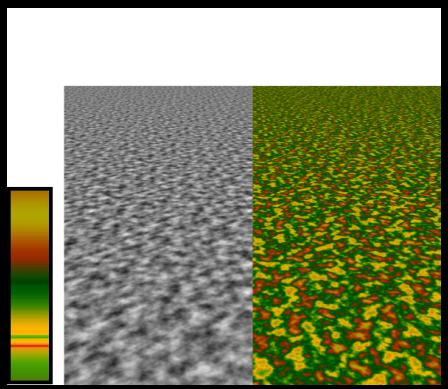
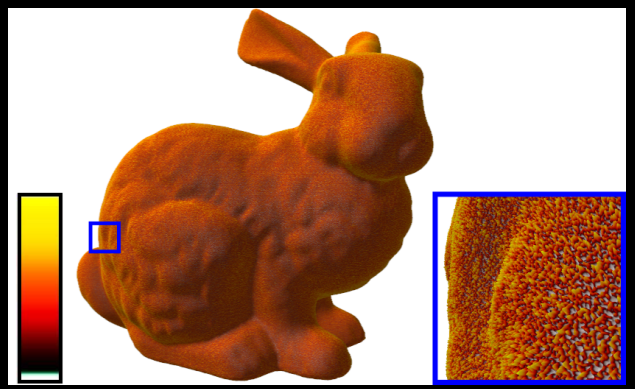
Filtering Color Mapped Textures and Surfaces

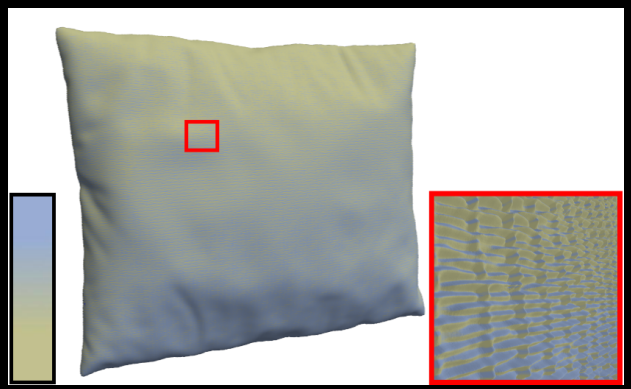
I3D 2013

Eric Heitz Derek Nowrouzezahrai Pierre Poulin Fabrice Neyret

INRIA–LJK (Université de Grenoble and CNRS)
LIGUM, Dept. I.R.O., Université de Montréal



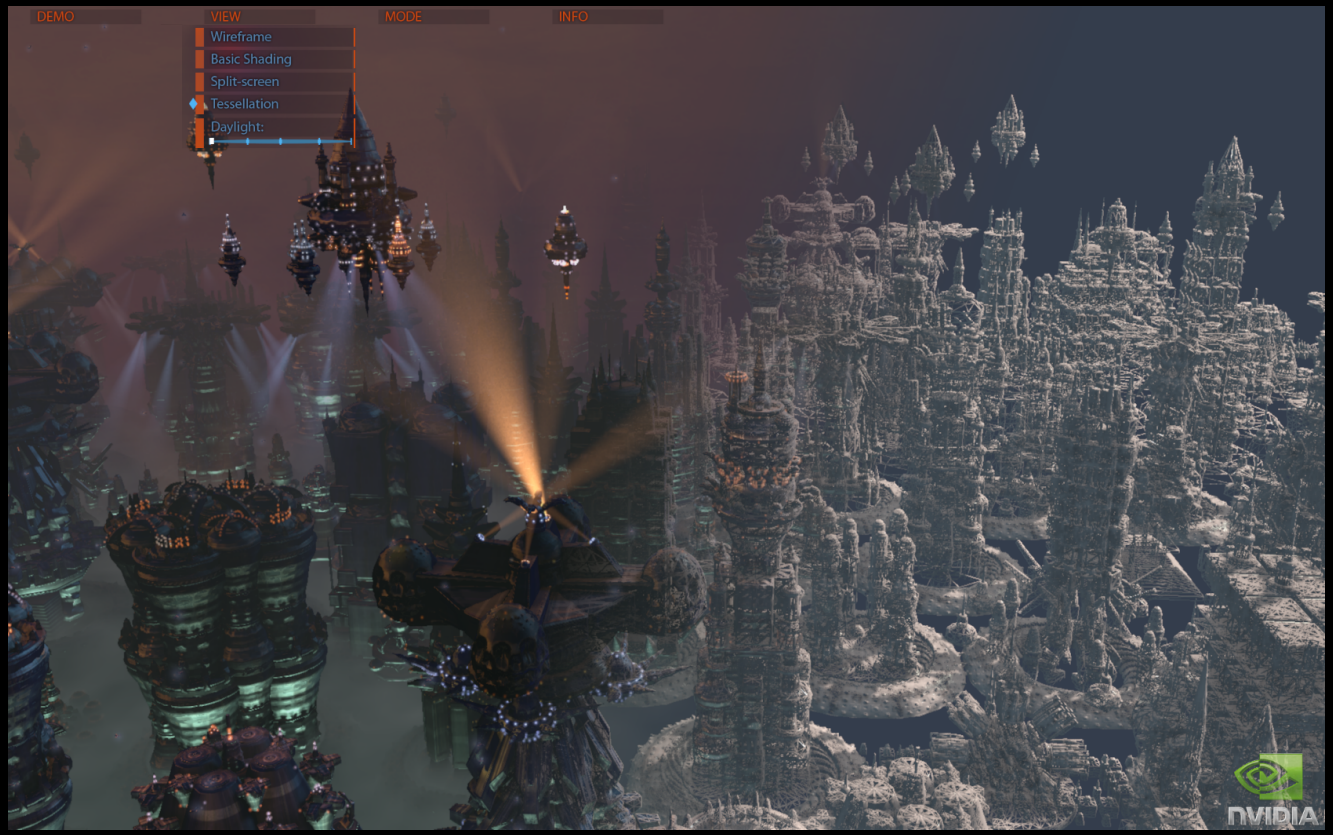




Introduction

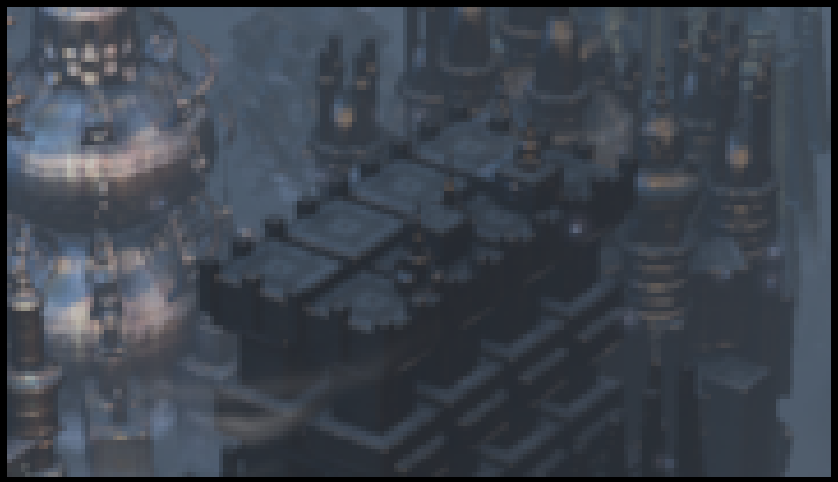
NVidia's Endless City Demo

- real-time
- amazing content and details



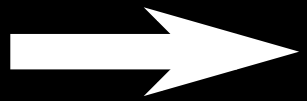
Introduction

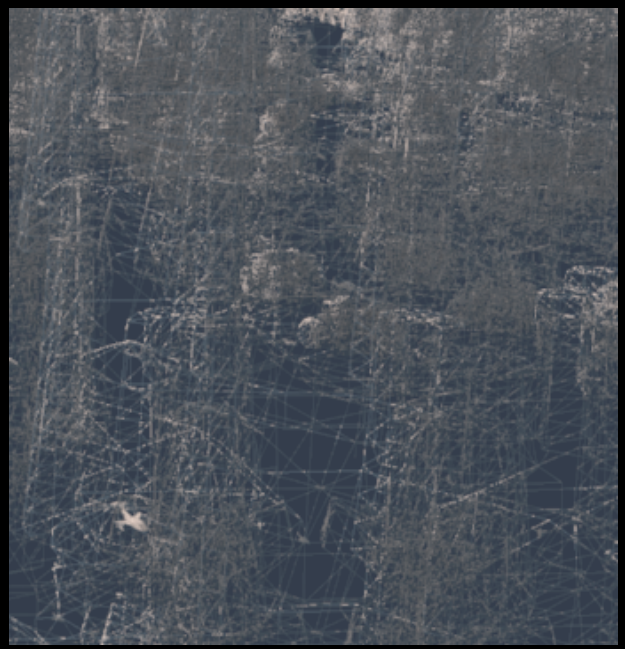
Level of detail transition



Introduction

Managing geometry level of detail is important... but is not enough

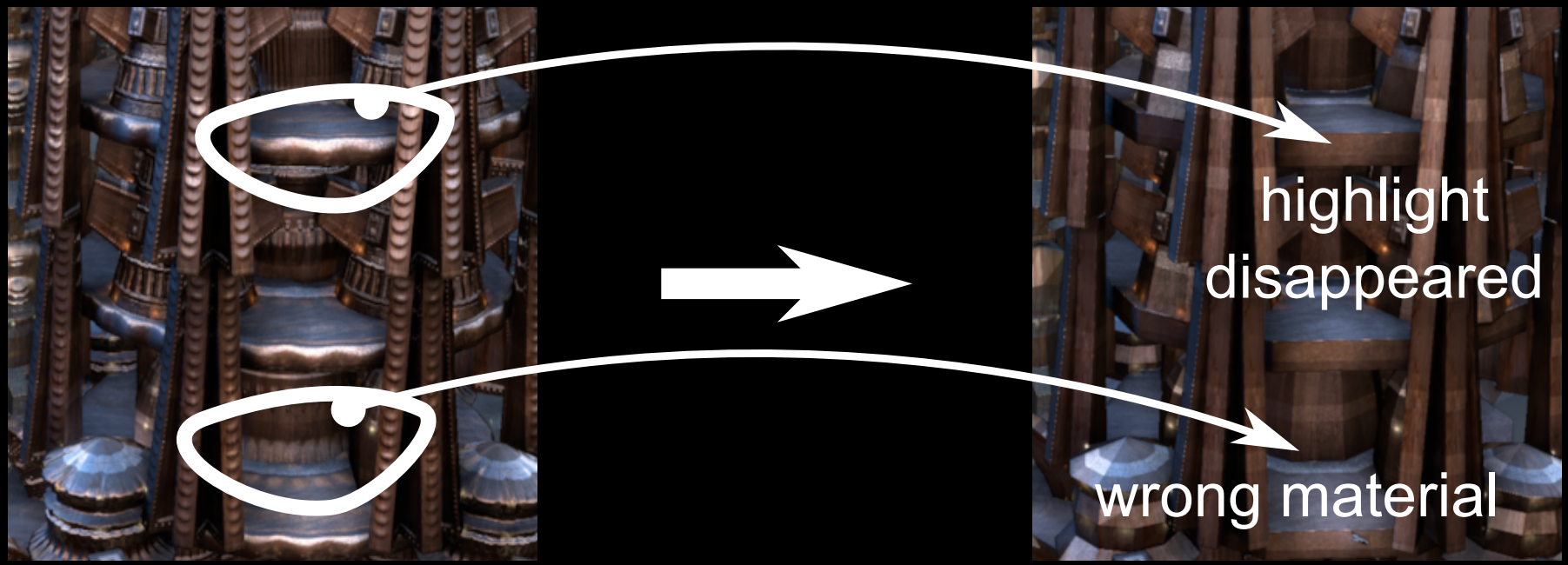




Introduction

Level of detail should also preserve appearance

- material
- BRDF
- normals
- visibility

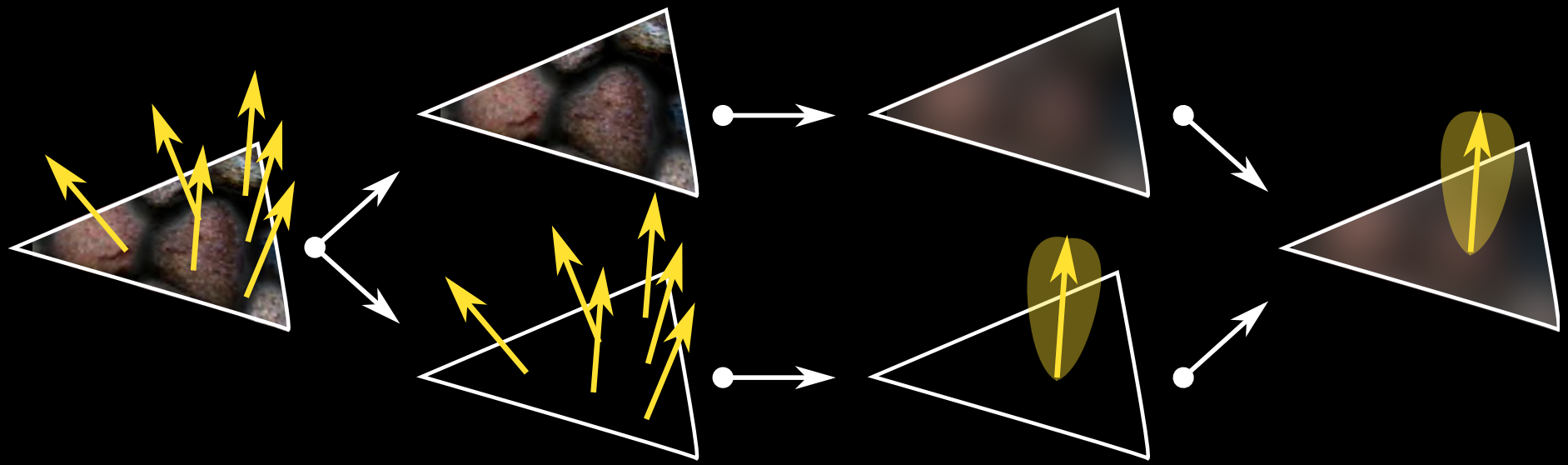


Introduction

Current CG model: coarse geometry + textures

$\rightarrow$ visibility, curvature and orientation are constant within triangles

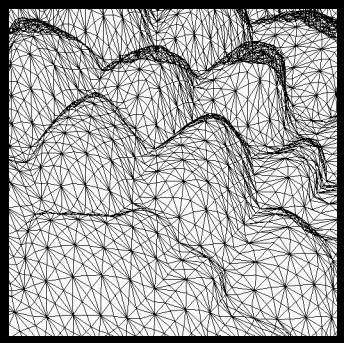
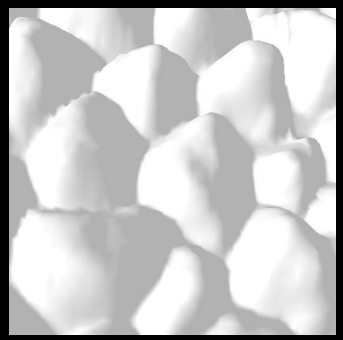
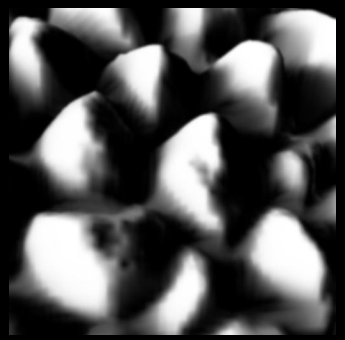
$\rightarrow$ textures can be filtered separately



separate colors and normals filtering

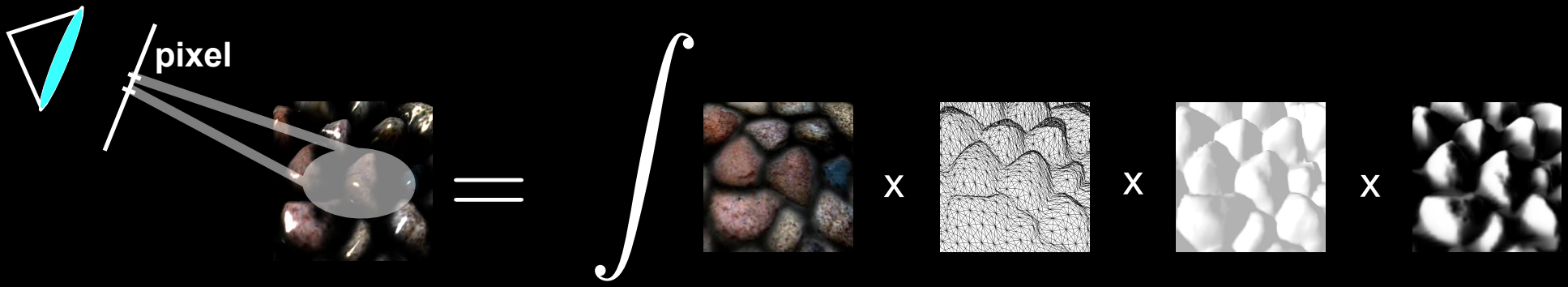
Introduction

What if geometry matches up texture resolution?

