

# コンピュータグラフィックス論

## － 画像処理(1) －

2018年7月12日

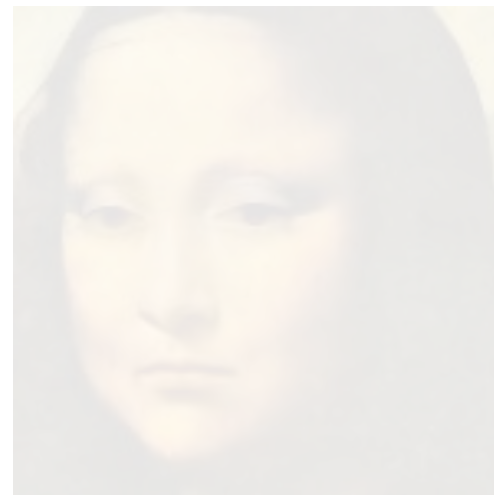
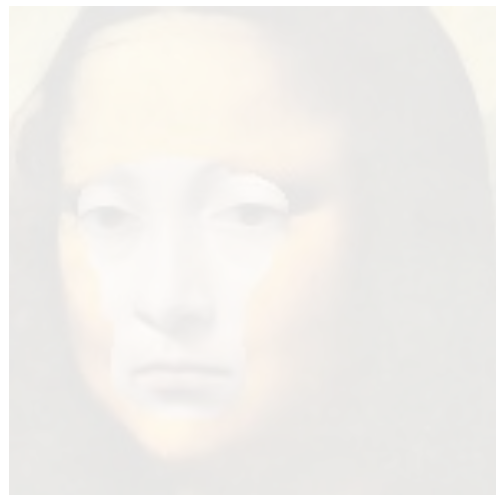
高山 健志

# 本日のトピック

- Edge-aware な画像処理



- Gradient-domain の画像処理



# Gaussian Filter による画像平滑化

- 「滑らかさ」パラメタ  $\sigma$



元画像



$\sigma = 2$



$\sigma = 5$



$\sigma = 10$

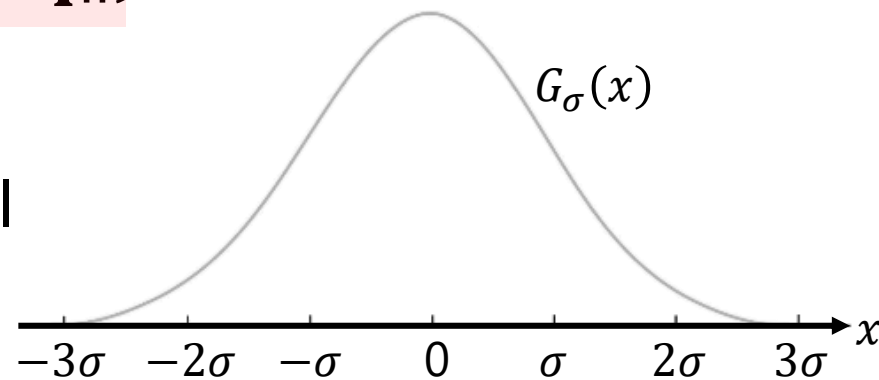
# Gaussian Filter の数式

- 画像  $I$  のピクセル位置  $\mathbf{p} = (p_x, p_y) \in \Omega$  における画素値を  $I_{\mathbf{p}}$  で表す
  - 解像度  $640 \times 480$  の場合、 $\Omega := \{1, \dots, 640\} \times \{1, \dots, 480\}$
- パラメタ  $\sigma$  による Gaussian Filter 適用後の画像を  $\text{GF}_{\sigma}[I]$  で表す

$$\text{GF}_{\sigma}[I]_{\mathbf{p}} := \frac{\sum_{\mathbf{q} \in \Omega} G_{\sigma}(\|\mathbf{p} - \mathbf{q}\|) I_{\mathbf{q}}}{\sum_{\mathbf{q} \in \Omega} G_{\sigma}(\|\mathbf{p} - \mathbf{q}\|)}$$

$W_{\mathbf{p}}$

- $G_{\sigma}(x) := \exp\left(-\frac{x^2}{2\sigma^2}\right)$  ← 半径  $\sigma$  の Gaussian Kernel

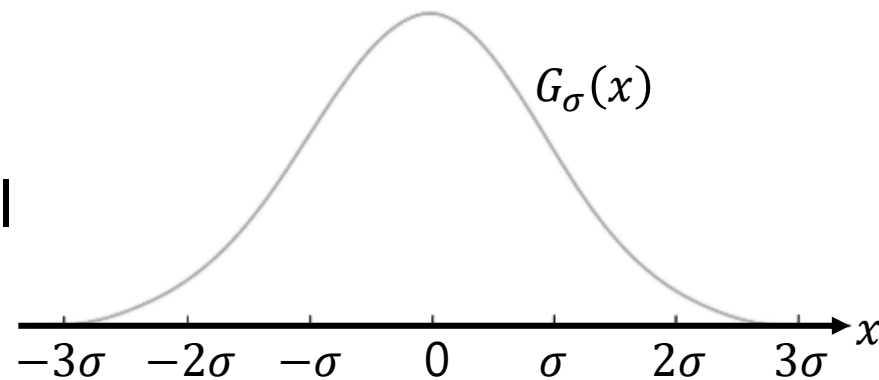


# Gaussian Filter の数式

- 画像  $I$  のピクセル位置  $\mathbf{p} = (p_x, p_y) \in \Omega$  における画素値を  $I_{\mathbf{p}}$  で表す
  - 解像度  $640 \times 480$  の場合、 $\Omega := \{1, \dots, 640\} \times \{1, \dots, 480\}$
- パラメタ  $\sigma$  による Gaussian Filter 適用後の画像を  $\text{GF}_{\sigma}[I]$  で表す

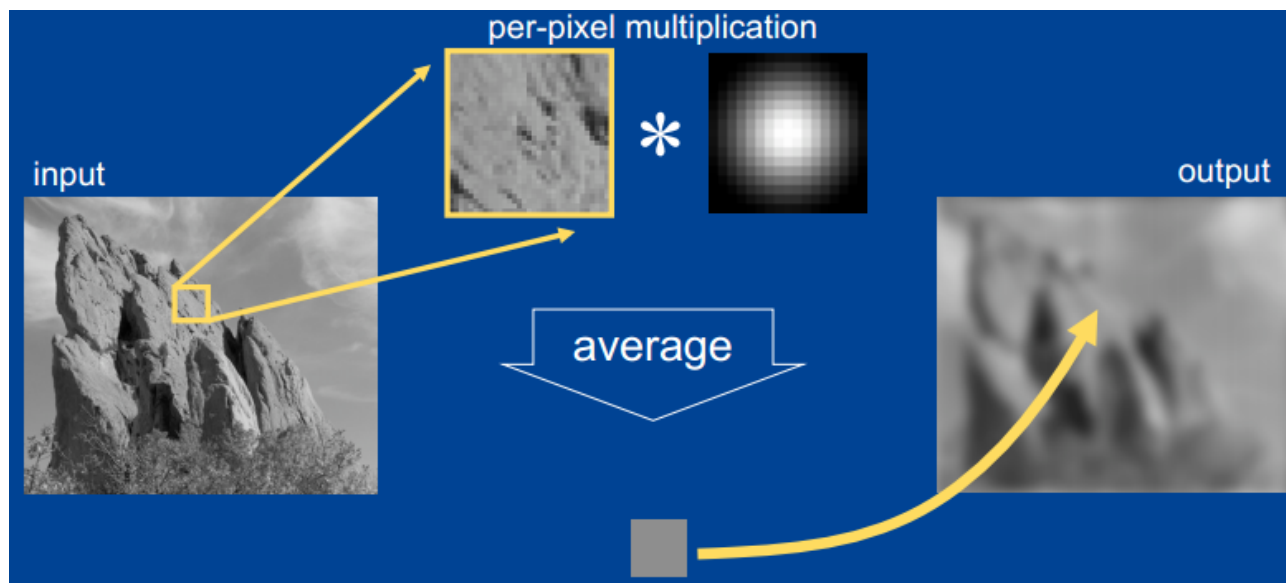
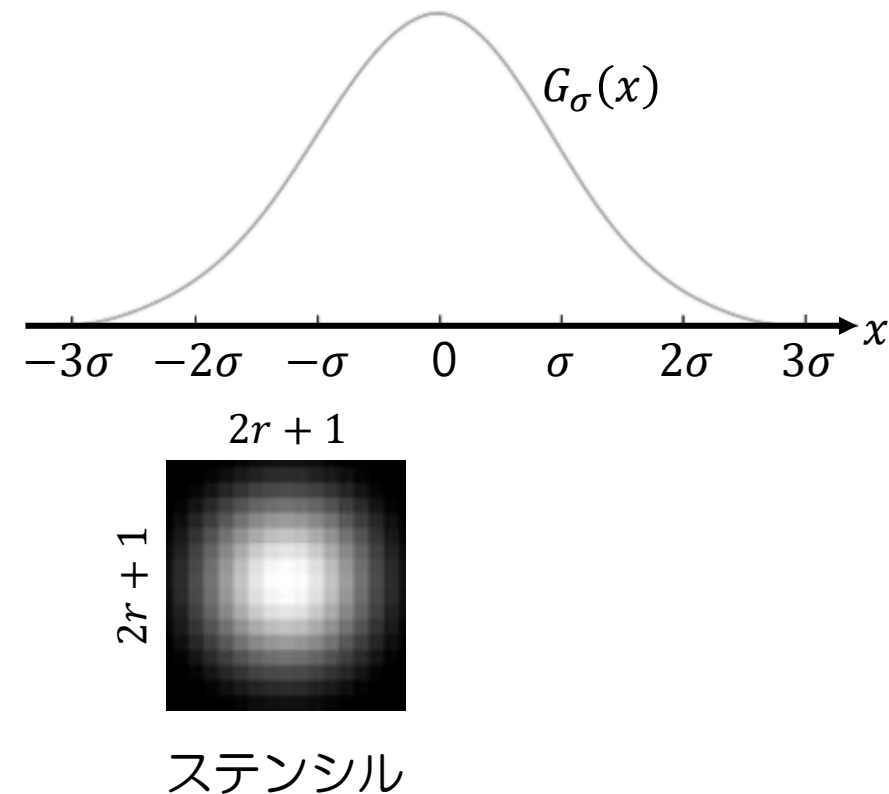
$$\text{GF}_{\sigma}[I]_{\mathbf{p}} := \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in \Omega} G_{\sigma}(\|\mathbf{p} - \mathbf{q}\|) I_{\mathbf{q}}$$

