

High-Performance By-Example Noise using a Histogram-Preserving Blending Operator

Eric Heitz
Unity Technologies

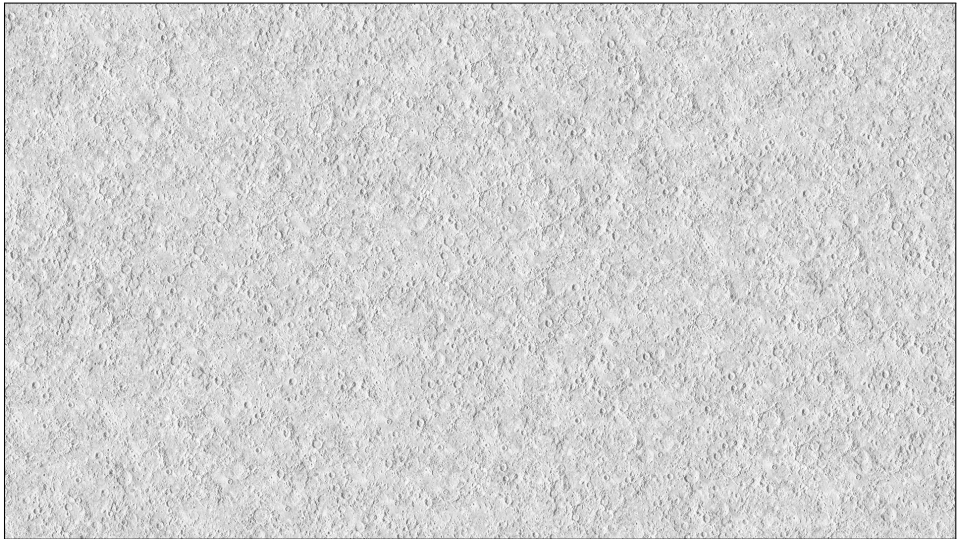
Fabrice Neyret
CNRS / Inria / U-Grenoble





Introduction

example



Introduction

example



Desired properties

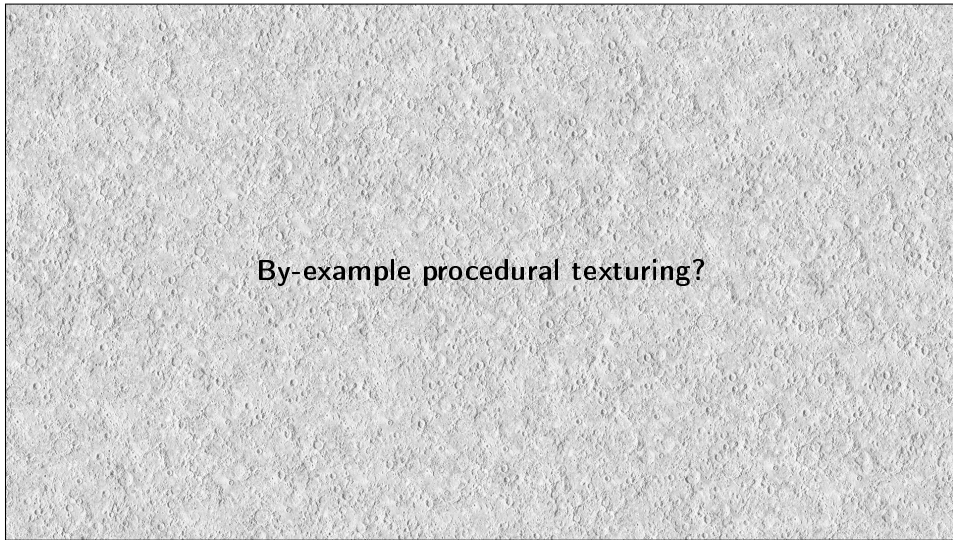
- Non-repetitive (no repeated tiles)
- Infinite
- Same appearance as example

Technical requirements

- Fast
- Small memory footprint
- On the fly (fragment shader only)

Introduction

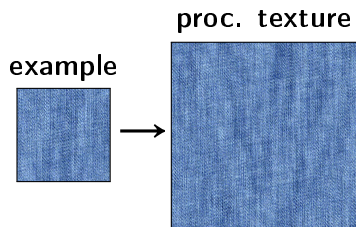
example



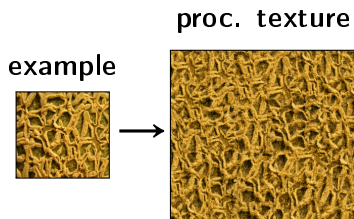
By-example procedural texturing?

Introduction

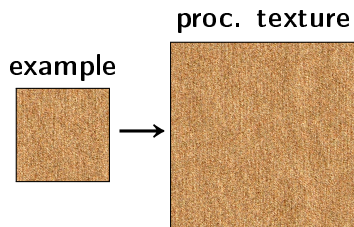
By-example procedural texturing in the academic world:



Gabor Noise by example
SIGGRAPH 2012



Local Random-Phase Noise
SIGGRAPH Asia 2014

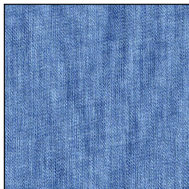


Texton Noise
CGF 2017

Introduction

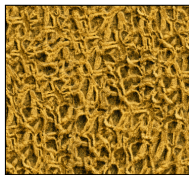
Too costly for games

20ms



Gabor Noise by example
SIGGRAPH 2012

22ms



Local Random-Phase Noise
SIGGRAPH Asia 2014

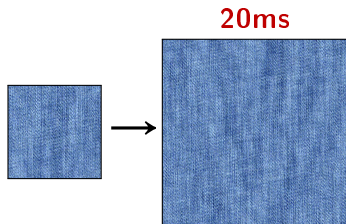
2ms



Texton Noise
CGF 2017

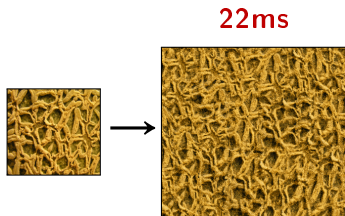
Introduction

Our method: same quality, orders of magnitude faster



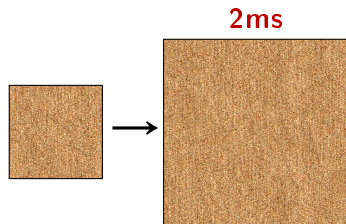
Gabor Noise by example
SIGGRAPH 2012

ours: 0.11ms
200× faster



Local Random-Phase Noise
SIGGRAPH Asia 2014

ours: 0.29ms
75× faster



Texton Noise
CGF 2017

ours: 0.11ms
20× faster

Introduction

Story

We tried the cheap low-quality approaches people use in practice.

We looked into the low-quality problem.

We fixed it.

It's going to be a journey in the world of blending.

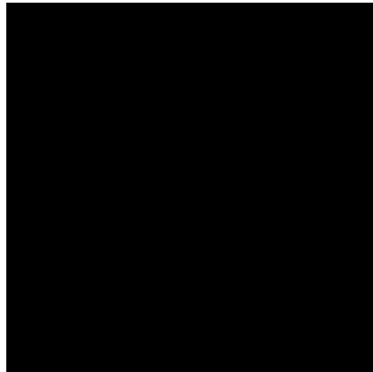
Cheap approach: texture bombing

Cheap approach: texture bombing

example



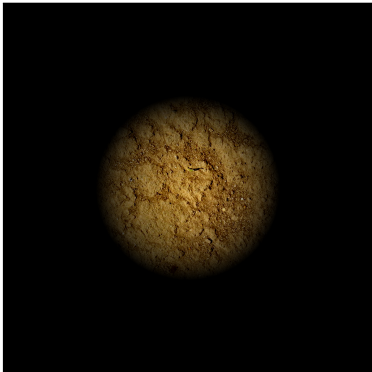
procedural texture



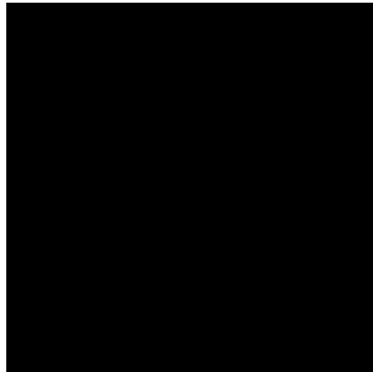
The user provides an input.

Cheap approach: texture bombing

example



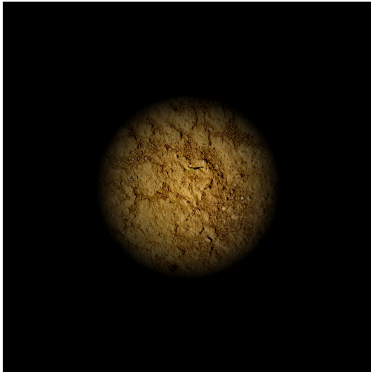
procedural texture



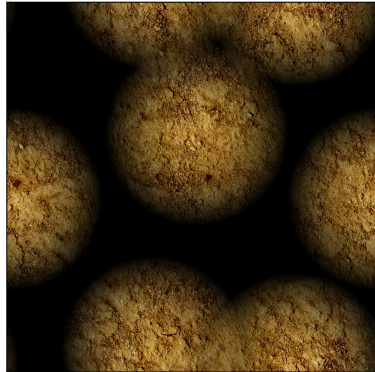
We choose random patches from the input...

