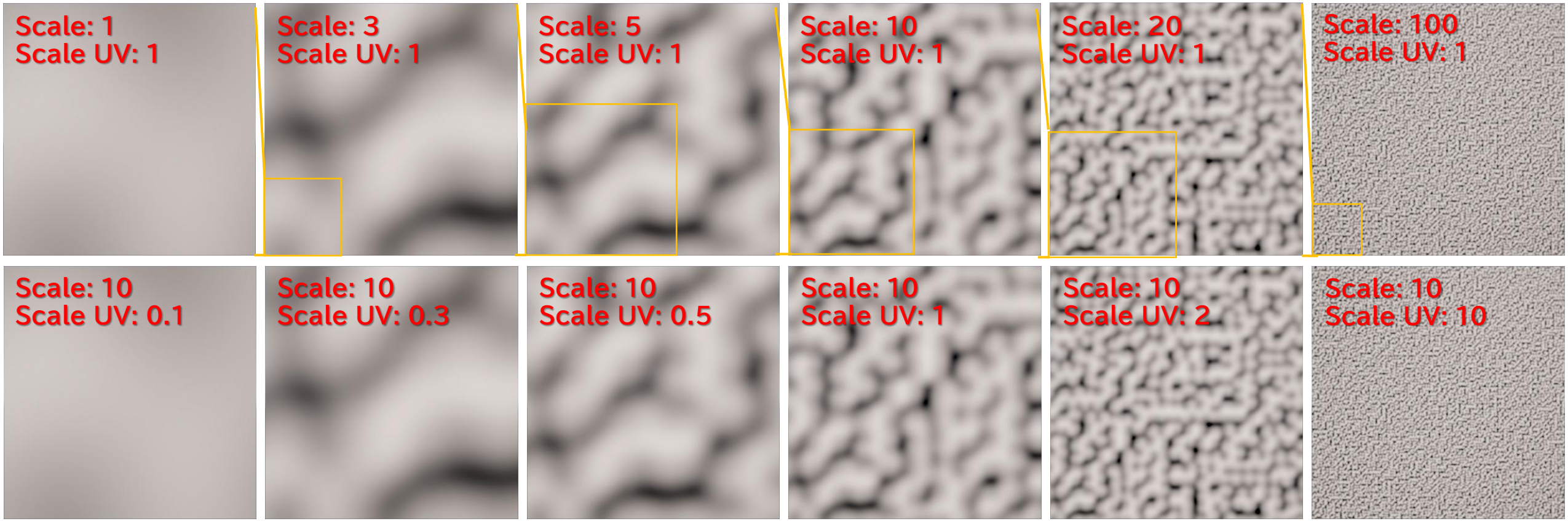



Vertex
Object Space
Position(3)
Object Space
Normal(3)
Object Space
Tangent(3)
Fragment
Base Color(3)
Tangent Space
Normal (Tangent Space)(3)
X 0
Metallic(1)
X 0.5
Smoothness(1)
HDR
Emission(3)
X 1
Ambient Occlusion(1)
Multiply
A(4)
B(4)
Out(4)
ScaleUV(1)
UV
Out(4)
Channel
UV0
Scale(1)
Gradient Noise
UV(2)
Scale(1)
Out(1)
Hash Type
Deterministic

パラメータに伴う変化



- ScaleはScale UVと同じでノイズ間隔を変更する



本日の内容

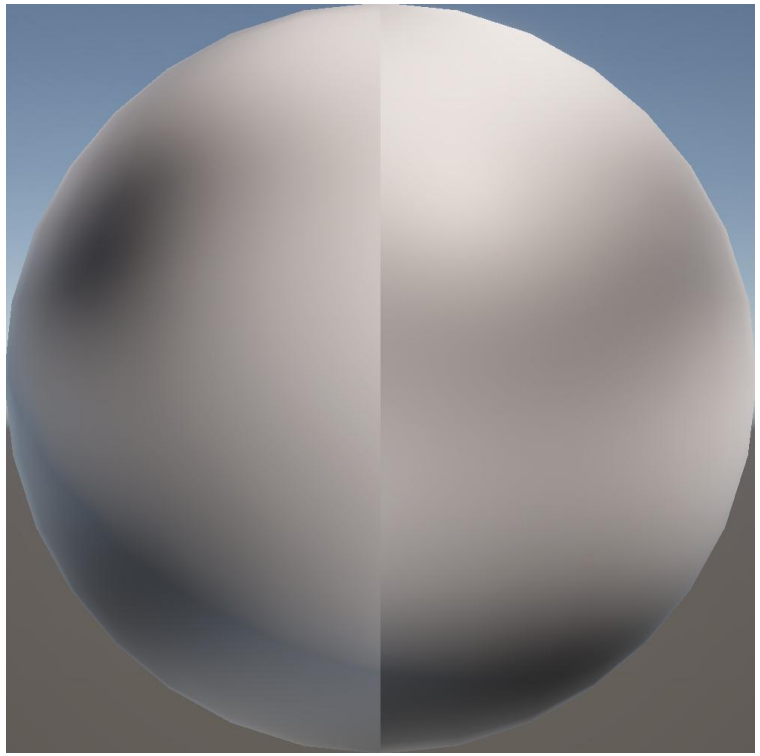
- ノイズの概要
- 基本的なノイズ
  - 一様ノイズ
  - ブロックノイズ
  - 年輪模様
  - 周期的ノイズ
  - Perlinノイズ
  - ボリュームノイズ
- FBM
- 加工したFBM
- ボロノイ図
- 炎
- ディゾルブ

ノイズの欠点再び

球に張った際にきれいにつながらない

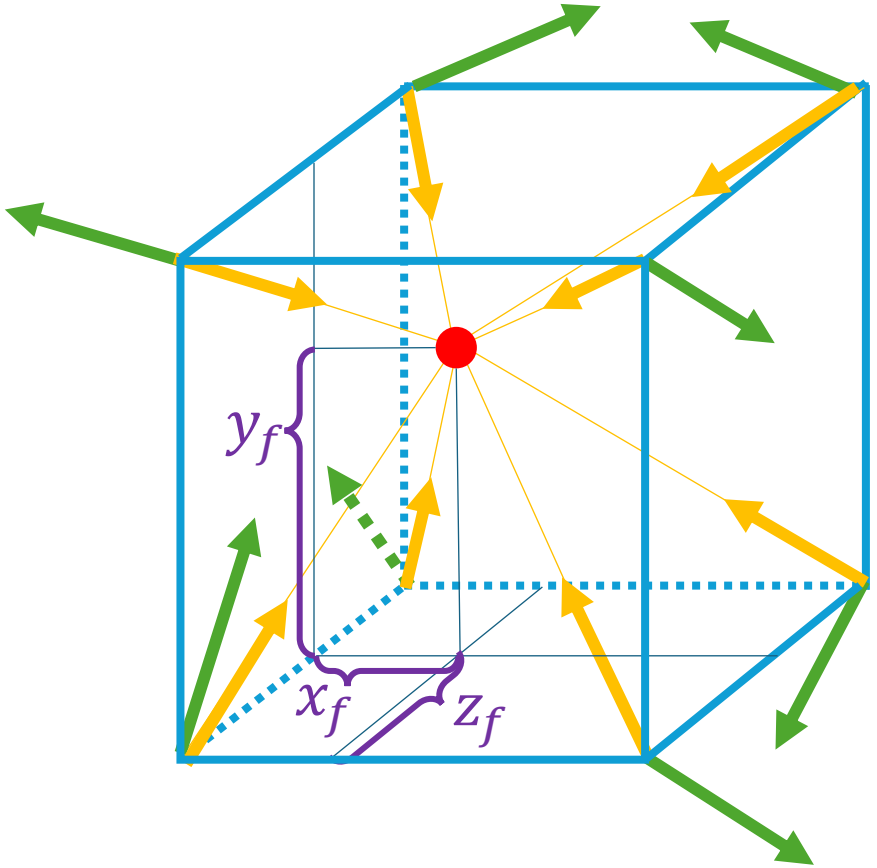
- 他の解決方法：3次元ノイズ
  - ノイズの引数を3つにして、各引数に座標値を与える
    - ノイズ関数は3次元の引数に対して一様なノイズを与える

$$noise(x, y, z)$$



Improved Noise

- 求めたい点の近傍の8頂点について
  - 乱数を設定
  - 求めたい点への向きのベクトルへの乱数の射影を計算
  - 乱数の射影を内挿
    - 内挿のパラメータはなだらかな関数



$$u = fade(x_f)$$
$$v = fade(y_f)$$
$$w = fade(z_f)$$

$$fade(t) = t^3(6t^2 - 15t + 10)$$
$$= t * t * t * (t * (t * 6 - 15) + 10)$$

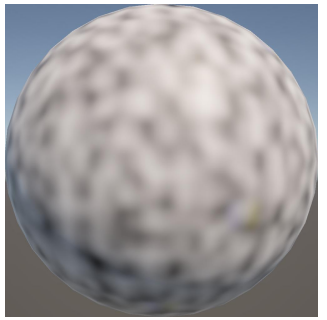
リファレンス実装

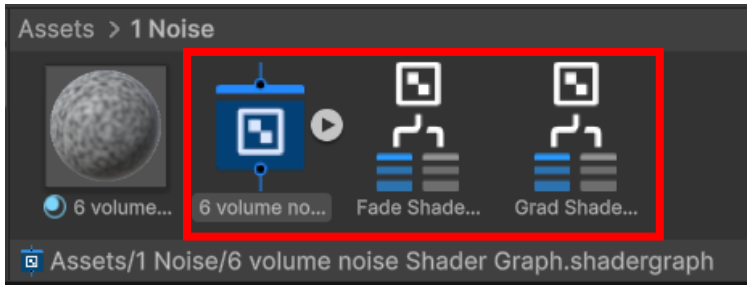
- 著者実装(Java)
- 乱数をハッシュテーブルとして実装
  - 3次元の乱数を作成するのではなく、サンプリング点への内積を取った際の値がランダムになるように1次元の乱数を使用

```
// JAVA REFERENCE IMPLEMENTATION OF IMPROVED NOISE - COPYRIGHT 2002 KEN PERLIN.

public final class ImprovedNoise {
   static public double noise(double x, double y, double z) {
      int X = (int)Math.floor(x) & 255,                  // FIND UNIT CUBE THAT
          Y = (int)Math.floor(y) & 255,                  // CONTAINS POINT.
          Z = (int)Math.floor(z) & 255;
      x -= Math.floor(x);                                // FIND RELATIVE X,Y,Z
      y -= Math.floor(y);                                // OF POINT IN CUBE.
      z -= Math.floor(z);
      double u = fade(x),                                // COMPUTE FADE CURVES
             v = fade(y),                                // FOR EACH OF X,Y,Z.
             w = fade(z);
      int A = p[X  ]+Y, AA = p[A]+Z, AB = p[A+1]+Z,      // HASH COORDINATES OF
          B = p[X+1]+Y, BA = p[B]+Z, BB = p[B+1]+Z;      // THE 8 CUBE CORNERS,

      return lerp(w, lerp(v, lerp(u, grad(p[AA  ], x  , y  , z   ),  // AND ADD
                                     grad(p[BA  ], x-1, y  , z   )), // BLENDED
                             lerp(u, grad(p[AB  ], x  , y-1, z   ),  // RESULTS
                                     grad(p[BB  ], x-1, y-1, z   ))),// FROM  8
                     lerp(v, lerp(u, grad(p[AA+1], x  , y  , z-1 ),  // CORNERS
                                     grad(p[BA+1], x-1, y  , z-1 )), // OF CUBE
                             lerp(u, grad(p[AB+1], x  , y-1, z-1 ),
                                     grad(p[BB+1], x-1, y-1, z-1 ))));
   }
   static double fade(double t) { return t * t * t * (t * (t * 6 - 15) + 10); }
   static double lerp(double t, double a, double b) { return a + t * (b - a); }
   static double grad(int hash, double x, double y, double z) {
      int h = hash & 15;                      // CONVERT LO 4 BITS OF HASH CODE
      double u = h<8 ? x : y,                 // INTO 12 GRADIENT DIRECTIONS.
             v = h<4 ? y : h==12||h==14 ? x : z;
      return ((h&1) == 0 ? u : -u) + ((h&2) == 0 ? v : -v);
   }
   static final int p[] = new int[512], permutation[] = { 151,160,137,91,90,15,
   131,13,201,95,96,53,194,233,7,225,140,36,103,30,69,142,8,99,37,240,21,10,23,
   190,  6,148,247,120,234,75,0,26,197,62,94,252,219,203,117,35,11,32,57,177,33,
   88,237,149,56,87,174,20,125,136,171,168,  68,175,74,165,71,134,139,48,27,166,
   77,146,158,231,83,111,229,122,60,211,133,230,220,105,92,41,55,46,245,40,244,
   102,143,54,  65,25,63,161,  1,216,80,73,209,76,132,187,208,  89,18,169,200,196,
   135,130,116,188,159,86,164,100,109,198,173,186,  3,64,52,217,226,250,124,123,
   5,202,38,147,118,126,255,82,85,212,207,206,59,227,47,16,58,17,182,189,28,42,
   223,183,170,213,119,248,152,  2,44,154,163,  70,221,153,101,155,167,  43,172,9,
   129,22,39,253,  19,98,108,110,79,113,224,232,178,185,  112,104,218,246,97,228,
   251,34,242,193,238,210,144,12,191,179,162,241,  81,51,145,235,249,14,239,107,
   49,192,214,  31,181,199,106,157,184,  84,204,176,115,121,50,45,127,  4,150,254,
   138,236,205,93,222,114,67,29,24,72,243,141,128,195,78,66,215,61,156,180
   };
   static { for (int i=0; i < 256 ; i++) p[256+i] = p[i] = permutation[i]; }
}
```

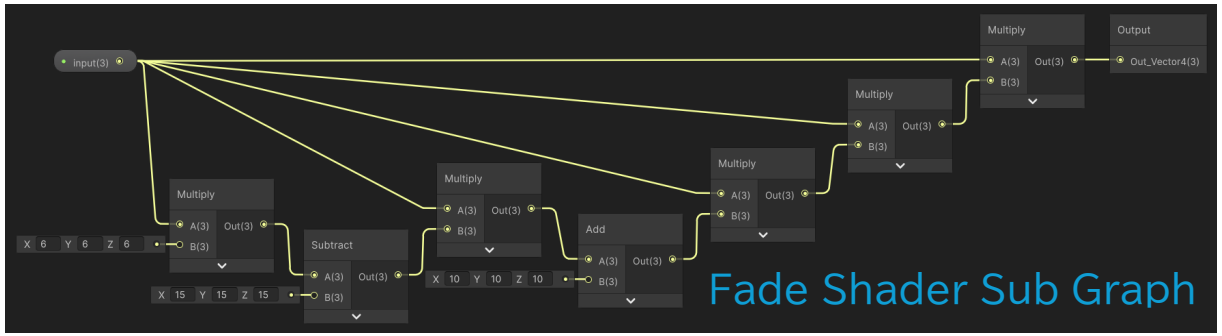
https://cs.nyu.edu/~perlin/noise/

課題

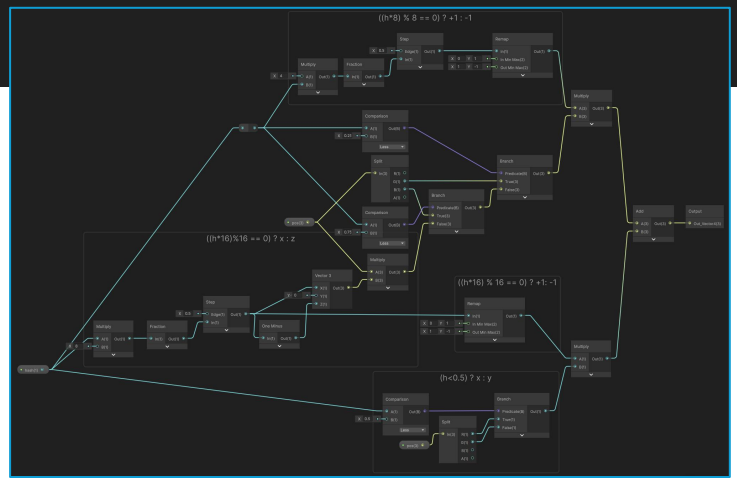




- Shader Graphを作成
  - 「1 Noise/5 gradient noise Material」に設定
- Sub Graphを作成
  - Fade Shader Sub Graph
  - Grad Shader Sub Graph
- Shader Graphを実装(ノード構成は次頁以降)
  - 変数を1つ追加(次は設定例)
    - Scale UV
      - Float型 (Mode: Slider)
      - 範囲: [0, 100]
      - 初期値: 5



Fade Shader Sub Graph

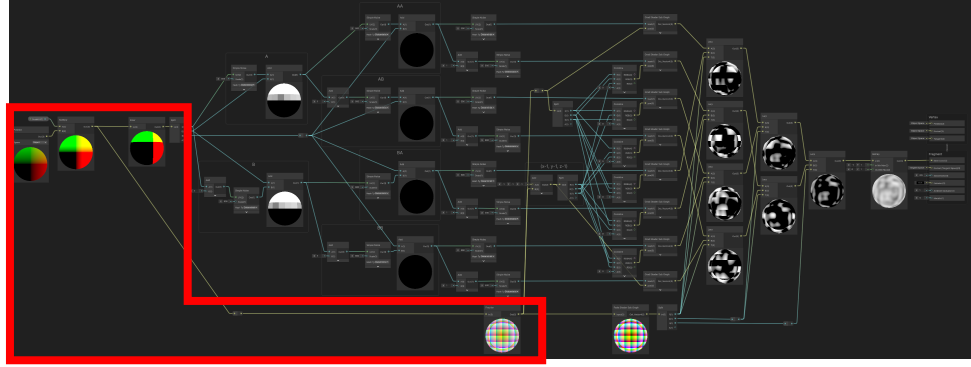


Grad Shader Sub Graph

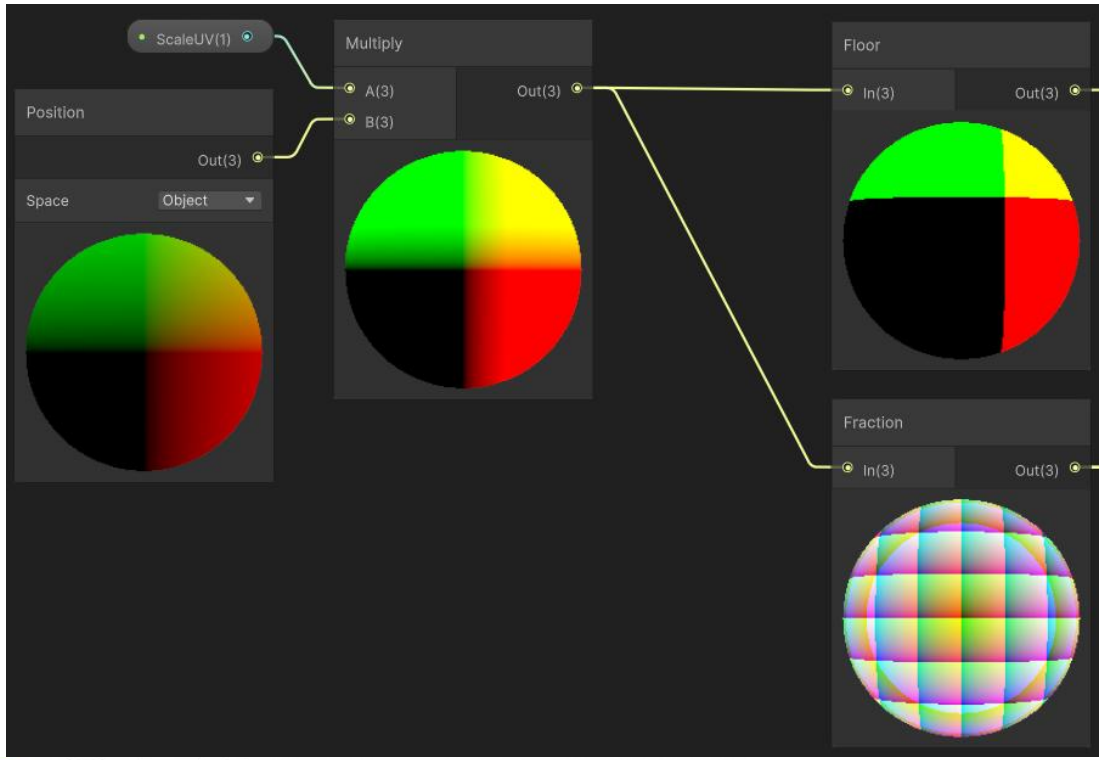
KOGEI TOKYO POLYTECHNIC UNIVERSITY

プログラムワークショップⅣ

整数部と小数部の分離

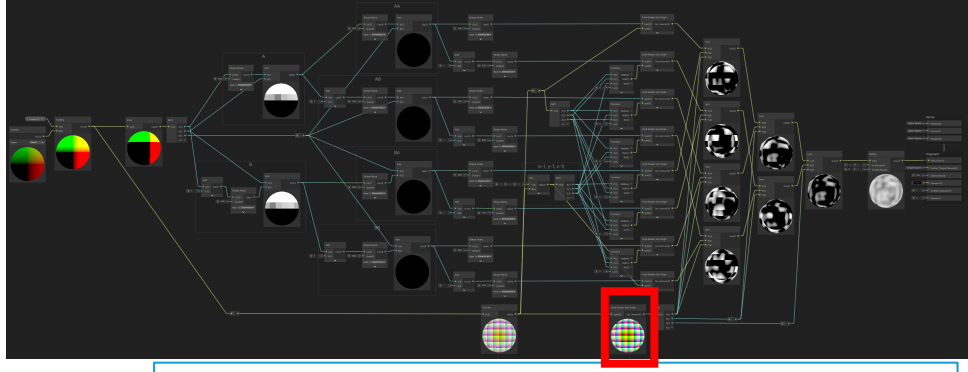


- 整数部：Floor
- 小数部：Frac




```
// JAVA REFERENCE IMPLEMENTATION OF IMPROVED NOISE - COPYRIGHT 2002 KEN PERLIN.

public final class ImprovedNoise {
   static public double noise(double x, double y, double z) {
      int X = (int)Math.floor(x) & 255,                  // FIND UNIT CUBE THAT
          Y = (int)Math.floor(y) & 255,                  // CONTAINS POINT.
          Z = (int)Math.floor(z) & 255;
      x -= Math.floor(x);                                // FIND RELATIVE X,Y,Z
      y -= Math.floor(y);                                // OF POINT IN CUBE.
      z -= Math.floor(z);
      double u = fade(x),                                // COMPUTE FADE CURVES
             v = fade(y),                                // FOR EACH OF X,Y,Z.
             w = fade(z);
      int A = p[X  ]+Y, AA = p[A]+Z, AB = p[A+1]+Z,      // HASH COORDINATES OF
          B = p[X+1]+Y, BA = p[B]+Z, BB = p[B+1]+Z;      // THE 8 CUBE CORNERS,

      return lerp(w, lerp(v, lerp(u, grad(p[AA  ], x  , y  , z   ),  // AND ADD
                                     grad(p[BA  ], x-1, y  , z   )), // BLENDED
                             lerp(u, grad(p[AB  ], x  , y-1, z   ),  // RESULTS
                                     grad(p[BB  ], x-1, y-1, z   ))),// FROM  8
                     lerp(v, lerp(u, grad(p[AA+1], x  , y  , z-1 ),  // CORNERS
                                     grad(p[BA+1], x-1, y  , z-1 )), // OF CUBE
                             lerp(u, grad(p[AB+1], x  , y-1, z-1 ),
                                     grad(p[BB+1], x-1, y-1, z-1 ))));
   }
   static double fade(double t) { return t * t * t * (t * (t * 6 - 15) + 10); }
   static double lerp(double t, double a, double b) { return a + t * (b - a); }
   static double grad(int hash, double x, double y, double z) {
      int h = hash & 15;                      // CONVERT LO 4 BITS OF HASH CODE
      double u = h<8 ? x : y,                 // INTO 12 GRADIENT DIRECTIONS.
             v = h<4 ? y : h==12||h==14 ? x : z;
      return ((h&1) == 0 ? u : -u) + ((h&2) == 0 ? v : -v);
   }
   static final int p[] = new int[512], permutation[] = { 151,160,137,91,90,15,
   131,13,201,95,96,53,194,233,7,225,140,36,103,30,69,142,8,99,37,240,21,10,23,
   190,  6,148,247,120,234,75,0,26,197,62,94,252,219,203,117,35,11,32,57,177,33,
   88,237,149,56,87,174,20,125,136,171,168,  68,175,74,165,71,134,139,48,27,166,
   77,146,158,231,83,111,229,122,60,211,133,230,220,105,92,41,55,46,245,40,244,
   102,143,54,  65,25,63,161,  1,216,80,73,209,76,132,187,208,  89,18,169,200,196,
   135,130,116,188,159,86,164,100,109,198,173,186,  3,64,52,217,226,250,124,123,
   5,202,38,147,118,126,255,82,85,212,207,206,59,227,47,16,58,17,182,189,28,42,
   223,183,170,213,119,248,152,  2,44,154,163,  70,221,153,101,155,167,  43,172,9,
   129,22,39,253,  19,98,108,110,79,113,224,232,178,185,  112,104,218,246,97,228,
   251,34,242,193,238,210,144,12,191,179,162,241,  81,51,145,235,249,14,239,107,
   49,192,214,  31,181,199,106,157,184,  84,204,176,115,121,50,45,127,  4,150,254,
   138,236,205,93,222,114,67,29,24,72,243,141,128,195,78,66,215,61,156,180
   };
   static { for (int i=0; i < 256 ; i++) p[256+i] = p[i] = permutation[i]; }
}
```

内挿パラメータの計算



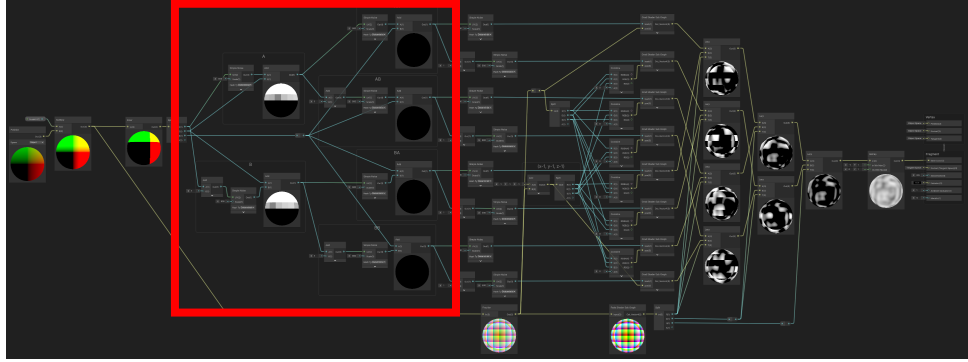
$$fade(t) = t^3(6t^2 - 15t + 10) = t * t * t * (t * (t * 6 - 15) + 10)$$

```
static double fade(double t) { return t * t * t * (t * (t * 6 - 15) + 10); }
```

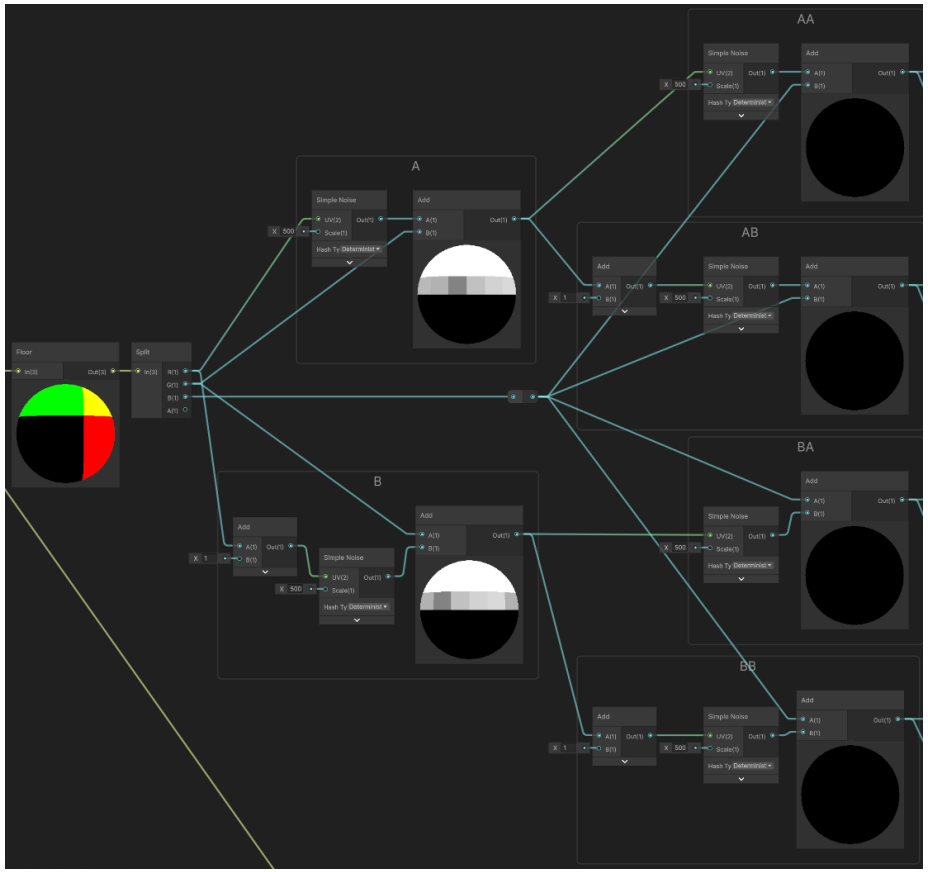
```
// JAVA REFERENCE IMPLEMENTATION OF IMPROVED NOISE - COPYRIGHT 2002 KEN PERLIN.

public final class ImprovedNoise {
   static public double noise(double x, double y, double z) {
      int X = (int)Math.floor(x) & 255,                  // FIND UNIT CUBE THAT
          Y = (int)Math.floor(y) & 255,                  // CONTAINS POINT.
          Z = (int)Math.floor(z) & 255;
      x -= Math.floor(x);                                // FIND RELATIVE X,Y,Z
      y -= Math.floor(y);                                // OF POINT IN CUBE.
      z -= Math.floor(z);
      double u = fade(x),                                // COMPUTE FADE CURVES
             v = fade(y),                                // FOR EACH OF X,Y,Z.
             w = fade(z);
      int A = p[X  ]+Y, AA = p[A]+Z,  AB = p[A+1]+Z,     // HASH COORDINATES OF
          B = p[X+1]+Y, BA = p[B]+Z,  BB = p[B+1]+Z;     // THE 8 CUBE CORNERS,
```

Fade Shader Sub Graph

乱数の生成



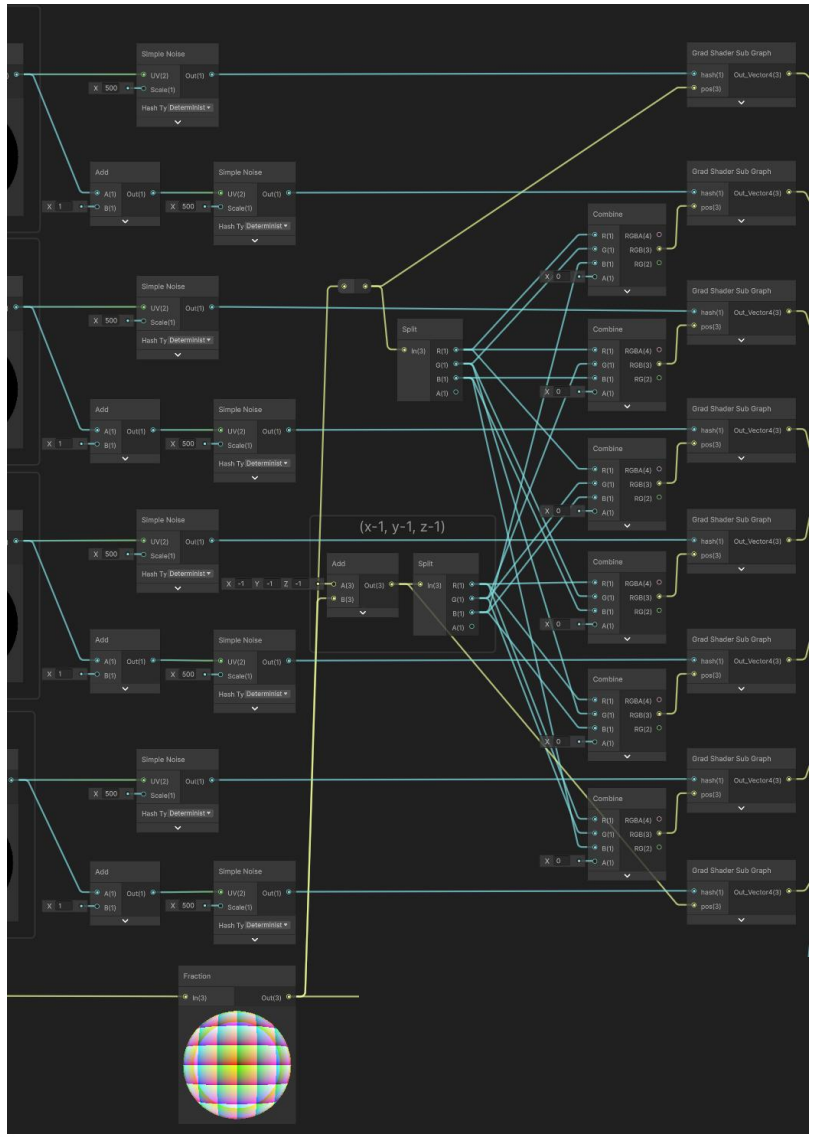
- ハッシュテーブルが手間なので、今回はノイズ関数を利用
  - テクスチャとしてテーブルを用意することで、テーブル参照は可能

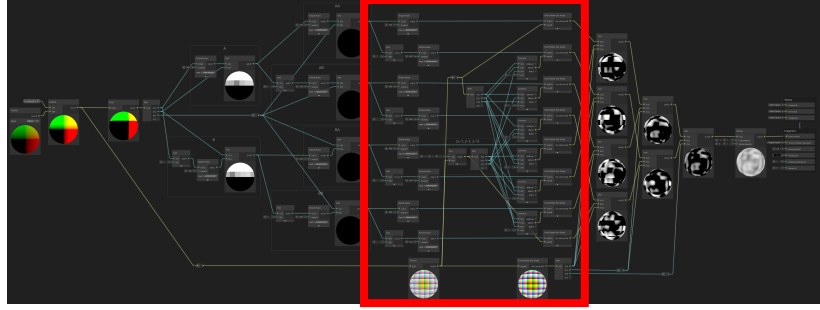



```
// JAVA REFERENCE IMPLEMENTATION OF IMPROVED NOISE - COPYRIGHT 2002 KEN PERLIN.

public final class ImprovedNoise {
   static public double noise(double x, double y, double z) {
      int X = (int)Math.floor(x) & 255,                  // FIND UNIT CUBE THAT
          Y = (int)Math.floor(y) & 255,                  // CONTAINS POINT.
          Z = (int)Math.floor(z) & 255;
      x -= Math.floor(x);                                // FIND RELATIVE X,Y,Z
      y -= Math.floor(y);                                // OF POINT IN CUBE.
      z -= Math.floor(z);
      double u = fade(x),                                // COMPUTE FADE CURVES
             v = fade(y),                                // FOR EACH OF X,Y,Z.
             w = fade(z);
      int A = p[X  ]+Y, AA = p[A]+Z, AB = p[A+1]+Z,      // HASH COORDINATES OF
          B = p[X+1]+Y, BA = p[B]+Z, BB = p[B+1]+Z;      // THE 8 CUBE CORNERS,

      return lerp(w, lerp(v, lerp(u, grad(p[AA  ], x  , y  , z   ),  // AND ADD
                                     grad(p[BA  ], x-1, y  , z   )), // BLENDED
                             lerp(u, grad(p[AB  ], x  , y-1, z   ),  // RESULTS
                                     grad(p[BB  ], x-1, y-1, z   ))),// FROM  8
                     lerp(v, lerp(u, grad(p[AA+1], x  , y  , z-1 ),  // CORNERS
                                     grad(p[BA+1], x-1, y  , z-1 )), // OF CUBE
                             lerp(u, grad(p[AB+1], x  , y-1, z-1 ),
                                     grad(p[BB+1], x-1, y-1, z-1 ))));
   }
   static double fade(double t) { return t * t * t * (t * (t * 6 - 15) + 10); }
   static double lerp(double t, double a, double b) { return a + t * (b - a); }
   static double grad(int hash, double x, double y, double z) {
      int h = hash & 15;                      // CONVERT LO 4 BITS OF HASH CODE
      double u = h<8 ? x : y,                 // INTO 12 GRADIENT DIRECTIONS.
             v = h<4 ? y : h==12||h==14 ? x : z;
      return ((h&1) == 0 ? u : -u) + ((h&2) == 0 ? v : -v);
   }
   static final int p[] = new int[512], permutation[] = { 151,160,137,91,90,15,
   131,13,201,95,96,53,194,233,7,225,140,36,103,30,69,142,8,99,37,240,21,10,23,
   190,  6,148,247,120,234,75,0,26,197,62,94,252,219,203,117,35,11,32,57,177,33,
   88,237,149,56,87,174,20,125,136,171,168,  68,175,74,165,71,134,139,48,27,166,
   77,146,158,231,83,111,229,122,60,211,133,230,220,105,92,41,55,46,245,40,244,
   102,143,54,  65,25,63,161,  1,216,80,73,209,76,132,187,208,  89,18,169,200,196,
   135,130,116,188,159,86,164,100,109,198,173,186,  3,64,52,217,226,250,124,123,
   5,202,38,147,118,126,255,82,85,212,207,206,59,227,47,16,58,17,182,189,28,42,
   223,183,170,213,119,248,152,  2,44,154,163,  70,221,153,101,155,167,  43,172,9,
   129,22,39,253,  19,98,108,110,79,113,224,232,178,185,  112,104,218,246,97,228,
   251,34,242,193,238,210,144,12,191,179,162,241,  81,51,145,235,249,14,239,107,
   49,192,214,  31,181,199,106,157,184,  84,204,176,115,121,50,45,127,  4,150,254,
   138,236,205,93,222,114,67,29,24,72,243,141,128,195,78,66,215,61,156,180
   };
   static { for (int i=0; i < 256 ; i++) p[256+i] = p[i] = permutation[i]; }
}
```

乱数のサンプリング点への射影

- Grad処理はSub Graphで作成
- それぞれの点に対応して、(x,y,z)と(x-1, y-1, z-1)をSplit & Combine





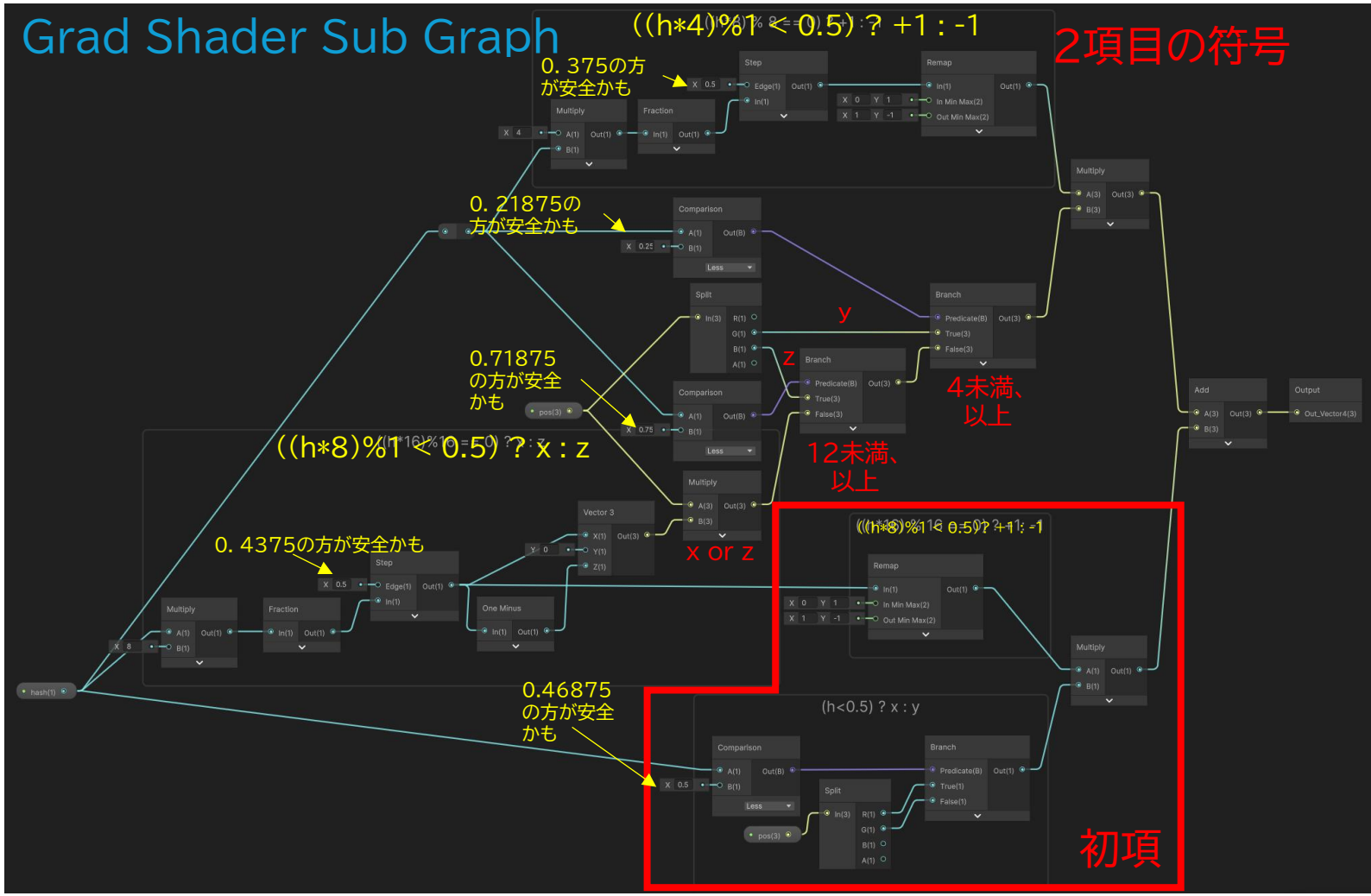
```
// JAVA REFERENCE IMPLEMENTATION OF IMPROVED NOISE - COPYRIGHT 2002 KEN PERLIN.

public final class ImprovedNoise {
   static public double noise(double x, double y, double z) {
      int X = (int)Math.floor(x) & 255,                  // FIND UNIT CUBE THAT
          Y = (int)Math.floor(y) & 255,                  // CONTAINS POINT.
          Z = (int)Math.floor(z) & 255;
      x -= Math.floor(x);                                // FIND RELATIVE X,Y,Z
      y -= Math.floor(y);                                // OF POINT IN CUBE.
      z -= Math.floor(z);
      double u = fade(x),                                // COMPUTE FADE CURVES
             v = fade(y),                                // FOR EACH OF X,Y,Z.
             w = fade(z);
      int A = p[X  ]+Y, AA = p[A]+Z, AB = p[A+1]+Z,      // HASH COORDINATES OF
          B = p[X+1]+Y, BA = p[B]+Z, BB = p[B+1]+Z;      // THE 8 CUBE CORNERS,

      return lerp(w, lerp(v, lerp(u, grad(p[AA  ], x  , y  , z   ),  // AND ADD
                                     grad(p[BA  ], x-1, y  , z   )), // BLENDED
                             lerp(u, grad(p[AB  ], x  , y-1, z   ),  // RESULTS
                                     grad(p[BB  ], x-1, y-1, z   ))),// FROM  8
                     lerp(v, lerp(u, grad(p[AA+1], x  , y  , z-1 ),  // CORNERS
                                     grad(p[BA+1], x-1, y  , z-1 )), // OF CUBE
                             lerp(u, grad(p[AB+1], x  , y-1, z-1 ),
                                     grad(p[BB+1], x-1, y-1, z-1 ))));
   }
   static double fade(double t) { return t * t * t * (t * (t * 6 - 15) + 10); }
   static double lerp(double t, double a, double b) { return a + t * (b - a); }
   static double grad(int hash, double x, double y, double z) {
      int h = hash & 15;                      // CONVERT LO 4 BITS OF HASH CODE
      double u = h<8 ? x : y,                 // INTO 12 GRADIENT DIRECTIONS.
             v = h<4 ? y : h==12||h==14 ? x : z;
      return ((h&1) == 0 ? u : -u) + ((h&2) == 0 ? v : -v);
   }
   static final int p[] = new int[512], permutation[] = { 151,160,137,91,90,15,
   131,13,201,95,96,53,194,233,7,225,140,36,103,30,69,142,8,99,37,240,21,10,23,
   190, 6,148,247,120,234,75,0,26,197,62,94,252,219,203,117,35,11,32,57,177,33,
   88,237,149,56,87,174,20,125,136,171,168, 68,175,74,165,71,134,139,48,27,166,
   77,146,158,231,83,111,229,122,60,211,133,230,220,105,92,41,55,46,245,40,244,
   102,143,54, 65,25,63,161, 1,216,80,73,209,76,132,187,208, 89,18,169,200,196,
   135,130,116,188,159,86,164,100,109,198,173,186, 3,64,52,217,226,250,124,123,
   5,202,38,147,118,126,255,82,85,212,207,206,59,227,47,16,58,17,182,189,28,42,
   223,183,170,213,119,248,152, 2,44,154,163, 70,221,153,101,155,167, 43,172,9,
   129,22,39,253, 19,98,108,110,79,113,224,232,178,185, 112,104,218,246,97,228,
   251,34,242,193,238,210,144,12,191,179,162,241, 81,51,145,235,249,14,239,107,
   49,192,214, 31,181,199,106,157,184, 84,204,176,115,121,50,45,127, 4,150,254,
   138,236,205,93,222,114,67,29,24,72,243,141,128,195,78,66,215,61,156,180
   };
   static { for (int i=0; i < 256 ; i++) p[256+i] = p[i] = permutation[i]; }
}
```

Grad Shader Sub Graph

- リファレンス実装は、複雑だが、分解すると理解しやすい

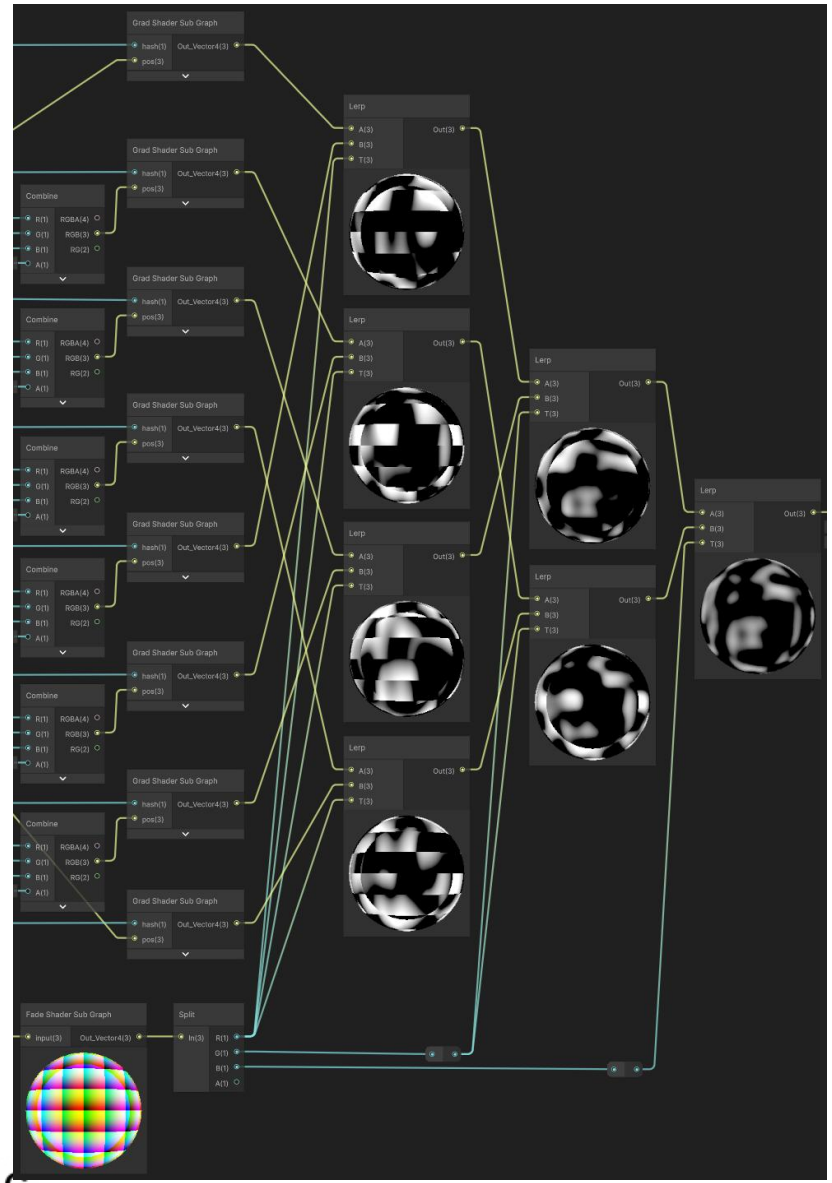
```
static double grad(int hash, double x, double y, double z) {
   int h = hash & 15;                      // CONVERT LO 4 BITS OF HASH CODE
   double u = h<8 ? x : y,                 // INTO 12 GRADIENT DIRECTIONS.
          v = h<4 ? y : h==12||h==14 ? x : z;
   return ((h&1) == 0 ? u : -u) + ((h&2) == 0 ? v : -v);
}
```

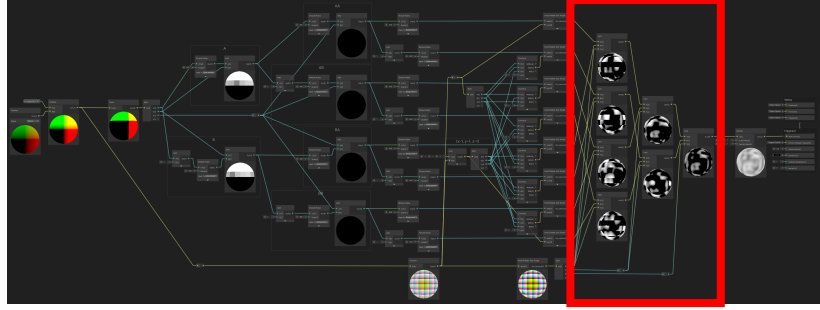
```
static double grad(int hash, double x, double y, double z)
{
   switch(hash & 0xF)
   {
      case 0x0: return  x + y;
      case 0x1: return -x + y;
      case 0x2: return  x - y;
      case 0x3: return -x - y;
      case 0x4: return  x + z;
      case 0x5: return -x + z;
      case 0x6: return  x - z;
      case 0x7: return -x - z;
      case 0x8: return  y + z;
      case 0x9: return -y + z;
      case 0xA: return  y - z;
      case 0xB: return -y - z;
      case 0xC: return  y + x;
      case 0xD: return -y + z;
      case 0xE: return  y - x;
      case 0xF: return -y - z;
   }
}
```



三重線形補間

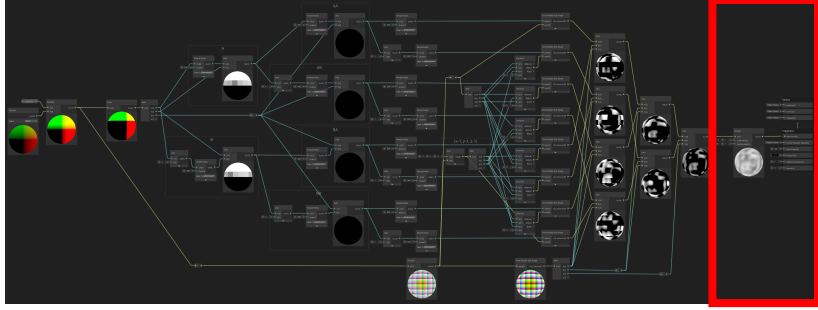
- Fadeを通したパラメータでx，y，z軸に関して線型補間



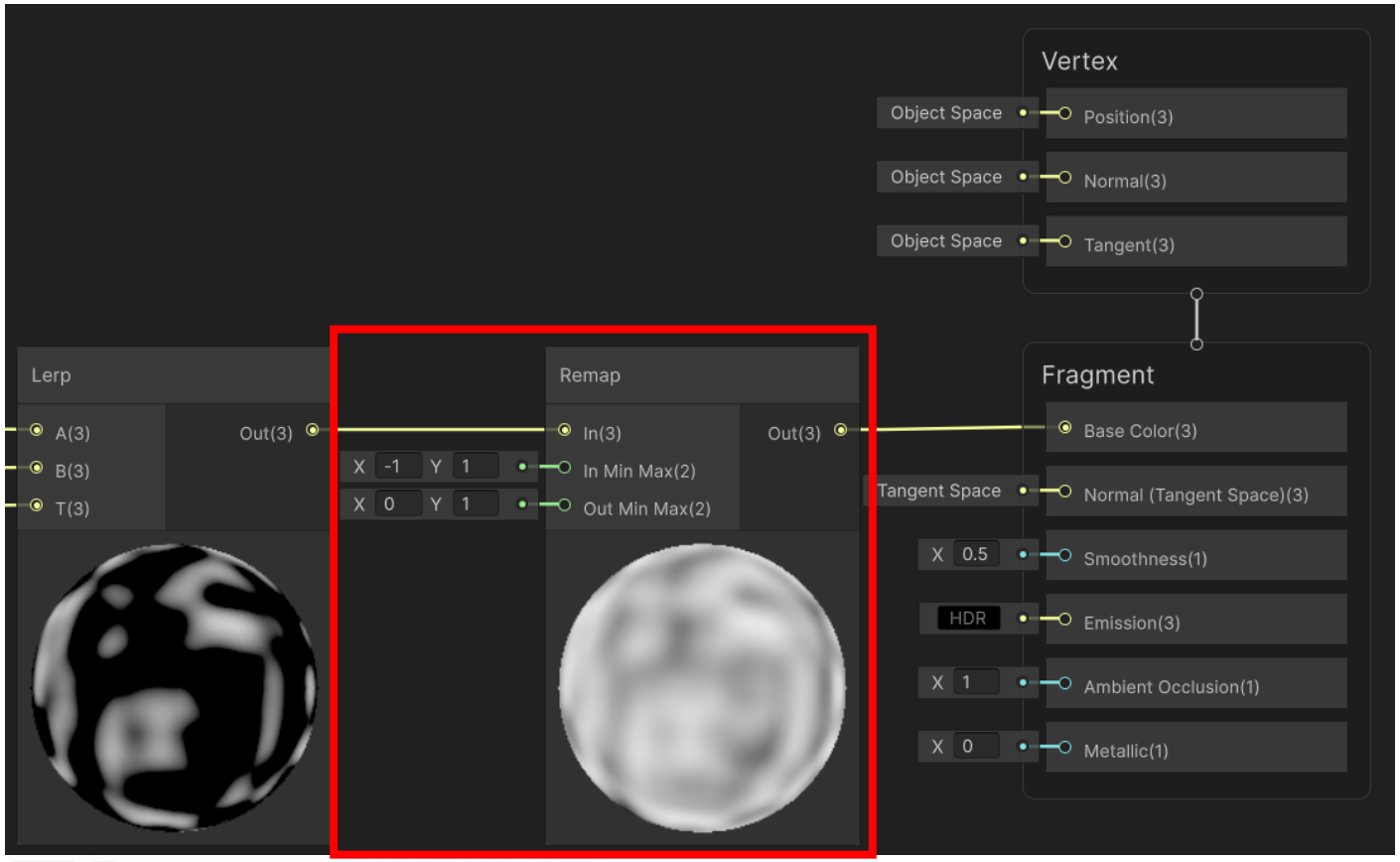


```
// JAVA REFERENCE IMPLEMENTATION OF IMPROVED NOISE - COPYRIGHT 2002 KEN PERLIN.

public final class ImprovedNoise {
   static public double noise(double x, double y, double z) {
      int X = (int)Math.floor(x) & 255,                  // FIND UNIT CUBE THAT
          Y = (int)Math.floor(y) & 255,                  // CONTAINS POINT.
          Z = (int)Math.floor(z) & 255;
      x -= Math.floor(x);                                // FIND RELATIVE X,Y,Z
      y -= Math.floor(y);                                // OF POINT IN CUBE.
      z -= Math.floor(z);
      double u = fade(x),                                // COMPUTE FADE CURVES
             v = fade(y),                                // FOR EACH OF X,Y,Z.
             w = fade(z);
      int A = p[X  ]+Y, AA = p[A]+Z, AB = p[A+1]+Z,      // HASH COORDINATES OF
          B = p[X+1]+Y, BA = p[B]+Z, BB = p[B+1]+Z;      // THE 8 CUBE CORNERS,

      retur lerp(w, lerp(v, lerp(u, grad(p[AA  ], x  , y  , z   ),  // AND ADD
                                    grad(p[BA  ], x-1, y  , z   )), // BLENDED
                            lerp(u, grad(p[AB  ], x  , y-1, z   ),  // RESULTS
                                    grad(p[BB  ], x-1, y-1, z   ))),// FROM  8
                    lerp(v, lerp(u, grad(p[AA+1], x  , y  , z-1 ),  // CORNERS
                                    grad(p[BA+1], x-1, y  , z-1 )), // OF CUBE
                            lerp(u, grad(p[AB+1], x  , y-1, z-1 ),
                                    grad(p[BB+1], x-1, y-1, z-1 ))));
   }
   static double fade(double t) { return t * t * t * (t * (t * 6 - 15) + 10); }
   static double lerp(double t, double a, double b) { return a + t * (b - a); }
   static double grad(int hash, double x, double y, double z) {
      int h = hash & 15;                      // CONVERT LO 4 BITS OF HASH CODE
      double u = h<8 ? x : y,                 // INTO 12 GRADIENT DIRECTIONS.
             v = h<4 ? y : h==12||h==14 ? x : z;
      return ((h&1) == 0 ? u : -u) + ((h&2) == 0 ? v : -v);
   }
   static final int p[] = new int[512], permutation[] = { 151,160,137,91,90,15,
   131,13,201,95,96,53,194,233,7,225,140,36,103,30,69,142,8,99,37,240,21,10,23,
   190, 6,148,247,120,234,75,0,26,197,62,94,252,219,203,117,35,11,32,57,177,33,
   88,237,149,56,87,174,20,125,136,171,168, 68,175,74,165,71,134,139,48,27,166,
   77,146,158,231,83,111,229,122,60,211,133,230,220,105,92,41,55,46,245,40,244,
   102,143,54, 65,25,63,161, 1,216,80,73,209,76,132,187,208, 89,18,169,200,196,
   135,130,116,188,159,86,164,100,109,198,173,186, 3,64,52,217,226,250,124,123,
   5,202,38,147,118,126,255,82,85,212,207,206,59,227,47,16,58,17,182,189,28,42,
   223,183,170,213,119,248,152, 2,44,154,163, 70,221,153,101,155,167, 43,172,9,
   129,22,39,253, 19,98,108,110,79,113,224,232,178,185, 112,104,218,246,97,228,
   251,34,242,193,238,210,144,12,191,179,162,241, 81,51,145,235,249,14,239,107,
   49,192,214, 31,181,199,106,157,184, 84,204,176,115,121,50,45,127, 4,150,254,
   138,236,205,93,222,114,67,29,24,72,243,141,128,195,78,66,215,61,156,180
   };
   static { for (int i=0; i < 256 ; i++) p[256+i] = p[i] = permutation[i]; }
}
```