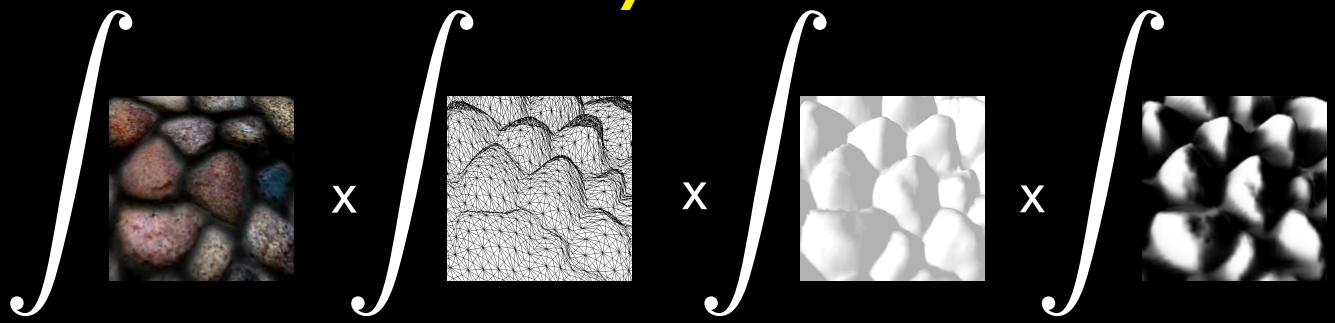
color		geometry		shading		visibility
	x		x		x	
	=		=		=	

Introduction

physically correct



$\neq$



physically wrong

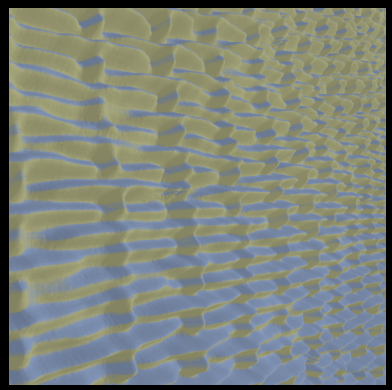
energy conservation violated

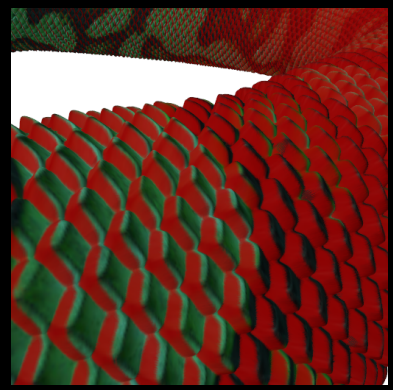
popping, aliasing, and inconsistent appearance

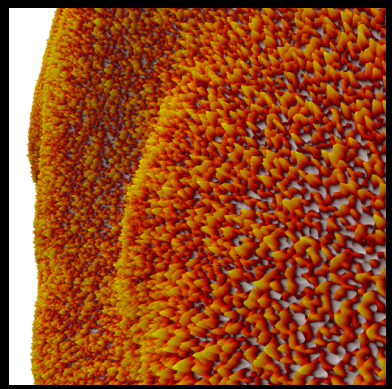
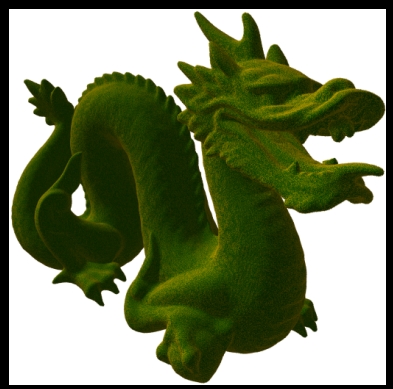
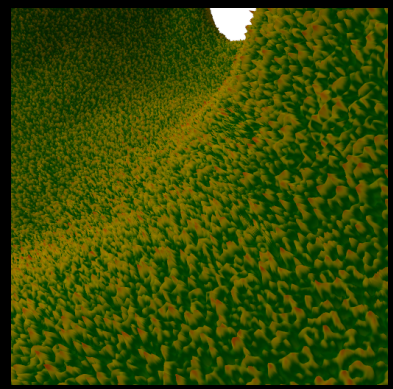
Introduction

We investigate how to

- filter correctly procedural surfaces
- with small-scale geometry, colors, and visibility

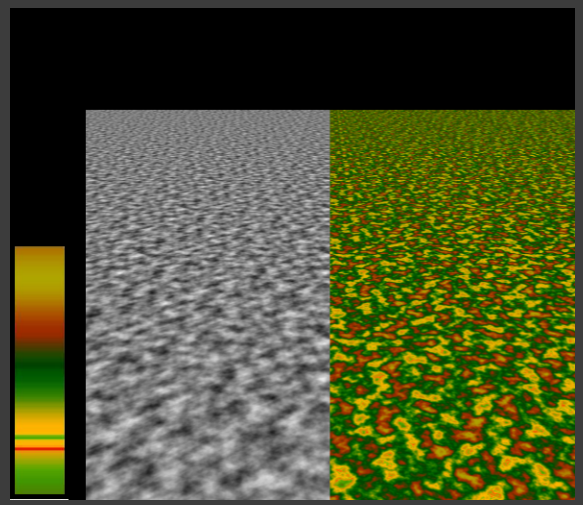





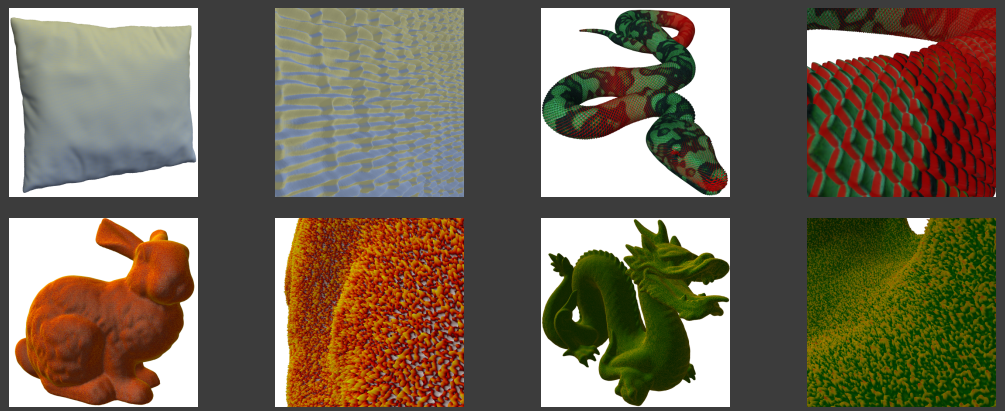




Plan

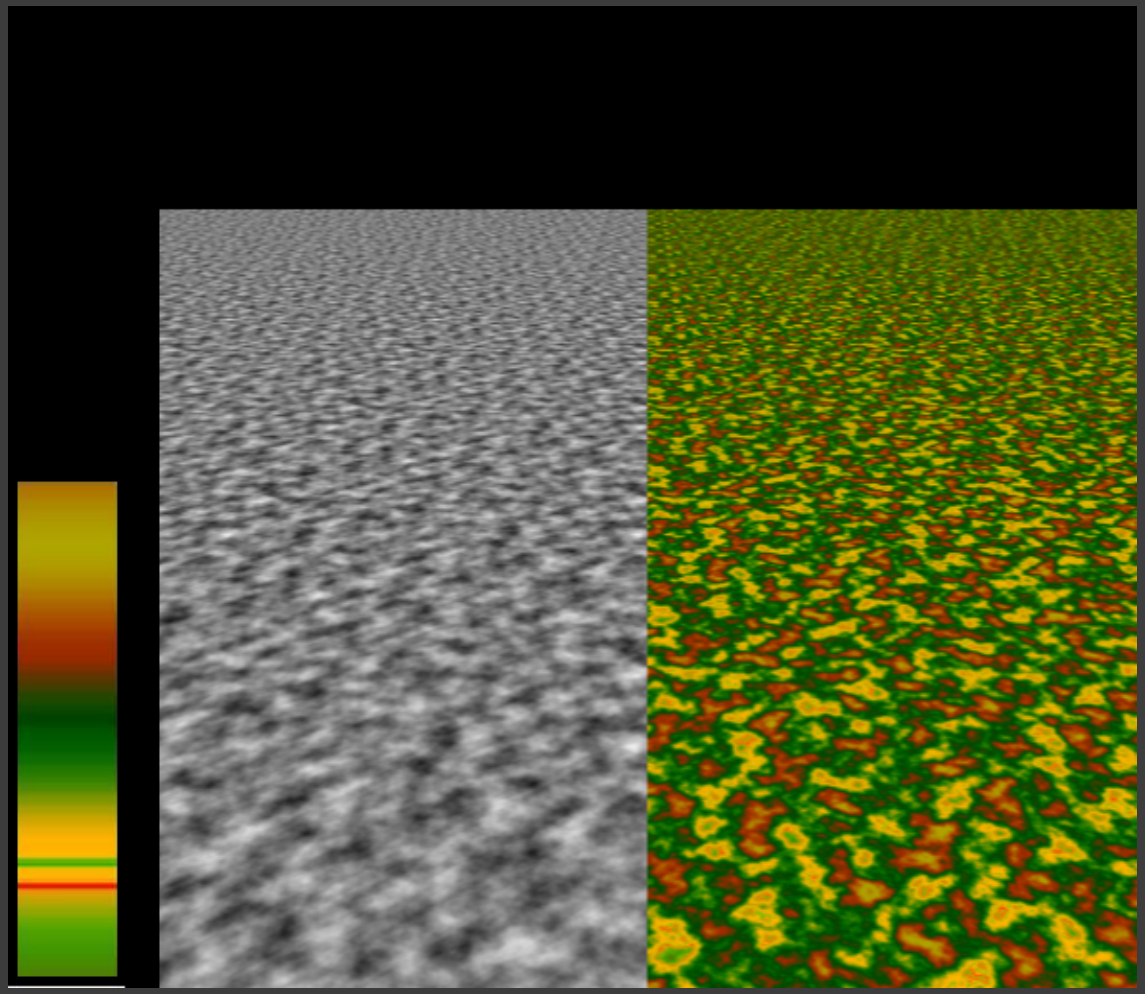
I. Filtering Procedural Textures (only colors)



II. Filtering Procedural Surfaces (colors and geometry)



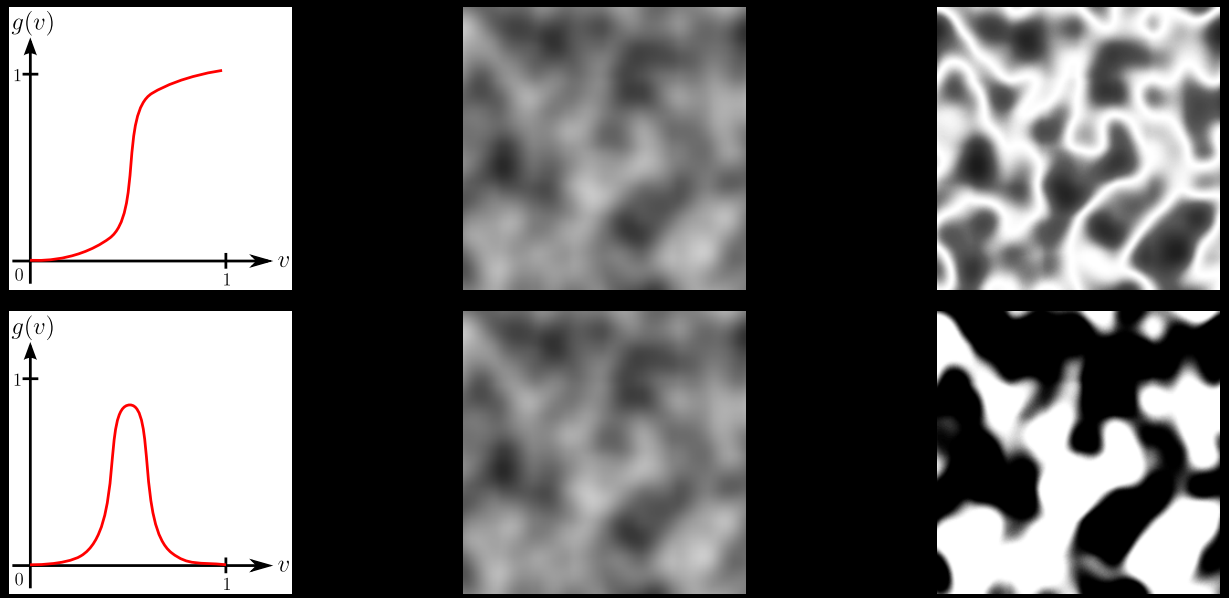
I. Filtering Procedural Textures



I.1 Definition

$$\text{texture}(x) = g(f(x))$$

$g \quad \circ \quad f \quad = \quad g(f)$



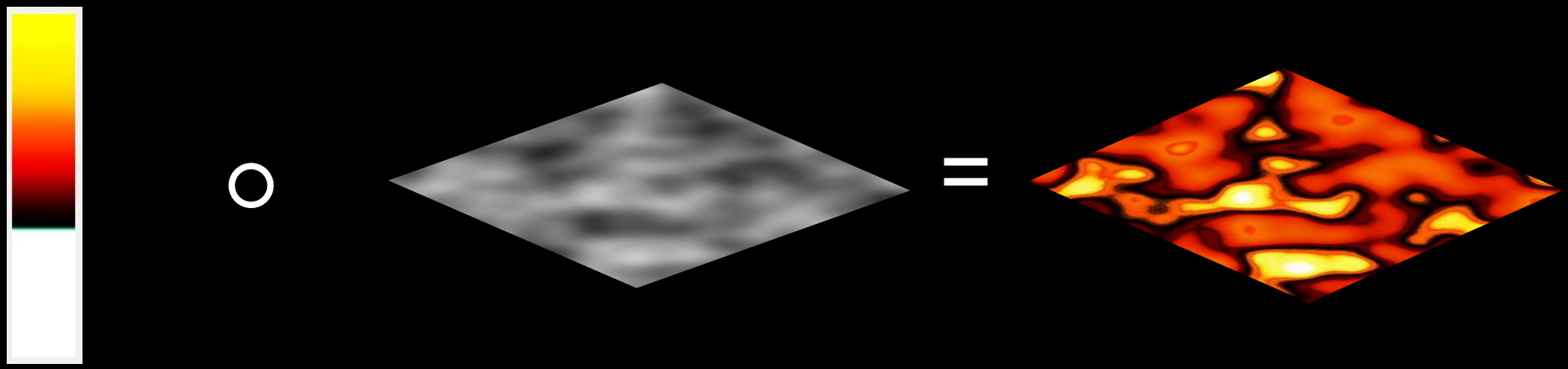
$f =$ **noise [Perlin83, Worley96, LLDD09]**

$g =$ **nonlinear transfer function**

I.1 Definition

Nonlinear operations summarized in color maps

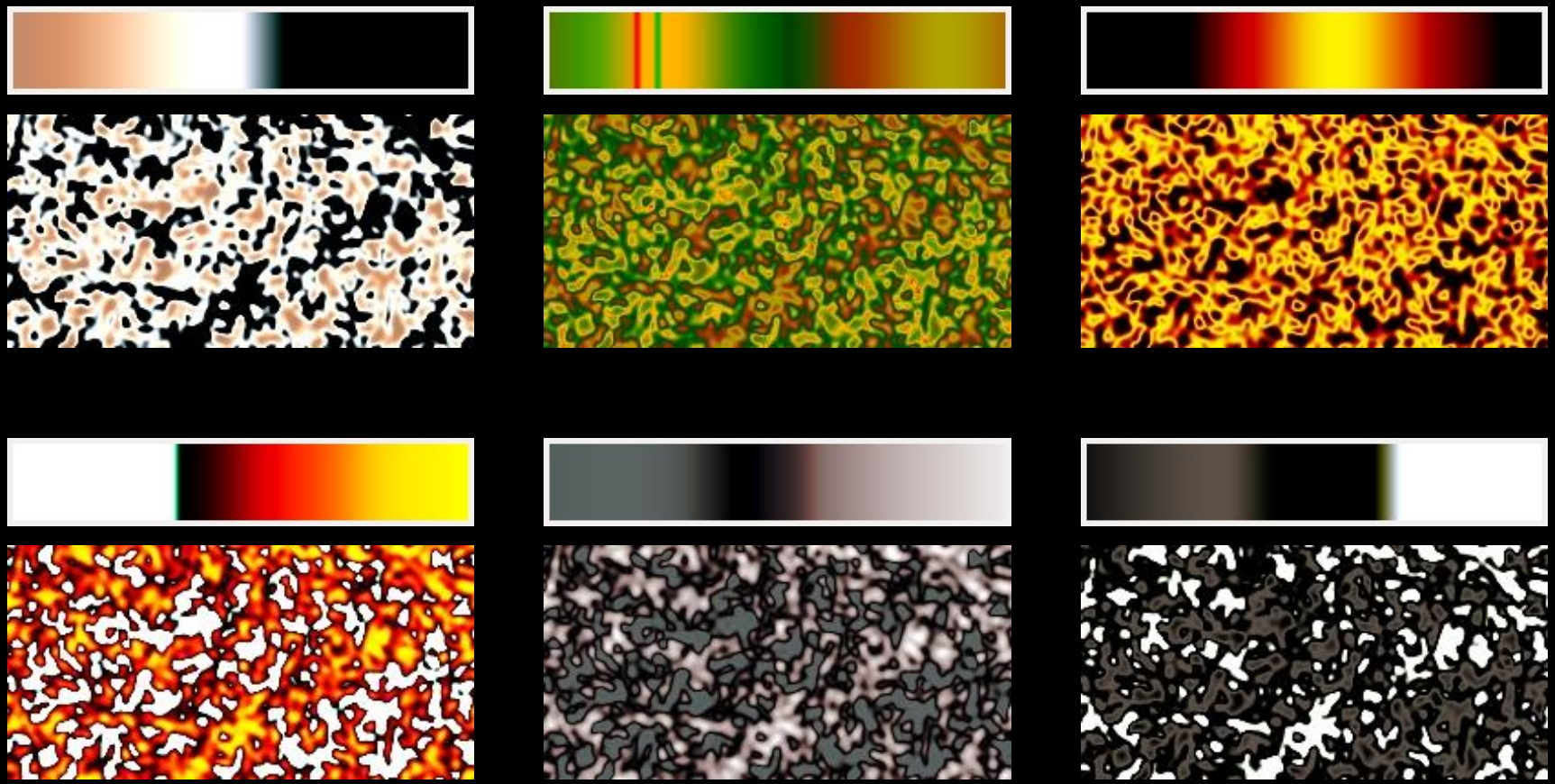
C_0 $\quad$ f $\quad$ $C_0 \circ f$



Color map $\quad$ **Grayscale function**

I.1 Definition

Nonlinear operations summarized in color maps



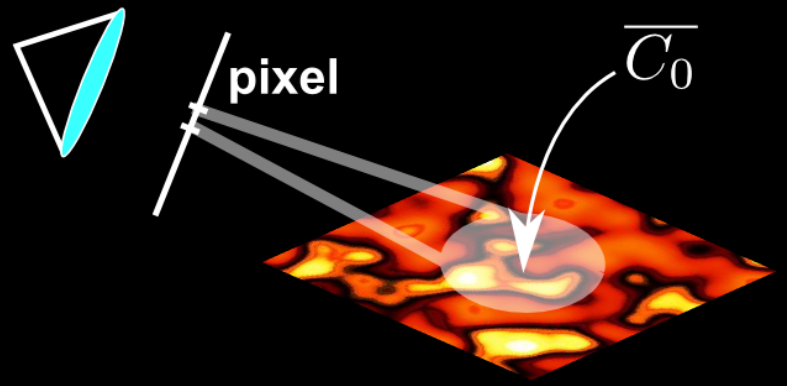
I.2 Problem

Important properties of procedural textures

- efficient on-the-fly evaluation
- intuitive control parameters

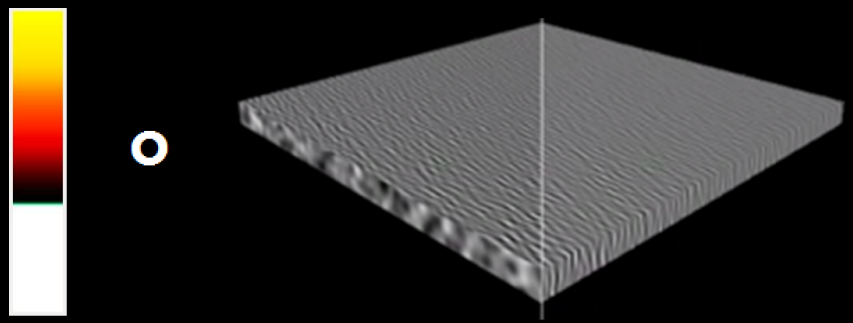
Filtering does not work

correct $\int C_0(f) \quad =$



$\neq$

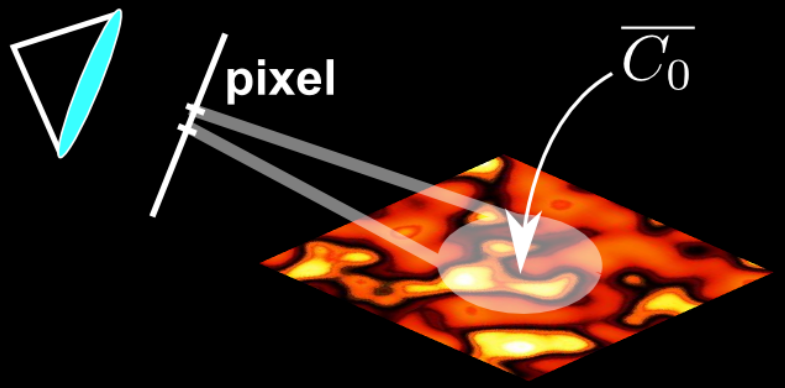
wrong $C_0(\int f) \quad =$



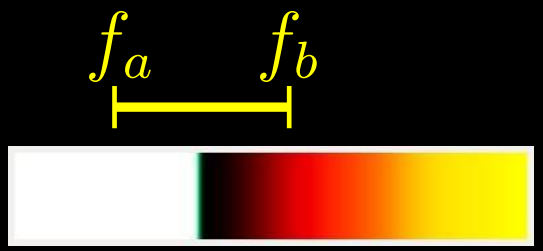
I.3 Previous Work

[RTBS92, HCKT99, LLDD09]

$$\bar{C}_0 = \int C_0(f(x))\, dx \quad \approx \quad \int_{f_a}^{f_b} C_0(f)\, df$$



