

デジタルメディア処理

担当: 井尻 敬

表記について

- スカラー変数はイタリック体 : a, b, c
- ベクトルは小文字ボールド体 : $\mathbf{a}, \mathbf{b}, \mathbf{c}$
- 行列は大文字ボールド体 : $\mathbf{A}, \mathbf{B}, \mathbf{C}$
- \mathcal{R} を利用して次元を明確に : $\mathbf{a} \in \mathcal{R}^3, \mathbf{A} \in \mathcal{R}^{3 \times 3}$
- 右肩 T は転置を表す : $\mathbf{a} = \begin{pmatrix} x \\ y \end{pmatrix}, \mathbf{a}^T = (x \ y)$
 $\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \mathbf{A}^T = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$

CG分野の論文ではよく見かける表記ですが、よく利用される表記は分野に依存するので注意してください。

Contents

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。
- 剛体変換・相似変換・アファイン変換・射影変換を正しく計算でき、それぞれの効果を説明できる

Contents

- 行列とベクトルの復習
- 線形変換（拡大・縮小 / 回転 / 鏡映 / せん断 / 合成）
- アファイン変換（平行移動 / 同次座標系）

線形代数の復習(1)

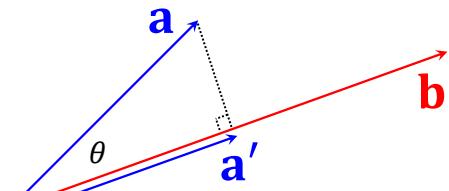
$$\mathbf{a} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \mathbf{b} = \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix} \text{ のとき以下の計算をせよ}$$

(1) $\mathbf{a} \cdot \mathbf{b}$

(2) $\mathbf{a} \times \mathbf{b}$

(3) $\|\mathbf{a}\|$

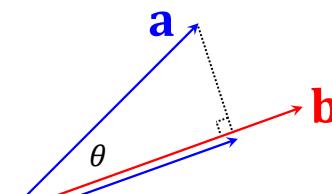
内積の意味



$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \theta$$

ベクトル \mathbf{a} をベクトル \mathbf{b} に射影し
両者の長さを掛け合わせたもの

$|\mathbf{b}| = 1$ のとき

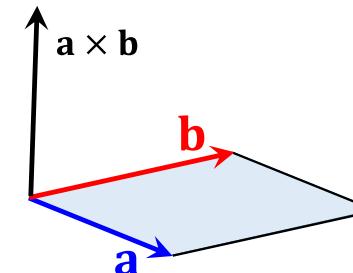


$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| \cos \theta$$

ベクトル \mathbf{a} をベクトル \mathbf{b} に射影した長さ

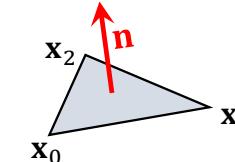
外積の意味

$$\mathbf{a} \times \mathbf{b} = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \times \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = \begin{pmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{pmatrix}$$



二つのベクトル \mathbf{a}, \mathbf{b} に直交するベクトルを返す
長さはベクトル \mathbf{a}, \mathbf{b} が作る平行四辺形の面積

よくある応用：
三角形ポリゴンの法線・面積計算



$$\text{法線: } \mathbf{n} = \frac{(\mathbf{x}_2 - \mathbf{x}_0) \times (\mathbf{x}_1 - \mathbf{x}_0)}{|(\mathbf{x}_2 - \mathbf{x}_0) \times (\mathbf{x}_1 - \mathbf{x}_0)|}$$

$$\text{面積: } S = \frac{1}{2} |(\mathbf{x}_2 - \mathbf{x}_0) \times (\mathbf{x}_1 - \mathbf{x}_0)|$$

線形代数の復習(2)

$\mathbf{a} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$, $\mathbf{A} = \begin{pmatrix} 1 & 0 & 3 \\ 4 & 5 & 0 \\ 0 & 2 & 1 \end{pmatrix}$, $\mathbf{B} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 2 \\ 3 & 2 & 1 \end{pmatrix}$ のとき以下の計算をせよ

(1) $\mathbf{A}\mathbf{a}$

(2) $\mathbf{a}^T \mathbf{A}$

(3) \mathbf{AB}

(4) \mathbf{A} の行列式 $|\mathbf{A}|$ を求めよ

線形代数の復習(3)

$\mathbf{A} = \begin{pmatrix} 2 & -1 \\ \frac{3}{2} & -\frac{1}{2} \end{pmatrix}$, $\mathbf{B} = \begin{pmatrix} 3 & 0 & -2 \\ 0 & 3 & 0 \\ 1 & 0 & 0 \end{pmatrix}$ の固有値と固有ベクトルを求めよ

線形代数の復習(4)

$A = \begin{pmatrix} 2 & -1 \\ \frac{3}{2} & -\frac{1}{2} \end{pmatrix}, B = \begin{pmatrix} 3 & 0 & -2 \\ 0 & 3 & 0 \\ 1 & 0 & 0 \end{pmatrix}$ を対角化せよ

線形代数の復習(5)

- 行列の対角化は、様々なところで利用する大切な概念
 - べき乗の高速計算 A^{30}
 - 極分解の \sqrt{A}
- 行列はいつも対角化できるわけではない
 - 『 $A \in R^{n \times n}$ が n 本の線形独立な固有ベクトルを持つとき、 A は対角化可能』 : A の持つ n 個の固有値がすべて異なれば、 n 本の線形独立な固有ベクトルが存在するので対角化可能。
 - 固有値が重複する（固有多項式が重解を持つ）場合に、対角化できないことがある。