出力



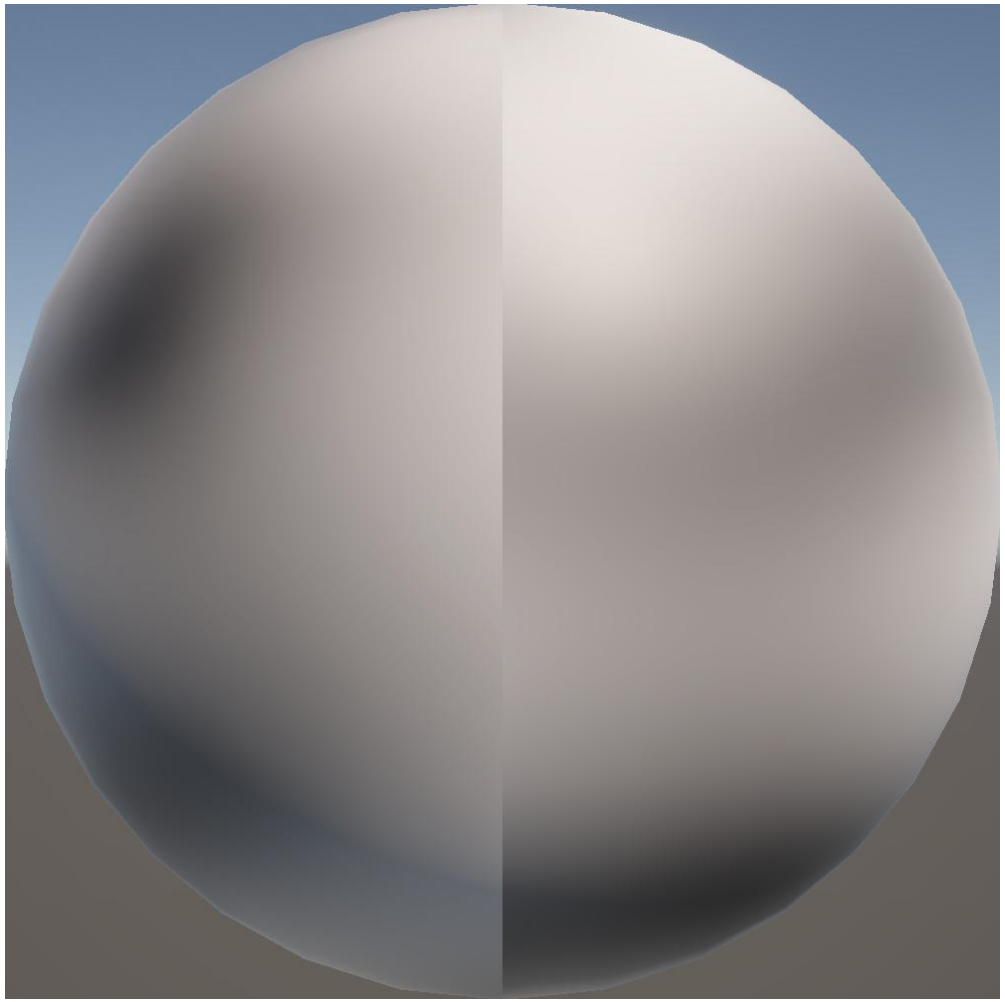
- 見やすいように[0,1]に出力を加工



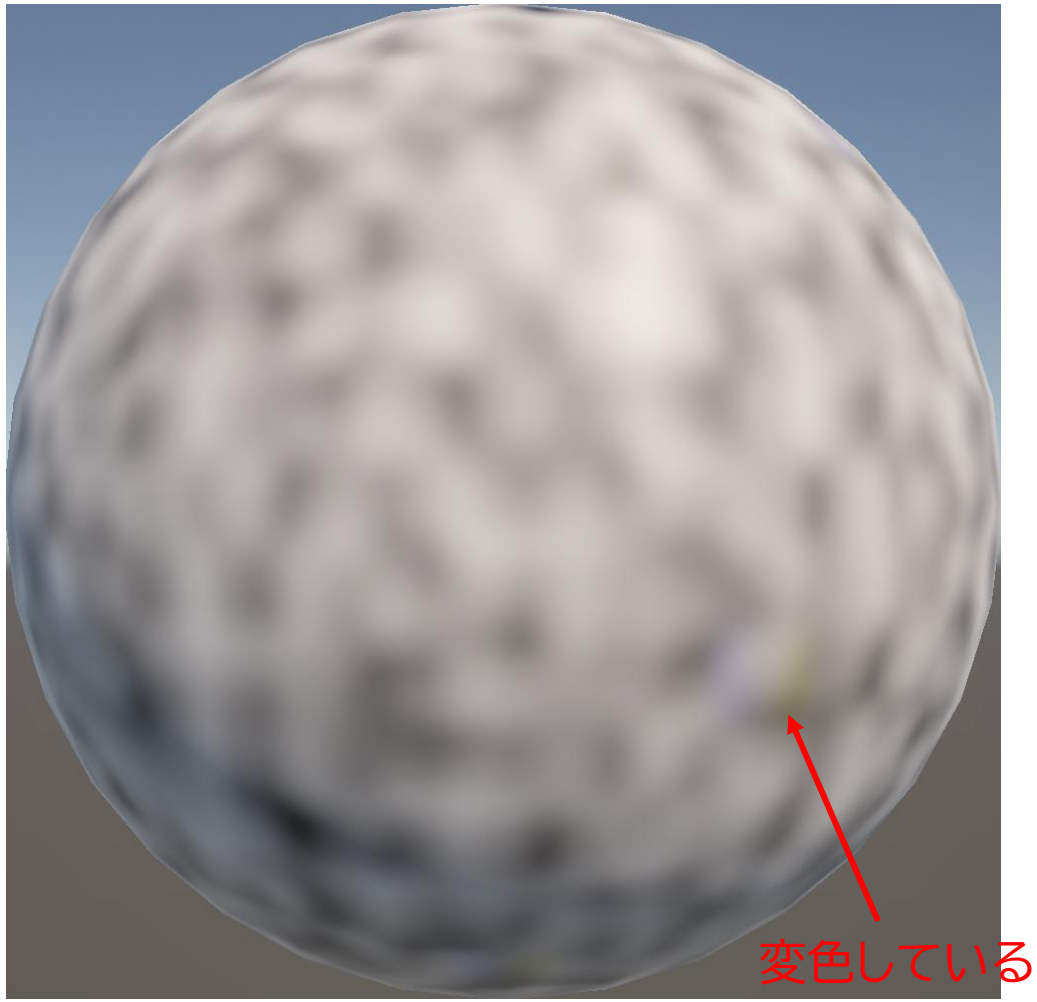

```
// JAVA REFERENCE IMPLEMENTATION OF IMPROVED NOISE - COPYRIGHT 2002 KEN PERLIN.

public final class ImprovedNoise {
   static public double noise(double x, double y, double z) {
      int X = (int)Math.floor(x) & 255,                  // FIND UNIT CUBE THAT
          Y = (int)Math.floor(y) & 255,                  // CONTAINS POINT.
          Z = (int)Math.floor(z) & 255;
      x -= Math.floor(x);                                // FIND RELATIVE X,Y,Z
      y -= Math.floor(y);                                // OF POINT IN CUBE.
      z -= Math.floor(z);
      double u = fade(x),                                // COMPUTE FADE CURVES
             v = fade(y),                                // FOR EACH OF X,Y,Z.
             w = fade(z);
      int A = p[X  ]+Y, AA = p[A]+Z, AB = p[A+1]+Z,      // HASH COORDINATES OF
          B = p[X+1]+Y, BA = p[B]+Z, BB = p[B+1]+Z;      // THE 8 CUBE CORNERS,

      return lerp(w, lerp(v, lerp(u, grad(p[AA  ], x  , y  , z   ),  // AND ADD
                                     grad(p[BA  ], x-1, y  , z   )), // BLENDED
                             lerp(u, grad(p[AB  ], x  , y-1, z   ),  // RESULTS
                                     grad(p[BB  ], x-1, y-1, z   ))),// FROM  8
                     lerp(v, lerp(u, grad(p[AA+1], x  , y  , z-1 ),  // CORNERS
                                     grad(p[BA+1], x-1, y  , z-1 )), // OF CUBE
                             lerp(u, grad(p[AB+1], x  , y-1, z-1 ),
                                     grad(p[BB+1], x-1, y-1, z-1 ))));
   }
   static double fade(double t) { return t * t * t * (t * (t * 6 - 15) + 10); }
   static double lerp(double t, double a, double b) { return a + t * (b - a); }
   static double grad(int hash, double x, double y, double z) {
      int h = hash & 15;                      // CONVERT LO 4 BITS OF HASH CODE
      double u = h<8 ? x : y,                 // INTO 12 GRADIENT DIRECTIONS.
             v = h<4 ? y : h==12||h==14 ? x : z;
      return ((h&1) == 0 ? u : -u) + ((h&2) == 0 ? v : -v);
   }
   static final int p[] = new int[512], permutation[] = { 151,160,137,91,90,15,
   131,13,201,95,96,53,194,233,7,225,140,36,103,30,69,142,8,99,37,240,21,10,23,
   190,  6,148,247,120,234,75,0,26,197,62,94,252,219,203,117,35,11,32,57,177,33,
   88,237,149,56,87,174,20,125,136,171,168,  68,175,74,165,71,134,139,48,27,166,
   77,146,158,231,83,111,229,122,60,211,133,230,220,105,92,41,55,46,245,40,244,
   102,143,54,  65,25,63,161,  1,216,80,73,209,76,132,187,208,  89,18,169,200,196,
   135,130,116,188,159,86,164,100,109,198,173,186,  3,64,52,217,226,250,124,123,
   5,202,38,147,118,126,255,82,85,212,207,206,59,227,47,16,58,17,182,189,28,42,
   223,183,170,213,119,248,152,  2,44,154,163,  70,221,153,101,155,167,  43,172,9,
   129,22,39,253,  19,98,108,110,79,113,224,232,178,185,  112,104,218,246,97,228,
   251,34,242,193,238,210,144,12,191,179,162,241,  81,51,145,235,249,14,239,107,
   49,192,214,  31,181,199,106,157,184,  84,204,176,115,121,50,45,127,  4,150,254,
   138,236,205,93,222,114,67,29,24,72,243,141,128,195,78,66,215,61,156,180
   };
   static { for (int i=0; i < 256 ; i++) p[256+i] = p[i] = permutation[i]; }
}
```

結果

2Dノイズ



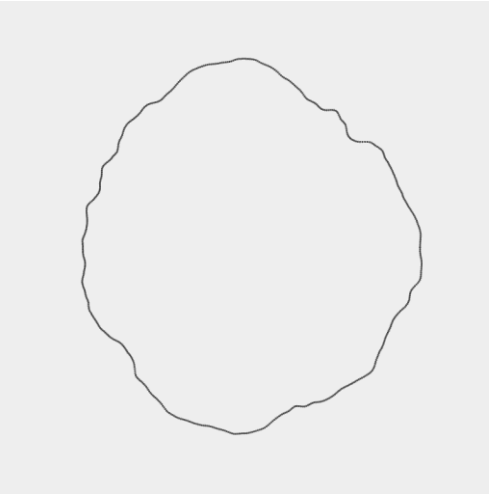
周期的ノイズ



その他の手法

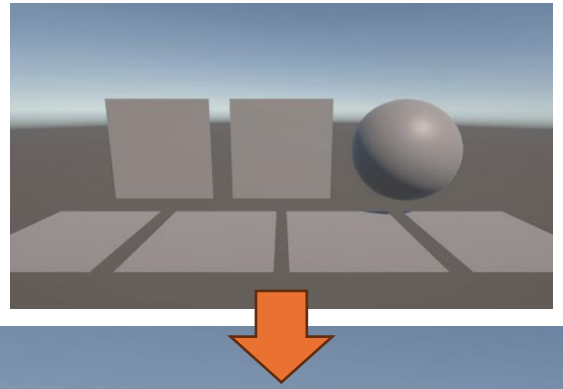
- 4次元ノイズを用意し、パラメータを円軌道にする手法もある
  - UnityではShader Graph Libraryでは用意されていないので、自分で4次元のノイズ関数を作る必要はある

$$noise(r_1 \sin \omega_1 t\,, r_1 \cos \omega_1 t\,, r_2 \sin \omega_2 t\,, r_2 \cos \omega_2 t)$$



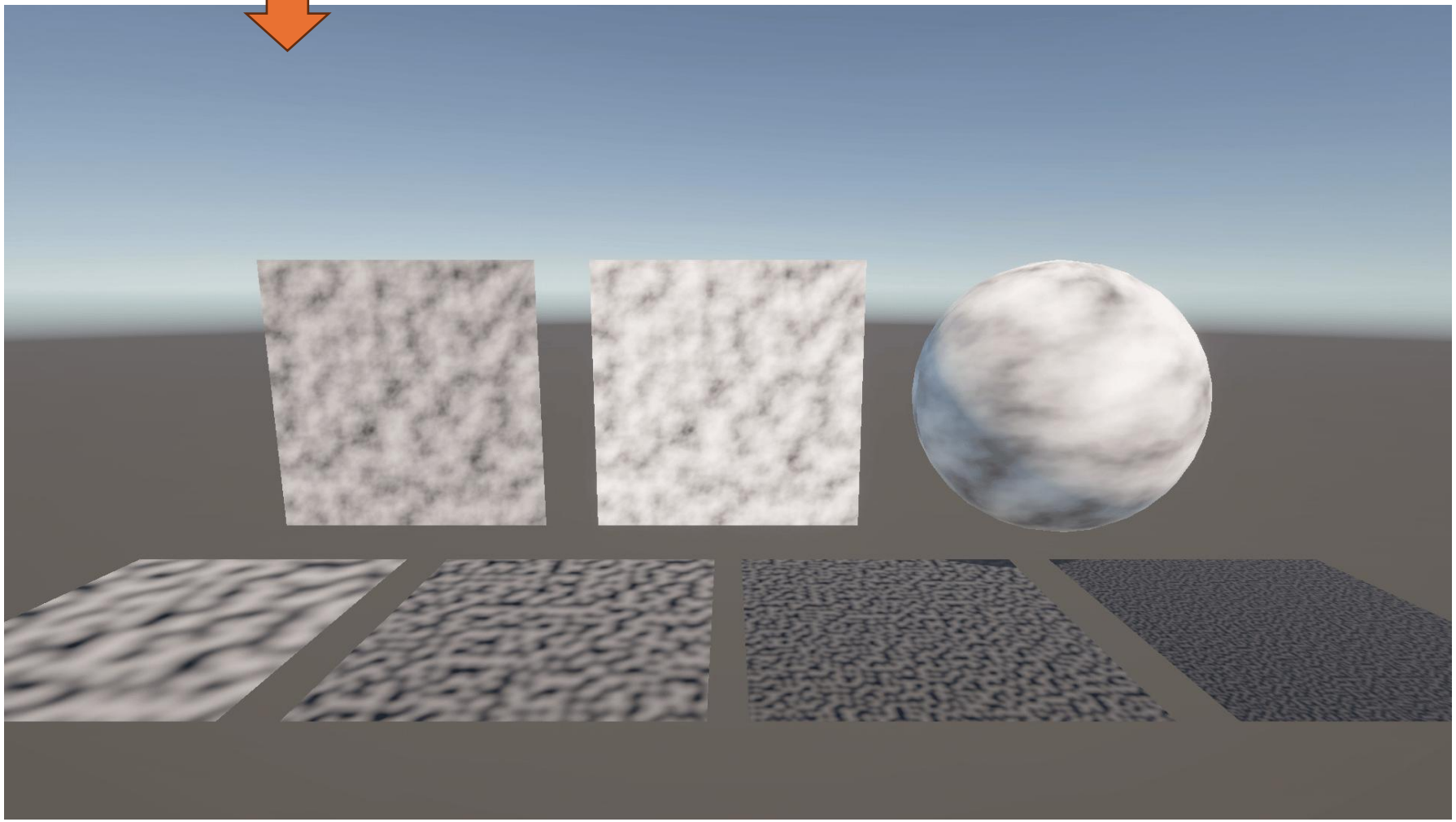
Deconbatch, 周期的パーリンノイズ, note. https://note.com/deconbatch/n/nc14219bfacc6

本日の内容



シーン：2 FBM Scene

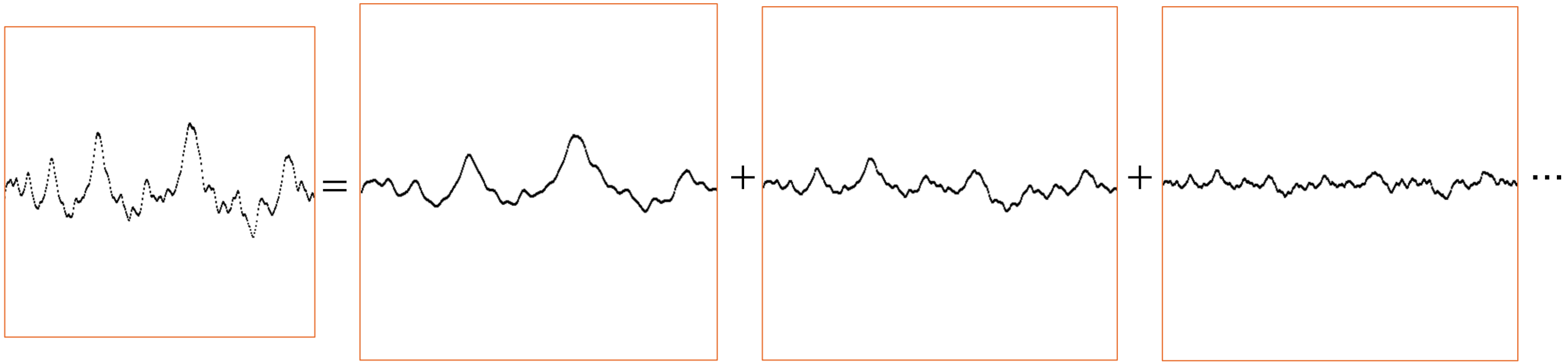
- ノイズの概要
- 基本的なノイズ
- FBM
- 加工したFBM
- ボロノイ図
- 炎
- ディゾルブ



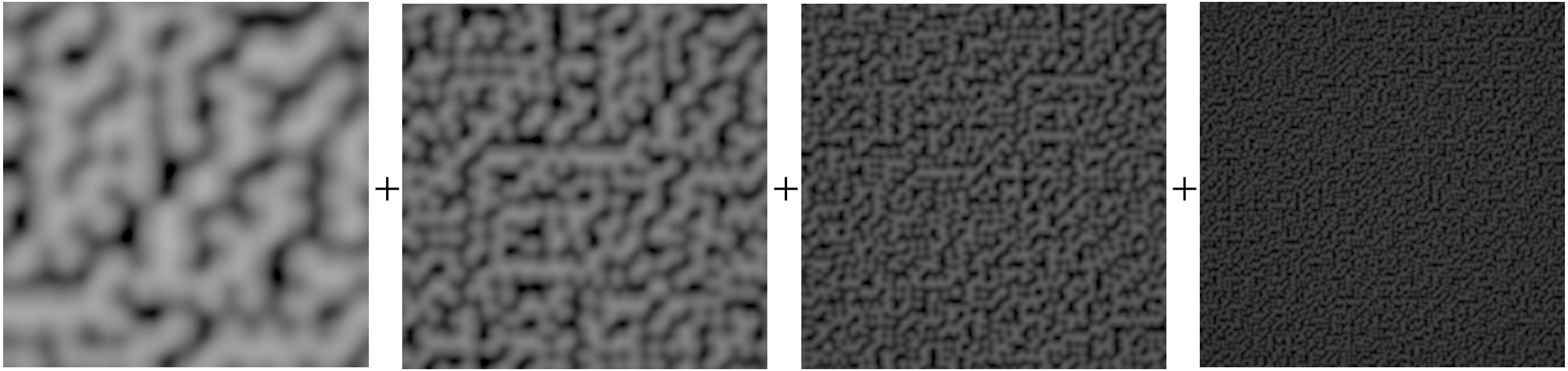
非整数ブラウン運動
fractional Brownian motion, fBm

$$\mathrm{fBm} = \sum_i \frac{noise(B^i x)}{A^i}$$

- 細かさを上げて、最大の幅を小さくした
  ノイズを複数組み合わせる
  - 自己相似性を持つ
  - 山脈など、自然の形状の再現に向く

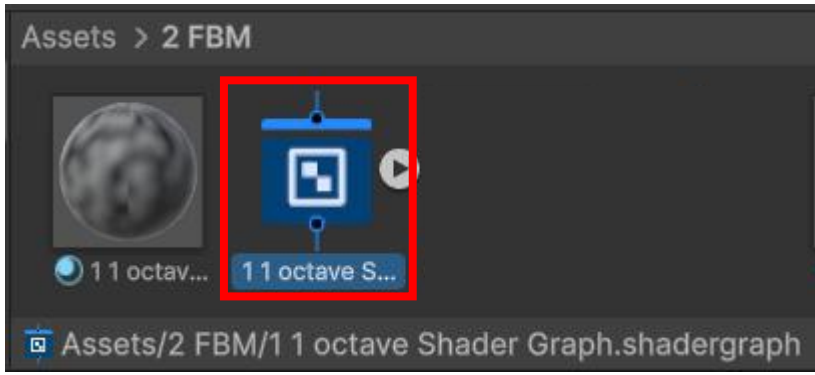


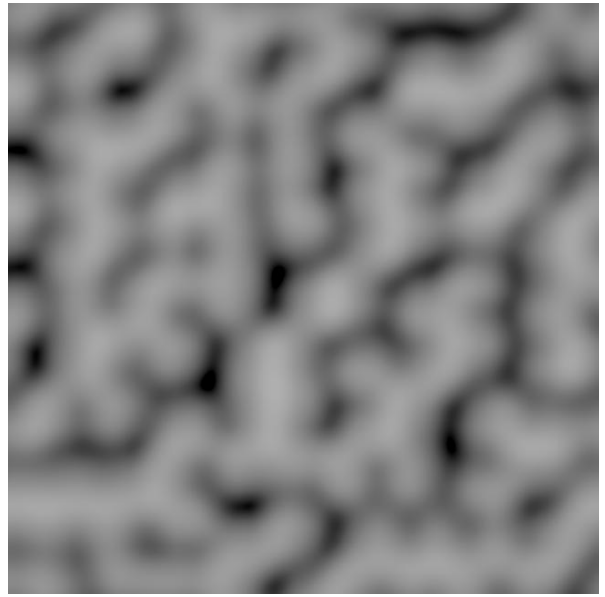
2次元の非整数ブラウン運動

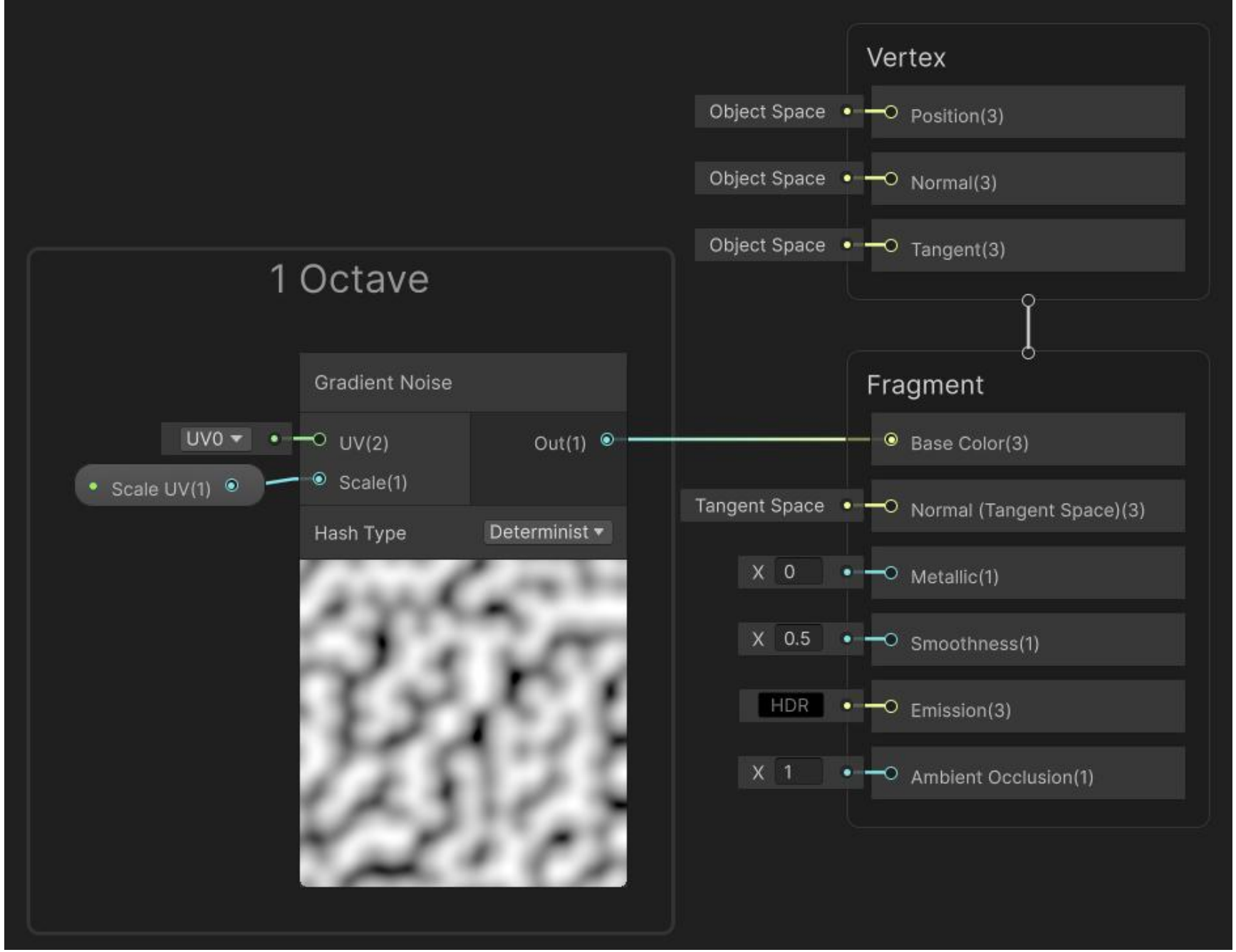


やってみよう: その1

- Shader Graphを作成
  - 「2 FBM/1 1 octave Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を1つ追加(次は設定例)
    - Scale UV
      - Float型 (Mode: Slider)
      - 範囲: [0, 100]
      - 初期値: 10

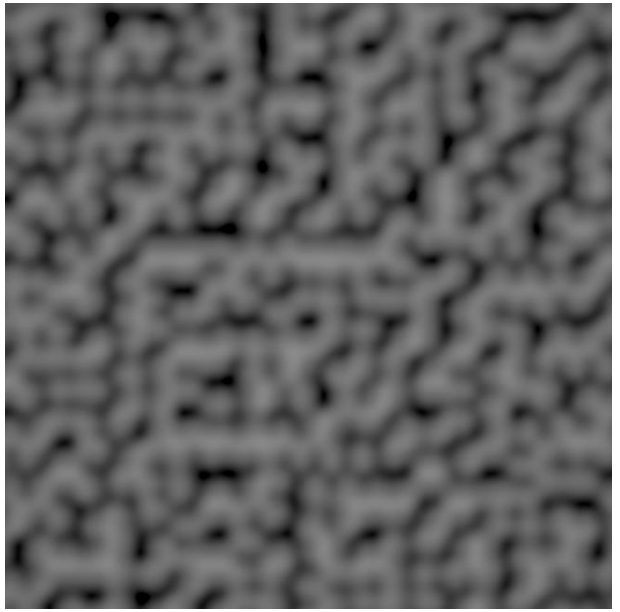





Vertex
Object Space
Position(3)
Object Space
Normal(3)
Object Space
Tangent(3)
1 Octave
Gradient Noise
UV0
UV(2)
Out(1)
Scale UV(1)
Scale(1)
Hash Type
Deterministi
Fragment
Base Color(3)
Tangent Space
Normal (Tangent Space)(3)
X 0
Metallic(1)
X 0.5
Smoothness(1)
HDR
Emission(3)
X 1
Ambient Occlusion(1)

やってみよう: その2

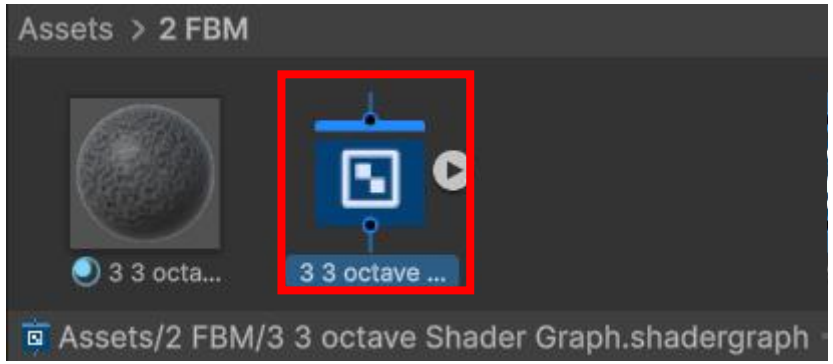
- Shader Graphを作成
  - 「2 FBM/2 2 octave Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を1つ追加(次は設定例)
    - Scale UV
      - Float型 (Mode: Slider)
      - 範囲: [0, 100]
      - 初期値: 10

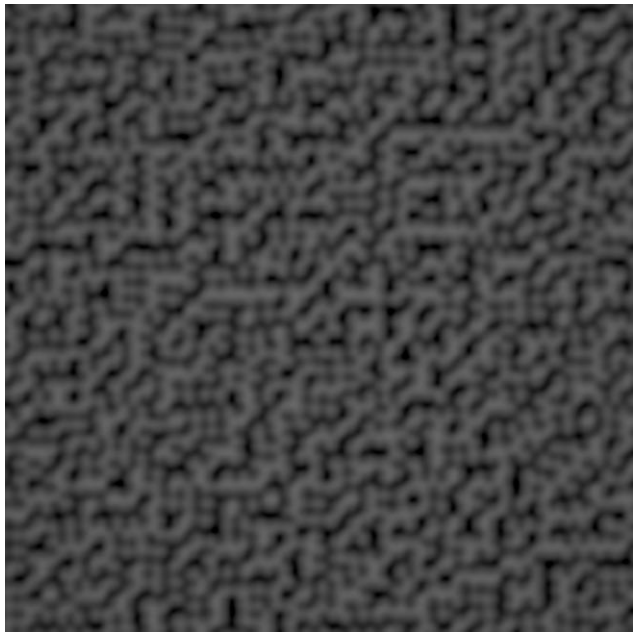


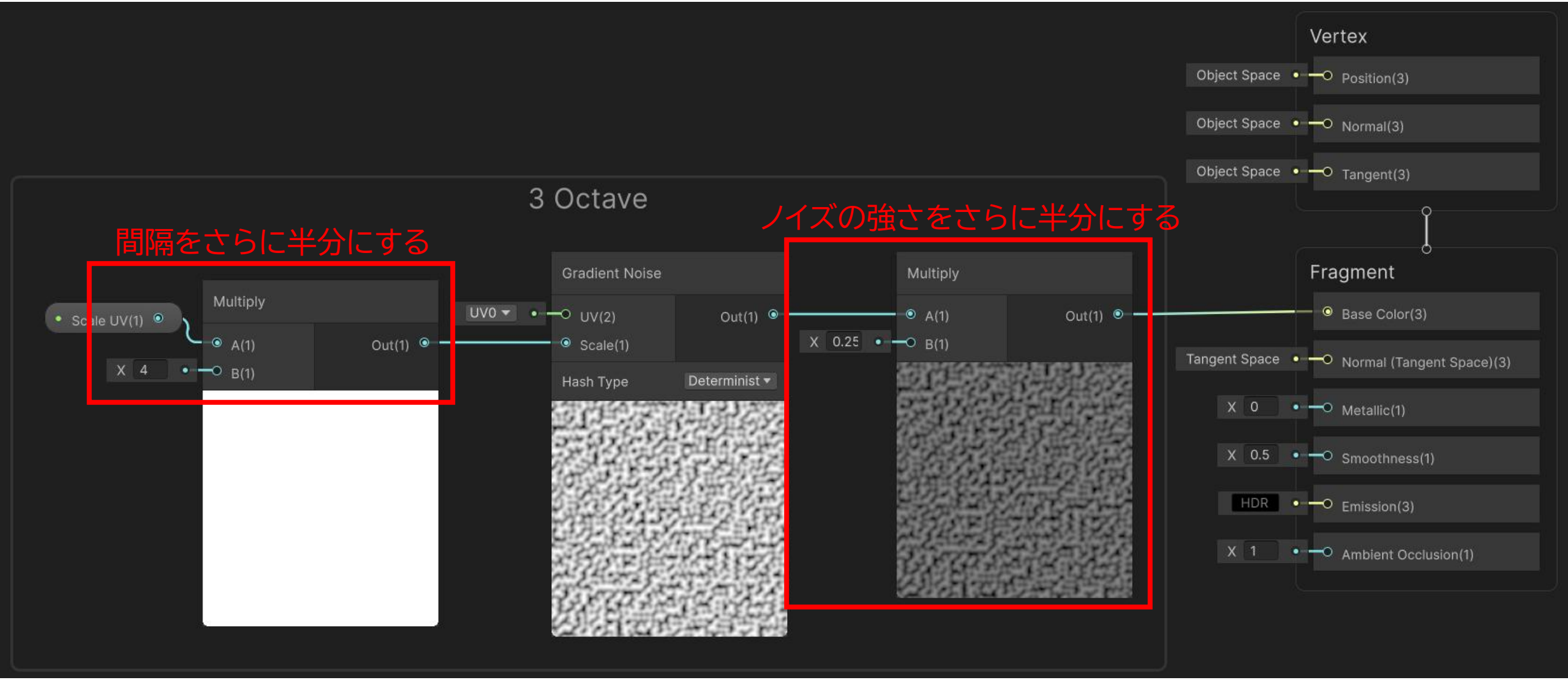

Vertex
Object Space
Position(3)
Object Space
Normal(3)
Object Space
Tangent(3)
2 Octave
間隔を半分にする
ノイズの強さを半分にする
Multiply
A(1)
B(1)
Out(1)
X 2
Gradient Noise
UV0
UV(2)
Scale(1)
Out(1)
Hash Type
Deterministic
Multiply
A(1)
B(1)
Out(1)
X 0.5
Fragment
Base Color(3)
Tangent Space
Normal (Tangent Space)(3)
X 0
Metallic(1)
X 0.5
Smoothness(1)
HDR
Emission(3)
X 1
Ambient Occlusion(1)

やってみよう: その3

- Shader Graphを作成
  - 「2 FBM/3 3 octave Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を1つ追加(次は設定例)
    - Scale UV
      - Float型 (Mode: Slider)
      - 範囲: [0, 100]
      - 初期値: 10

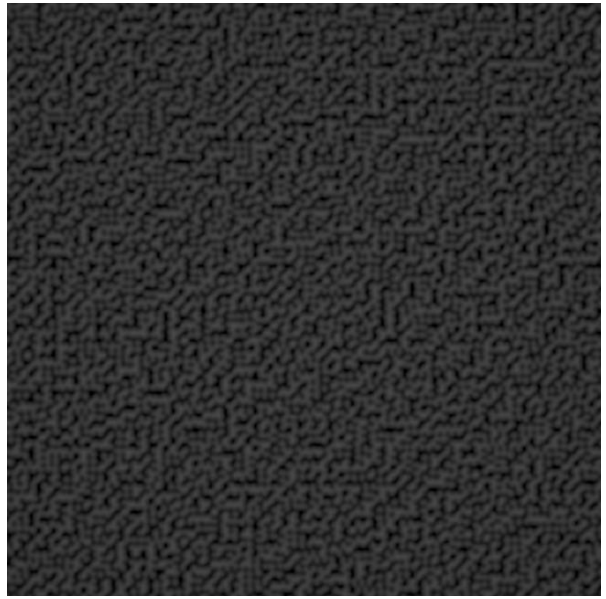


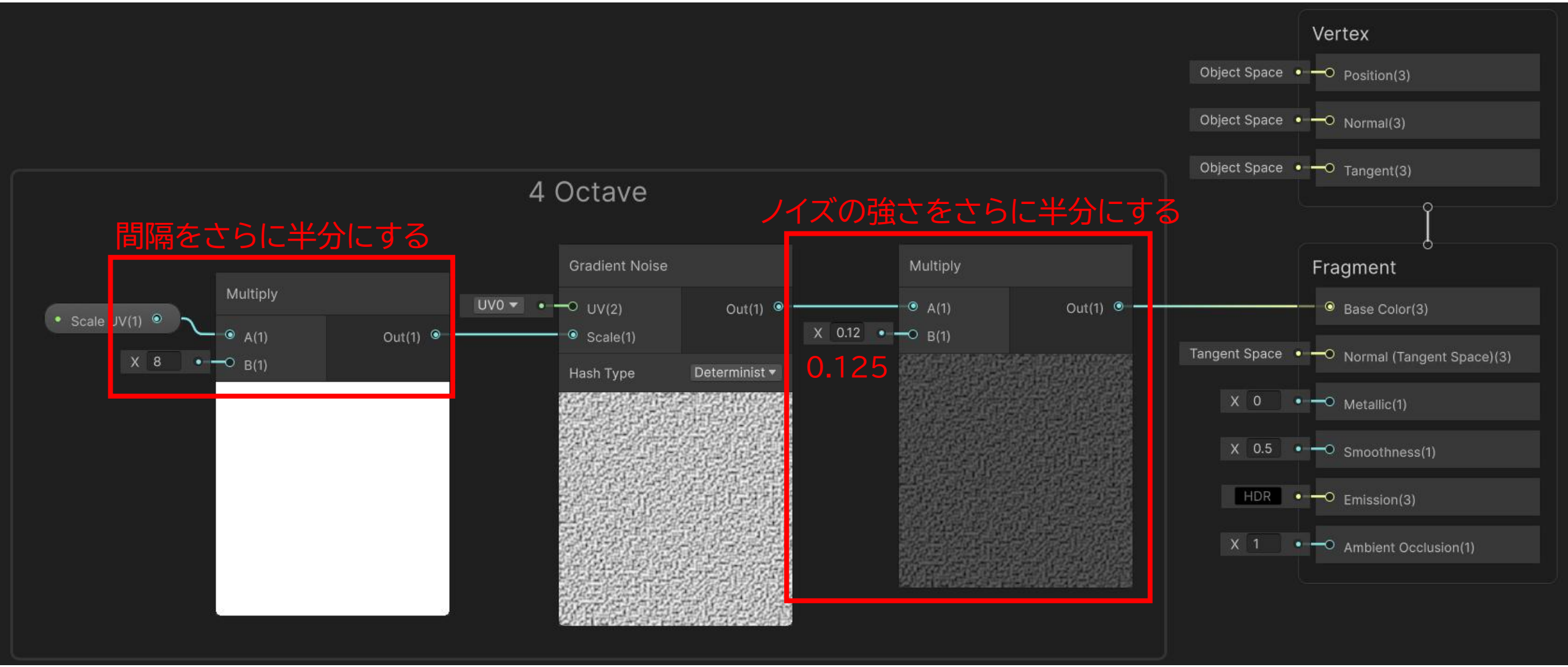



Vertex
Object Space
Position(3)
Object Space
Normal(3)
Object Space
Tangent(3)
3 Octave
間隔をさらに半分にする
ノイズの強さをさらに半分にする
Multiply
A(1)
B(1)
Out(1)
X 4
UV0
Gradient Noise
UV(2)
Scale(1)
Out(1)
Hash Type
Deterministic
X 0.25
Multiply
A(1)
B(1)
Out(1)
Fragment
Base Color(3)
Tangent Space
Normal (Tangent Space)(3)
X 0
Metallic(1)
X 0.5
Smoothness(1)
HDR
Emission(3)
X 1
Ambient Occlusion(1)

やってみよう: その4

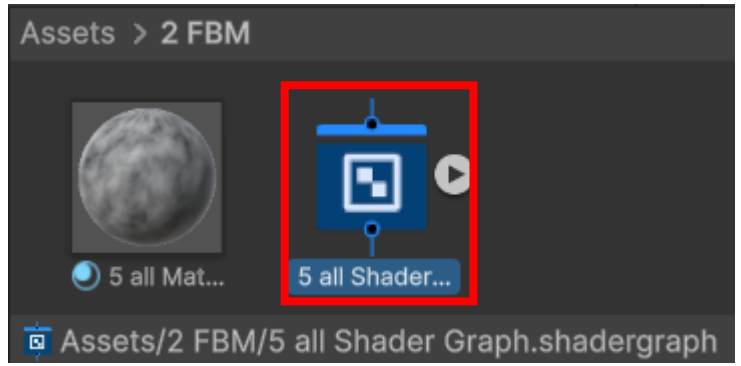
- Shader Graphを作成
  - 「2 FBM/4 4 octave Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を1つ追加(次は設定例)
    - Scale UV
      - Float型 (Mode: Slider)
      - 範囲: [0, 100]
      - 初期値: 10



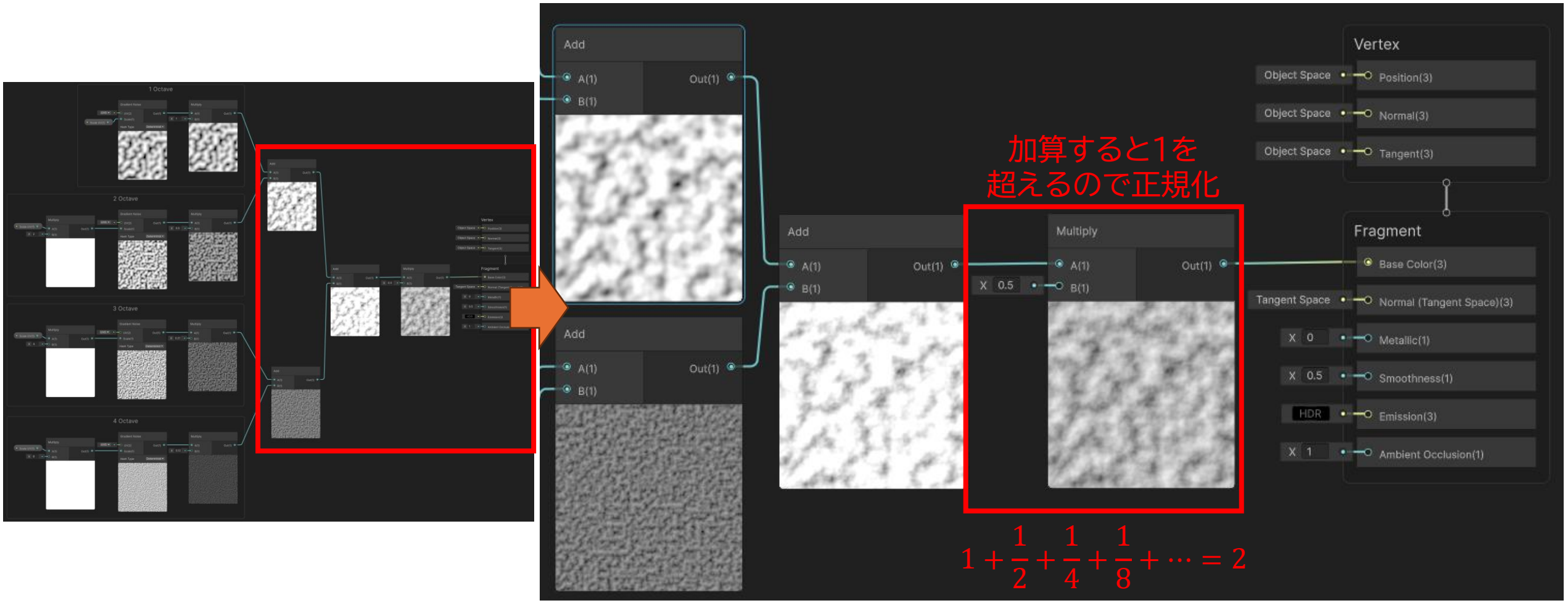

Vertex
Object Space
Position(3)
Object Space
Normal(3)
Object Space
Tangent(3)
4 Octave
間隔をさらに半分にする
ノイズの強さをさらに半分にする
Multiply
Scale
UV(1)
A(1)
X 8
B(1)
Out(1)
UV0
Gradient Noise
UV(2)
Scale(1)
Out(1)
Hash Type
Deterministic
X 0.12
0.125
Multiply
A(1)
B(1)
Out(1)
Fragment
Base Color(3)
Tangent Space
Normal (Tangent Space)(3)
X 0
Metallic(1)
X 0.5
Smoothness(1)
HDR
Emission(3)
X 1
Ambient Occlusion(1)

やってみよう: その5

- Shader Graphを作成
  - 「2 FBM/5 all Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を1つ追加(次は設定例)
    - Scale UV
      - Float型 (Mode: Slider)
      - 範囲: [0, 100]
      - 初期値: 10

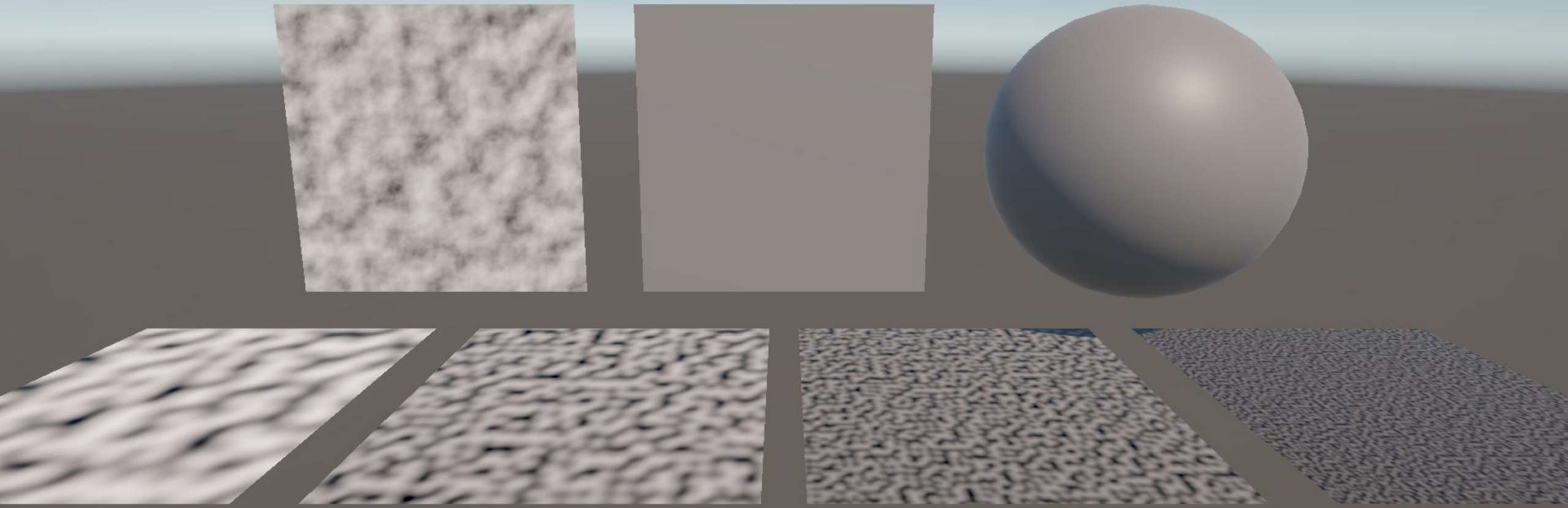


今までの結果を加算


1 Octave
2 Octave
3 Octave
4 Octave
Add
A(1)
B(1)
Out(1)
Add
A(1)
B(1)
Out(1)
Add
A(1)
B(1)
Out(1)
X 0.5
Multiply
A(1)
B(1)
Out(1)
加算すると1を
超えるので正規化
Vertex
Object Space
Position(3)
Object Space
Normal(3)
Object Space
Tangent(3)
Fragment
Base Color(3)
Tangent Space
Normal (Tangent Space)(3)
X 0
Metallic(1)
X 0.5
Smoothness(1)
HDR
Emission(3)
X 1
Ambient Occlusion(1)
$1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots=2$

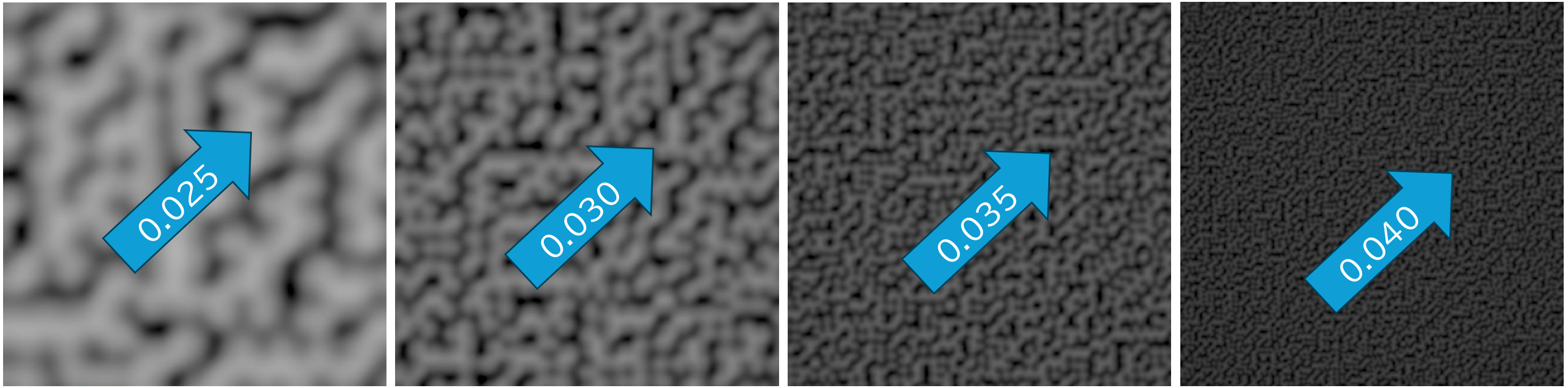
ここまでの状態

- さらにオクターブ数を増やせば細かな模様にできる

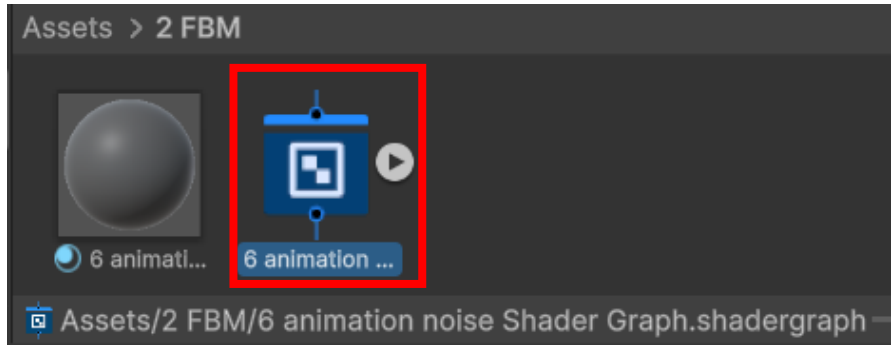


アニメーション

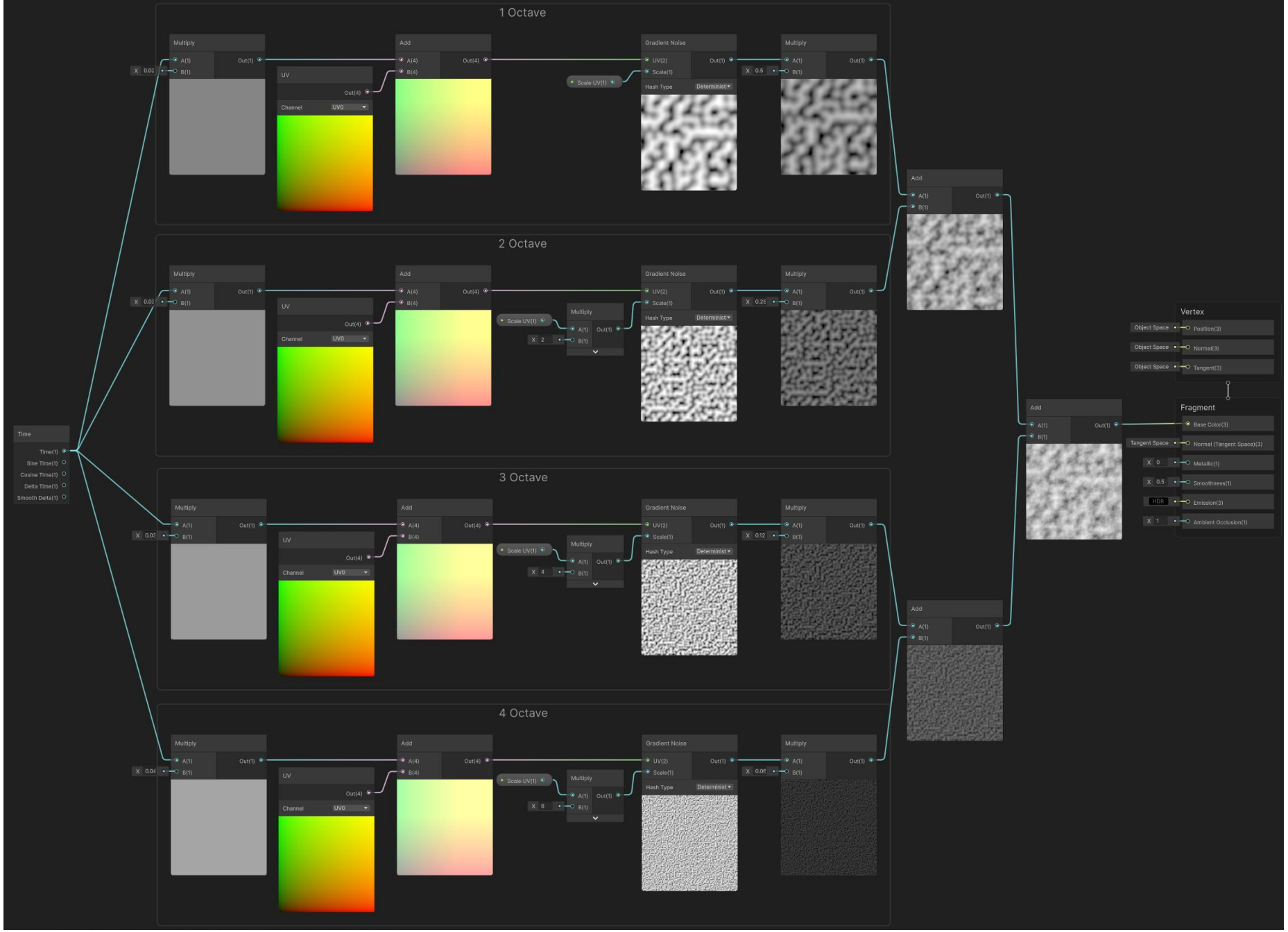
- スクロールさせてみる
- オクターブ毎に速度を変えると複雑な流れに見える



やってみよう: その6

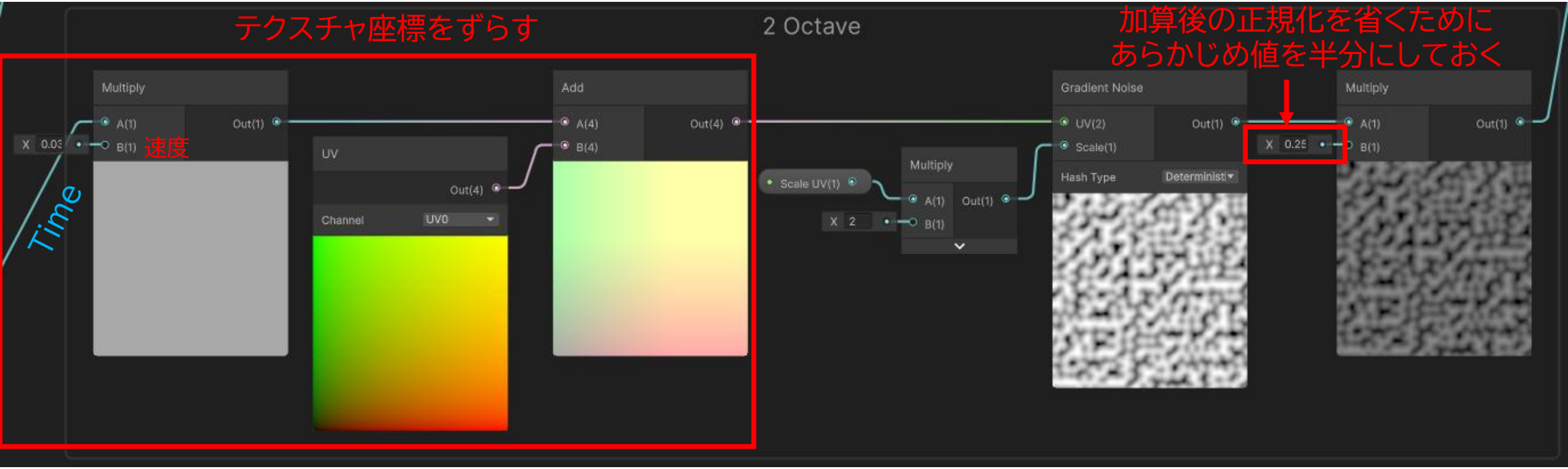


- Shader Graphを作成
  - 「2 FBM/6 animation noise Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を2つ追加(次は設定例)
    - Scale UV
      - Float型 (Mode: Slider)
      - 範囲: [0, 100]
      - 初期値: 10

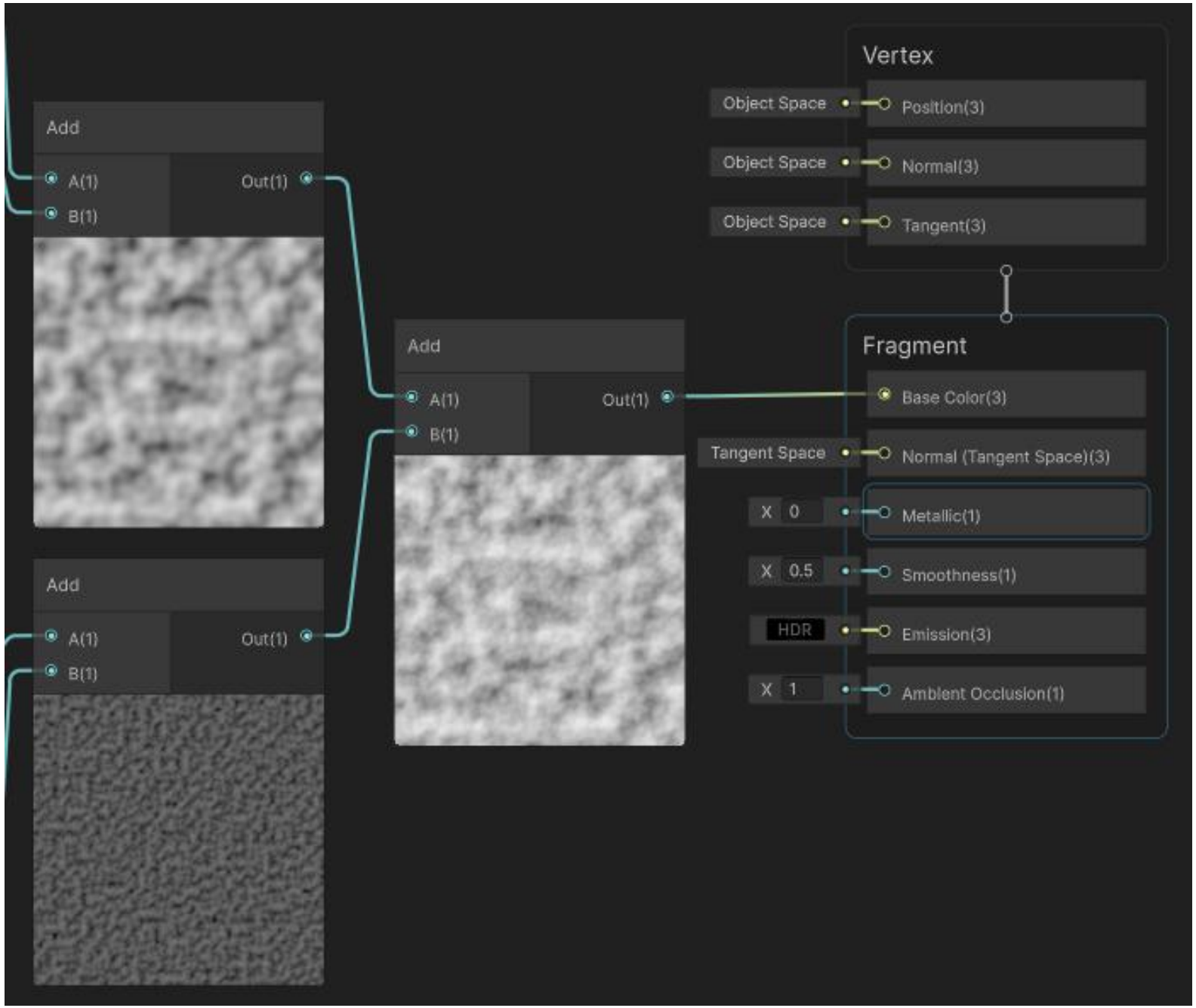

1 Octave
2 Octave
3 Octave
4 Octave
Time
Multiply
Add
UV
Gradient Noise
Scale UV(1)
Hash Type
Deterministic
Vertex
Object Space
Position(3)
Normal(3)
Tangent(3)
Fragment
Base Color(3)
Tangent Space
Normal (Tangent Space)(3)
Metallic(1)
Smoothness(1)
Emission(3)
Ambient Occlusion(1)

各オクターブ

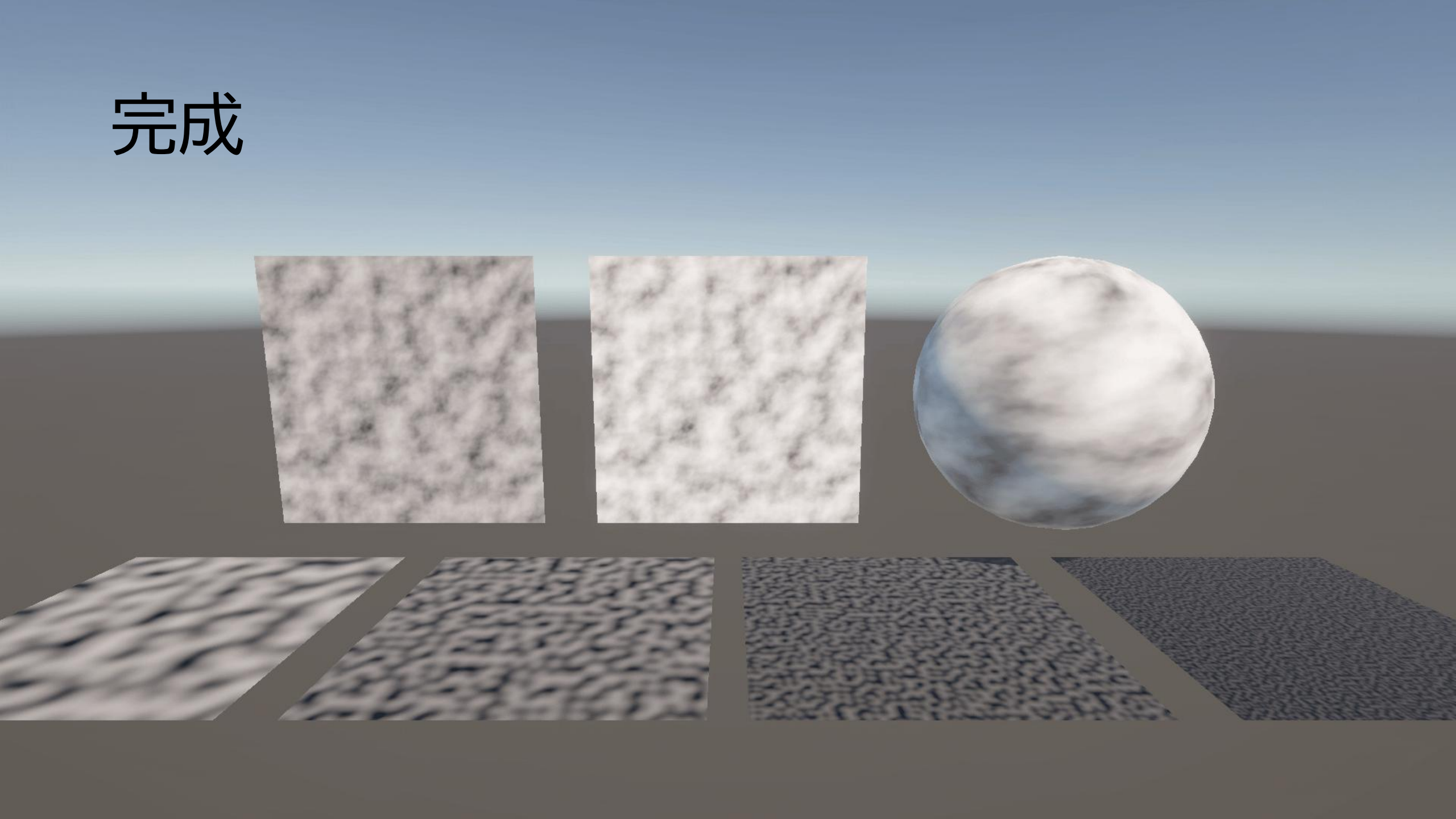
- UV座標を時間でずらす（加算）



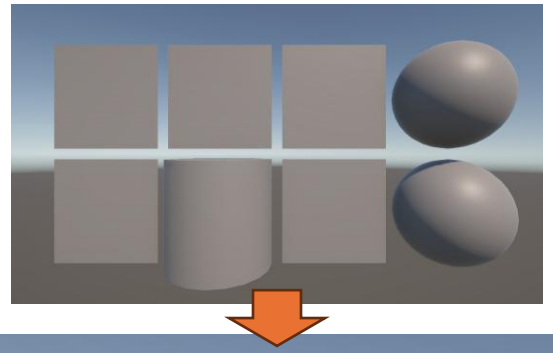
最後に加算


Add
A(1)
B(1)
Out(1)
Add
A(1)
B(1)
Out(1)
Add
A(1)
B(1)
Out(1)
Vertex
Object Space
Position(3)
Object Space
Normal(3)
Object Space
Tangent(3)
Fragment
Base Color(3)
Tangent Space
Normal (Tangent Space)(3)
X 0
Metallic(1)
X 0.5
Smoothness(1)
HDR
Emission(3)
X 1
Ambient Occlusion(1)