Cheap approach: texture bombing

example



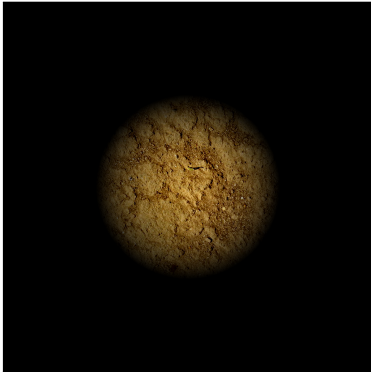
procedural texture



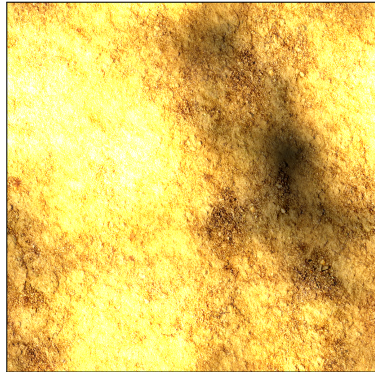
...and we splat them randomly...

Cheap approach: texture bombing

example



procedural texture



...until we get enough coverage.

Cheap approach: texture bombing

example



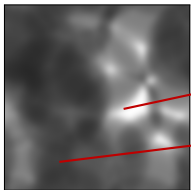
procedural texture



We divide each pixel by the density of patches.

Cheap approach: texture bombing

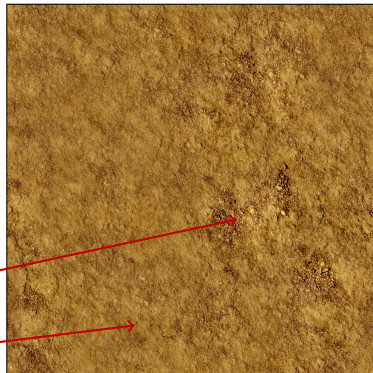
- ✓ high performance (0.2ms)
- ✗ heterogeneous contrasts
- ✗ varying number of splats/pixel



high contrast

low contrast

procedural texture



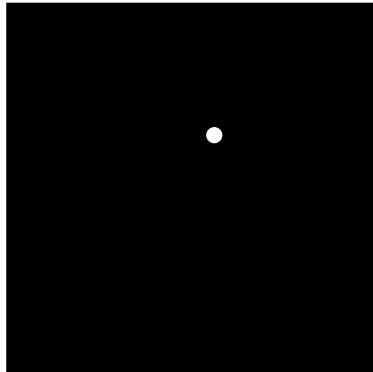
Improved cheap approach: tiling and blending

Improved cheap approach: tiling and blending

example



procedural texture



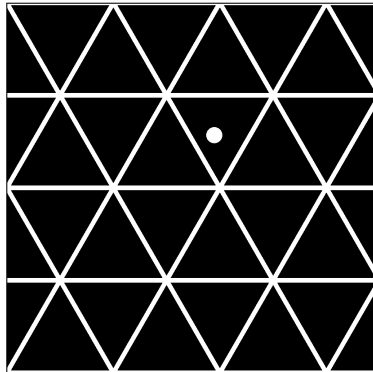
We want to evaluate the procedural texture at this point.

Improved cheap approach: tiling and blending

example



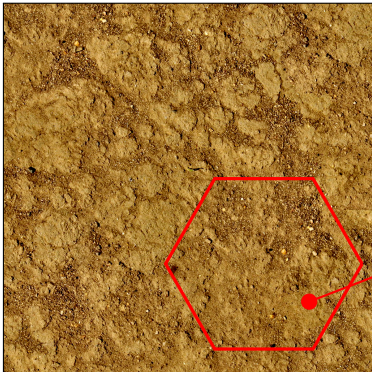
procedural texture



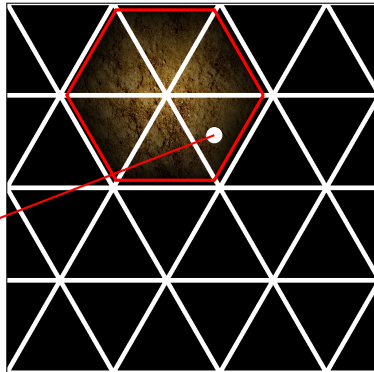
We partition the texture space on a triangle grid.

Improved cheap approach: tiling and blending

example

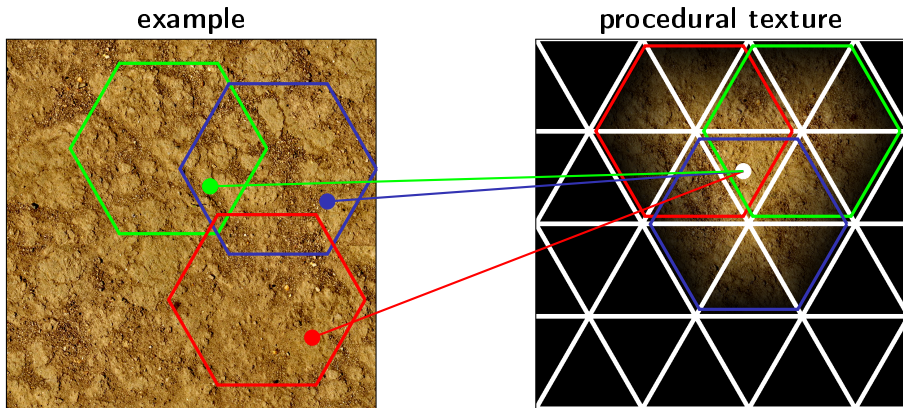


procedural texture



Each triangle vertex yields a random offset to fetch the example.

Improved cheap approach: tiling and blending



We blend the 3 fetches: $X = w_1 X_1 + w_2 X_2 + w_3 X_3$.

Improved cheap approach: tiling and blending

Fragment shader

1. Get triangle vertices and blending weights

$$V_1, V_2, V_3 \quad w_1, w_2, w_3$$

2. Compute random offsets at vertices

$$\text{offset}_i = \text{hash}(V_i)$$

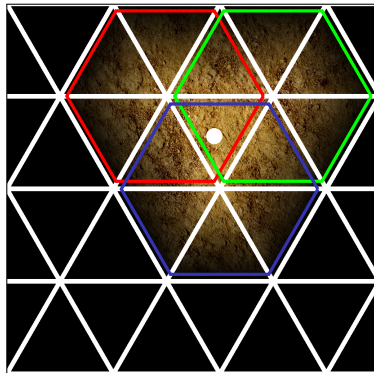
3. Fetch example with random offsets

$$X_i = \text{texture}(\text{uv} + \text{offset}_i)$$

4. Blend

$$X = w_1 X_1 + w_2 X_2 + w_3 X_3$$

procedural texture



Improved cheap approach: tiling and blending

example



procedural texture



what you expect

Improved cheap approach: tiling and blending

example



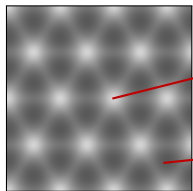
procedural texture



what you get

Improved cheap approach: tiling and blending

- ✓ high performance (0.1ms)
- ✓ constant number of splats/pixel
- ✗ still heterogeneous contrast!



high contrast

low contrast

procedural texture

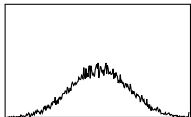
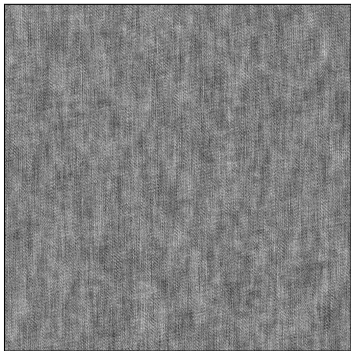


what you get

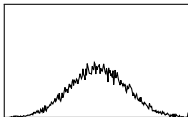
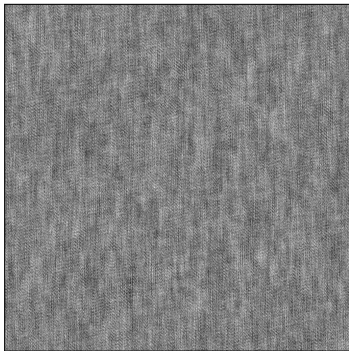
What's the problem with contrast?

What's the problem with contrast?

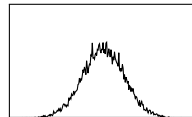
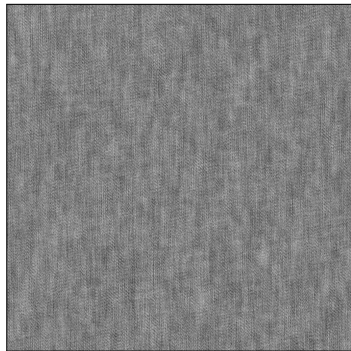
X_1



X_2

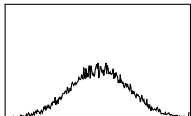
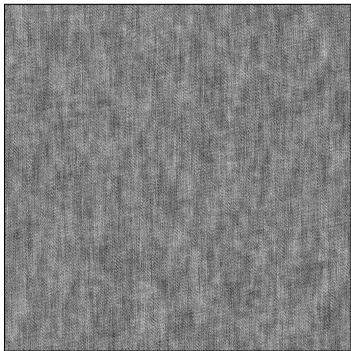


$\frac{1}{2} X_1 + \frac{1}{2} X_2$

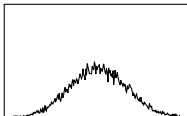
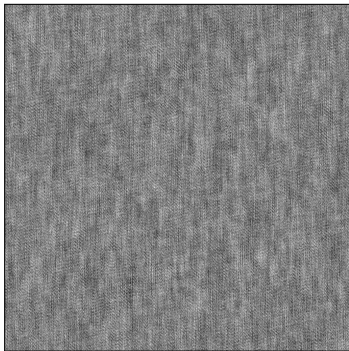


What's the problem with contrast?