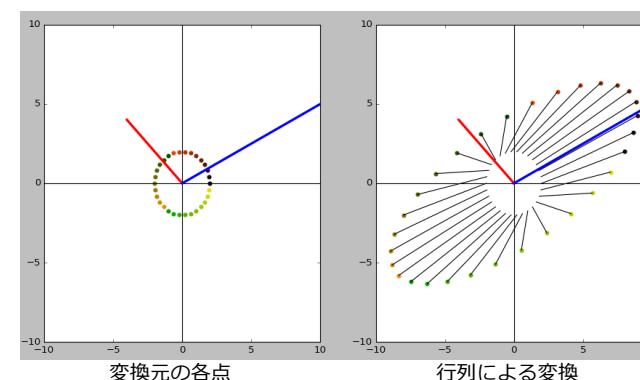
例) 行列 $\begin{pmatrix} 3 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 2 \end{pmatrix}$ を対角化せよ

固有値・固有ベクトルの意味

行列 $\begin{pmatrix} a & b \\ c & d \end{pmatrix} \in R^{2 \times 2}$ について

固有値・固有ベクトルの意味

EigenVector.py



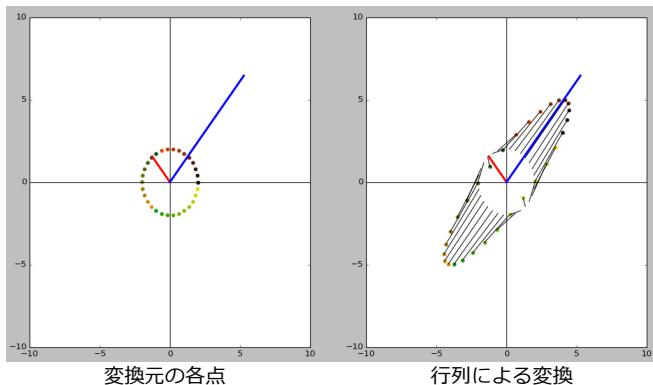
$$\begin{pmatrix} 4 & 2 \\ 1 & 3 \end{pmatrix}$$

固有値・固有ベクトル
 $2, \begin{pmatrix} -1 \\ 1 \end{pmatrix}, 5, \begin{pmatrix} 2 \\ 1 \end{pmatrix}$

赤線・青線は固有ベクトル
黒線は変換による移動を示す

円周上の点群を変換すると楕円上に乗る
楕円の主軸と固有ベクトルは一致しない（一致するのは特殊な場合）
固有ベクトル上の点は、変換後も固有ベクトル上に乗る

固有値・固有ベクトルの意味



$$\begin{pmatrix} 2 & 1 \\ 1.5 & 2 \end{pmatrix}$$

固有値・固有ベクトル

$$\frac{-\sqrt{6}+4}{2}, \left(-\frac{\sqrt{6}}{3} \right), \frac{\sqrt{6}+4}{2}, \left(\frac{\sqrt{6}}{3} \right)$$

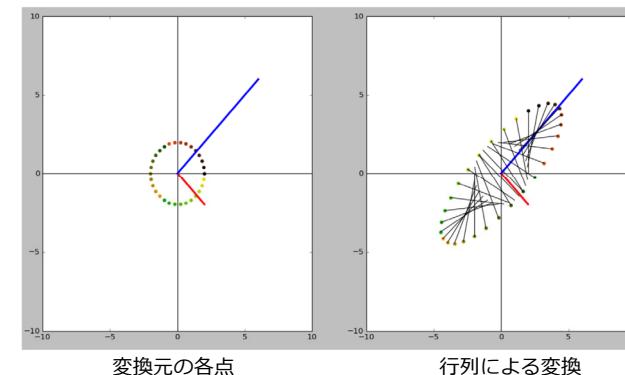
赤線・青線は固有ベクトル
黒線は変換による移動を示す

円周上の点群を変換すると楕円上に乗る

楕円の主軸と固有ベクトルは一致しない（一致するのは特殊な場合）

固有ベクトル上の点は、変換後も固有ベクトル上に乗る

固有値・固有ベクトルの意味



$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$$

固有値・固有ベクトル

$$-1, \begin{pmatrix} -1 \\ 1 \end{pmatrix}, 3, \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

赤線・青線は固有ベクトル
黒線は変換による移動を示す

固有ベクトル上の点は、変換後も固有ベクトル上に乗る

対称行列の固有ベクトルは互いに直行する

対称行列による変換では、楕円の主軸と固有ベクトルが一致

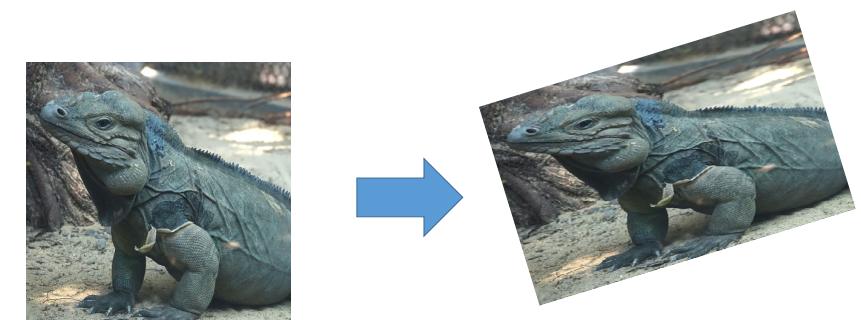
※ この例では固有値が負なので固有ベクトルに対して鏡面変換が起こっている

まとめ：ベクトルと行列の復習

- 画像処理（とCG）に頻出する行列・ベクトル演算の基礎を復習した
 - 行列とベクトルの積
 - 内積・外積
 - 逆行列
 - 固有値・固有ベクトル
 - 対角化