完成

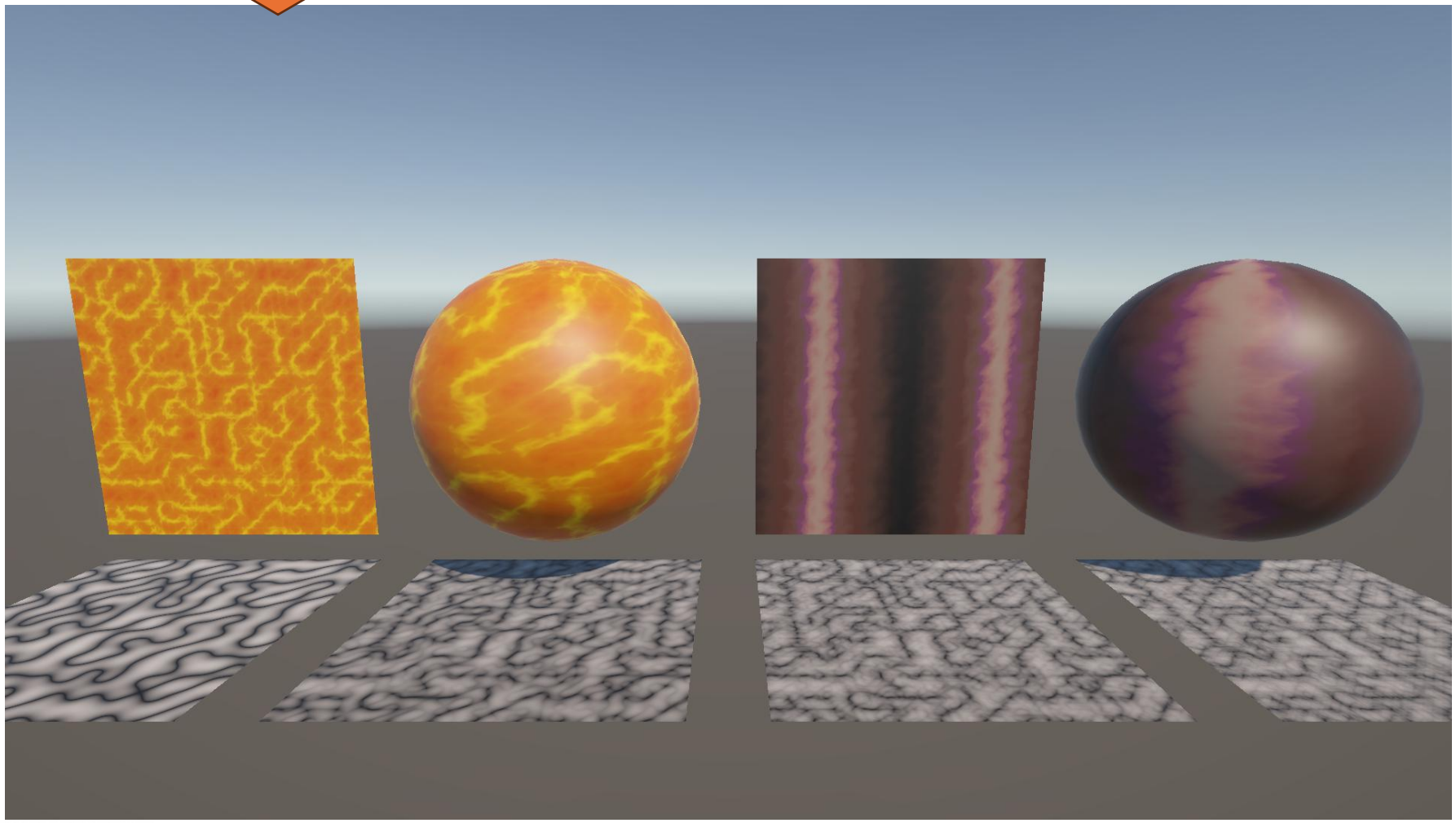


本日の内容



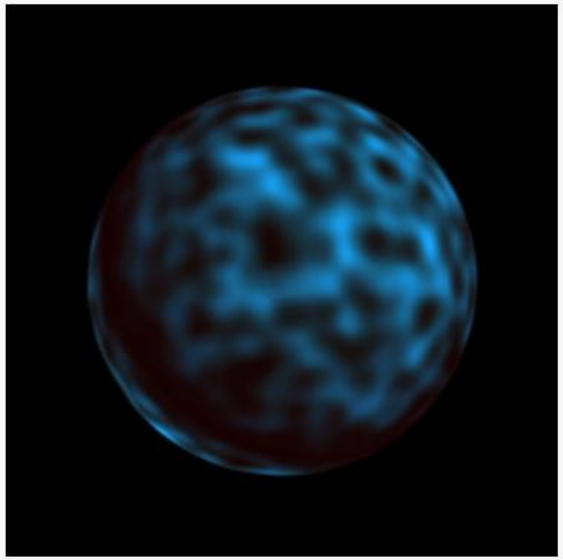
シーン：3 Pattern Scene

- ノイズの概要
- 基本的なノイズ
- FBM
- 加工したFBM
- ボロノイ図
- 炎
- ディゾルブ

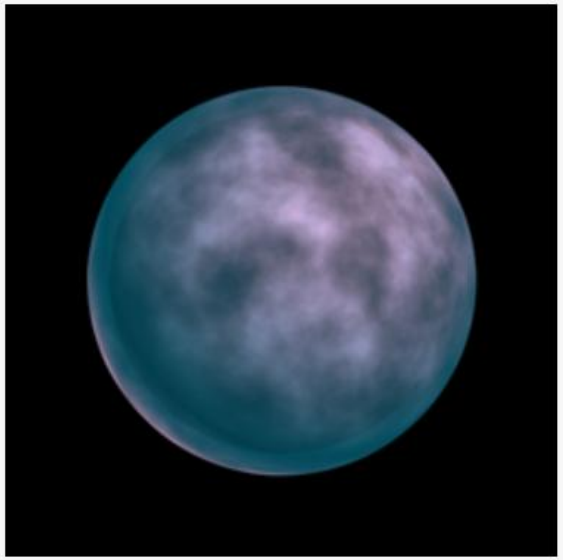


Perlin ノイズの応用

- 足し合わせる際の各段階や合成した結果にノイズを加工すると多彩な表情をもたせられる

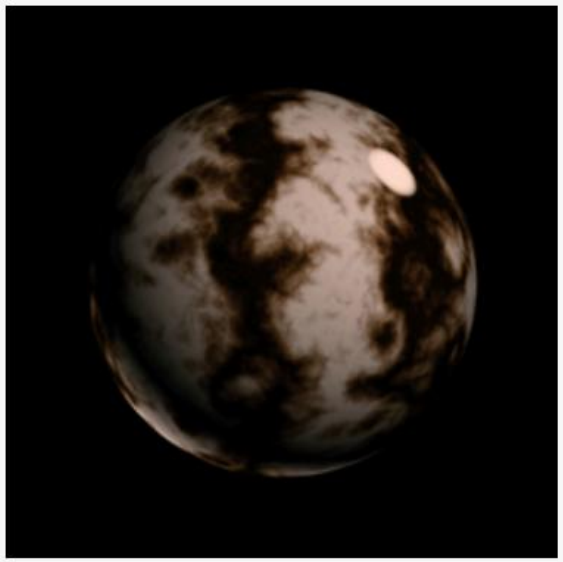


ベースのパーリンノイズを球体に適用した例



パーリンノイズの合成関数の一例。
$\Sigma(1/2^i \cdot noise(2^i x))$

1つ前のものに絶対値を適用した合成ノイズ関数。$\Sigma ABS(1/2^i \cdot noise(2^i x))$



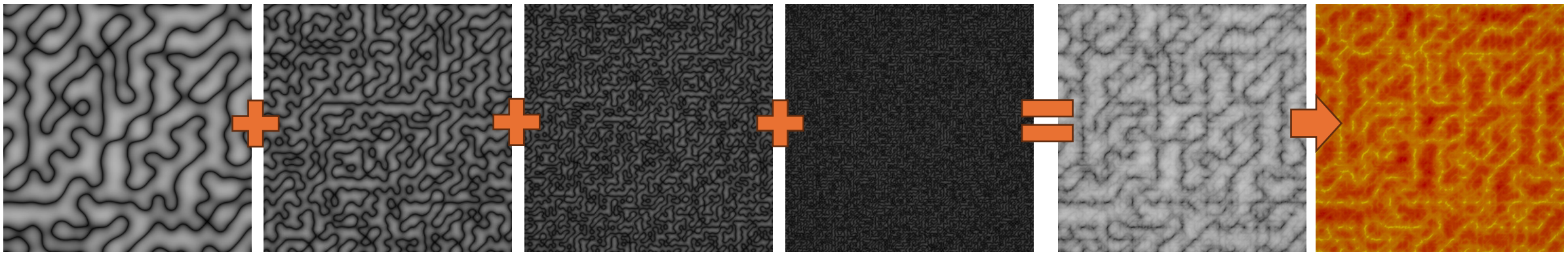
三角関数を用いた合成ノイズ関数。
$SIN(x+\Sigma ABS(1/2^i \cdot noise(2^i x)))$

惑星表面のような模様に挑戦

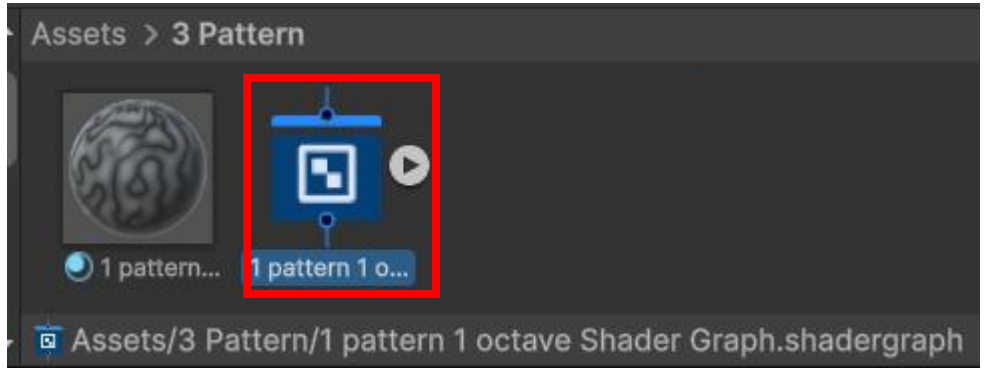
- ノイズの絶対値の和

$$\sum_i \frac{\left|noise(2^i x)\right|}{2^i}$$

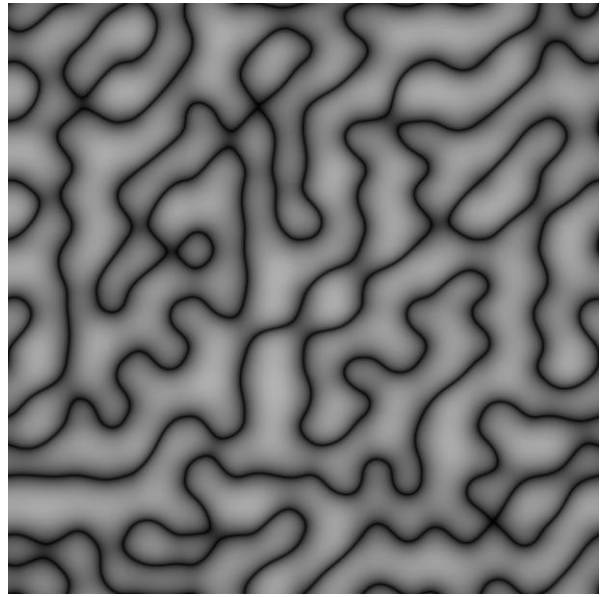
色の変換



やってみよう: その1

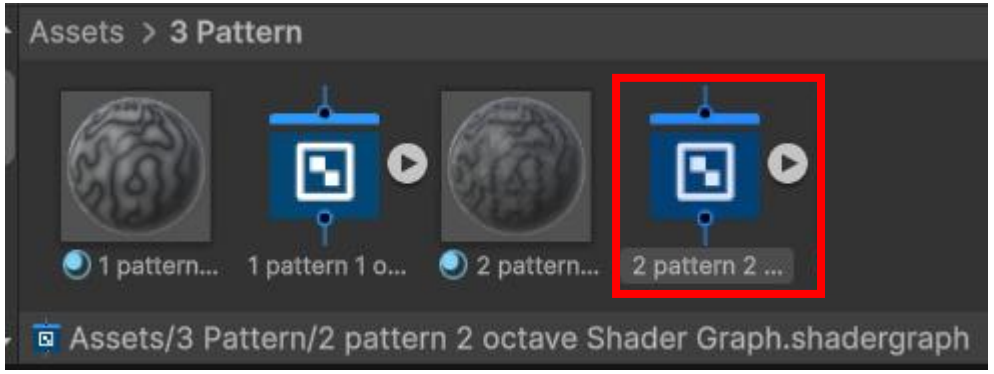


- Shader Graphを作成
  - 「3 Pattern/1 pattern 1 octave Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を1つ追加(次は設定例)
    - Scale UV
      - Float型 (Mode: Slider)
      - 範囲: [0, 100]
      - 初期値: 10

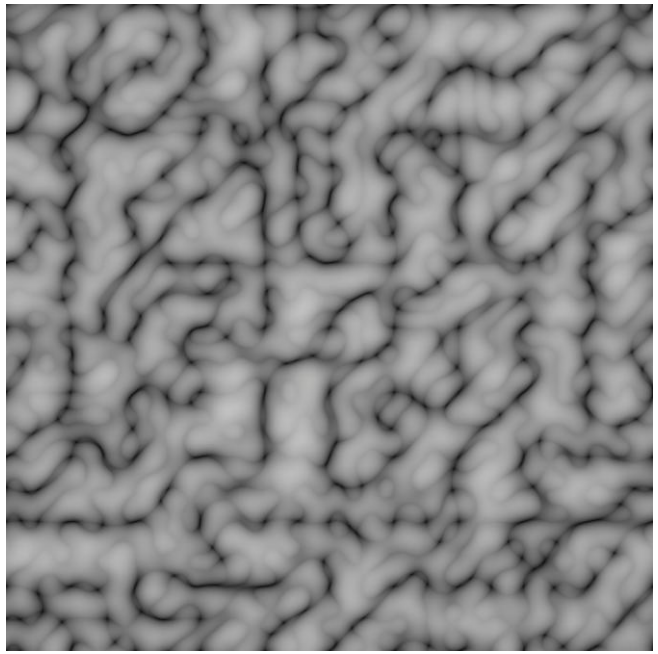


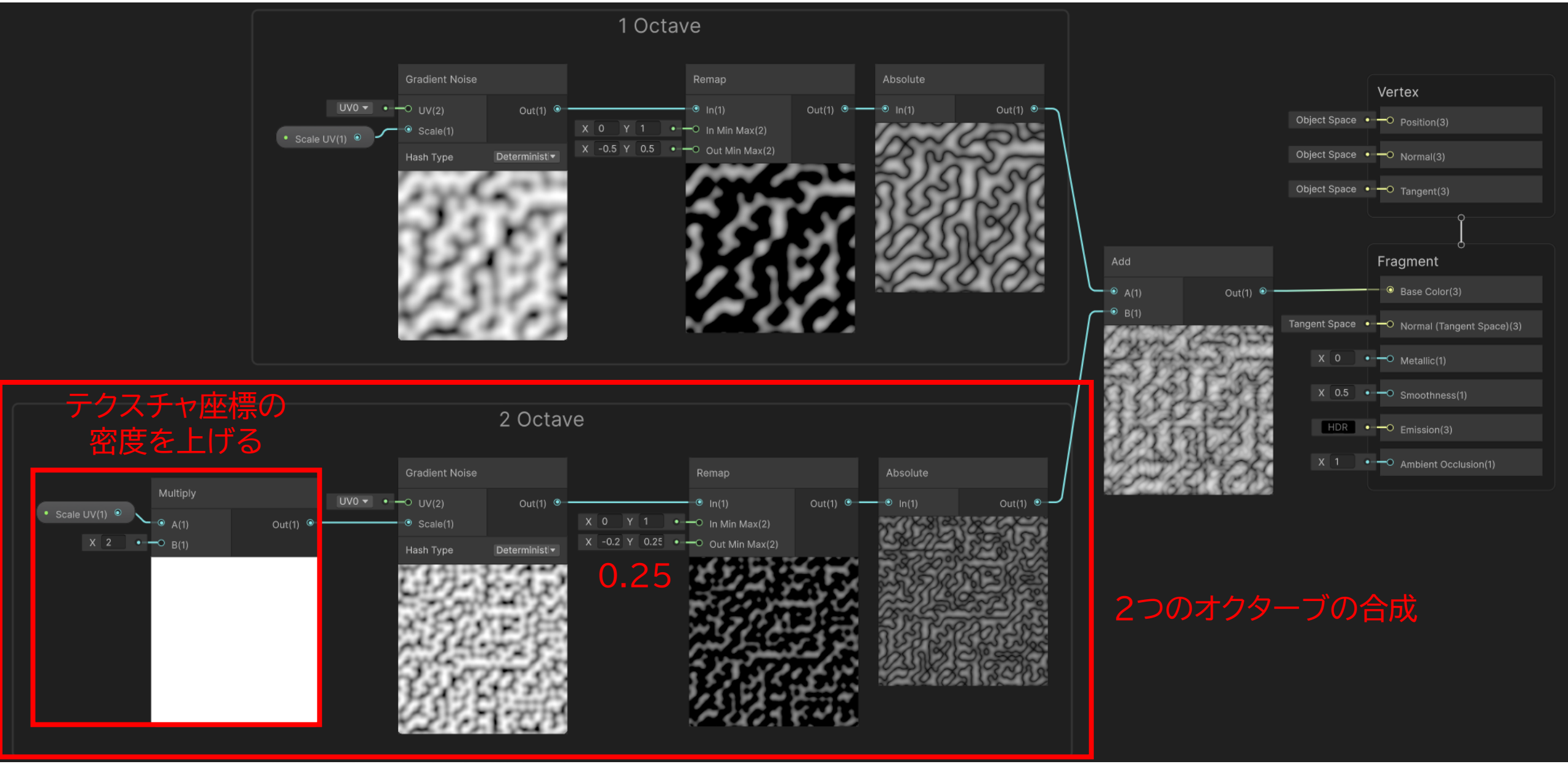
通常のノイズは正値なので
0中心で[-0.5,+0.5]に収める

やってみよう: その2

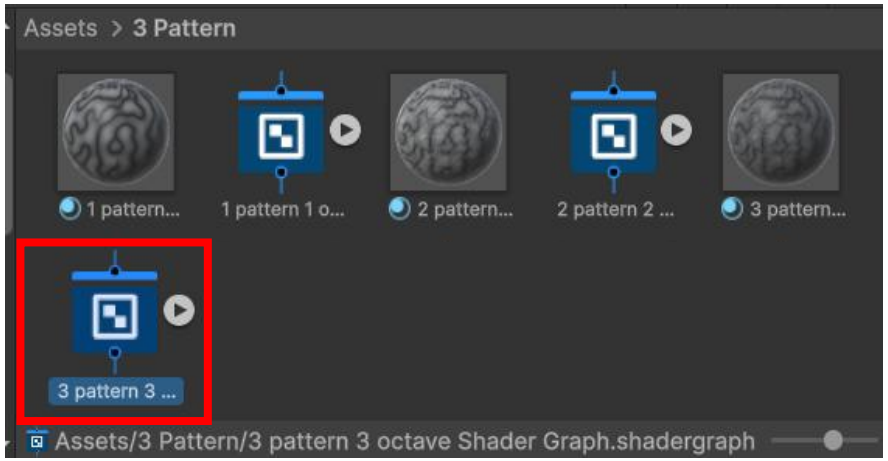


- Shader Graphを作成
  - 「3 Pattern/2 pattern 2 octave Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を1つ追加(次は設定例)
    - Scale UV
      - Float型 (Mode: Slider)
      - 範囲: [0, 100]
      - 初期値: 10

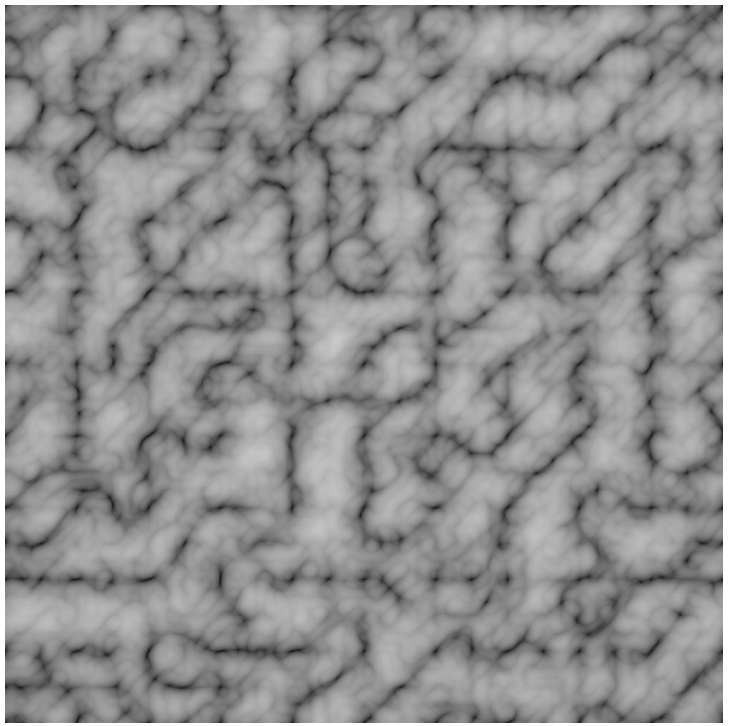


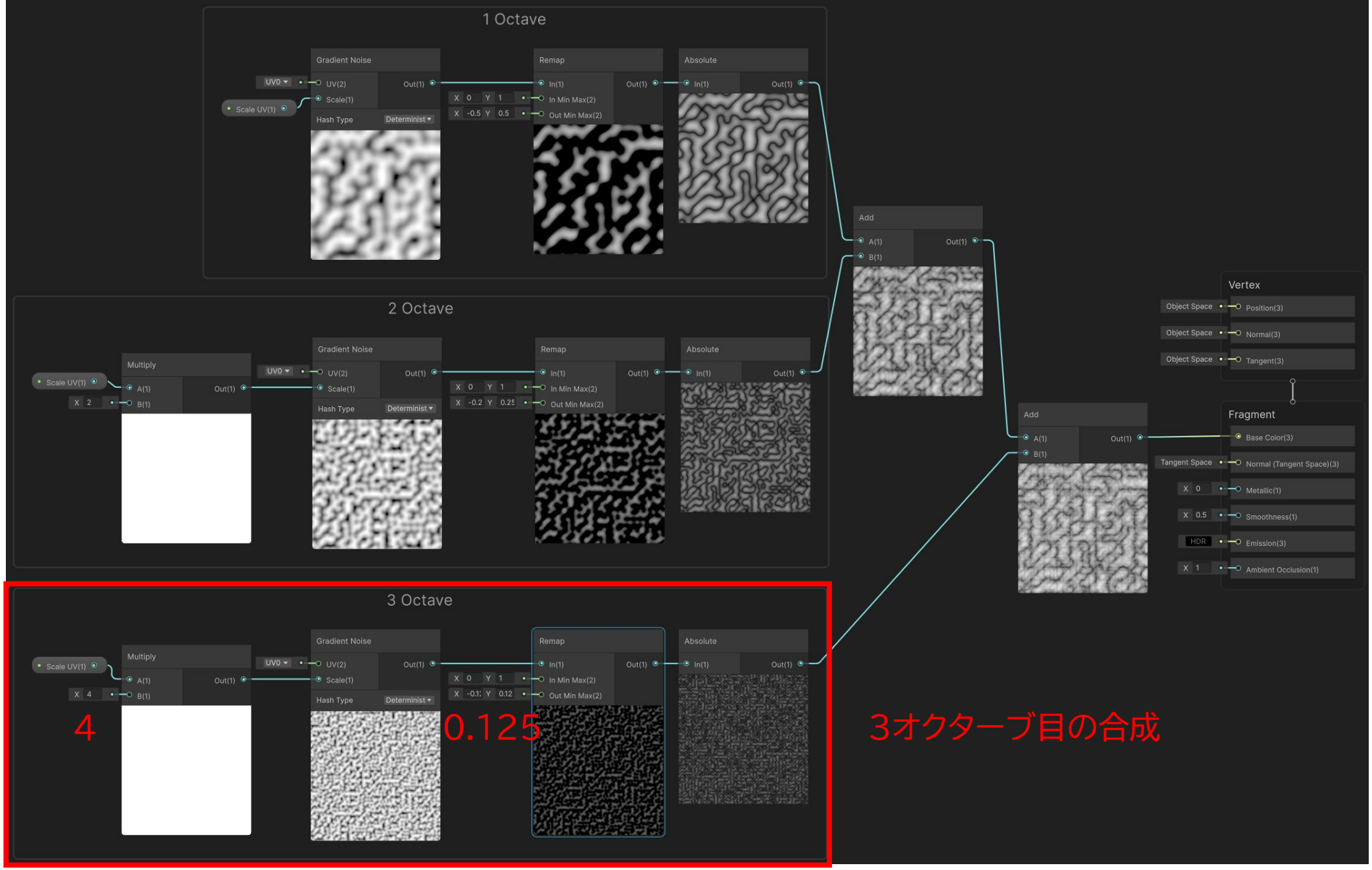

1 Octave
Gradient Noise
UV0
UV(2)
Out(1)
Scale UV(1)
Scale(1)
Hash Type
Deterministi
Remap
In(1)
Out(1)
X 0 Y 1
In Min Max(2)
X -0.5 Y 0.5
Out Min Max(2)
Absolute
In(1)
Out(1)
Vertex
Object Space
Position(3)
Object Space
Normal(3)
Object Space
Tangent(3)
Add
A(1)
B(1)
Out(1)
Fragment
Base Color(3)
Tangent Space
Normal (Tangent Space)(3)
X 0
Metallic(1)
X 0.5
Smoothness(1)
HDR
Emission(3)
X 1
Ambient Occlusion(1)
テクスチャ座標の
密度を上げる
2 Octave
Multiply
Scale UV(1)
A(1)
X 2
B(1)
Out(1)
UV0
Gradient Noise
UV(2)
Out(1)
Scale(1)
Hash Type
Deterministi
X 0 Y 1
X -0.2 Y 0.25
0.25
Remap
In(1)
Out(1)
In Min Max(2)
Out Min Max(2)
Absolute
In(1)
Out(1)
2つのオクターブの合成

やってみよう: その3

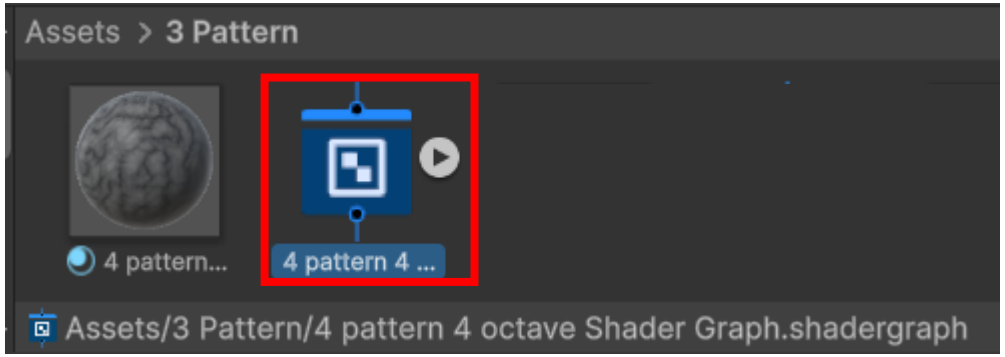


- Shader Graphを作成
  - 「3 Pattern/3 pattern 3 octave Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を1つ追加(次は設定例)
    - Scale UV
      - Float型 (Mode: Slider)
      - 範囲: [0, 100]
      - 初期値: 10

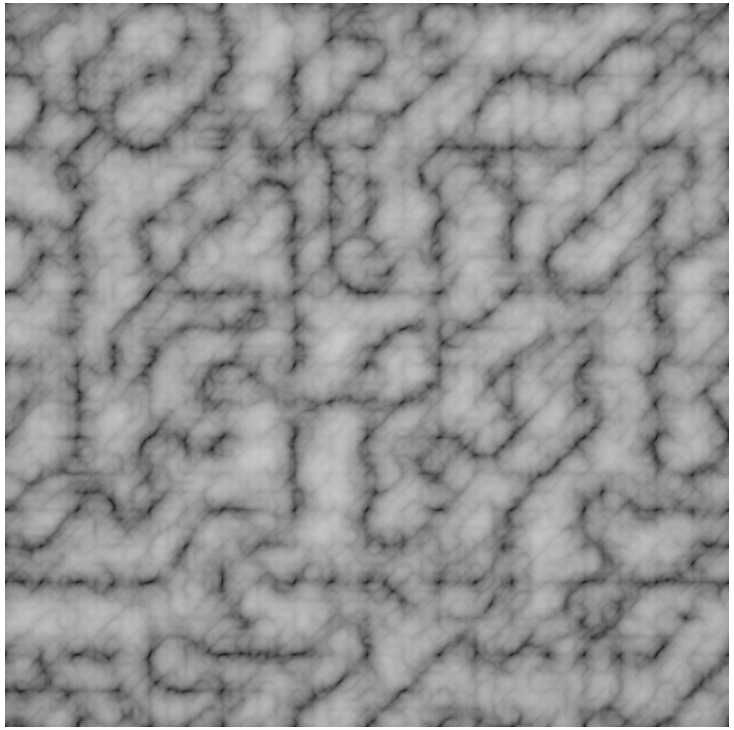


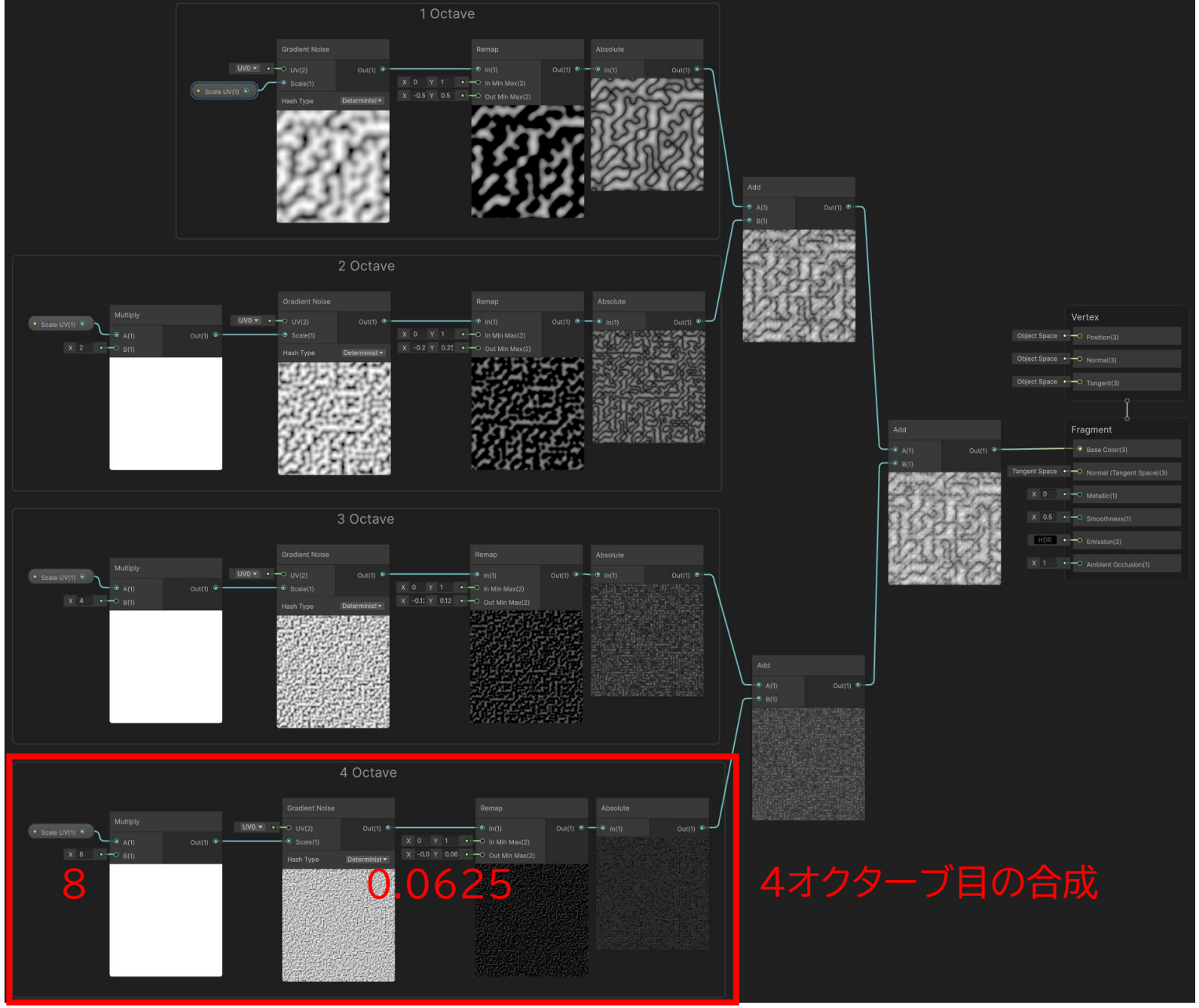

1 Octave
Gradient Noise
UV0
UV(2)
Scale UV(1)
Scale(1)
Out(1)
Hash Type
Deterministic
Remap
In(1)
In Min Max(2)
Out Min Max(2)
X 0 Y 1
X -0.5 Y 0.5
Absolute
2 Octave
Multiply
A(1)
B(1)
X 2
X -0.2 Y 0.25
3 Octave
X 4
X 0 Y 0.12
4
0.125
3オクターブ目の合成
Add
Vertex
Object Space
Position(3)
Normal(3)
Tangent(3)
Fragment
Base Color(3)
Tangent Space
Normal (Tangent Space)(3)
Metallic(1)
Smoothness(1)
HDR
Emission(3)
Ambient Occlusion(1)

やってみよう: その4



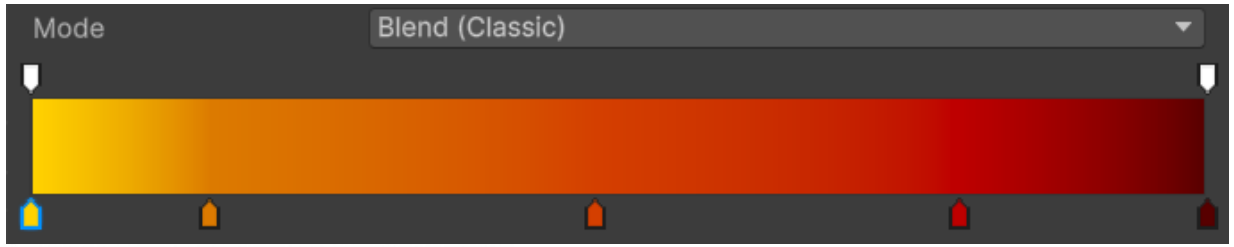
- Shader Graphを作成
  - 「3 Pattern/4 pattern 4 octave Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を1つ追加(次は設定例)
    - Scale UV
      - Float型 (Mode: Slider)
      - 範囲: [0, 100]
      - 初期値: 10

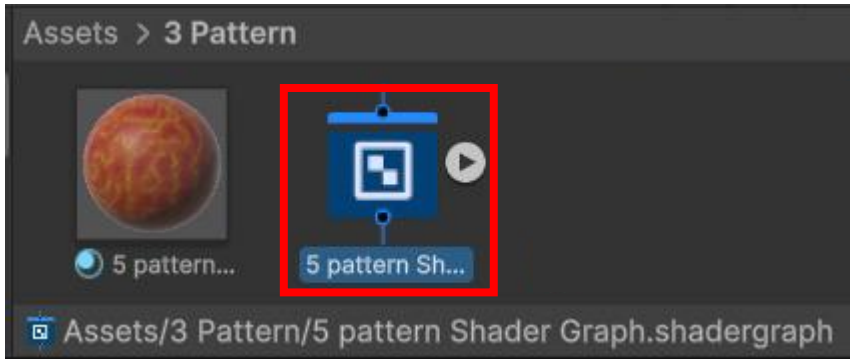


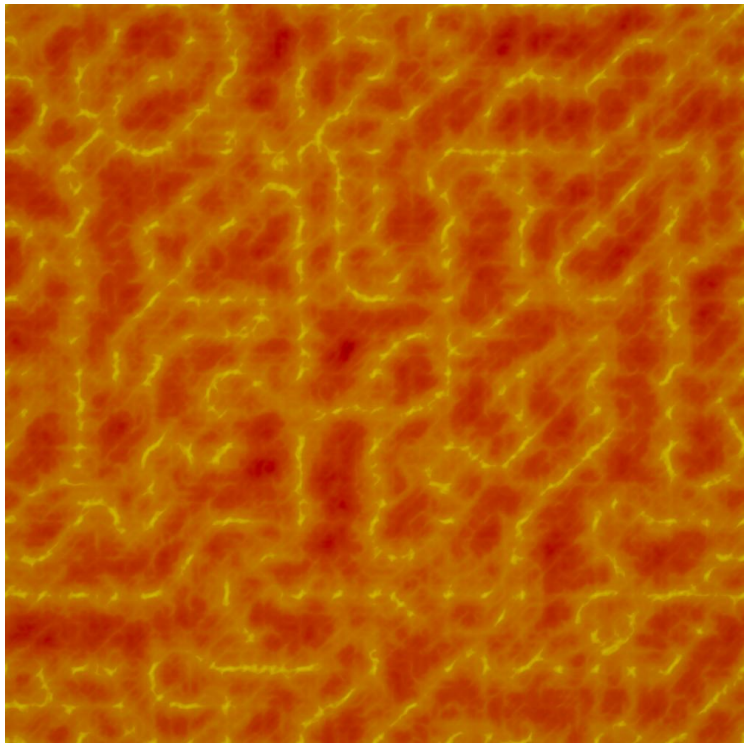

1 Octave
Gradient Noise
Remap
Absolute
2 Octave
Multiply
3 Octave
4 Octave
8
0.0625
Add
Vertex
Fragment
4オクターブ目の合成

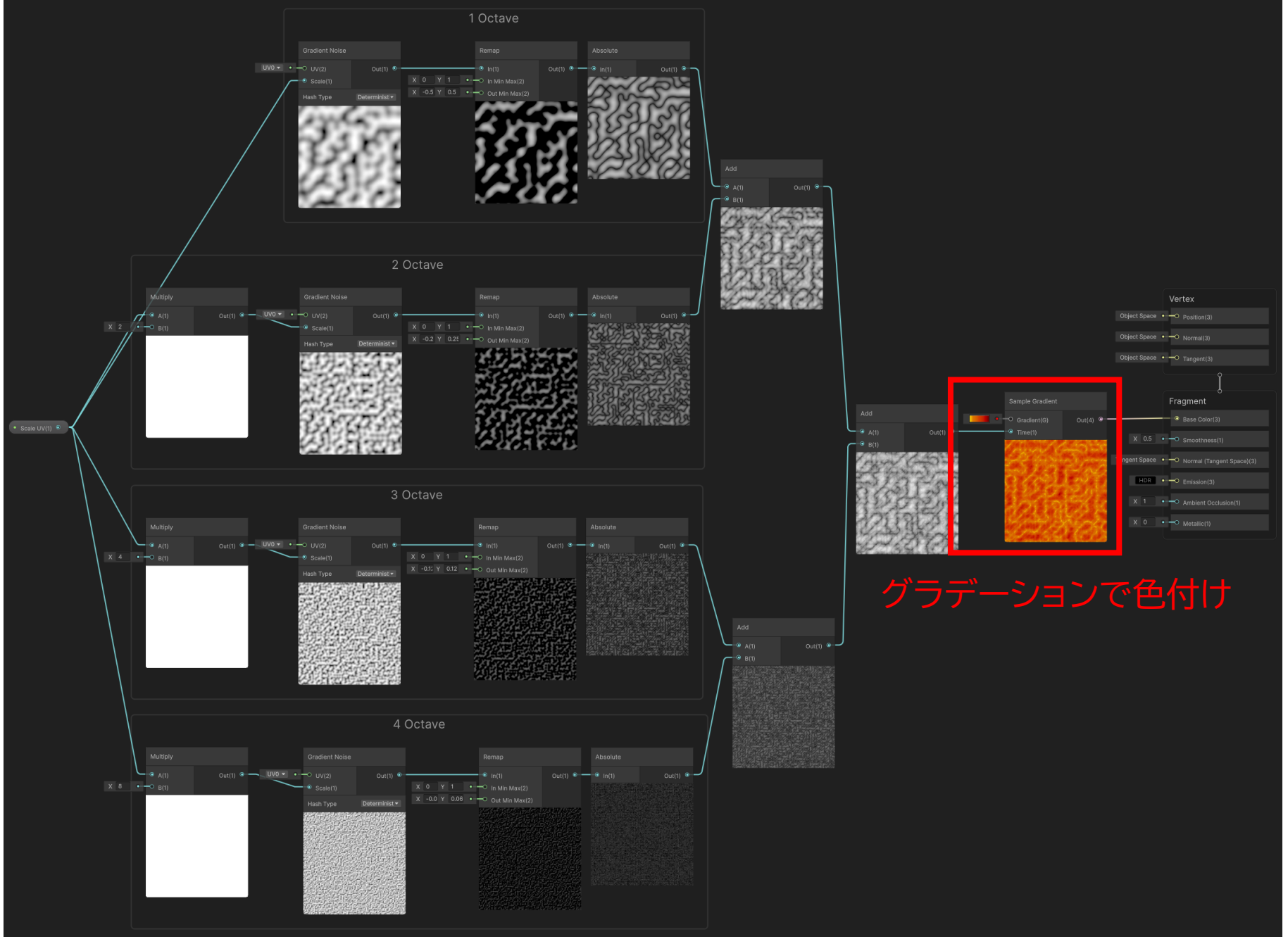
やってみよう: その5

- Shader Graphを作成
  - 「3 Pattern/5 pattern Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を1つ追加(次は設定例)
    - Scale UV
      - Float型 (Mode: Slider)
      - 範囲: [0, 100]
      - 初期値: 10
  - グラデーションを導入(いい感じに調整)



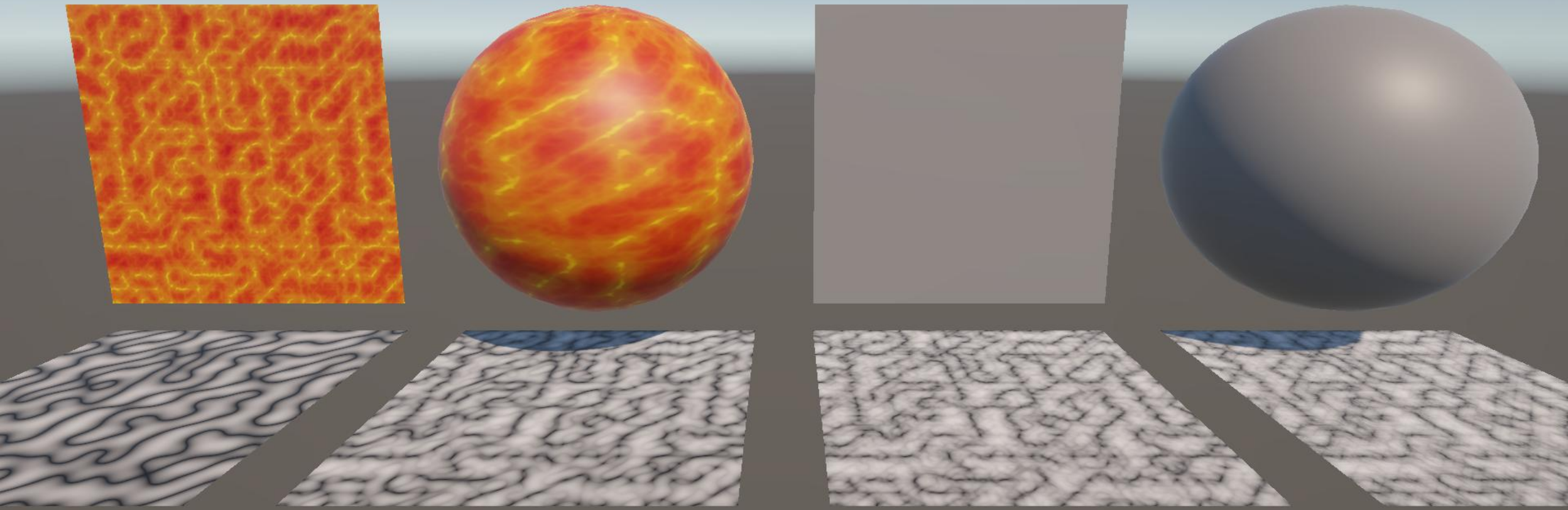




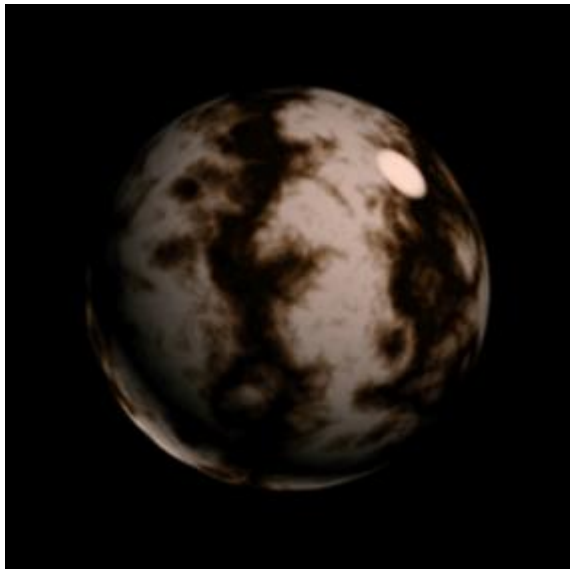

1 Octave
Gradient Noise
Remap
Absolute
2 Octave
Multiply
3 Octave
4 Octave
Add
Sample Gradient
Vertex
Fragment
グラデーションで色付け

ここまでの状態

- 「Scale UV」を変えて、模様がどうなるか見てみよう



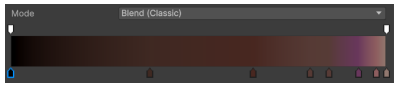
大理石のような模様に挑戦



- ノイズの絶対値の和

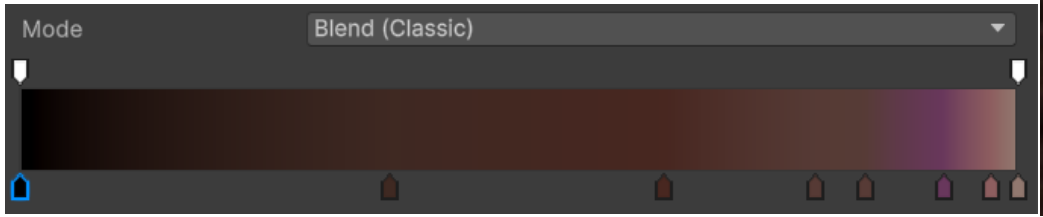
$$sin\left(x + \sum_i \frac{|noise(2^i x)|}{2^i}\right)$$

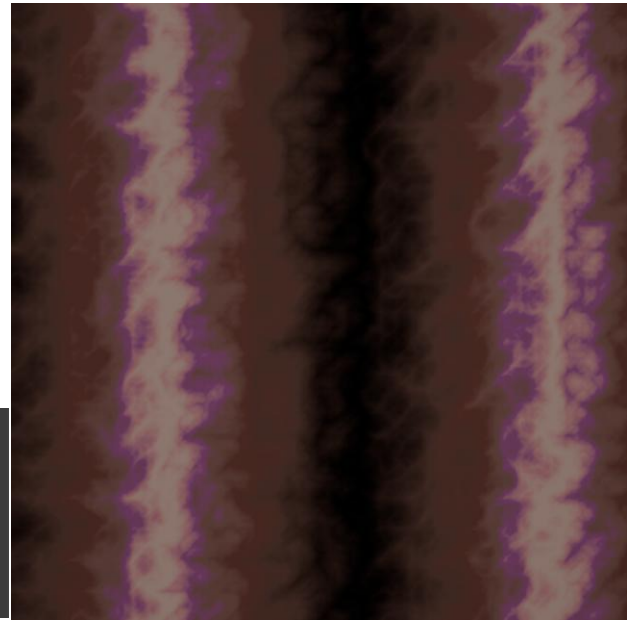
色の変換

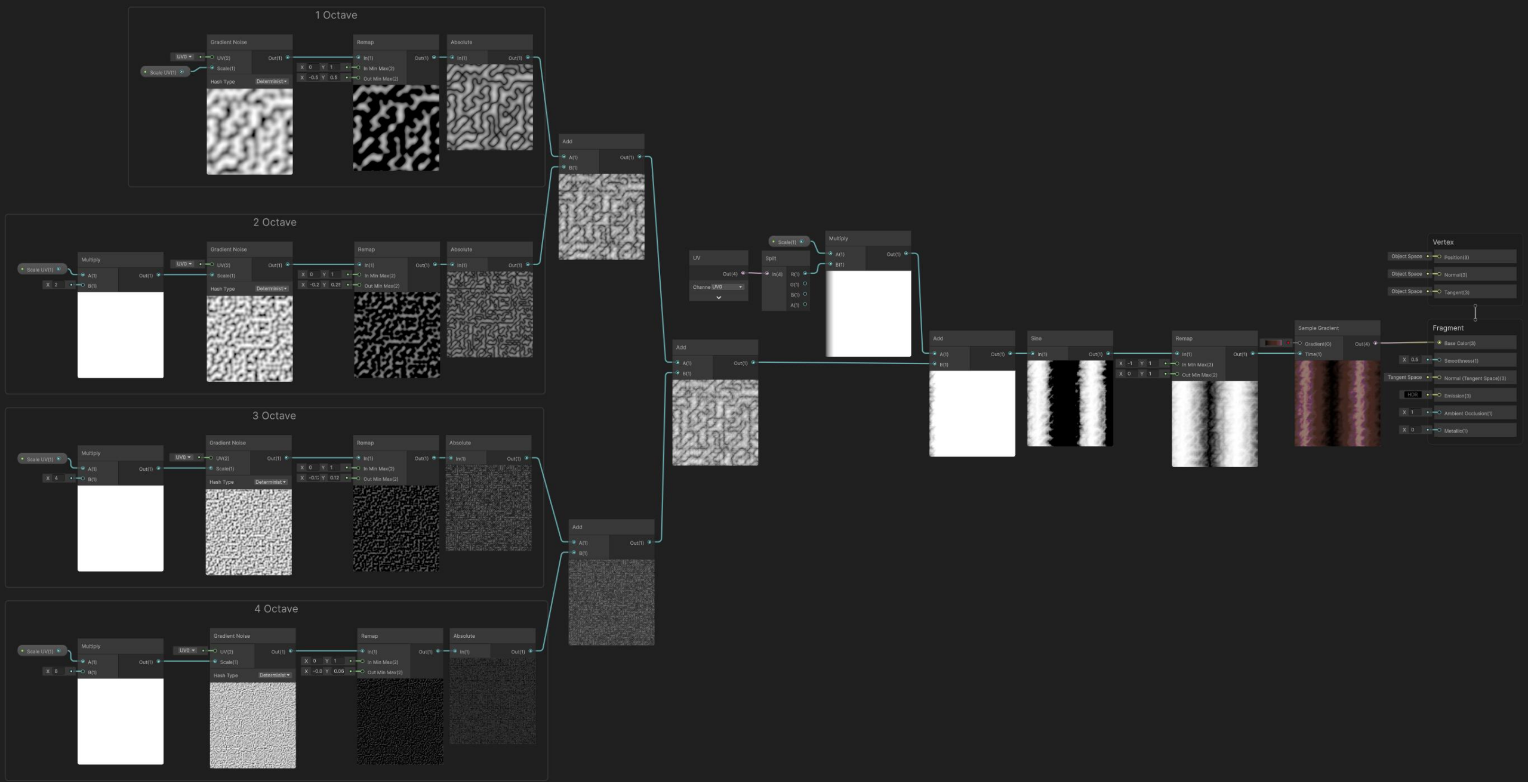


やってみよう: その6

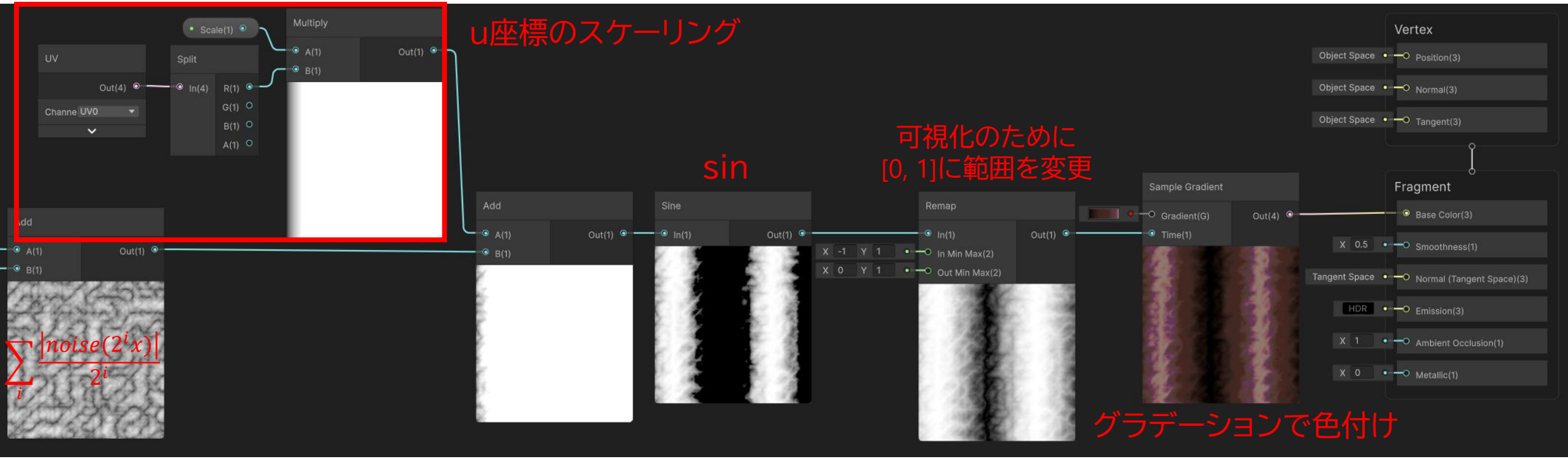
- Shader Graphを作成
  - 「3 Pattern/6 marble Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を2つ追加(次は設定例)
    - Scale UV
      - Float型 (Mode: Slider)
      - 範囲: [0, 100]
      - 初期値: 10
    - Scale: sin内のxのスケーリング
      - Float型 (Mode: Slider)
      - 範囲: [0, 100]
      - 初期値: 10
  - グラデーションを導入
    - いい感じに調整



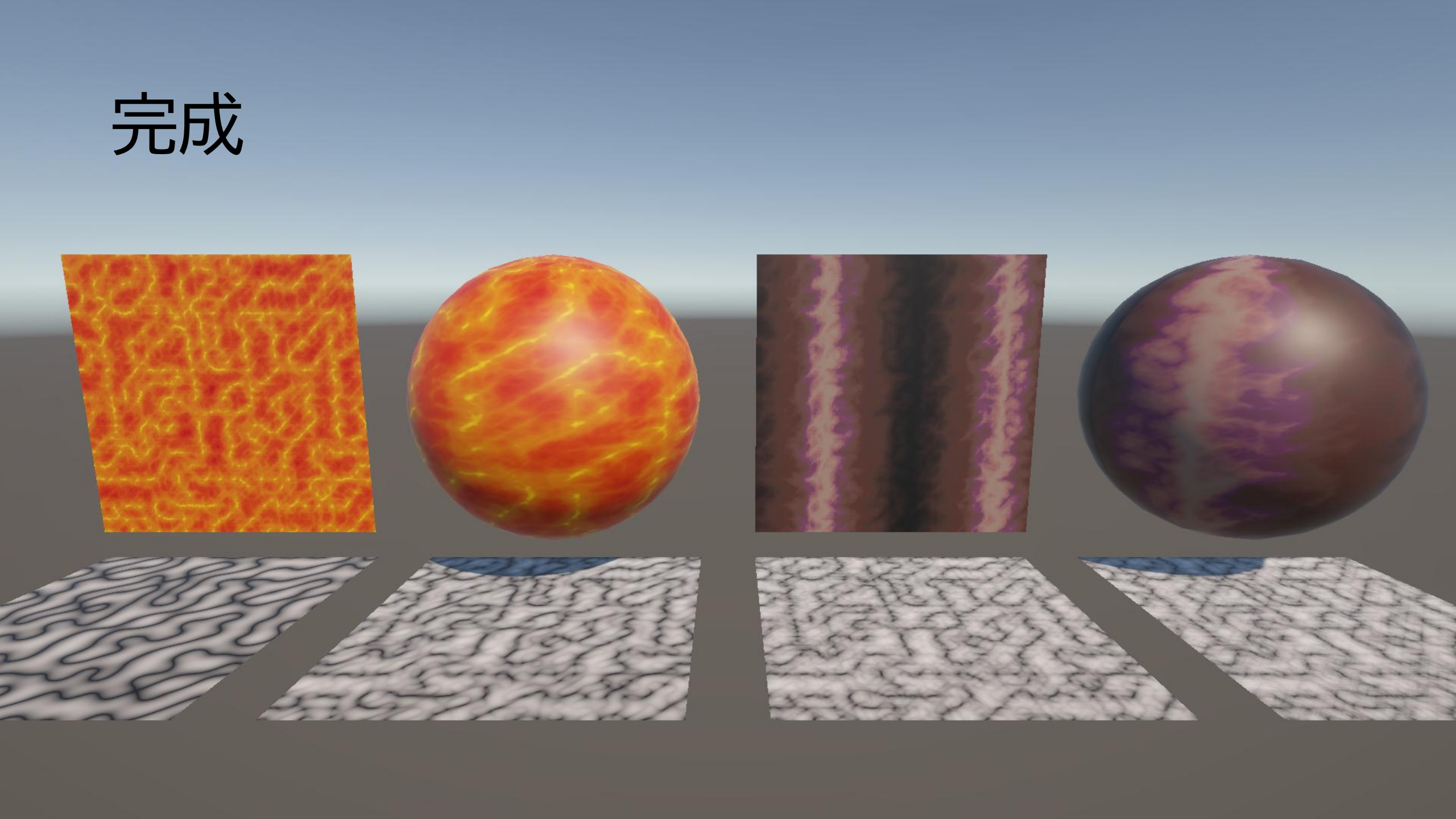



1 Octave
Gradient Noise
Remap
Absolute
2 Octave
Multiply
3 Octave
4 Octave
Add
UV
Split
Sine
Sample Gradient
Vertex
Fragment

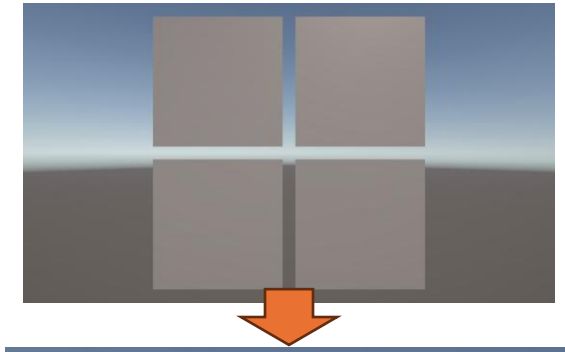
fBm作成後


u座標のスケーリング
sin
可視化のために
[0, 1]に範囲を変更
グラデーションで色付け
Scale(1)
Multiply
A(1)
B(1)
Out(1)
UV
Out(4)
Channe UV0
Split
In(4)
R(1)
G(1)
B(1)
A(1)
Add
Sine
In(1)
Remap
In Min Max(2)
Out Min Max(2)
X -1 Y 1
X 0 Y 1
Sample Gradient
Gradient(G)
Time(1)
Vertex
Object Space
Position(3)
Normal(3)
Tangent(3)
Fragment
Base Color(3)
X 0.5
Smoothness(1)
Tangent Space
Normal (Tangent Space)(3)
HDR
Emission(3)
X 1
Ambient Occlusion(1)
X 0
Metallic(1)
$\sum_i \frac{|noise(2^i x)|}{2^i}$