$\approx$



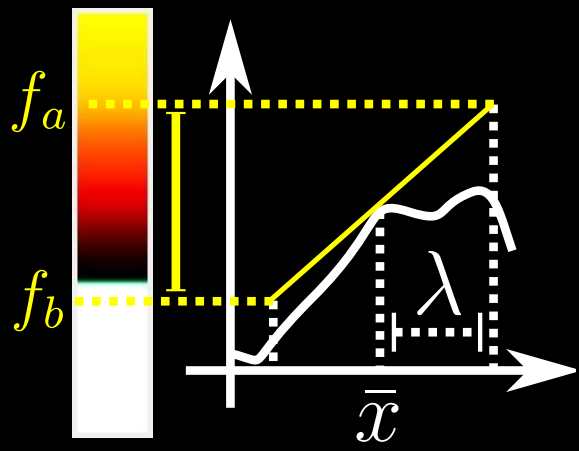
texture filtering $\approx$ **color map**

I.3 Previous Work

Box Filtering

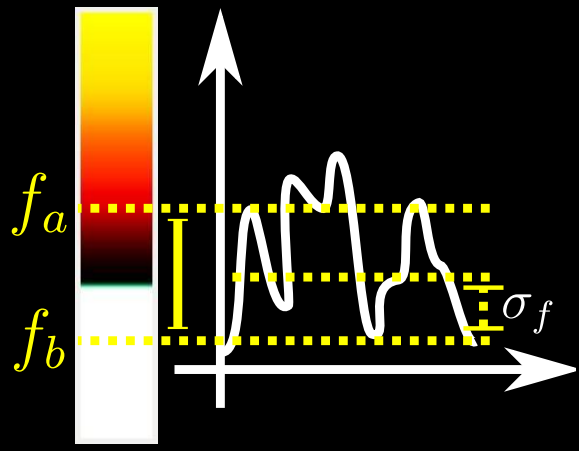
[Hart99]

- works only locally (first-order expansion)
- gradient ill-defined

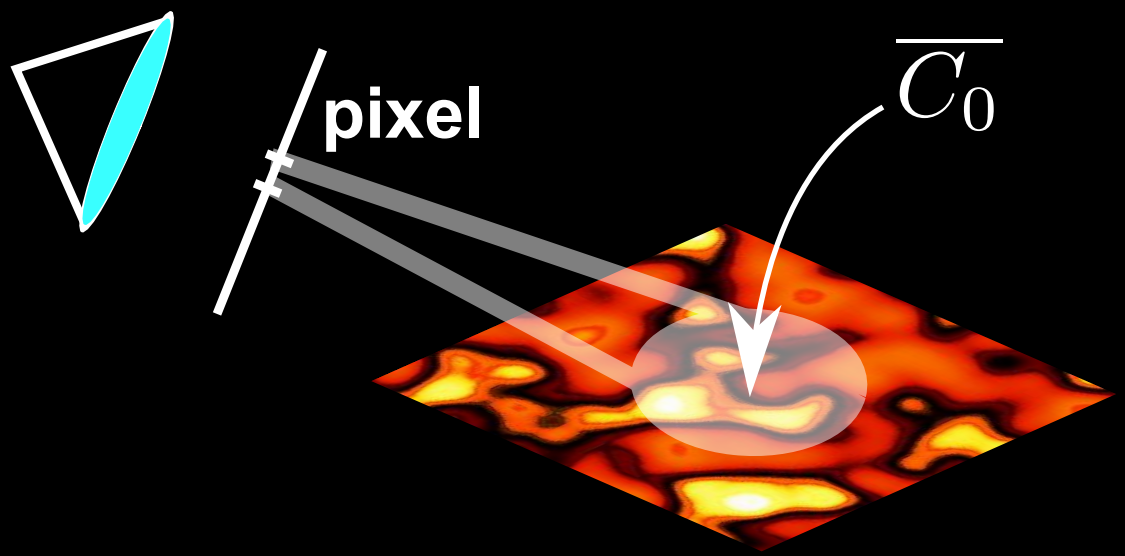


[Lagae09]

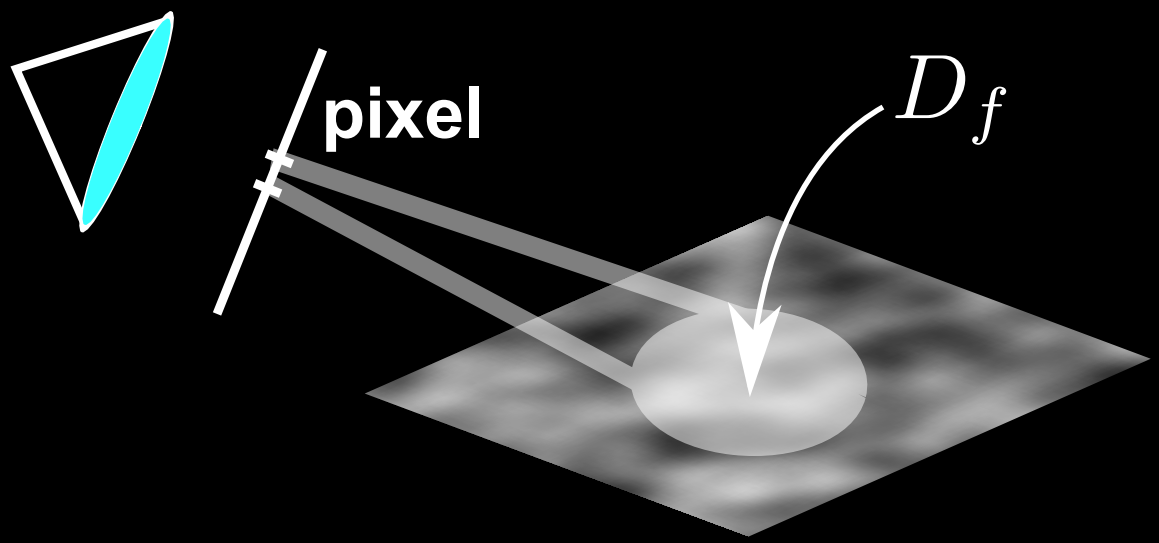
- better but Box Filter is still not ideal



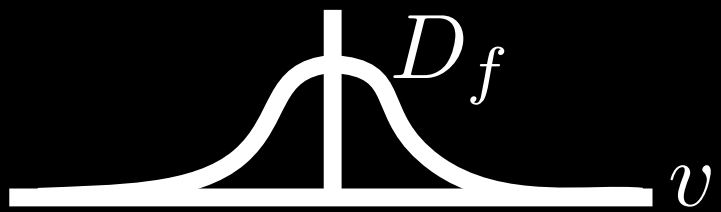
I.4 Our Method



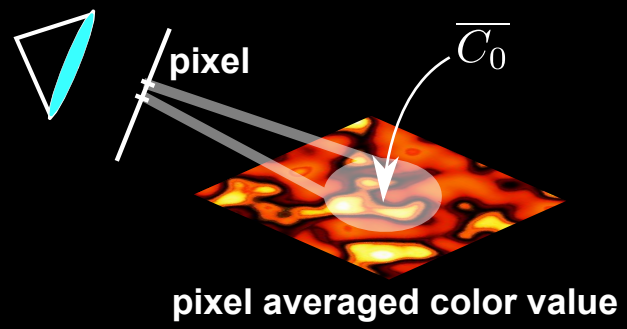
I.4 Our Method



grayscale value distribution within pixel footprint



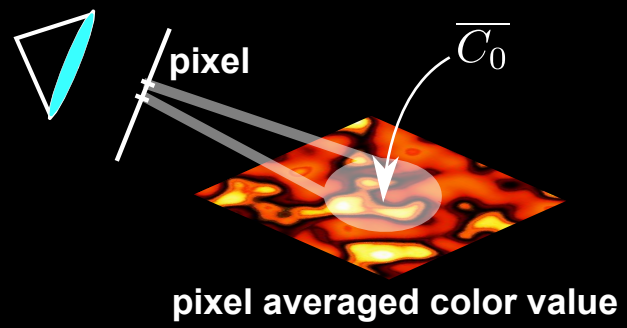
I.4 Our Method



is exactly the inner product

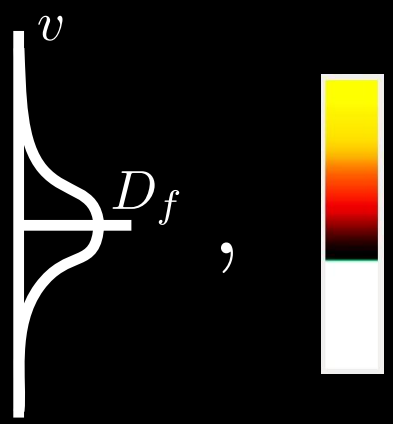
$$\overline{C_0} = \; < \; v \; D_f \; , \; \; >$$

I.4 Our Method



is exactly the inner product

$$\overline{C_0} = \; < \; D_f \; , \; \; >$$



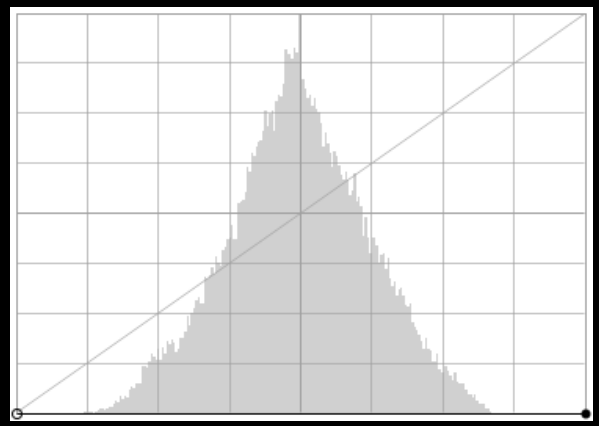
1. **How to represent grayscale distributions?**
2. **How to compute the inner product?**

I.4 Our Method

We choose Gaussian distributions

- noise = Gaussian process
- lightweight, analytical

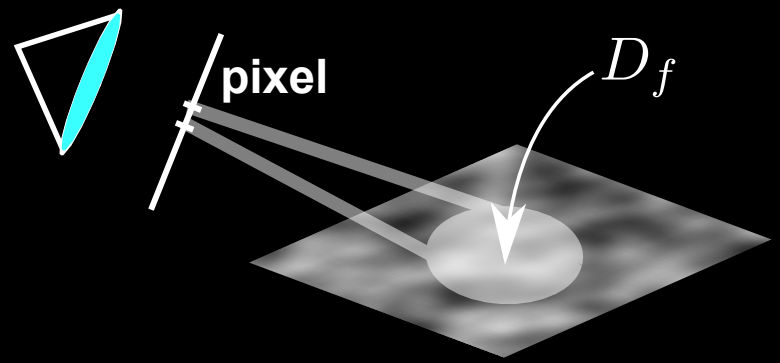
noise



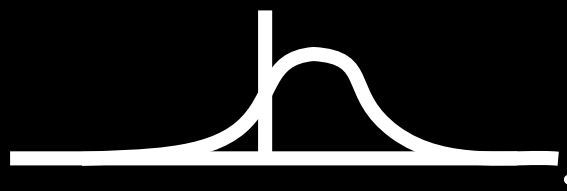
noise histogram

I.4 Our Method

1. fetch / compute
noise Gaussian distribution

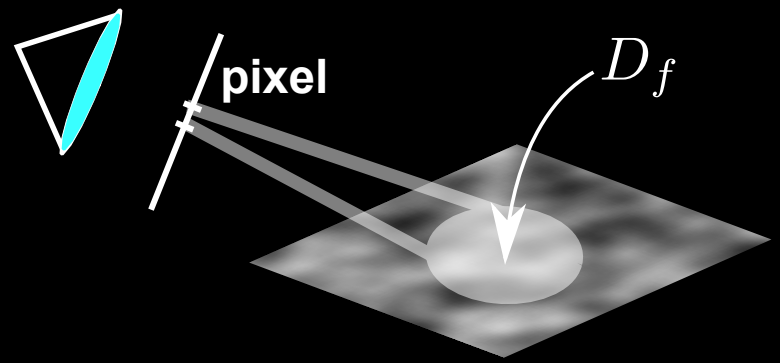


$$D_f = \mathcal{N}(\overline{f}, \sigma_f^2)$$

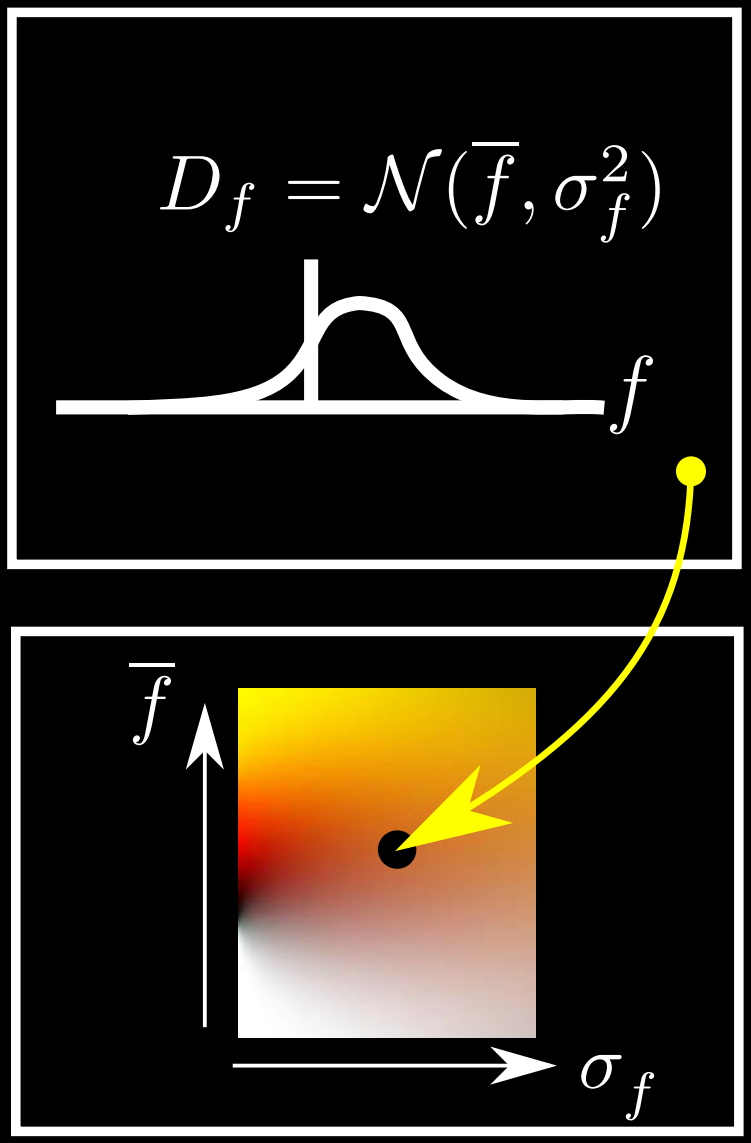


I.4 Our Method

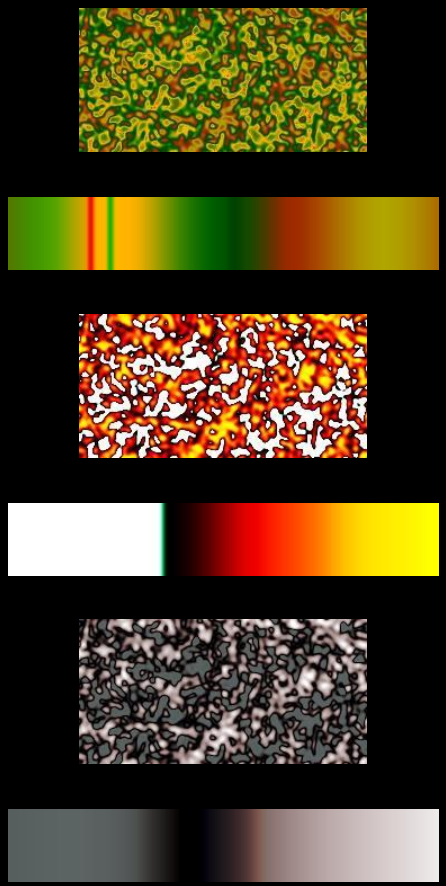
1. fetch / compute
noise Gaussian distribution