今日復習した内容は色々な分野で頻繁に出てくるので覚えてください

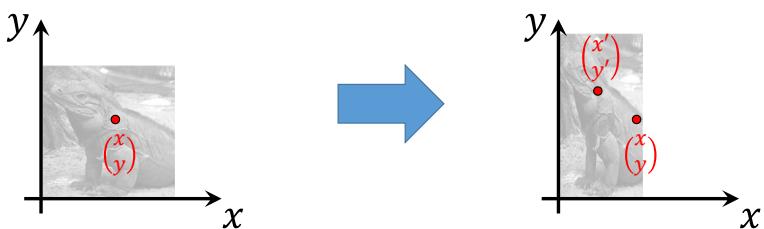
画像の線形変換



- 2x2行列は2次元空間中の任意の点を移動させる作用を持つ
→ 行列の積により画像の変形が表現できる（線形変換）

線形変換

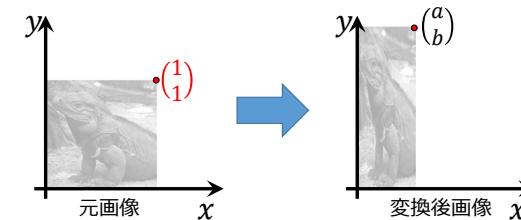
- 画像は2次元座標系に配置されているとする
 - 教科書に合わせて左下を原点とする
 - 環境(Windowsとか)によっては左上が原点のことも多い
- 空間内の全ての点 $\begin{pmatrix} x \\ y \end{pmatrix}$ に行列 $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ をかけ、 $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$ と変形する
 - 2次元空間(画像)が行列 $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ により歪められる
 - 線形変換は行列の形で、**拡大縮小・回転・鏡映・せん断**、に分類される
 - 変換の**合成**も行なえる



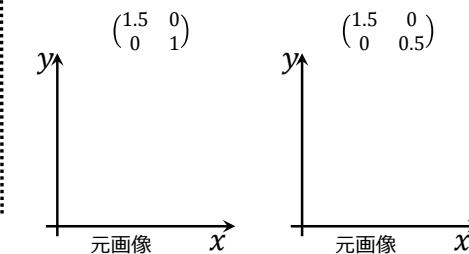
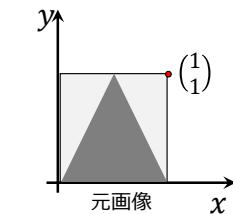
拡大縮小

X軸方向に a 倍、Y軸方向に b 倍する変換

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$



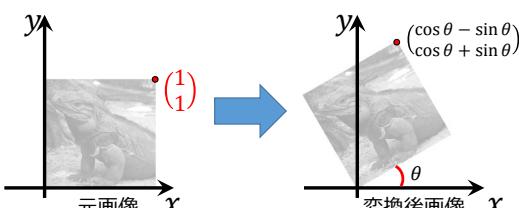
練) 変換結果を図示し
点(1,1)の移動後の座標を示せ



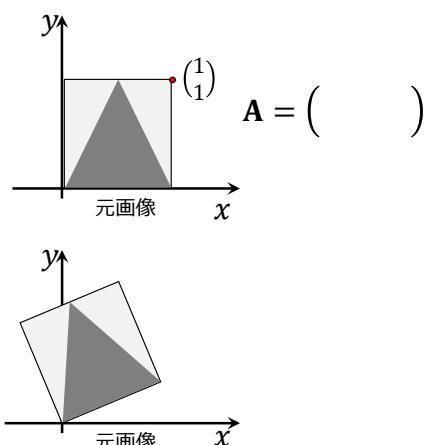
回転

原点を中心に角度 θ だけ回転する変換

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$



練) $\theta = \pi/6$ の回転行列Aを示せ
Aにより下画像の変換結果を図示せよ
また、点(1,1)の移動後の座標を示せ

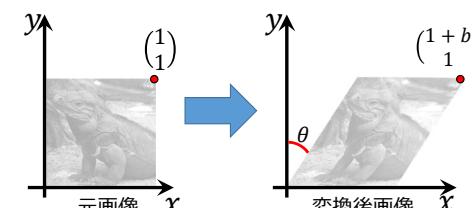


せん断 (スキュー)

X軸方向に角度 θ だけ歪める変換

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & b \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

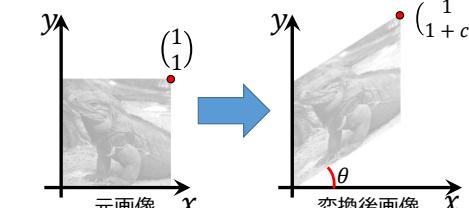
ただし $b = \tan \theta$



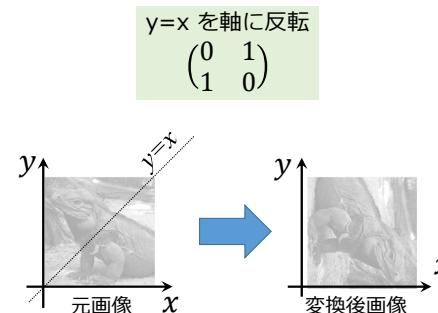
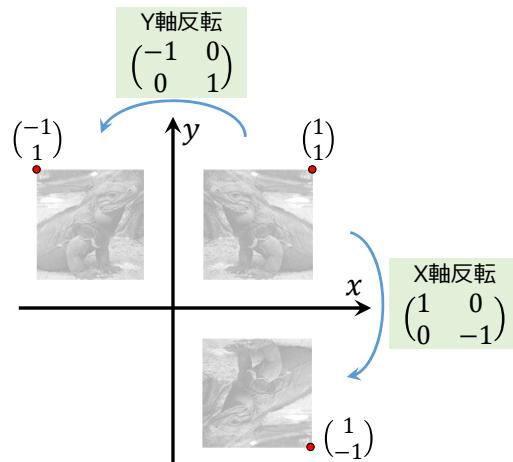
Y軸方向に角度 θ だけ歪める変換