完成

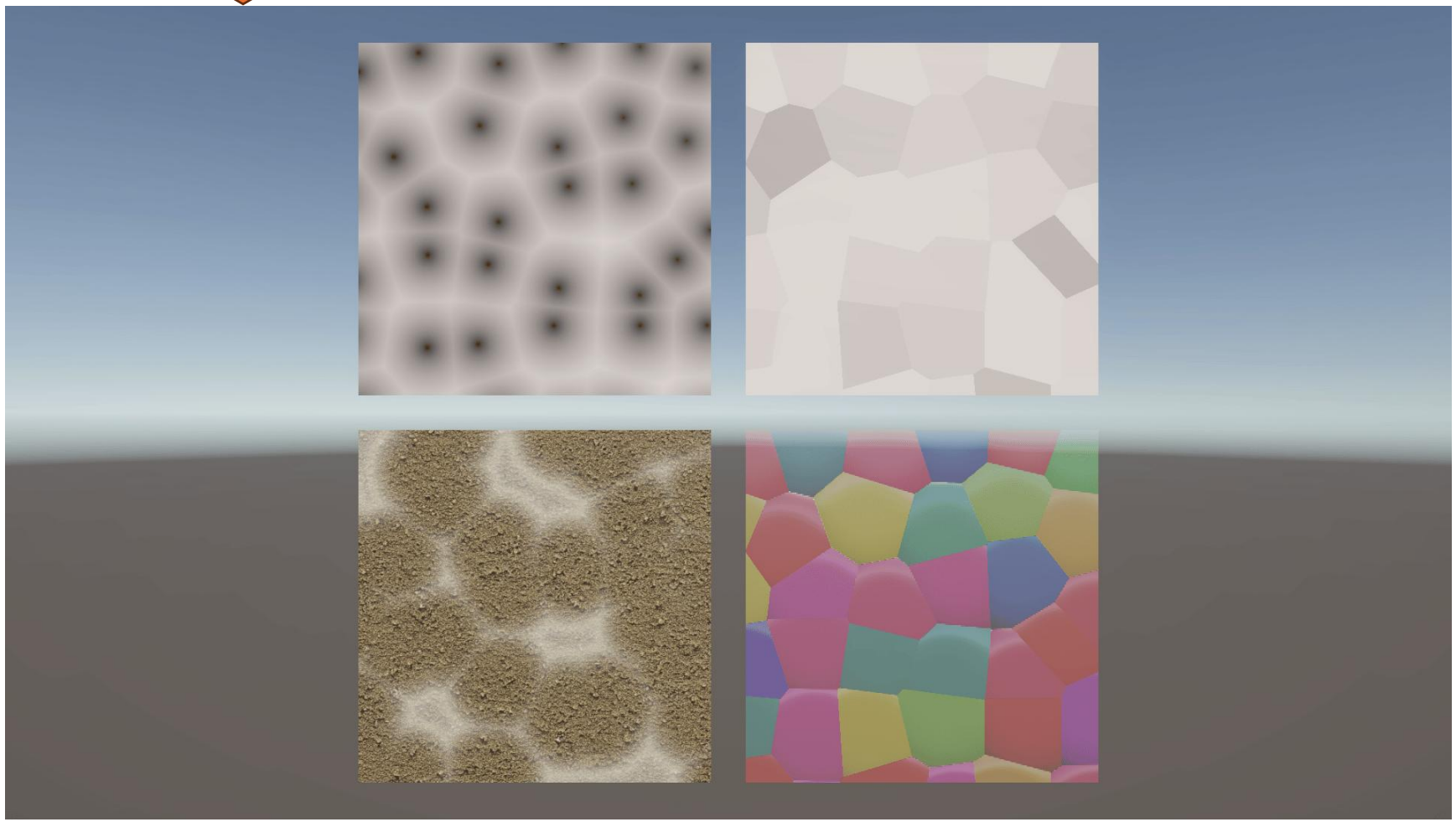


本日の内容



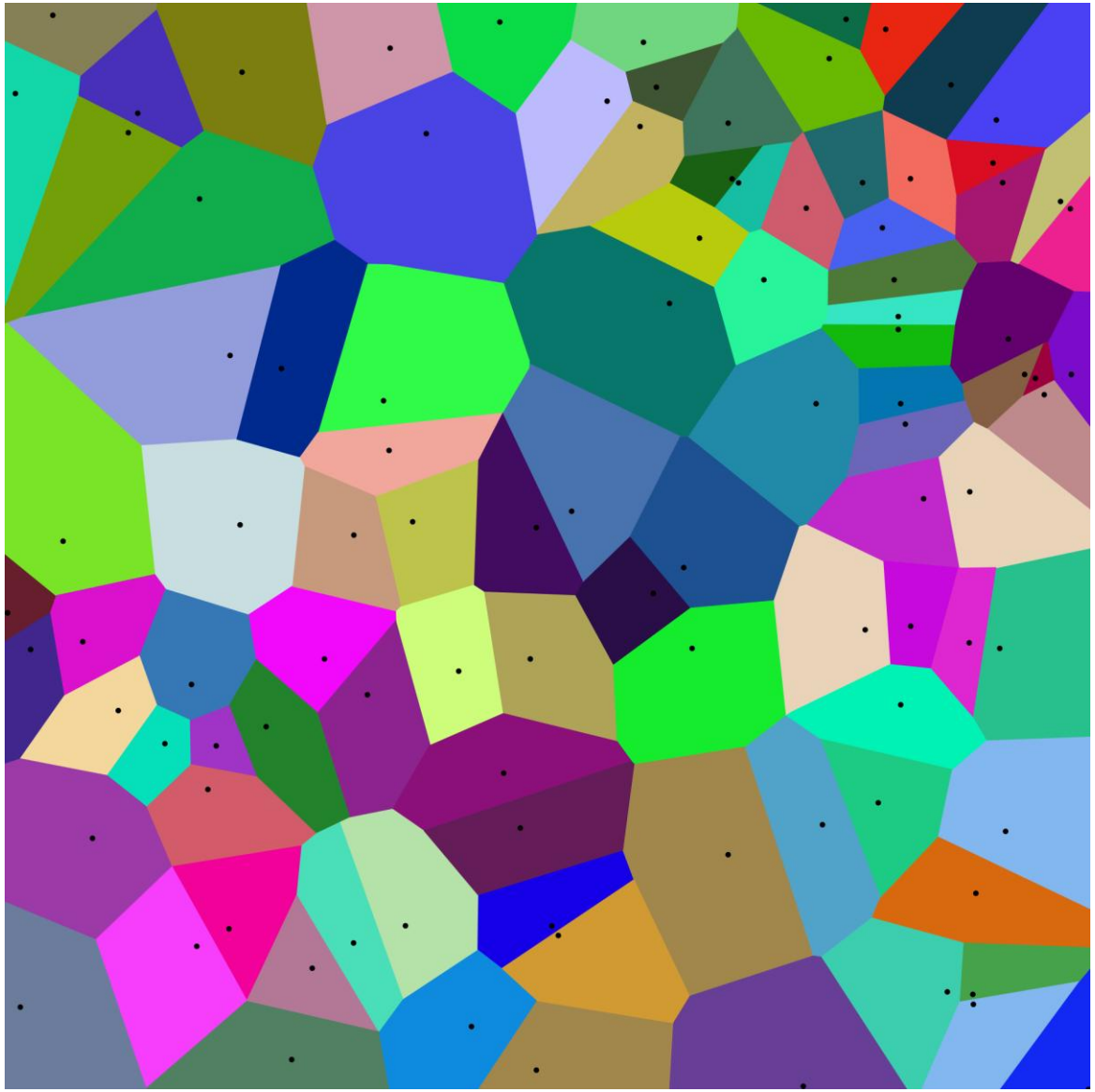
シーン：4 Voronoi Scene

- ノイズの概要
- 基本的なノイズ
- FBM
- 加工したFBM
- ボロノイ図
  - 概要
  - 母点からの距離
  - 集光模様
  - セルインデックス
  - ステンドグラス風表現
- 炎
- ディゾルブ



ボロノイ図

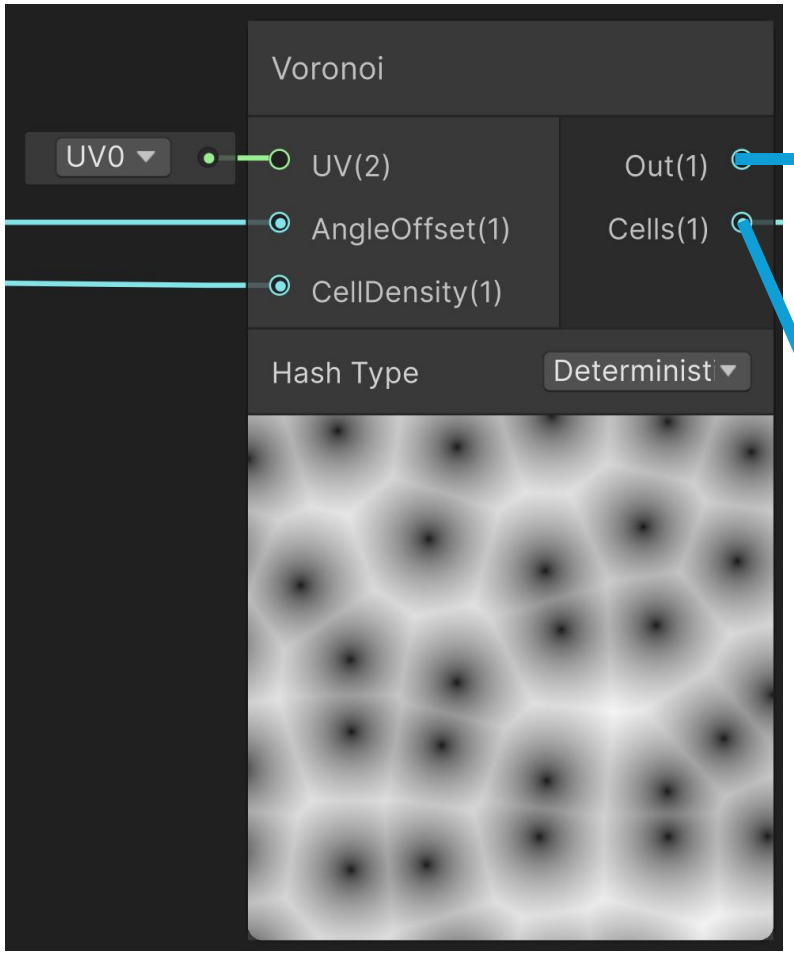
- 距離空間上の各位置に対して、配置された複数の点(母点)に対して、最も近い母点で領域分割した図形
- ドロネー図の双対図形
  - 母点を頂点とするドロネー図の各辺の中点から、垂直二等分線を他の線分の交点まで引いた図形
  - ドロネー図:頂点を辺でつないだ際にどの他の辺とも交差せず、3辺をで作る3角形の頂点を通る円(超球)が他の頂点を含んでいない図形

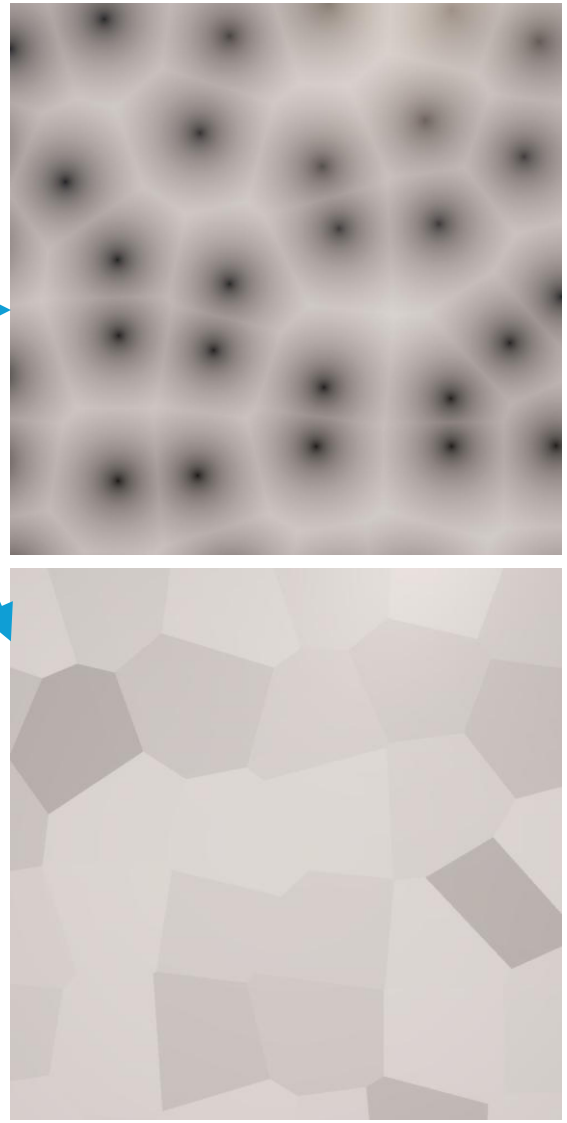


https://ja.wikipedia.org/wiki/%E3%83%9C%E3%83%AD%E3%83%8E%E3%82%A4%E5%9B%B3

Unity でのボロノイノード

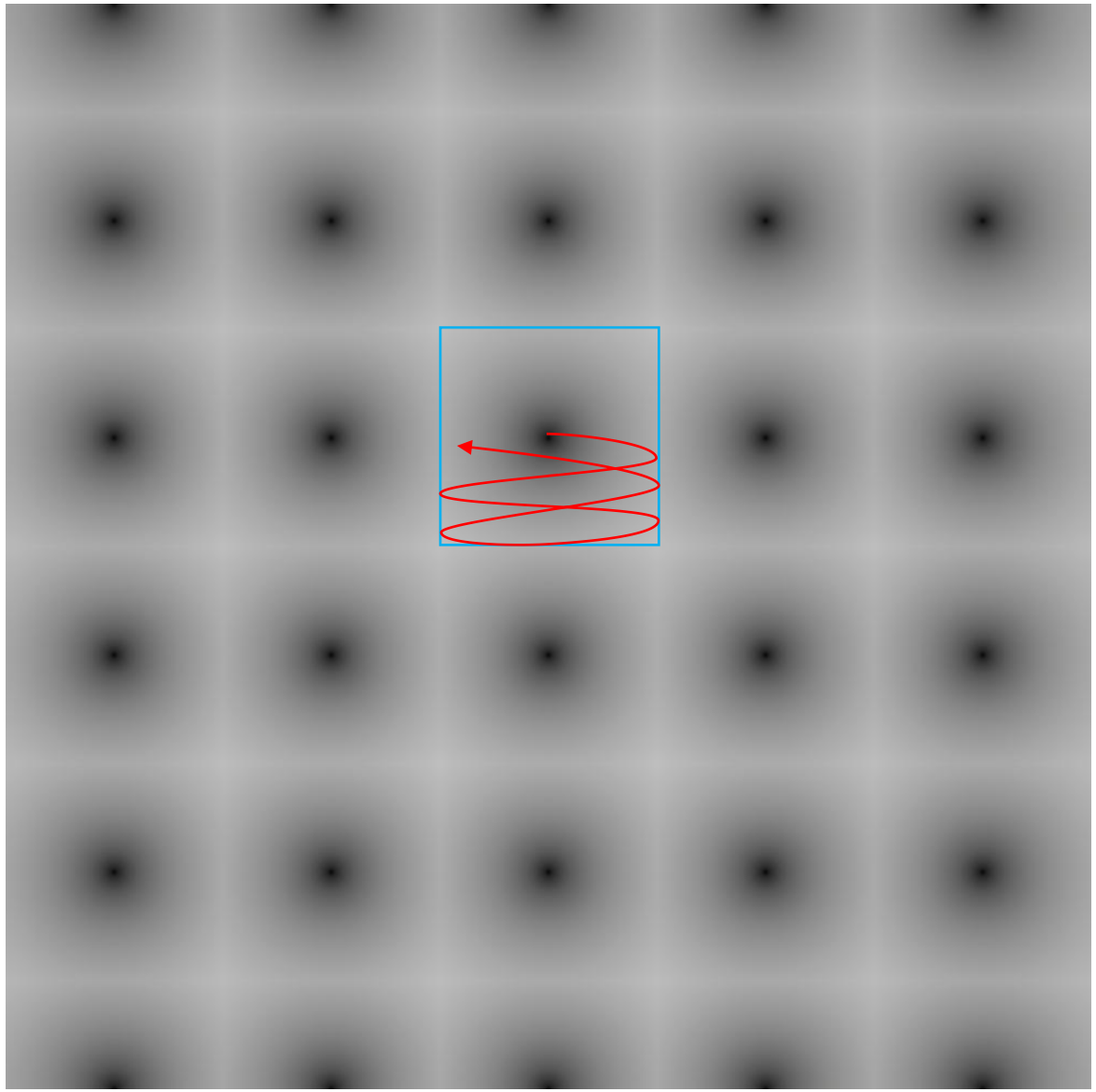
- ボロノイ図の情報を出力
  - Out:母点からの距離
  - Cells:セルのインデックス





Unityでのボロノイ図

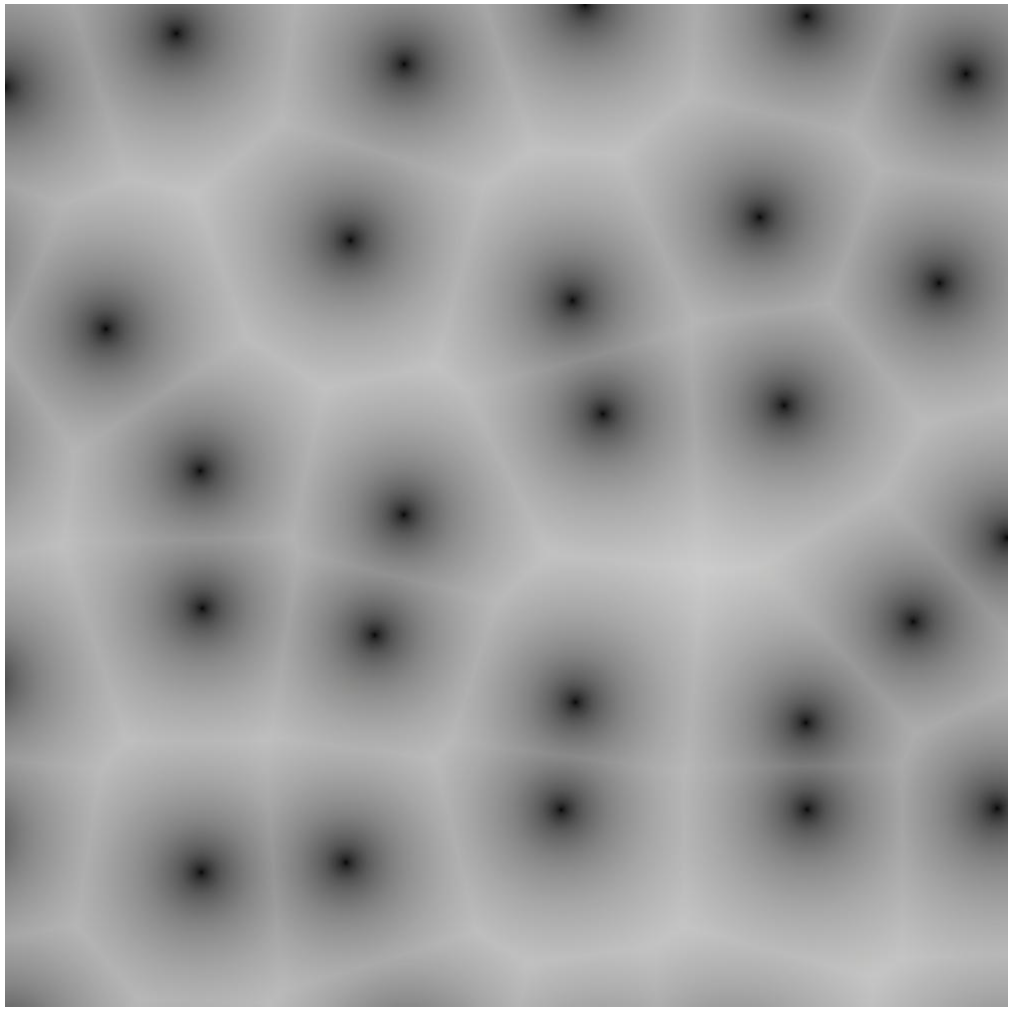
- ブロック分割された中に母点は置かれ、その中をパラメータ（Angle Offset）の値に応じて移動する
  - 最近傍の母点は自分がいるブロックとその近傍の9ブロック内に存在する
    - 一般的な母点の位置に対するドロネー図の生成は困難であるが、この配置なら各ピクセルでの最近傍の母点の位置の検出時間はO(1)



本日の内容

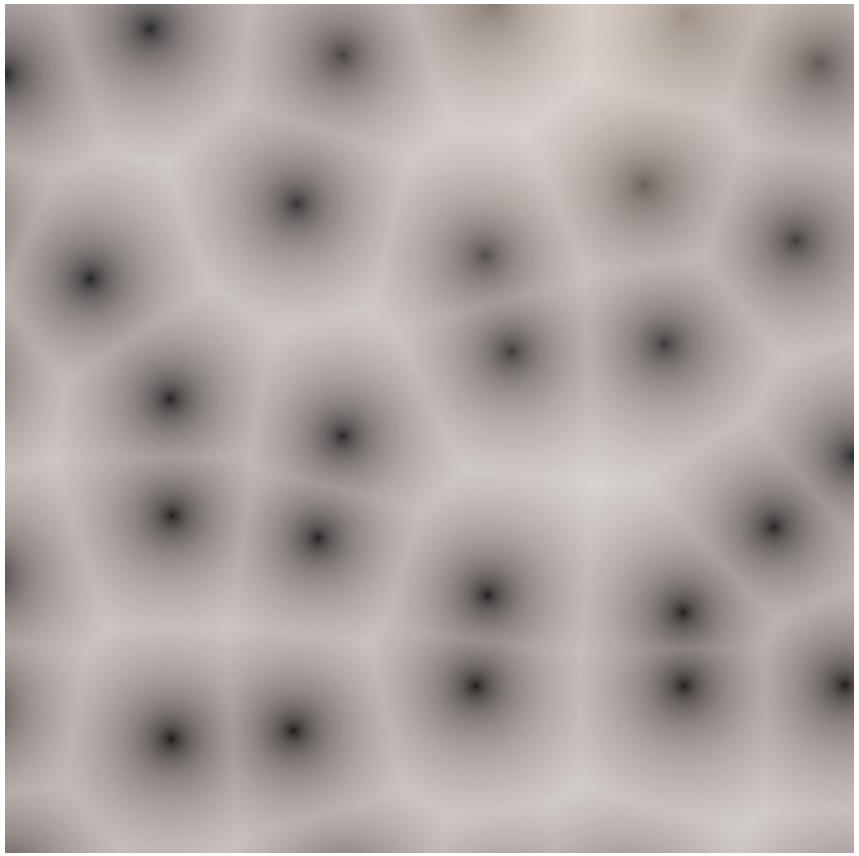
- ノイズの概要
- 基本的なノイズ
- FBM
- 加工したFBM
- ボロノイ図
  - 概要
  - 母点からの距離
  - 集光模様
  - セルインデックス
  - ステンドグラス風表現
- 炎
- ディゾルブ

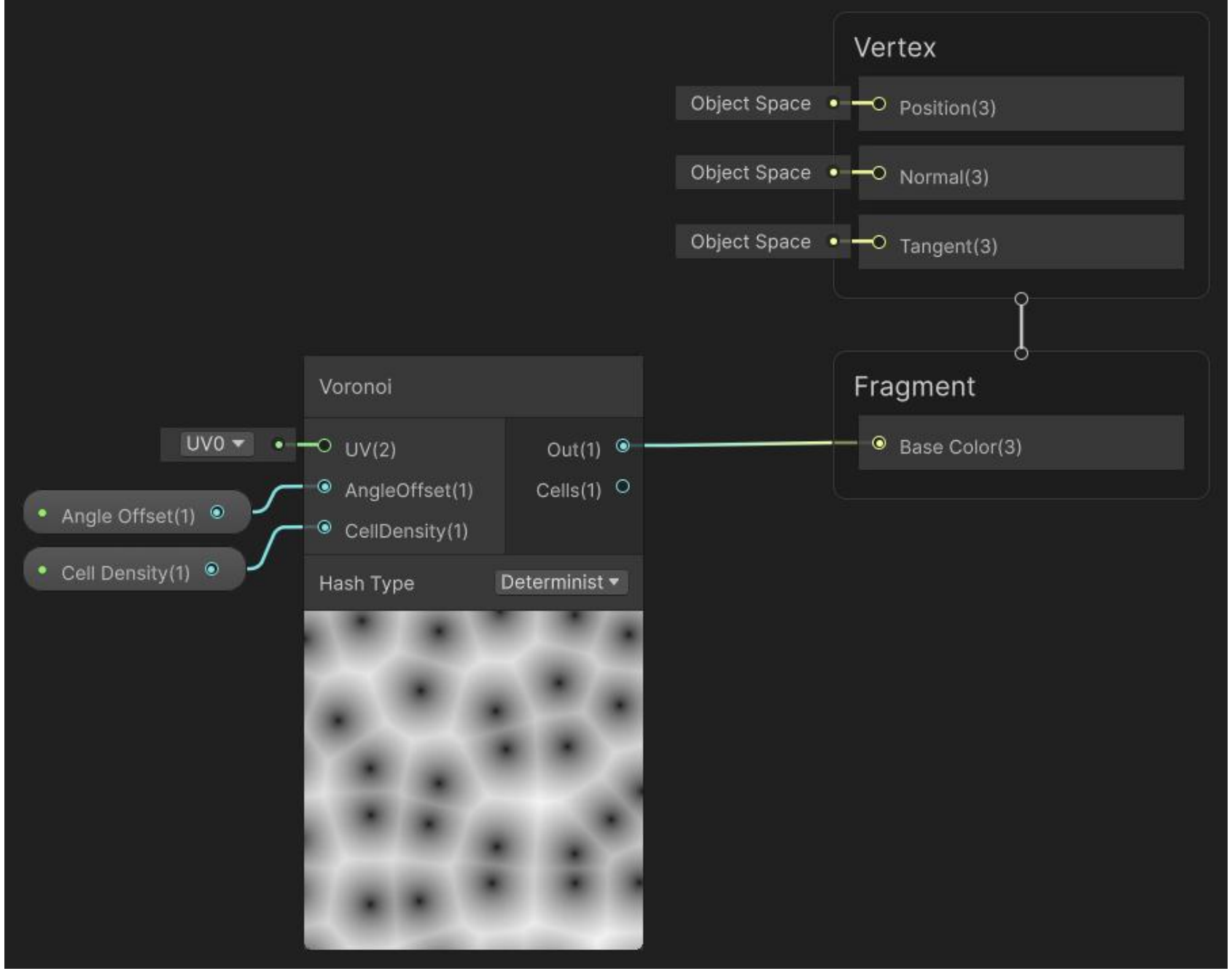
シーン：4 Voronoi Scene



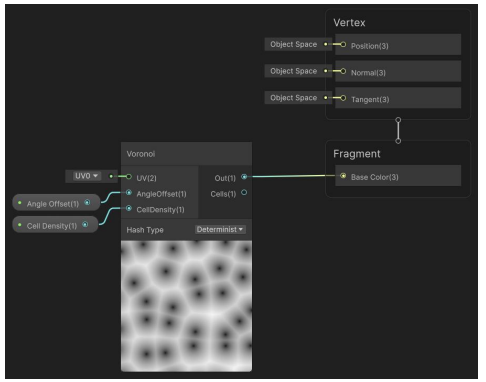
やってみよう

- Shader Graphを作成
  - 「4 Voronoi/1 voronoi Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を2つ追加(次は設定例)
    - Angle Offset
      - Float型 (Mode: Slider)
      - 範囲: [0, 10]
      - 初期値: 2
    - Cell Density
      - Float型 (Mode: Slider)
      - 範囲: [0, 10]
      - 初期値: 5




Vertex
Object Space
Position(3)
Object Space
Normal(3)
Object Space
Tangent(3)
Fragment
Base Color(3)
Voronoi
UV0
UV(2)
Out(1)
AngleOffset(1)
Cells(1)
CellDensity(1)
Angle Offset(1)
Cell Density(1)
Hash Type
Determinist

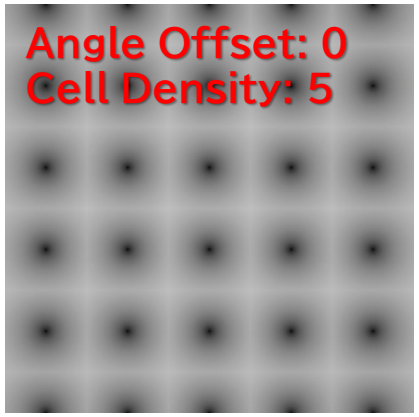
パラメータに伴う変化

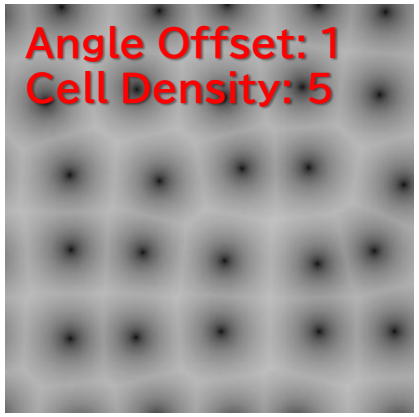


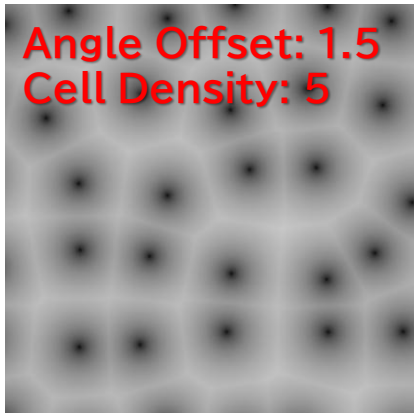
Angle Offset

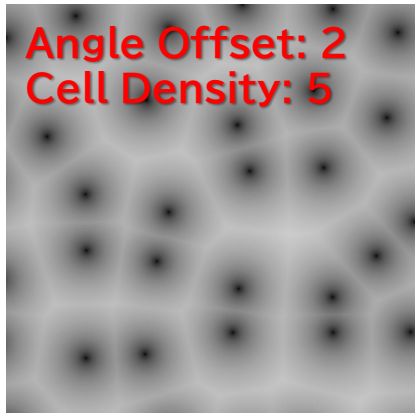
Cell Density

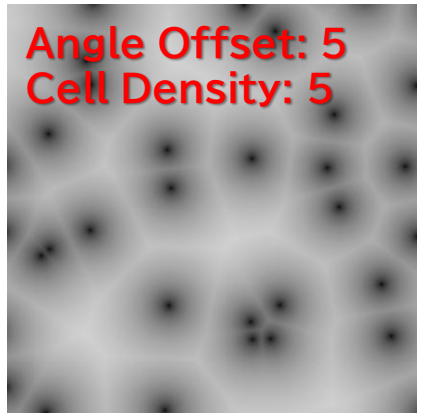
- Angle Offset: 母点の位置、Cell Density：母点の位置

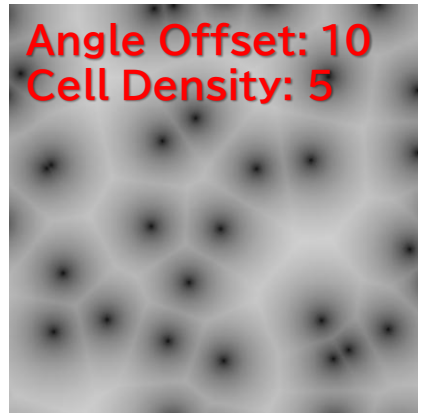


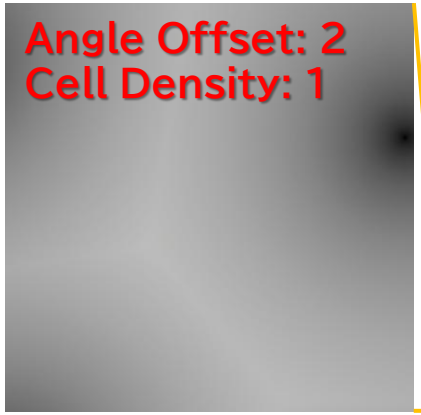


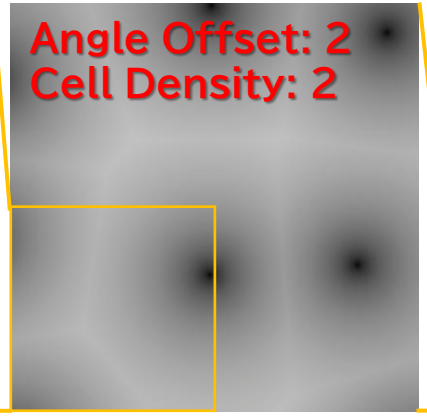


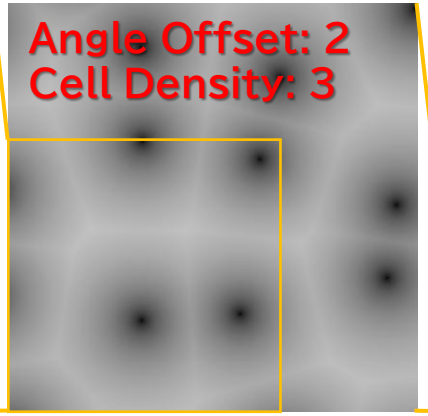


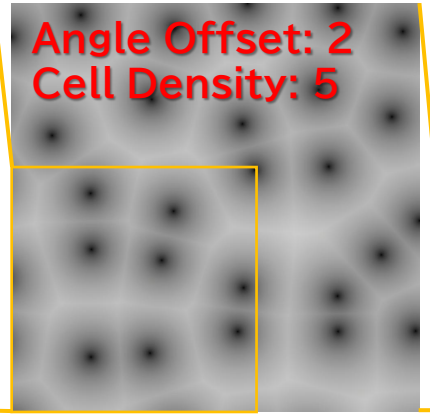


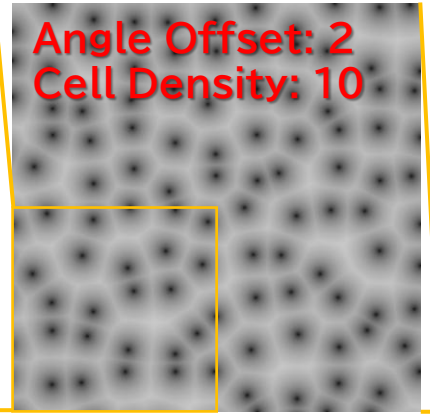


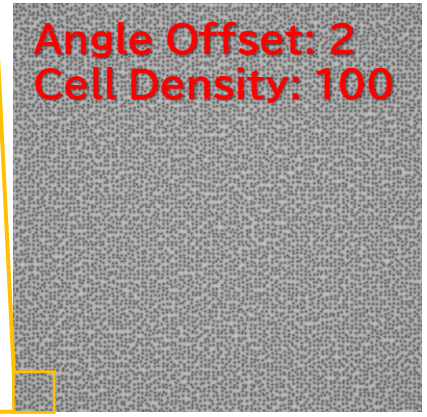












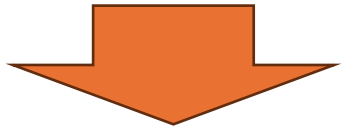
本日の内容

- ノイズの概要
- 基本的なノイズ
- FBM
- 加工したFBM
- ボロノイ図
  - 概要
  - 母点からの距離
  - 集光模様
  - セルインデックス
  - ステンドグラス風表現
- 炎
- ディゾルブ

シーン：4 Voronoi Scene

ボロノイ図の応用

- Angle Offsetを変えると図を連続的に変化できる
- ボロノイ図のエッジは亀甲文様的
  - 3～8角形の集合

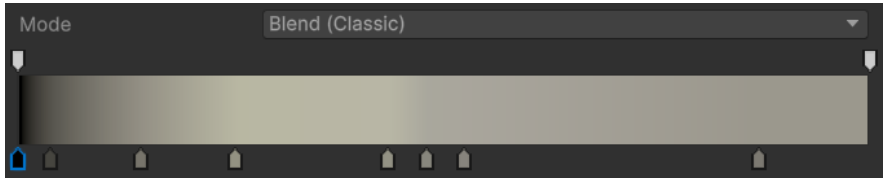


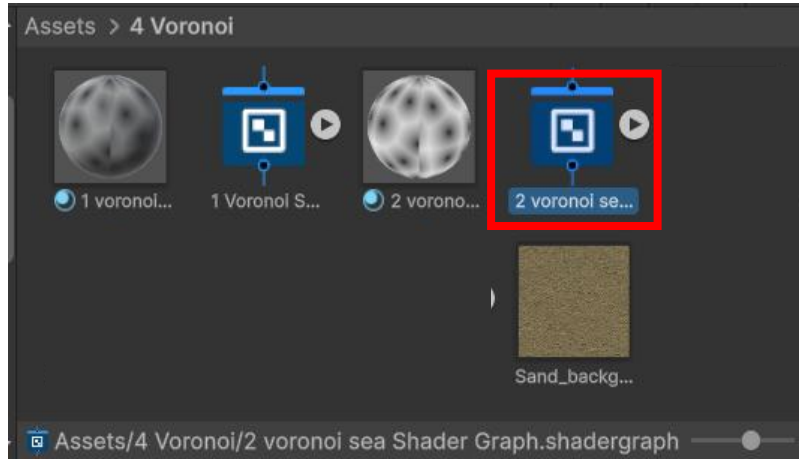
- 今回は、波がつくる集光模様として扱ってみる
  - Angle Offsetを周期的に連続に変化させることで、模様を揺らめかせる

Jean-Marc Kuffer, Woman freediver underwater in the dark with sunlight creating caustics and reflecting in the mask.

やってみよう

- Shader Graphを作成
  - 「4 Voronoi/2 voronoi sea Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を4つ追加(次は設定例)
    - Angle Offset: 模様が動く基準の位置
      - Float型 (Mode: Slider), 範囲: [0, 10], 初期値: 4.4
    - Power: 光の鋭さの調整用
      - Float型 (Mode: Slider), 範囲: [0, 10],初期値: 6.1
    - Wave Size:模様の変化の大きさ
      - Float型 (Mode: Slider), 範囲: [0, 10], 初期値: 1.2
    - Frequency: 模様が変化する角速度
      - Float型 (Mode: Slider), 範囲: [0, 10], 初期値: 1
  - グラデーション
    - 光の色付け
  - 画像
    - 地面などの模様を用意しよう






地面の色
パラメータ「Wave Size」の大きさで
「Angle Offset」を振動させる
色付け
発光として
処理
時間で動くsin
べき乗を使って
アクセントをつける
Sample Texture 2D
Sand_backg
Texture(T2)
UV0
UV(2)
Sampler(SS)
RGBA(4)
R(1)
G(1)
B(1)
A(1)
Type
Default
Space
Tangent
Vertex
Object Space
Position(3)
Normal(3)
Tangent(3)
Fragment
Base Color(3)
Smoothness(1)
Normal (Tangent Space)(3)
Ambient Occlusion(1)
Emission(3)
Metallic(1)
Sample Gradient
Gradient(G)
Time(1)
Out(4)
Power
A(1)
B(1)
Out(1)
Power(1)
Voronoi
UV(2)
AngleOffset(1)
CellDensity(1)
Cells(1)
Hash Type
Deterministic
Add
Angle Offset(1)
Multiply
Wave Size(1)
Sine
In(1)
Frequency(1)
Time
Time(1)
Sine Time(1)
Cosine Time(1)
Delta Time(1)
Smooth Delta(1)

特性

- Angle Offsetへの入力が0を通るとそろった絵ができてしまう
- 大きく動かすと母点が円のように動くことが見える

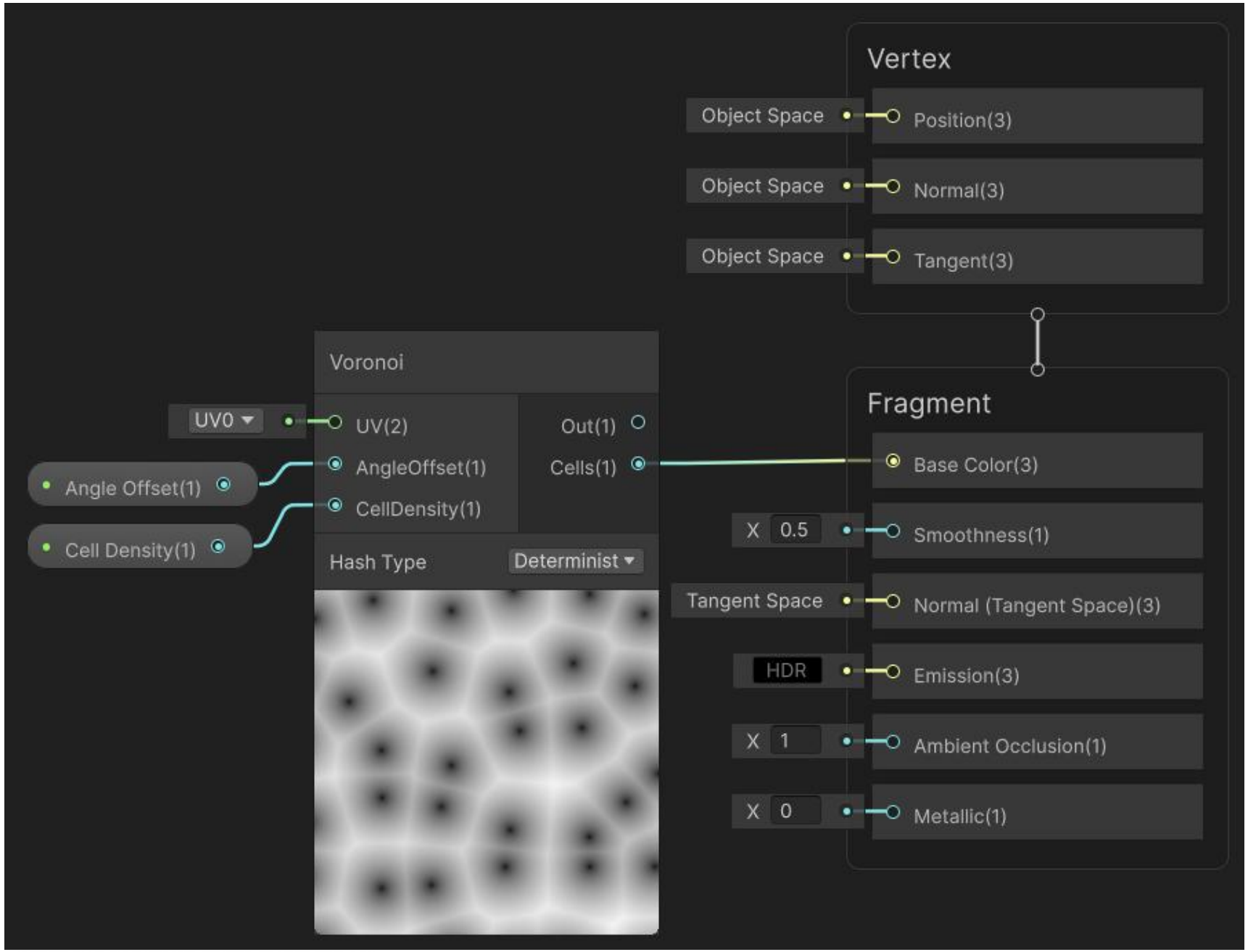
本日の内容

- ノイズの概要
- 基本的なノイズ
- FBM
- 加工したFBM
- ボロノイ図
  - 概要
  - 母点からの距離
  - 集光模様
  - セルインデックス
  - ステンドグラス風表現
- 炎
- ディゾルブ

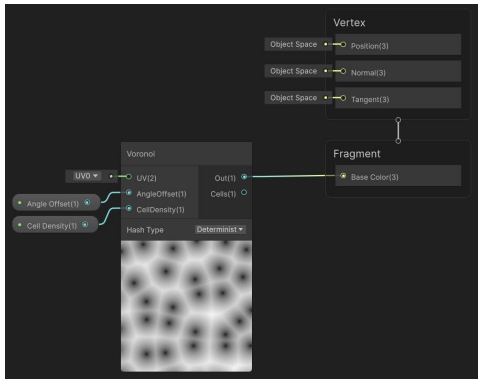
シーン：4 Voronoi Scene

やってみよう

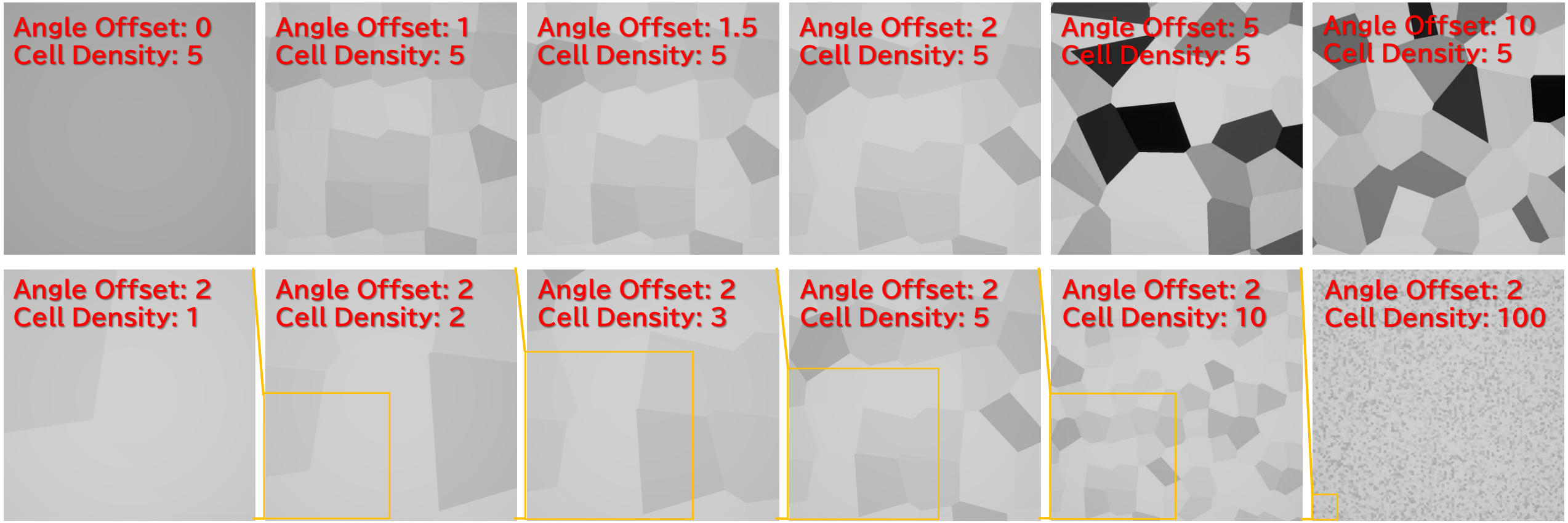
- Shader Graphを作成
  - 「4 Voronoi/3 voronoi cells Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を2つ追加(次は設定例)
    - Angle Offset
      - Float型 (Mode: Slider)
      - 範囲: [0, 10]
      - 初期値: 2
    - Cell Density
      - Float型 (Mode: Slider)
      - 範囲: [0, 10]
      - 初期値: 5


Vertex
Object Space
Position(3)
Object Space
Normal(3)
Object Space
Tangent(3)
Voronoi
UV0
UV(2)
Out(1)
Angle Offset(1)
AngleOffset(1)
Cells(1)
Cell Density(1)
CellDensity(1)
Hash Type
Deterministic
Fragment
Base Color(3)
X 0.5
Smoothness(1)
Tangent Space
Normal (Tangent Space)(3)
HDR
Emission(3)
X 1
Ambient Occlusion(1)
X 0
Metallic(1)

パラメータに伴う変化



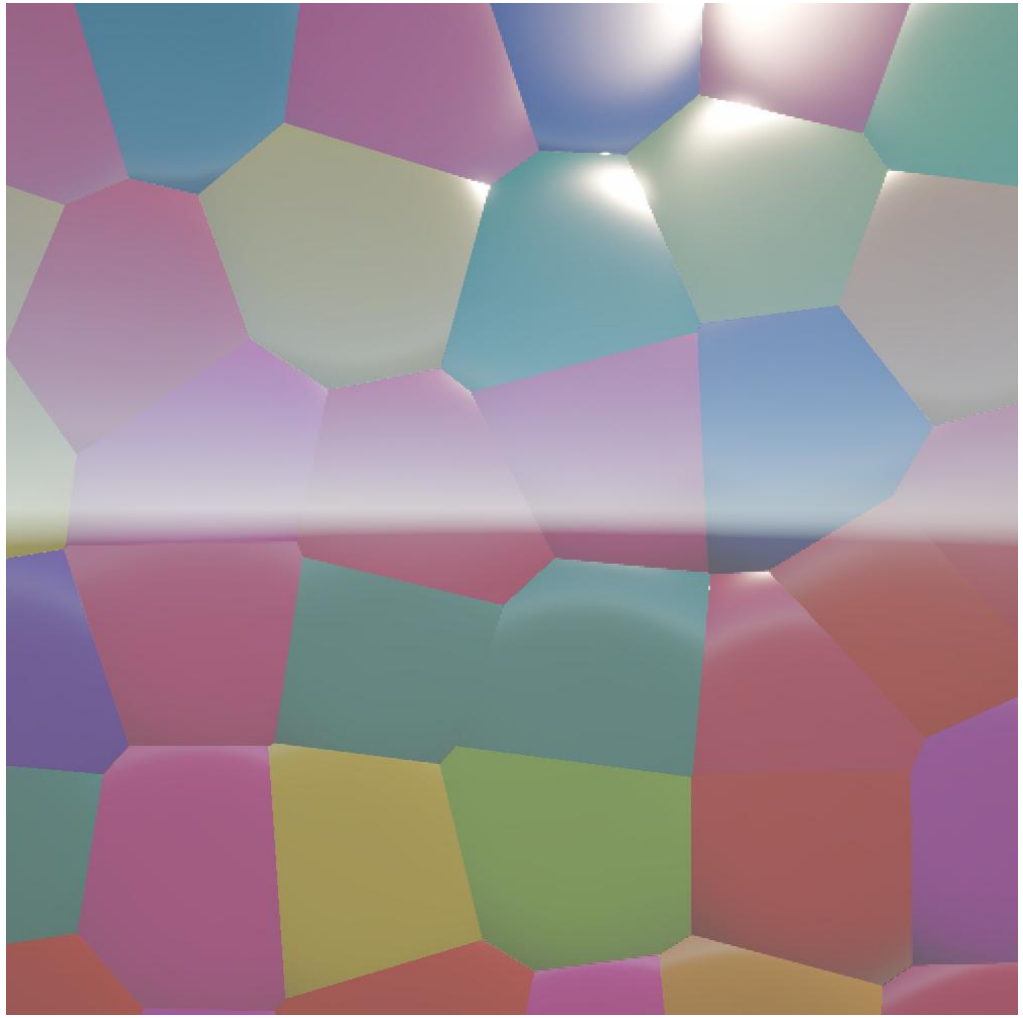
- Angle Offset: 母点の位置、Cell Density:母点の位置



本日の内容

- ノイズの概要
- 基本的なノイズ
- FBM
- 加工したFBM
- ボロノイ図
  - 概要
  - 母点からの距離
  - 集光模様
  - セルインデックス
  - ステンドグラス風表現
- 炎
- ディゾルブ

シーン：4 Voronoi Scene

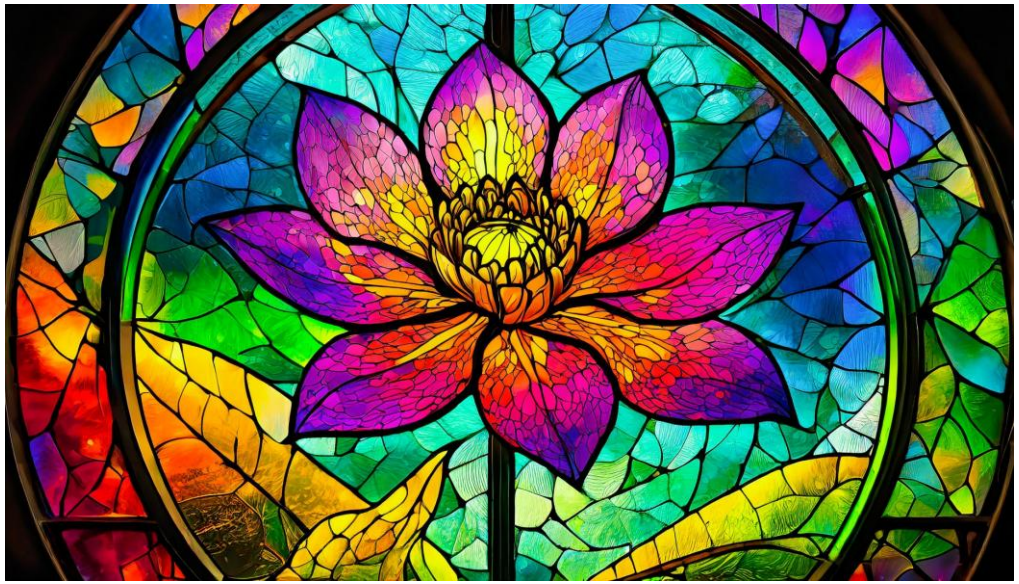


ボロノイ図の応用

- セルインデックスは空間を非一様に分割する

- ステンドグラスの模様として扱ってみる

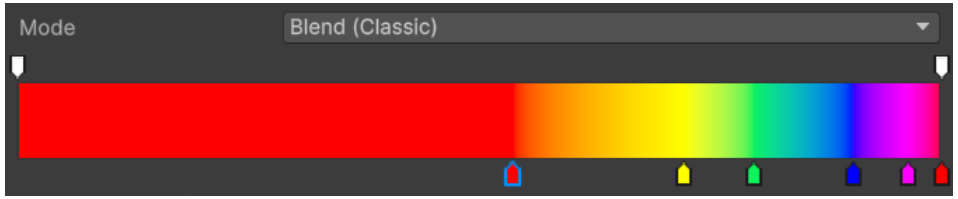
- ひびの表現も良いかもね



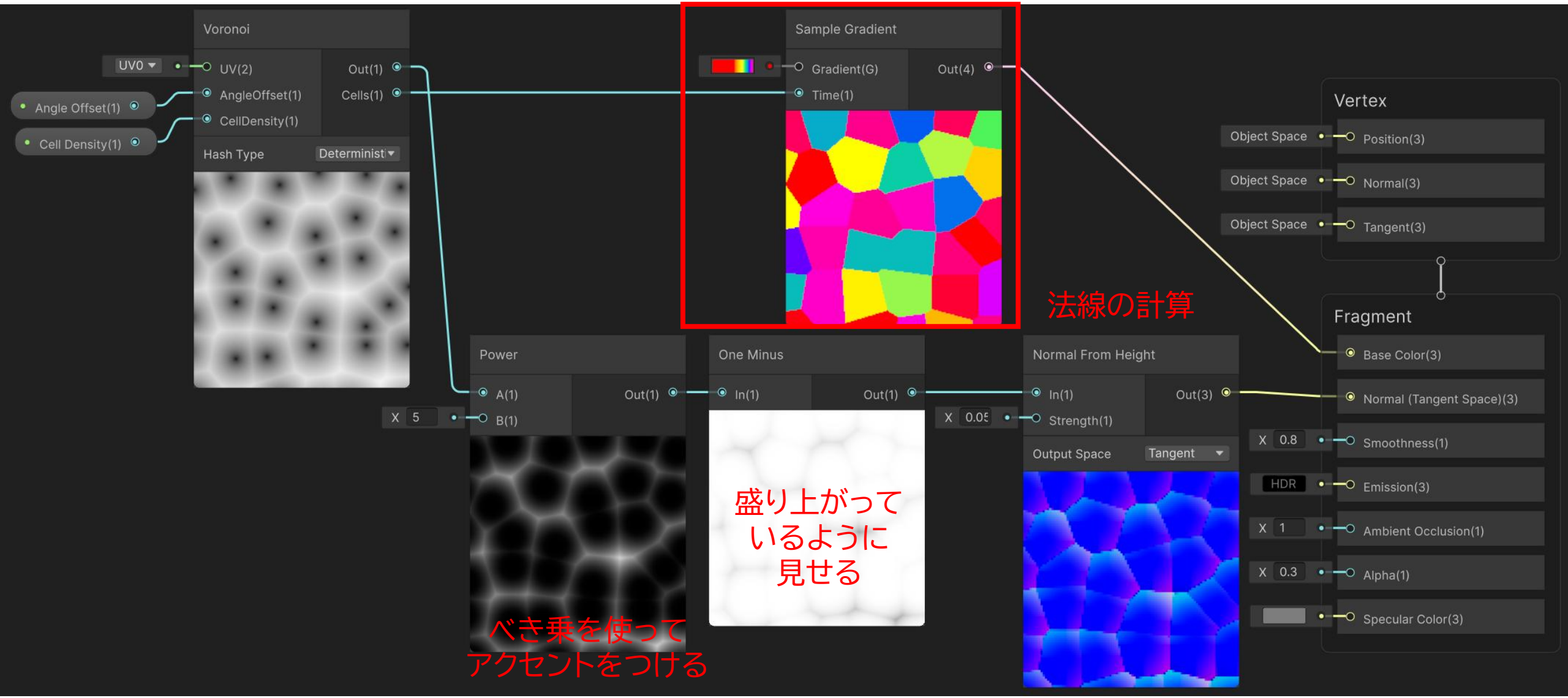
ステンドグラスの窓、イラスト
https://www.publicdomainpictures.net/jp/view-image.php?image=600059

やってみよう

- Shader Graphを作成
  - 「4 Voronoi/4 voronoi glass Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を2つ追加(次は設定例)
    - Angle Offset
      - Float型 (Mode: Slider)
      - 範囲: [0, 10]
      - 初期値: 2
    - Cell Density
      - Float型 (Mode: Slider)
      - 範囲: [0, 10]
      - 初期値: 5
  - グラデーション
    - 模様の色付け



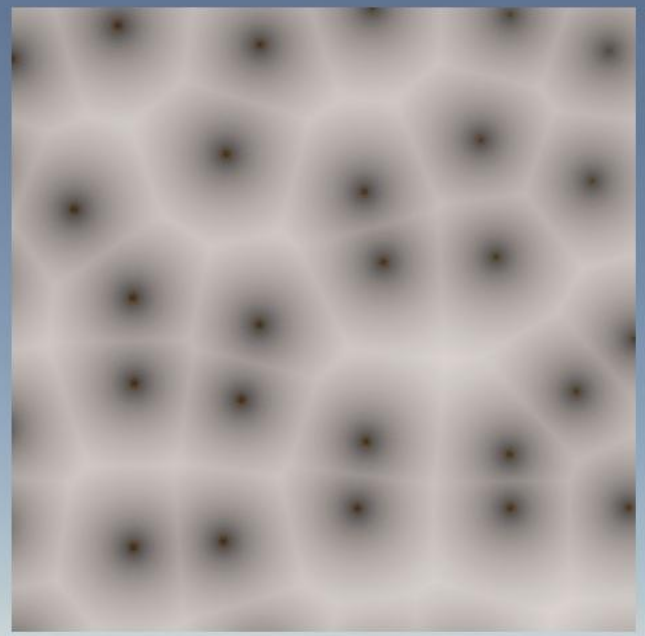
セルのインデックスを
グラデーションで色をばらつかせる

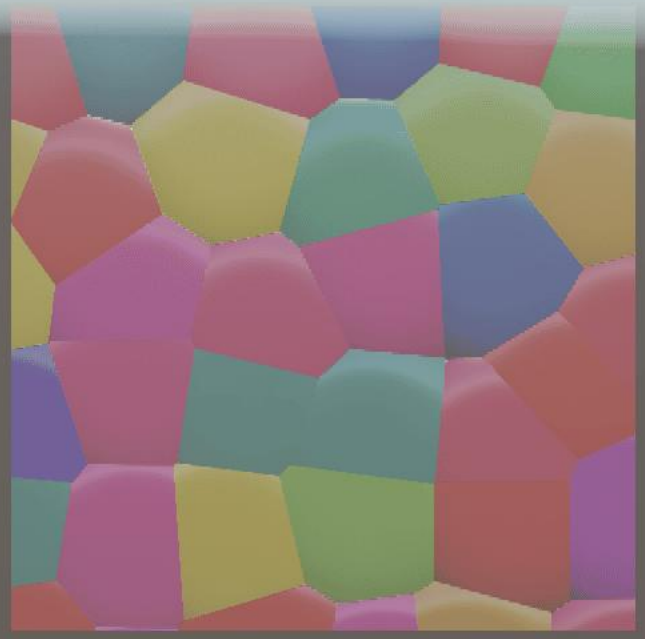


特性

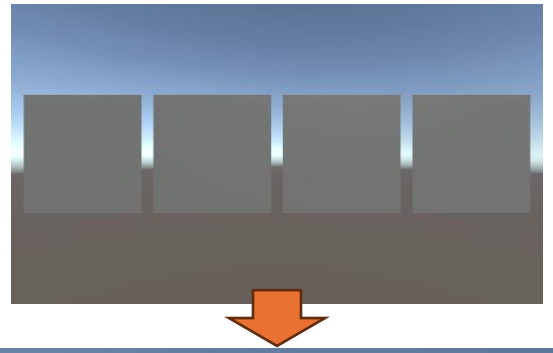
- 模様のデザインができない
- IDの値がよくわからない
  - 調べれば出てくるかな？

完成



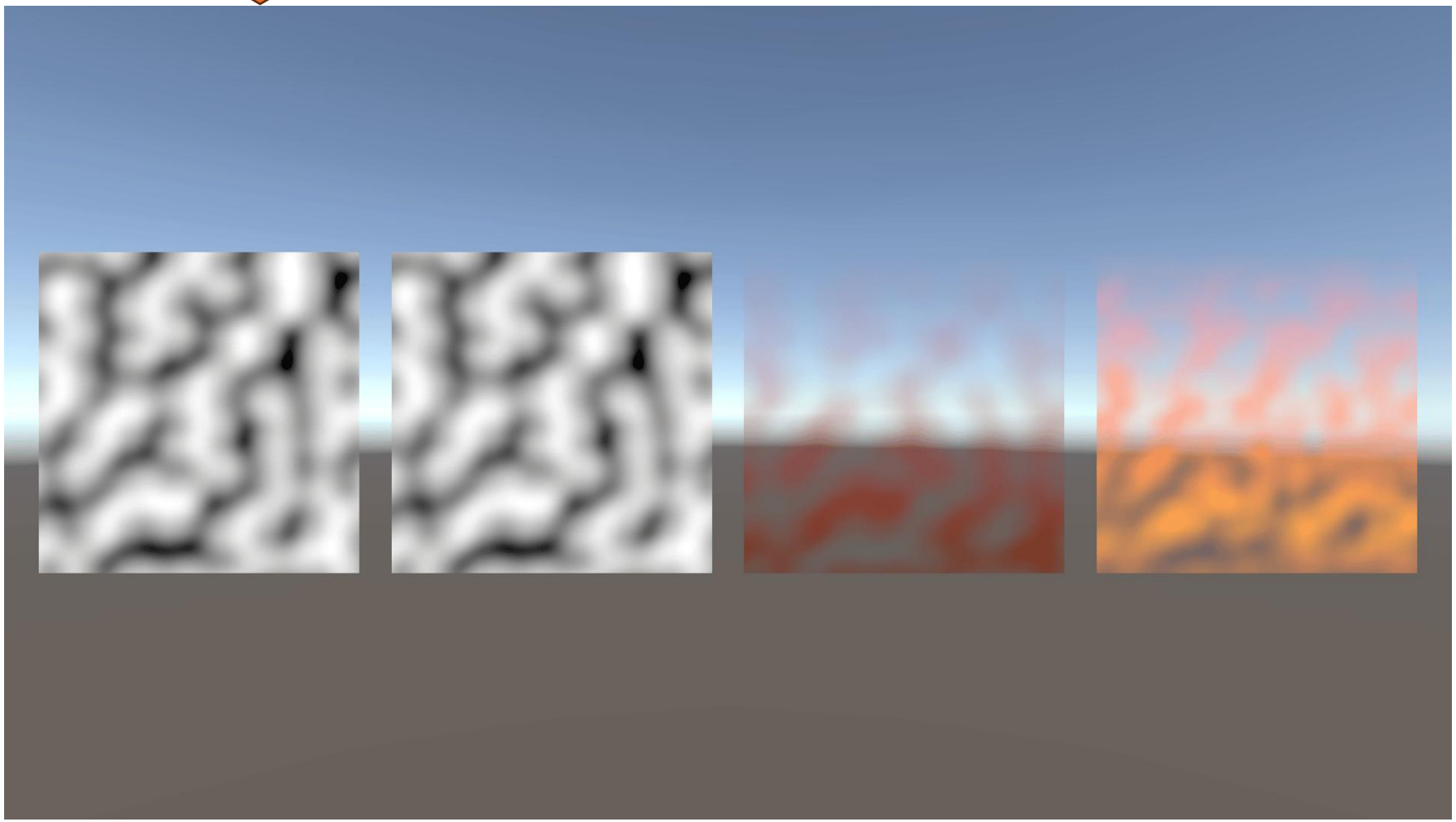


本日の内容



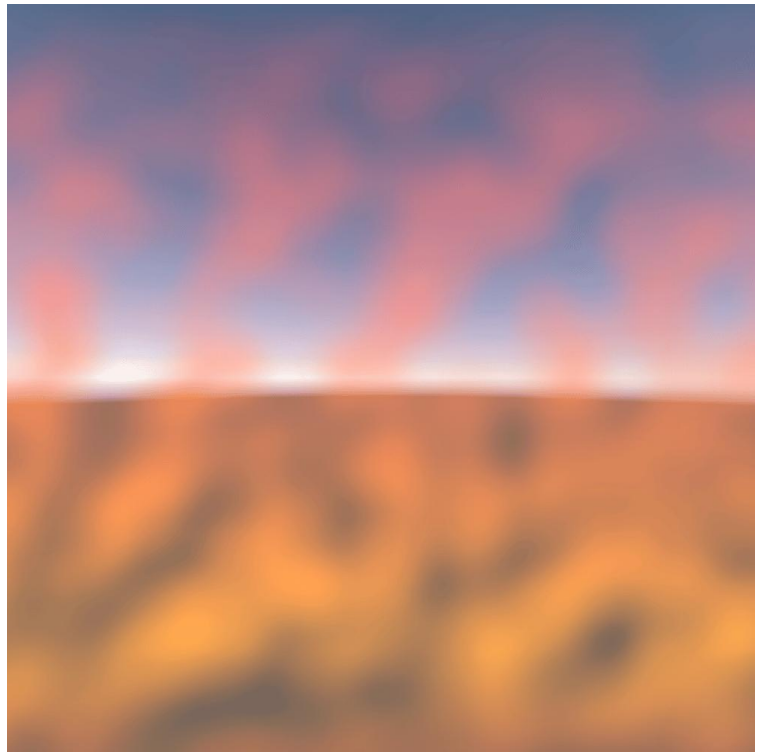
シーン：5 Fire Scene

- ノイズの概要
- 基本的なノイズ
- FBM
- 加工したFBM
- ボロノイ図
- 炎
- ディゾルブ



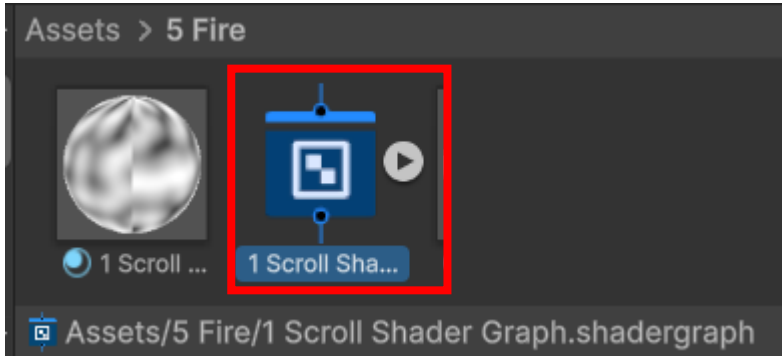
ノイズの応用の応用

- 炎
  - パーティクルでの表現が多い
    - パーティクルエディタがあれば、エフェクトデザイナーが単独で作りこめる
  - パーティクルの弱点
    - 半透明の重ね合わせ
      - 塗りつぶし回数を減らせない
        - 現在のコンピュータのボトルネックはメモリ転送なので、方向性として悪い
  - 一回の描画でそれなりの表現ができるとその方が速度が速い状況もある
    - 実際には、プロファイルをきちんと取って比較しましょう



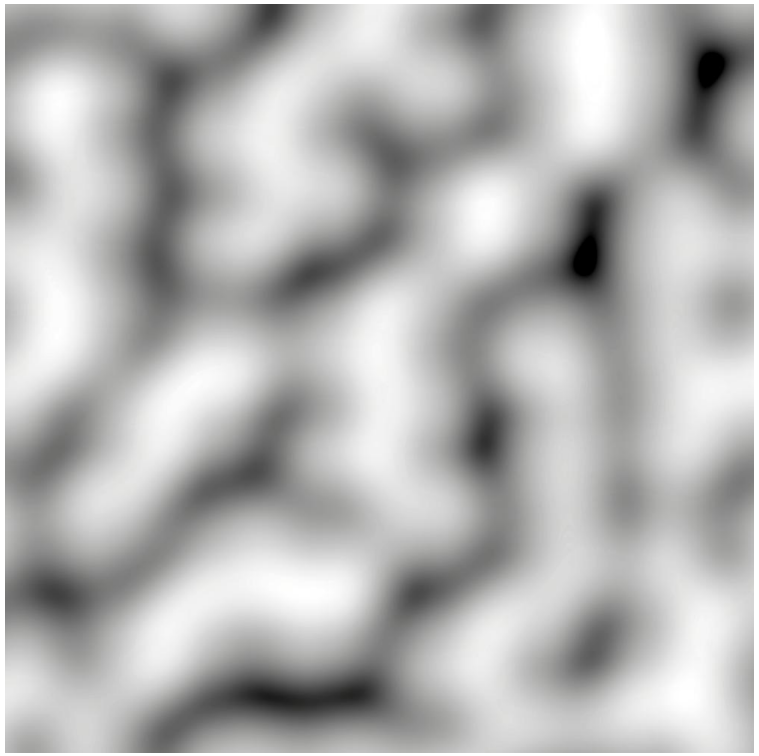
Jean-Marc Kuffer, Woman freediver underwater in the dark with sunlight creating caustics and reflecting in the mask.