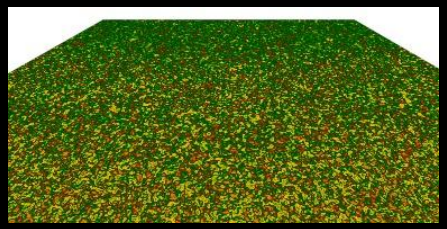
2. fetch pre-convoluted
color map



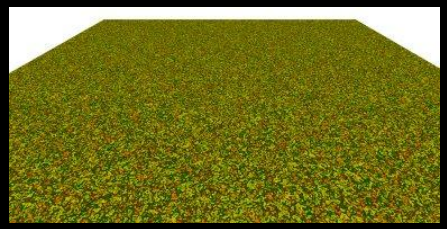
I.5 Results



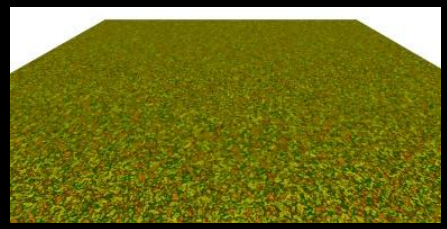
Color map & close view



Naïve filtering

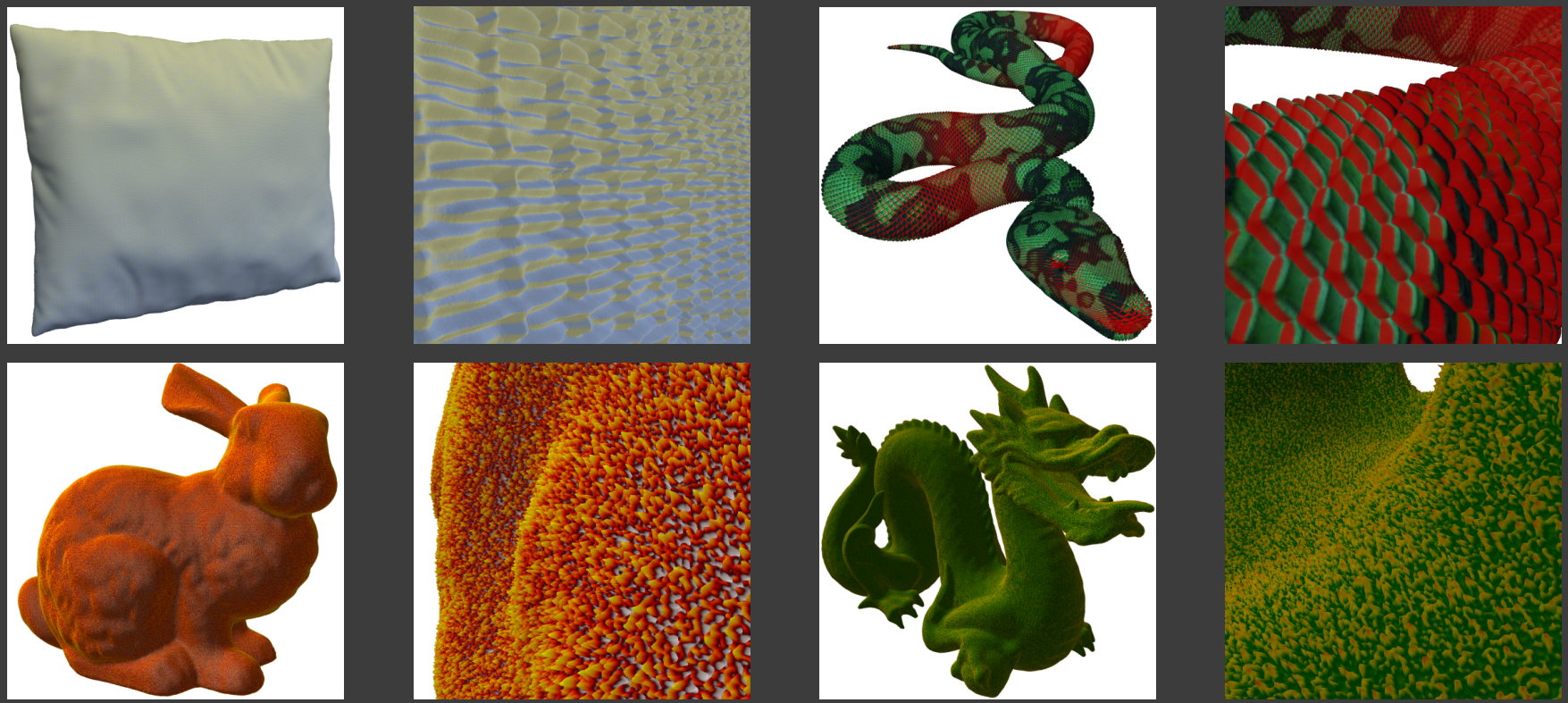


Ground Truth



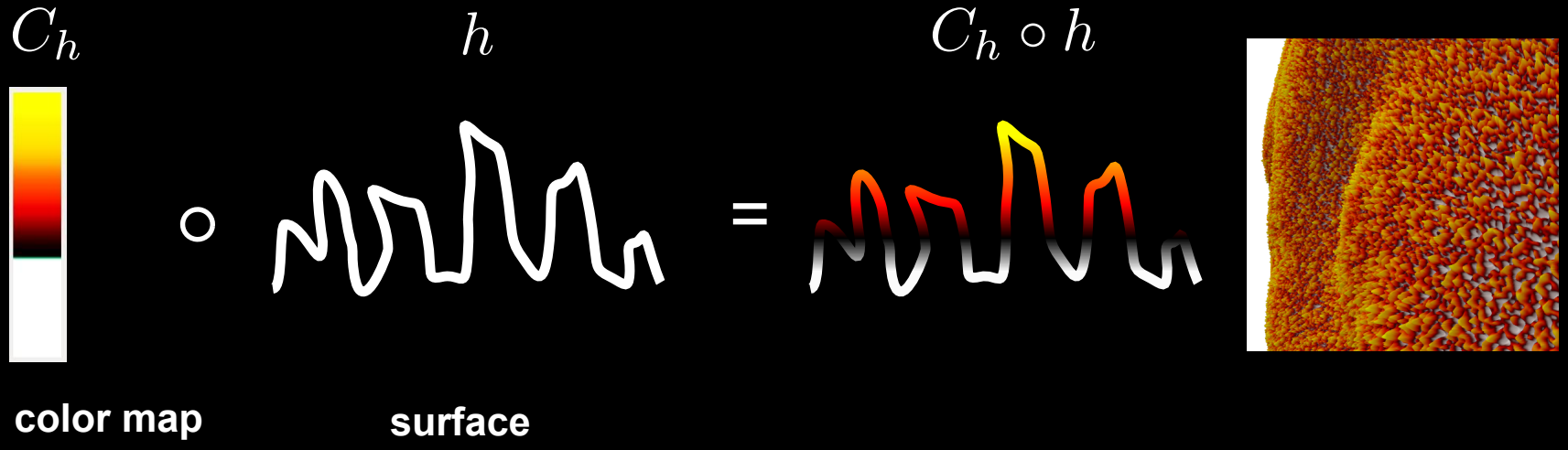
Our model

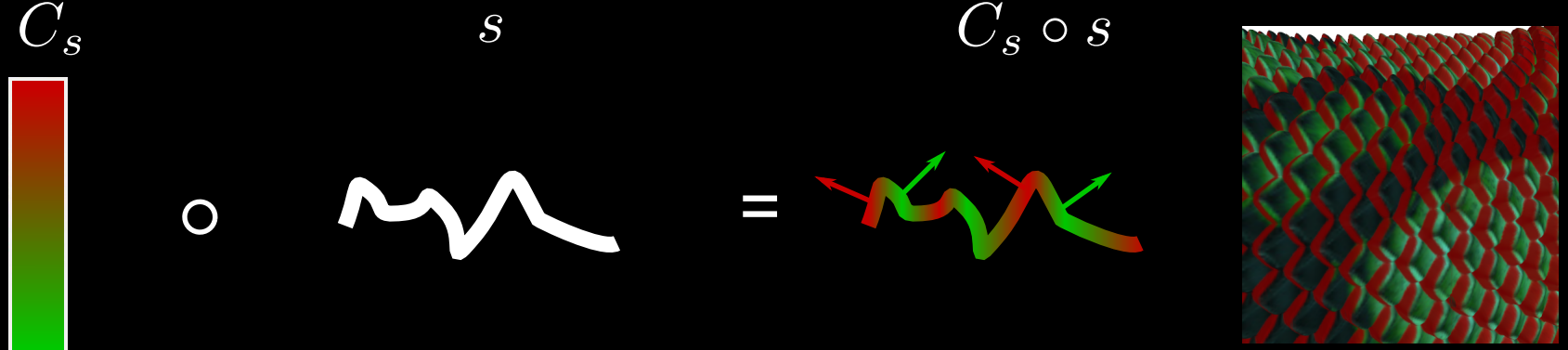
II. Filtering Procedural Surfaces



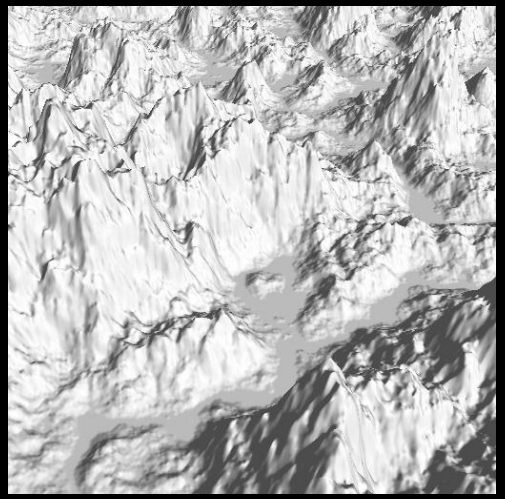
II.1 Color Mapping Surfaces

Correlations with height and slope



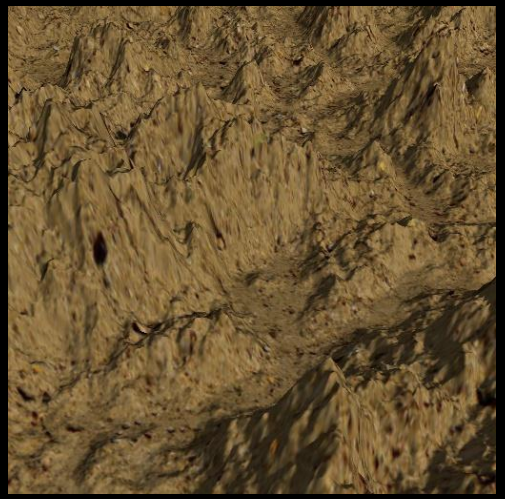


II.1 Color Mapping Surfaces



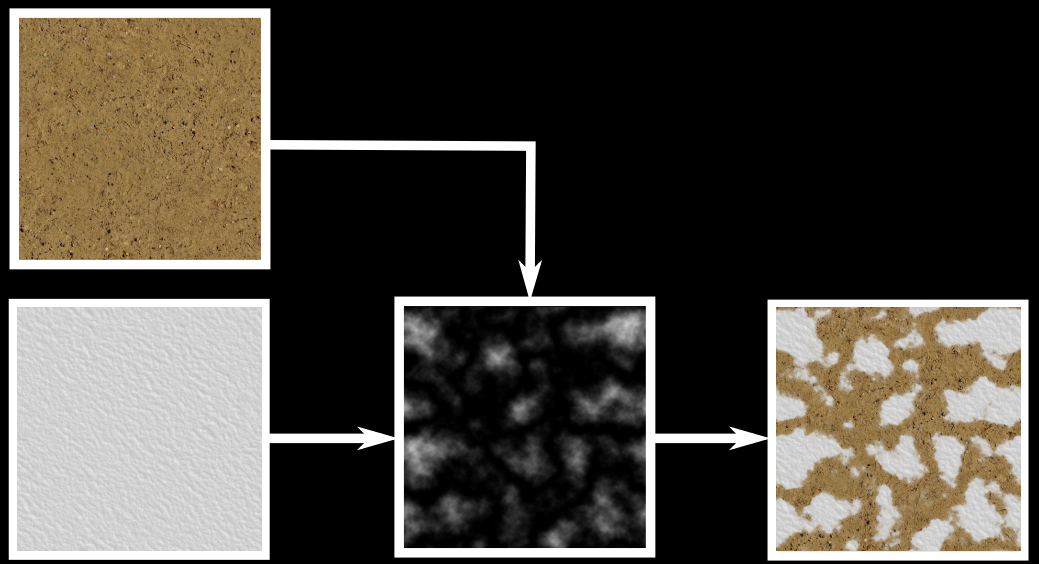
II.1 Color Mapping Surfaces

ground textures

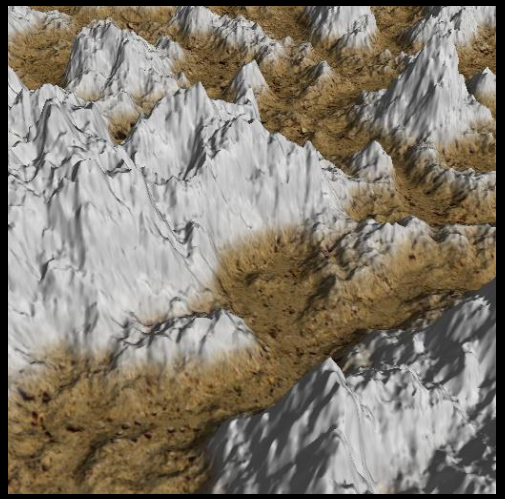


II.1 Color Mapping Surfaces

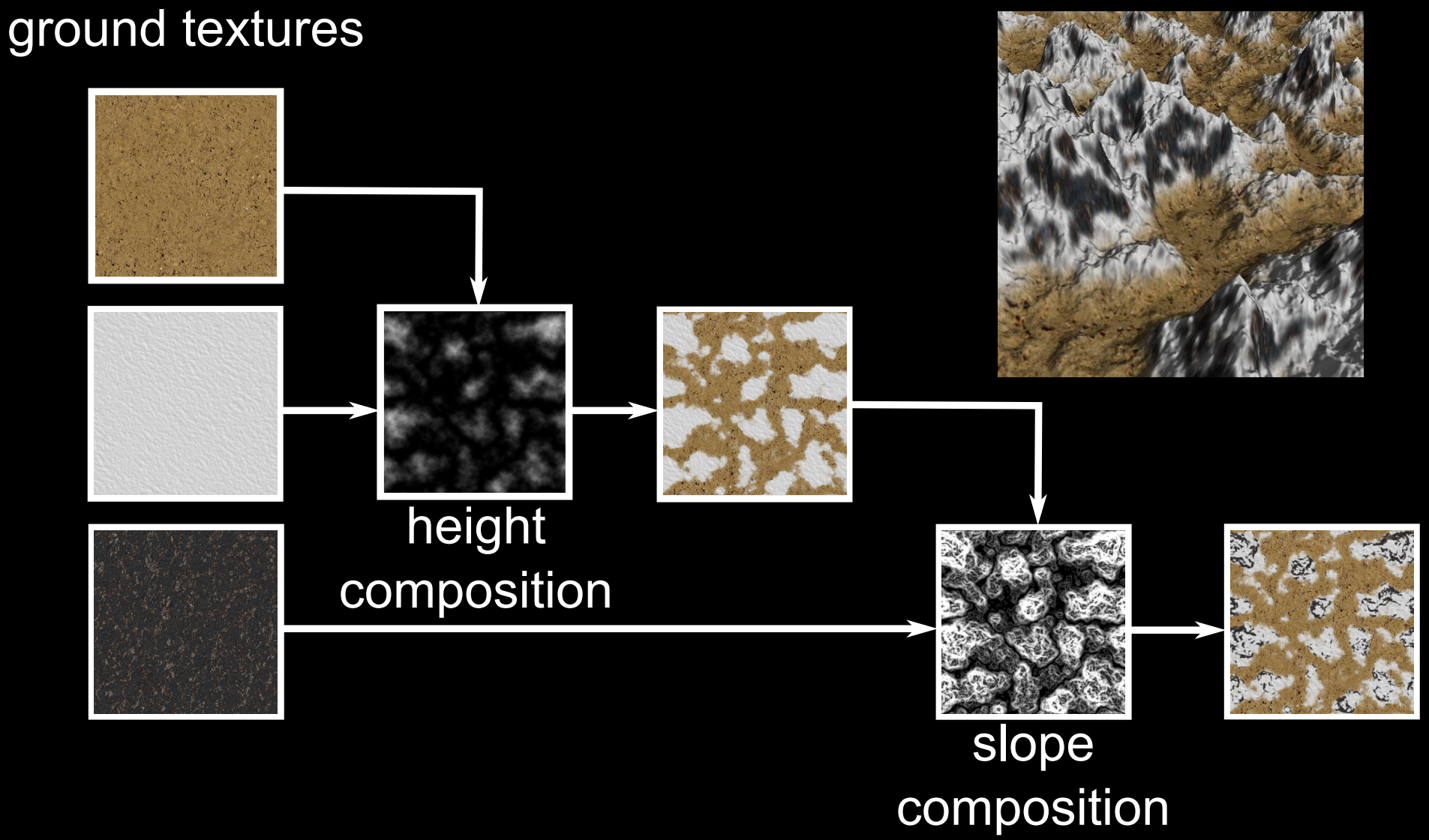
ground textures



height
composition

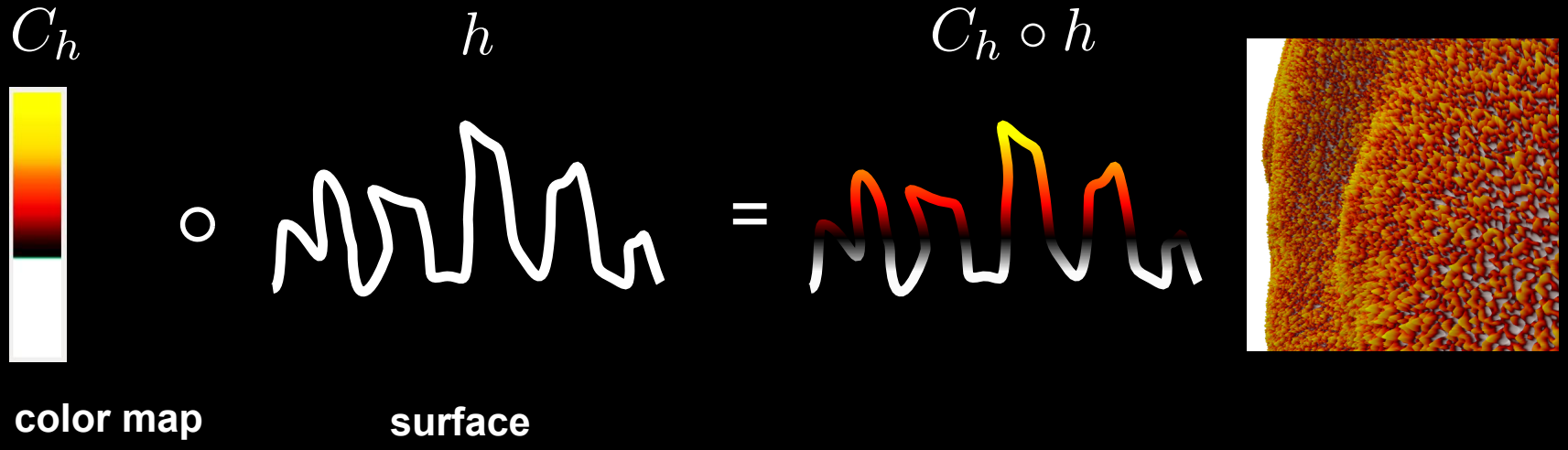


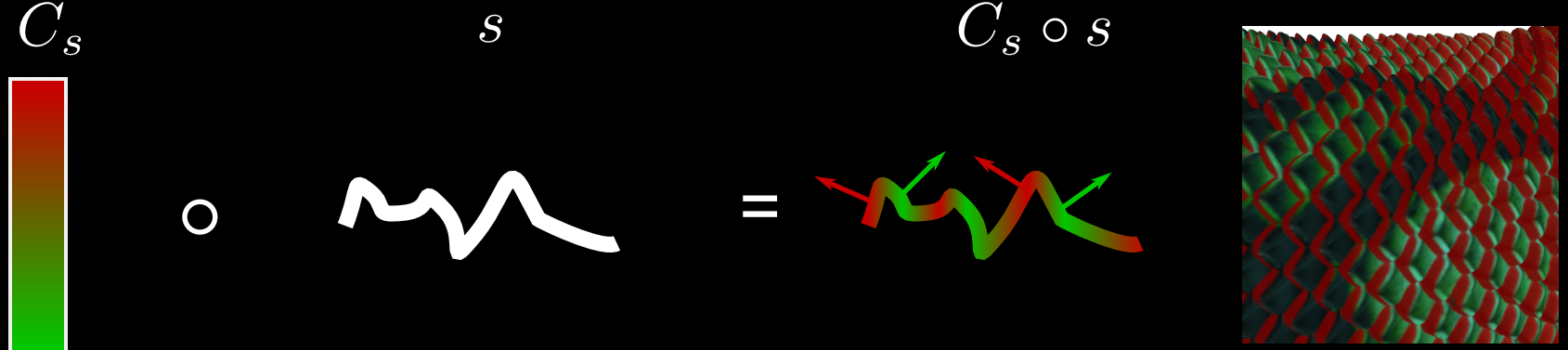
II.1 Color Mapping Surfaces



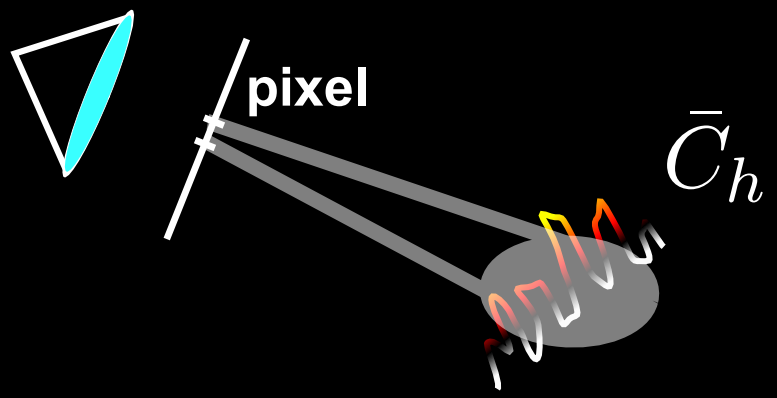
II.1 Color Mapping Surfaces

Correlations with height and slope



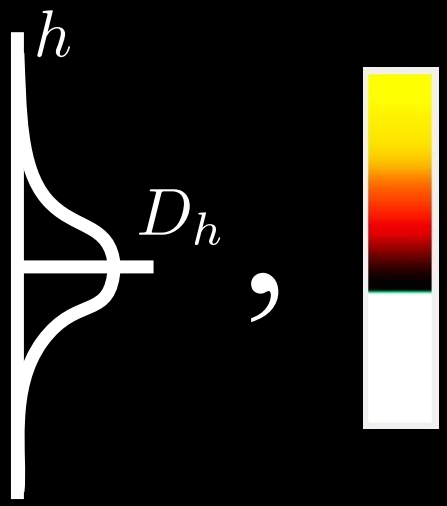


II.2 Problem

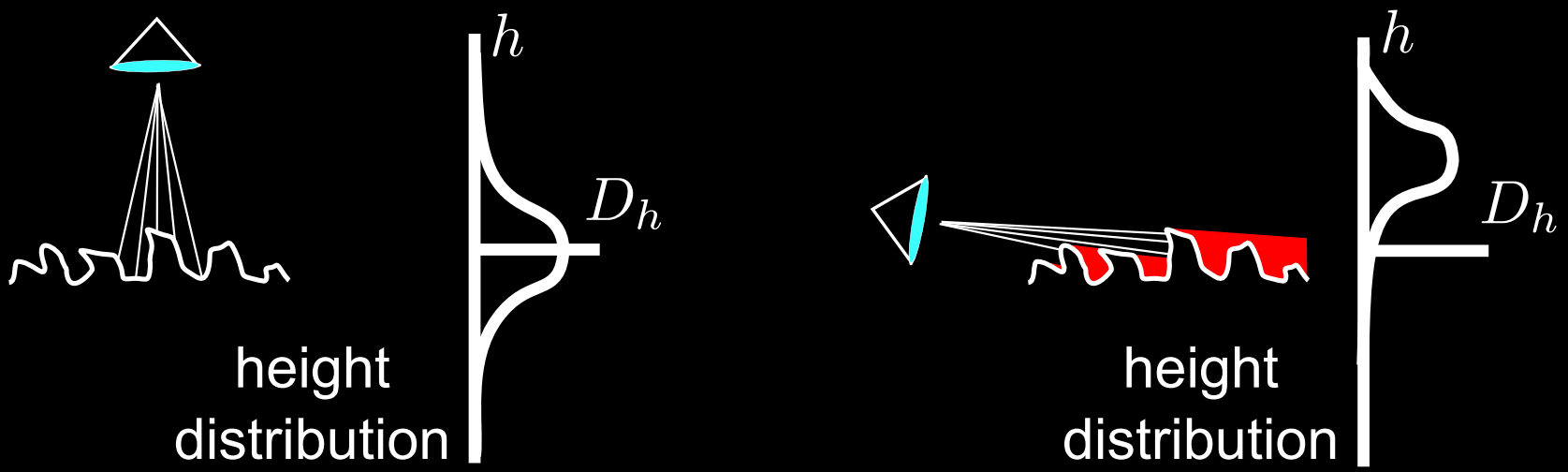


is exactly the inner product

$$\bar{C}_h = < D_h \; , \; >$$

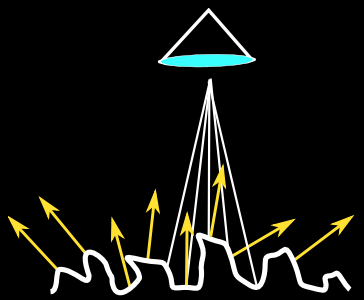


II.2 Problem

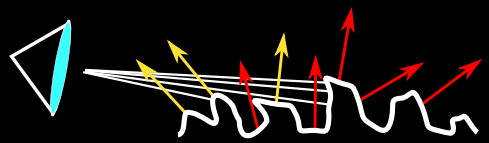


Height distribution D_h is view- and light-dependent because of masking and shadowing

II.2 Problem