- $G_{\sigma}(x) := \exp\left(-\frac{x^2}{2\sigma^2}\right)$  ← 半径  $\sigma$  の Gaussian Kernel



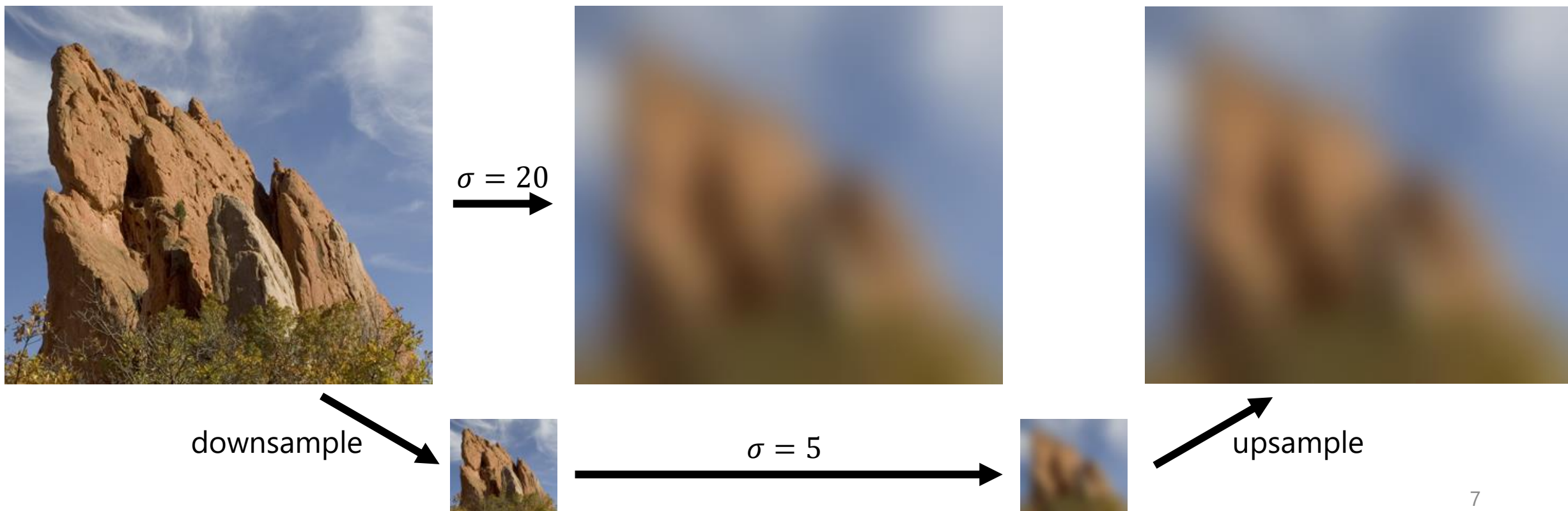
# Gaussian Filter の実装

- $G_\sigma(3\sigma) \approx 0 \rightarrow$  遠くのピクセルは無視できる
- $r := \text{ceil}(3\sigma)$  として  $(2r + 1) \times (2r + 1)$  のステンシル上で重みを前計算



# Kernel 半径 $\sigma$ が非常に大きい場合

- そのまま計算すると時間がかかる
- 代替法：downsample  $\rightarrow$  小さい  $\sigma$  で平滑化  $\rightarrow$  upsample





# Detail Extraction & Enhancement



-



smoothed

=



detail



+ 3 ×



detail

=



enhanced



# Edge-aware な画像平滑化を使うと・・・

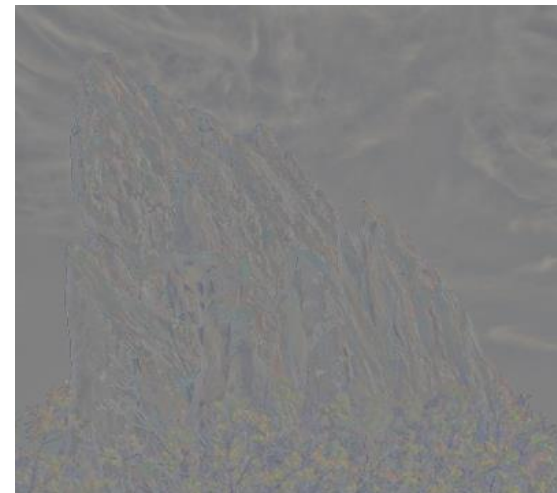


-



smoothed

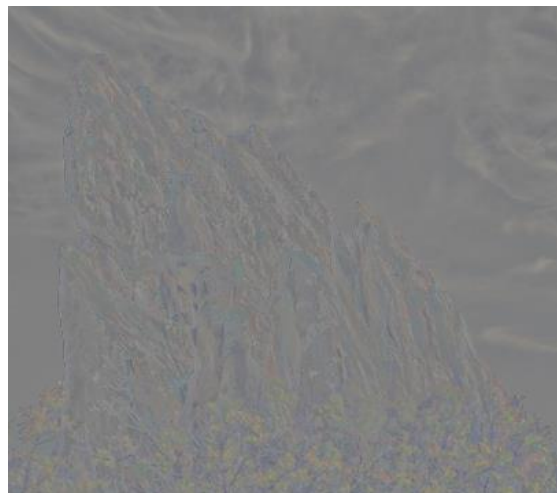
=



detail



+ 3 ×



detail

=



enhanced

# Bilateral Filter による edge-aware な平滑化

- 二つのパラメタ

- $\sigma_s$  : ピクセルの **位置** に関する平滑化の範囲
- $\sigma_r$  : ピクセルの **色** に関する平滑化の範囲

$$\text{BF}_{\sigma_s, \sigma_r}[I]_{\mathbf{p}} := \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in \Omega} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) G_{\sigma_r}(\|I_{\mathbf{p}} - I_{\mathbf{q}}\|) I_{\mathbf{q}}$$