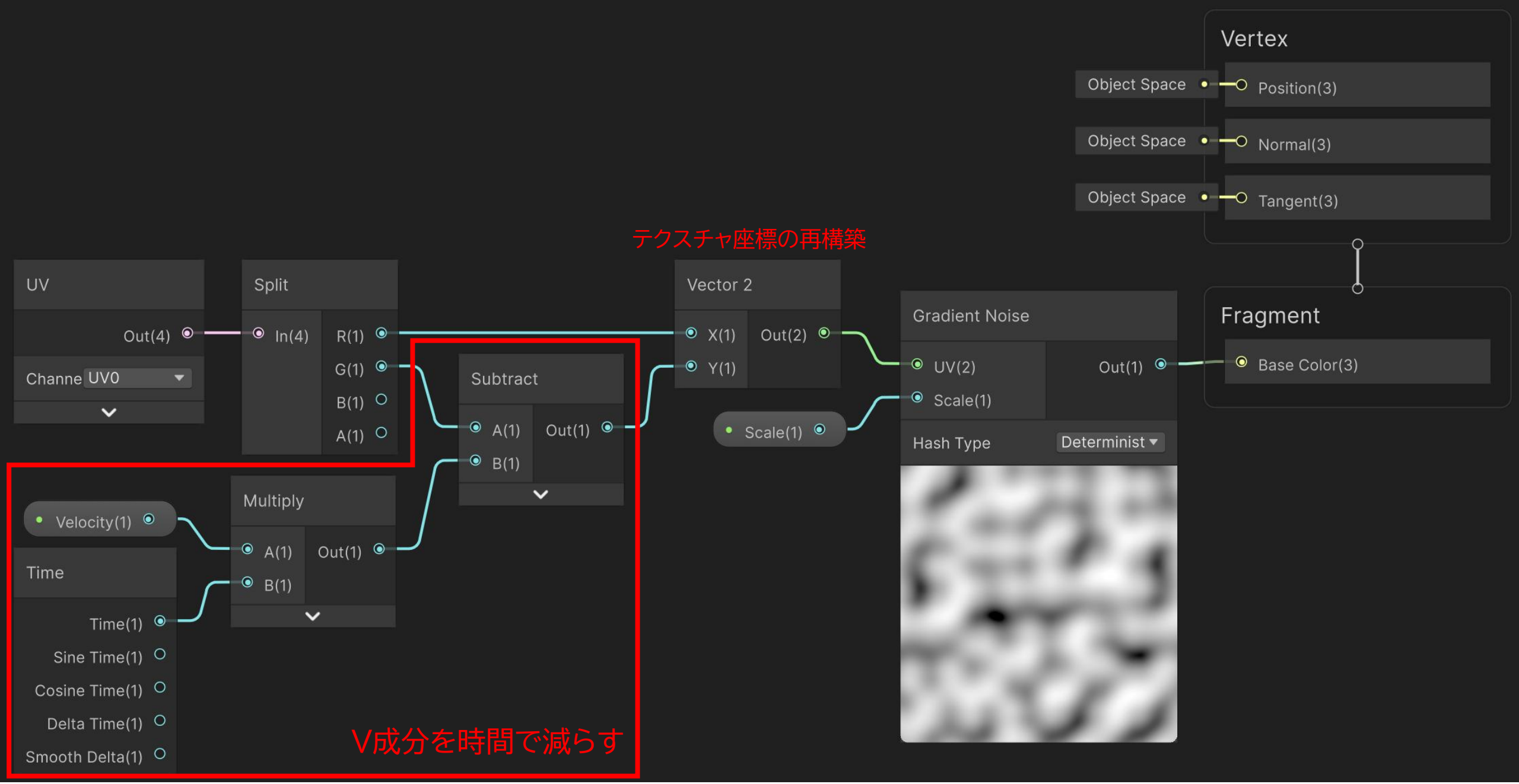
やってみよう：その1



ノイズを上方向にスクロール

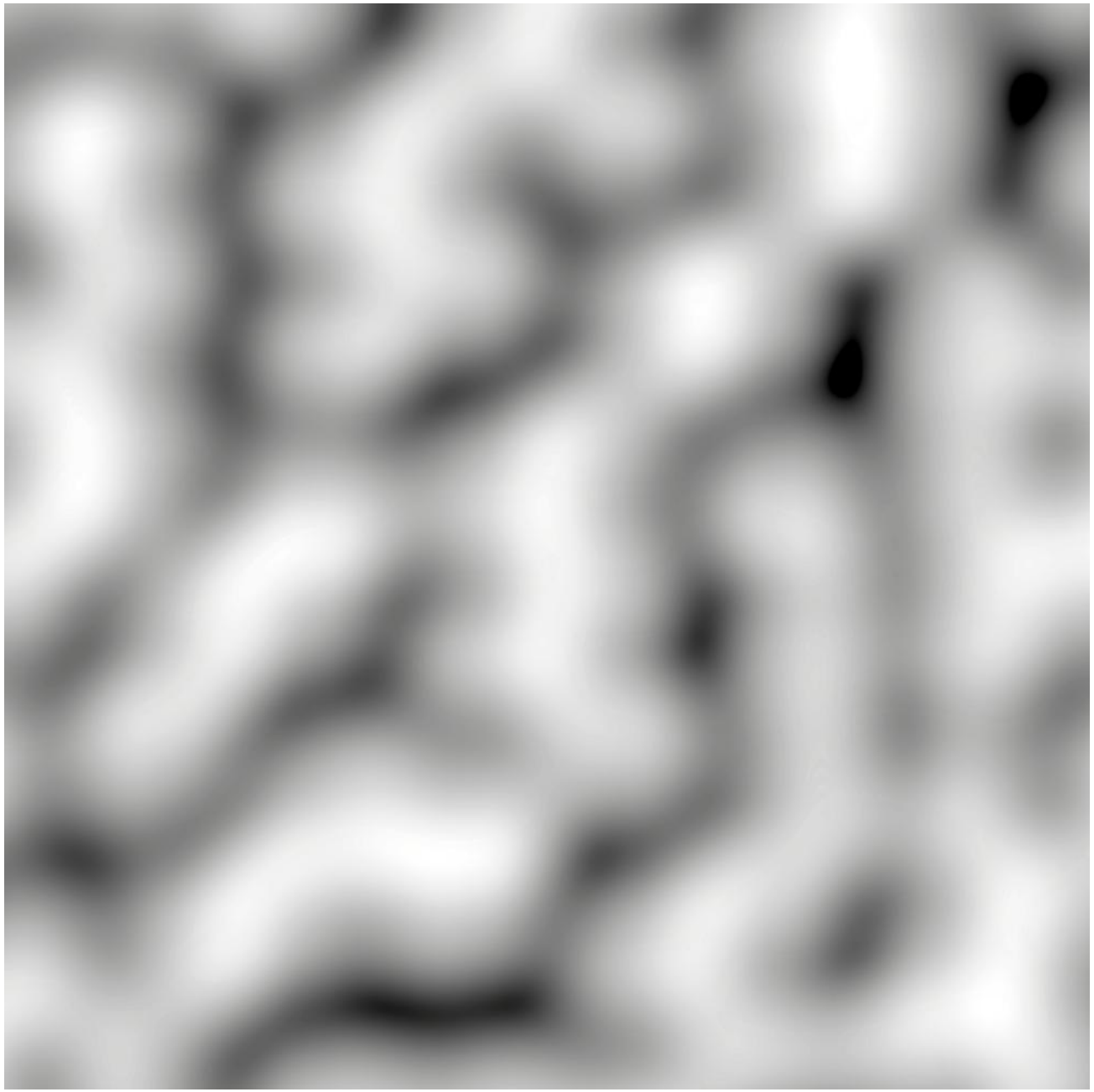
- Shader Graphを作成
  - 「5 Fire/1 Scroll Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を2つ追加(次は設定例)
    - Scale: ノイズの細かさ
      - Float型 (Mode: Slider)
      - 範囲：[0, 100]
      - 初期値: 7
    - Velocity：上昇速度
      - Float型 (Mode: Slider)
      - 範囲：[0, 10]
      - 初期値: 0.7



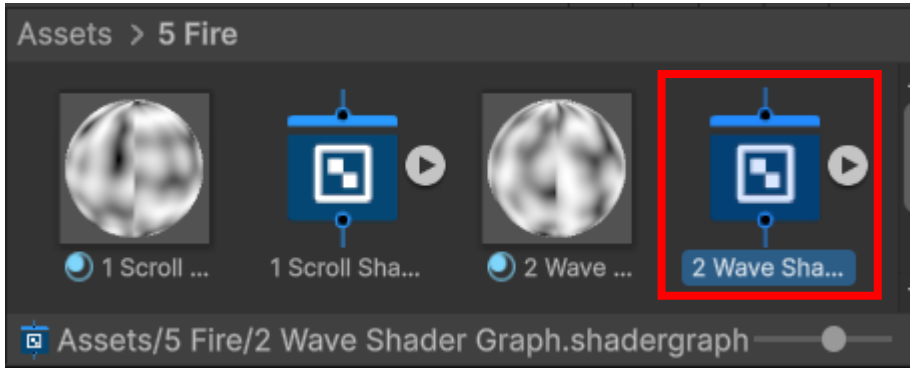

Vertex
Object Space
Position(3)
Object Space
Normal(3)
Object Space
Tangent(3)
テクスチャ座標の再構築
UV
Out(4)
Channe UV0
Split
In(4)
R(1)
G(1)
B(1)
A(1)
Subtract
A(1)
B(1)
Out(1)
Vector 2
X(1)
Y(1)
Out(2)
Scale(1)
Gradient Noise
UV(2)
Scale(1)
Out(1)
Hash Type
Determinist
Fragment
Base Color(3)
Velocity(1)
Multiply
A(1)
B(1)
Out(1)
Time
Time(1)
Sine Time(1)
Cosine Time(1)
Delta Time(1)
Smooth Delta(1)
V成分を時間で減らす

ひとまずの完成

- パラメータを変化させて
  どのように変わるか感じよう
  - Scale: ノイズの細かさ
  - Velocity：上昇速度

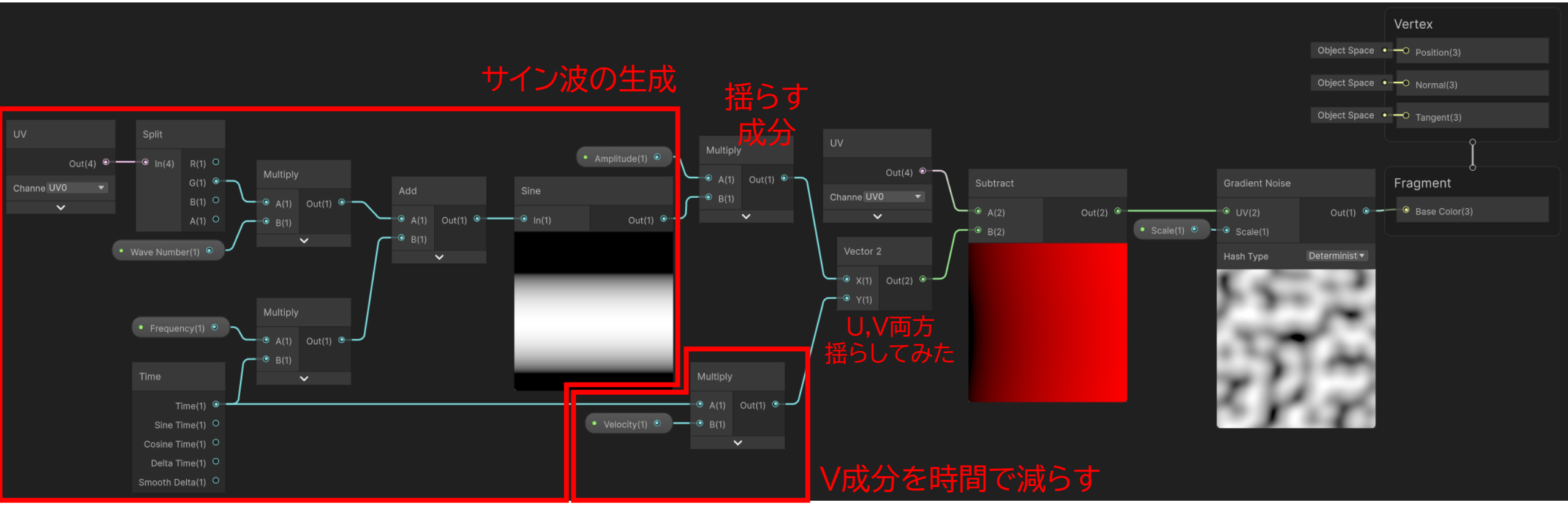


やってみよう:その2



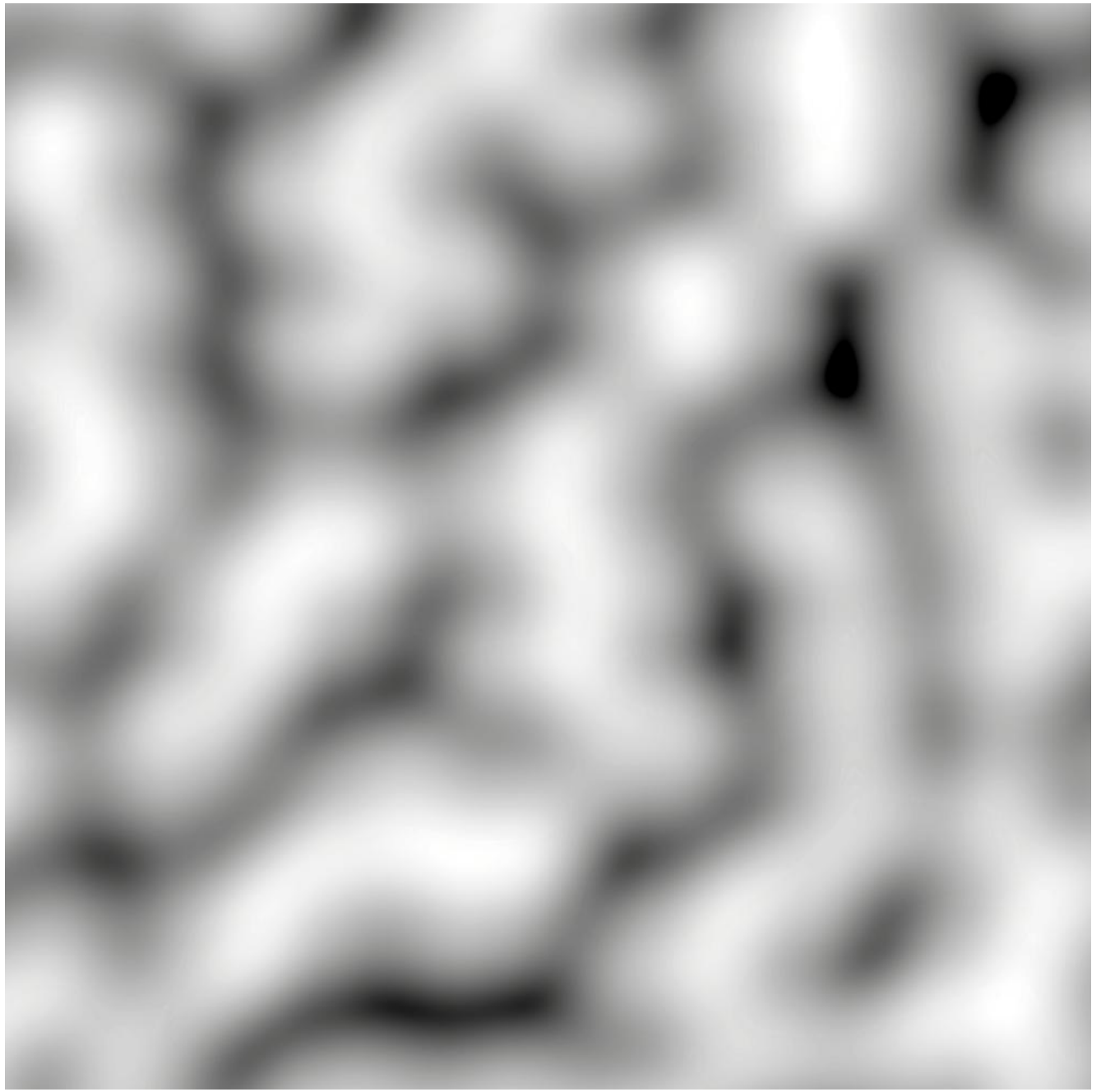
ノイズを左右に揺らす
- UVテクスチャ座標にサイン波を加算

- Shader Graphを作成
  - 「5 Fire/2 Wave Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を5つ追加(次は設定例)
    - Scale: ノイズの細かさ
      - 範囲: [0, 100] 初期値: 7 Float型 (Mode: Slider)
    - Velocity:上昇速度
      - 範囲: [0, 10] 初期値: 0.7 Float型 (Mode: Slider)
    - Amplitude:揺らす大きさ
      - 範囲: [0, 1] 初期値: 0.03 Float型 (Mode: Slider)
    - Wave Number: ゆらすサイン波の波数
      - 範囲: [0, 100] 初期値: 5 Float型 (Mode: Slider)
    - Frequency: ゆらすサイン波の角速度
      - 範囲: [0, 100] 初期値: 5 Float型 (Mode: Slider)


サイン波の生成
揺らす
成分
UV
Out(4)
Channe UV0
Split
In(4)
R(1)
G(1)
B(1)
A(1)
Wave Number(1)
Multiply
A(1)
B(1)
Out(1)
Add
Amplitude(1)
Sine
In(1)
Frequency(1)
Time
Time(1)
Sine Time(1)
Cosine Time(1)
Delta Time(1)
Smooth Delta(1)
Velocity(1)
Vector 2
X(1)
Y(1)
Out(2)
U,V両方
揺らしてみた
Subtract
Scale(1)
Gradient Noise
UV(2)
Hash Type
Deterministic
Vertex
Object Space
Position(3)
Normal(3)
Tangent(3)
Fragment
Base Color(3)
V成分を時間で減らす

ひとまずの完成

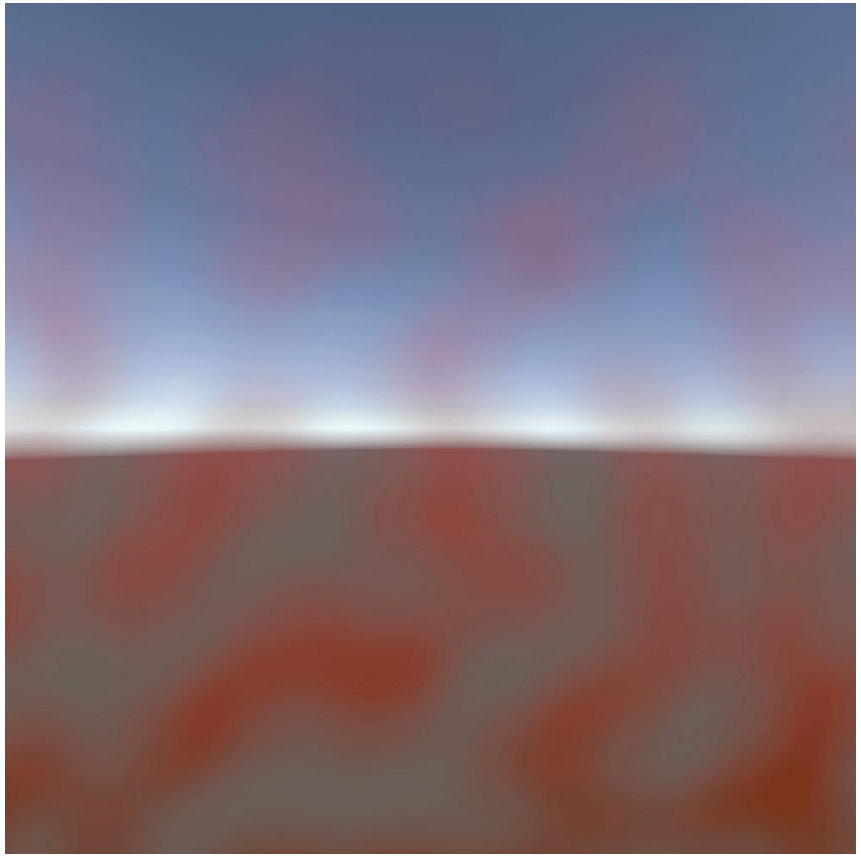
- パラメータを変化させて
  どのように変わるか感じよう
  - Scale
    - ノイズの細かさ
  - Velocity
    - 上昇速度
  - Amplitude
    - 揺らす大きさ
  - Wave Number
    - ゆらすサイン波の波数
  - Frequency
    - ゆらすサイン波の角速度

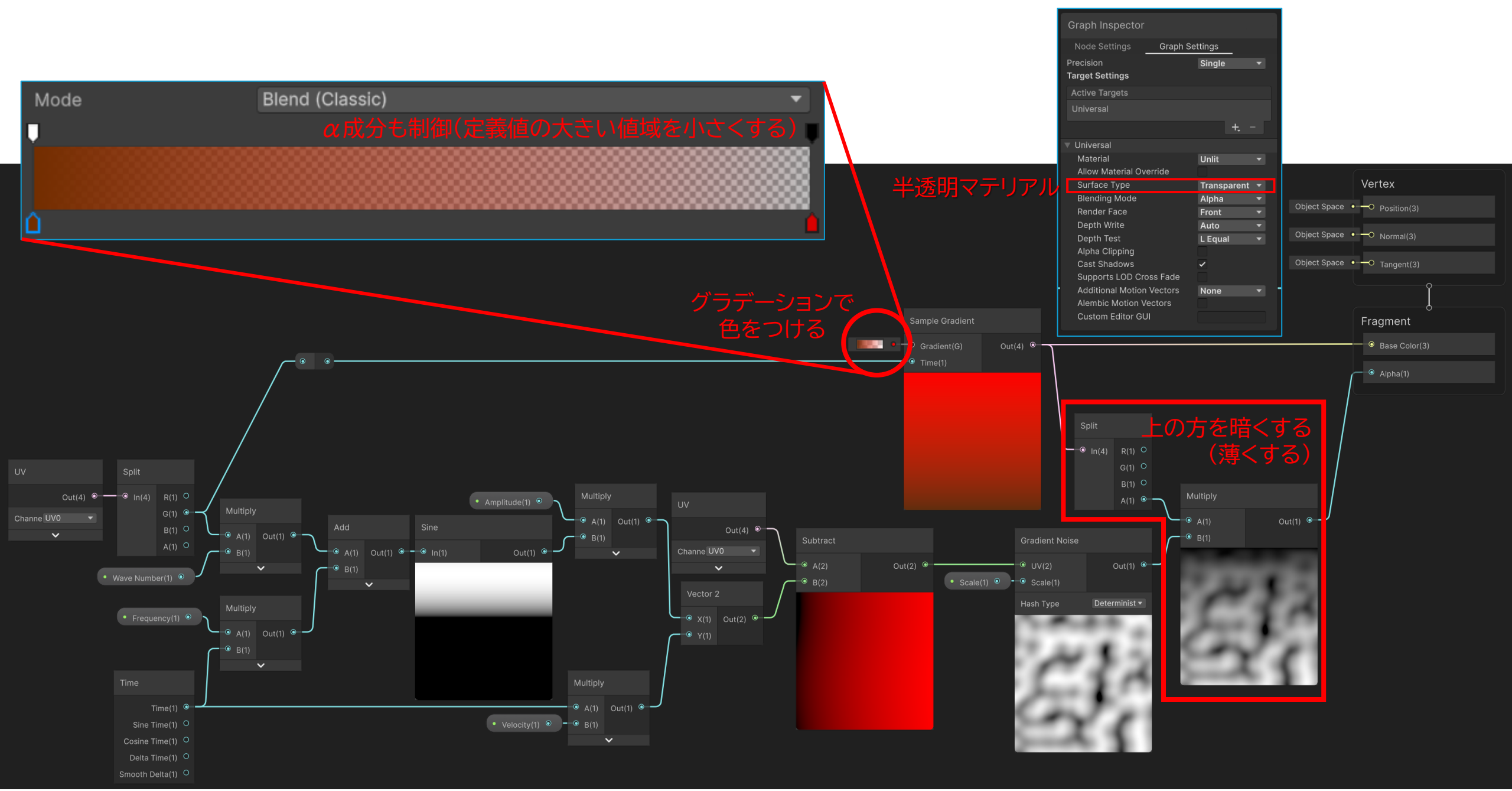


やってみよう:その3

色をつける

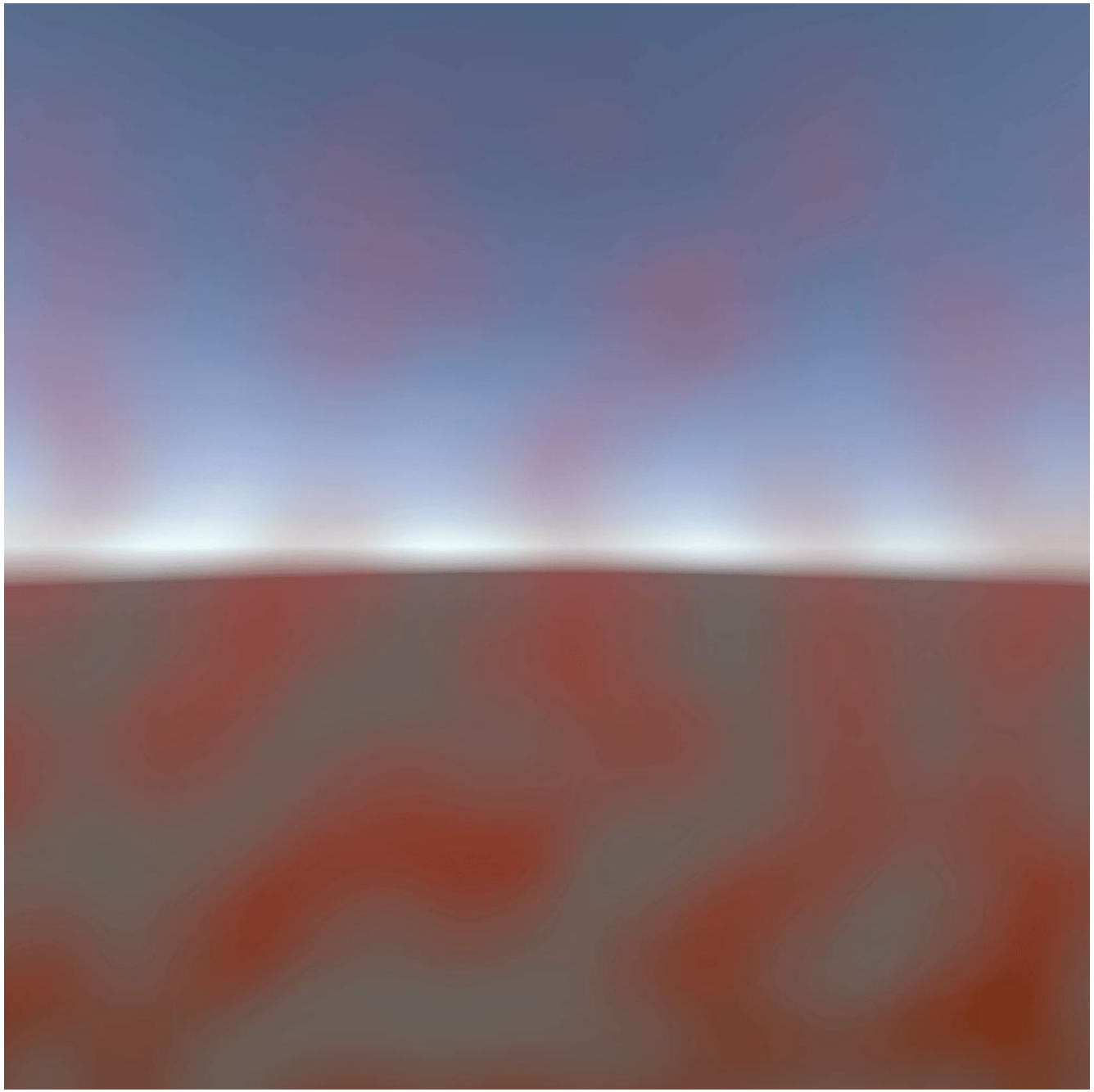
- Shader Graphを作成
  - 「5 Fire/3 Color Fire Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を5つ追加(次は設定例)
    - Scale: ノイズの細かさ
      - 範囲: [0, 100] 初期値: 7 Float型 (Mode: Slider)
    - Velocity:上昇速度
      - 範囲: [0, 10] 初期値: 0.7 Float型 (Mode: Slider)
    - Amplitude:揺らす大きさ
      - 範囲: [0, 1] 初期値: 0.03 Float型 (Mode: Slider)
    - Wave Number: ゆらすサイン波の波数
      - 範囲: [0, 100] 初期値: 5 Float型 (Mode: Slider)
    - Frequency: ゆらすサイン波の角速度
      - 範囲: [0, 100] 初期値: 5 Float型 (Mode: Slider)




Mode
Blend (Classic)
α成分も制御（定義値の大きい値域を小さくする）
半透明マテリアル
Graph Inspector
Node Settings
Graph Settings
Precision
Single
Target Settings
Active Targets
Universal
Universal
Material
Unlit
Allow Material Override
Surface Type
Transparent
Blending Mode
Alpha
Render Face
Front
Depth Write
Auto
Depth Test
L Equal
Alpha Clipping
Cast Shadows
Supports LOD Cross Fade
Additional Motion Vectors
None
Alembic Motion Vectors
Custom Editor GUI
グラデーションで
色をつける
Sample Gradient
Gradient(G)
Time(1)
Out(4)
Vertex
Object Space
Position(3)
Normal(3)
Tangent(3)
Fragment
Base Color(3)
Alpha(1)
上の方を暗くする
（薄くする）
Split
In(4)
R(1)
G(1)
B(1)
A(1)
Multiply
A(1)
B(1)
Out(1)
UV
Out(4)
Channe UV0
Wave Number(1)
Frequency(1)
Add
Sine
In(1)
Amplitude(1)
Velocity(1)
Time
Time(1)
Sine Time(1)
Cosine Time(1)
Delta Time(1)
Smooth Delta(1)
Vector 2
X(1)
Y(1)
Out(2)
Subtract
A(2)
B(2)
Scale(1)
Gradient Noise
UV(2)
Hash Type
Determinist

ひとまずの完成

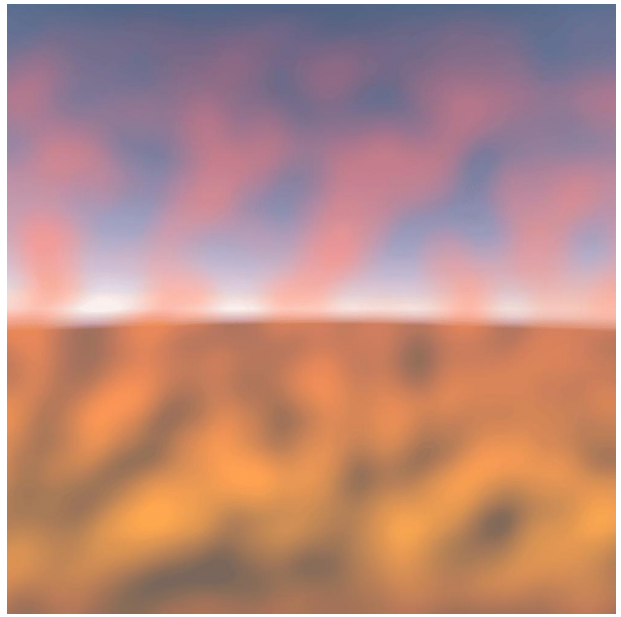
- パラメータを変化させて
  どのように変わるか感じよう
  - Scale
    - ノイズの細かさ
  - Velocity
    - 上昇速度
  - Amplitude
    - 揺らす大きさ
  - Wave Number
    - ゆらすサイン波の波数
  - Frequency
    - ゆらすサイン波の角速度

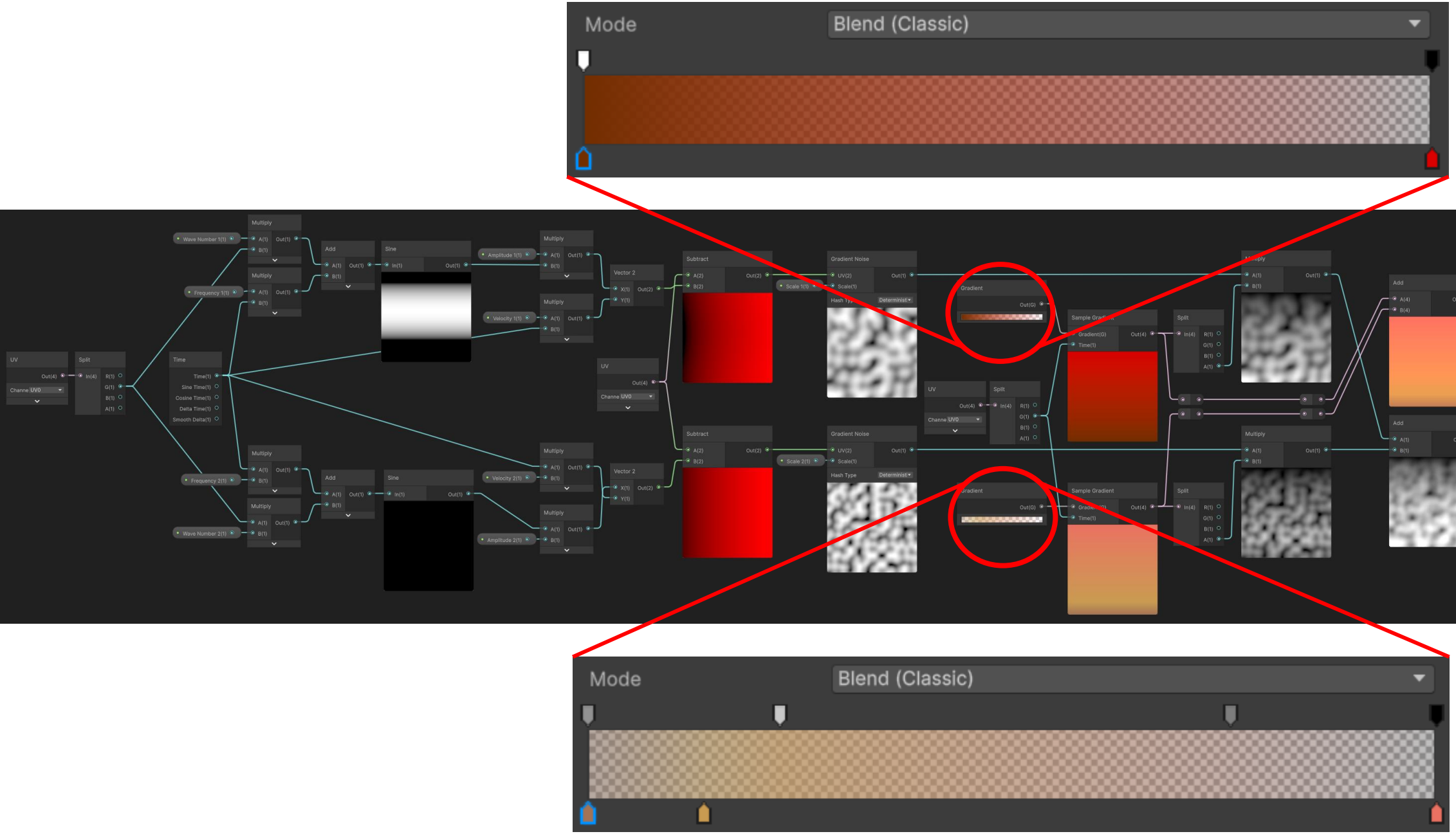


やってみよう：その4

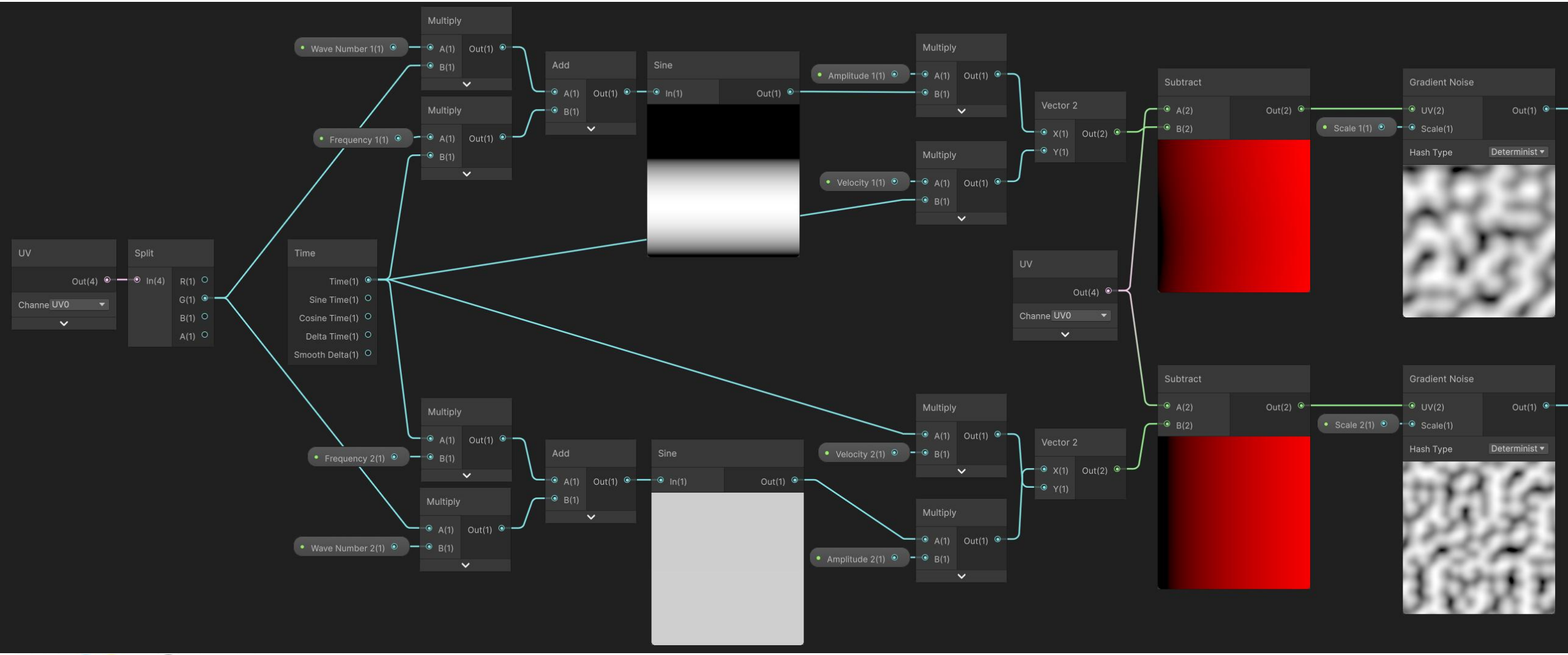
複数のノイズを合わせて見た目を複雑にする

- Shader Graphを作成
  - 「5 Fire/4 Fire Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を2セット追加(次は設定例)
    - Scale 1: 初期値: 7, Scale 2: 初期値: 10
      - ノイズの細かさ 範囲: [0, 100] Float型 (Mode: Slider)
    - Velocity 1: 初期値: 0.7, Velocity 2: 初期値: 2
      - 上昇速度 範囲: [0, 10] Float型 (Mode: Slider)
    - Amplitude 1:初期値: 0.03, Amplitude 2:初期値: 0.10
      - 揺らす大きさ 範囲: [0, 1] Float型 (Mode: Slider)
    - Wave Number 1: 初期値: 5, Wave Number 2:初期値: 0.01
      - ゆらすサイン波の波数 範囲: [0, 100] Float型 (Mode: Slider)
    - Frequency 1: 初期値: 5, Frequency 2: 初期値: 5
      - ゆらすサイン波の角速度 範囲: [0, 100] Float型 (Mode: Slider)



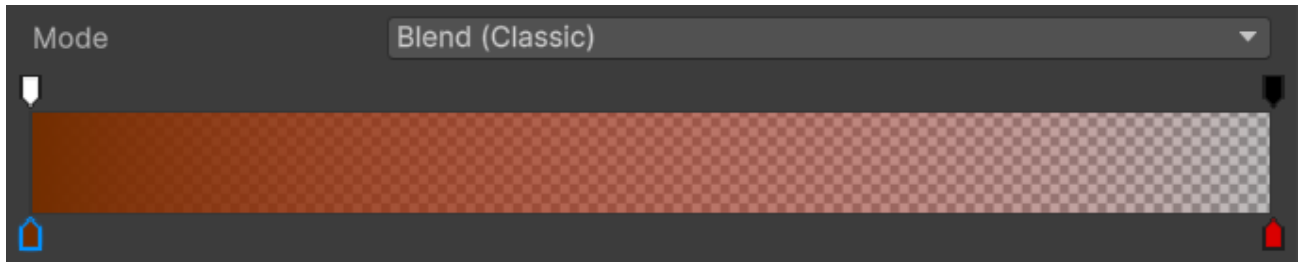

Mode
Blend (Classic)
Mode
Blend (Classic)
Vertex
Fragment
Gradient Noise
Sample Gradient
Gradient
Subtract
Multiply
Add
Sine
Split
Vector 2
Time
UV

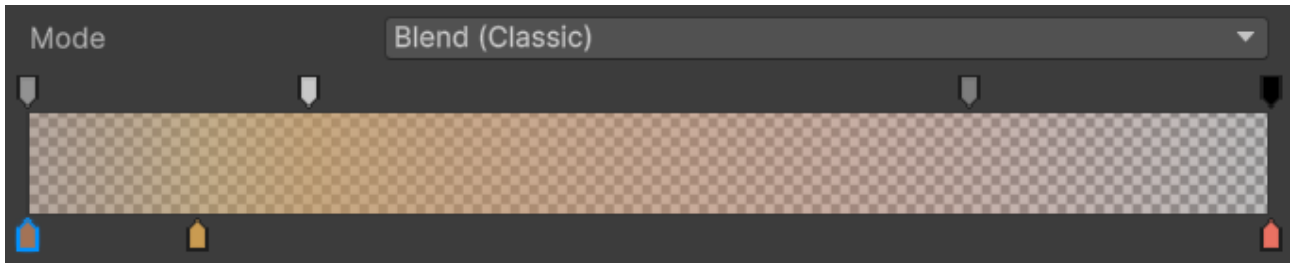
ノイズの生成

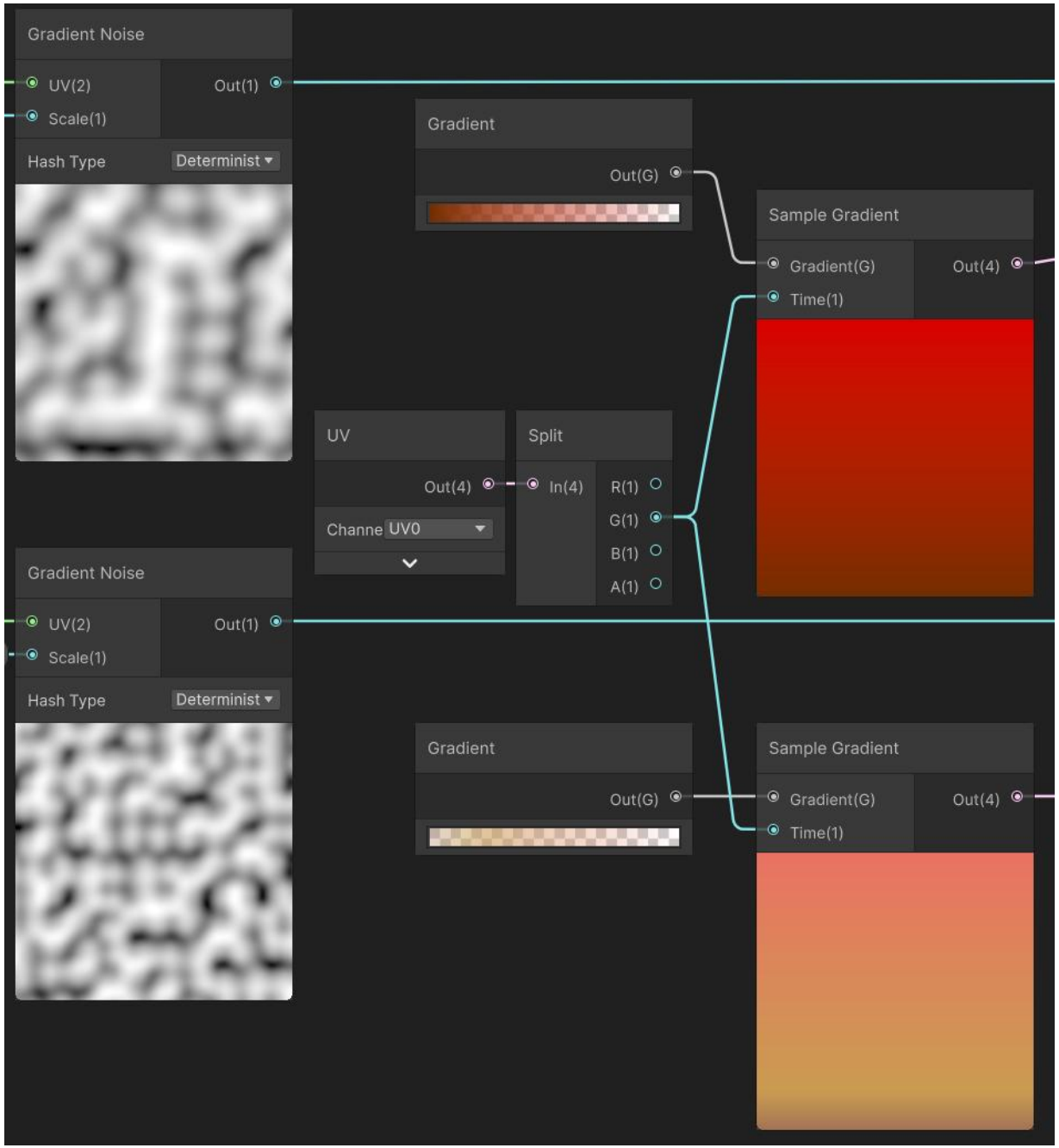

Multiply
Wave Number 1(1)
A(1)
B(1)
Out(1)
Add
Sine
In(1)
Amplitude 1(1)
Multiply
Vector 2
X(1)
Y(1)
Out(2)
Subtract
A(2)
B(2)
Gradient Noise
UV(2)
Scale(1)
Hash Type
Determinist
Scale 1(1)
Frequency 1(1)
Velocity 1(1)
UV
Out(4)
Channe UV0
Split
In(4)
R(1)
G(1)
B(1)
A(1)
Time
Time(1)
Sine Time(1)
Cosine Time(1)
Delta Time(1)
Smooth Delta(1)
Frequency 2(1)
Velocity 2(1)
Amplitude 2(1)
Wave Number 2(1)
Scale 2(1)

ノイズの色付け

- 異なるグラデーションの設定
  - 不透明度も差をつける

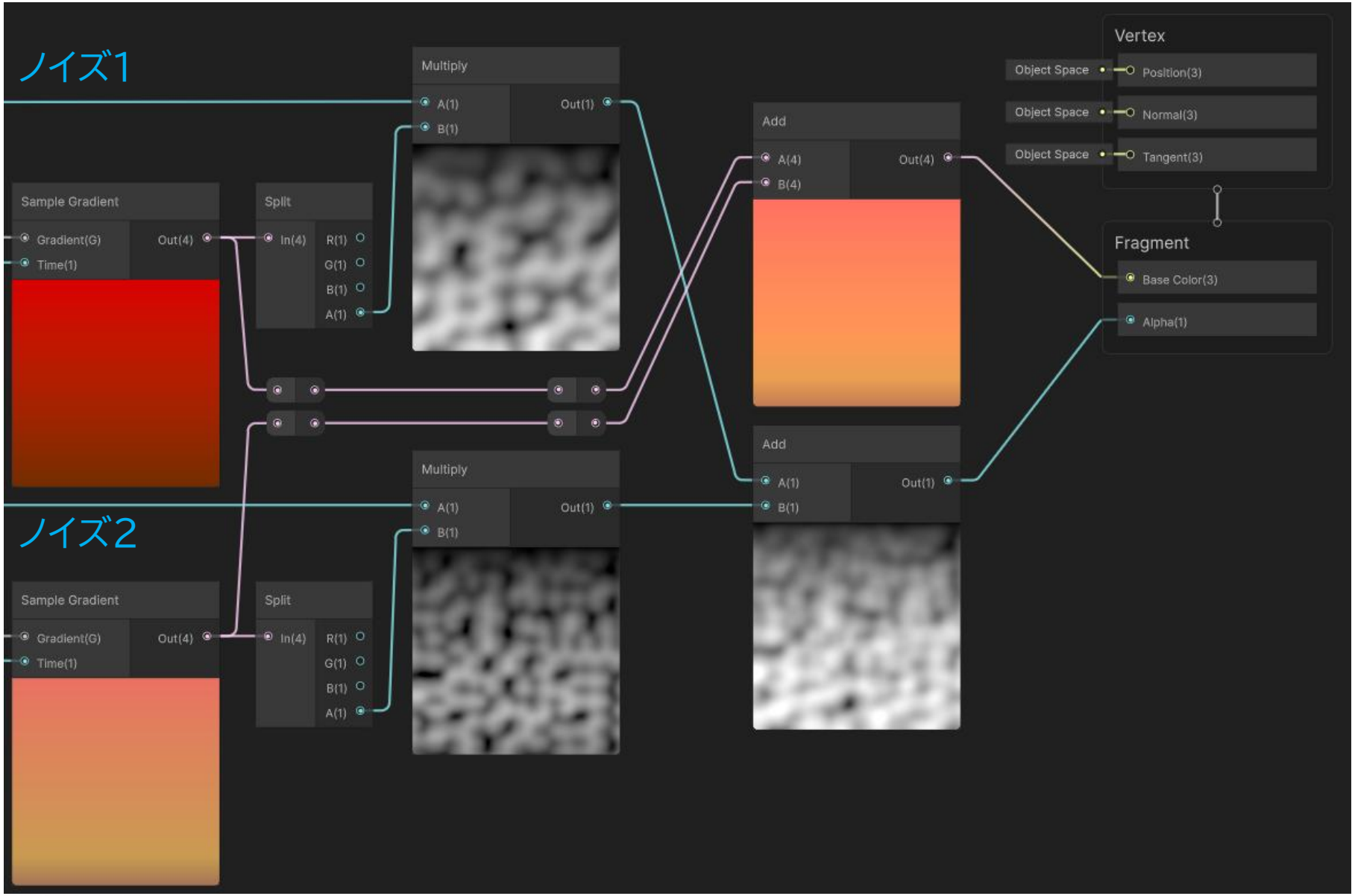




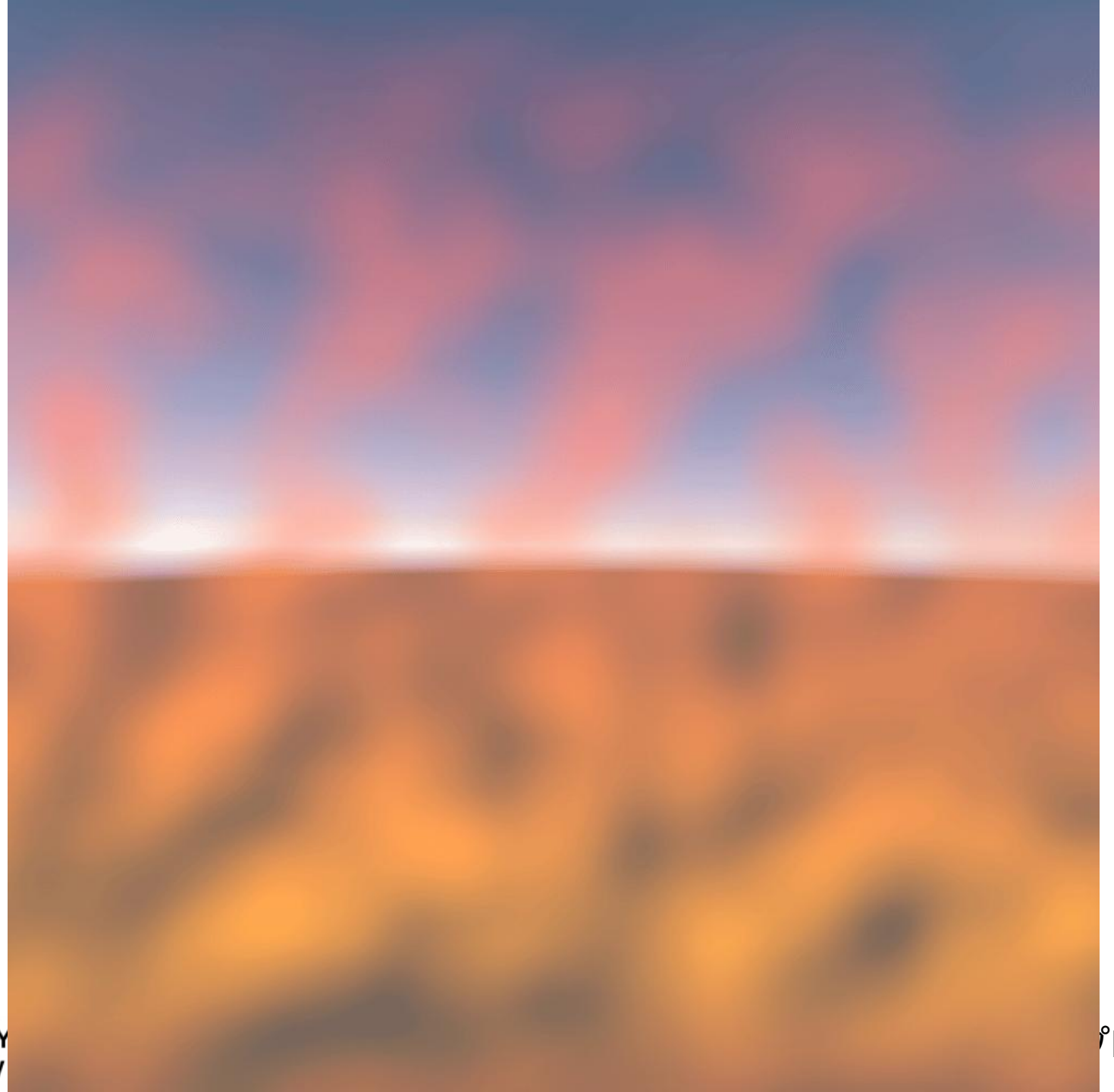


出力

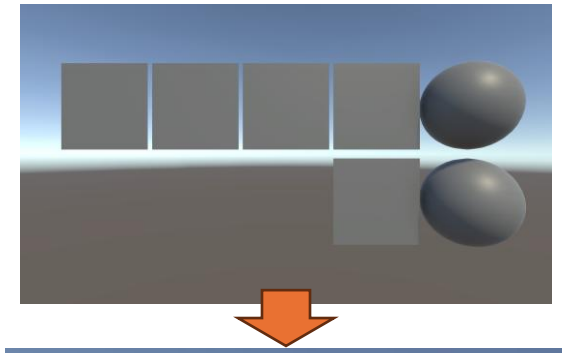
- 色とαを別々に加算して出力
- α成分だけノイズを乗せる



完成

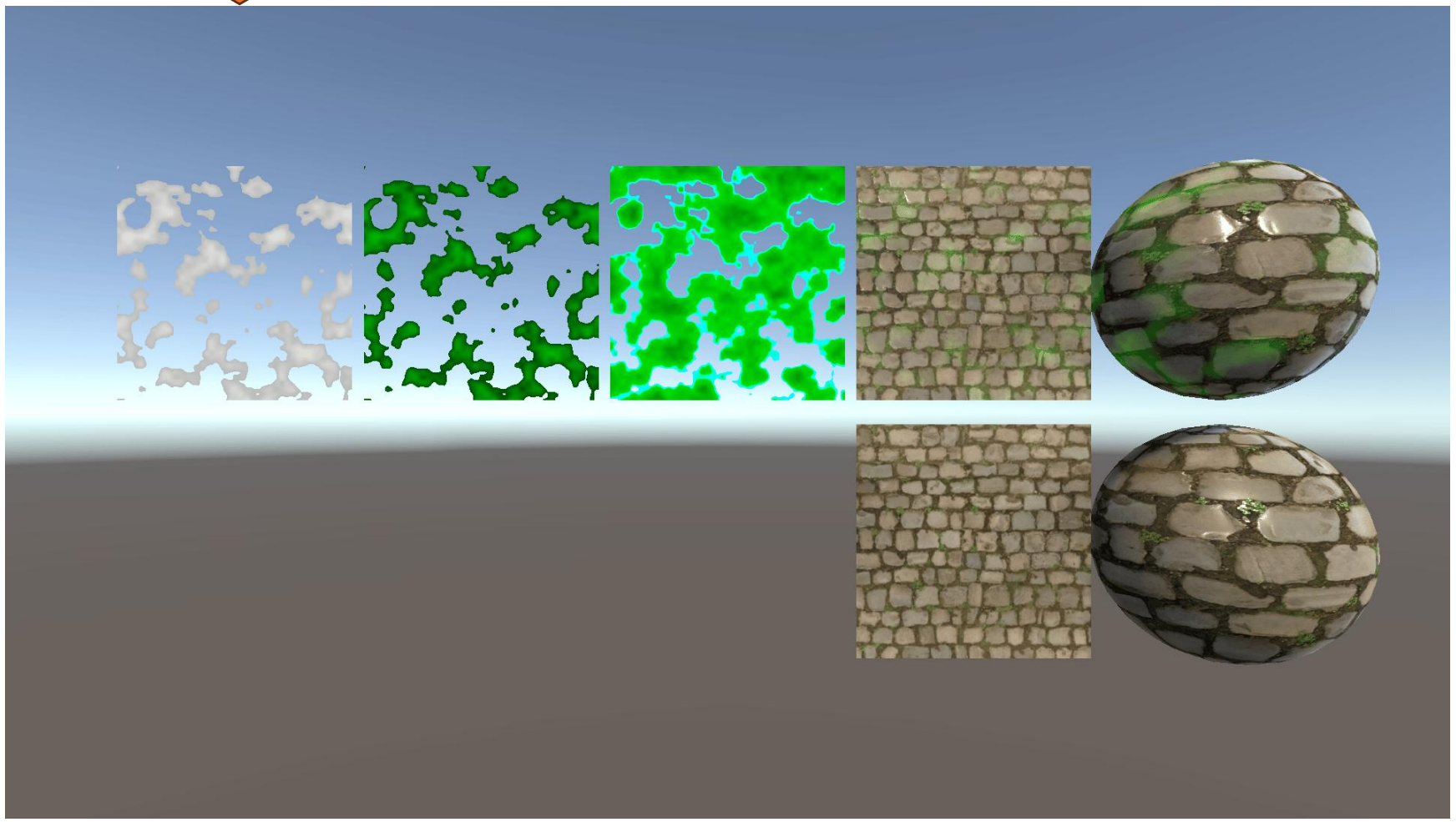


本日の内容



シーン：6 Dissolve Scene

- ノイズの概要
- 基本的なノイズ
- FBM
- 加工したFBM
- ボロノイ図
- 炎
- ディゾルブ



フェード

出典: フリー百科事典『ウィキペディア（Wikipedia）』

映像編集 [編集]