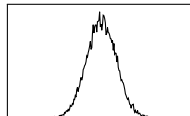
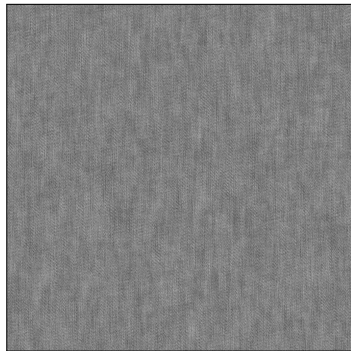
X_1



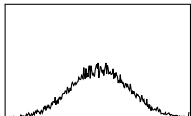
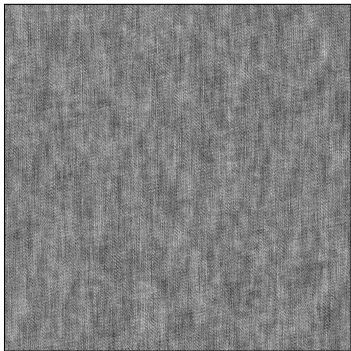
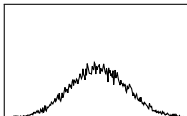
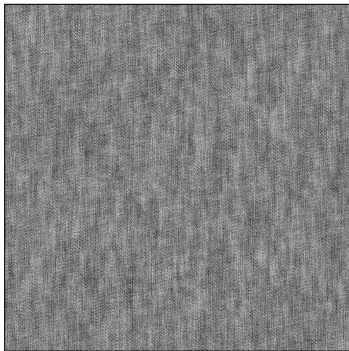
X_2



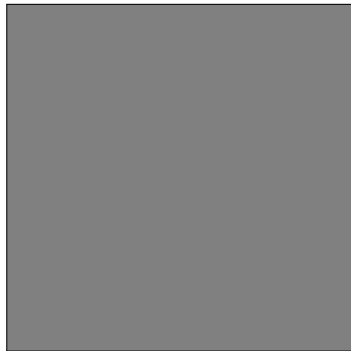
$$\frac{1}{4} \sum_{i=1}^4 X_i$$



What's the problem with contrast?

 X_1  X_2 

$$\frac{1}{1024} \sum_{i=1}^{1024} X_i$$



What's the problem with contrast?

$$\frac{1}{1024} \sum_{i=1}^{1024} X_i$$

contrast = variance

$$\text{var} \left(\frac{1}{N} \sum_{i=1}^N X_i \right) = \frac{1}{N} \text{var}(X_i)$$

Averaging reduces the variance.



Can we solve this contrast (variance) problem?

Can we solve this contrast (variance) problem?

$$X^{\text{blended}} = \sum_{i=1}^N w_i X_i \quad \leftarrow \text{linear blending}$$

$$\mathbb{E}[X^{\text{blended}}] = \mathbb{E}[X_i] \quad \checkmark$$

$$\text{var}(X^{\text{blended}}) = \underbrace{\left(\sum_{i=1}^N w_i^2 \right)}_{\text{less than 1}} \text{var}(X_i) \quad \times$$

Can we solve this contrast (variance) problem?

1. linear blending

$$X^{\text{blended}} = \frac{\sum_{i=1}^N w_i X_i}{\sqrt{\sum_{i=1}^N w_i^2}}$$

2. fix variance

$$\begin{aligned} \mathbb{E}[X^{\text{blended}}] &\neq \mathbb{E}[X_i] && \text{✗} \\ \text{var}(X^{\text{blended}}) &= \text{var}(X_i) && \text{✓} \end{aligned}$$

Can we solve this contrast (variance) problem?

1. linear blending

2. center on 0

$$X^{\text{blended}} = \frac{\sum_{i=1}^N w_i X_i - \mathbb{E}[X_i]}{\sqrt{\sum_{i=1}^N w_i^2}} + \mathbb{E}[X_i]$$

4. center on expectation

3. fix variance

$$\begin{aligned}\mathbb{E}[X^{\text{blended}}] &= \mathbb{E}[X_i] && \checkmark \\ \text{var}(X^{\text{blended}}) &= \text{var}(X_i) && \checkmark\end{aligned}$$

Can we solve this contrast (variance) problem?

1. linear blending

2. center on 0

$$X^{\text{blended}} = \frac{\sum_{i=1}^N w_i X_i - \mathbb{E}[X_i]}{\sqrt{\sum_{i=1}^N w_i^2}} + \mathbb{E}[X_i]$$

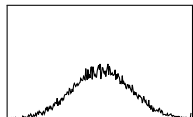
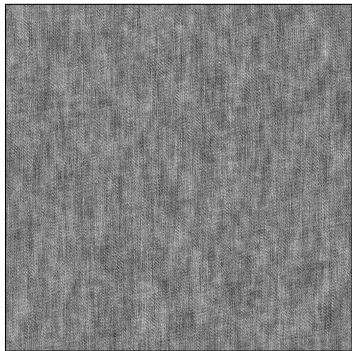
4. center on expectation

3. fix variance

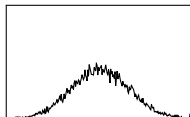
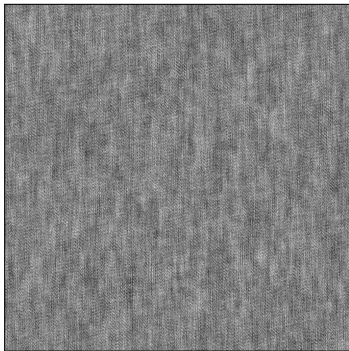
This is a **variance-preserving blending** operator.

Can we solve this contrast (variance) problem?

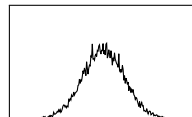
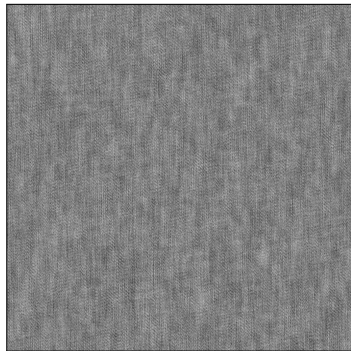
X_1



X_2

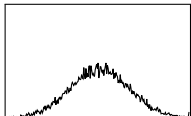
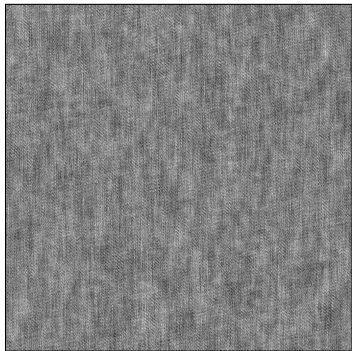


$\frac{1}{2} X_1 + \frac{1}{2} X_2$

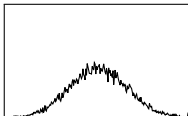
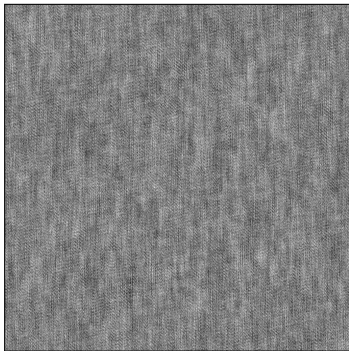


Can we solve this contrast (variance) problem?

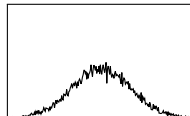
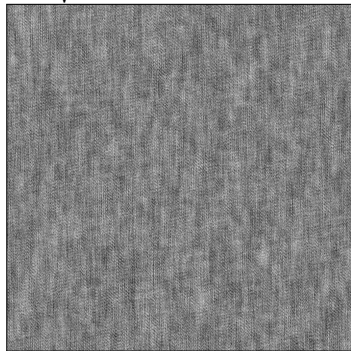
X_1



X_2



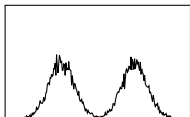
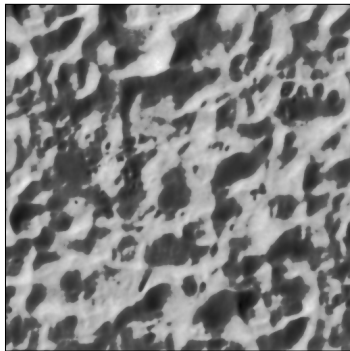
$$\frac{\frac{1}{2} X_1 + \frac{1}{2} X_2 - \mathbb{E}[X]}{\sqrt{\frac{1}{2^2} + \frac{1}{2^2}}} + \mathbb{E}[X]$$



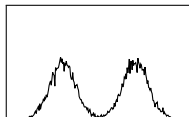
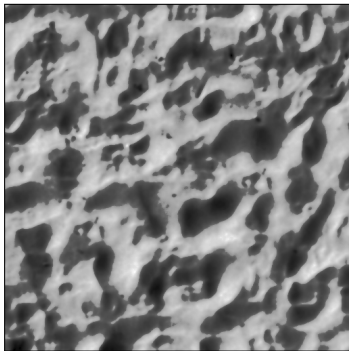
Is it really just about contrast?