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ c & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

ただし $c = \tan \theta$

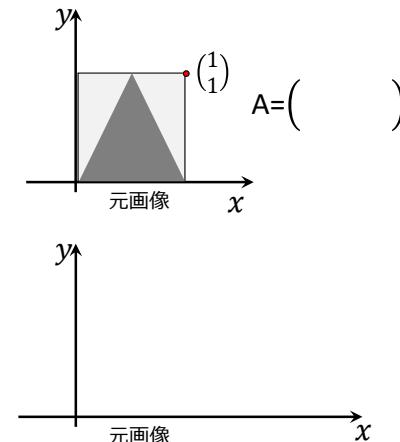


鏡映: 直線に対して反転する変換



- 練習
- $\theta = \pi/4$ の x 軸方向せん断変換 A を示せ
 - A による下画像の変換結果を図示せよ
 - A による点 $(1,1)$ の移動後の座標を示せ

- $\theta = \pi$ の回転変換行列を示せ
- Y 軸に対して鏡映変換し、さらに X 軸に対して鏡映変換する変換をひとつの行列で示せ

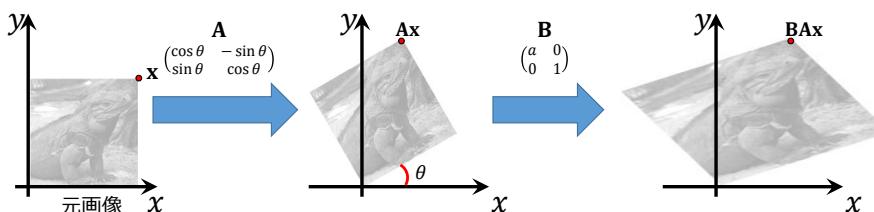


線形変換：合成

2つ以上の変換を続けて行う状況を考える

(例) θ 回転し、さらに x 軸方向に a 倍に拡大

→ 複数の連続した変換はひとつの線形変換で表現できる



この2ステップの変換は、 $C = BA = \begin{pmatrix} a \cos \theta & -a \sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$ というひとつの線形変換とみなせる

ちょっと蛇足ですが。。。

- 角度 $\theta + \phi$ 回転する回転行列は $\begin{pmatrix} \cos(\theta + \phi) & -\sin(\theta + \phi) \\ \sin(\theta + \phi) & \cos(\theta + \phi) \end{pmatrix}$ と定義される

- 一方 θ 回転してから ϕ 回転しても同じことなので、

$$\begin{pmatrix} \cos(\theta + \phi) & -\sin(\theta + \phi) \\ \sin(\theta + \phi) & \cos(\theta + \phi) \end{pmatrix} = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{pmatrix}$$

この右辺を整理すると

$$\begin{pmatrix} \cos(\theta + \phi) & -\sin(\theta + \phi) \\ \sin(\theta + \phi) & \cos(\theta + \phi) \end{pmatrix} = \begin{pmatrix} \cos \theta \cos \phi - \sin \theta \sin \phi & -\sin \theta \cos \phi - \cos \theta \sin \phi \\ \sin \theta \cos \phi + \cos \theta \sin \phi & \cos \theta \cos \phi - \sin \theta \sin \phi \end{pmatrix}$$

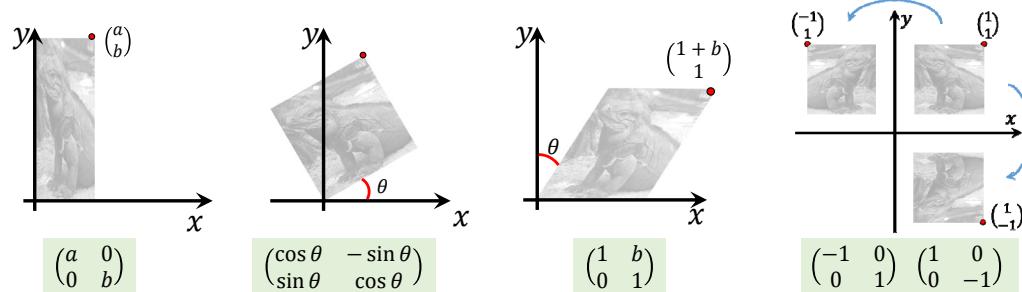
となり

$$\cos(\theta + \phi) = \cos \theta \cos \phi - \sin \theta \sin \phi$$

$$\sin(\theta + \phi) = \sin \theta \cos \phi + \cos \theta \sin \phi$$

が現れる（もう覚えなくていい）

画像の線形変換：まとめ



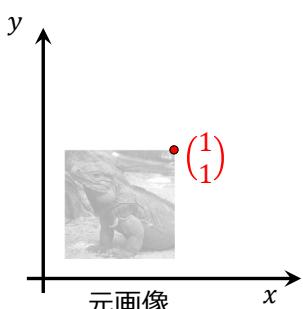
- 行列を2次元空間の任意の点にかけることで空間を変形できる
→ 画像の変形に利用できる
- 線形変換は、**拡大縮小・回転・鏡映・せん断**、という変換に分類される
- 変換の合成も行なえる