フェードは映像編集技術用語のひとつである。フェードイン（fade-in あるいは fade-up）とフェードアウト (fade-out) の2種類がある。

フェードインは「一色の状態から徐々に映像が見えている状態に移り変わること」であり、フェードアウトは「映像が見えている状態から徐々に一色に移り変わること」である。「一色の状態」は、古くは黒が多かったが、編集機材の発展に伴い黒縛りはなくなり、現在では「ホワイトフェード（白からのフェードイン、白へのフェードアウト）」なども多用されている。主として物語の展開の上でひと区切りつける必要がある場合に使われる。

2つの映像の片方をフェードアウト (FO) し同時に他方をフェードイン (FI) することで画面を切り替える手法を**クロスフェード・ディゾルブ (Dissolve) という。オーバーラップは、英語のDissolveに該当する和製造語であり、英語の専門用語ではあくまでDissolveという。**

フェードイン

フェードアウト

クロスフェード

moyasi エフェクト館

エフェクトに関係する記事上げてきます
エフェクト関係の記事の内容は緩く募集中
自分がわかる範囲であれば書いていこうかなと。
2018-06-08

【UE4】マテリアルでディゾルブエフェクト

Unreal Engine 4　マテリアル制作

Verは4.18です

こういうオパシティマスクと、発光を組み合わせた様なものを、ディゾルブエフェクトというらしいです。

プロフィール

tktk(たかつき) (id:tktknkyo)
主にエフェクト関係の記事を上げていこうかと思います。
✚読者になる 79

検索

記事を検索

最新記事

unity Asset Store | dissolve

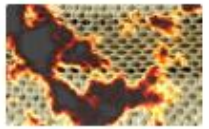
Assets | Tools | Services | By Unity | Industries

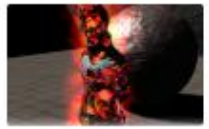
1-41 of 41 results for dissolve Sort by Relevance View Results 48

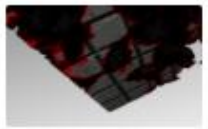
dissolve ×


DAVIT NASKIDASH...
Advanced Dissolve
★★★★★ (43)
$20

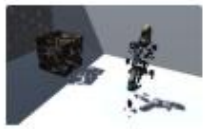
DIMENSION FIVE
Beautiful Dissolves
★★★★★ (50)
$15

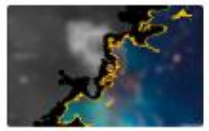

MOONFLOWER CA...
Dissolve Edge
★★★★★ (25)
FREE


3Y3NET
Dissolve effect
★★★★☆ (18)
$10

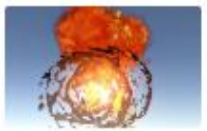

STEFAN SIGL
Dissolve Shader
★★★★★ (30)
$7


TI GAMES
Dissolve PRO
★★★★★ (5)
$25


SWORD-MASTER
SwordMaster Dissolv...
(not enough ratings)
$8


JIM
Dissolve Burning Ima...
(not enough ratings)
$4.99


RAFAEL MELO
Simple Dissolve Sha...
★★★★★ (10)
FREE


MOONFLOWER CA...
Particle Dissolve Sha...
★★★★★ (35)
FREE


CICONIA STUDIO
Sci-Fi - Dissolve Shad...
★★★★☆ (4)
$9.99


ANDREI MELUTA
Dissolve Shaders Pa...
★★★★☆ (32)
$10

ANDREI MELUTA
Dissolve Shaders Mo...
★★★★★ (10)
$5

DENNIS MAIER
Dissolve Texture Pac...
★★★★☆ (11)
FREE

DENNIS MAIER
Dissolve Shader Bun...
(not enough ratings)
$7


WABO
Holographic Dissolve...
★★★★☆ (13)
$5

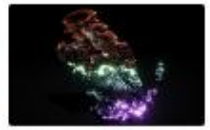
GEMMINE MEDIA
Gradient, Transition a...
★★★★★ (3)
$25

GLOOMY STUDIO
Knight with Sword & ...
(not enough ratings)
$4.99

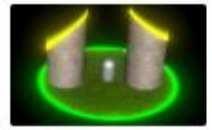

STUDIO DCT
Stylized Firing VFX
(not enough ratings)
$5.90

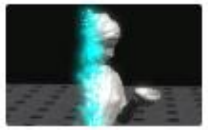

MARTIN REINTGES
Hologram Shader Pa...
★★★★★ (10)
$15

ECHOLOGIN
ShaderOne [BETA]
★★★★★ (14)
$40


OINE
Materialize FX
★★★★☆ (16)
$9

ENIXION
Hyper Card
★★★★★ (13)
$45


OO.D.Y
Amazing World Fading
(not enough ratings)
$9.90


OO.D.Y
Hyperspace Teleport
(not enough ratings)
$15

ALLASSTAR
Special FX Shaders
★★★★☆ (5)
$4.99


DIAMOND FOX
Particle effects 1
★★★★★ (17)
$9.99


MOONFLOWER CA...
Disperse Pixels
★★★★☆ (9)
$20


FORGE3D
Sci-Fi Effects
★★★★★ (357)
$29.99

GLOOMY STUDIO
RPG Character Pack ...
(not enough ratings)
$12

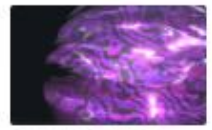
3Y3NET
Super Effects Pack
★★★☆☆ (3)
$20

DIGITAL SALMON
FADE | ScreenFX
★★★★☆ (4)
$25

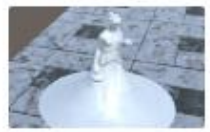
SIBERIAN PRO
Obliterate Effect Pack
(not enough ratings)
$12


CICONIA STUDIO
Sci-Fi Shaders Bundle
★★★★★ (3)
$19.98


CICONIA STUDIO
Sci-Fi - Hologram Sh...
(not enough ratings)
$9.99


CICONIA STUDIO
Sci-Fi - Force Field Sh...
(not enough ratings)
$9.99


SZPRO
Fragmentation Shader
(not enough ratings)
$5


OO.D.Y
Melt Your Mesh
(not enough ratings)
$12


VOROS GERGELY
2D shader Technics (...
★★★★★ (7)
FREE

GOLDMAN STUDIO
Aura Materials
(not enough ratings)
$4.99

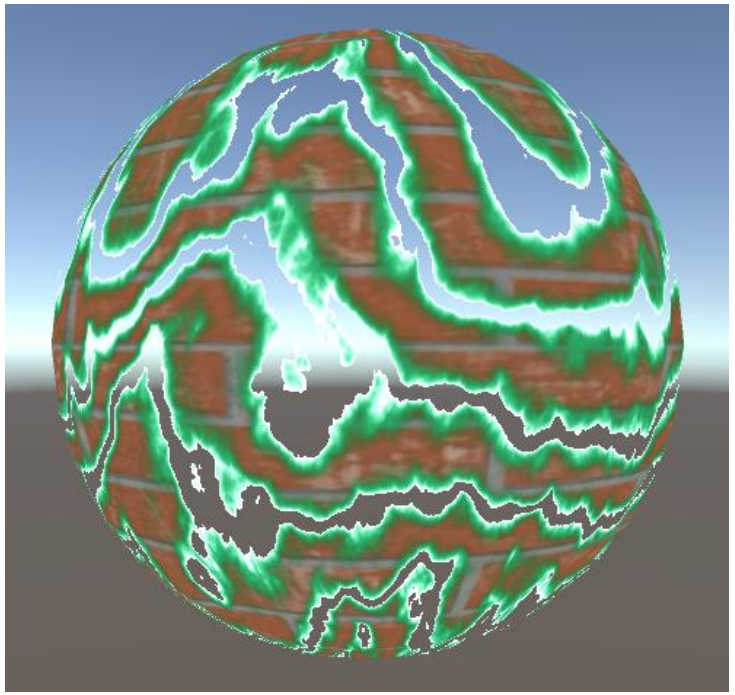
TI GAMES
TI Shader Bundle
★★★★★ (11)
$70

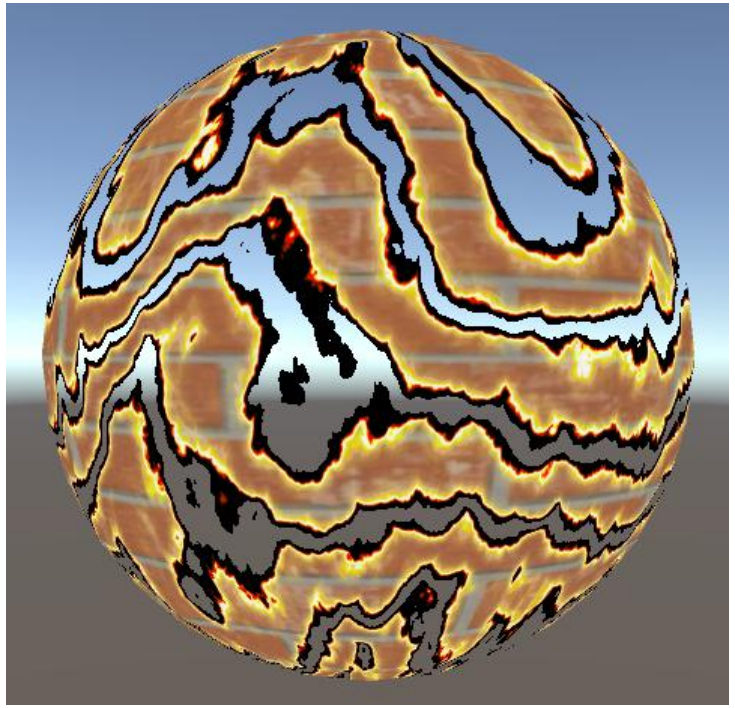
KOGE

ークショップIV

今回やること

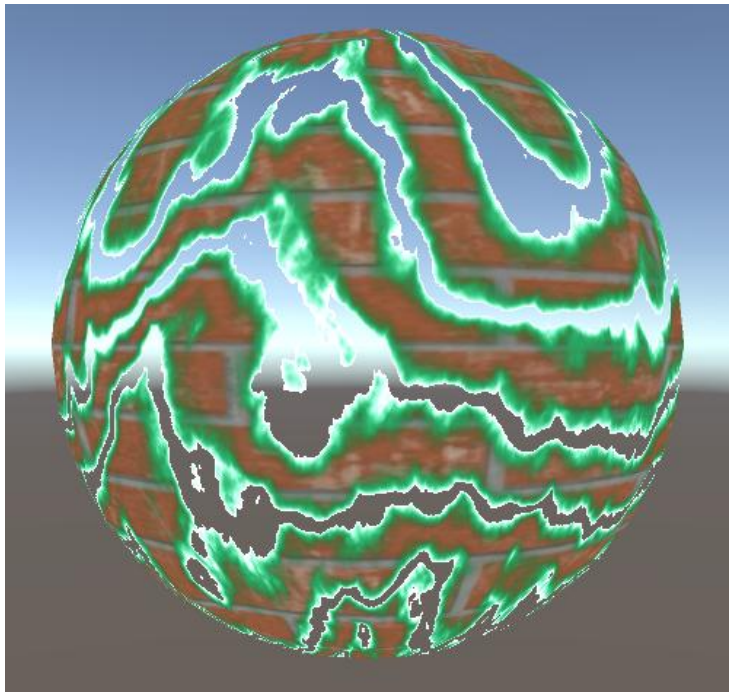
- ノイズ関数を使って、ディゾルブを実現しよう
  1. ノイズによるαクリップ
  2. 色をつける
  3. 消える縁を明るくする
  4. 物体が消えていく
  5. 別パターンを作成

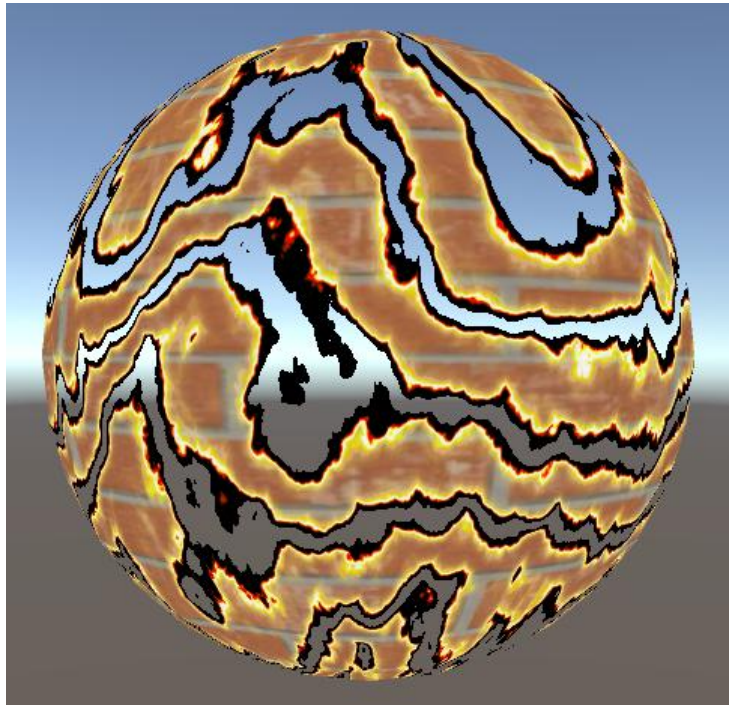




今回やること

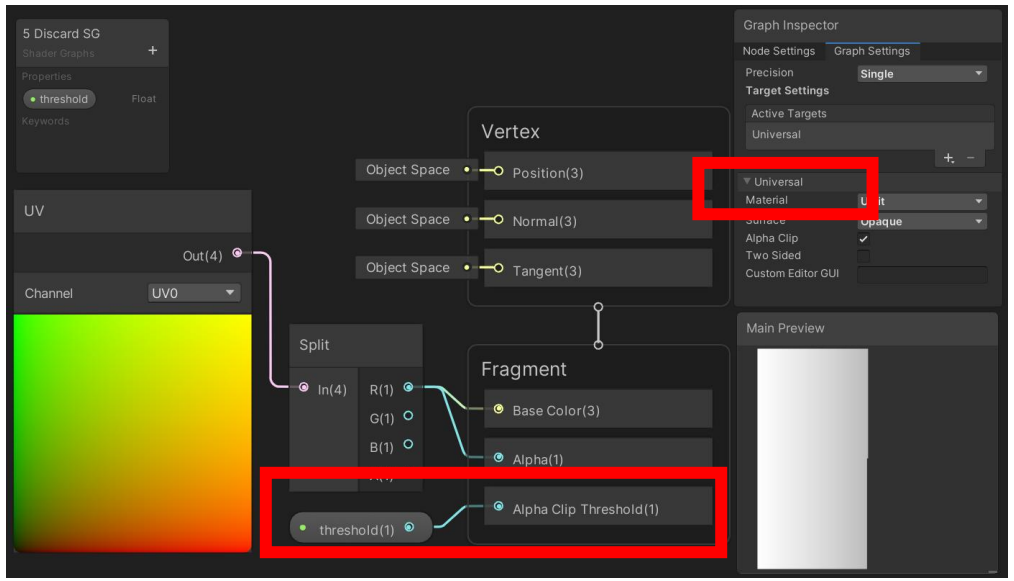
- ノイズ関数を使って、ディゾルブを実現しよう
  1. ノイズによるαクリップ
  2. 色をつける
  3. 消える縁を明るくする
  4. 物体が消えていく
  5. 別パターンを作成



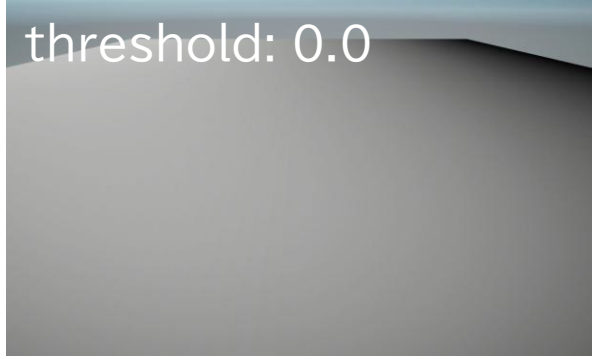


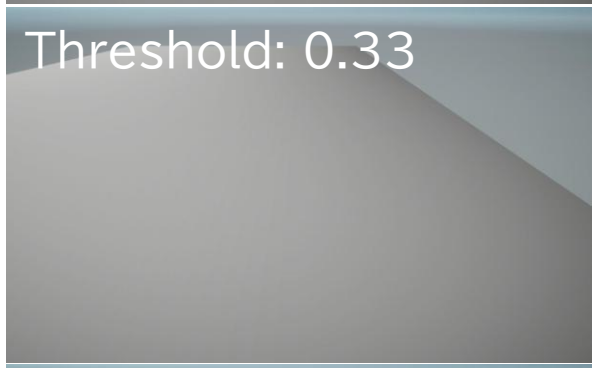
Alpha Clip

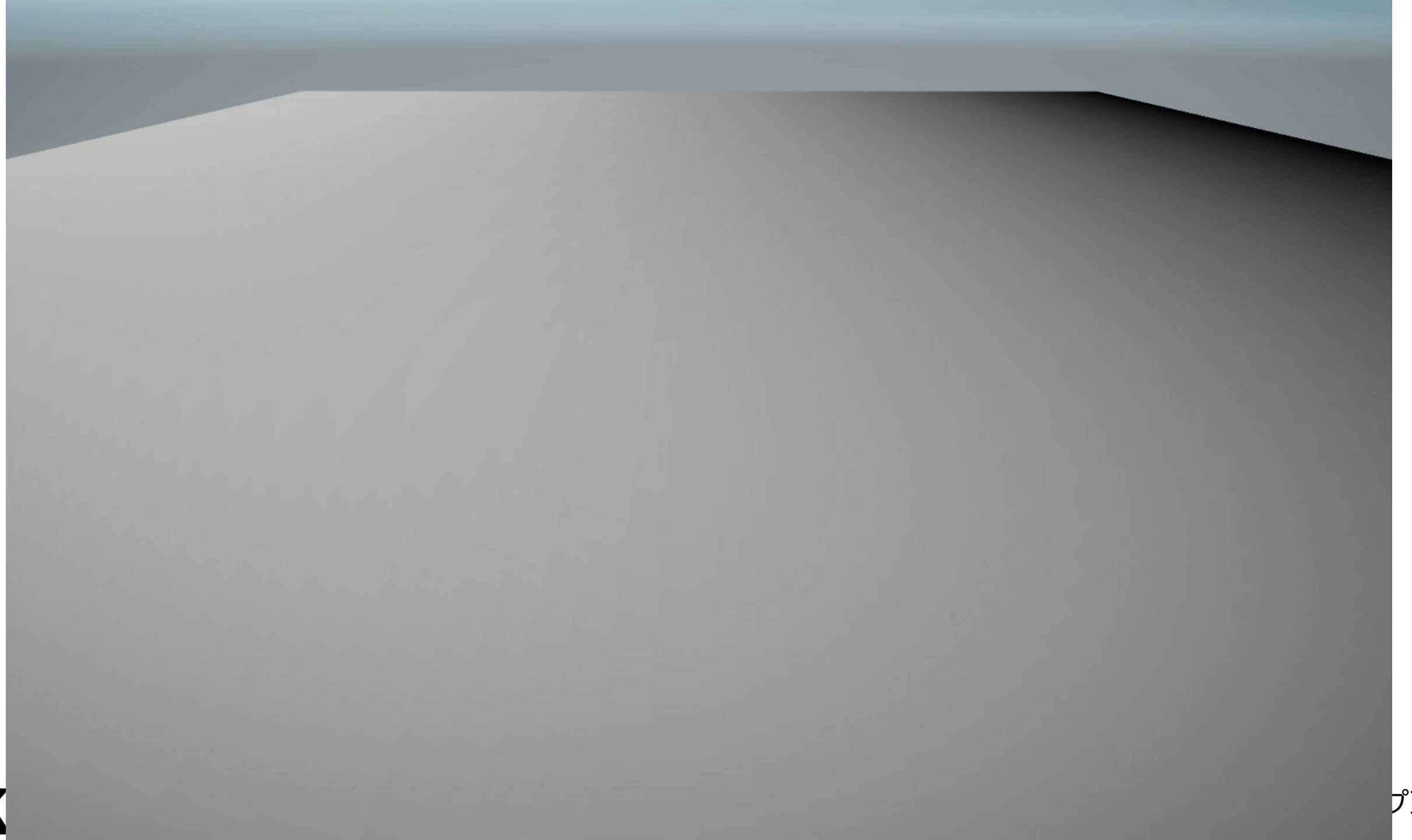
- フラグメントの破棄
  - 早期Zカリングが無効になるので、パフォーマンスは悪い
  - Aブレンディングの不透明で合成するよりはまし
- パラメータの値を変えることで、少しづつ削ることができる
  - 例えばテクスチャ座標値



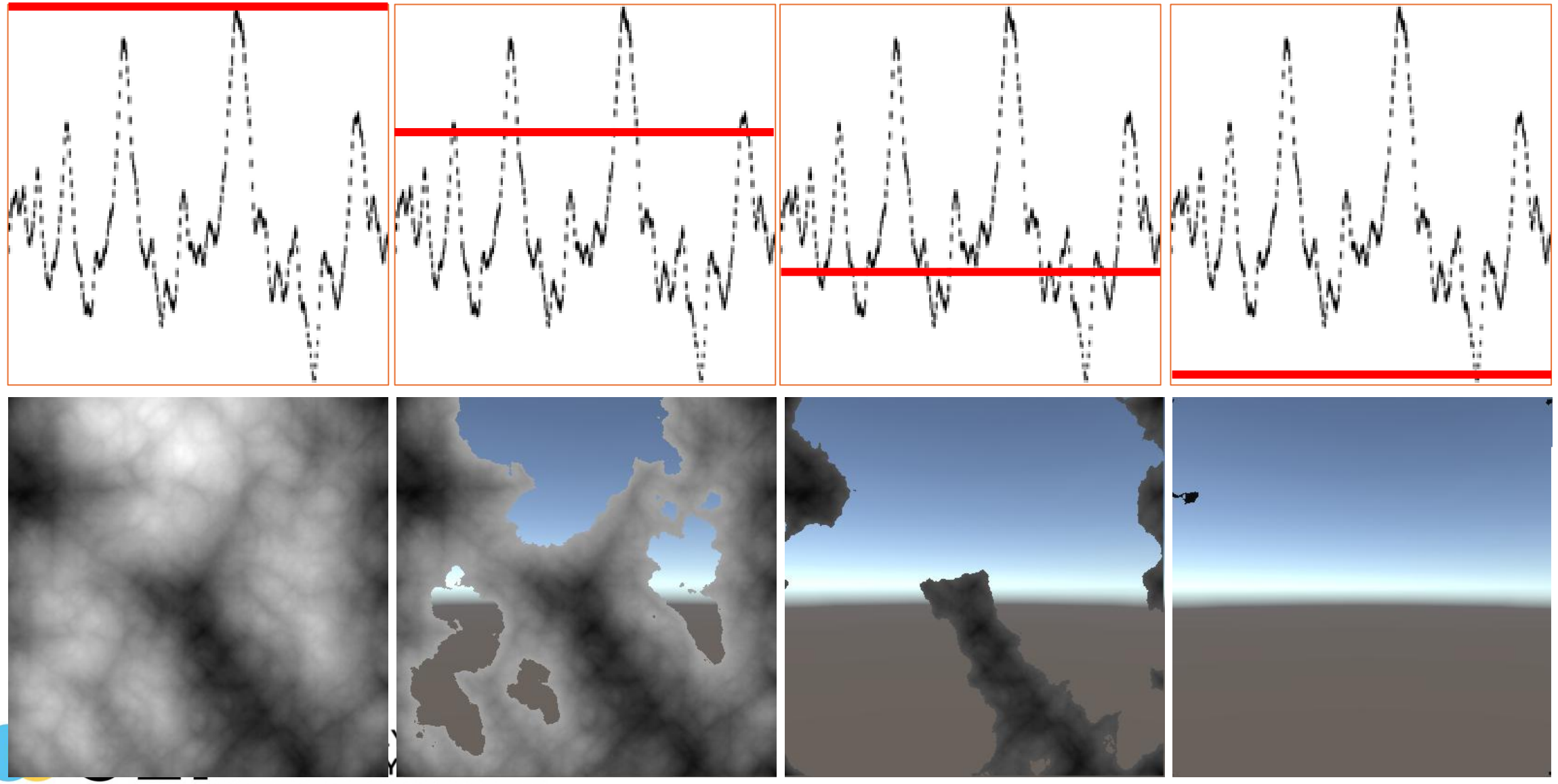
5 Alpha Clip Scene



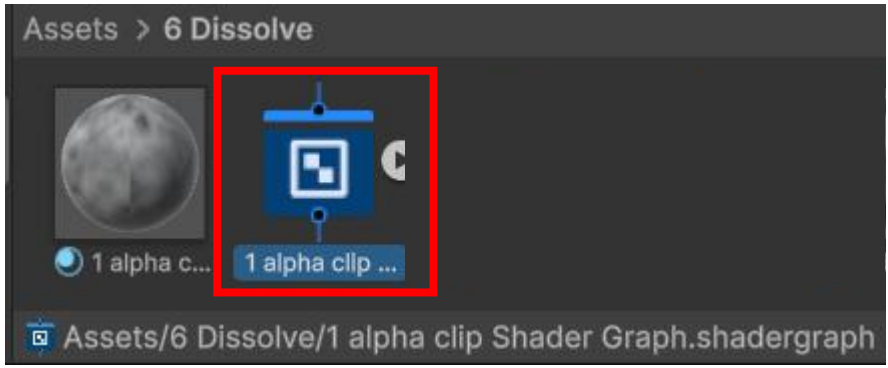




ノイズによるAlpha Clip

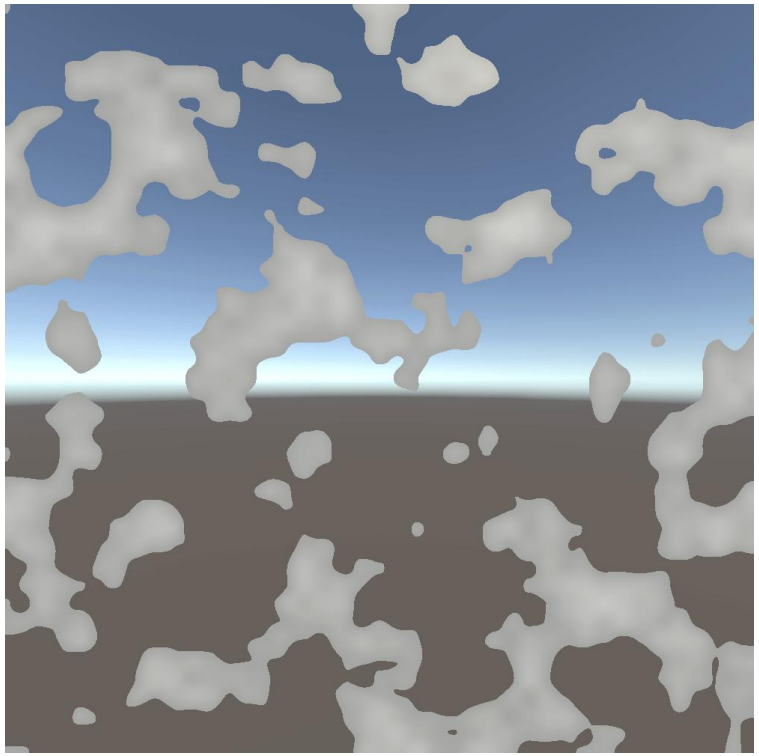


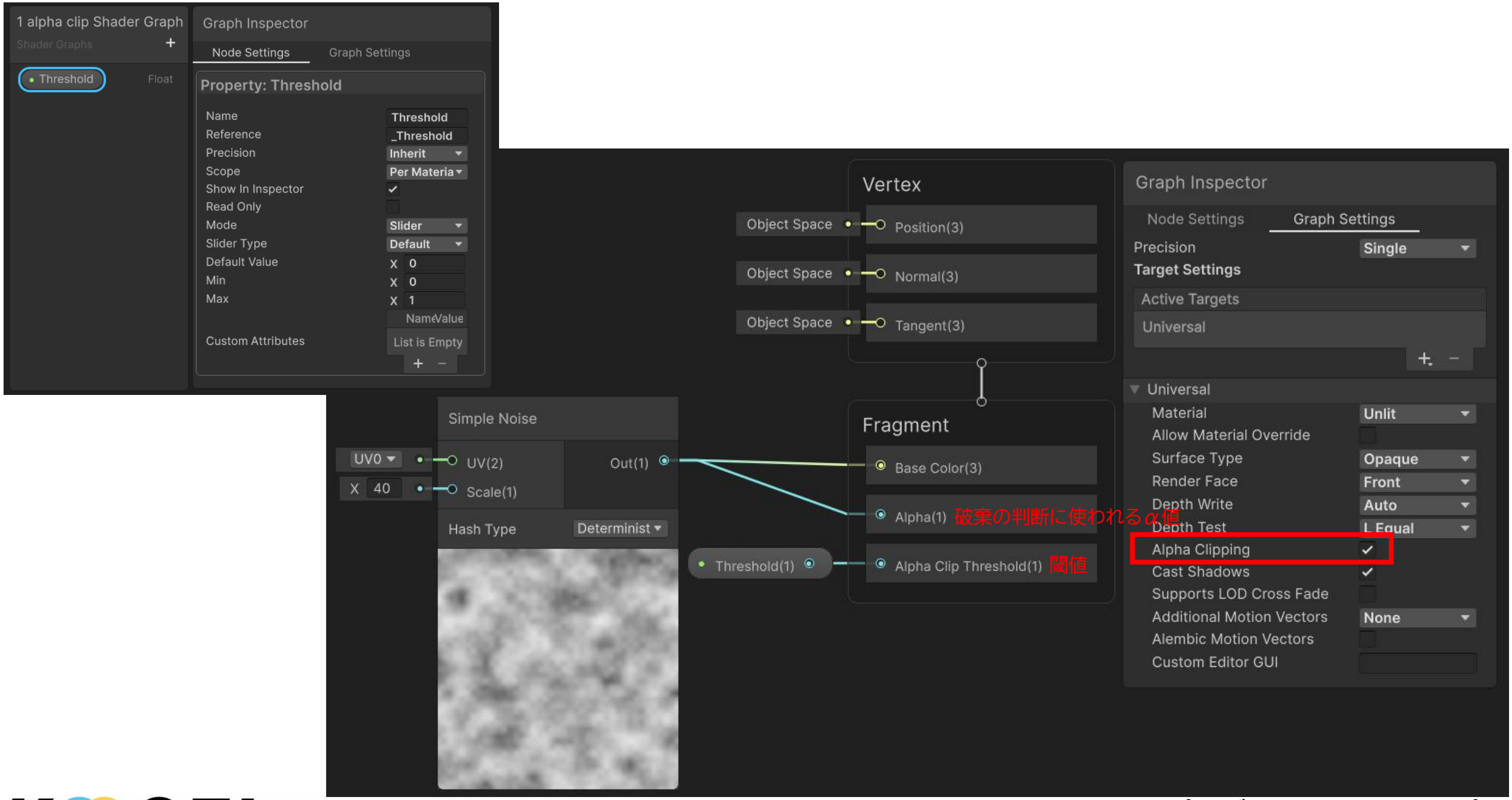
やってみよう：その1



ノイズによるαクリップ

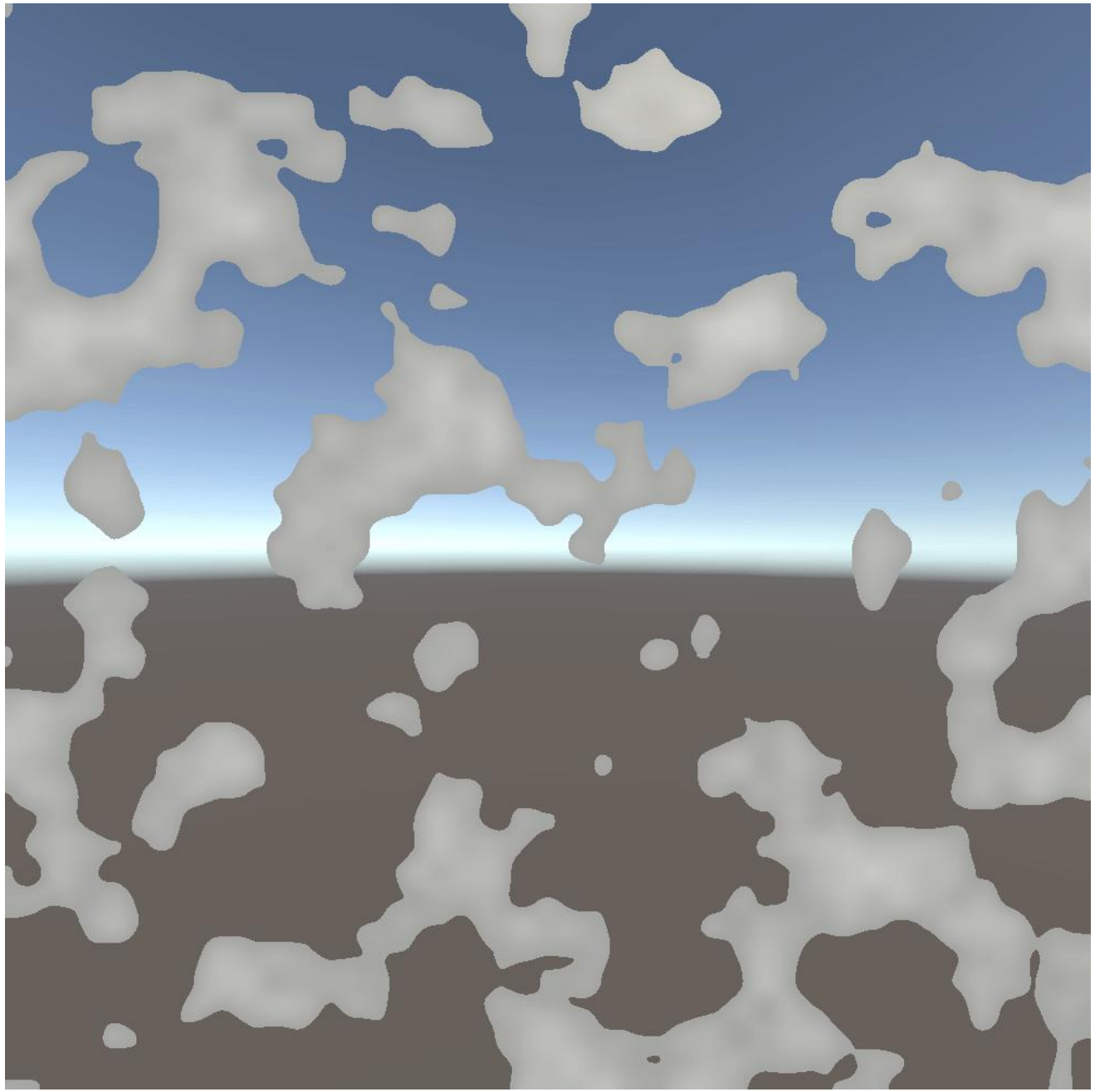
- Shader Graphを作成
  - 「6 Dissolve/1 alpha clip Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を1つ追加(次は設定例)
    - Threshold: 破棄される閾値
      - Float型 (Mode: Slider)
      - 範囲: [0, 1]
      - 初期値: 0




1 alpha clip Shader Graph
Shader Graphs
Threshold
Float
Graph Inspector
Node Settings
Graph Settings
Property: Threshold
Name
Threshold
Reference
_Threshold
Precision
Inherit
Scope
Per Materia
Show In Inspector
Read Only
Mode
Slider
Slider Type
Default
Default Value
X 0
Min
X 0
Max
X 1
Name
Value
Custom Attributes
List is Empty
Vertex
Object Space
Position(3)
Object Space
Normal(3)
Object Space
Tangent(3)
Graph Inspector
Node Settings
Graph Settings
Precision
Single
Target Settings
Active Targets
Universal
Universal
Material
Unlit
Allow Material Override
Surface Type
Opaque
Render Face
Front
Depth Write
Auto
Depth Test
L Equal
Alpha Clipping
Cast Shadows
Supports LOD Cross Fade
Additional Motion Vectors
None
Alembic Motion Vectors
Custom Editor GUI
Simple Noise
UV0
UV(2)
Out(1)
X 40
Scale(1)
Hash Type
Deterministic
Fragment
Base Color(3)
Alpha(1)
破棄の判断に使われるα値
Threshold(1)
Alpha Clip Threshold(1)
閾値

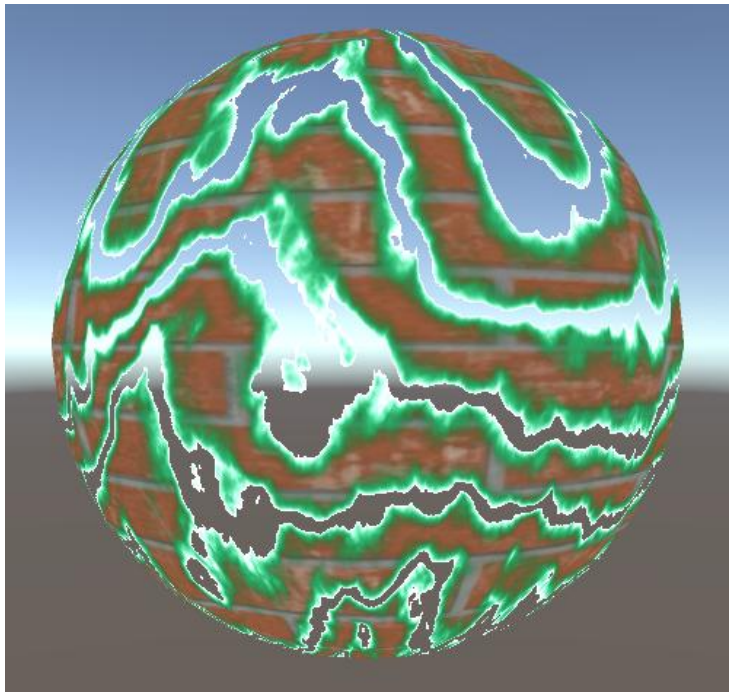
ここまでの状態

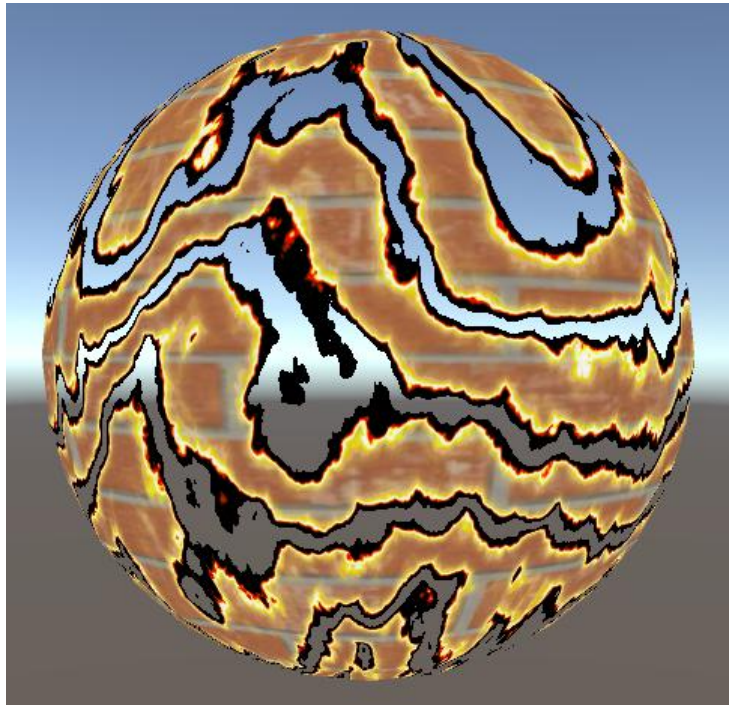
- マテリアルのスライダーを操作して観察しよう



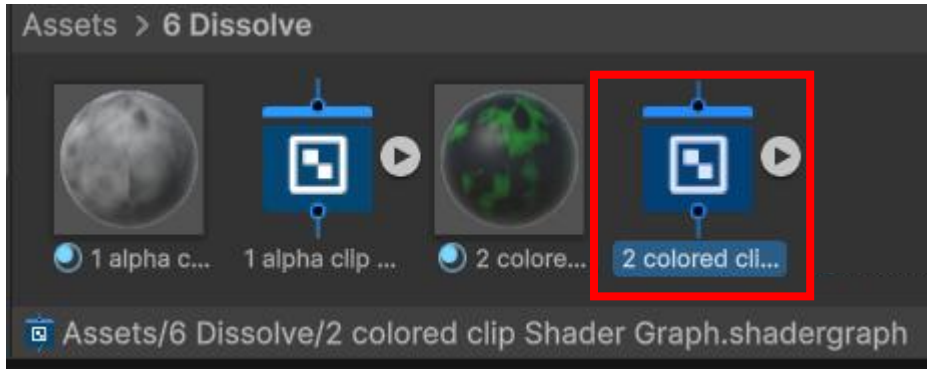
今回やること

- ノイズ関数を使って、ディゾルブを実現しよう
  1. ノイズによるαクリップ
  2. 色をつける
  3. 消える縁を明るくする
  4. 物体が消えていく
  5. 別パターンを作成



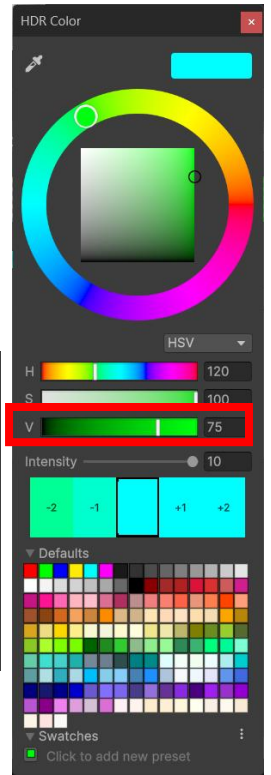


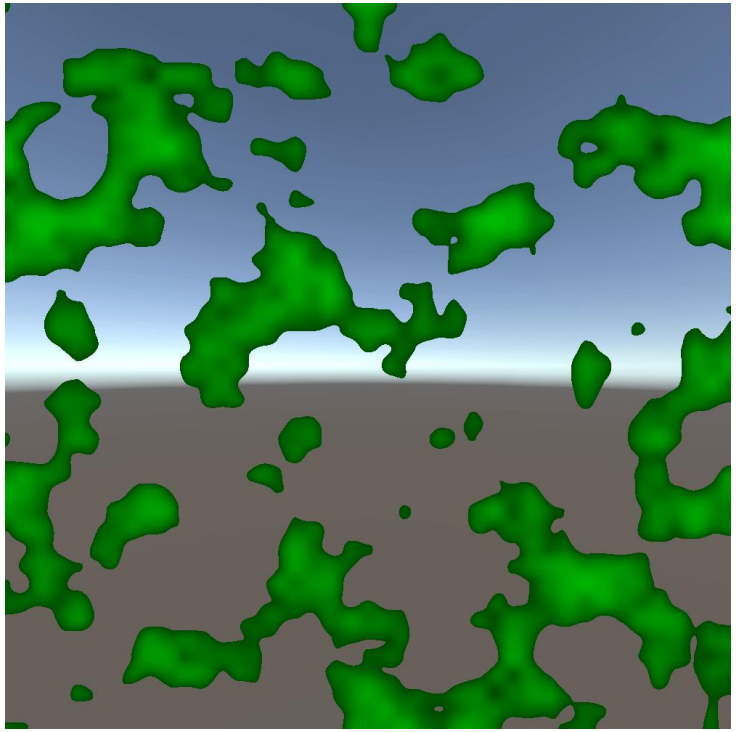
やってみよう:その2

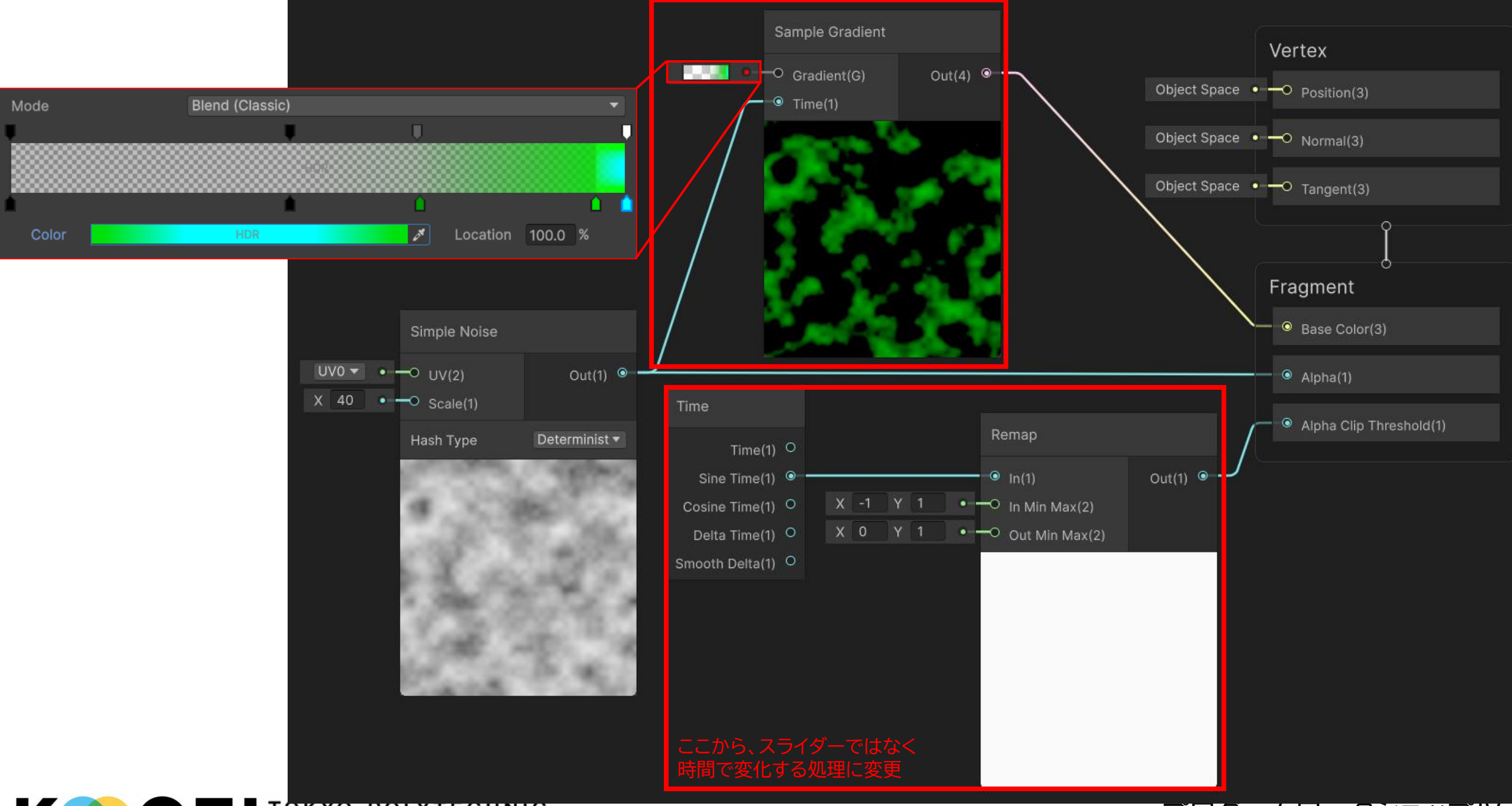


色をつける

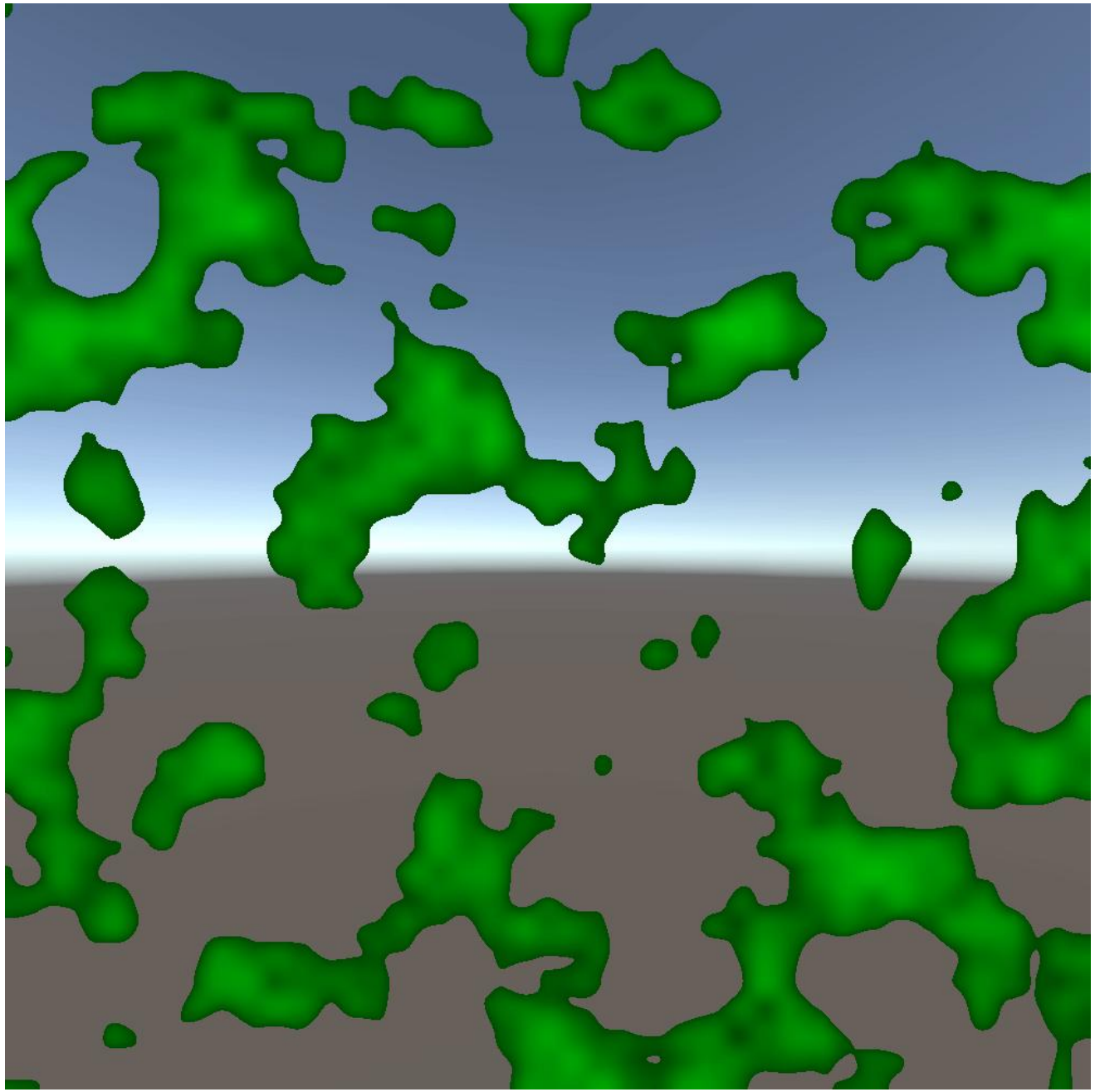
- Shader Graphを作成
  - 「6 Dissolve/2 colored clip Material」に設定
- Shader Graphを実装
  （ノード構成は次頁）
  - グラデーションを導入
    - HDRを使うと、強い光も設定可能



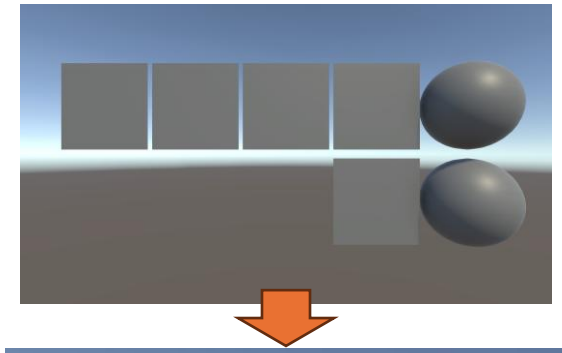



Mode
Blend (Classic)
Color
HDR
Location
100.0
%
Sample Gradient
Gradient(G)
Time(1)
Out(4)
Vertex
Object Space
Position(3)
Object Space
Normal(3)
Object Space
Tangent(3)
Fragment
Base Color(3)
Alpha(1)
Alpha Clip Threshold(1)
Simple Noise
UV0
UV(2)
X
40
Scale(1)
Out(1)
Hash Type
Deterministic
Time
Time(1)
Sine Time(1)
Cosine Time(1)
Delta Time(1)
Smooth Delta(1)
Remap
In(1)
Out(1)
X
-1
Y
1
In Min Max(2)
X
0
Y
1
Out Min Max(2)
ここから、スライダーではなく
時間で変化する処理に変更

ここまでの状態

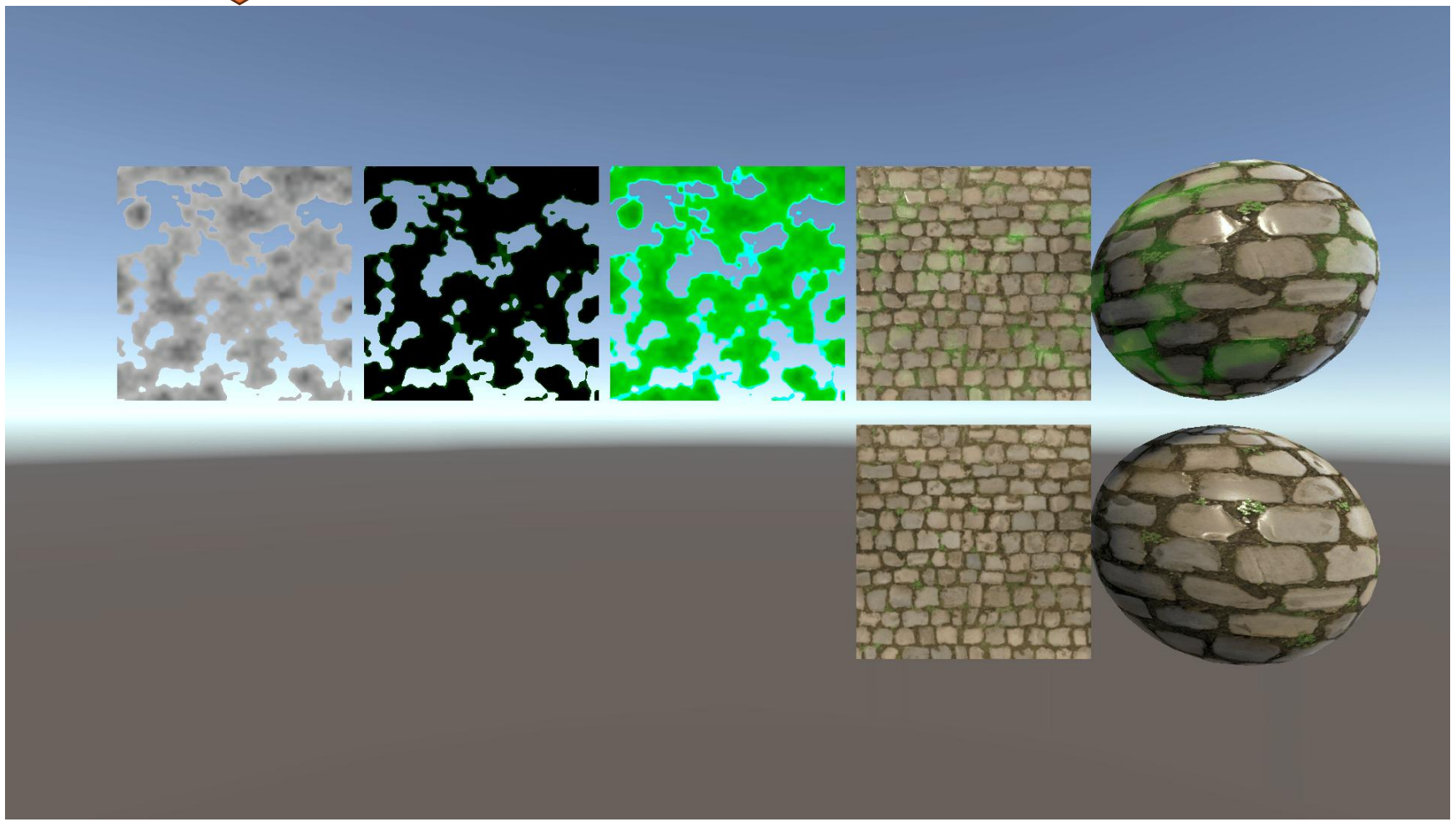


本日の内容



シーン：6 Dissolve Scene

- ノイズの概要
- 基本的なノイズ
- FBM
- 加工したFBM
- ボロノイ図
- 炎
- ディゾルブ



フェード

出典: フリー百科事典『ウィキペディア（Wikipedia）』

映像編集 [編集]

