

# デジタルメディア処理1

担当: 井尻 敬

1

## スケジュール

- 09/26 インTRODクシヨN1 : デジタル画像とは, 量子化と標本化, Dynamic Range
- 10/03 インTRODクシヨN2 : デジタルカメラ, 人間の視覚, 表色系
- 10/10 フィルタ処理1 : トーンカーブ, 線形フィルタ
- 10/17 フィルタ処理2 : 非線形フィルタ, ハーフトーン
- 10/24 フィルタ処理3 : 離散フーリエ変換と周波数フィルタリング
- 11/07 前半のまとめと中間試験
- 11/14 画像処理演習 : python入門 (演習室)
- 11/21 画像処理演習 : フィルタ処理 (演習室)
- 11/28 画像処理演習 : フィルタ処理 (演習室)
- 12/05 画像処理演習 : フィルタ処理 (演習室)
- 12/12 画像の幾何変換 1 : アファイン変換
- 12/19 画像の幾何変換 2 : 画像の補間
- 01/16 画像復元 : ConvolutionとDe-convolution (変更する可能性有り)
- 01/23 後半のまとめと期末試験

## 画像の幾何変換 2

### 達成目標

- 画像の幾何学変換を計算でき, その効果を説明できる.
- 画像変形の際に必要な"補間法"について, その計算法と効果を説明できる

### Contents

- 画像の変換
- 画像補間
- イメージモザイク (パノラマ合成)

3

## 復習

- 行列演算
- 同次形式
- アファイン変換

※先週の内容は、画像処理/CG/VRなどの分野で必須です。もし怪しい部分があったら時間をかけて復習してください

4



## 画像の変換

5

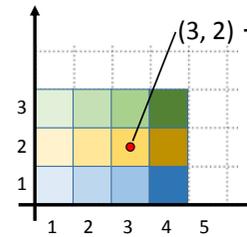
## 画像の変換

X軸方向に1.7倍, Y軸方向に2倍に拡大する変換を考える

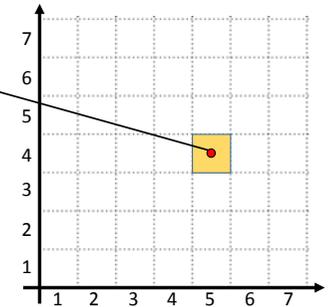
画素の幅を1とする

各画素を変換先に移動してみると…

4 x 3 画素の小さな画像



$(3, 2) \rightarrow (5.1, 4) \rightarrow (5, 4)$   
少数は四捨五入



画素の中心が  
整数座標になると仮定

6

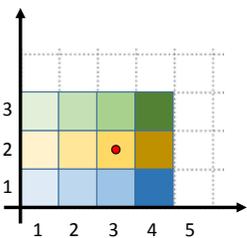
## 画像の変換

X軸方向に1.7倍, Y軸方向に2倍に拡大する変換を考える

画素の幅を1とする

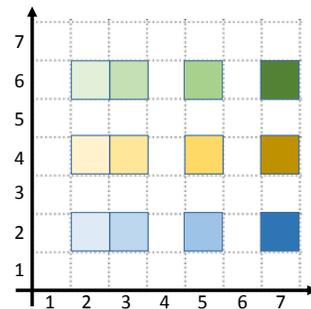
各画素を変換先に移動してみると…

4 x 3 画素の小さな画像



画素の中心が  
整数座標になると仮定

$(1, 1) \rightarrow (2, 2)$   
 $(1, 2) \rightarrow (2, 4)$   
 $(1, 3) \rightarrow (2, 6)$   
 $(2, 1) \rightarrow (3, 2)$   
 $(2, 2) \rightarrow (3, 4)$   
 $\vdots$   
 $\vdots$   
 $(4, 1) \rightarrow (7, 2)$   
 $(4, 2) \rightarrow (7, 4)$   
 $(4, 3) \rightarrow (7, 6)$