Affine変換と同次座標系

平行移動

X軸・Y軸方向に $a \cdot b$ だけ平行移動する変換

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} a \\ b \end{pmatrix}$$

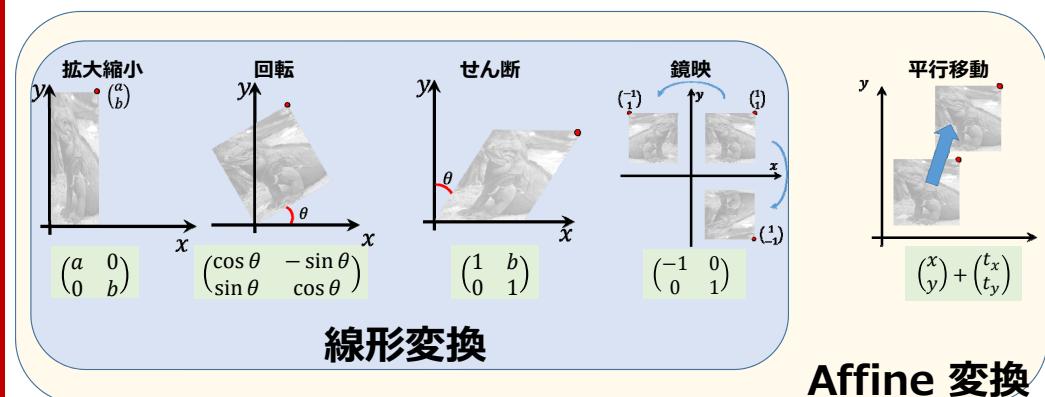


※（今一度確認）この変換は元画像のある二次元空間の任意の点 (x, y) を $(x+a, y+b)$ に変換したという意味

※これは行列の積ではないので
線形変換ではない

Affine変換

- 平行移動と線形変換の組み合わせで得られる変換のこと
- 英語発音は「アファイン」，アフィンと読む人も多い



同次座標系表現

2次元座標 $\begin{pmatrix} x \\ y \end{pmatrix}$ を 3次元ベクトル $\begin{pmatrix} wx \\ wy \\ w \end{pmatrix}$ と表記する方法

同じ 2 次元座標を表す同次座標を同値であると言い、式では『～』記号で表す

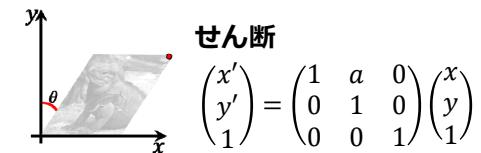
例) 2次元座標 $\begin{pmatrix} 2 \\ 5 \end{pmatrix}$ は、同次座標で $\begin{pmatrix} 2 \\ 5 \\ 1 \end{pmatrix}$ や $\begin{pmatrix} 4 \\ 10 \\ 2 \end{pmatrix}$ や $\begin{pmatrix} 8 \\ 20 \\ 4 \end{pmatrix}$ と表せる

例) 同次座標 $\begin{pmatrix} 2 \\ 5 \\ 1 \end{pmatrix}$ と $\begin{pmatrix} 8 \\ 20 \\ 4 \end{pmatrix}$ は同値である、 $\begin{pmatrix} 2 \\ 5 \\ 1 \end{pmatrix} \sim \begin{pmatrix} 8 \\ 20 \\ 4 \end{pmatrix}$

とりあえず $w = 1$ の場合を考える

2次元座標 $\begin{pmatrix} x \\ y \end{pmatrix}$ は、同次座標では $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ と表記できる

Affine変換の 同次座標表現



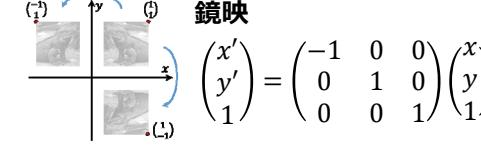
拡大縮小

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

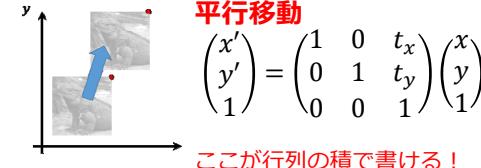
回転

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

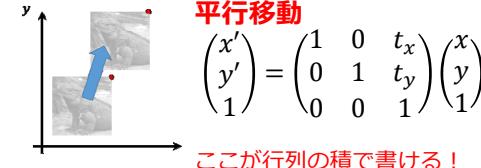
せん断



鏡映



平行移動



同時座標表現の利点

平行移動を行列の積で表せる

つまりアフィン変換を行列の積の形で表現できる

拡大縮小

$$\begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}$$

回転

$$\begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

せん断

$$\begin{pmatrix} 1 & b \\ 0 & 1 \end{pmatrix}$$

鏡映

$$\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$$

平行移動

$$\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

$$\begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

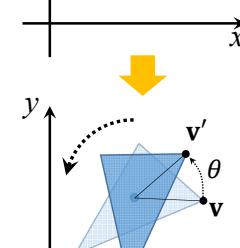
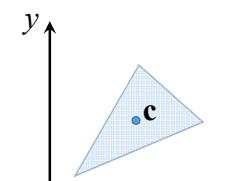
$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

同時座標表現の利点

例: 重心 (c_x, c_y) を中心で反時計回りに θ 回転

通常の2次元座標表現

$$\mathbf{v}' = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \left(\begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} c_x \\ c_y \end{pmatrix} \right) + \begin{pmatrix} c_x \\ c_y \end{pmatrix}$$



同次座標系表現

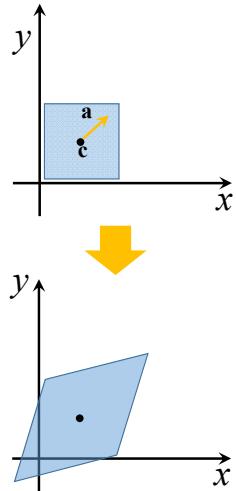
$$\mathbf{v}' = \begin{pmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -c_x \\ 0 & 1 & -c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

$$\mathbf{v}' = \mathbf{T}(\mathbf{c})\mathbf{R}(\theta)\mathbf{T}(-\mathbf{c})\mathbf{v} \quad \leftarrow \text{順番に変換行列を掛ける}$$