Is it really just about contrast?

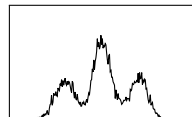
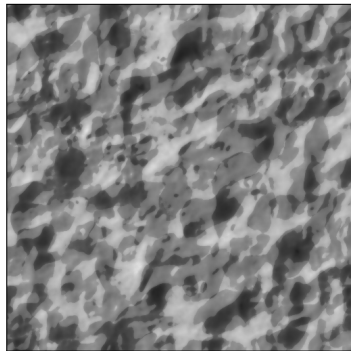
X_1



X_2

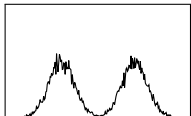
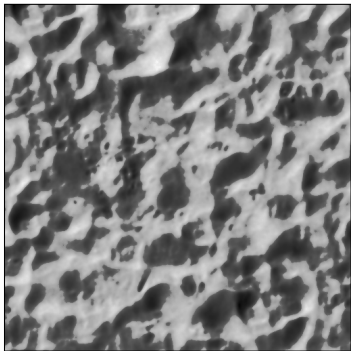


$\frac{1}{2} X_1 + \frac{1}{2} X_2$

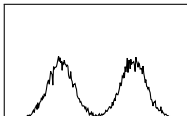
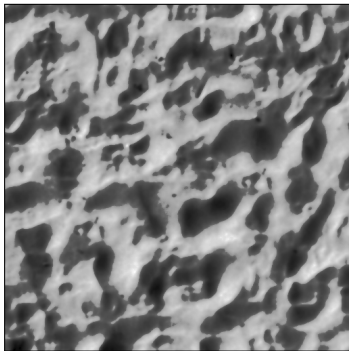


Is it really just about contrast?

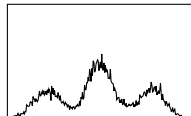
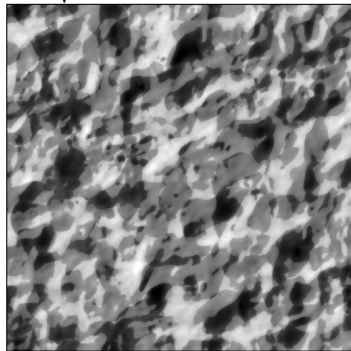
X_1



X_2

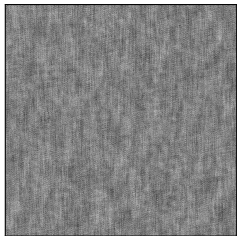


$$\frac{\frac{1}{2} X_1 + \frac{1}{2} X_2 - \mathbb{E}[X]}{\sqrt{\frac{1}{2^2} + \frac{1}{2^2}}} + \mathbb{E}[X]$$

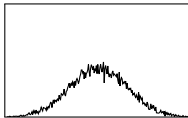
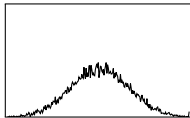
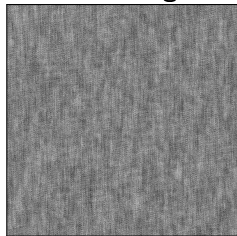


Is it really just about contrast?

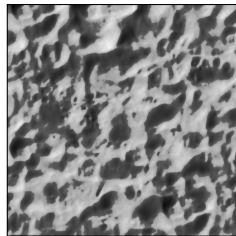
example



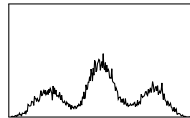
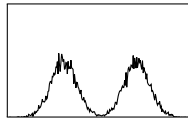
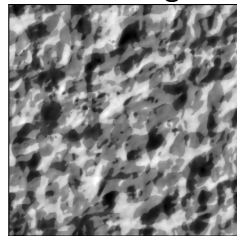
variance-preserving
blending



example



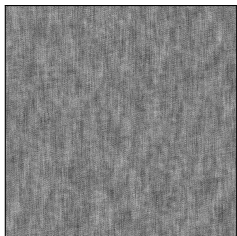
variance-preserving
blending



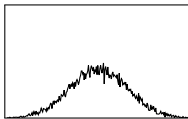
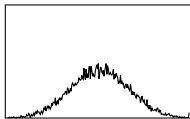
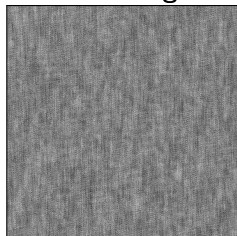
Answer: no, it's not just about contrast. It's about the full histogram.

Is it really just about contrast?

example

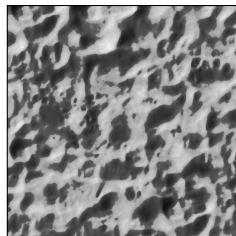


variance-preserving
blending

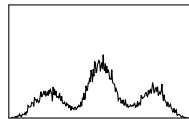
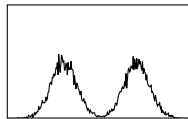
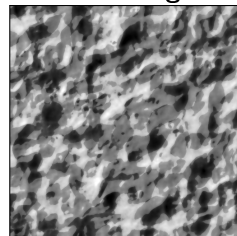


Special case: **Gaussian** histogram

example



variance-preserving
blending

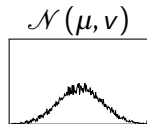


General case: **arbitrary** histogram

Is it really just about contrast?

Special case:

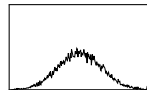
Independent **Gaussian** random variables: G_1, \dots, G_n \longrightarrow



Blending weights: w_1, \dots, w_n

Variance-preserving blending = histogram-preserving blending

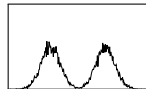
$$\frac{w_1 G_1 + \dots + w_n G_n - \mathbb{E}[G]}{\sqrt{w_1^2 + \dots + w_n^2}} + \mathbb{E}[G] \longrightarrow$$



Is it really just about contrast?

General case:

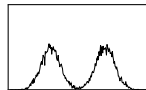
Independent **arbitrary** random variables: X_1, \dots, X_n \longrightarrow



Blending weights: W_1, \dots, W_n

Histogram-preserving blending

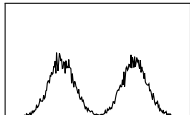
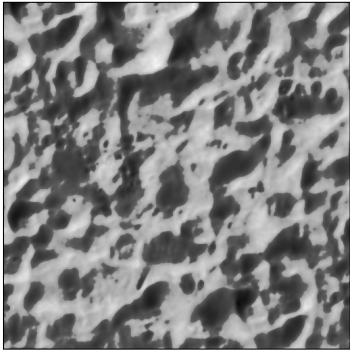
?



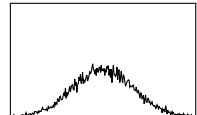
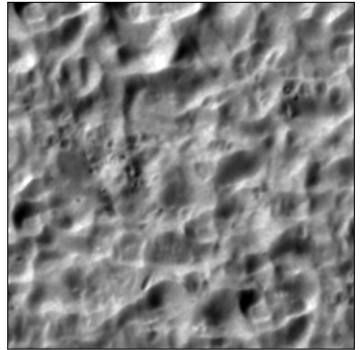
Histogram-preserving blending

Histogram-preserving blending

non Gaussian, hard to blend



Gaussian, simple to blend

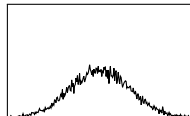
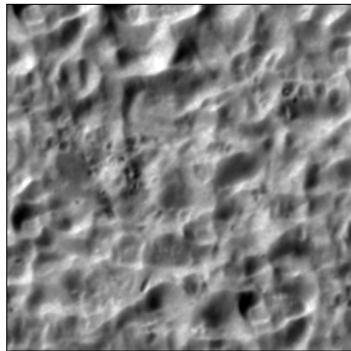
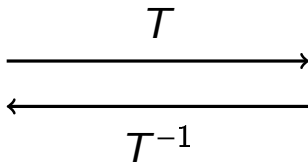
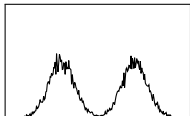
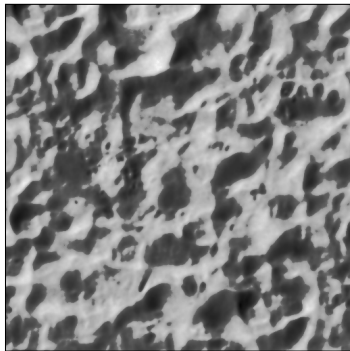


Histogram-preserving blending

$$X = T^{-1}(G)$$

histogram transformation

$$G = T(X)$$

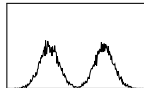


T and T^{-1} are 1D functions
precomputed and stored in LUTs

Histogram-preserving blending

Input:

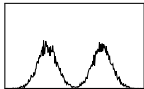
$$X_1, \dots, X_n$$



Histogram-preserving blending

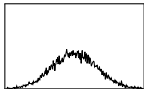
Input:

$$X_1, \dots, X_n$$



1. “Gaussianize” (fetch LUT):

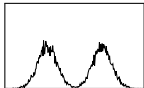
$$T(X_1), \dots, T(X_n)$$



Histogram-preserving blending

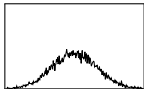
Input:

$$X_1, \dots, X_n$$



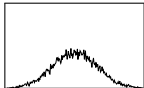
1. “Gaussianize” (fetch LUT):

$$T(X_1), \dots, T(X_n)$$



2. Variance-preserving blending:

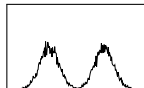
$$\frac{w_1 T(X_1) + \dots + w_n T(X_n)}{\sqrt{w_1^2 + \dots + w_n^2}}$$



Histogram-preserving blending

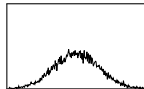
Input:

$$X_1, \dots, X_n$$



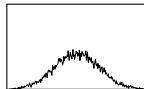
1. “Gaussianize” (fetch LUT):

$$T(X_1), \dots, T(X_n)$$



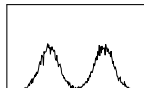
2. Variance-preserving blending:

$$\frac{w_1 T(X_1) + \dots + w_n T(X_n)}{\sqrt{w_1^2 + \dots + w_n^2}}$$



3. “unGaussianize” (fetch LUT):

$$T^{-1} \left(\frac{w_1 T(X_1) + \dots + w_n T(X_n)}{\sqrt{w_1^2 + \dots + w_n^2}} \right)$$

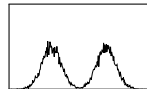


Histogram-preserving blending

General case:

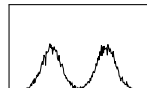
Independent **arbitrary** random variables: X_1, \dots, X_n

Blending weights: w_1, \dots, w_n



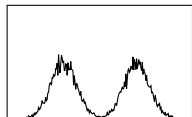
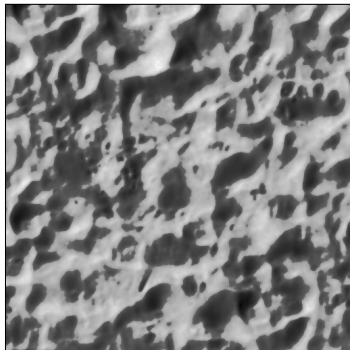
Histogram-preserving blending:

$$T^{-1} \left(\frac{w_1 T(X_1) + \dots + w_n T(X_n)}{\sqrt{w_1^2 + \dots + w_n^2}} \right)$$

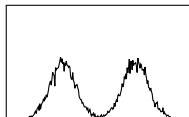
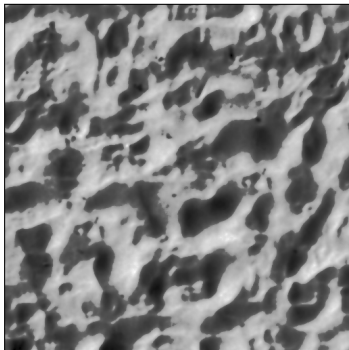


Histogram-preserving blending

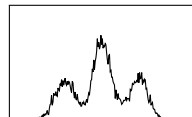
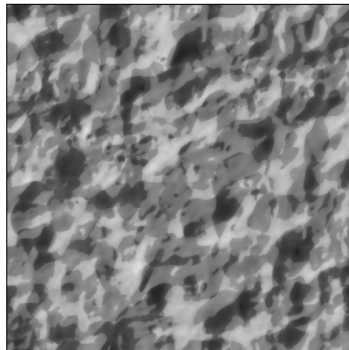
X_1



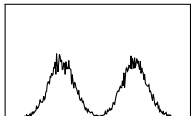
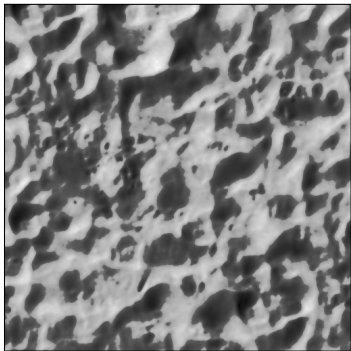
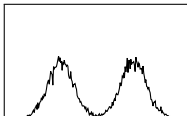
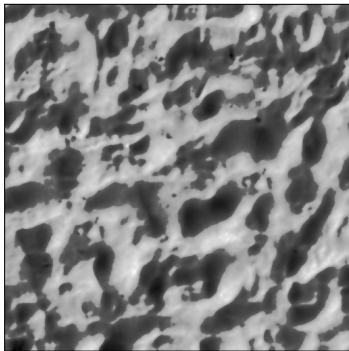
X_2



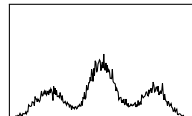
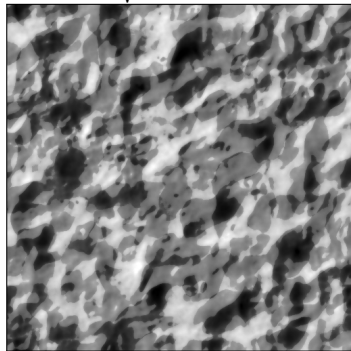
$\frac{1}{2} X_1 + \frac{1}{2} X_2$



Histogram-preserving blending

 X_1  X_2 

$$\frac{\frac{1}{2} X_1 + \frac{1}{2} X_2}{\sqrt{\frac{1}{2^2} + \frac{1}{2^2}}}$$

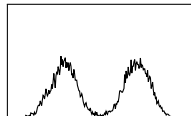
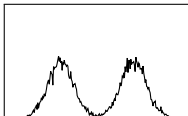
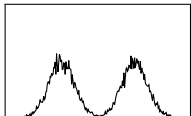
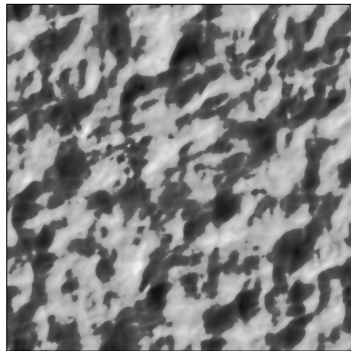
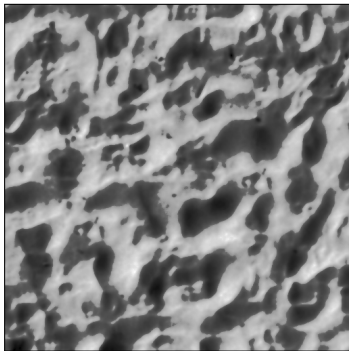
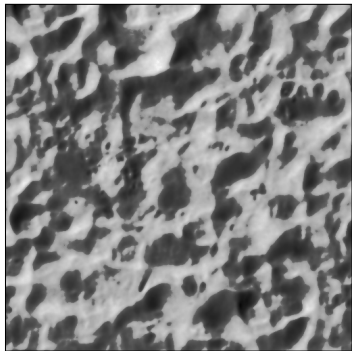


Histogram-preserving blending

$$T^{-1} \left(\frac{\frac{1}{2} T(X_1) + \frac{1}{2} T(X_2)}{\sqrt{\frac{1}{2^2} + \frac{1}{2^2}}} \right)$$

X_1

X_2

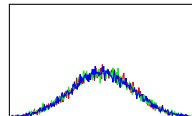
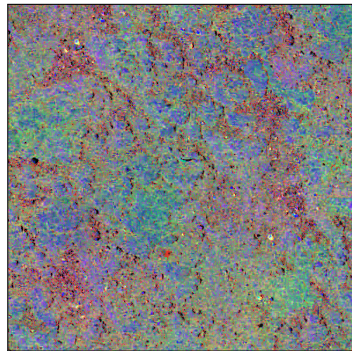
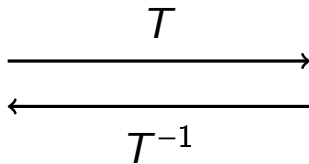
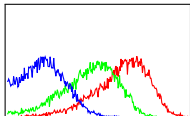


Histogram-preserving blending

$$X = T^{-1}(G)$$

3D histogram transformation

$$G = T(X)$$

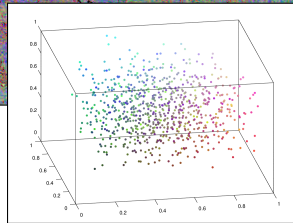
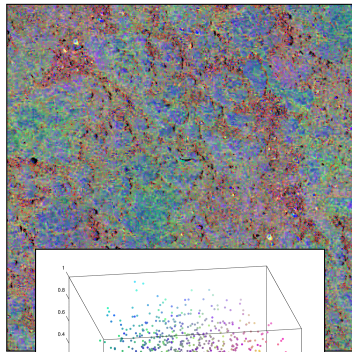
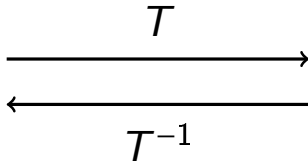
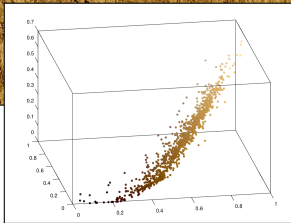
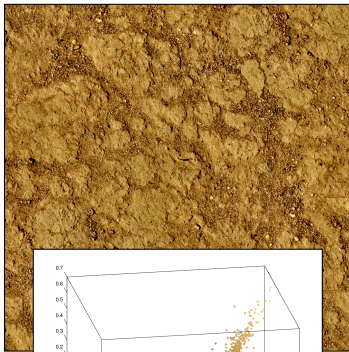