7

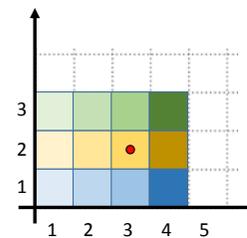
## 画像の変換

各画素を変換先に移動すると、飛び飛びの画像ができてしまう (拡大時)

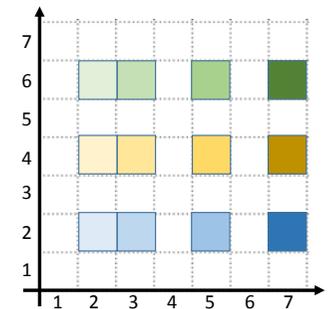
ほしかったのはもっと密な画像…

**そこで、通常は逆変換を考えます！**

4 x 3 画素の小さな画像



画素の中心が  
整数座標になると仮定



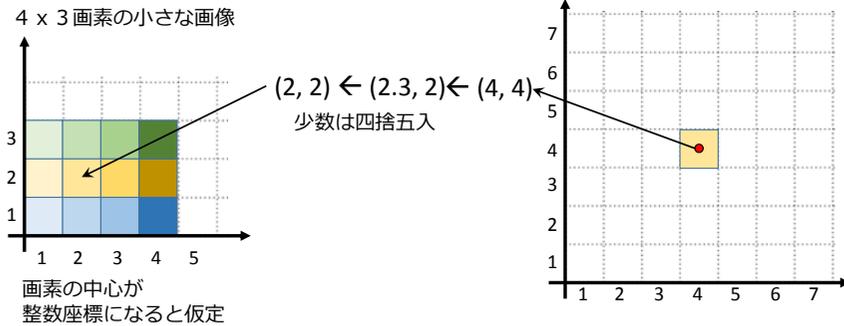
8

## 画像の変換

所望の変換は、X軸方向に1.7倍、Y軸方向に2倍

この逆変換は、X軸方向に1/1.7倍、Y軸方向に1/2倍

変換後画像の各画素に逆変換を施し、元画像における画素位置を取得する



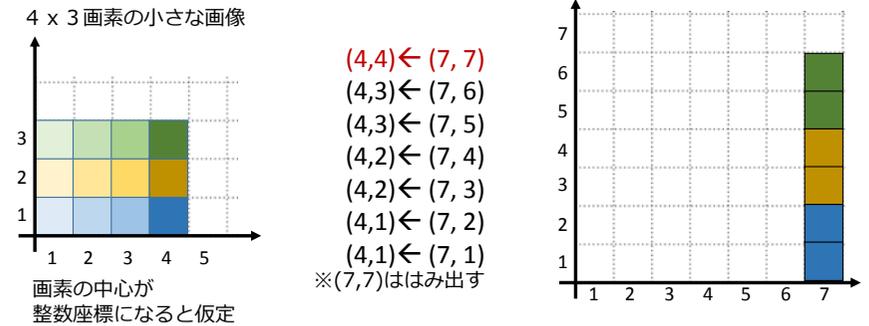
9

## 画像の変換

所望の変換は、X軸方向に1.7倍、Y軸方向に2倍

この逆変換は、X軸方向に1/1.7倍、Y軸方向に1/2倍

変換後画像の各画素に逆変換を施し、元画像における画素位置を取得する



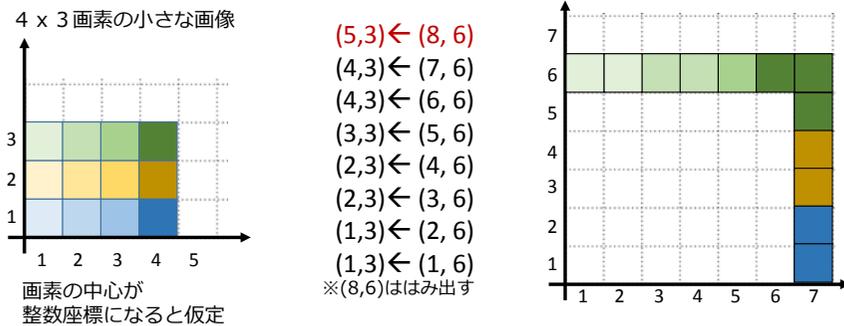
10

## 画像の変換

所望の変換は、X軸方向に1.7倍、Y軸方向に2倍

この逆変換は、X軸方向に1/1.7倍、Y軸方向に1/2倍