変換すべてが行列の形で書けるので

- 変換の順序が分かりやすい
- 変換行列の積を一つの行列として前計算可能: $\mathbf{v}' = \mathbf{M} \mathbf{v}$

同時座標表現の利点：もう少し複雑な例



例) 重心(c_x, c_y)を固定して軸a方向に s倍

通常の2次元座標表現

$$\mathbf{v}' = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} s & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \cos -\theta & -\sin -\theta \\ \sin -\theta & \cos -\theta \end{pmatrix} \left(\begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} c_x \\ c_y \end{pmatrix} \right) + \begin{pmatrix} c_x \\ c_y \end{pmatrix}$$

同次座標系表現

$$\mathbf{v}' = \mathbf{T}(\mathbf{c}) \mathbf{R}(\theta) \mathbf{S}(s, 1) \mathbf{R}(-\theta) \mathbf{T}(-\mathbf{c}) \mathbf{v}$$

$$\mathbf{T}(\mathbf{c}) = \begin{pmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{R}(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{S}(a, b) = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{v} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

θ はaとx軸の成す角

変換すべてが行列の形で書けるので

- 変換の順序が分かりやすい
- 変換行列の積を一つの行列として前計算可能: $\mathbf{v}' = \mathbf{M} \mathbf{v}$

まとめ：Affine 変換と同次座標系

平行移動と線形変換（回転・拡大など行列積による変換）の組み合わせで得られる変換を**Affine変換**と呼ぶ

同次座標系：二次元空間の点 $\begin{pmatrix} x \\ y \end{pmatrix}$ を $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ や $\begin{pmatrix} wx \\ wy \\ w \end{pmatrix}$ 表現

Affine変換の基本的な変換行列は以下の通り表現できる

拡大 Scaling	回転 Rotation	せん断 Skew	鏡映 Reflectance	平行移動 Translation
$\begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$

射影変換

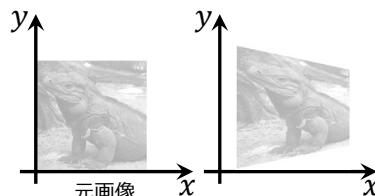
- Affine変換の基本変換は、左下2要素は0、右下は1
- これらを合成しても、左下の2要素は0、右下は1のまま

$$\begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \quad \text{合成} \quad \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}$$

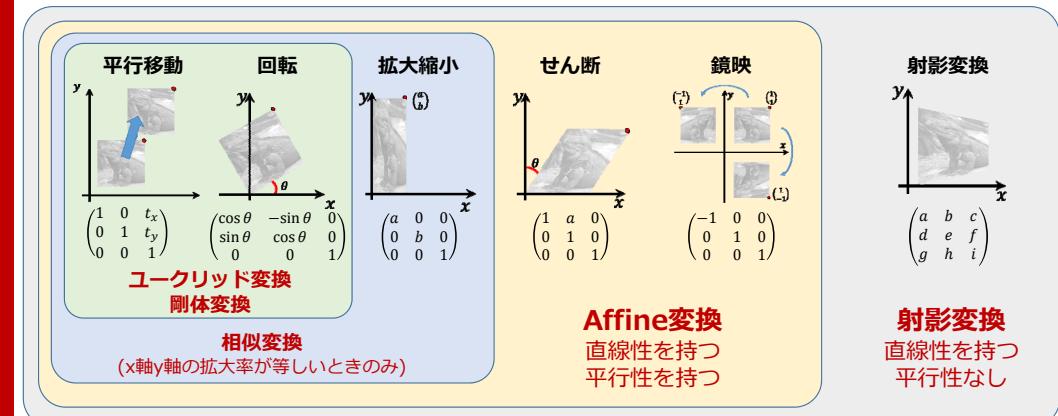
- 同次座標系表現において、より一般的な下の形の変換を**射影変換**と呼ぶ

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} \sim \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

射影変換では、第三要素が1でなくなる可能性があるので、この式は『=』ではなく『同値～』を使って表現される



変換の名称と包含関係



イメージモザイキング（パノラマ合成）

イメージモザイキング

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



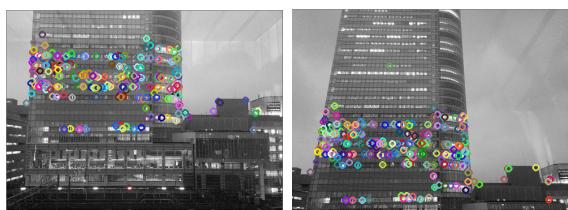
パノラマ合成1: 入力画像について特頂点を検出する



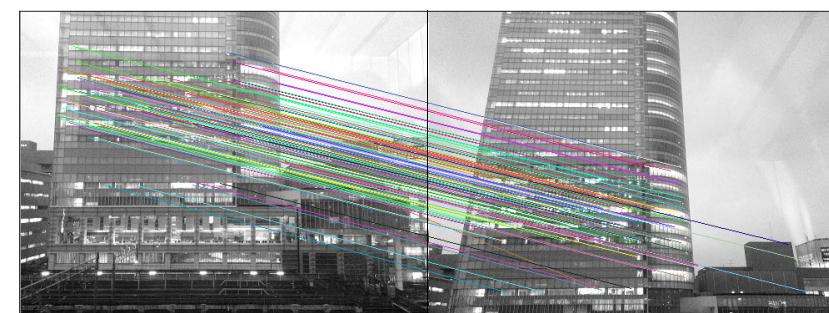
- 特徴点：角やエッジなど、顕著な局所的变化がある場所
- 特徴点検出アルゴリズムはSIFT, SURF, Eigen, Harrisなどが有名