すべて  $\sigma_s = 10$



元画像



$\sigma_r = 32$



$\sigma_r = 128$



$\sigma_r = 512_{10}$

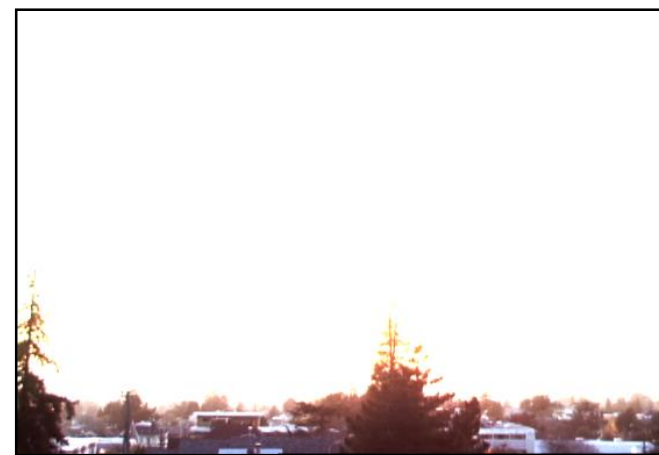
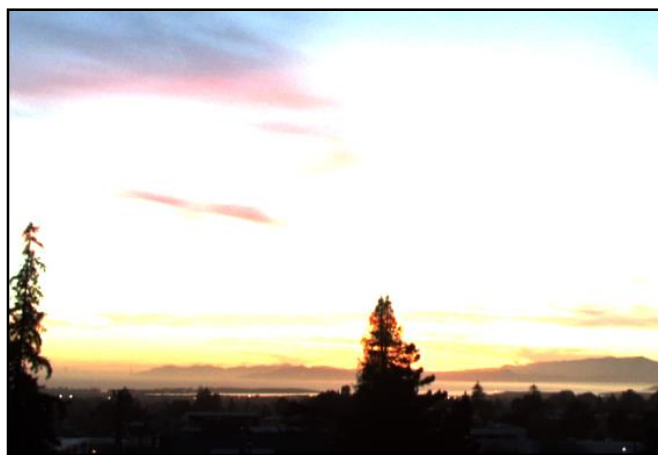
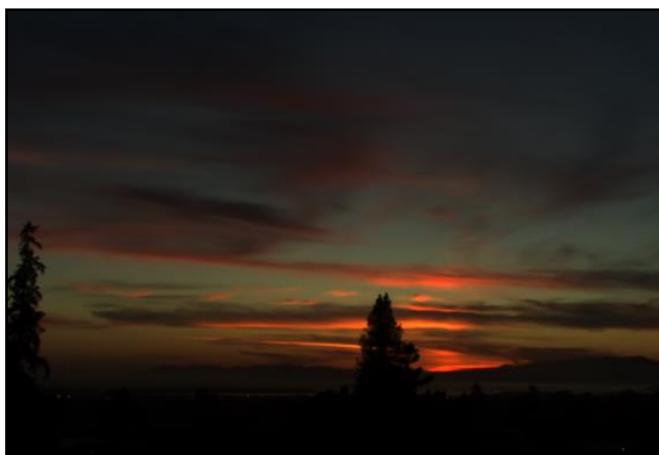
# Bilateral Filter の応用 : Stylization





# Bilateral Filter の応用：Tone Mapping

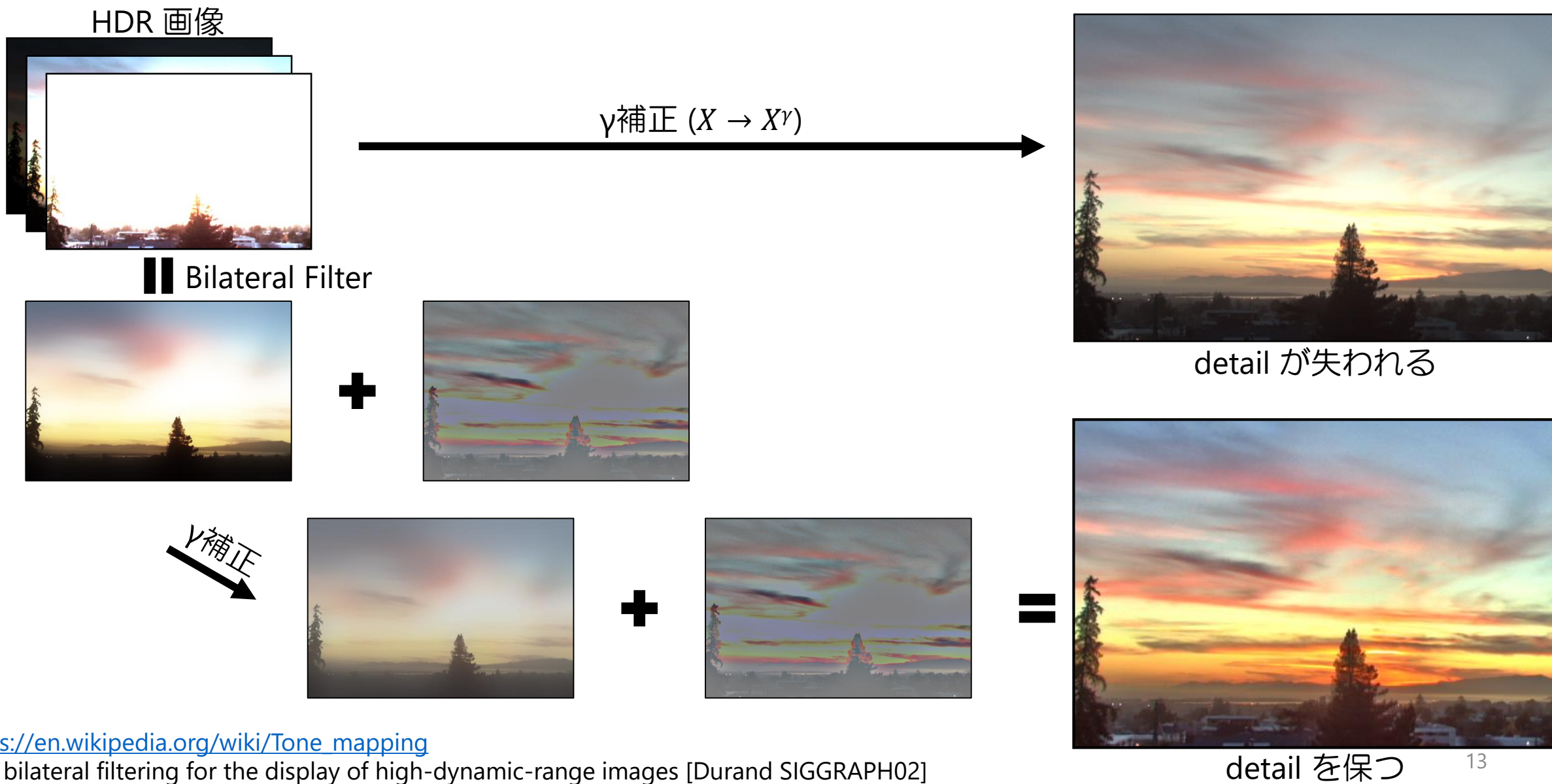
- 24bitカラー画像の各成分の範囲：1~255
- 現実世界の光の強さの範囲：1~ $10^5$ 
  - **H**igh **D**ynamic **R**ange 画像
  - 露光時間を変えて撮影することで計測可能