フェードは映像編集技術用語のひとつである。フェードイン（fade-in あるいは fade-up）とフェードアウト (fade-out) の2種類がある。

フェードインは「一色の状態から徐々に映像が見えている状態に移り変わること」であり、フェードアウトは「映像が見えている状態から徐々に一色に移り変わること」である。「一色の状態」は、古くは黒が多かったが、編集機材の発展に伴い黒縛りはなくなり、現在では「ホワイトフェード（白からのフェードイン、白へのフェードアウト）」なども多用されている。主として物語の展開の上でひと区切りつける必要がある場合に使われる。

2つの映像の片方をフェードアウト (FO) し同時に他方をフェードイン (FI) することで画面を切り替える手法を**クロスフェード・ディゾルブ (Dissolve) という。オーバーラップは、英語のDissolveに該当する和製造語であり、英語の専門用語ではあくまでDissolveという。**

フェードイン

フェードアウト

クロスフェード

moyasi エフェクト館

エフェクトに関係する記事上げてきます
エフェクト関係の記事の内容は緩く募集中
自分がわかる範囲であれば書いていこうかなと。
2018-06-08

【UE4】マテリアルでディゾルブエフェクト

Unreal Engine 4　マテリアル制作

Verは4.18です

こういうオパシティマスクと、発光を組み合わせた様なものを、ディゾルブエフェクトというらしいです。

プロフィール

tktk(たかつき) (id:tktknkyo)
主にエフェクト関係の記事を上げていこうかと思います。
✚読者になる 79

検索

記事を検索

最新記事

unity Asset Store | dissolve

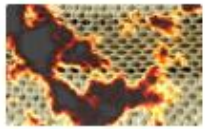
Assets | Tools | Services | By Unity | Industries

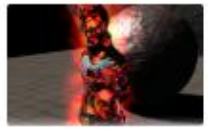
1-41 of 41 results for dissolve Sort by Relevance View Results 48

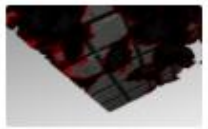
dissolve ×


DAVIT NASKIDASH...
Advanced Dissolve
★★★★★ (43)
$20

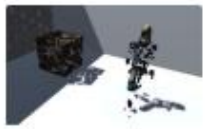
DIMENSION FIVE
Beautiful Dissolves
★★★★★ (50)
$15

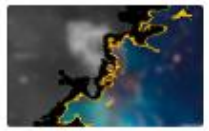

MOONFLOWER CA...
Dissolve Edge
★★★★★ (25)
FREE


3Y3NET
Dissolve effect
★★★★☆ (18)
$10

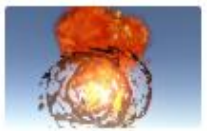

STEFAN SIGL
Dissolve Shader
★★★★★ (30)
$7


TI GAMES
Dissolve PRO
★★★★★ (5)
$25


SWORD-MASTER
SwordMaster Dissolv...
(not enough ratings)
$8


JIM
Dissolve Burning Ima...
(not enough ratings)
$4.99


RAFAEL MELO
Simple Dissolve Sha...
★★★★★ (10)
FREE


MOONFLOWER CA...
Particle Dissolve Sha...
★★★★★ (35)
FREE


CICONIA STUDIO
Sci-Fi - Dissolve Shad...
★★★★☆ (4)
$9.99


ANDREI MELUTA
Dissolve Shaders Pa...
★★★★☆ (32)
$10

ANDREI MELUTA
Dissolve Shaders Mo...
★★★★★ (10)
$5

DENNIS MAIER
Dissolve Texture Pac...
★★★★☆ (11)
FREE

DENNIS MAIER
Dissolve Shader Bun...
(not enough ratings)
$7


WABO
Holographic Dissolve...
★★★★☆ (13)
$5

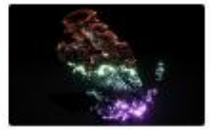
GEMMINE MEDIA
Gradient, Transition a...
★★★★★ (3)
$25

GLOOMY STUDIO
Knight with Sword & ...
(not enough ratings)
$4.99

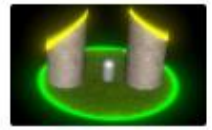

STUDIO DCT
Stylized Firing VFX
(not enough ratings)
$5.90

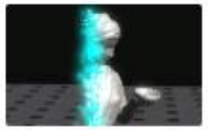

MARTIN REINTGES
Hologram Shader Pa...
★★★★★ (10)
$15

ECHOLOGIN
ShaderOne [BETA]
★★★★★ (14)
$40


OINE
Materialize FX
★★★★☆ (16)
$9

ENIXION
Hyper Card
★★★★★ (13)
$45


OO.D.Y
Amazing World Fading
(not enough ratings)
$9.90


OO.D.Y
Hyperspace Teleport
(not enough ratings)
$15

ALLASSTAR
Special FX Shaders
★★★★☆ (5)
$4.99


DIAMOND FOX
Particle effects 1
★★★★★ (17)
$9.99


MOONFLOWER CA...
Disperse Pixels
★★★★☆ (9)
$20


FORGE3D
Sci-Fi Effects
★★★★★ (357)
$29.99

GLOOMY STUDIO
RPG Character Pack ...
(not enough ratings)
$12

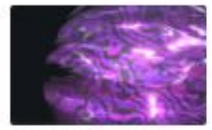
3Y3NET
Super Effects Pack
★★★☆☆ (3)
$20

DIGITAL SALMON
FADE | ScreenFX
★★★★☆ (4)
$25

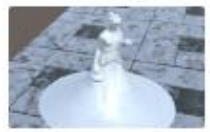
SIBERIAN PRO
Obliterate Effect Pack
(not enough ratings)
$12


CICONIA STUDIO
Sci-Fi Shaders Bundle
★★★★★ (3)
$19.98


CICONIA STUDIO
Sci-Fi - Hologram Sh...
(not enough ratings)
$9.99


CICONIA STUDIO
Sci-Fi - Force Field Sh...
(not enough ratings)
$9.99


SZPRO
Fragmentation Shader
(not enough ratings)
$5


OO.D.Y
Melt Your Mesh
(not enough ratings)
$12


VOROS GERGELY
2D shader Technics (...
★★★★★ (7)
FREE

GOLDMAN STUDIO
Aura Materials
(not enough ratings)
$4.99

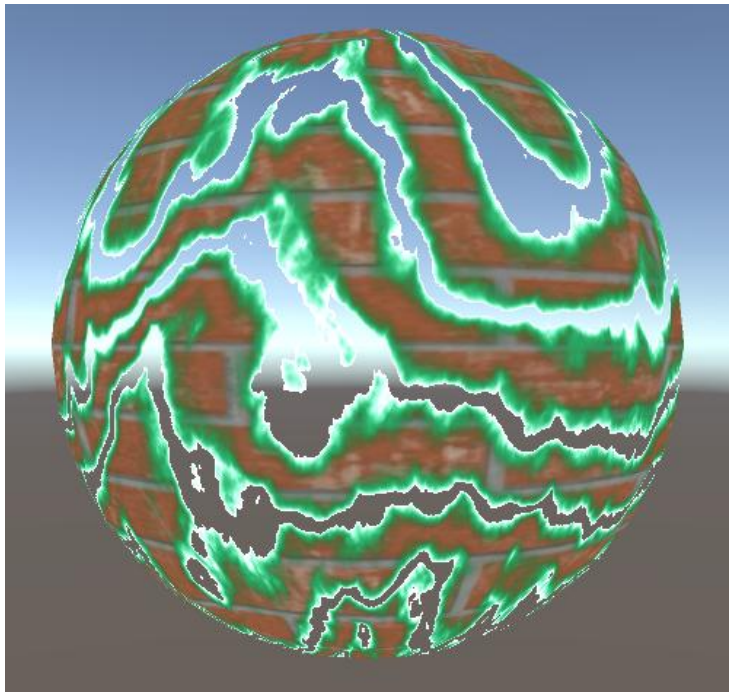
TI GAMES
TI Shader Bundle
★★★★★ (11)
$70

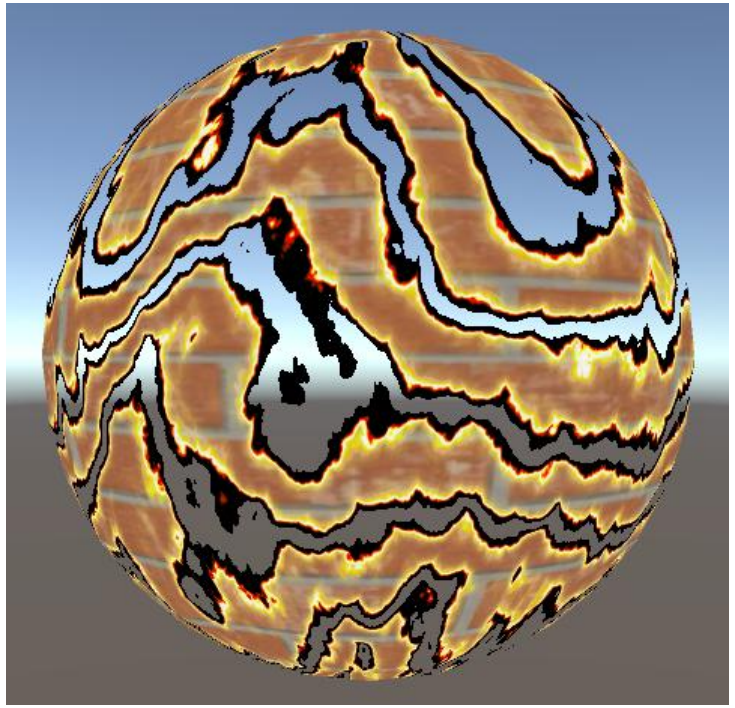
KOGE

ークショップIV

今回やること

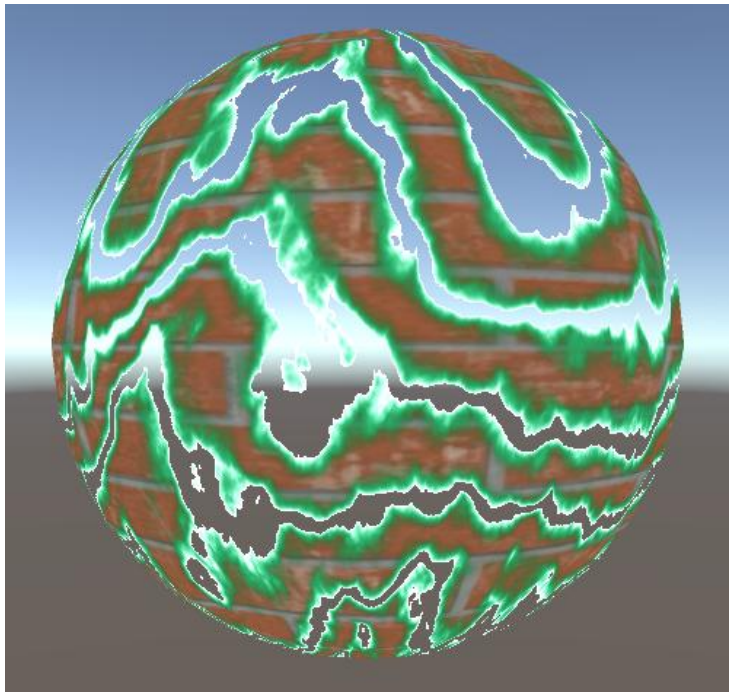
- ノイズ関数を使って、ディゾルブを実現しよう
  1. ノイズによるαクリップ
  2. 色をつける
  3. 消える縁を明るくする
  4. 物体が消えていく
  5. 別パターンを作成

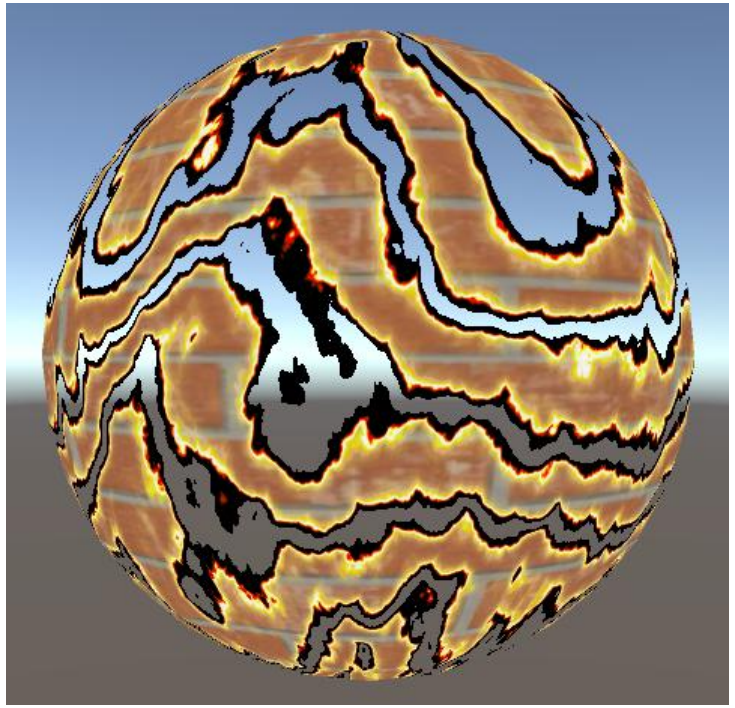




今回やること

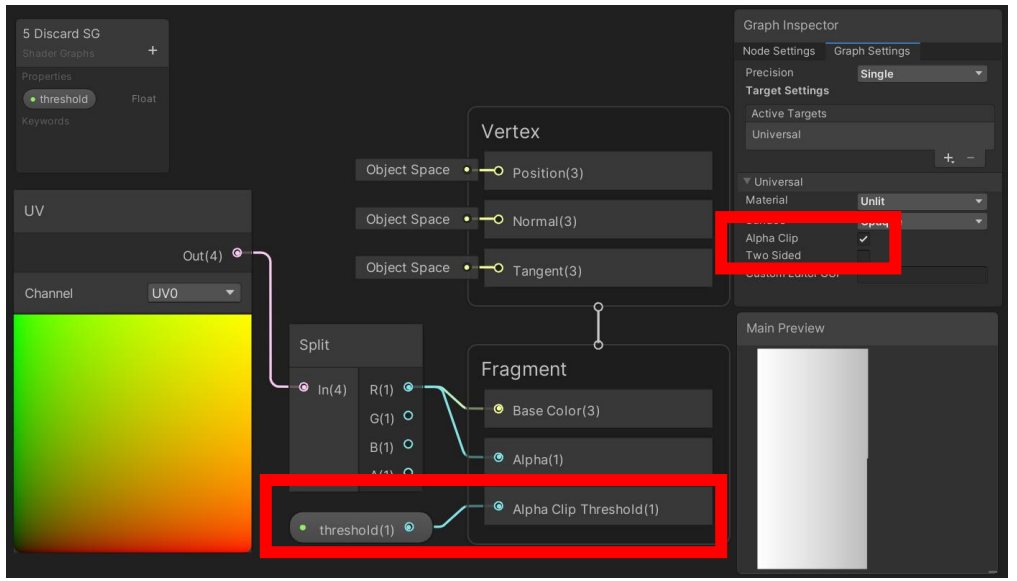
- ノイズ関数を使って、ディゾルブを実現しよう
  1. ノイズによるαクリップ
  2. 色をつける
  3. 消える縁を明るくする
  4. 物体が消えていく
  5. 別パターンを作成

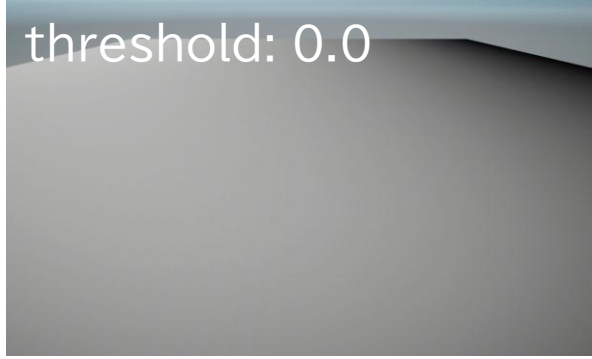


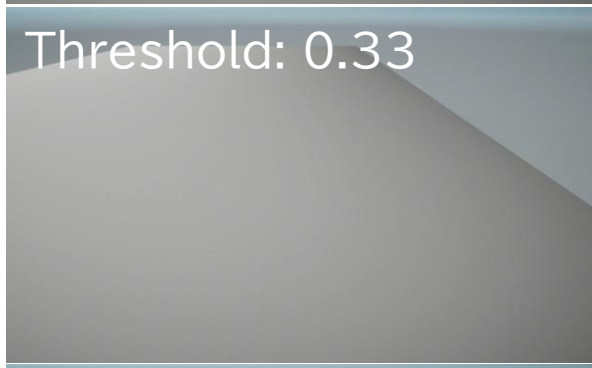


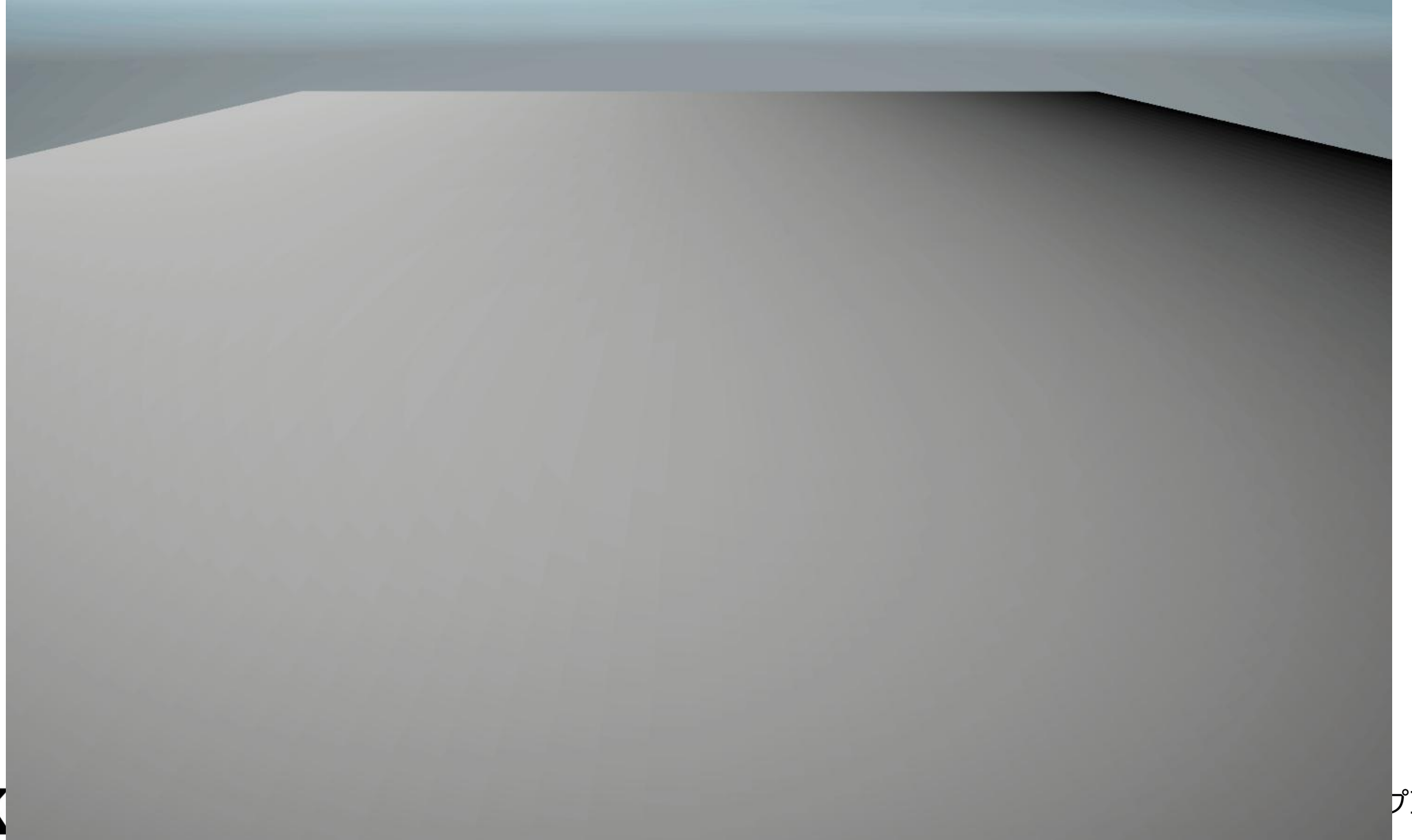
Alpha Clip

- フラグメントの破棄
  - 早期Zカリングが無効になるので、パフォーマンスは悪い
  - Aブレンディングの不透明で合成するよりはまし
- パラメータの値を変えることで、少しづつ削ることができる
  - 例えばテクスチャ座標値

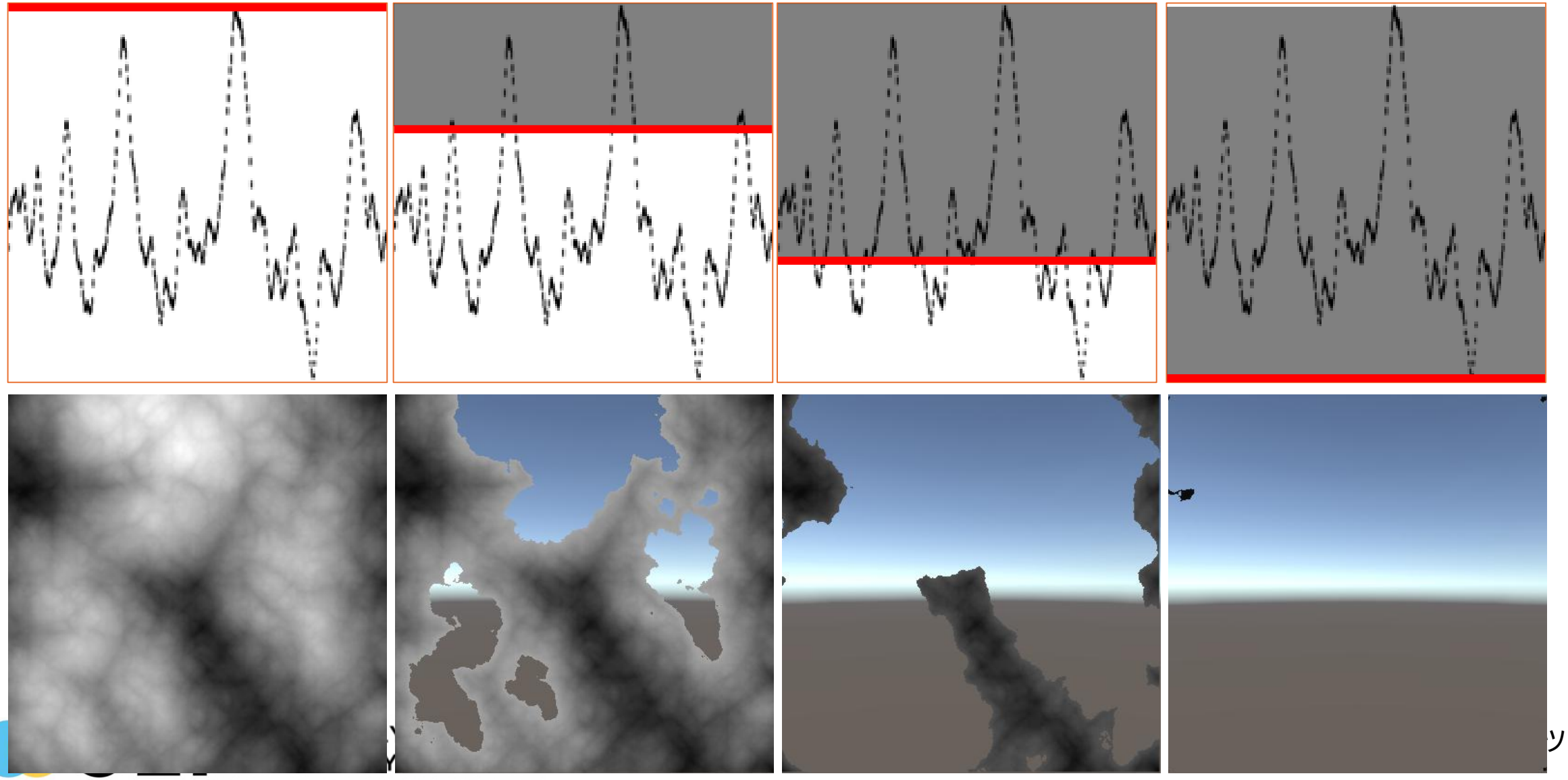




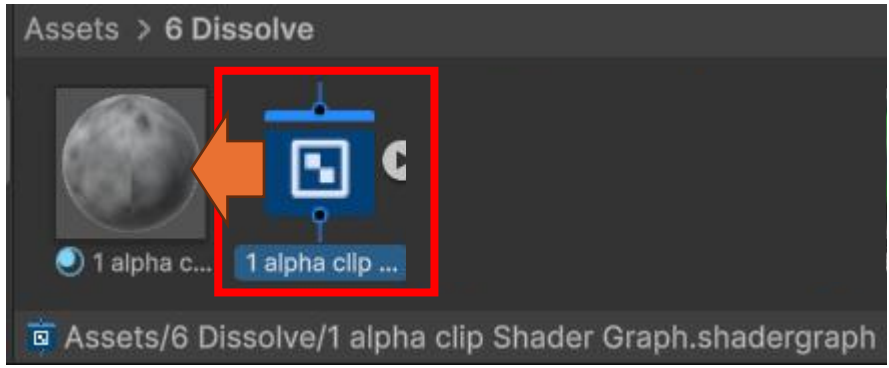




ノイズによるAlpha Clip

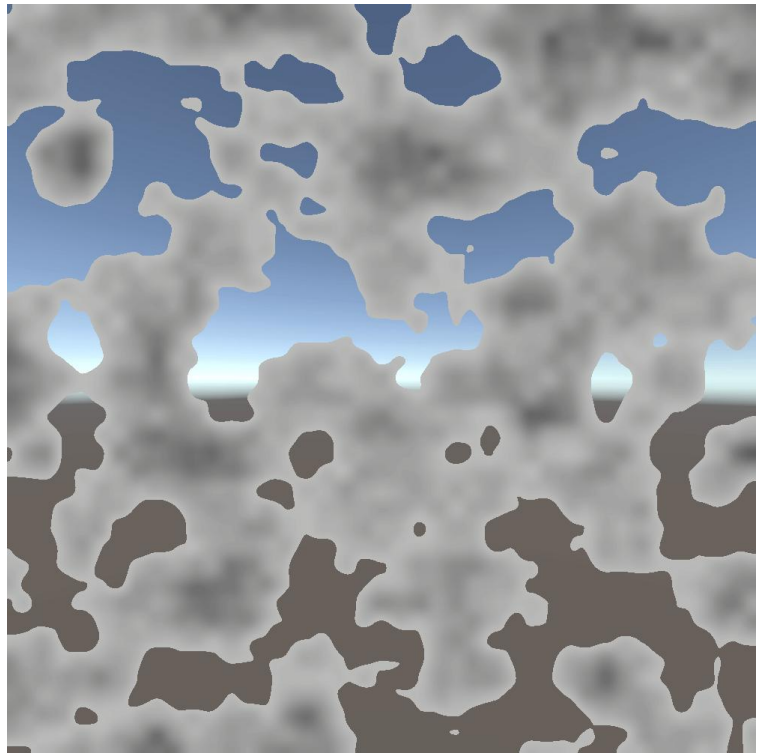


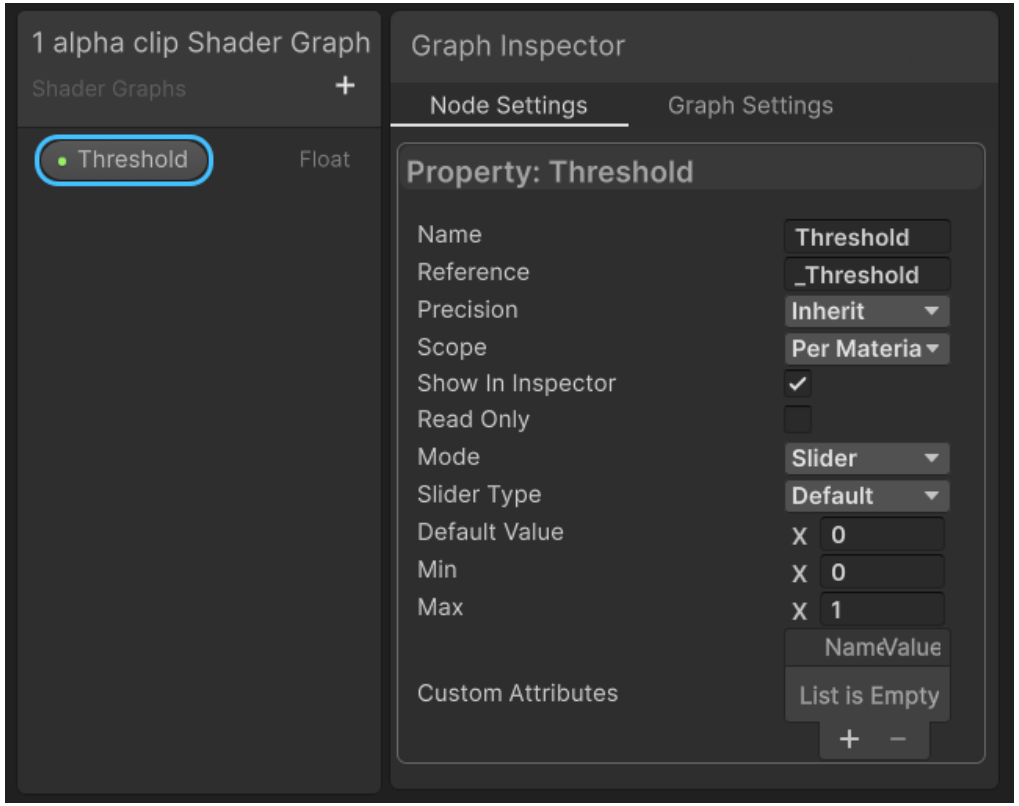
やってみよう：その1

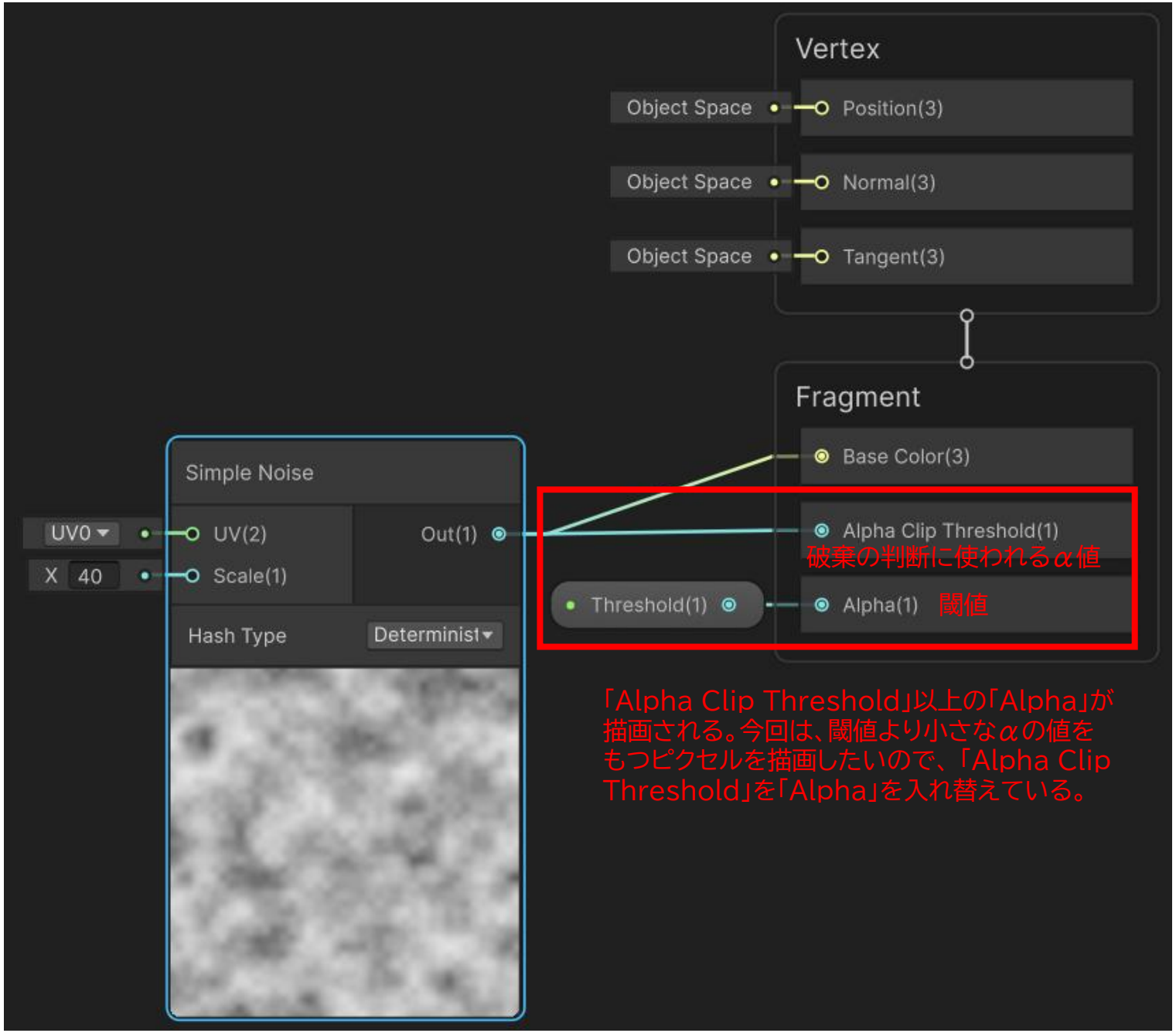


ノイズによるαクリップ

- Shader Graphを作成
  - 「6 Dissolve/1 alpha clip Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を1つ追加(次は設定例)
    - Threshold: 破棄される閾値
      - Float型 (Mode: Slider)
      - 範囲: [0, 1]
      - 初期値: 0



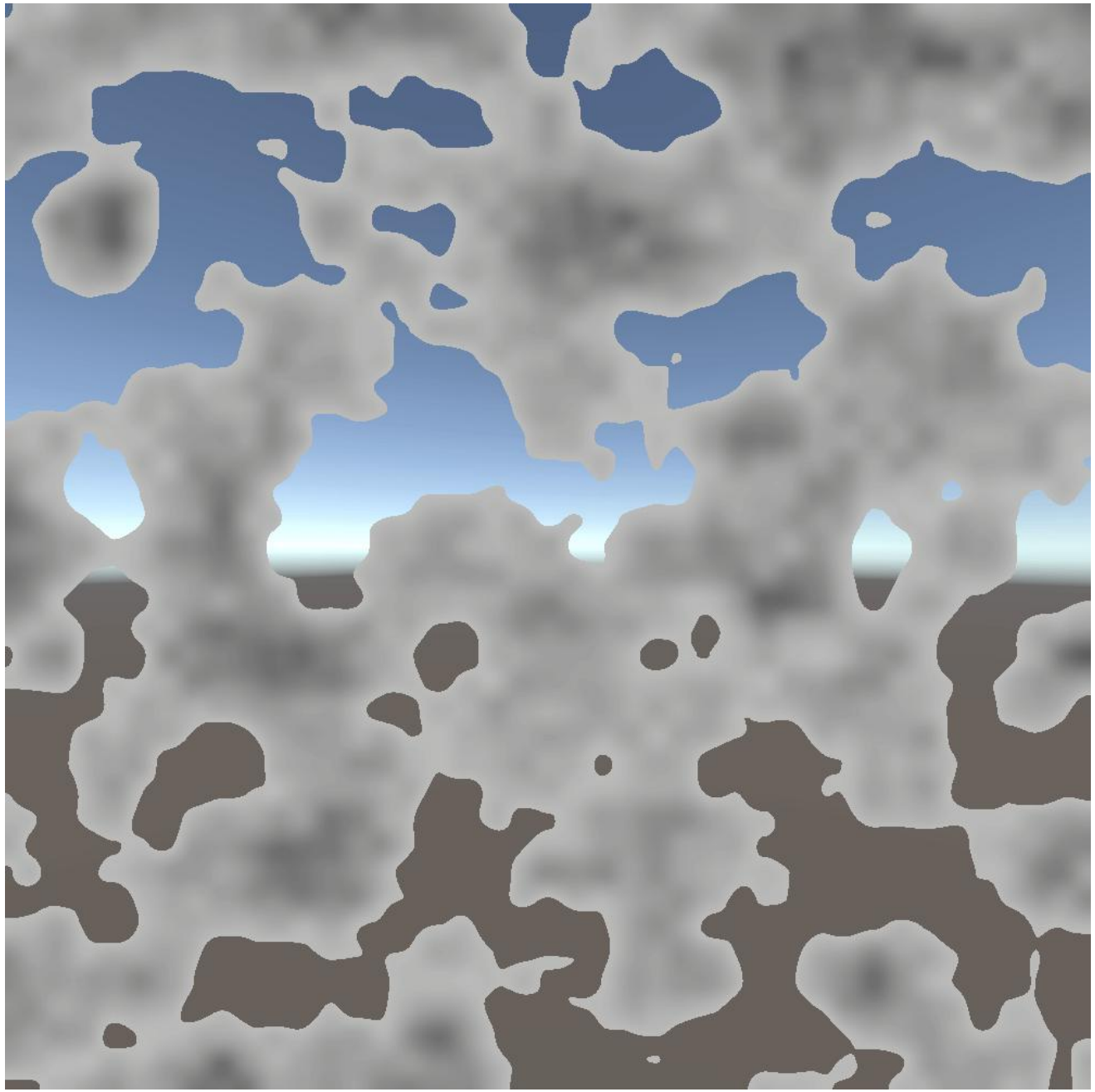




「Alpha Clip Threshold」以上の「Alpha」が描画される。今回は、閾値より小さなαの値をもつピクセルを描画したいので、「Alpha Clip Threshold」を「Alpha」を入れ替えている。

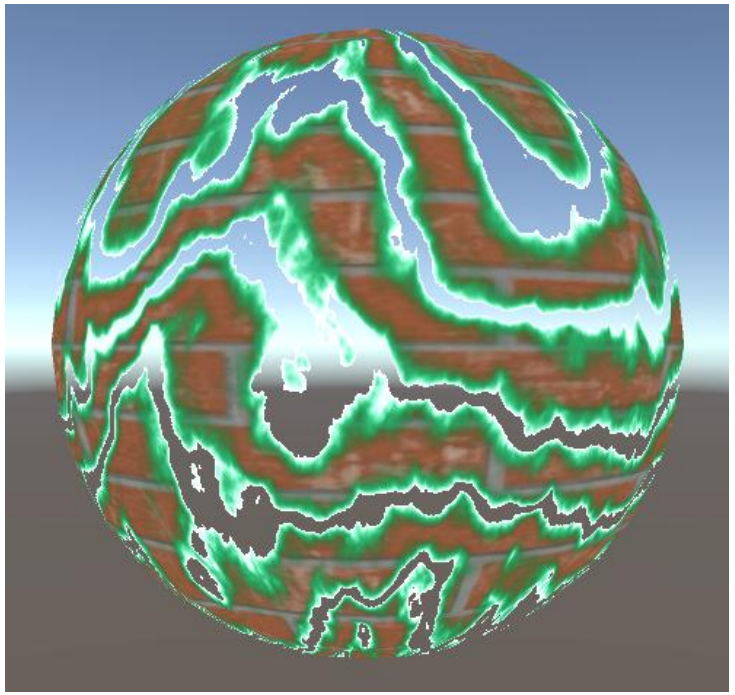
ここまでの状態

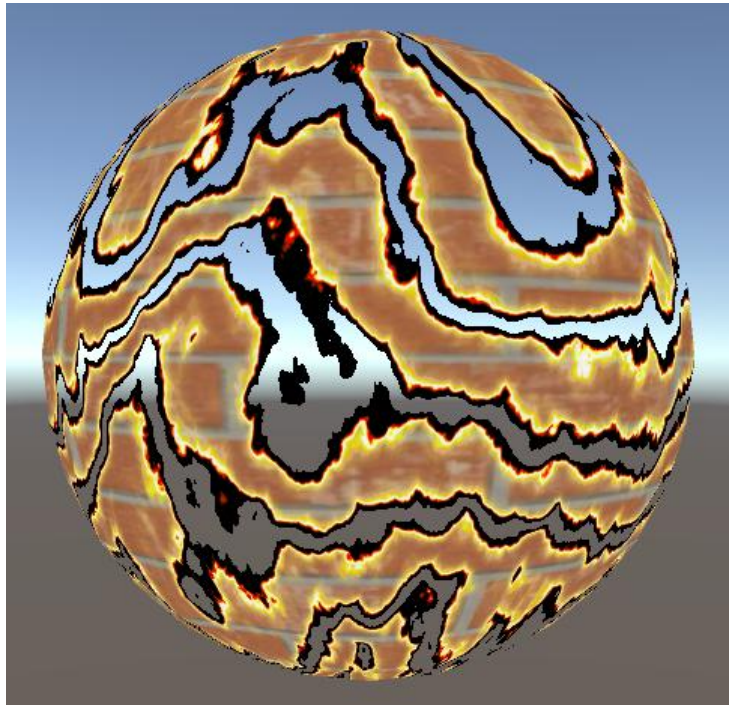
- マテリアルのスライダーを操作して観察しよう



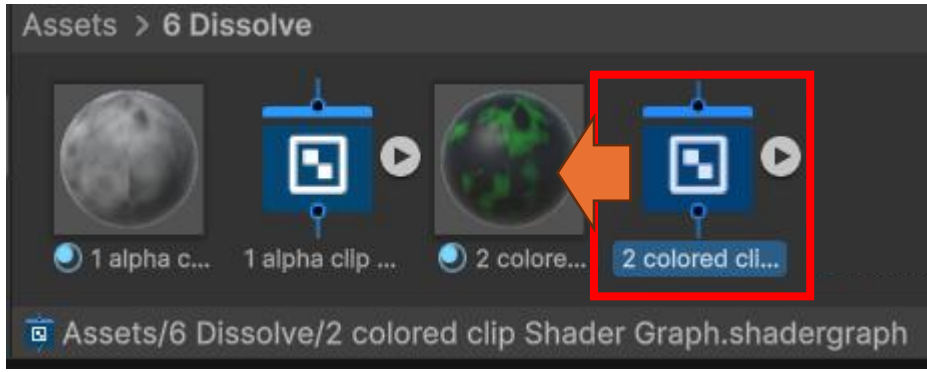
今回やること

- ノイズ関数を使って、ディゾルブを実現しよう
  1. ノイズによるαクリップ
  2. 色をつける
  3. 消える縁を明るくする
  4. 物体が消えていく
  5. 別パターンを作成



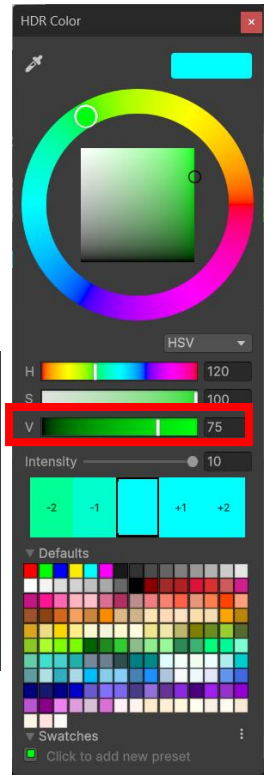


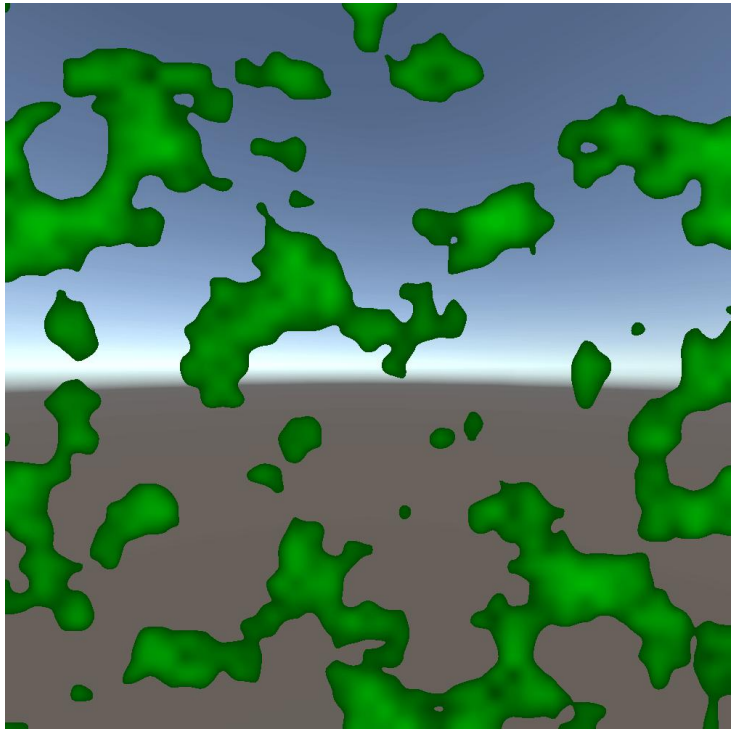
やってみよう:その2

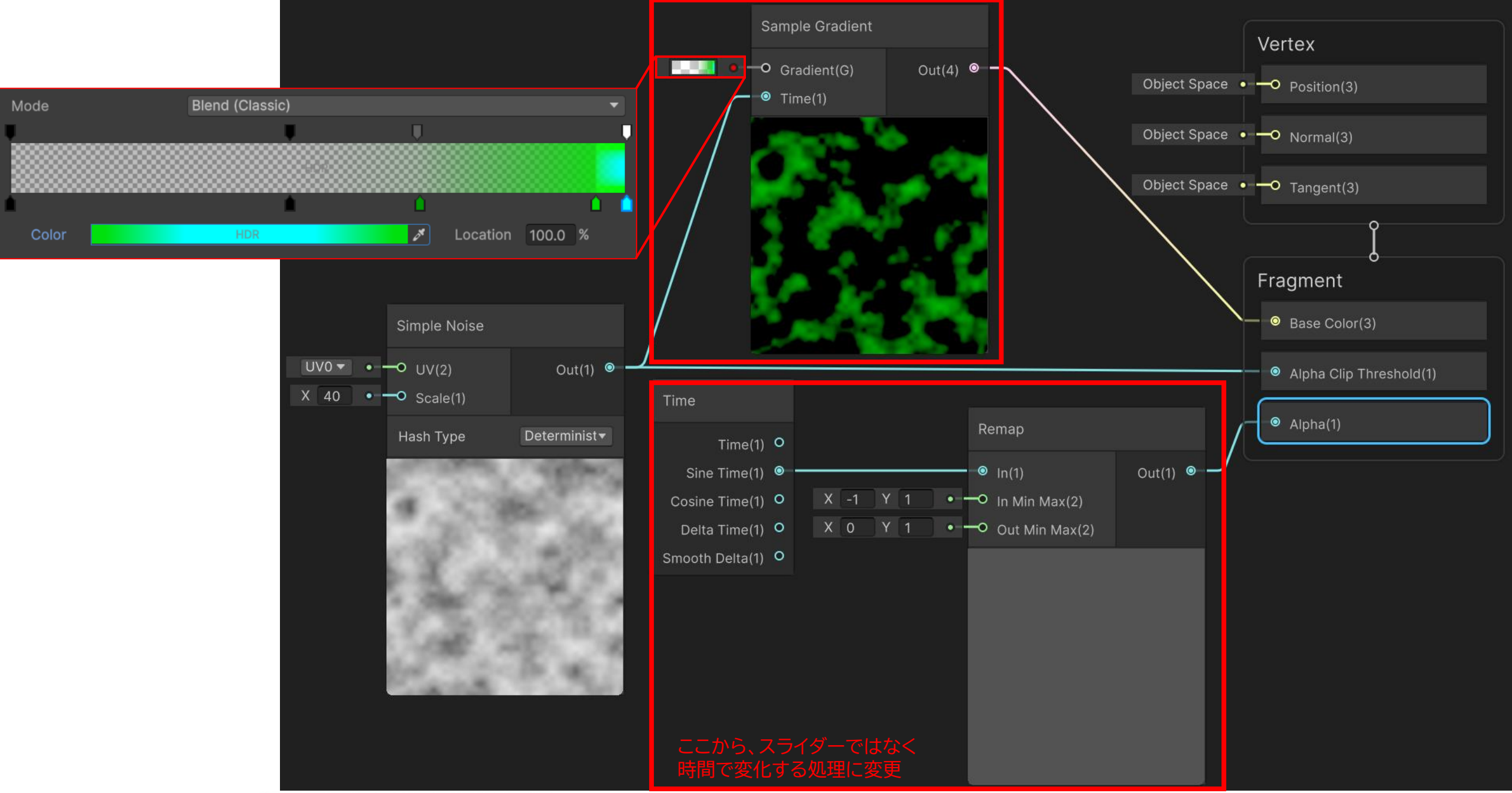


色をつける

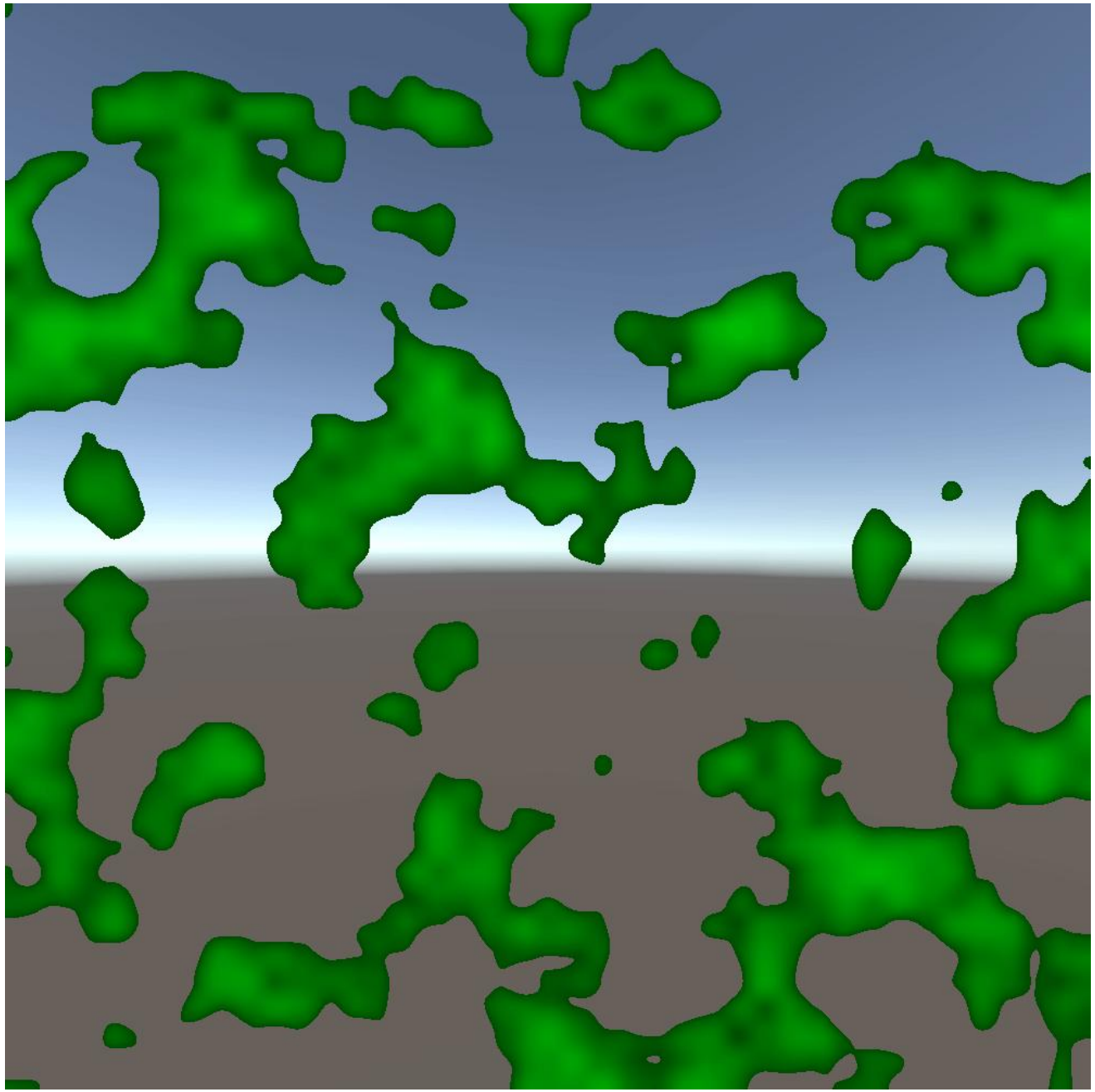
- Shader Graphを作成
  - 「6 Dissolve/2 colored clip Material」に設定
- Shader Graphを実装
  （ノード構成は次頁）
  - グラデーションを導入
    - HDRを使うと、強い光も設定可能





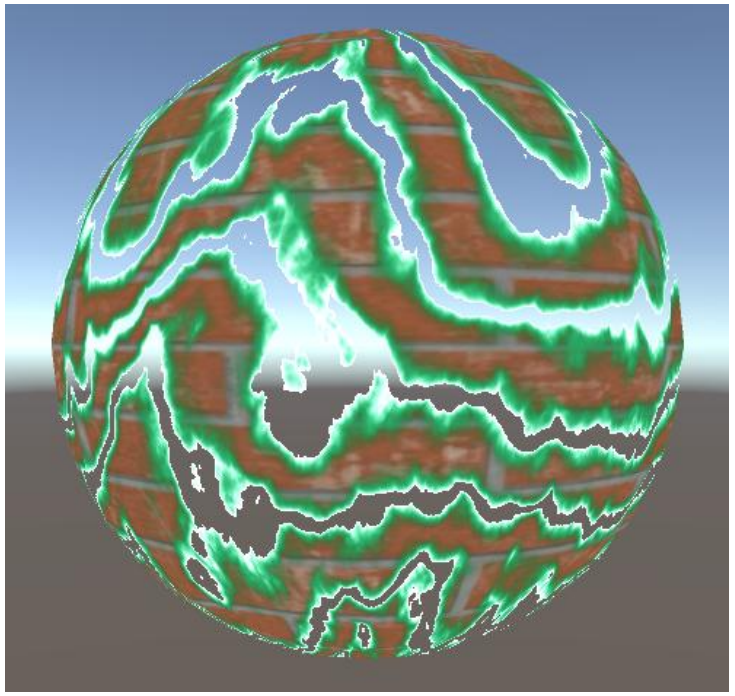

Mode
Blend (Classic)
Color
HDR
Location
100.0
%
Sample Gradient
Gradient(G)
Time(1)
Out(4)
Vertex
Object Space
Position(3)
Object Space
Normal(3)
Object Space
Tangent(3)
Fragment
Base Color(3)
Alpha Clip Threshold(1)
Alpha(1)
Simple Noise
UV0
UV(2)
Out(1)
X 40
Scale(1)
Hash Type
Deterministi
Time
Time(1)
Sine Time(1)
Cosine Time(1)
Delta Time(1)
Smooth Delta(1)
Remap
In(1)
Out(1)
X -1 Y 1
In Min Max(2)
X 0 Y 1
Out Min Max(2)
ここから、スライダーではなく
時間で変化する処理に変更

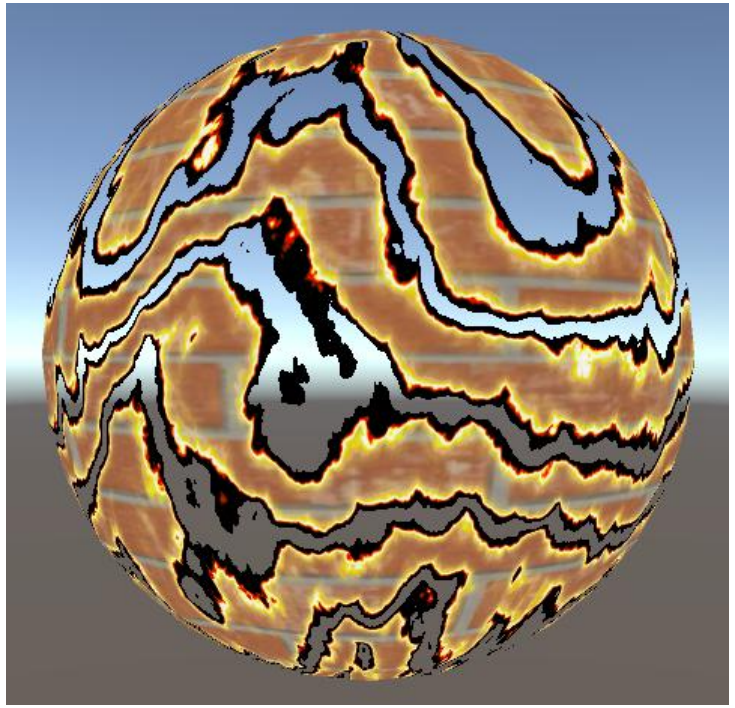
ここまでの状態



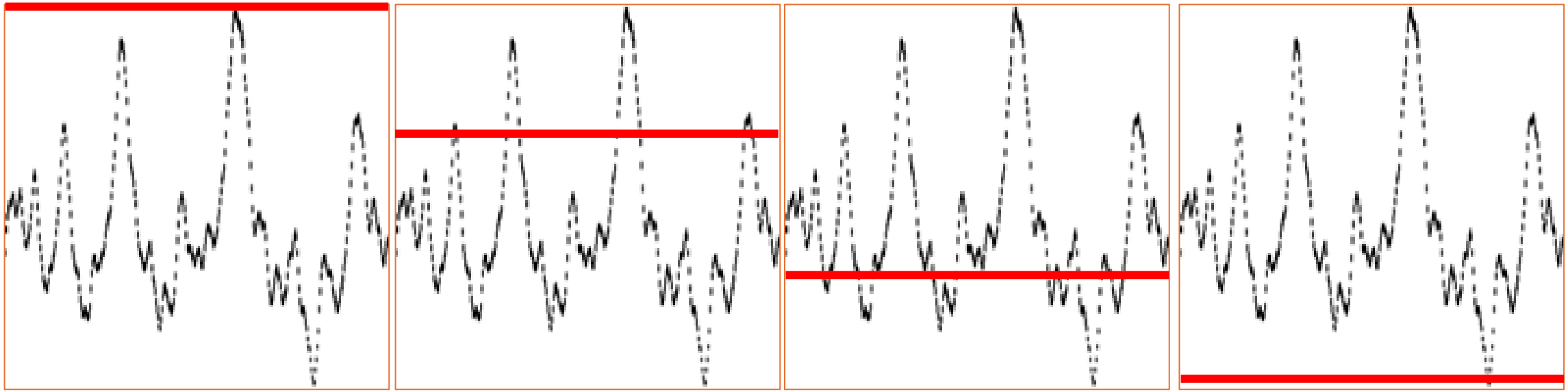
今回やること

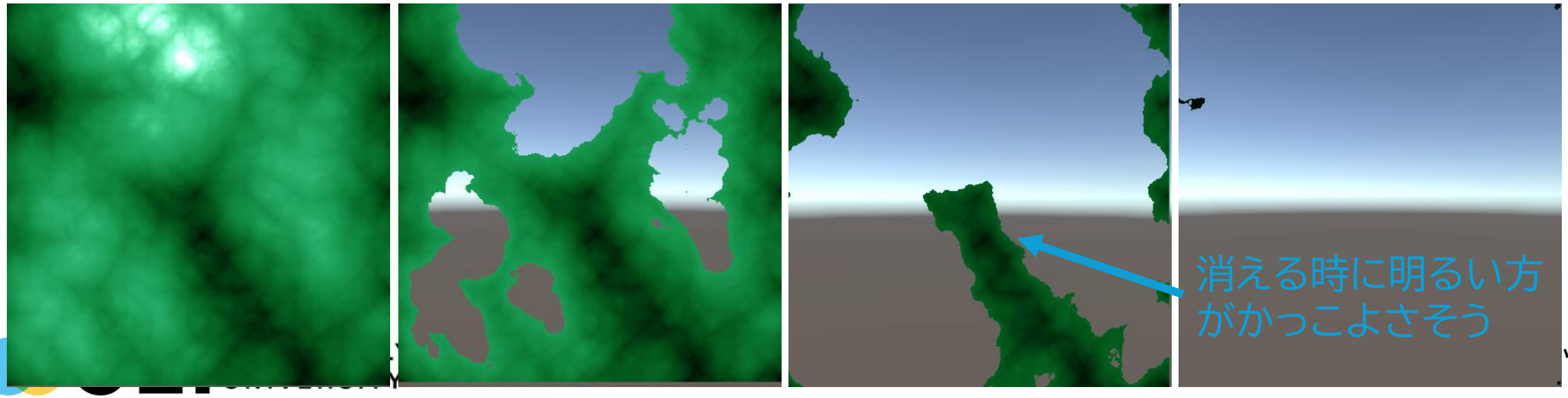
- ノイズ関数を使って、ディゾルブを実現しよう
  1. ノイズによるαクリップ
  2. 色をつける
  3. 消える縁を明るくする
  4. 物体が消えていく
  5. 別パターンを作成