Histogram-preserving blending

$$X = T^{-1}(G)$$

3D histogram transformation

$$G = T(X)$$

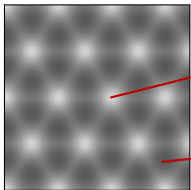


T and T^{-1} are 3D functions that map the point clouds

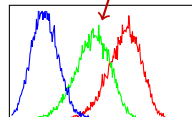
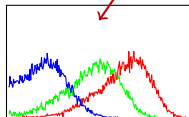
Our procedural texturing method

Our procedural texturing method

- ✓ high performance (0.29ms)
- ✓ constant number of splats/pixel
- ✗ heterogeneous contrast
- ✗ other issues (ghosting, wrong colors)



procedural texture
(linear blending)

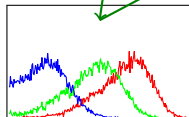
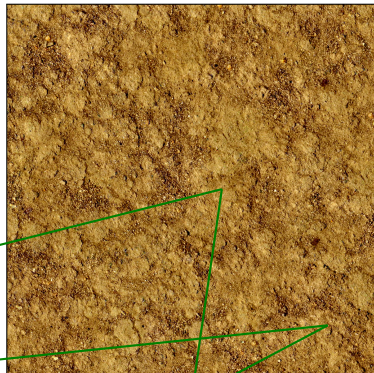


Performance at 1920×1080 on a 980 Geforce GTX

Our procedural texturing method

- ✓ high performance (0.29ms)
- ✓ constant number of splats/pixel
- ✓ homogeneous contrast
- ✓ prevents other issues

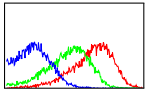
procedural texture
(histogram-preserving blending)



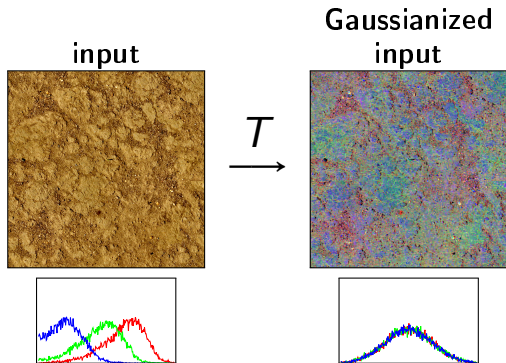
Performance at 1920×1080 on a 980 Geforce GTX

Our procedural texturing method

input

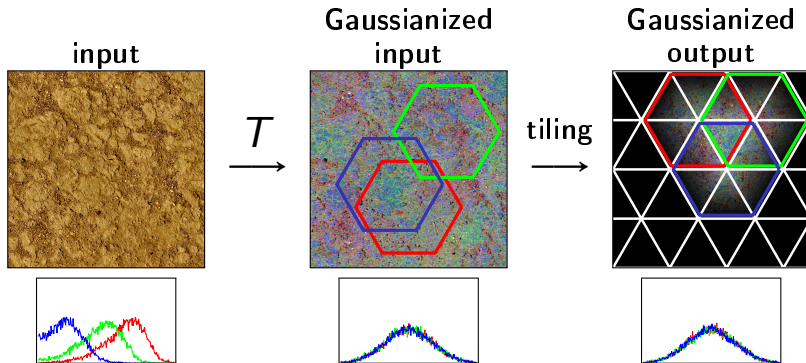


Our procedural texturing method



step 1
(precomputed)

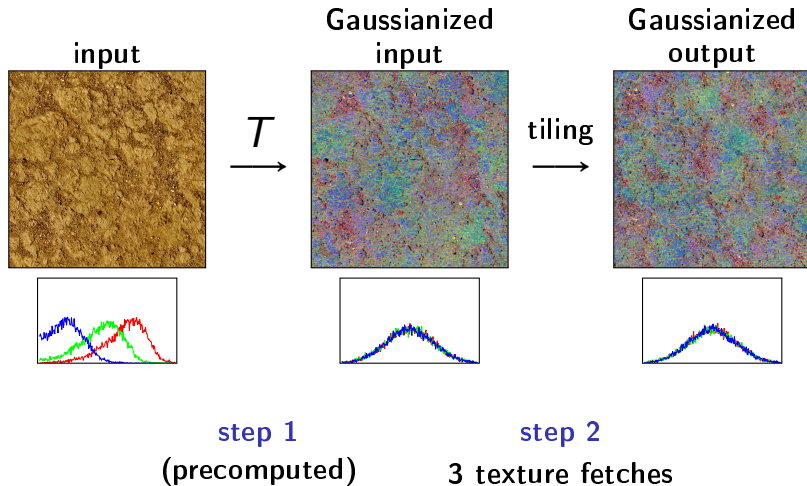
Our procedural texturing method



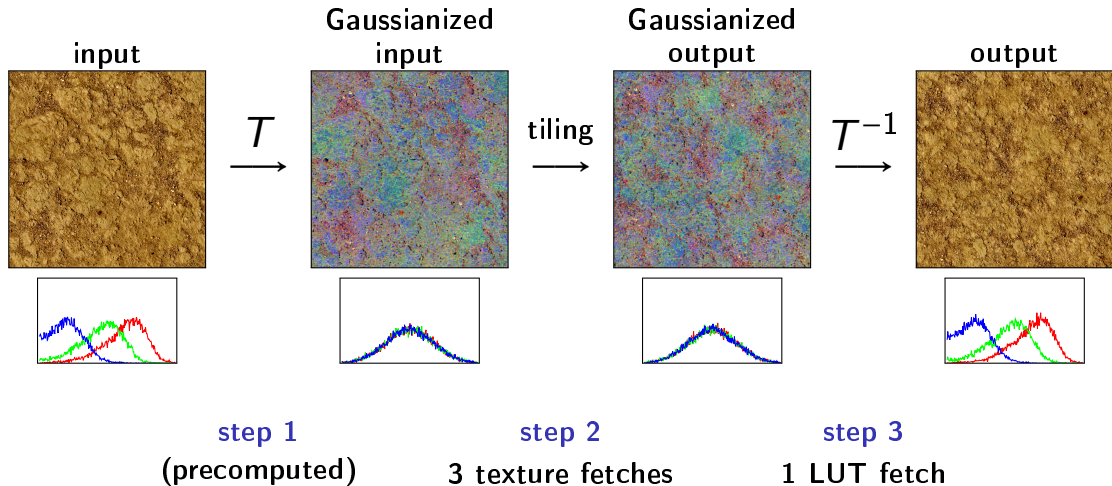
step 1
(precomputed)