短い露光時間

長い露光時間

# Bilateral Filter の応用 : Tone Mapping



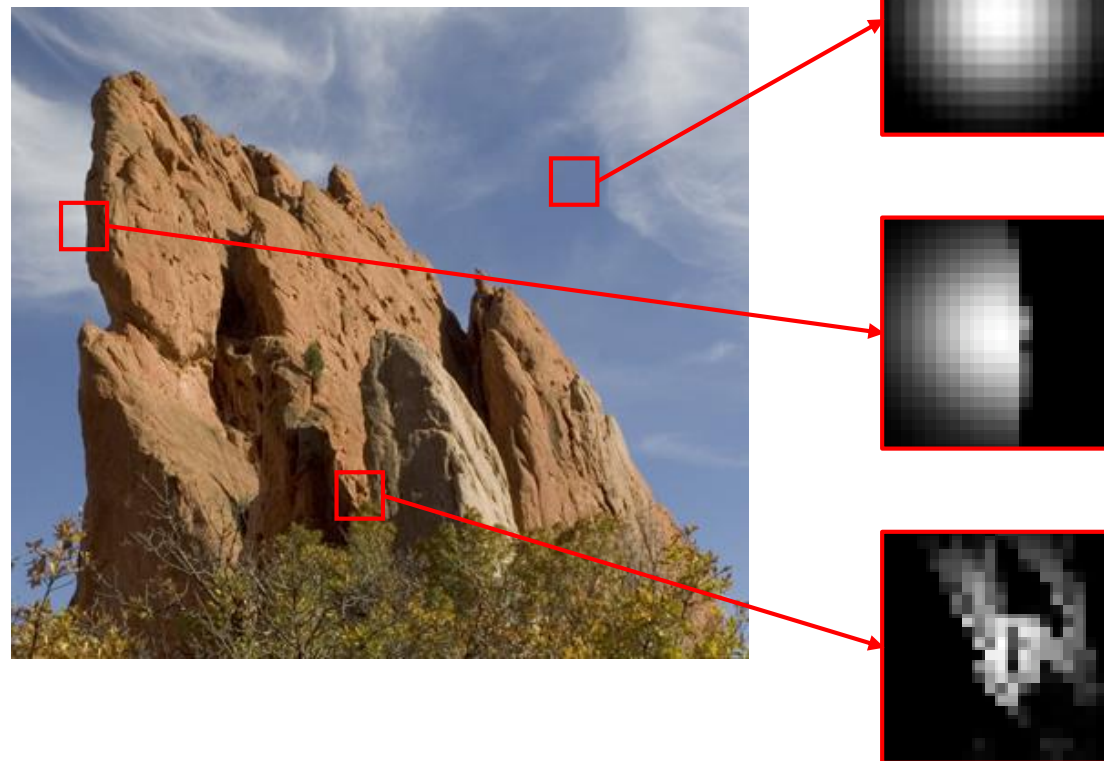


# Bilateral Filter のナイスな実装

$$\sum_{\mathbf{q} \in \Omega} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) G_{\sigma_r}(\|I_{\mathbf{p}} - I_{\mathbf{q}}\|) I_{\mathbf{q}}$$

- ピクセル位置  $\mathbf{p} \in \Omega$  ごとに  
ステンシルの再計算が必要  
→ 遅い

- (基本課題)



# Bilateral Filter に対するもう一つの見方

- ピクセル位置  $\mathbf{p}$  と画素値  $I_{\mathbf{p}}$  から特徴ベクトル  $\mathbf{f}_{\mathbf{p}} := \left( \frac{\mathbf{p}}{\sigma_s}, \frac{I_{\mathbf{p}}}{\sigma_r} \right)$  を定義

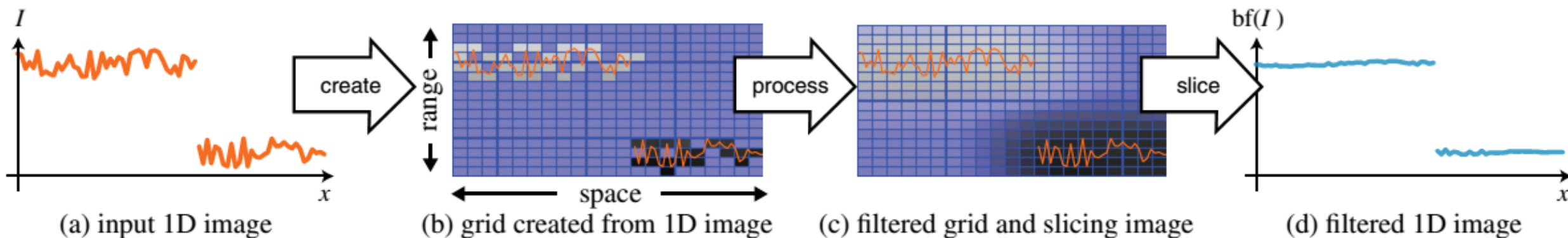
- Bilateral Filter の重みは、特徴ベクトル同士の Euclid 距離を Gaussian Kernel に代入したものに等しい

$$\begin{aligned} & G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) G_{\sigma_r}(\|I_{\mathbf{p}} - I_{\mathbf{q}}\|) \\ &= \exp\left(-\frac{\|\mathbf{p} - \mathbf{q}\|^2}{2\sigma_s^2}\right) \exp\left(-\frac{\|I_{\mathbf{p}} - I_{\mathbf{q}}\|^2}{2\sigma_r^2}\right) \\ &= \exp\left(-\frac{\|\mathbf{f}_{\mathbf{p}} - \mathbf{f}_{\mathbf{q}}\|^2}{2}\right) \\ &= G_1(\|\mathbf{f}_{\mathbf{p}} - \mathbf{f}_{\mathbf{q}}\|) \end{aligned}$$

- Bilateral Filter は、特徴空間におけるサンプル集合  $\{\mathbf{f}_{\mathbf{p}}\}$  に対して半径 1 の Gaussian Filter をかけるのと同義  
→ 計算が単純化

# Bilateral Grid [Paris06; Chen07]

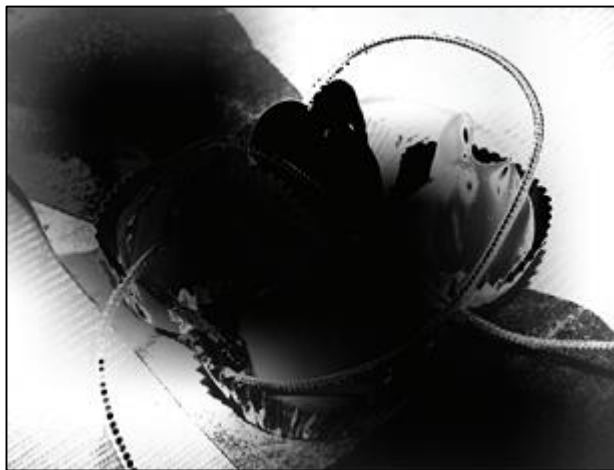
- 3D 特徴ベクトルを (X座標, Y座標, 輝度) として定義し、サンプル集合  $\{f_p\}$  を 3D 配列上にマッピング
- $\sigma_s$  と  $\sigma_r$  が大きいほど、配列の解像度を低くできる → 計算コスト低減



# 特徴空間を介した重みマップの生成



白い scribble → 重み=1 の制約  
黒い scribble → 重み=0 の制約



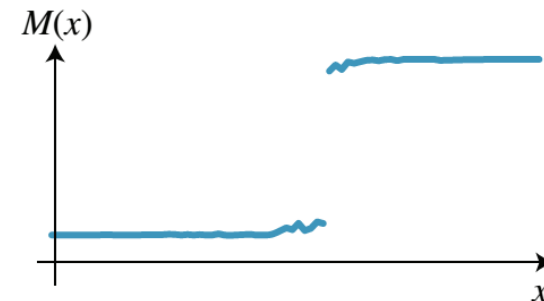
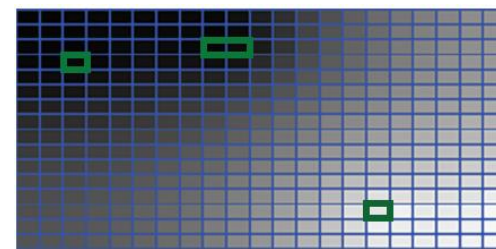
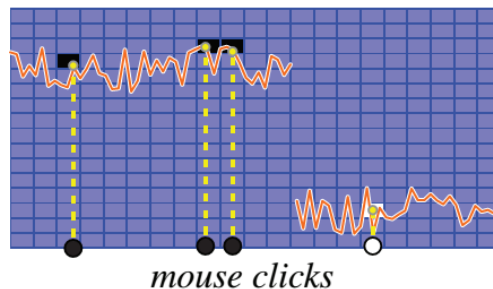
重みマップ



利用例：色味の変更

- 様々な呼ばれ方：Edit Propagation, Matting, Segmentation

- Bilateral Grid 上で Laplace 方程式を解く



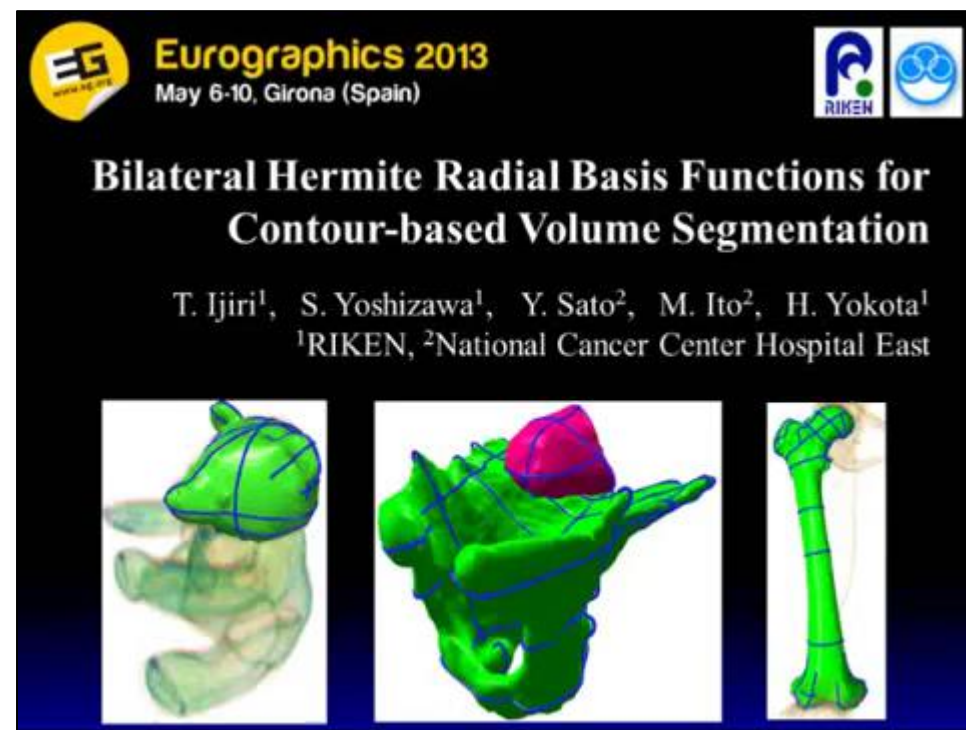


# 特徴空間を介した重みマップの生成

RBF で補間 [Li10]  
(目的：画像と動画の編集)