変換後画像の各画素に逆変換を施し、元画像における画素位置を取得する



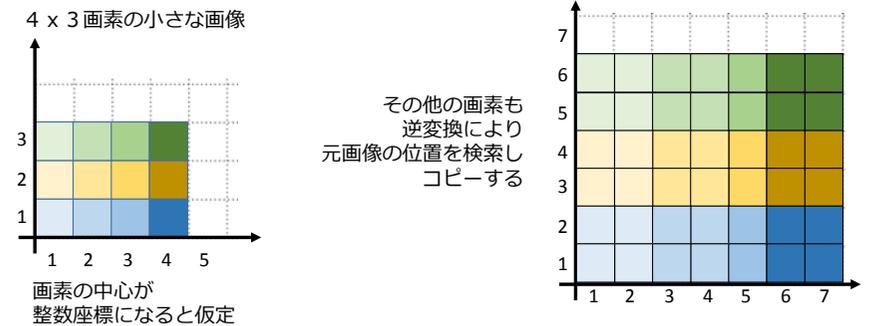
11

## 画像の変換

所望の変換は、X軸方向に1.7倍、Y軸方向に2倍

この逆変換は、X軸方向に1/1.7倍、Y軸方向に1/2倍

変換後画像の各画素に逆変換を施し、元画像における画素位置を取得する



12

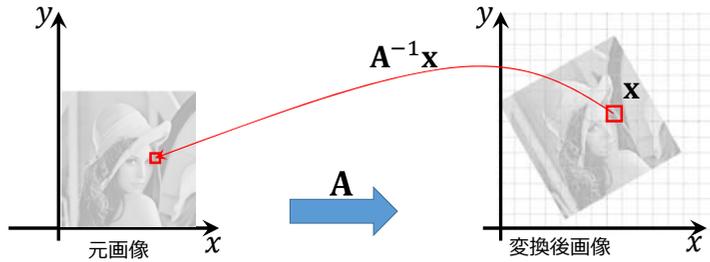
## 画像の変換

任意の変換について

誤) 変換元画像の各画素の行き先を計算する

正) 変換先の各画素に逆変換を掛け、元画像を参照する

※ X軸方向に0倍のような変換をすると逆変換が存在しないので注意

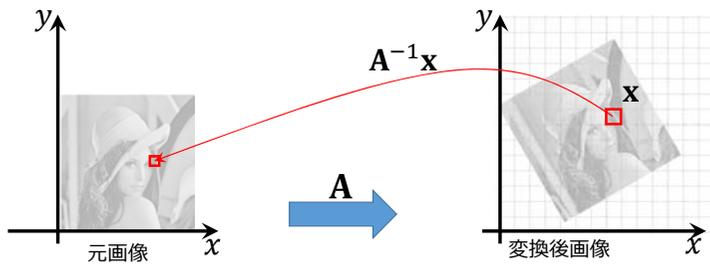


13

## 画像の補間

14

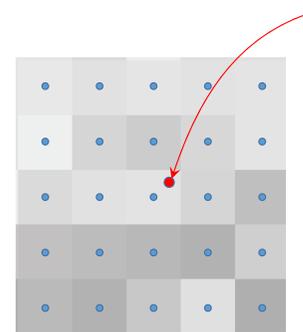
なぜ画像補間が必要か？



- 画像変換時には、逆変換を計算し元画像の画素を参照する
- 参照先を拡大してみると。。。。

15

なぜ画像補間が必要か？



- 赤点: サンプリングしたい場所
- 青点: 画素値が存在する場所

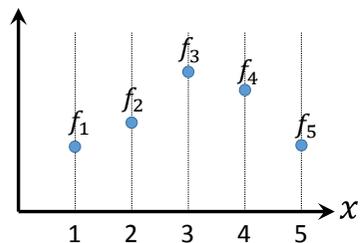
赤点の場所の画素値は？

- 一番近い画素値を使う??
- 近傍画素を混ぜる??