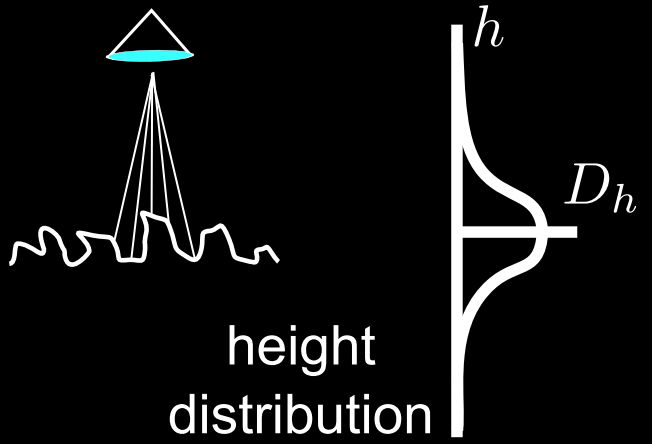
slope (normal)
distribution



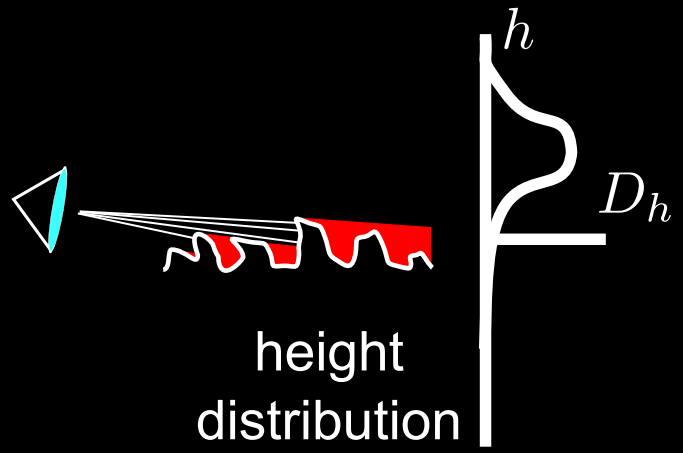
slope (normal)
distribution

Slope distribution D_s is view-dependent because of back-face culling and area weighting

II.2 Problem

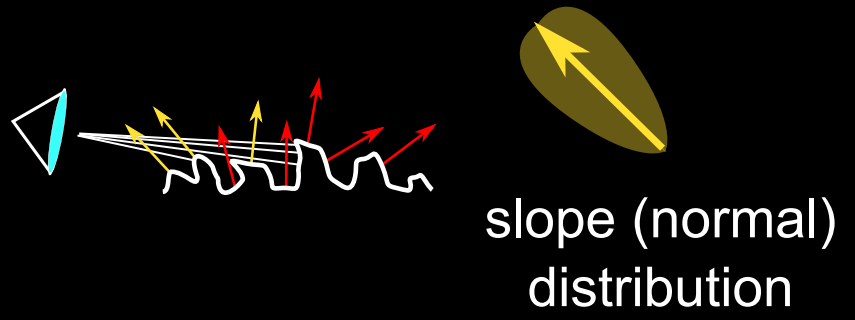


height distribution



height distribution

slope (normal) distribution



slope (normal) distribution

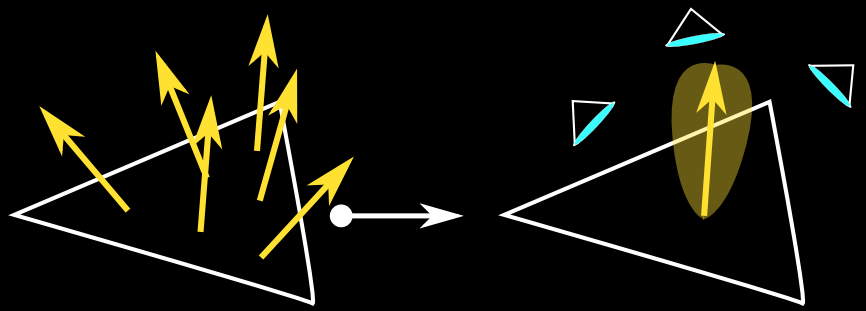
These schemes are qualitative.
How to model quantitatively view-dependency?

II.2 Problem

View-dependency is usually NOT taken in account

e.g.

- **[Smooth Transitions between Bump Rendering Algorithms], SIGGRAPH 93**
- **[Multiresolution Reflectance Filtering], EGSR05**
- **[Frequency Domain Normal Map Filtering], SIGGRAPH 07**
- **[LEAN Mapping], I3D 10**



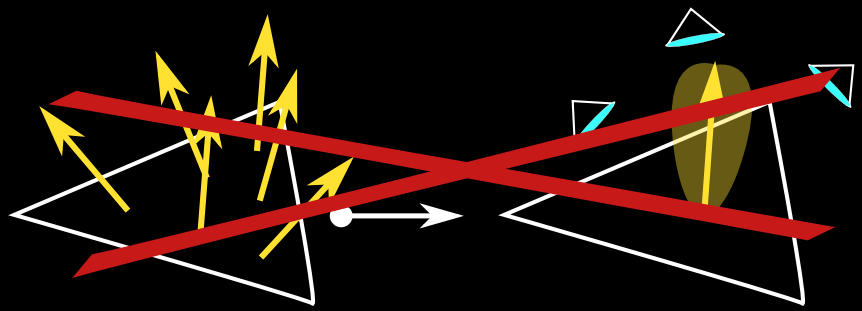
normal filtering on flat surface

II.2 Problem

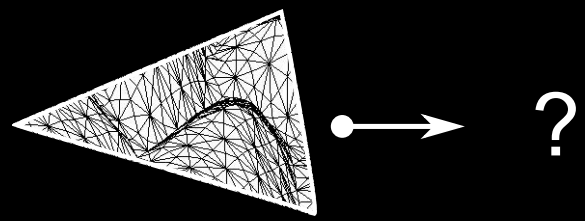
View-dependency is usually NOT taken in account

e.g.

- **[Smooth Transitions between Bump Rendering Algorithms], SIGGRAPH 93**
- **[Multiresolution Reflectance Filtering], EGSR05**
- **[Frequency Domain Normal Map Filtering], SIGGRAPH 07**
- **[LEAN Mapping], I3D 10**



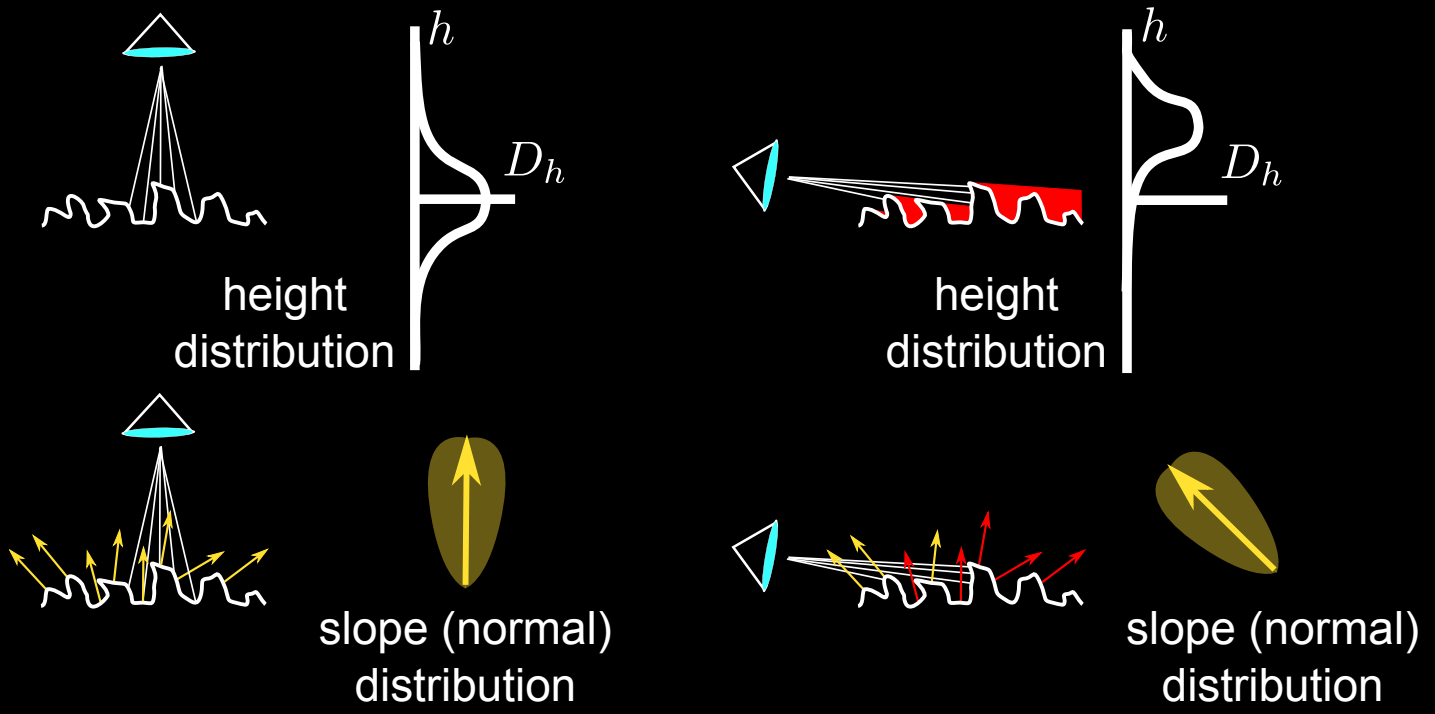
normal filtering on flat surface



normal filtering on detailed surface

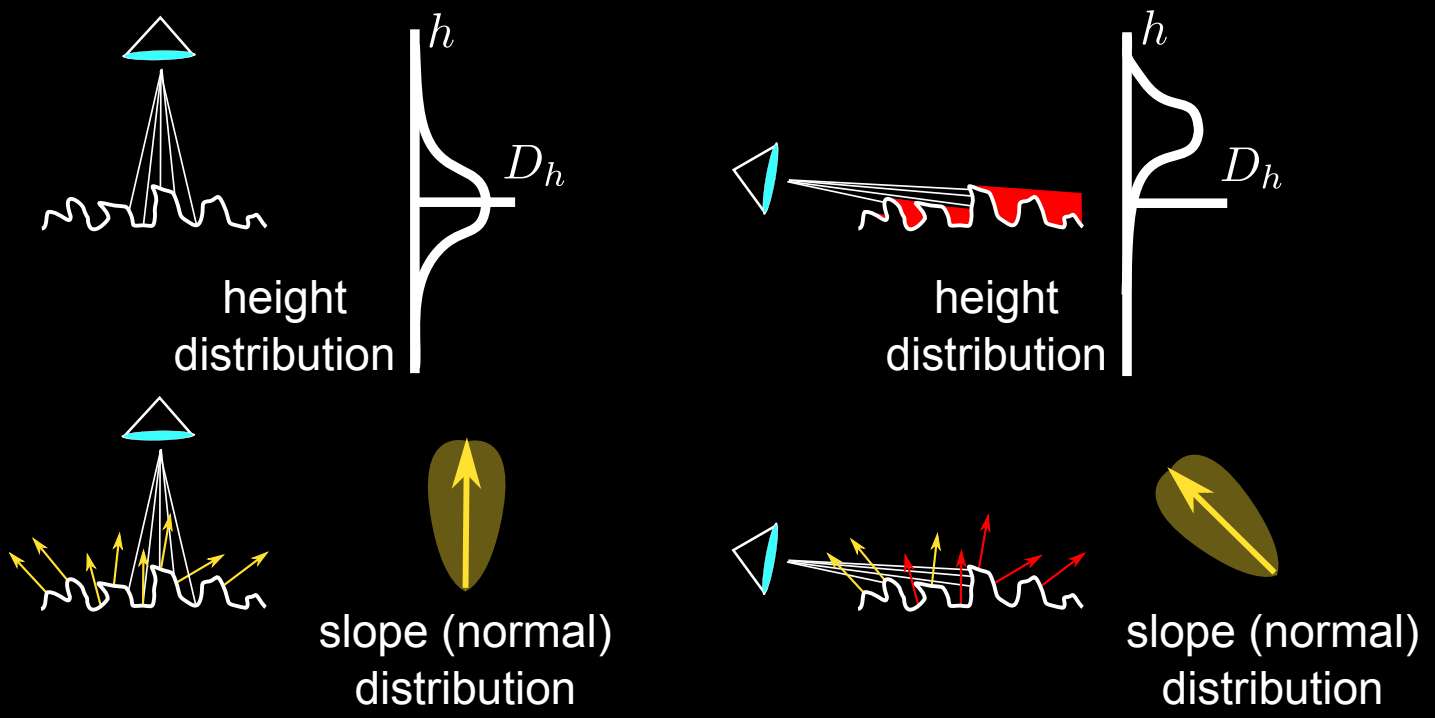
II.3 Our Method

How to model and compute view-dependency?