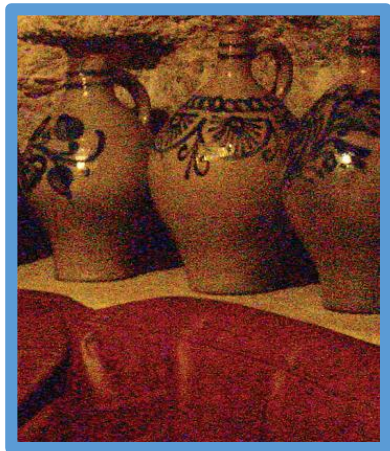
Hermite RBF で補間 [Ijiri13]  
(目的：CT volume の領域分割)



[https://www.youtube.com/watch?v=mL6ig\\_OaQAA](https://www.youtube.com/watch?v=mL6ig_OaQAA)



# Bilateral Filter の拡張 : Joint (Cross) Bilateral Filter



フラッシュ無し写真 A

☺ 色味は良い

☹ ノイズが大きい、ボケ気味

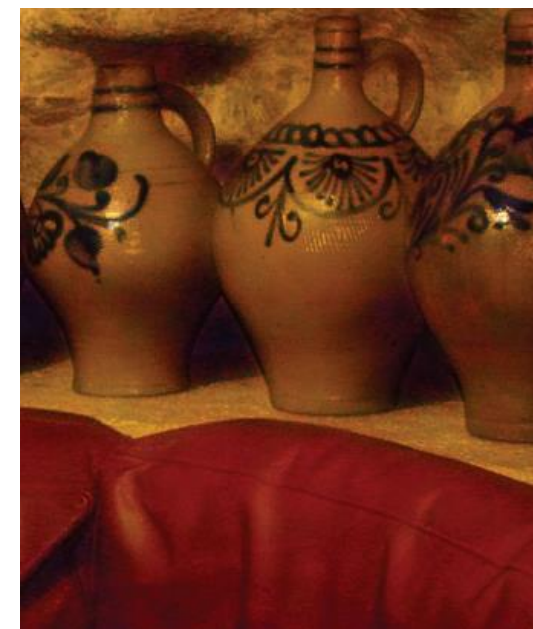


フラッシュ有り写真 F

☹ 色味は悪い

☺ ノイズが小さい、クッキリ

JBF 適用結果



$$\text{JBF}_{\sigma_s, \sigma_r}(A, F)_p := \frac{1}{W_p} \sum_{q \in \Omega} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(\|F_p - F_q\|) A_q$$

# Bilateral Filter の拡張：Non-Local Means Filter

- ピクセル  $p$  を中心とする  $7 \times 7$  領域の画素値から成る  
近傍ベクトル  $\mathbf{n}_p$  によって、特徴空間を定義

$$\text{NLMF}_\sigma(I)_p := \frac{1}{W_p} \sum_{q \in \Omega} G_\sigma(\|\mathbf{n}_p - \mathbf{n}_q\|) I_q$$



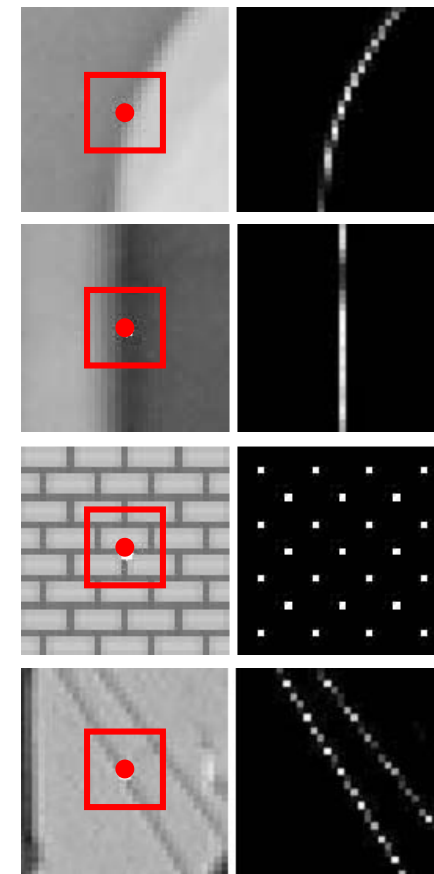
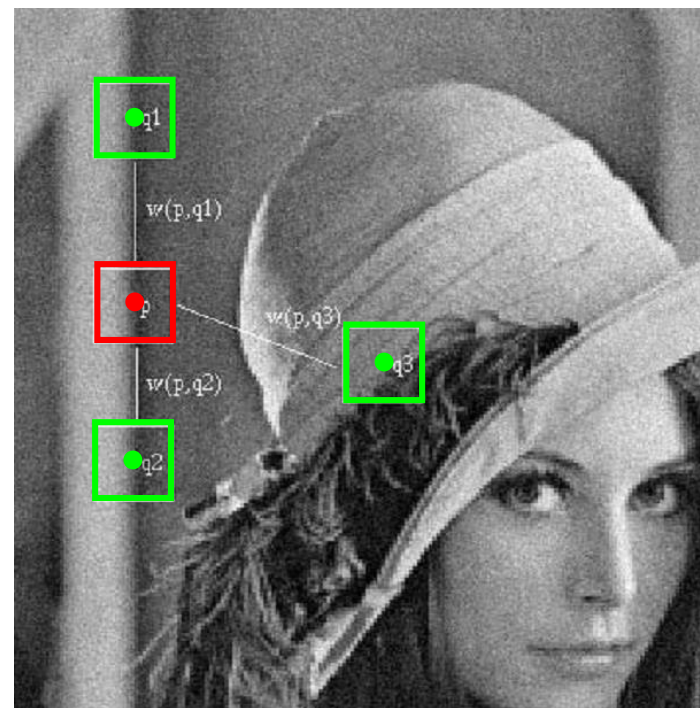
Noisy input



Bilateral

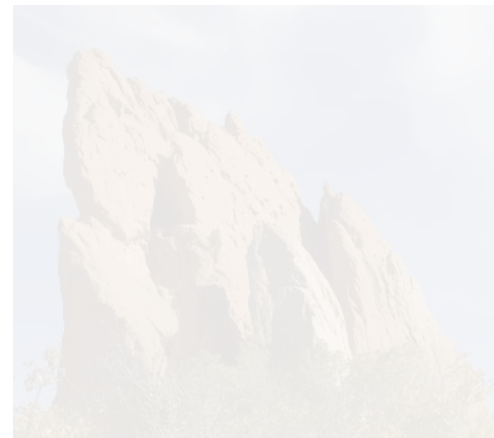
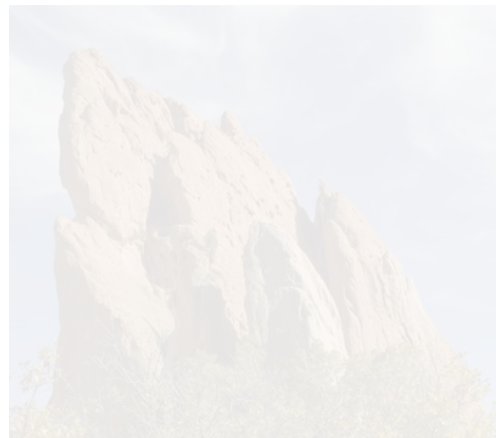


NL Means



# 本日のトピック

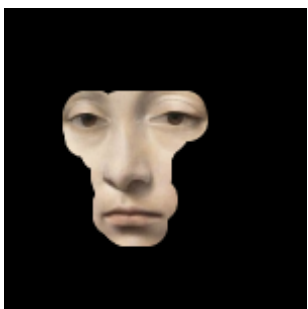
- Edge-aware な画像処理



- Gradient-domain の画像処理



# シナリオ：Source 画像を Dest. 画像へ挿入



Source



Dest.