→ 補間する

16

## 補間法 (1D)



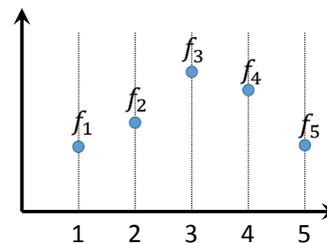
入力：画素値  $f_i$   
 $x$ が整数の位置のみに値が存在

出力： $f_i$ を補間した関数 $g(x)$

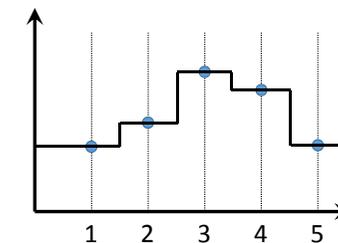
※修正：連続関数 $g(x)$  → 関数 $g(x)$

17

## 補間法 (1D) : Nearest Neighbor



入力：画素値  $f_i$



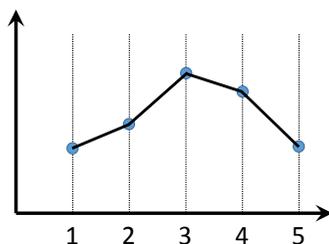
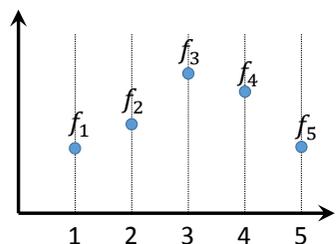
最近傍画素の値を使う

$$g(x) = f_{\lfloor x+0.5 \rfloor}$$

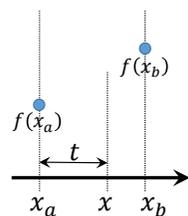
※ $\lfloor t \rfloor$ はガウス記号:  $t$ を超えない最大の整数

18

## 補間法 (1D) : Linear Interpolation



入力：画素値  $f_i$



前後2画素を線形に補間する

$$g(x) = (1-t)f_{x_a} + tf_{x_b}$$

$$x_a = \lfloor x \rfloor,$$

$$x_b = \lfloor x \rfloor + 1,$$

$$t = x - \lfloor x \rfloor$$

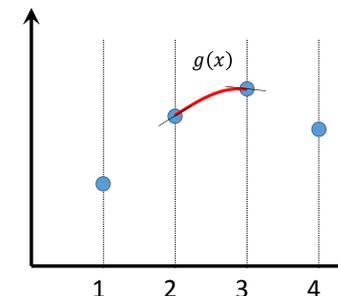
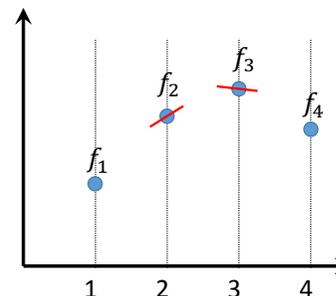
19

## 補間法 (1D) : Hermite Cubic Spline Interpolation

区間 $[2,3]$ を補間するとき

※赤字は講義後修正したもの

- $f_2, f_3$ における勾配も制約する
- 勾配制約を計算するため $f_1, f_2, f_3, f_4$ を利用する



20

## 補間法 (1D) : Hermite Cubic Spline Interpolation

※赤字は講義後修正したもの

$$g(x) \text{ は3次の関数であるとする,} \\ g(x) = ax^3 + bx^2 + cx + d \quad \text{for } x \in [0,1] \quad \dots (1)$$

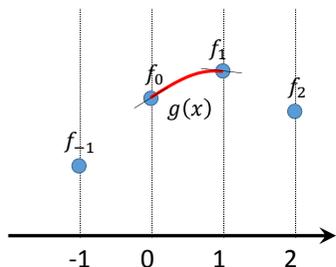
入力 : 画素値  $f_{-1}, f_0, f_1, f_2$   
下図の区間  $[0,1]$  の補間を考える

$$\text{境界において画素値を満たすため,} \\ g(0) = f_0, \quad g(1) = f_1 \quad \dots (2)$$

$$\text{境界における勾配を4点を用いて指定} \\ g'(0) = \frac{1}{2}(f_1 - f_{-1}), \quad g'(1) = \frac{1}{2}(f_2 - f_0) \quad \dots (3)$$