II.3 Our Method

How to model and compute view-dependency?

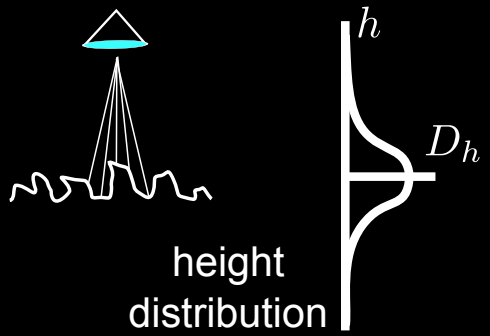


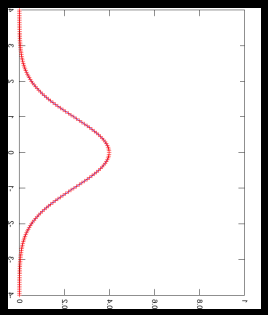
→ micro-facet theory is the appropriate paradigm

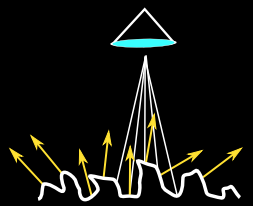
- physically sound formulation
- accurate analytical approximations

II.3 Our Method

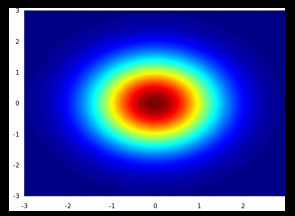
We derive accurate analytical approximations

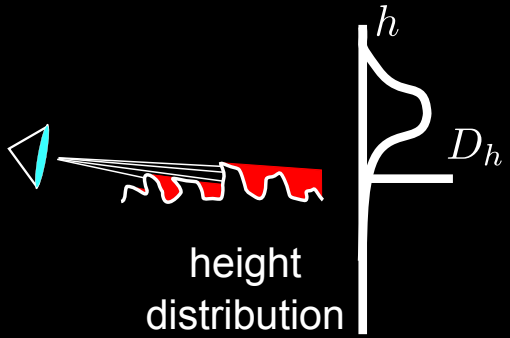


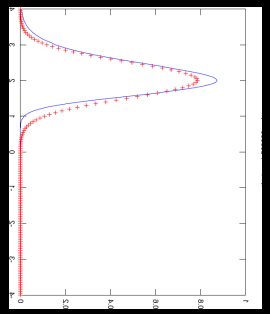


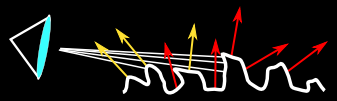


slope (normal)
distribution

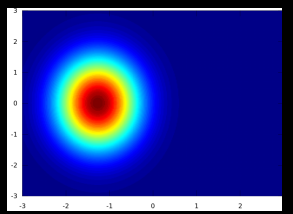






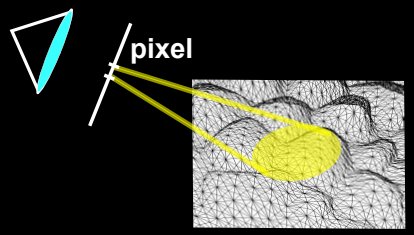


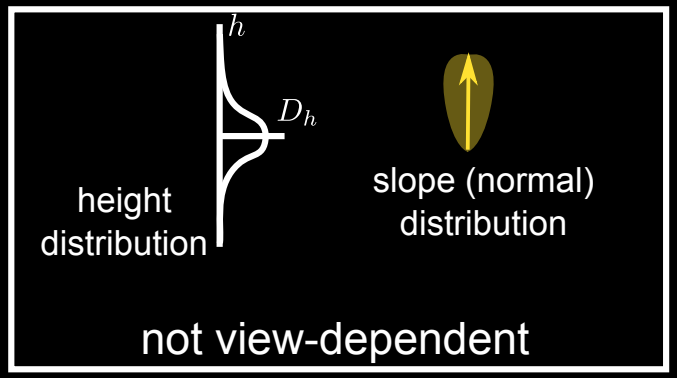
slope (normal)
distribution



II.3 Our Method

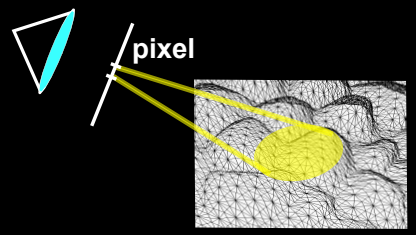
1. fetch / compute surface parameters



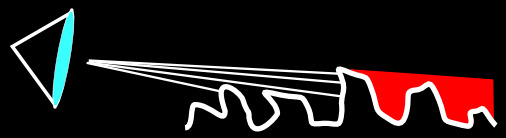


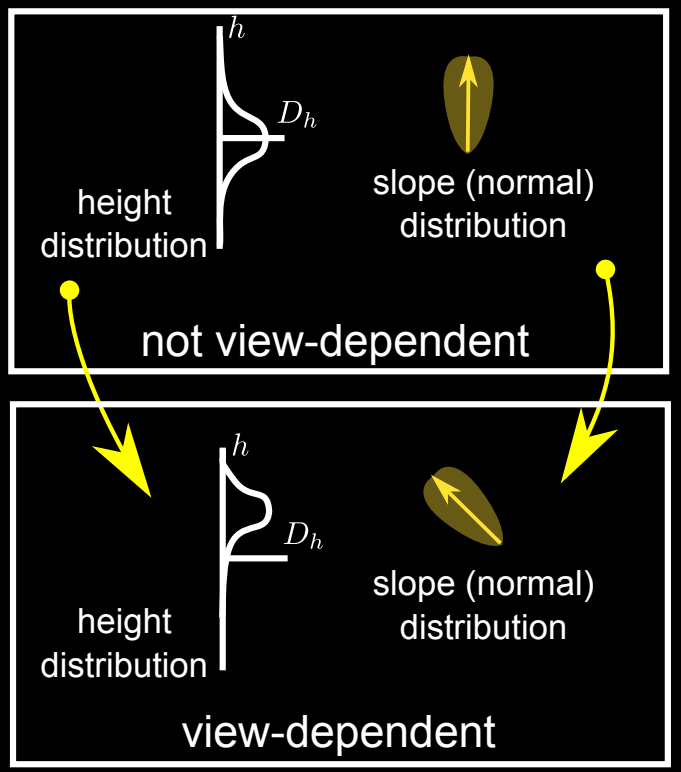
II.3 Our Method

1. fetch / compute surface parameters



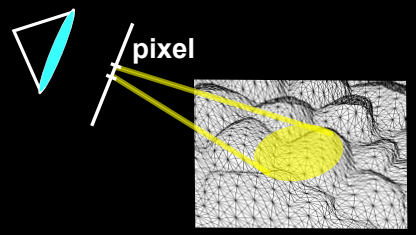
2. compute view-dependent surface parameters



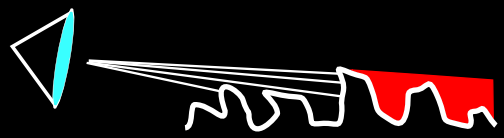


II.3 Our Method

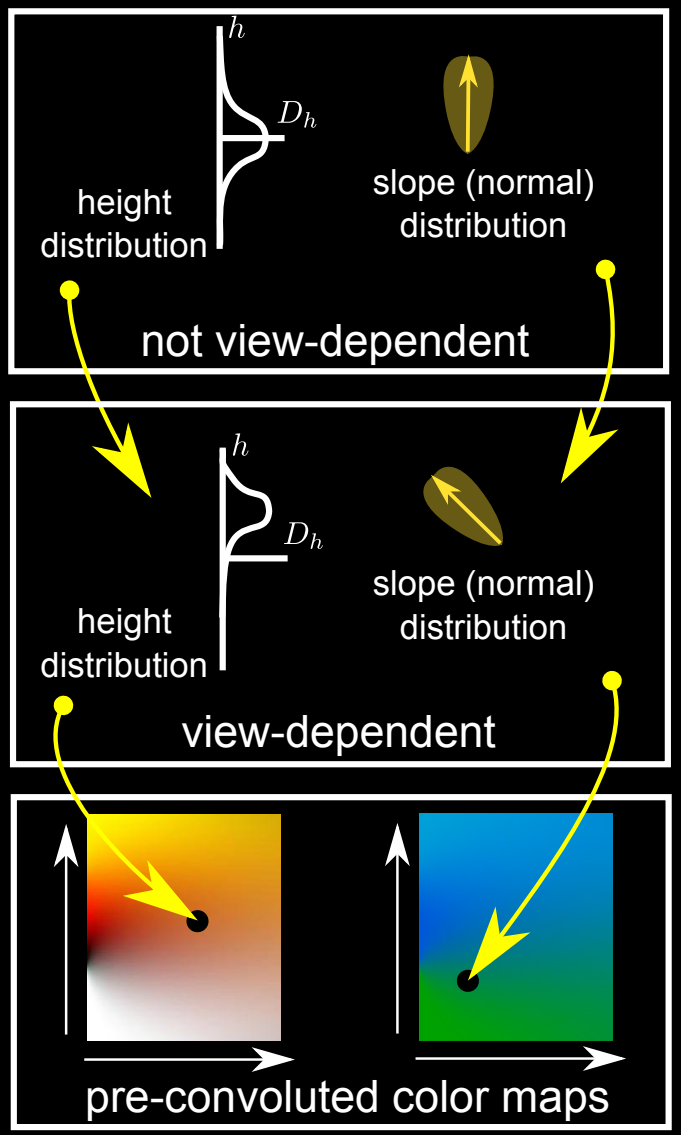
1. fetch / compute surface parameters



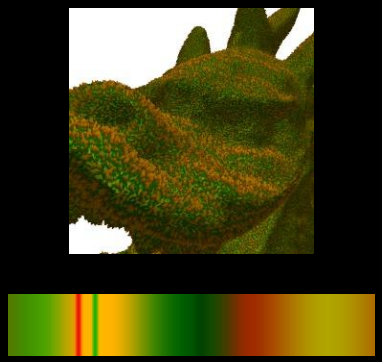
2. compute view-dependent surface parameters

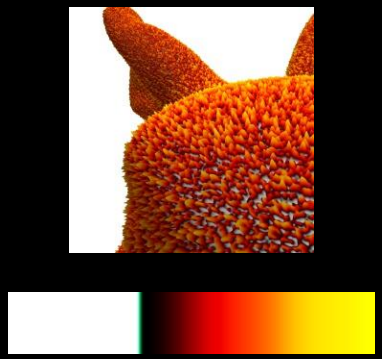


3. fetch pre-convoluted color maps



II.4 Results



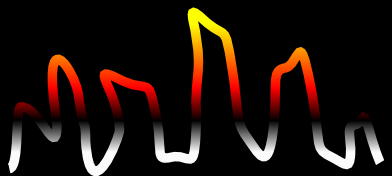


Color map & close view

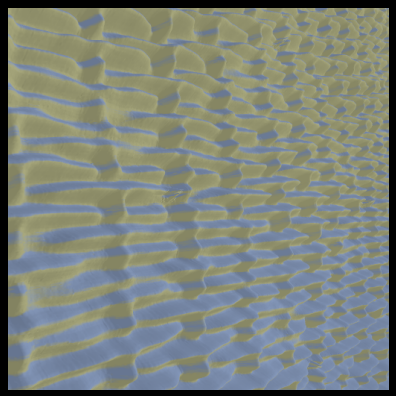
Texture filtering

Ground Truth

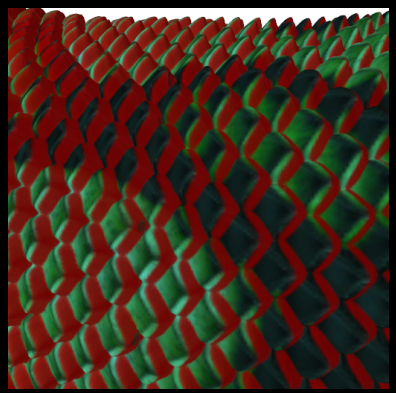
Surface filtering



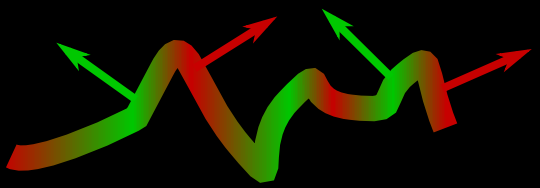
II.4 Results








Close view | Texture filtering | Ground Truth | Surface filtering

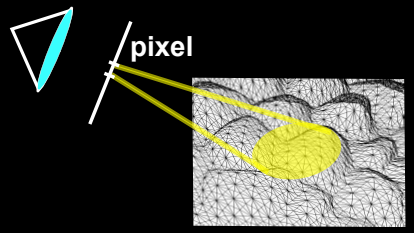


II.4 Results

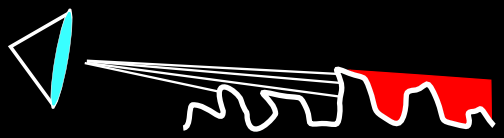
	 Texture filtering	 Ground Truth	 Surface Filtering
Tessellation ON	130Hz	$<$1Hz 256 samples/pixel	130Hz
Tessellation OFF	869Hz	3Hz 256 samples/pixel	800Hz

II.5 Implementation

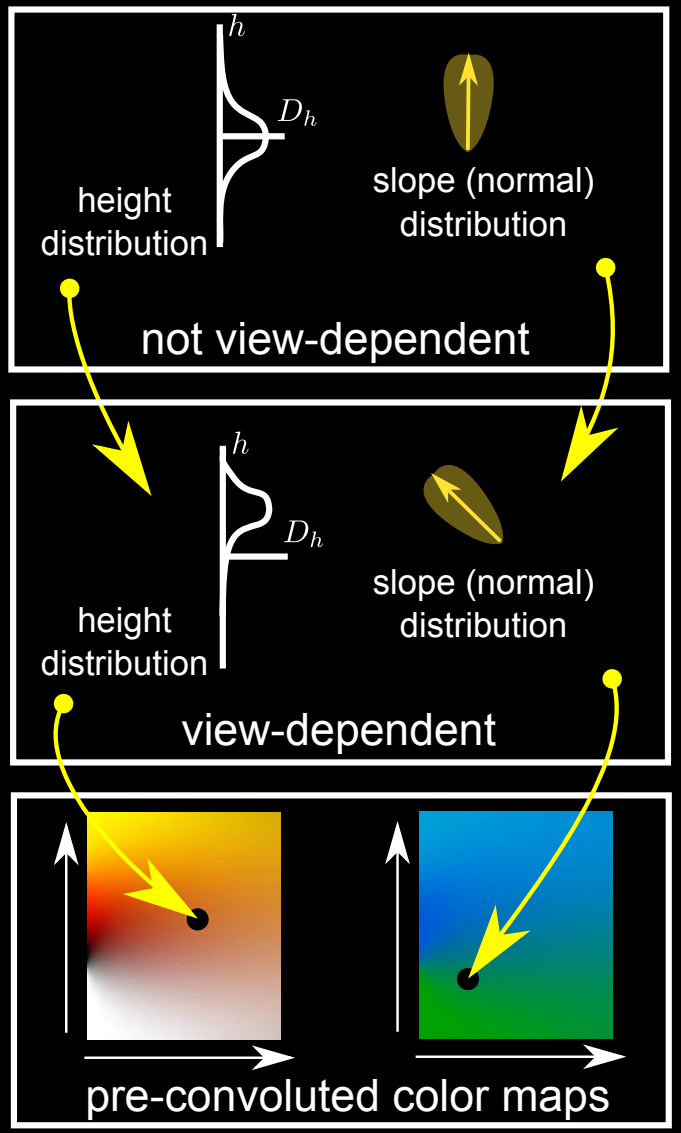
1. fetch / compute surface parameters



2. compute view-dependent surface parameters

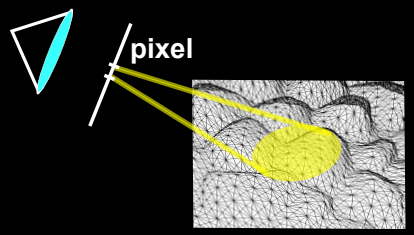


3. fetch pre-convoluted color maps

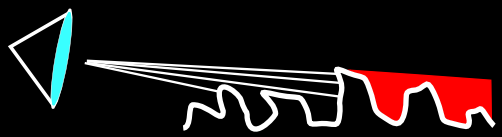


II.5 Implementation

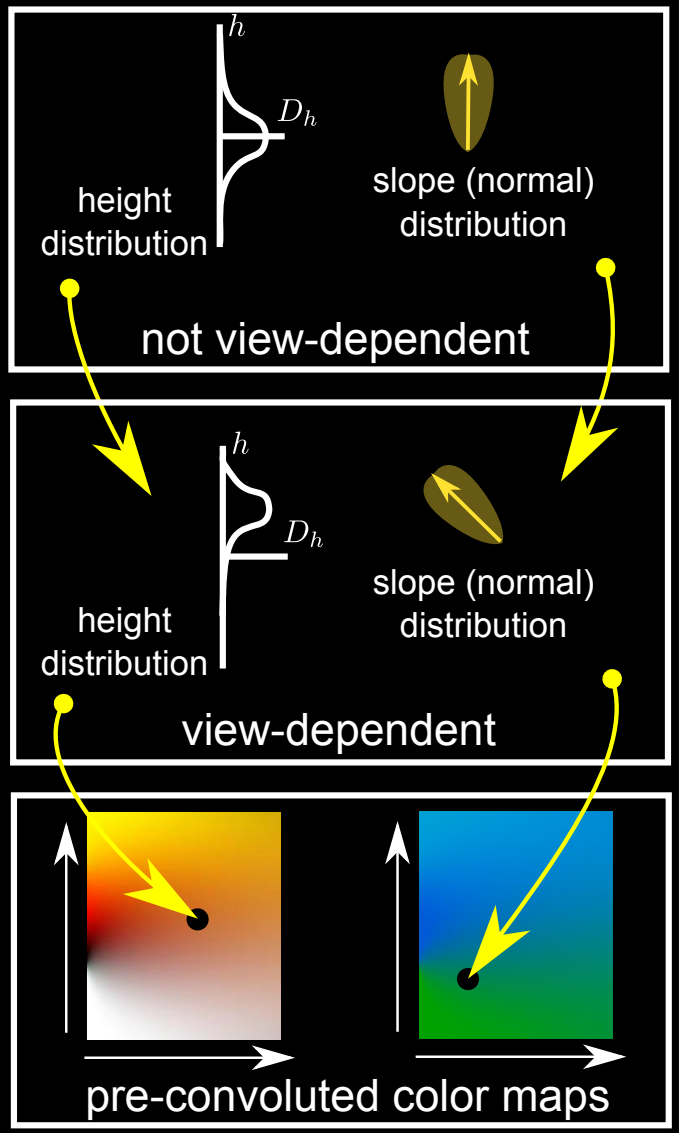
1. fetch / compute surface parameters



2. compute view-dependent surface parameters



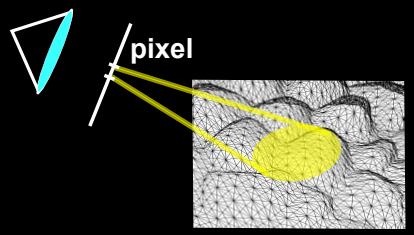
3. fetch pre-convoluted color maps



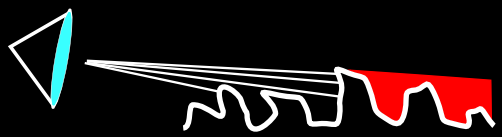
$\bar{h}$ average height (used for tessellation)