単純な上書き



境界をぼかしてみる



Gradient-domain 処理

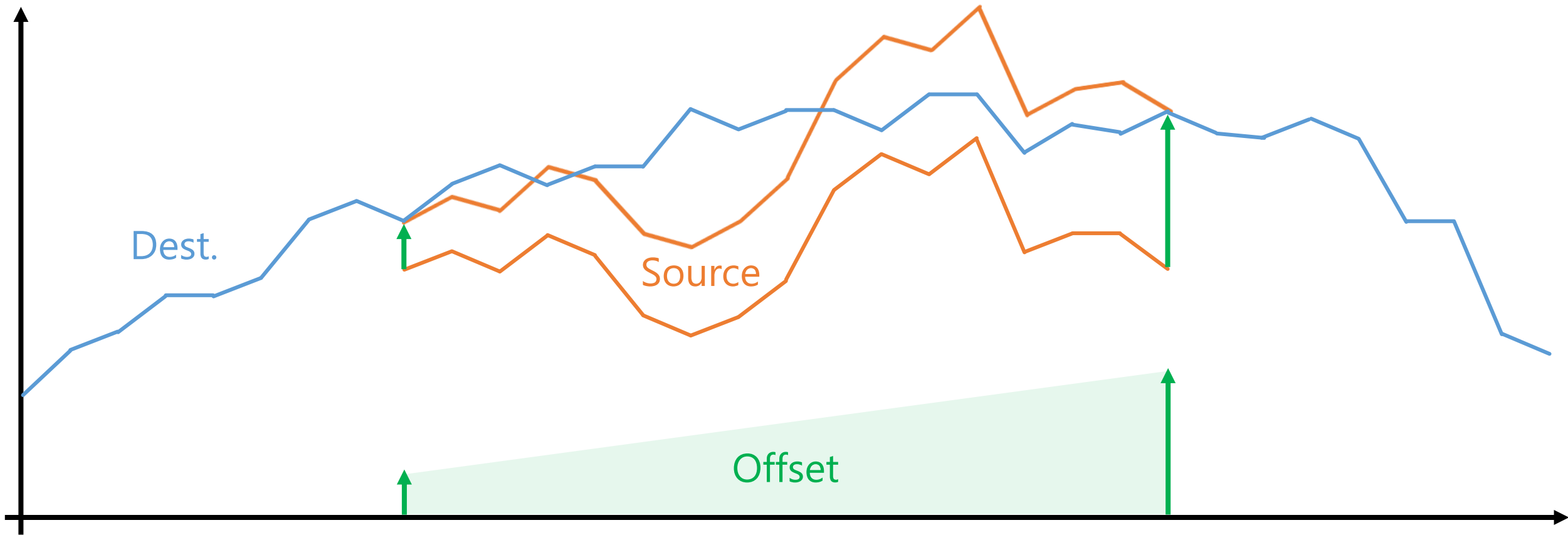


# シナリオ：複数写真からパノラマ合成





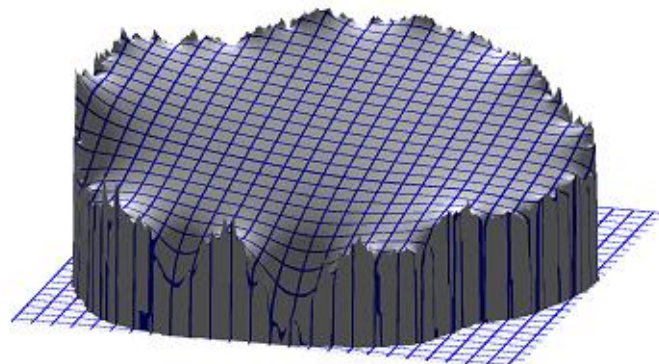
# 1D grayscale 画像の場合の考察



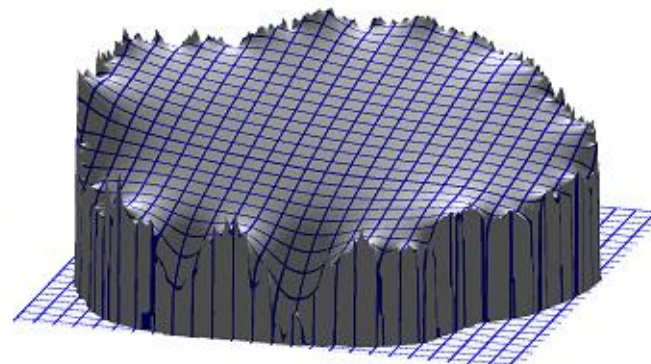
# 2D の場合：Offset by Laplace Membrane



(a) Source patch

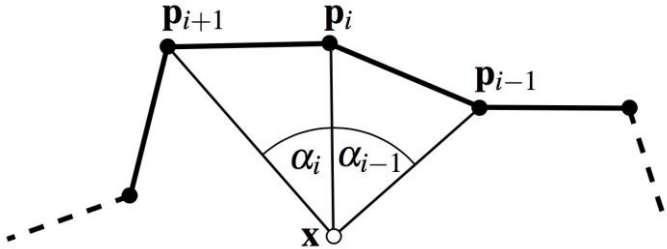


(b) Laplace membrane



(c) Mean-value membrane

- ディリクレ境界条件の下で Laplace 方程式を解く
- Mean Value Coordinatesを用いた高速な近似

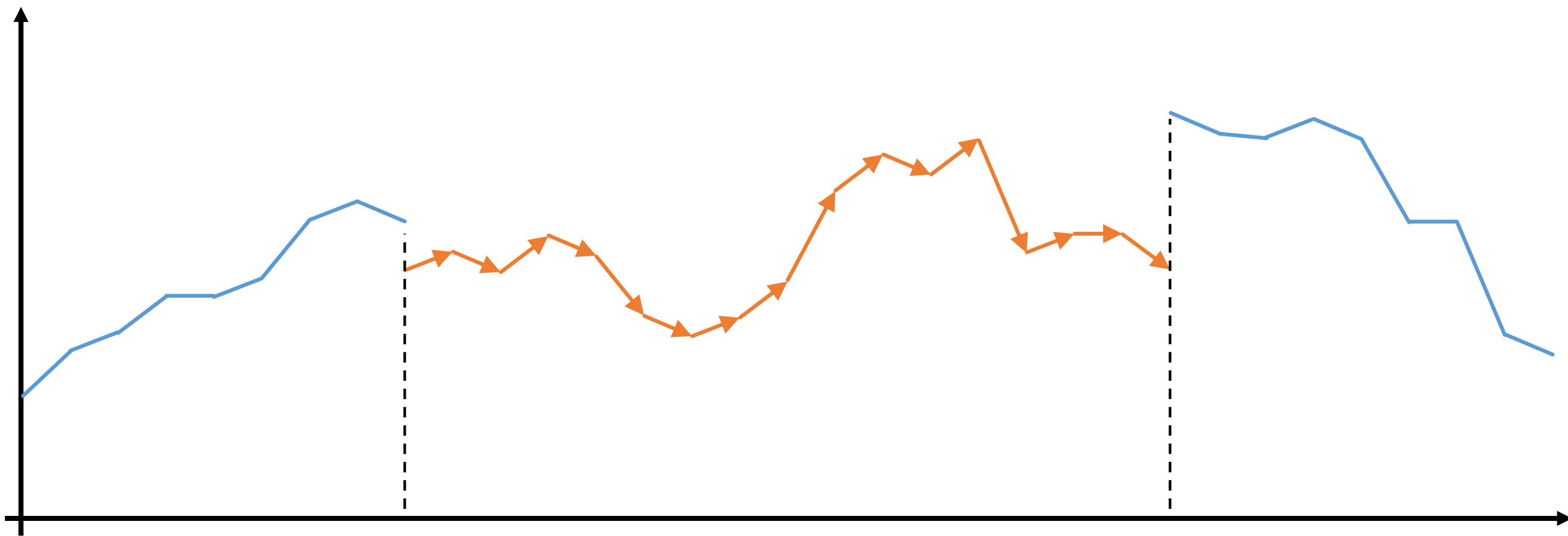

$$\lambda_i(\mathbf{x}) \propto \frac{\tan(\alpha_{i-1}/2) + \tan(\alpha_i/2)}{\|\mathbf{p}_i - \mathbf{x}\|}$$
$$\tilde{f}(\mathbf{x}) = \sum_{i=0}^{m-1} \lambda_i(\mathbf{x}) f(\mathbf{p}_i)$$