式(1)(2)(3)より  $g(x)$  が求まる

$$g(x) = \frac{1}{2} \begin{pmatrix} 1 & x & x^2 & x^3 \end{pmatrix} \begin{pmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{pmatrix} \begin{pmatrix} f_{-1} \\ f_0 \\ f_1 \\ f_2 \end{pmatrix}$$



21

## 補間法 (1D) : Cubic Convolution Interpolation [1]

※赤字は講義後修正したもの

教科書で紹介されているのはこれ  
下図の区間  $[0,1]$  の補間を考える  
 $x = -1, 0, 1, 2$  の画素値を  $f_{-1}, f_0, f_1, f_2$  とする

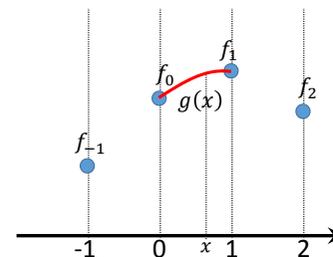
$$g(x) \text{ を4つの画素値の重み付け和で表現する} \\ g(s) = h(t_{-1})f_{-1} + h(t_0)f_0 + h(t_1)f_1 + h(t_2)f_2$$

$$\text{ただし, } t_i \text{ は, } x \text{ から画素までの距離} \\ t_{-1} = x + 1, \quad t_0 = x, \quad t_1 = 1 - x, \quad t_2 = 2 - x$$

重み関数は以下の通り定義される[1]

$$h(t) = \begin{cases} (a+2)|t|^3 - (a+3)|t|^2 + 1 & \text{if } |t| \leq 1 \\ a|t|^3 - 5a|t|^2 + 8a|t| - 4a & \text{if } 1 \leq |t| \leq 2 \\ 0 & \text{otherwise} \end{cases}$$

$a$  はユーザが決める変数,  $a = -0.5$  とするとよい[1]

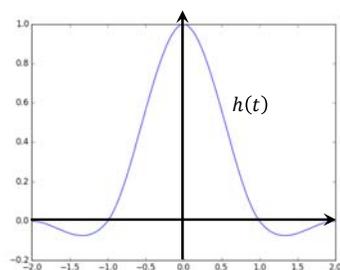


[1] R. Keys, Cubic convolution interpolation for digital image processing, IEEE TASSP 1981.

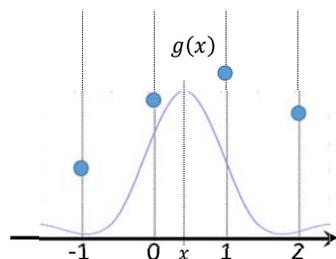
22

## 補間法 (1D) : Cubic Convolution Interpolation [1]

$$h(t) = \begin{cases} (a+2)|t|^3 - (a+3)|t|^2 + 1 & \text{if } |t| \leq 1 \\ a|t|^3 - 5a|t|^2 + 8a|t| - 4a & \text{if } 1 \leq |t| \leq 2 \\ 0 & \text{otherwise} \end{cases}$$



$$h(0) = 1 \\ h(n) = 0 \quad n \text{ は } 0 \text{ でない整数}$$



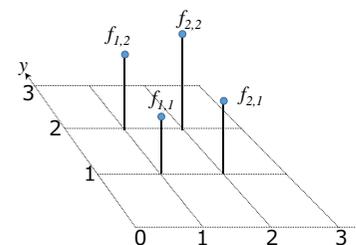
$h(x)$  を求めたい位置  $x$  に重ね  
周囲4画素の重みを決定する

[1] R. Keys, Cubic convolution interpolation for digital image processing, IEEE TASSP 1981.