II.5 Implementation

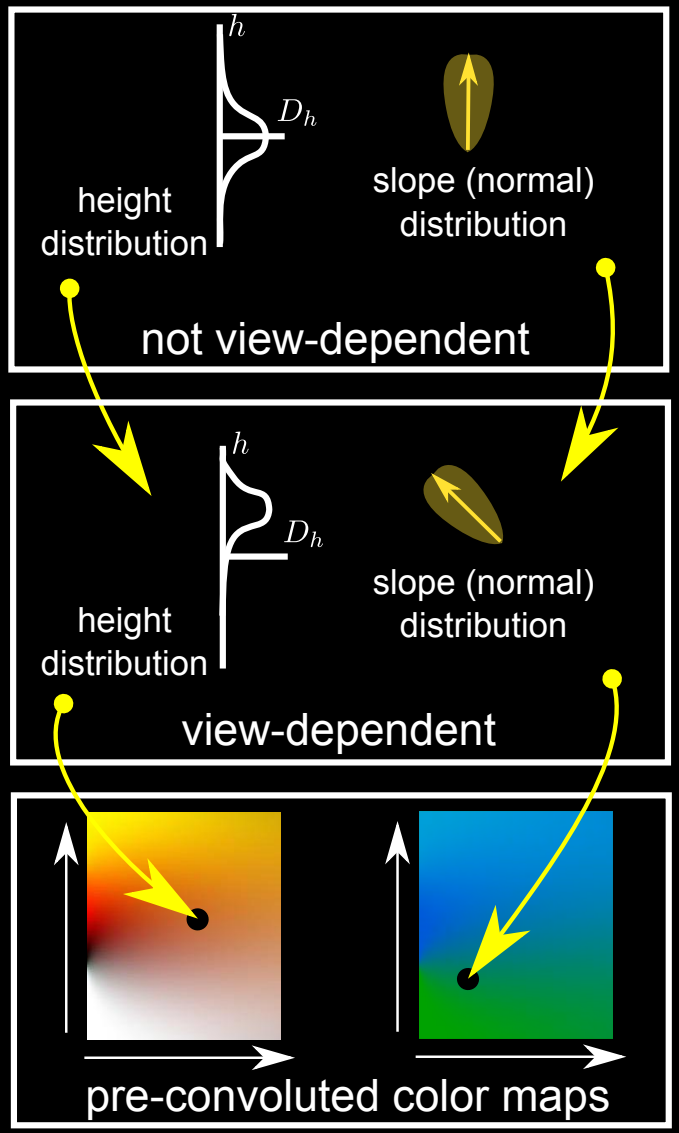
1. fetch / compute surface parameters



2. compute view-dependent surface parameters



3. fetch pre-convoluted color maps

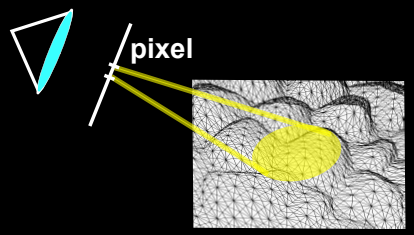


$\bar{h}$ average height (used for tessellation)

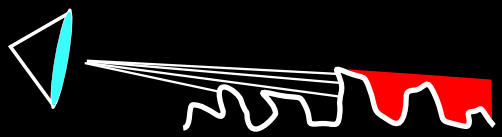
$\bar{h}^2$ height second momentum

II.5 Implementation

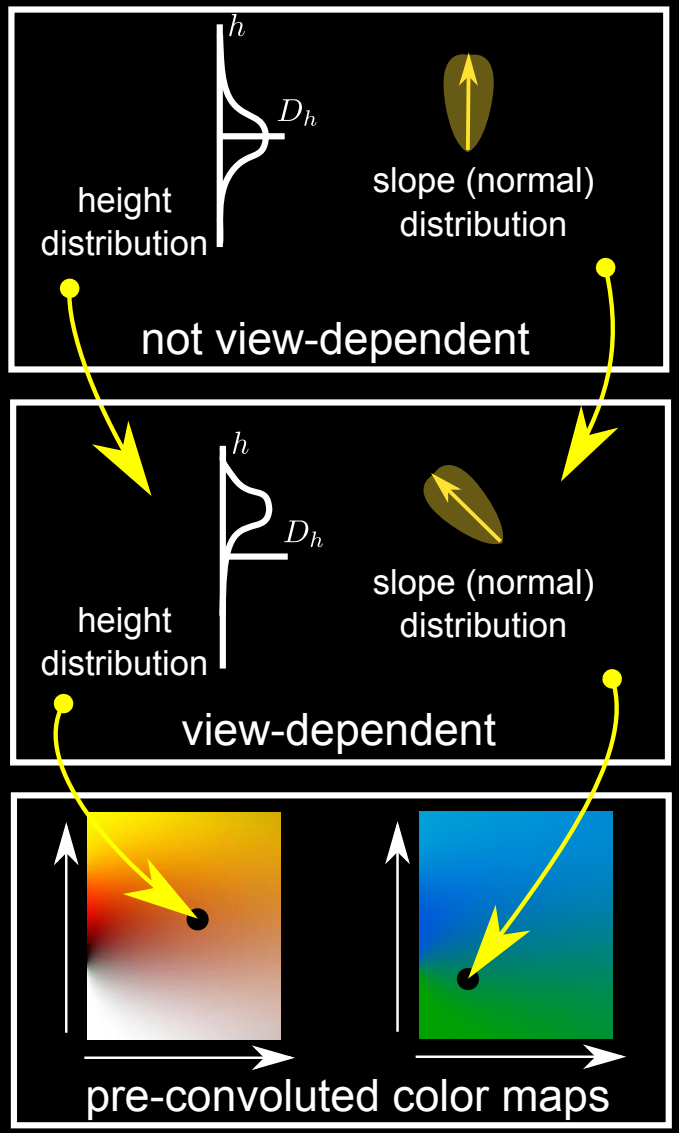
1. fetch / compute surface parameters



2. compute view-dependent surface parameters



3. fetch pre-convoluted color maps



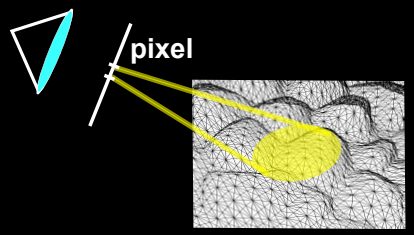
$\bar{h}$ average height (used for tessellation)

$\bar{h}^2$ height second momentum

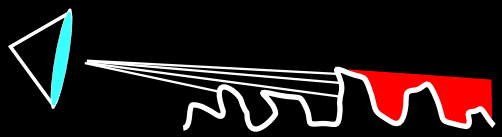
$\bar{s}_x$ $\bar{s}_y$ average slope/normal (used for shading)

II.5 Implementation

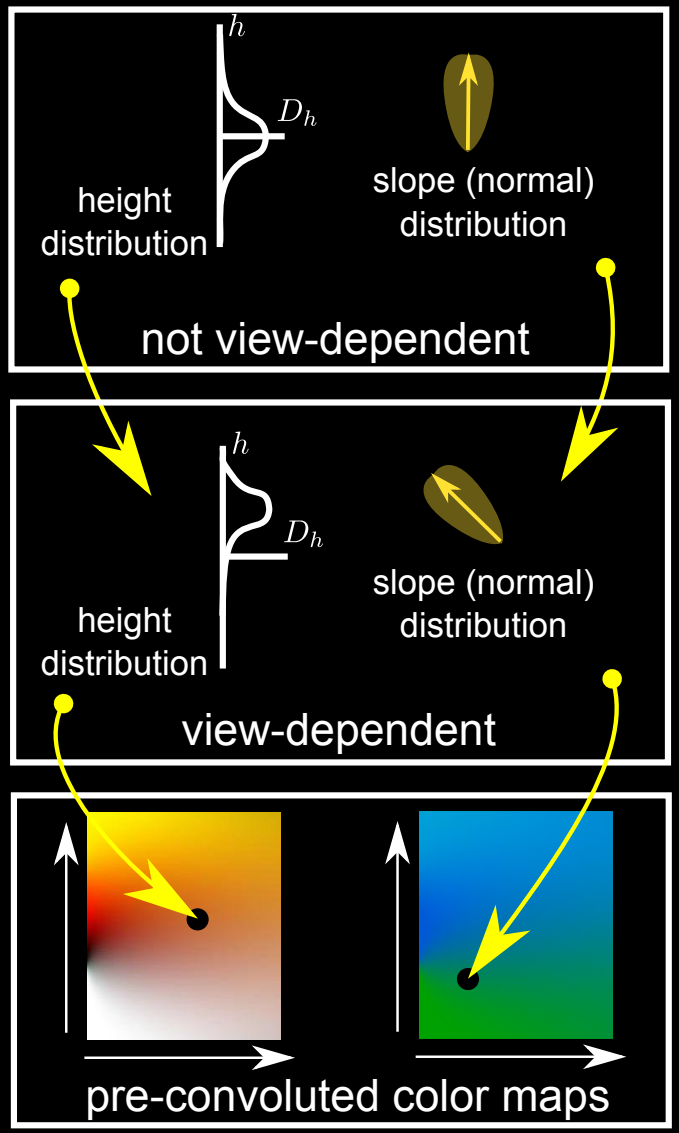
1. fetch / compute surface parameters



2. compute view-dependent surface parameters



3. fetch pre-convoluted color maps



$\bar{h}$ average height (used for tessellation)

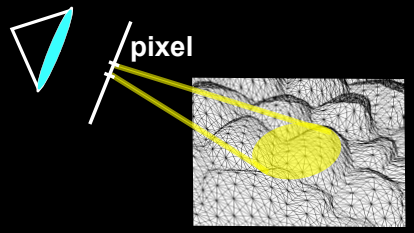
$\bar{h}^2$ height second momentum

$\bar{s}_x$ $\bar{s}_y$ average slope/normal (used for shading)

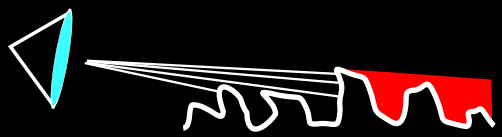
$\begin{bmatrix} \sigma_{s_x}^2 & c_{xy} \\ c_{xy} & \sigma_{s_y}^2 \end{bmatrix}$ slope covariance matrix (used in LEAN Mapping)

II.5 Implementation

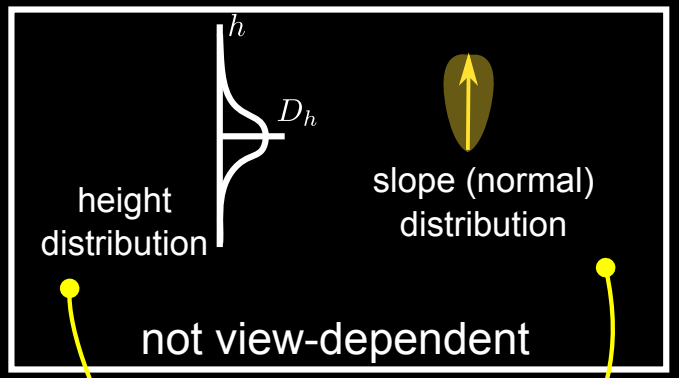
1. fetch / compute surface parameters

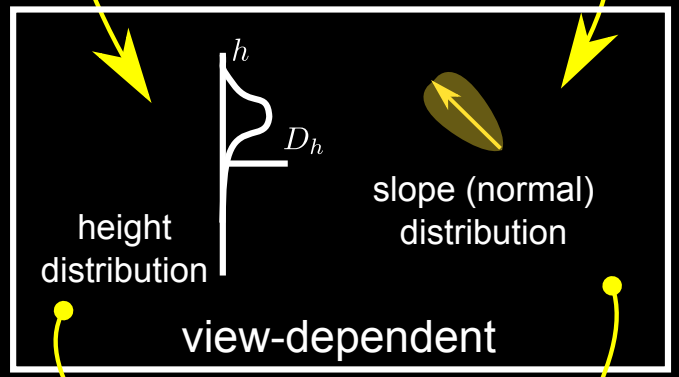


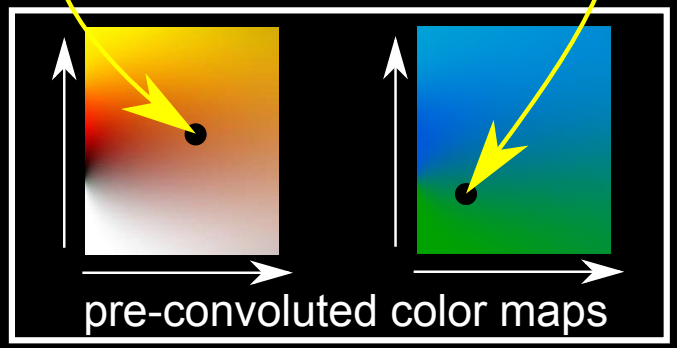
2. compute view-dependent surface parameters



3. fetch pre-convoluted color maps







$\bar{h}$ average height (used for tessellation)

$\bar{h}^2$ height second momentum

$\bar{s}_x$ $\bar{s}_y$ average slope/normal (used for shading)

$\begin{bmatrix} \sigma^2_{s_x} & c_{xy} \\ c_{xy} & \sigma^2_{s_y} \end{bmatrix}$ slope covariance matrix (used in LEAN Mapping)

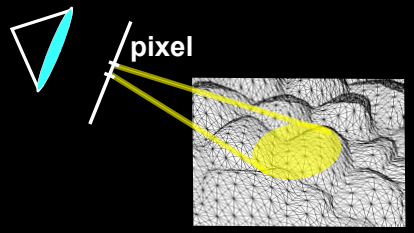
implement and call 2 functions

- function for height distribution

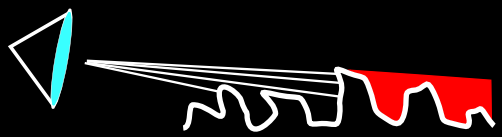
- function for slope distribution

II.5 Implementation

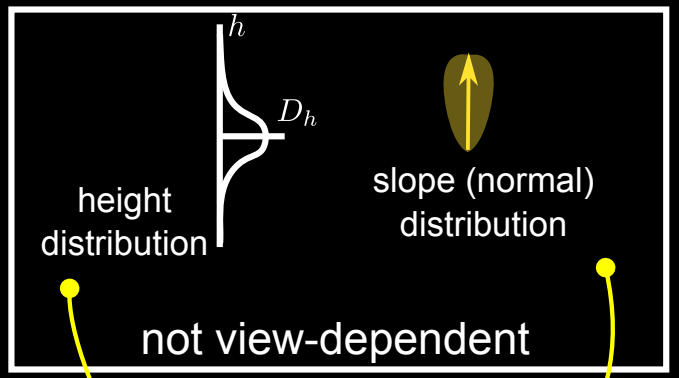
1. fetch / compute surface parameters

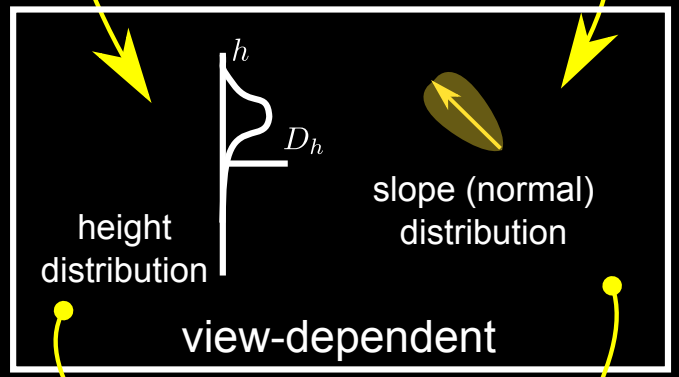


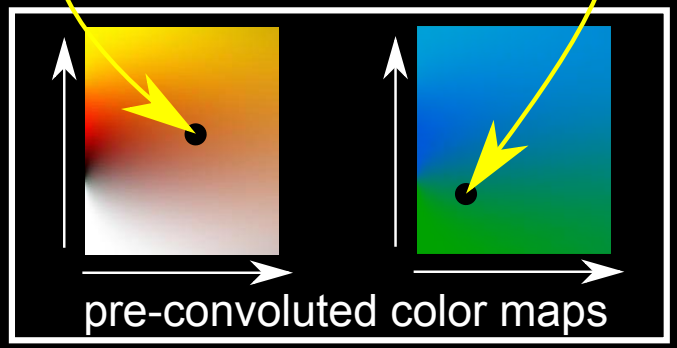
2. compute view-dependent surface parameters



3. fetch pre-convoluted color maps







$\bar{h}$ average height (used for tessellation)

$\bar{h}^2$ height second momentum

$\bar{s}_x$ $\bar{s}_y$ average slope/normal (used for shading)

$\begin{bmatrix} \sigma^2_{s_x} & c_{xy} \\ c_{xy} & \sigma^2_{s_y} \end{bmatrix}$ slope covariance matrix (used in LEAN Mapping)

implement and call 2 functions

- function for height distribution

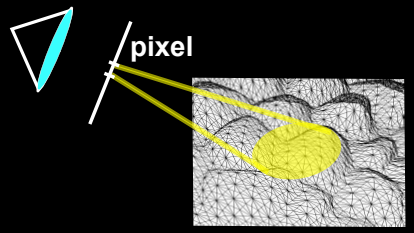
2 lines of code

(formula in the paper)