## Coordinates for Instant Image Cloning

Zeev Farbman  
Gil Hoffer  
Yaron Lipman  
Daniel Cohen-Or  
Dani Lischinski

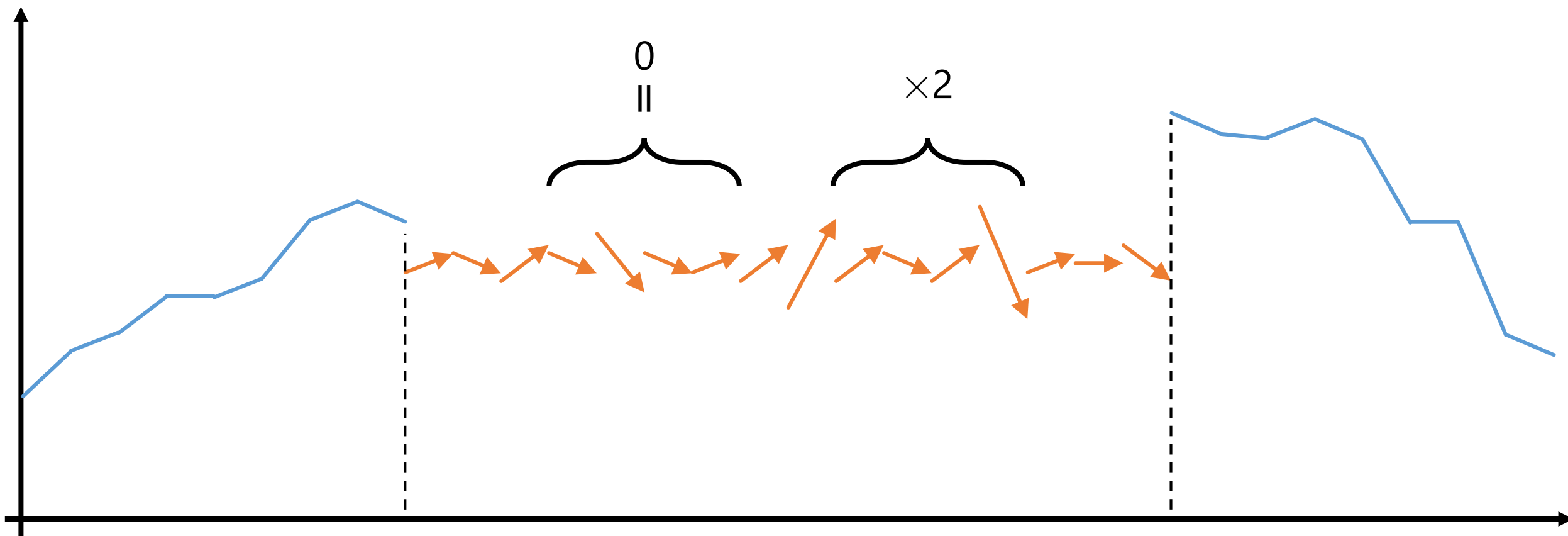
ACM Transactions on Graphics  
(Proceedings of ACM SIGGRAPH 2009)

# 単純な cloning 以外の gradient-domain 処理



# 単純な cloning 以外の gradient-domain 処理

Gradient を好き勝手に操作する！



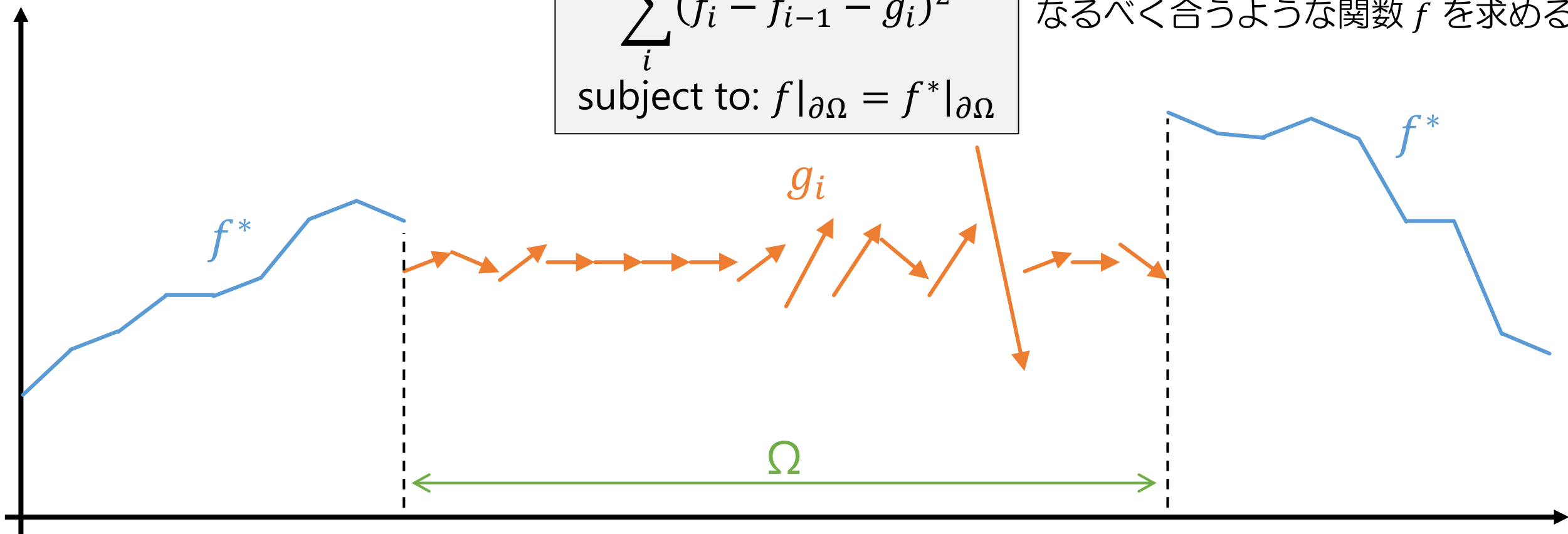
# 単純な cloning 以外の gradient-domain 処理

Find  $\{f_i\}$  that minimize

$$\sum_i (f_i - f_{i-1} - g_i)^2$$

subject to:  $f|_{\partial\Omega} = f^*|_{\partial\Omega}$

ユーザが与えた目標勾配  $\{g_i\}$  になるべく合うような関数  $f$  を求める





## 1D の場合

Find  $\{f_i\}$  that minimize

$$\sum_i (f_i - f_{i-1} - g_i)^2$$

subject to:  $f|_{\partial\Omega} = f^*|_{\partial\Omega}$

## 2D の場合

Find  $f(x, y)$  that minimizes

$$\int_{(x,y) \in \Omega} \|\nabla f(x, y) - \mathbf{g}(x, y)\|^2$$

subject to:  $f|_{\partial\Omega} = f^*|_{\partial\Omega}$



- Gradient-domain 画像処理の基本：

ユーザが好き勝手に与えた目標勾配  
ベクトル場  $\mathbf{g}$  になるべく合うような  
画像  $f$  を、Poisson 方程式を解いて求める

Solve **Poisson equation**:

$$\Delta f = \nabla \cdot \mathbf{g}$$

subject to:  $f|_{\partial\Omega} = f^*|_{\partial\Omega}$

# Target gradient の与え方：Mixing Gradients

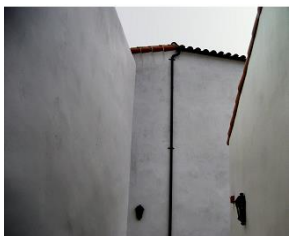
- Source 勾配と Dest. 勾配のうち大きい方を使う  
➔ 平坦な部分は clone されない