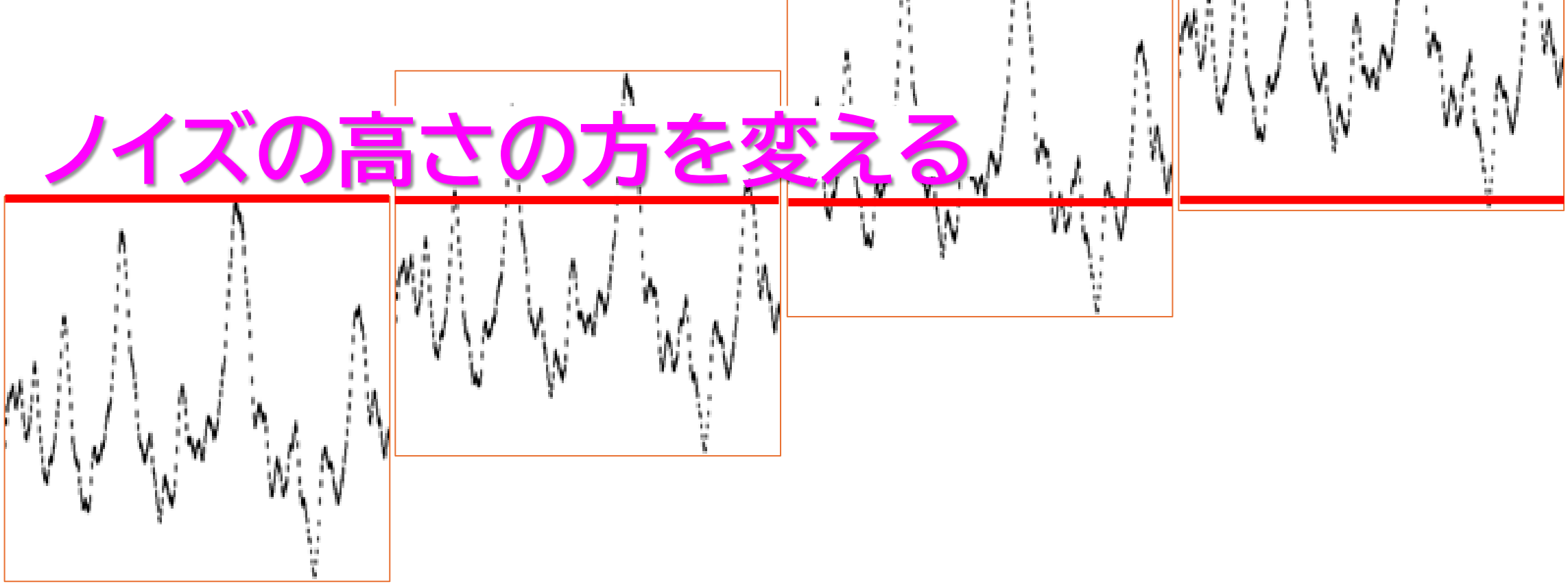


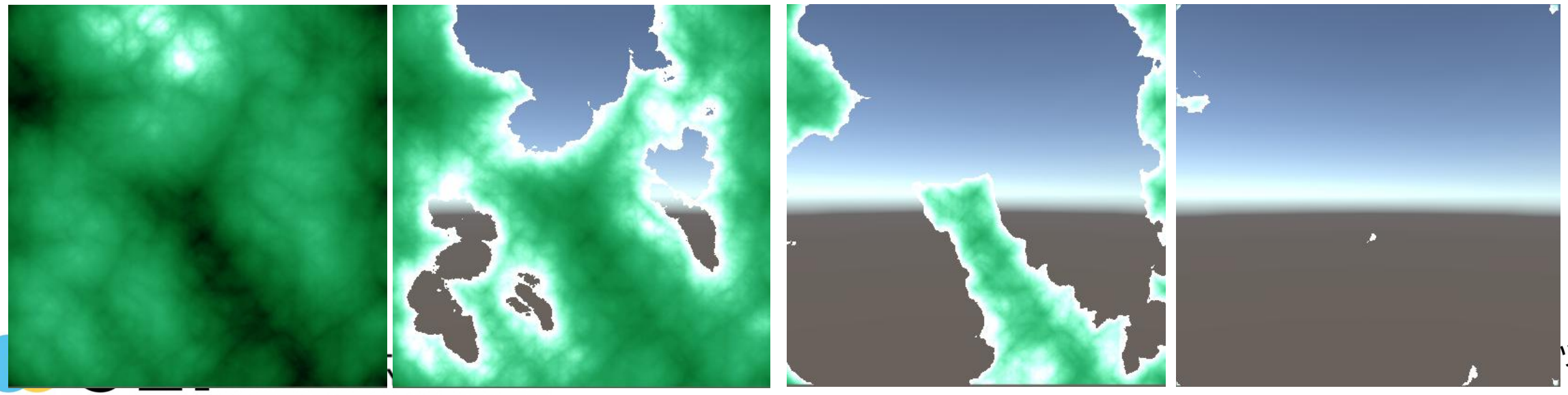
観察してみると





ノイズの高さの方を変える

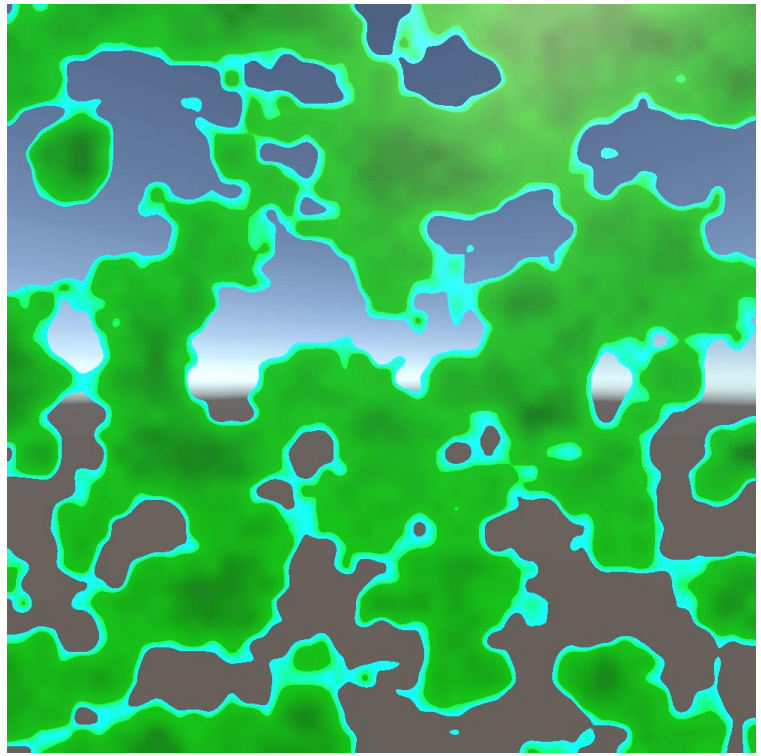


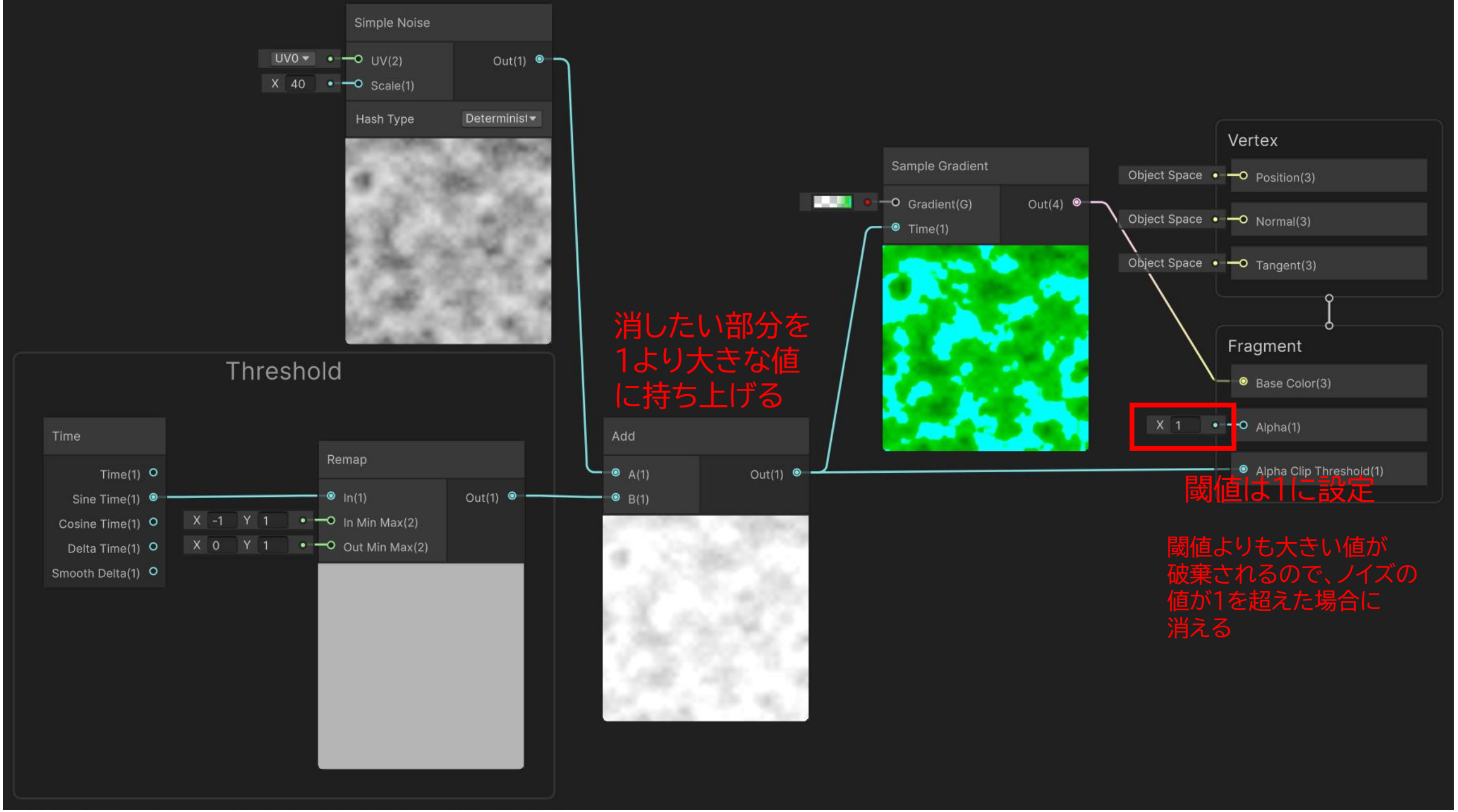


やってみよう:その3

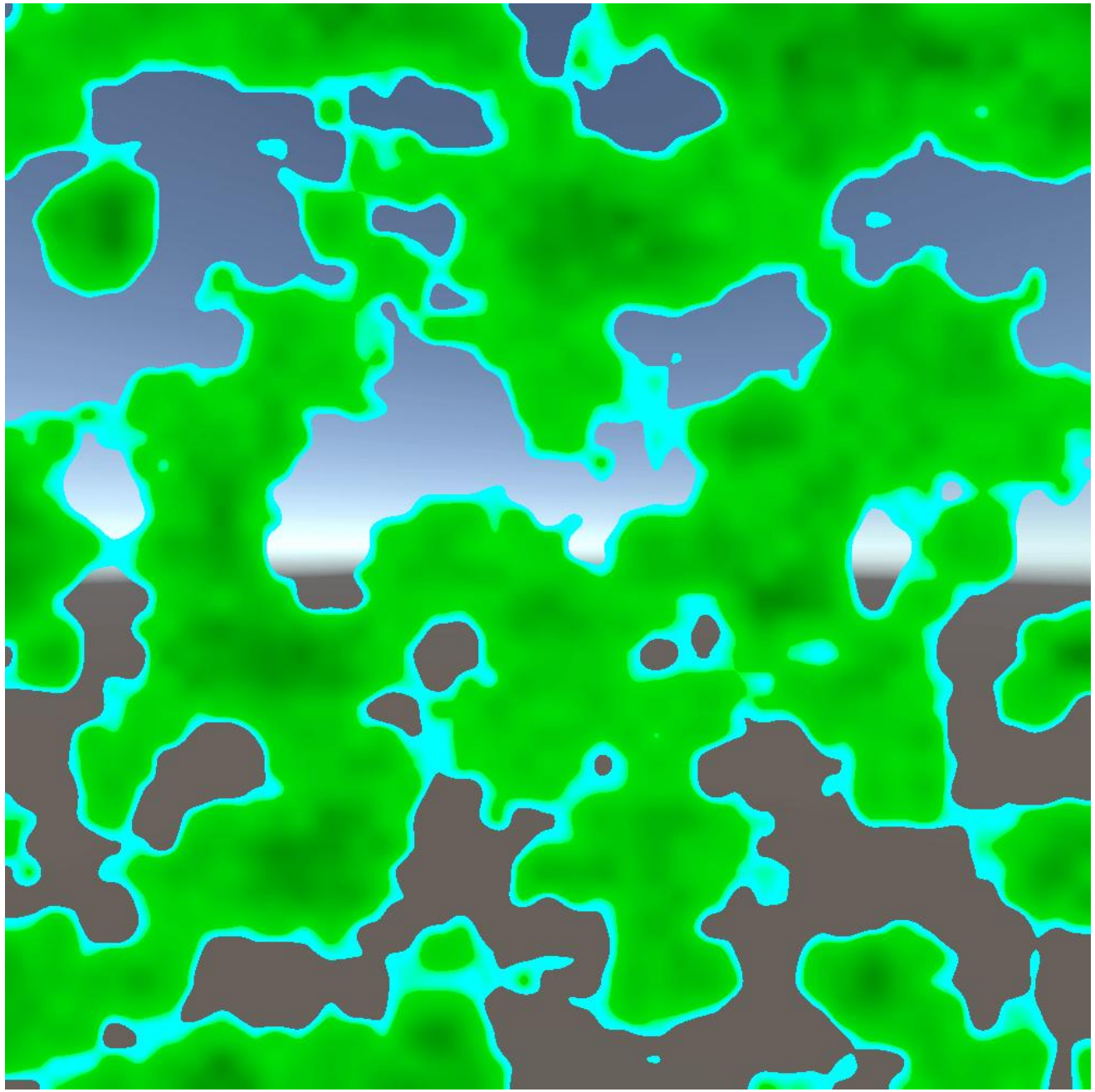
消える縁を明るくする

- Shader Graphを作成
  - 「6 Dissolve/3 dissolve Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - グラデーションを導入
    - 同じグラデーション



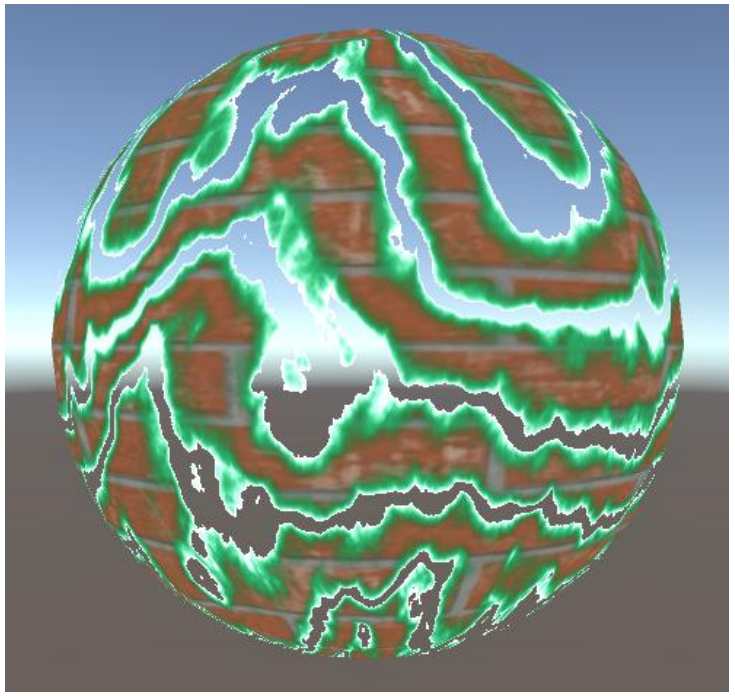

Simple Noise
UV0
X 40
UV(2)
Scale(1)
Out(1)
Hash Type
Deterministi
Threshold
Time
Time(1)
Sine Time(1)
Cosine Time(1)
Delta Time(1)
Smooth Delta(1)
Remap
In(1)
In Min Max(2)
Out Min Max(2)
X -1 Y 1
X 0 Y 1
Out(1)
消したい部分を
1より大きな値
に持ち上げる
Add
A(1)
B(1)
Out(1)
Sample Gradient
Gradient(G)
Time(1)
Out(4)
Vertex
Object Space
Position(3)
Normal(3)
Tangent(3)
Fragment
Base Color(3)
X 1
Alpha(1)
Alpha Clip Threshold(1)
閾値は1に設定
閾値よりも大きい値が
破棄されるので、ノイズの
値が1を超えた場合に
消える

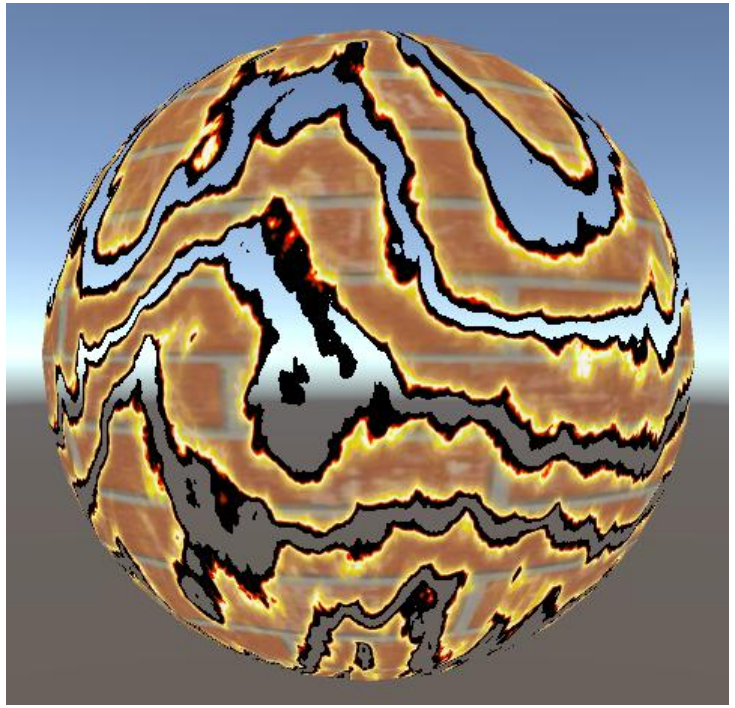
ここまでの状態



今回やること

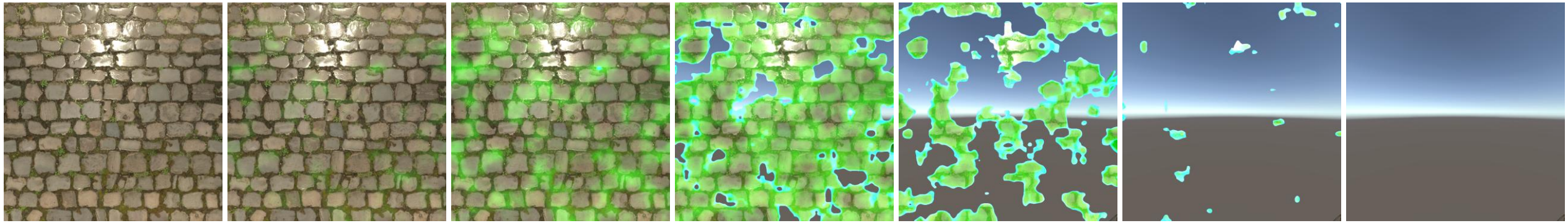
- ノイズ関数を使って、ディゾルブを実現しよう
  1. ノイズによるαクリップ
  2. 色をつける
  3. 消える縁を明るくする
  4. 物体が消えていく
  5. 別パターンを作成



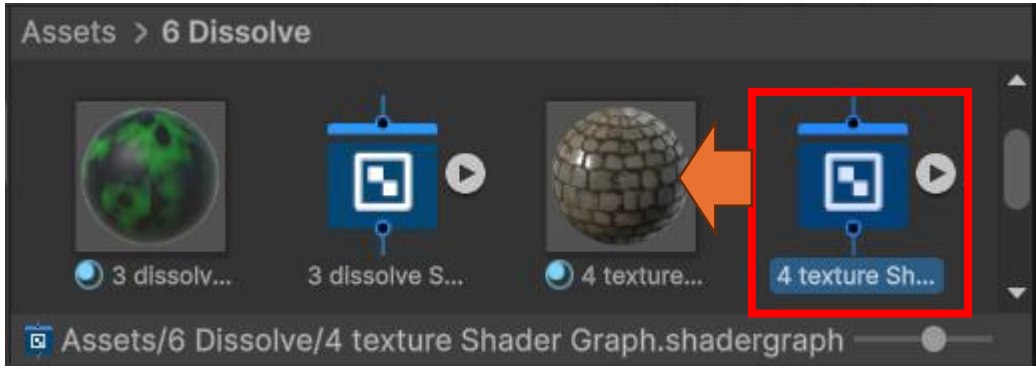


物体を消す

- 物体の描画時にノイズとαクリッピングを乗せる
  - ここでは加算

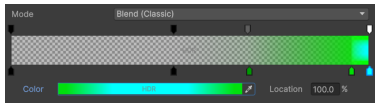


やってみよう:その4

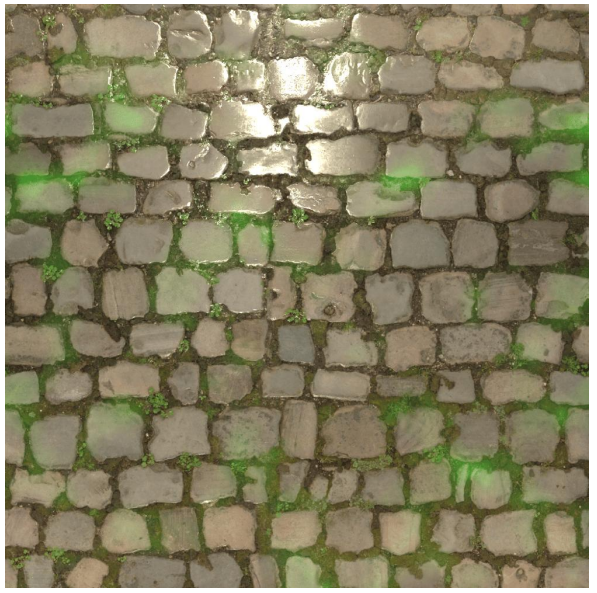


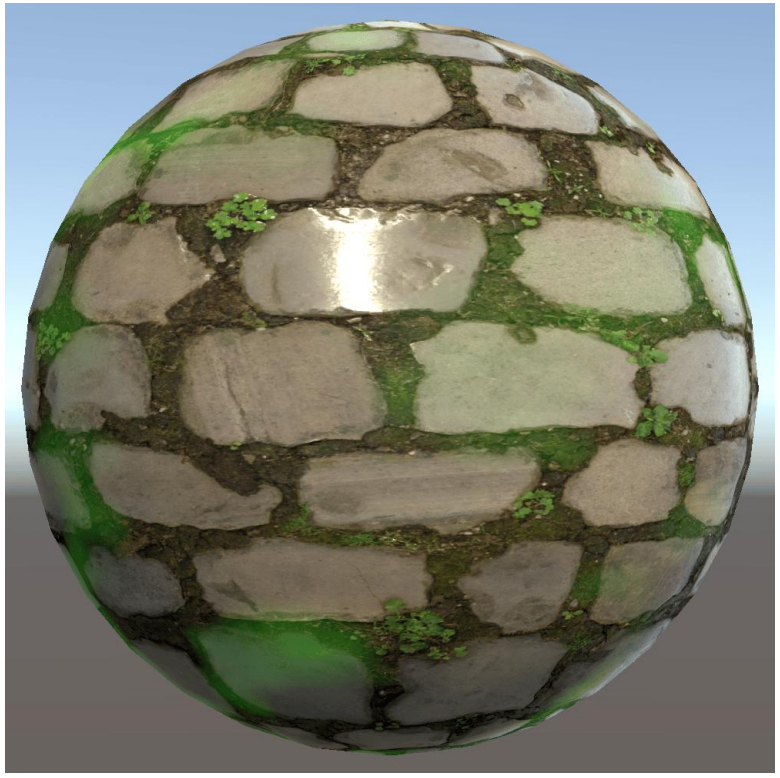
物体が消えていく

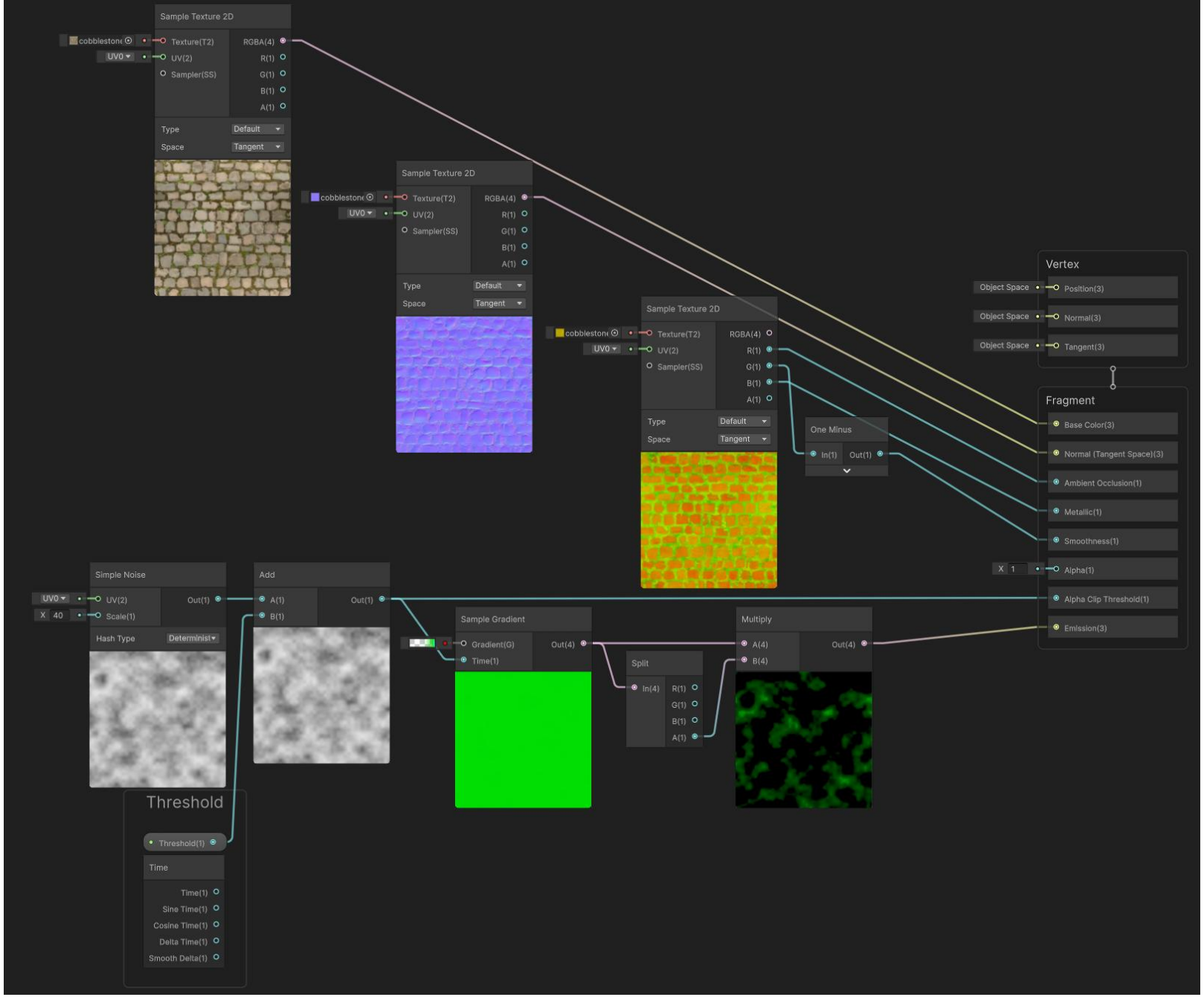
- Shader Graphを作成
  - 「6 Dissolve/4 texture Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を1つ追加(次は設定例)
    - Threshold: 破棄される閾値
      - Float型 (Mode: Slider)
      - 範囲: [-1, 1]
      - 初期値: -1
  - グラデーションを導入
    - 同じグラデーション



  - 画像
    - 地面などの模様のPBRテクスチャ

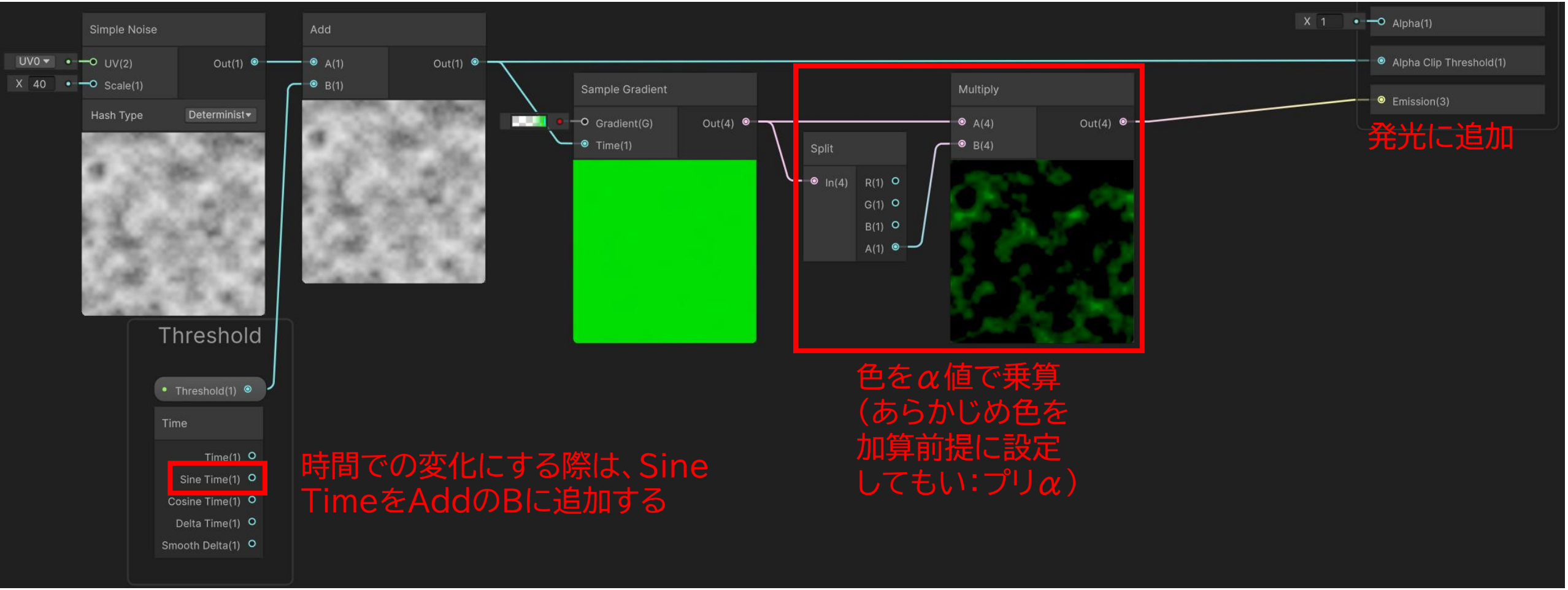




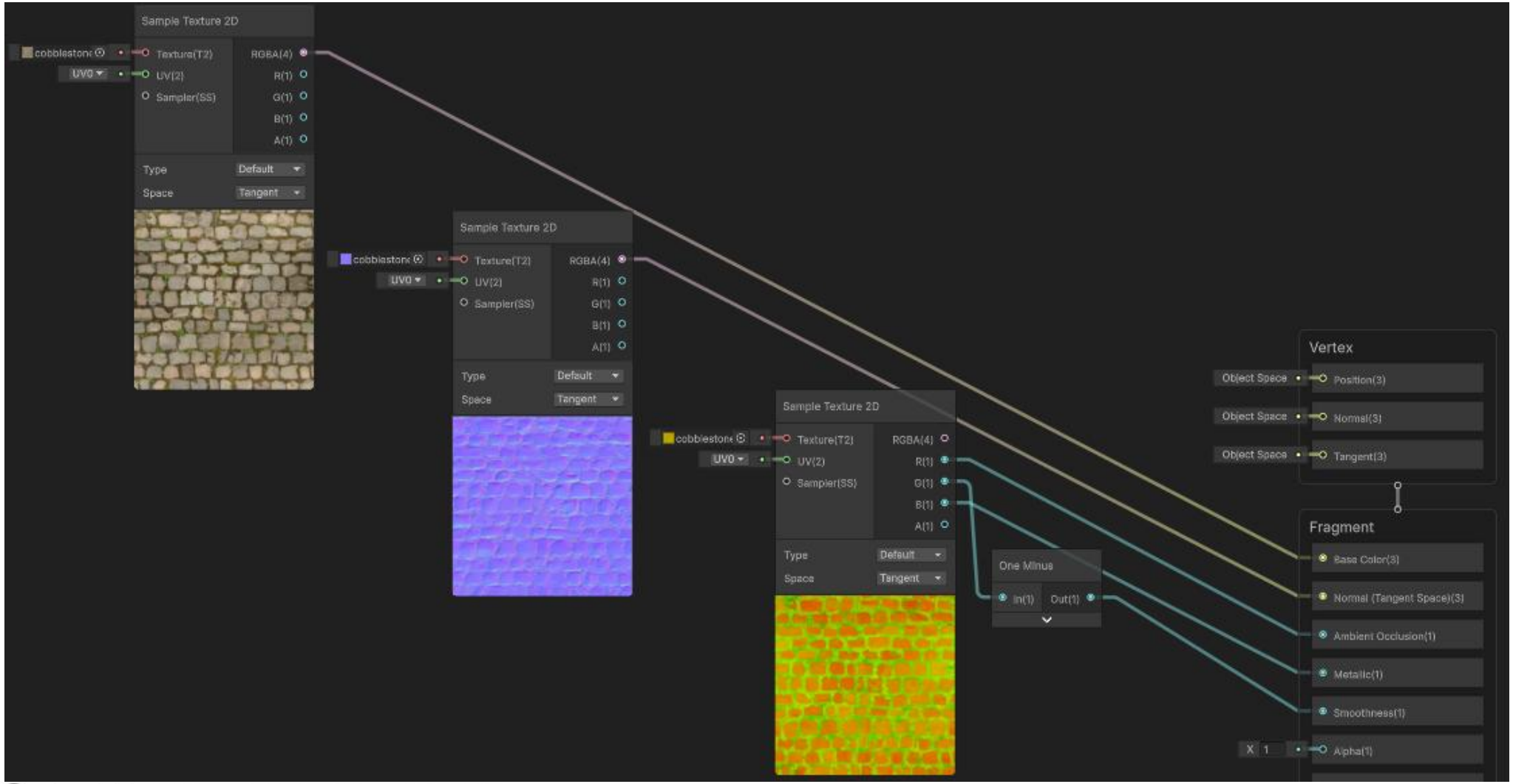


Lit Shader Graph

α成分と発光

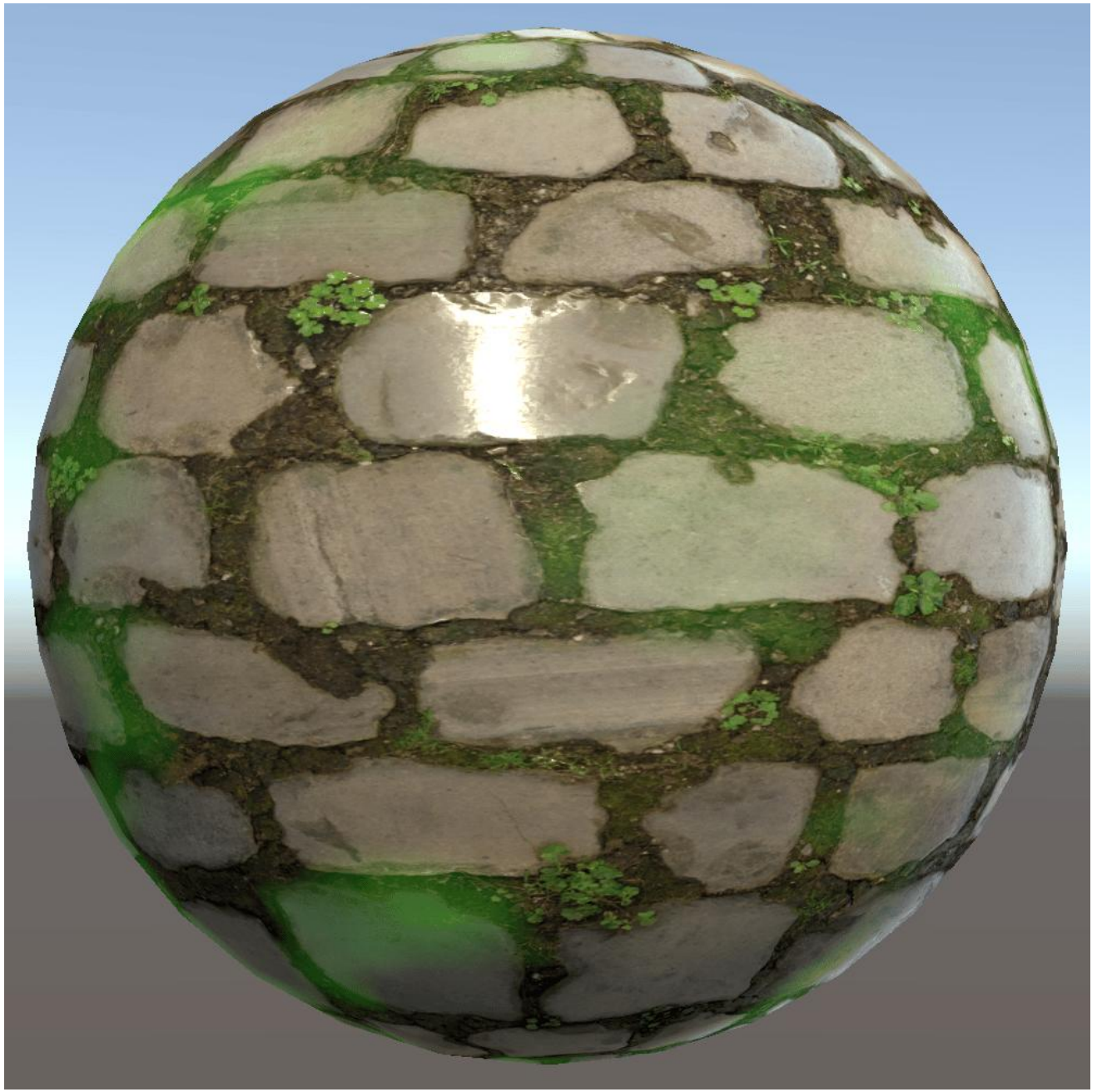

Simple Noise
UV0
UV(2)
Out(1)
X 40
Scale(1)
Hash Type
Deterministic
Add
A(1)
B(1)
Out(1)
Sample Gradient
Gradient(G)
Time(1)
Out(4)
Split
In(4)
R(1)
G(1)
B(1)
A(1)
Multiply
A(4)
B(4)
Out(4)
X 1
Alpha(1)
Alpha Clip Threshold(1)
Emission(3)
発光に追加
Threshold
Threshold(1)
Time
Time(1)
Sine Time(1)
Cosine Time(1)
Delta Time(1)
Smooth Delta(1)
時間での変化にする際は、Sine TimeをAddのBに追加する
色をα値で乗算（あらかじめ色を加算前提に設定してもい：プリα）

物体の色:通常通りに各テクスチャを設定



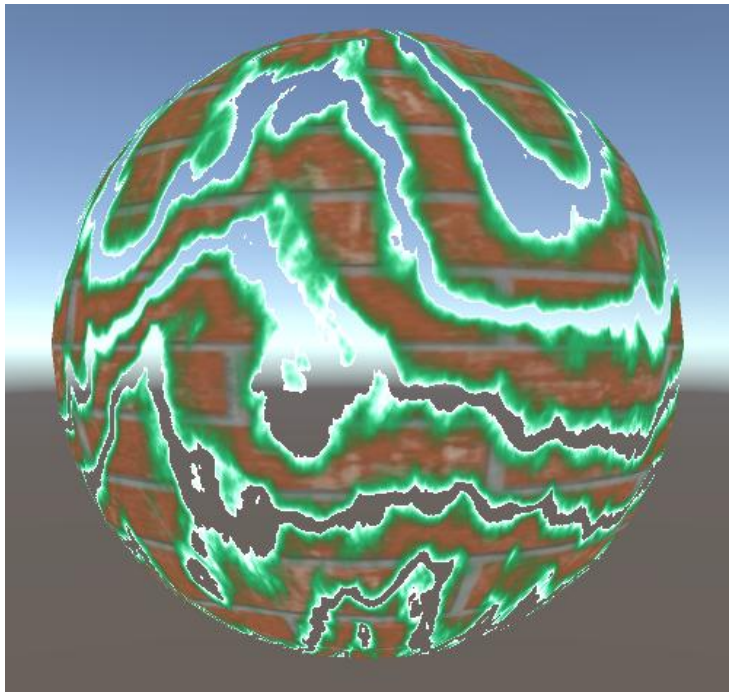
完成1

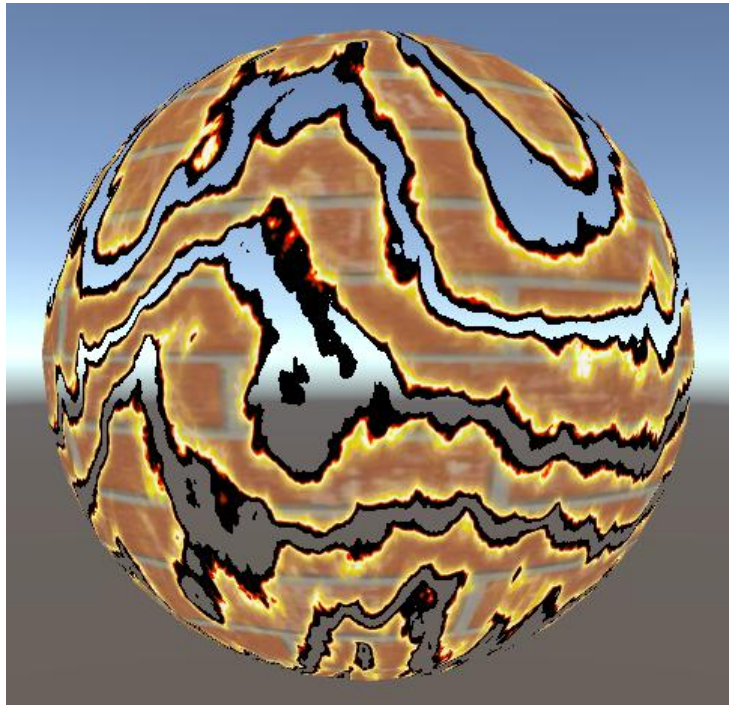
- マテリアルのスライダーを操作して観察しよう



今回やること

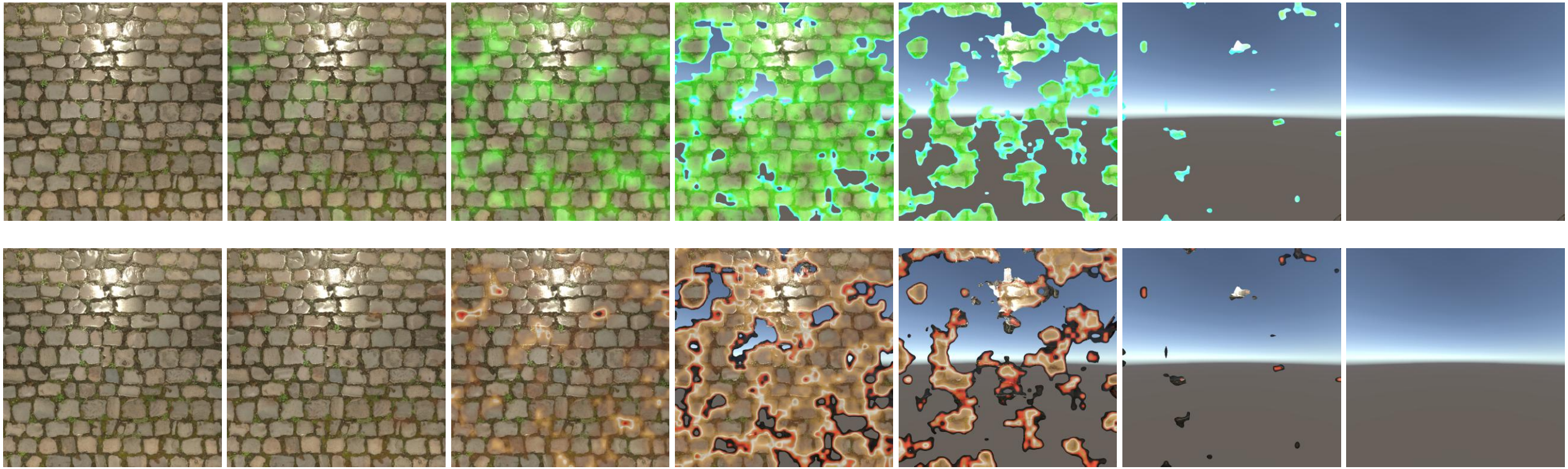
- ノイズ関数を使って、ディゾルブを実現しよう
  1. ノイズによるαクリップ
  2. 色をつける
  3. 消える縁を明るくする
  4. 物体が消えていく
  5. 別パターンを作成



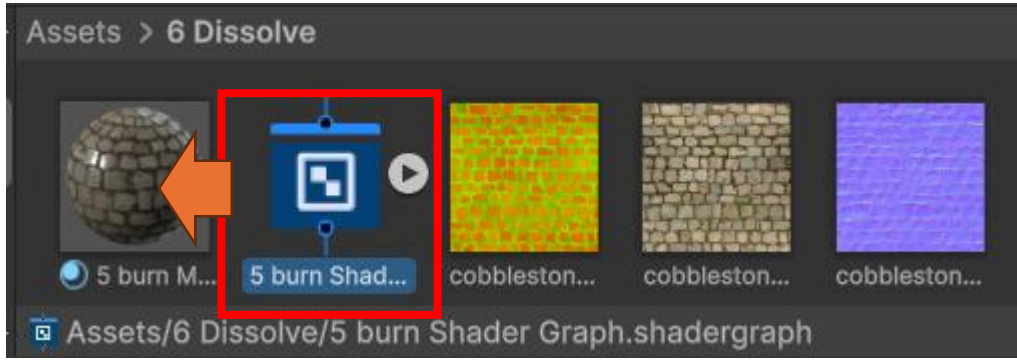


燃えカスのような表現

- 加算では明るくなるだけなので、黒くしていくことはできない
  - 内挿で色を変える



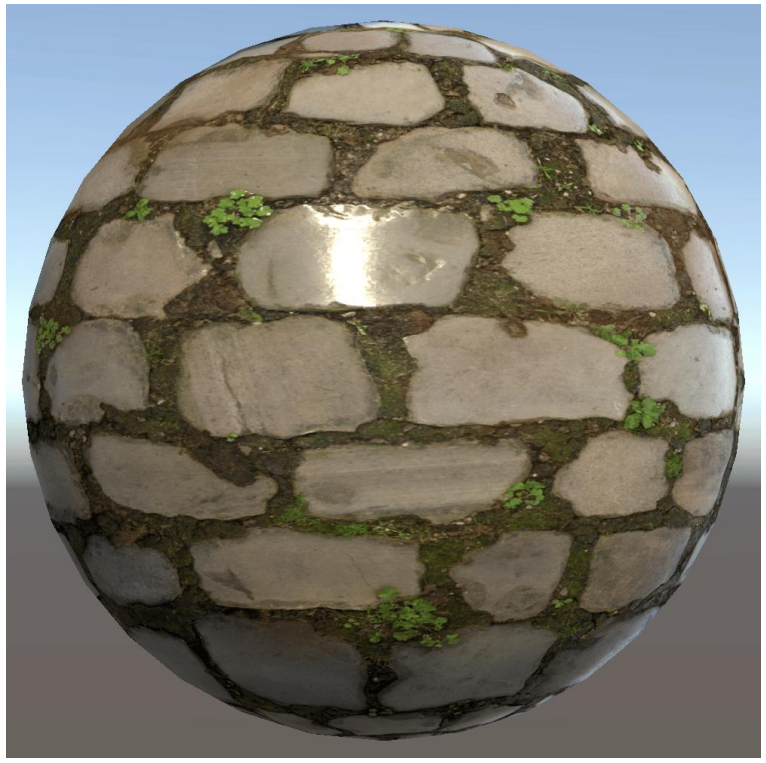
やってみよう:その5

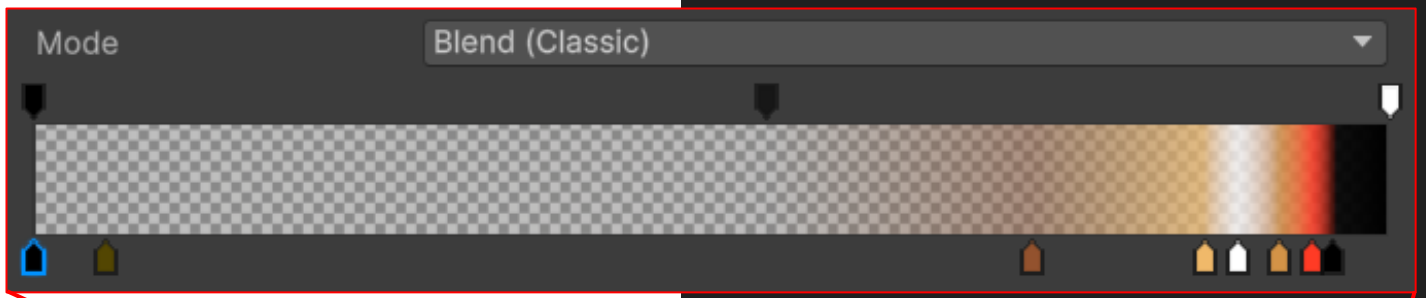


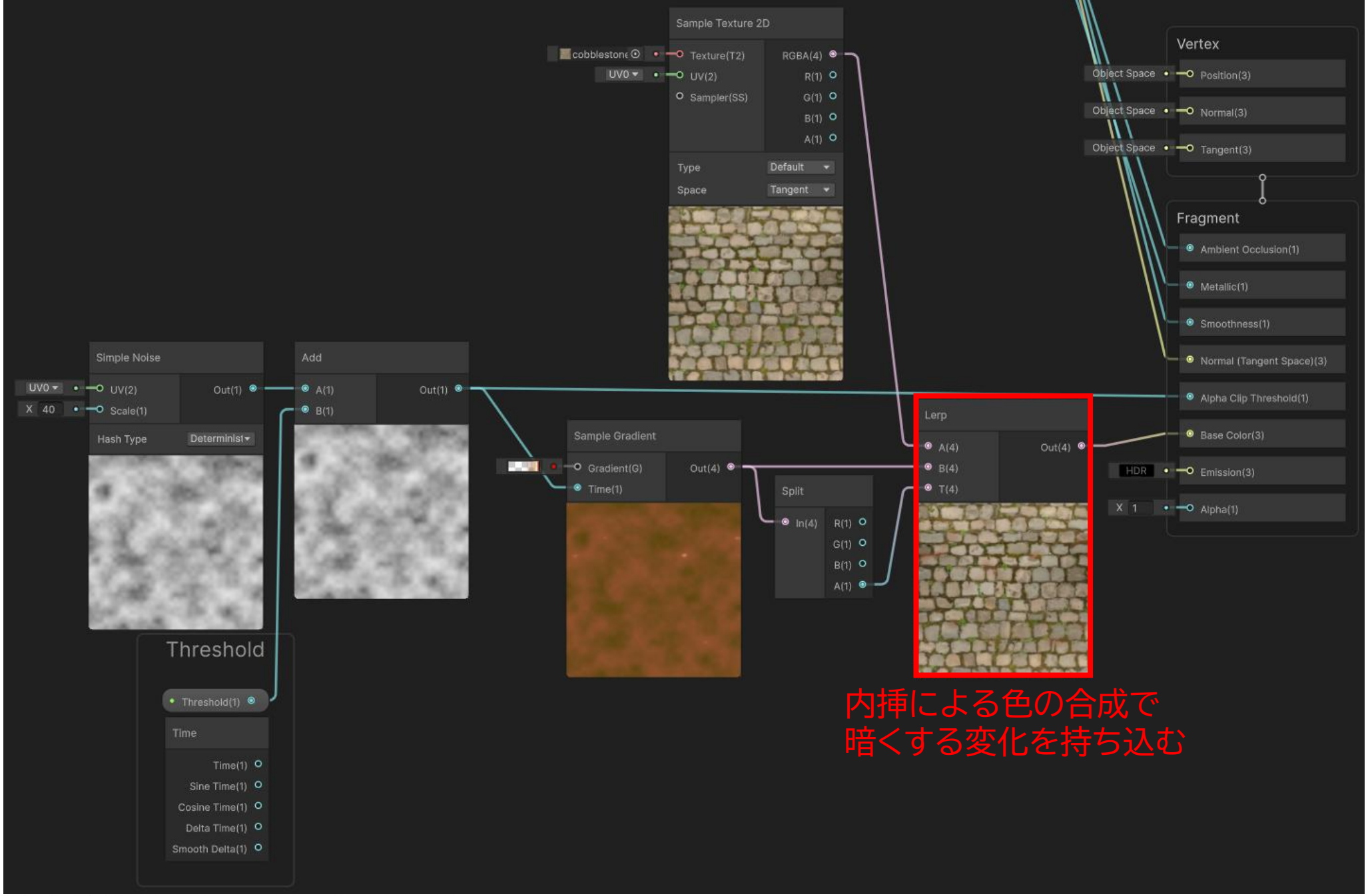
別パターンを作成

- Shader Graphを作成
  - 「6 Dissolve/5 burn Material」に設定
- Shader Graphを実装(ノード構成は次頁)
  - 変数を1つ追加(次は設定例)
    - Threshold: 破棄される閾値
      - Float型 (Mode: Slider)
      - 範囲: [-1, 1]
      - 初期値: -1
  - グラデーションを導入
    - 違うグラデーション

  - 画像
    - 地面などの模様のPBRテクスチャ

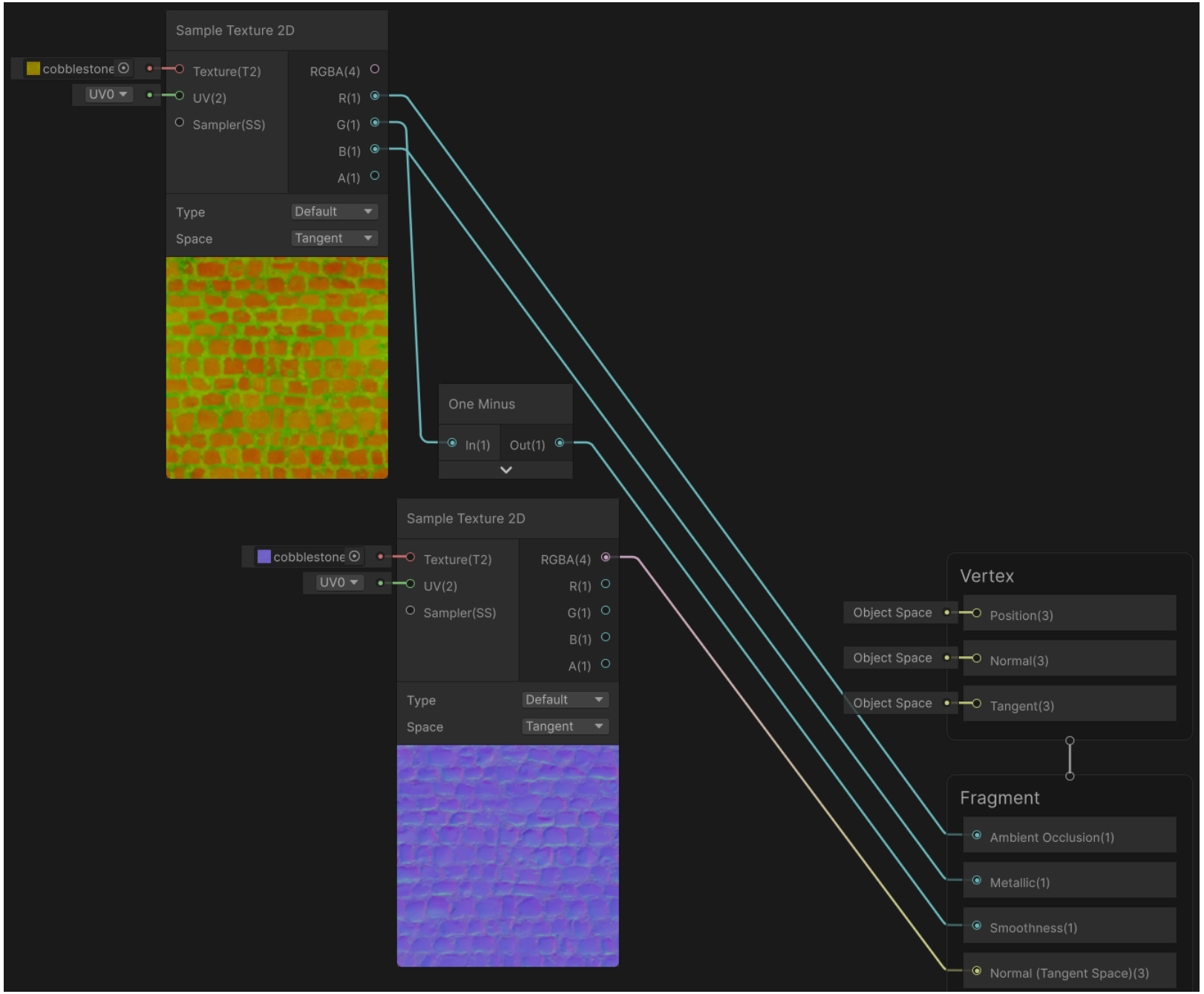



Mode
Blend (Classic)



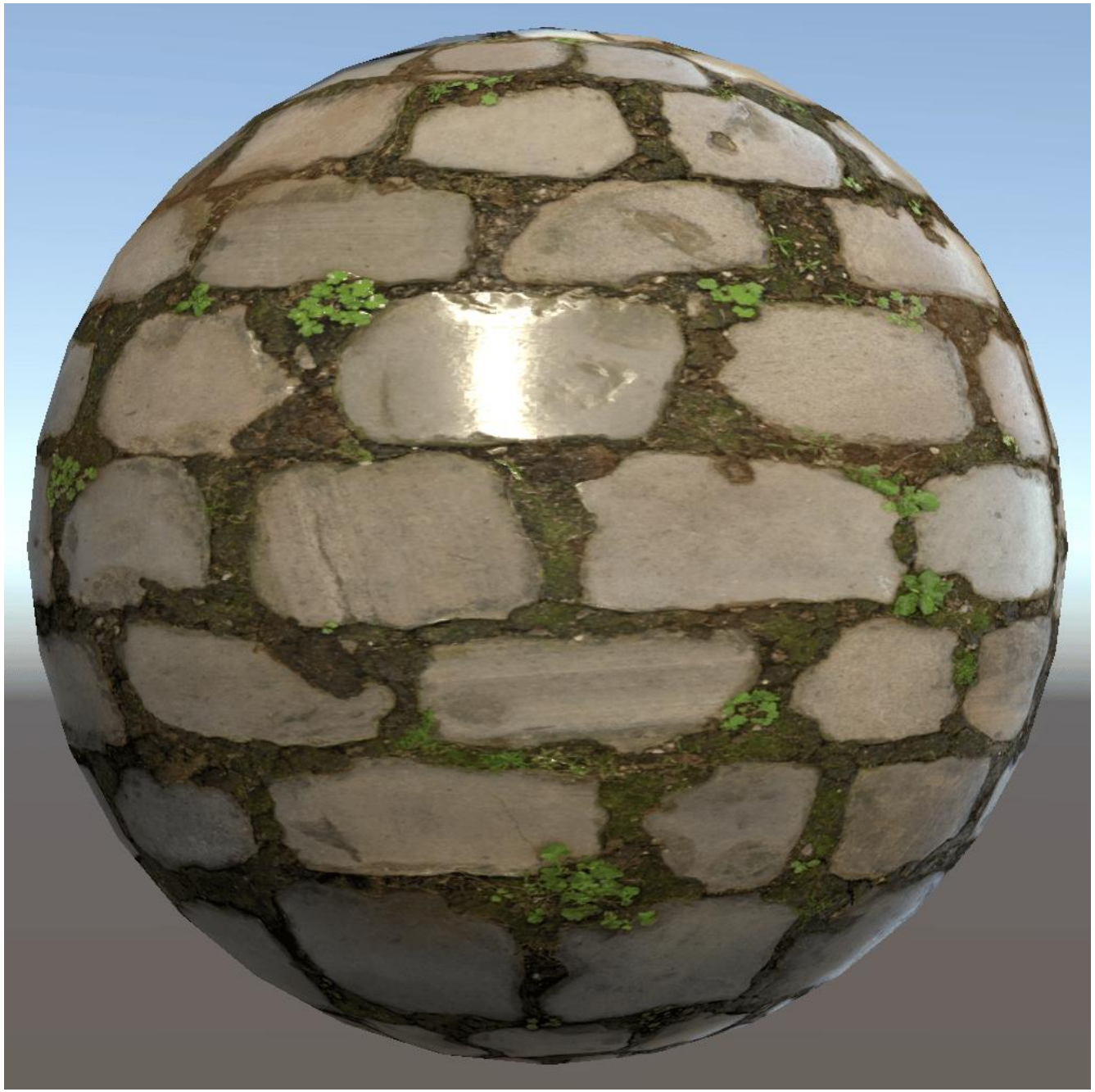
内挿による色の合成で
暗くする変化を持ち込む

その他は同じ


Sample Texture 2D
cobblestone
UV0
Texture(T2)
UV(2)
Sampler(SS)
RGBA(4)
R(1)
G(1)
B(1)
A(1)
Type
Default
Space
Tangent
One Minus
In(1)
Out(1)
Sample Texture 2D
cobblestone
UV0
Texture(T2)
UV(2)
Sampler(SS)
RGBA(4)
R(1)
G(1)
B(1)
A(1)
Type
Default
Space
Tangent
Vertex
Object Space
Position(3)
Object Space
Normal(3)
Object Space
Tangent(3)
Fragment
Ambient Occlusion(1)
Metallic(1)
Smoothness(1)
Normal (Tangent Space)(3)

完成2

- マテリアルのスライダーを操作して観察しよう



まとめ

- ノイズの概要
- 基本的なノイズ
  - シンプルノイズとPerlinノイズを試す
  - ノイズから模様(ブロック・年輪)をプロシージャルに作る
  - 3Dノイズを作る
- FBM
  - 細かさを変えたPerlinノイズを重ねてより複雑な見た目にする
  - 模様をアニメーションさせる
- 加工したFBM
  - Perlinノイズを重ねて大理石の模様を作る
- ボロノイ図
  - ボロノイノードを試す
  - パラメータを動かして、アニメーションする集光模様を実現する
  - 簡易的なステンドグラスをプロシージャルに作る
- 炎
  - 立ち上る炎をノイズから作る
- ディゾルブ
  - 物体が複雑に消える表現を作る