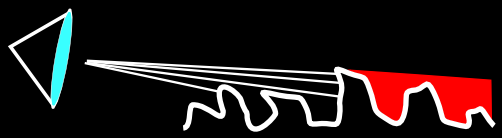
- function for slope distribution

II.5 Implementation

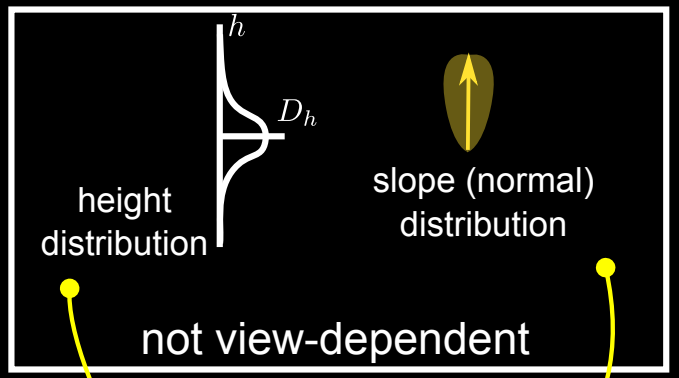
1. fetch / compute surface parameters

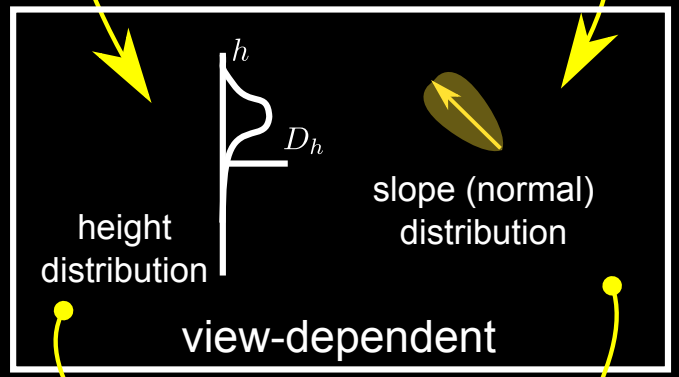


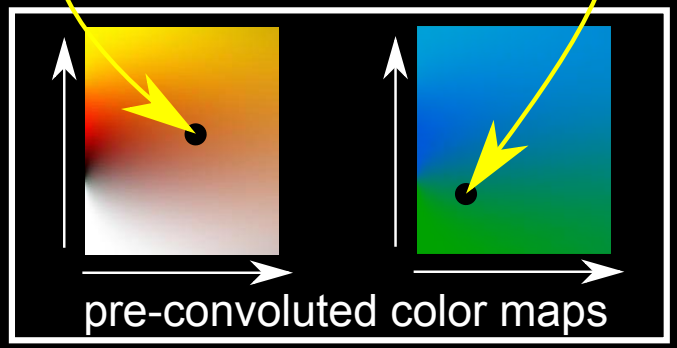
2. compute view-dependent surface parameters



3. fetch pre-convoluted color maps







$\bar{h}$ average height (used for tessellation)

$\bar{h}^2$ height second momentum

$\bar{s}_x$ $\bar{s}_y$ average slope/normal (used for shading)

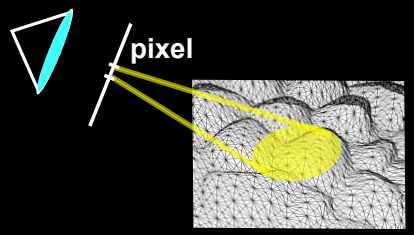
$\begin{bmatrix} \sigma^2_{s_x} & c_{xy} \\ c_{xy} & \sigma^2_{s_y} \end{bmatrix}$ slope covariance matrix (used in LEAN Mapping)

implement and call 2 functions

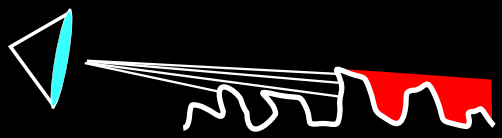
- function for height distribution
  2 lines of code
  (formula in the paper)
- function for slope distribution
  given in supplemental material

II.5 Implementation

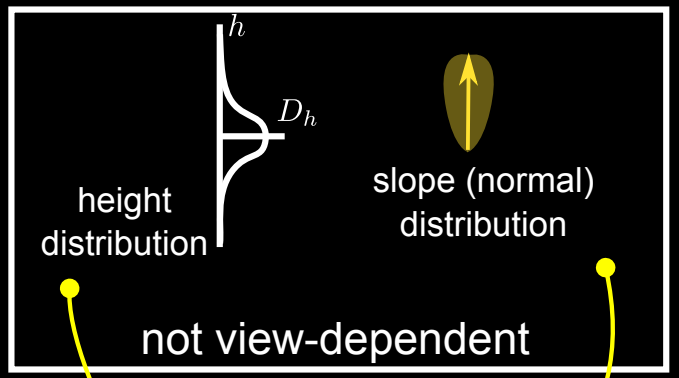
1. fetch / compute surface parameters

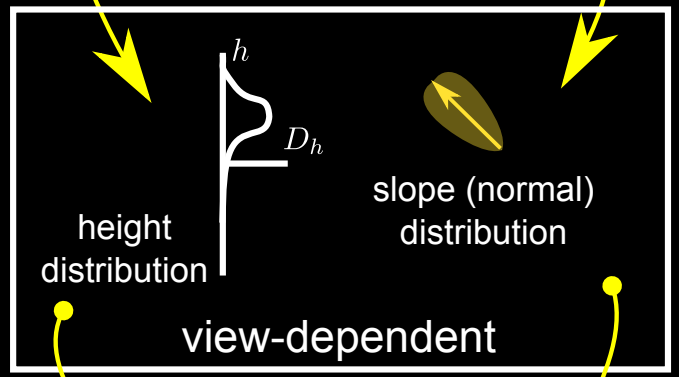


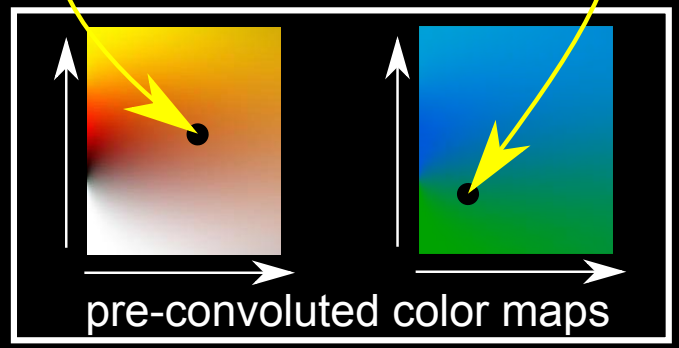
2. compute view-dependent surface parameters



3. fetch pre-convoluted color maps







$\bar{h}$ average height (used for tessellation)

$\bar{h}^2$ height second momentum

$\bar{s}_x$ $\bar{s}_y$ average slope/normal (used for shading)

$\begin{bmatrix} \sigma_{s_x}^2 & c_{xy} \\ c_{xy} & \sigma_{s_y}^2 \end{bmatrix}$ slope covariance matrix (used in LEAN Mapping)

implement and call 2 functions

- function for height distribution
  2 lines of code
  (formula in the paper)
- function for slope distribution
  given in supplemental material

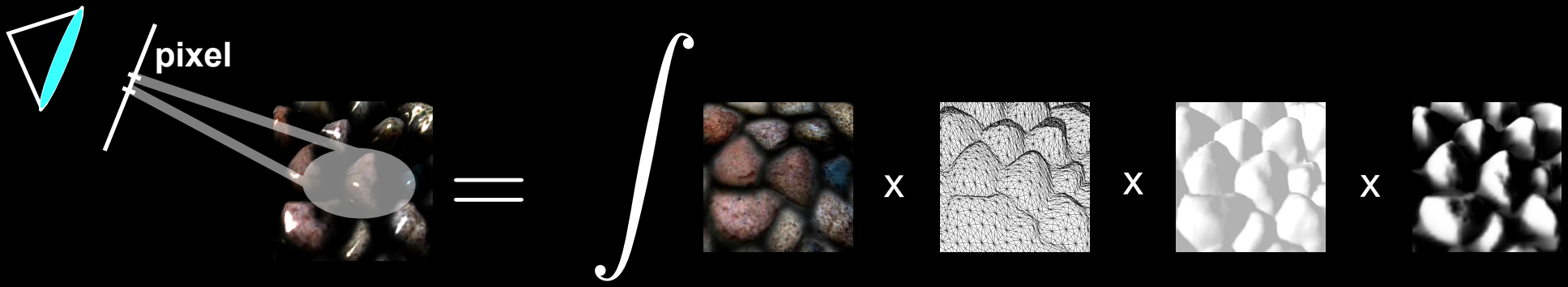
pre-convoluted color maps

in practice we use 256^2 textures with 4 color components
= 256KB per pre-convoluted

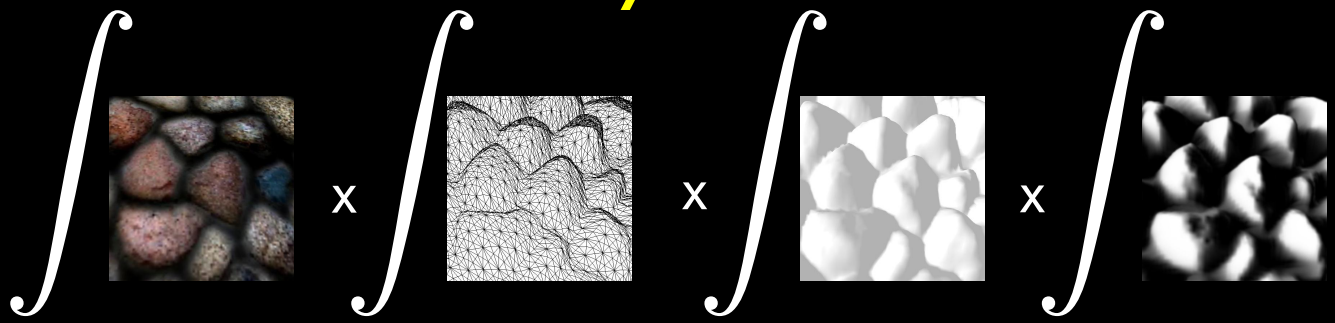
Conclusion and Future Work

Introduction

physically correct



$$\neq$$



physically wrong

energy conservation violated

popping, aliasing, and inconsistent appearance

Conclusion and Future Work

Our LoD framework

- colors/material
- smale-scale geometry
- visibility
- no BRDF

Conclusion and Future Work

Our LoD framework

- colors/material
- smale-scale geometry
- visibility
- no BRDF

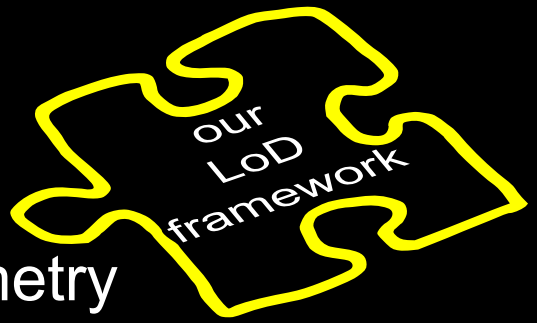
Existing
physically-based rendering

- physically-based BRDFs
- physically incorrect LoD

Conclusion and Future Work

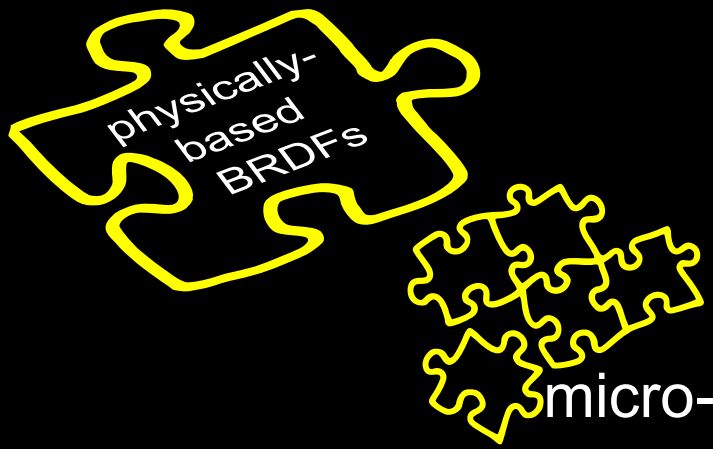
Our LoD framework

- colors/material
- smale-scale geometry
- visibility
- no BRDF



Existing
physically-based rendering

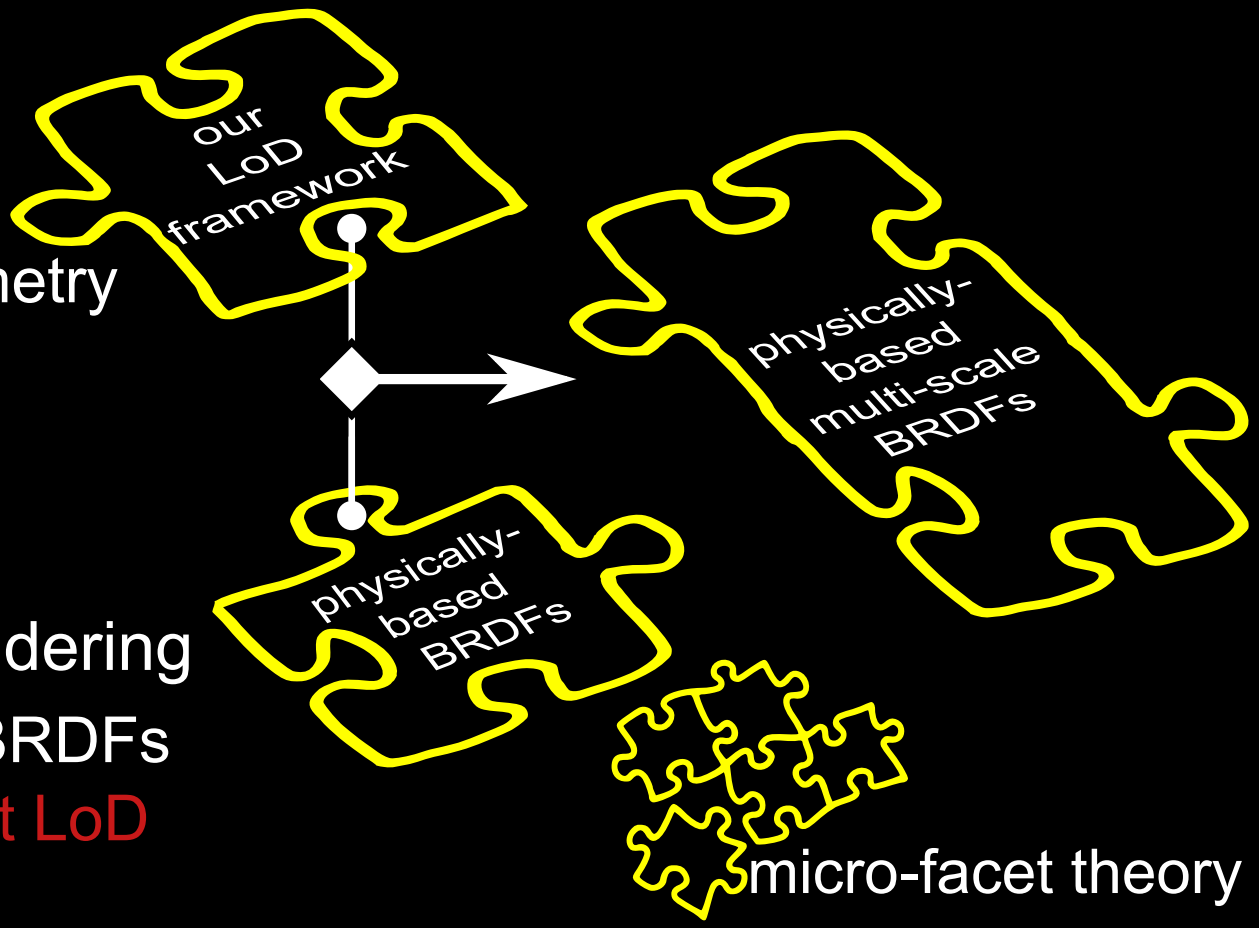
- physically-based BRDFs
- physically incorrect LoD

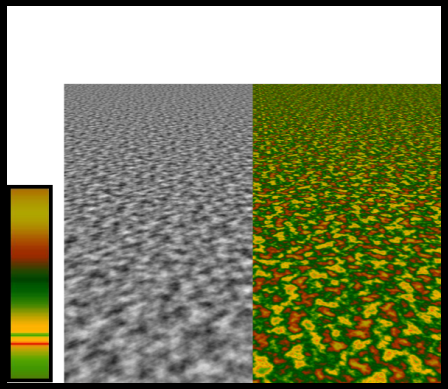


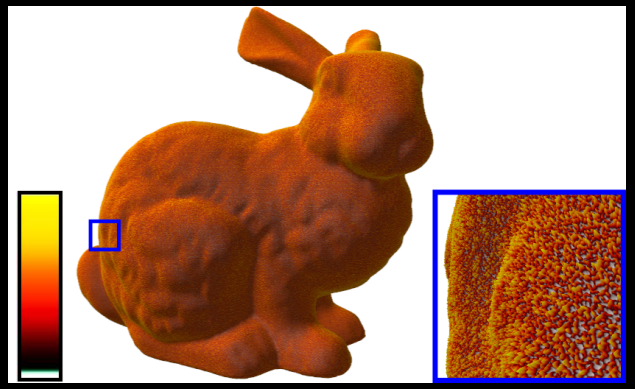
Conclusion and Future Work

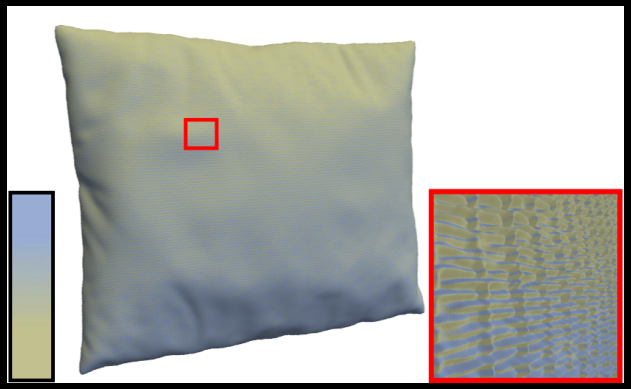
Our LoD framework
- colors/material
- smale-scale geometry
- visibility
- no BRDF

Existing
physically-based rendering
- physically-based BRDFs
- physically incorrect LoD

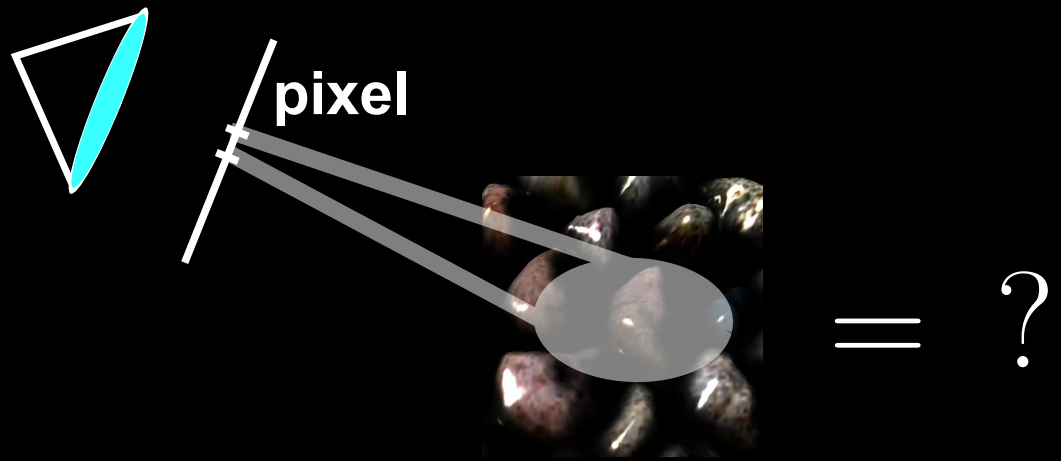








Thank you!



$= \; ?$