※特徴点については3年前期のコンピュータビジョンで詳しく解説

※左図はAKAZE algorithmを利用した結果

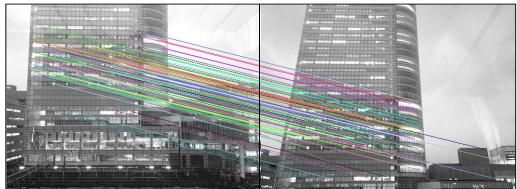


パノラマ合成2. 特徴点の対応付け



- 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
- 上図では最も似た特徴点を全探索で

パノラマ合成3. 変換行列の計算

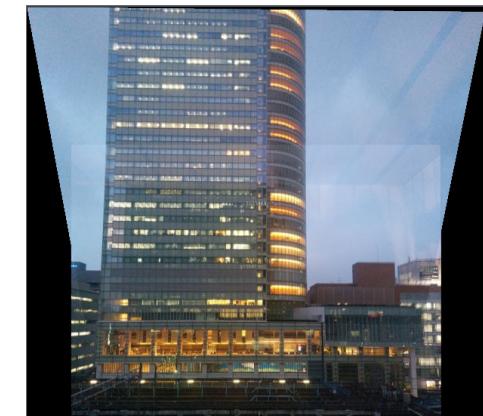
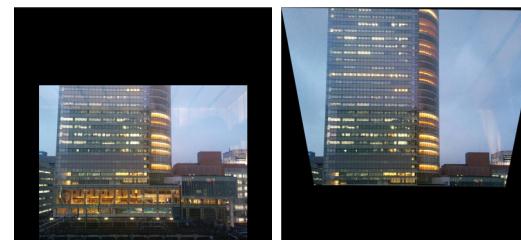


- 対応特徴点の位置が重なるよう右画像を射影変換
- つまり、対応点となるべく一致させる行列

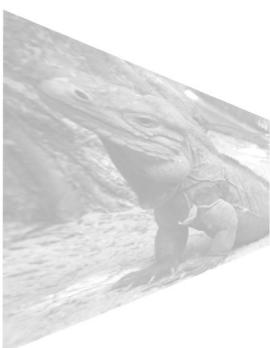
$$H = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{pmatrix}$$

- 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 4個の特徴点の組)
 - 選択した特徴点の組を用いて変換 H を導出
 - 変換 H によりほか全て特徴点を変換する
特徴点が対応点の十分近くに変換された → Inlier
特徴点の変換先が対応点から遠い → Outlier
 - 1~3を繰り返しInlier数が最多の H を出力

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り → 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]

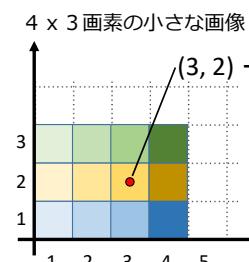


補足資料 画像の変換

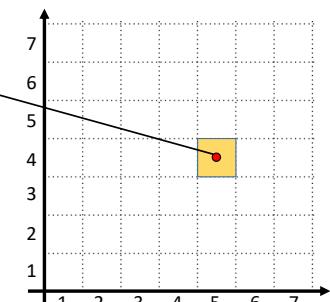
画像変換を行う際に変形後の画像の画素値を実際に計算する方法を紹介します

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…



画素の中心が
整数座標になると仮定

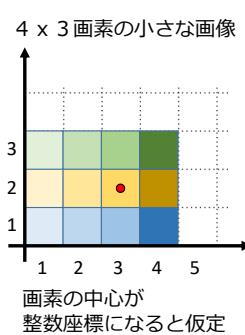


画像の変換

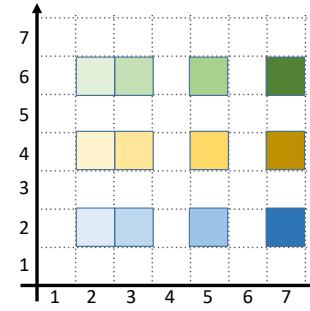
X軸方向に1.7倍, Y軸方向に2倍に拡大する変換を考える

画素の幅を1とする

各画素を変換先に移動してみると…



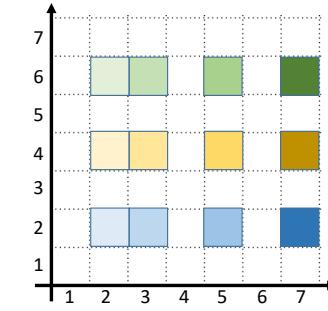
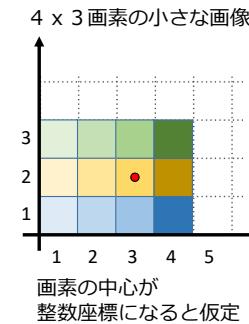
$(1, 1) \rightarrow (2, 2)$
 $(1, 2) \rightarrow (2, 4)$
 $(1, 3) \rightarrow (2, 6)$
 $(2, 1) \rightarrow (3, 2)$
 $(2, 2) \rightarrow (3, 4)$
⋮
 $(4, 1) \rightarrow (7, 2)$
 $(4, 2) \rightarrow (7, 4)$
 $(4, 3) \rightarrow (7, 6)$