23

## 補間法 (2D)

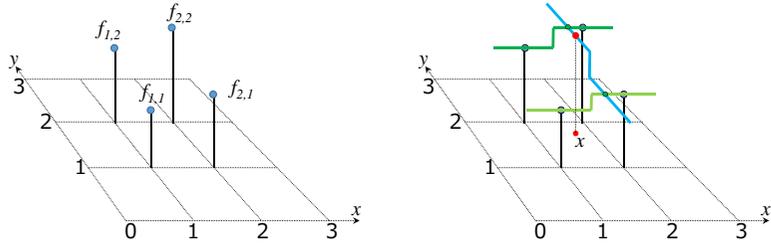
解説した各手法を2次元に拡張する  
 $x$  軸方向に補間し,  $y$  軸方向に補完する  
2次元補間は、bi-\* という名前になる



この図では、破線の交差部分に  
画素中心があるとする

24

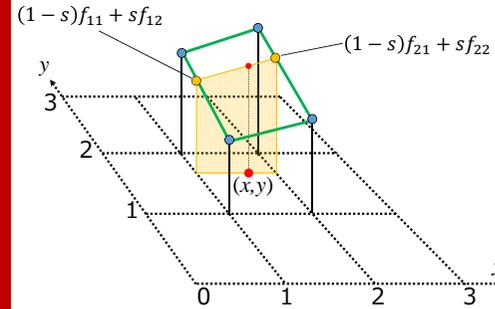
## 補間法 (2D) : Nearest neighbor



最近傍画素値を利用する  
 $g(x) = f_{\lfloor x+0.5 \rfloor, \lfloor y+0.5 \rfloor}$

25

## 補間法 (2D) : Linear Interpolation



$x \in [1,2], y \in [1,2]$ の範囲を  
 画素 $f_{11}, f_{12}, f_{21}, f_{22}$ より補間する

$$g(x, y) = (1-t) \begin{pmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{pmatrix} \begin{pmatrix} 1-s \\ s \end{pmatrix}$$

$$t = x - 1, s = y - 1$$

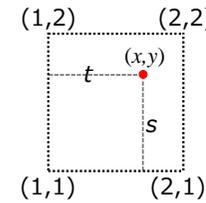
上式はなにをしてるのか?

1. まず $x=1, x=2$ において $y$ 軸方向に線形補間し2点を取得 (黄点)

$$(1-s)f_{11} + sf_{12}, (1-s)f_{21} + sf_{22}$$

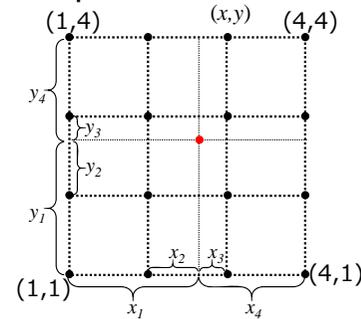
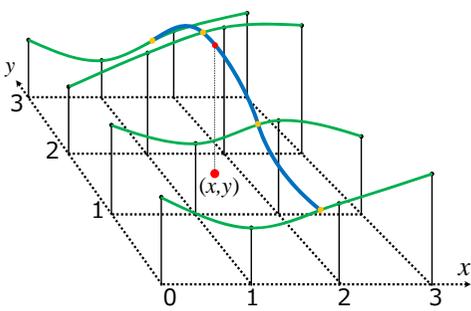
2. 得られた2点を $x$ 軸方向に線形補間 (赤点)

$$(1-t)((1-s)f_{11} + sf_{12}) + t((1-s)f_{21} + sf_{22})$$



26

## 補間法 (2D) : Bicubic Convolution Interpolation



$x \in [1,2], y \in [1,2]$ の範囲を近傍16画素 $f_{xy}$ より補間する

$$g(x, y) = (h(x_1) h(x_2) h(x_3) h(x_4)) \begin{pmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ f_{31} & f_{32} & f_{33} & f_{34} \\ f_{41} & f_{42} & f_{43} & f_{44} \end{pmatrix} \begin{pmatrix} h(y_1) \\ h(y_2) \\ h(y_3) \\ h(y_4) \end{pmatrix}$$

$h(t)$ は1次元補間と同様,  $x_i, y_i$ は右上図の通り定義される.