step 2
3 texture fetches

Our procedural texturing method



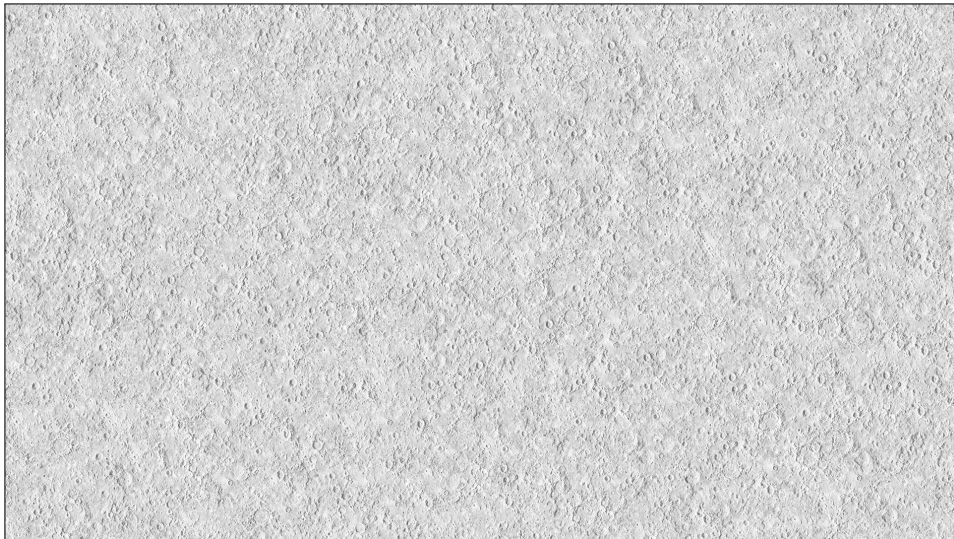
Our procedural texturing method



Our procedural texturing method

procedural texture, 0.29ms at 1920×1080 on a 980 Geforce GTX

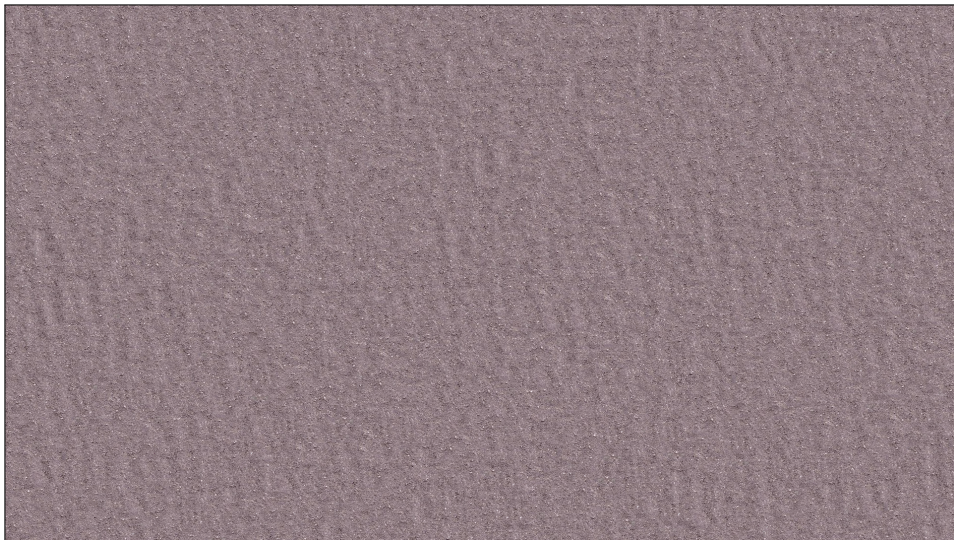
input
 256^2



Our procedural texturing method

procedural texture, 0.29ms at 1920×1080 on a 980 Geforce GTX

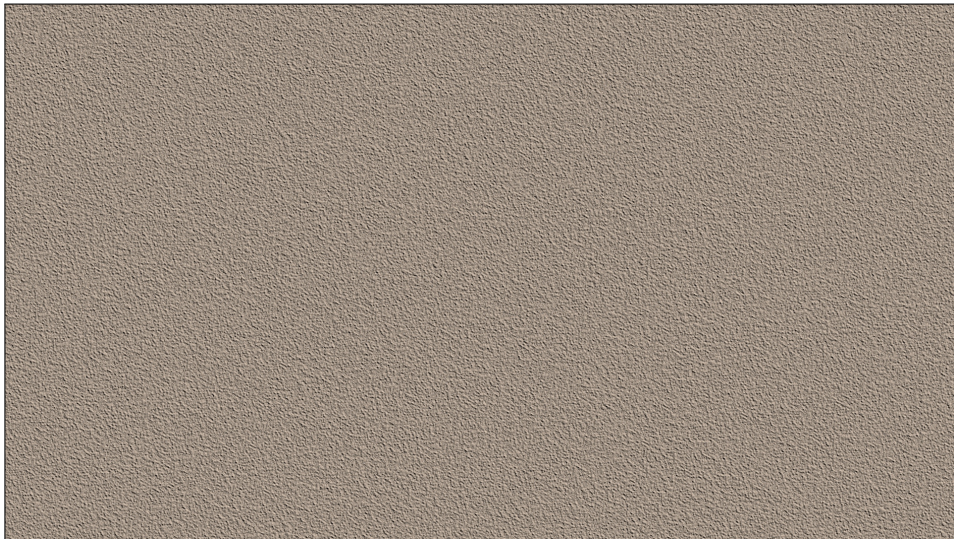
input
 256^2



Our procedural texturing method

procedural texture, 0.29ms at 1920×1080 on a 980 Geforce GTX

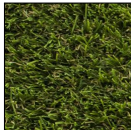
input
 256^2



Our procedural texturing method

procedural texture, 0.29ms at 1920×1080 on a 980 Geforce GTX

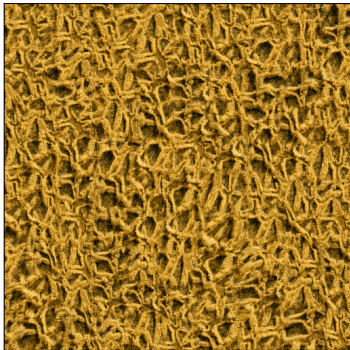
input
 256^2



Our procedural texturing method

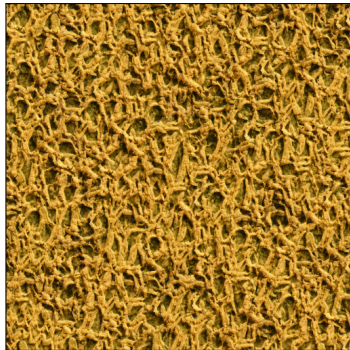
[LRPN2014]

analytic + texture fetches
22ms



our result

3 texture + 1 LUT fetches
0.29ms



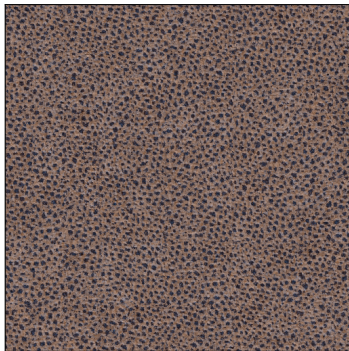
example



Our procedural texturing method

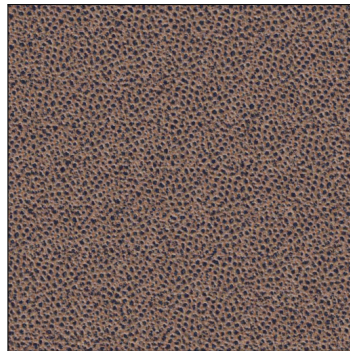
[LRPN2014]

analytic + texture fetches
22ms

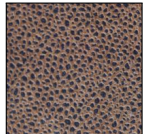


our result

3 texture + 1 LUT fetches
0.29ms



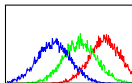
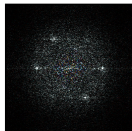
example



Our procedural texturing method

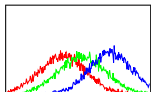
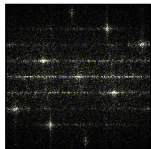
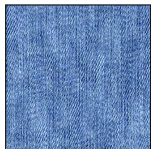
procedural texture, ~~0.29ms~~ 0.11ms at 1920×1080 on a 980 Geforce GTX

Gaussian
example
 256^2

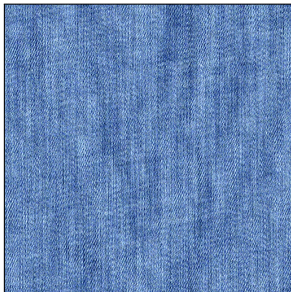


Our procedural texturing method

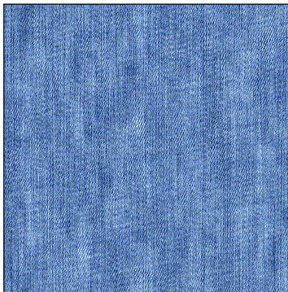
Gaussian
example
 256^2



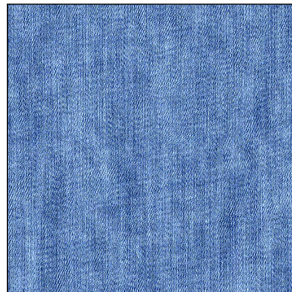
[GNBE2012]
analytic calculations
20ms



[TextonNoise2017]
30 texture fetches
2.3ms

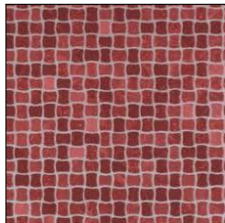


our result
3 texture fetches
0.11ms

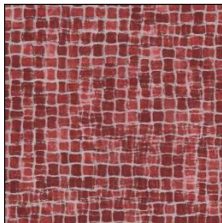


Our procedural texturing method

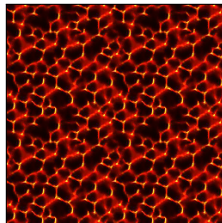
example



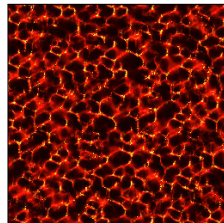
our result



example



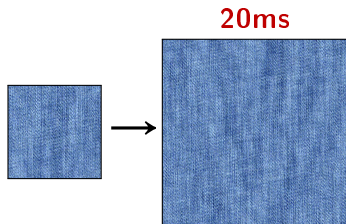
our result



Same failure cases as previous work: strong patterns and spatial correlations.