# Target gradient の与え方：Edge Brush

- 物体輪郭に沿った勾配をコピーし、ストロークに沿って貼り付け
- GPU 実装の Poisson solver によってリアルタイム動作

Before



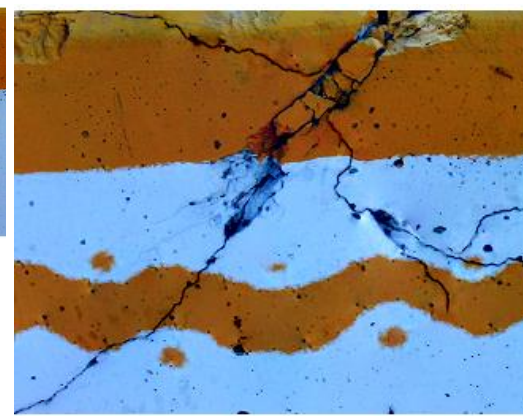
After



Before



After



Edge Brush

Source Picture:  
"stucco\_hall2"  
by clayjar  
via flickr



# Target gradient の与え方：元の gradient を操作



選択範囲内でのみ増幅・減衰  
→ Local Tone Mapping



エッジ検出された場所以外ではゼロにする  
→ Stylization

# おまけ：Gradient-domain の形状処理



# Gradient-domain 形状処理

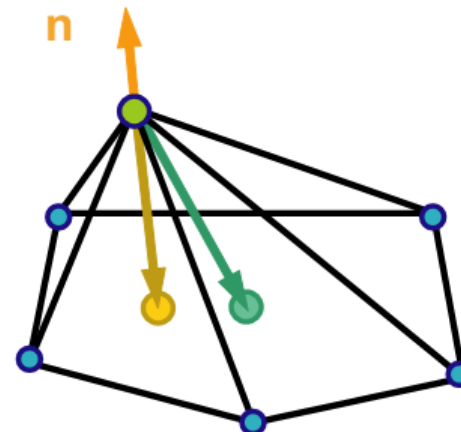
Find  $\{ \mathbf{v}_i \}$  that minimize

$$\sum_{(i,j) \in E} w_{ij} \| \mathbf{v}_i - \mathbf{v}_j - \overline{\mathbf{e}}_{ij} \|^2$$

subject to:  $\mathbf{v}_c = \mathbf{v}_c^*, c \in I_C$

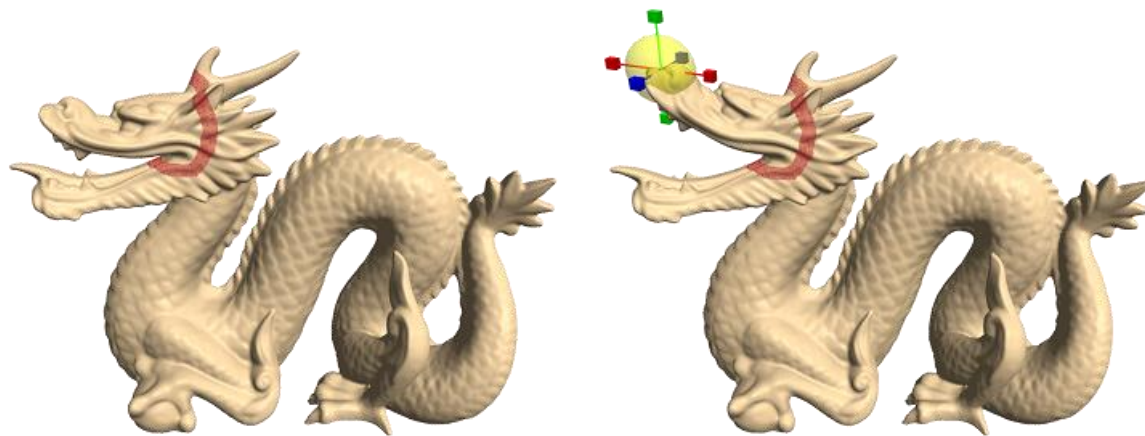
元形状の辺ベクトル  
→ 目標勾配

いくつかの頂点の位置制約  
→ 境界条件



Poisson 方程式

$$\mathbf{L} \mathbf{x} = \delta$$



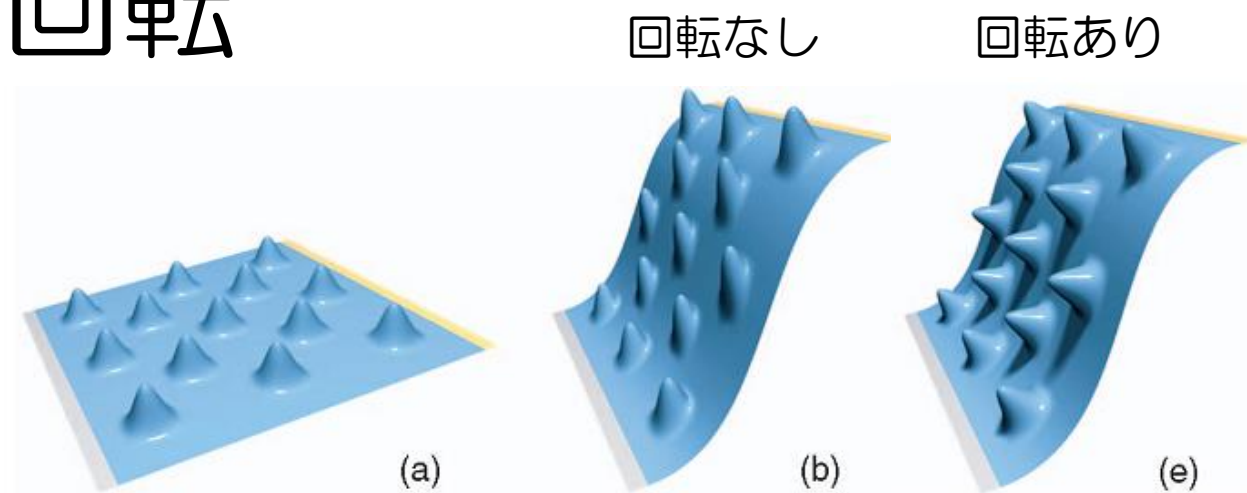
Mesh editing with poisson-based gradient field manipulation [Yu SIGGRAPH04]

Laplacian surface editing [Sorkine SGP04]

Interfaces and algorithms for the creation, modification, and optimization of surface meshes [Nealen PhD07]

# 変形に伴う局所領域の回転

- 目標勾配も合わせて回転させないといけない
  - 非線形で難しい！

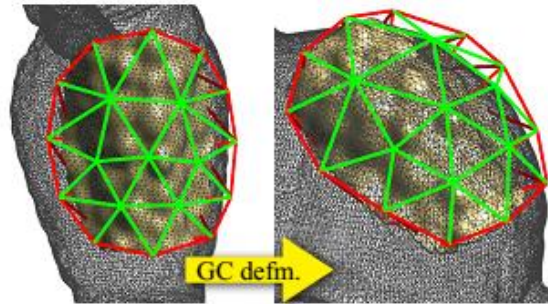


- Local-global 最適化アルゴリズム [Sorkine07]
  - Local step: 頂点座標を固定し、SVD で局所領域の回転を計算
  - Global step: 局所領域の回転を固定し、Poisson 方程式を解いて頂点座標を更新

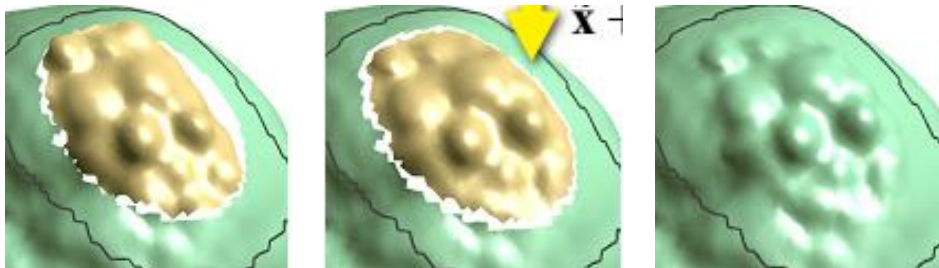


# GeoBrush: サーフエスメッシュのためのクローンブラシ

- 変形計算を 2 ステップに分解：
  - 局所領域の回転  
→ cage-based な方法で高速に計算



- 正確なオフセット  
→ 画像合成用の GPU Poisson ソルバ を流用



## GeoBrush: Interactive Mesh Geometry Cloning

Kenshi Takayama<sup>1,2</sup>  
Ryan Schmidt<sup>3</sup>  
Karan Singh<sup>3</sup>  
Takeo Igarashi<sup>1</sup>  
Tamy Boubekeur<sup>4</sup>  
Olga Sorkine<sup>2</sup>

<sup>1</sup>The University of Tokyo

<sup>2</sup>New York University

<sup>3</sup>University of Toronto

<sup>4</sup>Telecom ParisTech - CNRS LTCI

[https://www.youtube.com/watch?v=FPscn\\_gG8E](https://www.youtube.com/watch?v=FPscn_gG8E)