左上図の通り

1. まず $x$ 軸に沿ってcubic補間
2. 得られた4点を利用し $y$ 軸に沿ってcubic補間

27

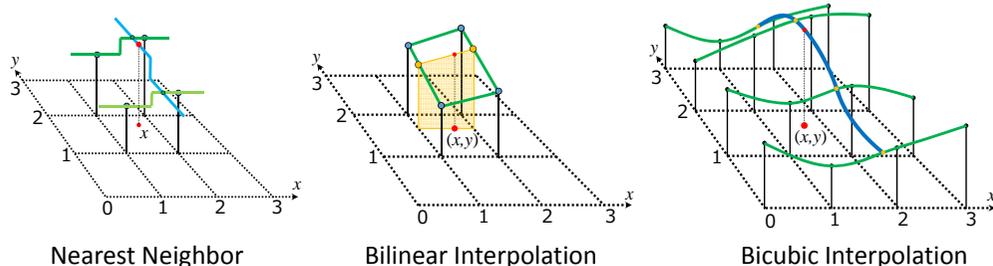
## 画像の補間法 : 例

教科書p171の図参照

28

## まとめ: 画像の補間法

- 画像の変換（特に拡大）の際、画像の画素と画素の間を参照する
- 周囲の画素を利用し、参照位置の画素値を決定する



- 様々なソフトウェアがこの変換(Bicubicが多い)を自動でかけてくれる

- **研究目的のデータ処理においては注意が必要** → デモ VoTraver volume rendering

29

## イメージモザイク (パノラマ合成)

30

## イメージモザイク

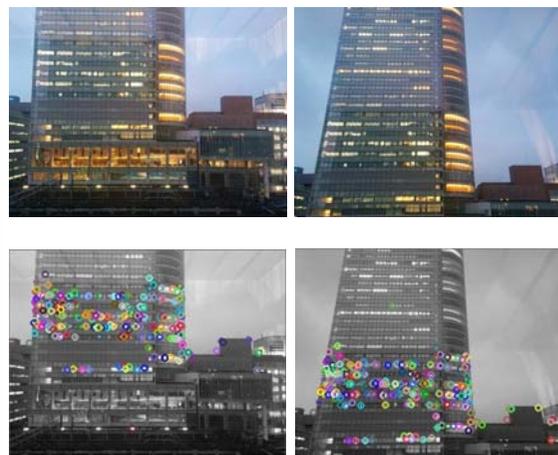
panorama.py

- ここまで紹介してきた画像変換の応用のひとつ
- 複数の画像を変形し重ね合わせて大きな画像を作成する技術



31

## パノラマ合成1: 入力画像について特頂点を検出する



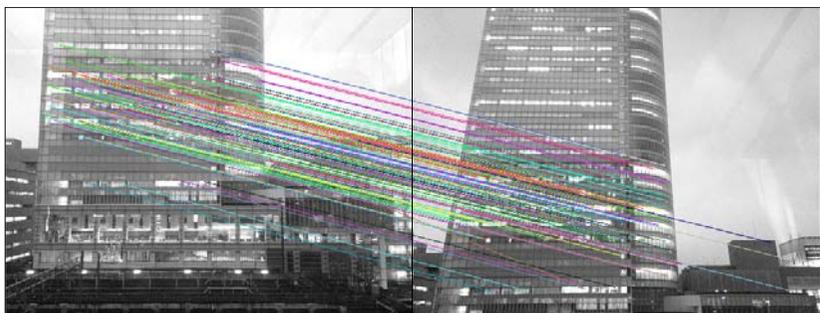
- **特徴点**: 角やエッジなど、顕著な局所的変化がある場所
- 特徴点検出アルゴリズムはSIFT, SURF, Eigen, Harrisなどが有名
- 特徴点は、その周囲の様子を記述する特徴ベクトルを持つ