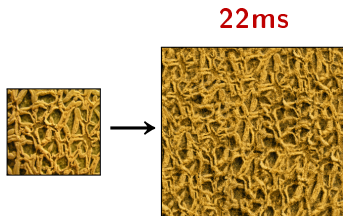
Our procedural texturing method

Our method: same quality, orders of magnitude faster



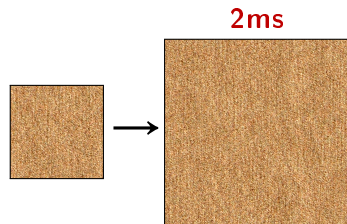
Gabor Noise by example
SIGGRAPH 2012

ours: 0.11ms
200× faster



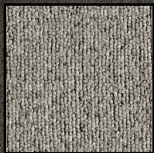
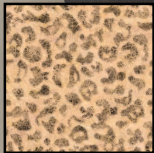
Local Random-Phase Noise
SIGGRAPH Asia 2014

ours: 0.29ms
75× faster



Texton Noise
CGF 2017

ours: 0.11ms
20× faster



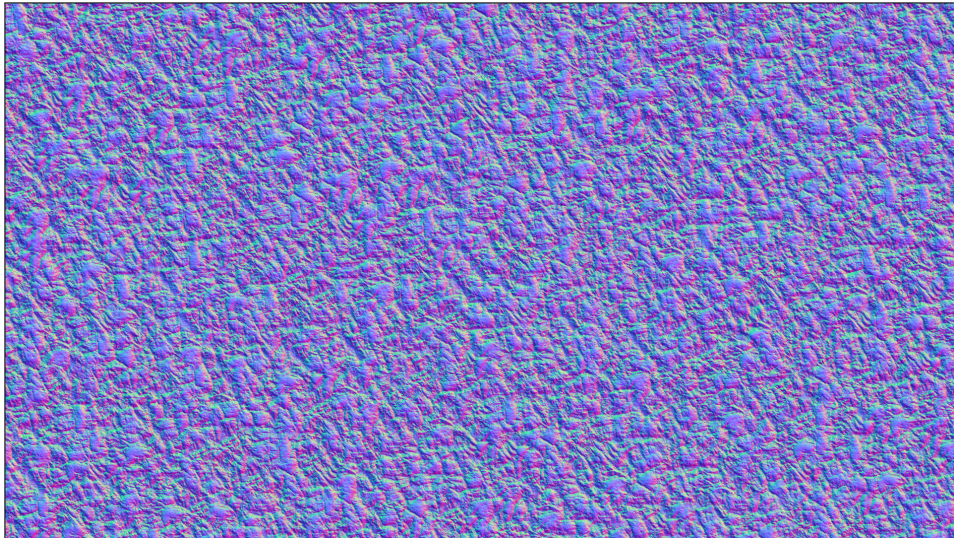
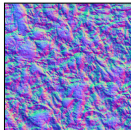
Plug-and-play fragment shader
0.29ms at 1920×1080 on a 980 Geforce GTX

Other applications of histogram-preserving blending

Other applications of histogram-preserving blending

procedural normal map

input
 256^2



Other applications of histogram-preserving blending

no normal map



tilled normal map



procedural normal map

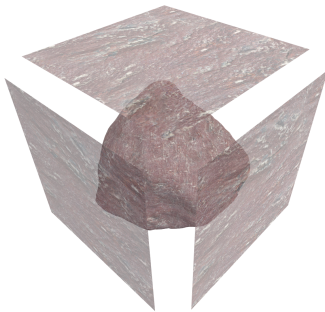


Other applications of histogram-preserving blending

texture



triplanar mapping...



...with **linear blending**



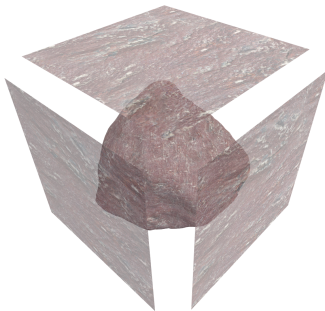
Application: texturing uv-free meshes with triplanar mapping.

Other applications of histogram-preserving blending

texture



triplanar mapping...



...with histogram-
preserving blending



Application: texturing uv-free meshes with triplanar mapping.

Other applications of histogram-preserving blending

Hashed Alpha Testing

Chris Wyman*
NVIDIA

Morgan McGuire
NVIDIA & Williams College



Figure 1: Hashed alpha testing avoids alpha-mapped geometry disappearing in the distance and reduces correlations in multisample alpha-to-coverage. We show a bearded man with (a) traditional alpha testing, (b) traditional, hardware-accelerated alpha-to-coverage, (c) hashed alpha testing, and (d) hashed alpha-to-coverage. Insets show the same geometry from further away, enlarged to better depict variations.

Wyman and McGuire

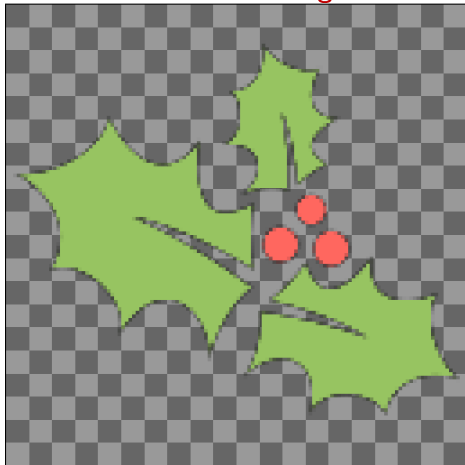
I3D 2017

Specialized solution for histogram-preserving blending with **2 uniform random numbers**.
Ours: **general** solution for histogram-preserving blending **N arbitrary inputs**.

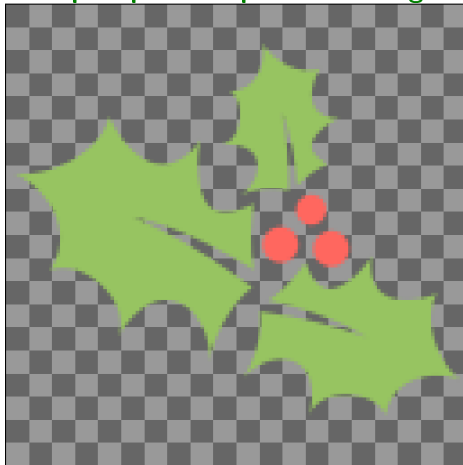
The importance of getting blending right

The importance of getting blending right

linear blending



alpha-premultiplied blending



Linearly interpolating/blending RGBA images introduces black borders.

The importance of getting blending right

Blending an HDR color into a U8 Buffer

Blogpost by Alan Wolfe 'demofox', July 2018



Problem: wrong exposition

Guilty: **blending**

Solution: **premultiplied blending**

The importance of getting blending right

The team color problem

Blogpost by Ben Golus, July 2018



Problem: color leaks

Guilty: blending

Solution: premultiplied blending

The importance of getting blending right



Marc Olano

@olanom

And yet again, premultiplied alpha is the answer. Premultiplied alpha is always the answer

The importance of getting blending right

Common wisdom

Whenever you blend **weighted** data, use **premultiplied blending**.

This prevents **leaking and mixing with undefined values**.

The importance of getting blending right

Common wisdom

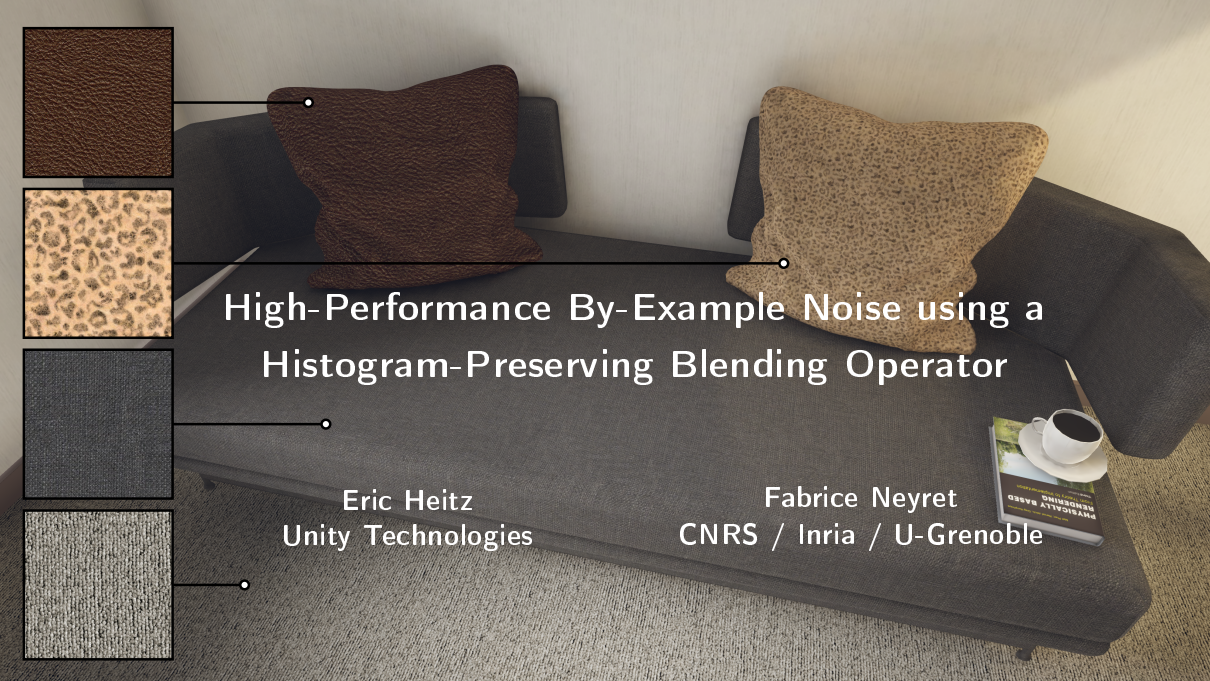
Whenever you blend **weighted** data, use **premultiplied blending**.

This prevents **leaking and mixing with undefined values**.

Main takeaway of this paper

Whenever you blend data chosen **randomly**, use **histogram-preserving blending**.

This prevents **lowering contrast, ghosting and wrong data to appear**.



High-Performance By-Example Noise using a Histogram-Preserving Blending Operator

Eric Heitz
Unity Technologies

Fabrice Neyret
CNRS / Inria / U-Grenoble