※特徴点については後で詳しく解説

※左図はAKAZE algorithmを利用した結果

32

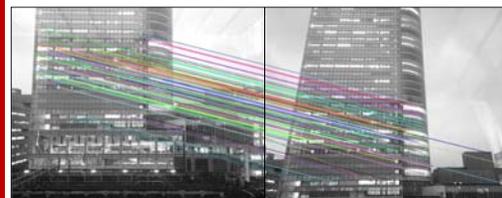
## パノラマ合成2. 特徴点の対応付け



- 各特徴点は局所領域の特徴を記述する特徴ベクトルを持つ
- 特徴ベクトルの類似性を利用して対応を計算する
- 上図ではBrute force algorithmを利用
  - 左画像の特徴点をひとつピックアップし、最も似た特徴点を右画像内から全検索

33

## パノラマ合成3. 変換行列の計算



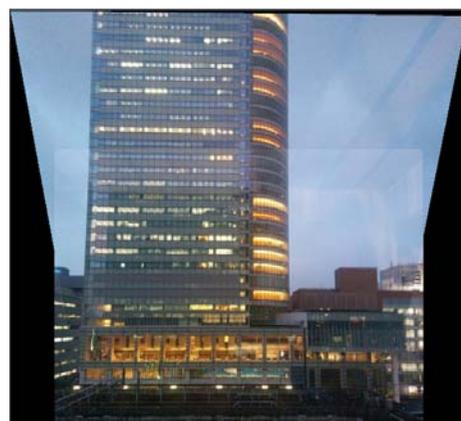
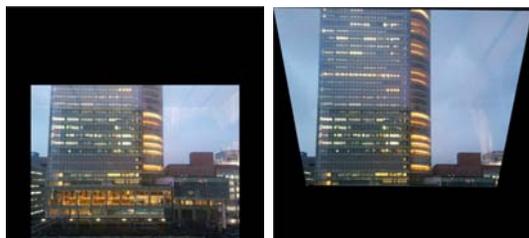
- 対応特徴点の位置が重なるよう右画像を射影変換
- つまり、対応点をなるべく一致させる行列

$$\mathbf{H} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{pmatrix} \text{を求めたい}$$

- RANSAC (Random Sample Consensus)
  - 未知変数推定に必要なデータを乱択する (未知変数が8個なので 8個の特徴点の組)
  - 選択した特徴点の組を用いて変換 $\mathbf{H}$ を導出
  - 変換 $\mathbf{H}$ によりほか全て特徴点を変換する  
特徴点に対応点の十分近くに变换された → Inlier  
特徴点の变换先が対応点から遠い → Outlier
  - 1~3を繰り返しInlier数が最大の $\mathbf{H}$ を出力

34

## パノラマ合成4. 画像の合成



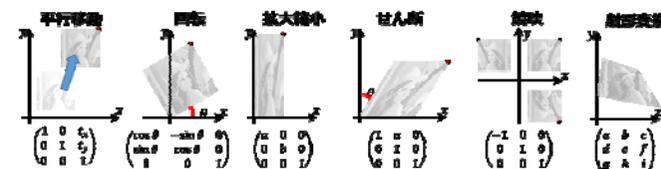
- 上図は単純な実装: 2画像が重なる部分は両者の平均を取る → シームが目立つ
- 目立たないシームを計算する手法 → [GraphCutTextures, SIGGRAPH 2003]
- 画像ピラミッドを利用する手法 → [A Multiresolution Spline With Application to Image Mosaics, TOG1983]

35

## まとめ: 画像の変換

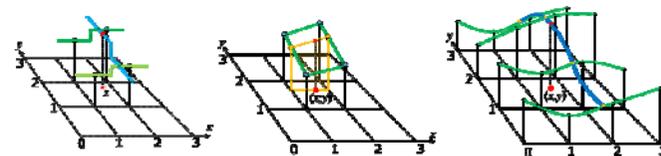
### 幾何学変換を紹介した

- Affine 変換
- 射影変換
- 同次座標系



### 補間法を紹介した

- Nearest Neighbor
- Bilinear Interpolation
- Bicubic Interpolation



### パノラマ合成を紹介した



36