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！

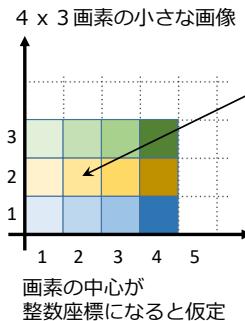


画像の変換

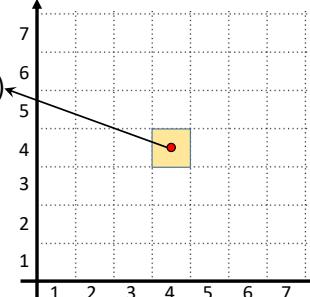
所望の変換は、X軸方向に1.7倍, Y軸方向に2倍

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



$(2, 2) \leftarrow (2.3, 2) \leftarrow (4, 4)$

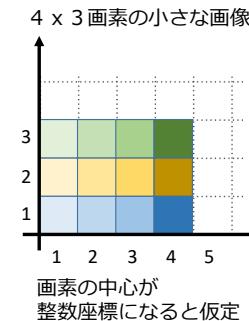


画像の変換

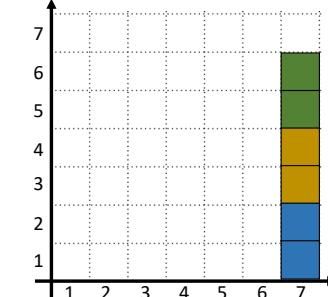
所望の変換は、X軸方向に1.7倍, Y軸方向に2倍

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



$(4,4) \leftarrow (7, 7)$
 $(4,3) \leftarrow (7, 6)$
 $(4,3) \leftarrow (7, 5)$
 $(4,2) \leftarrow (7, 4)$
 $(4,2) \leftarrow (7, 3)$
 $(4,1) \leftarrow (7, 2)$
 $(4,1) \leftarrow (7, 1)$

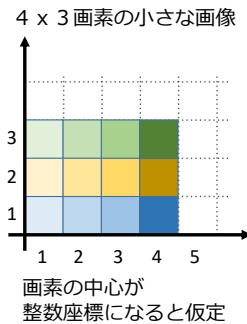


画像の変換

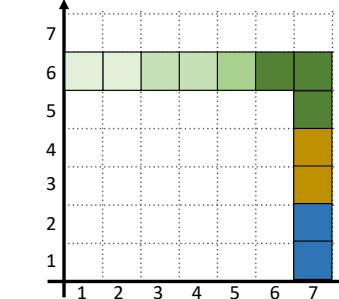
所望の変換は、X軸方向に1.7倍、Y軸方向に2倍

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



$(5,3) \leftarrow (8, 6)$
 $(4,3) \leftarrow (7, 6)$
 $(4,3) \leftarrow (6, 6)$
 $(3,3) \leftarrow (5, 6)$
 $(2,3) \leftarrow (4, 6)$
 $(2,3) \leftarrow (3, 6)$
 $(1,3) \leftarrow (2, 6)$
 $(1,3) \leftarrow (1, 6)$

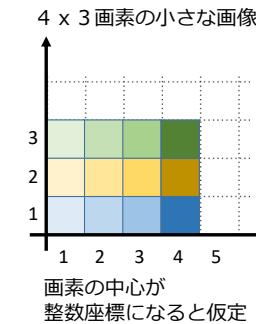


画像の変換

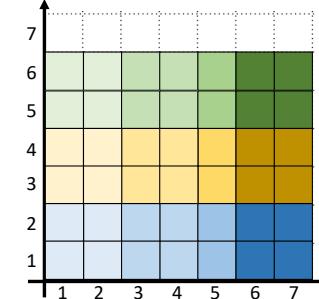
所望の変換は、X軸方向に1.7倍、Y軸方向に2倍

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



他の画素も
逆変換により
元画像の位置を検索し
コピーする



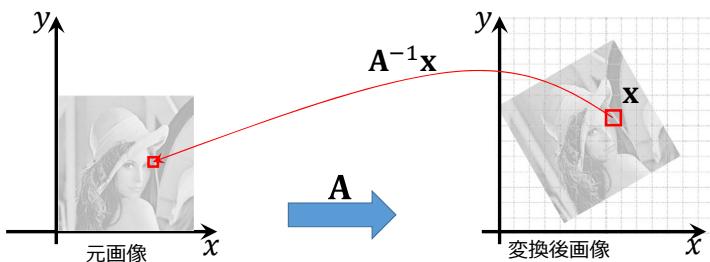
画像の変換

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

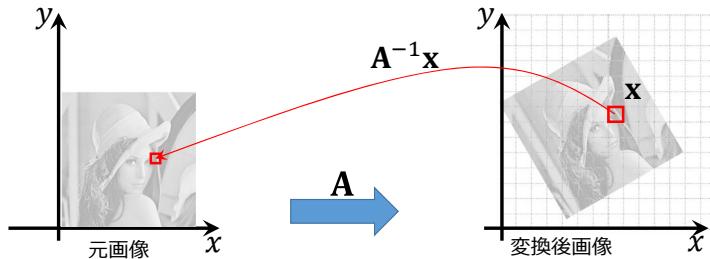
正) 変換先の各画素に逆変換を掛け、元画像を参照する

※ 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



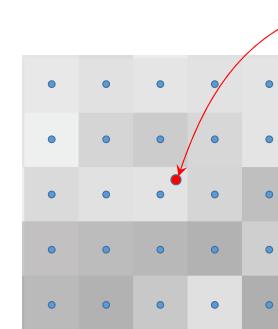
画像の補間

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。。。

なぜ画像補間が必要か？



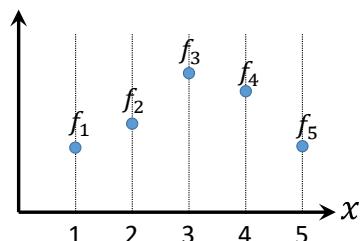
- ・赤点: サンプリングしたい場所
- ・青点: 画素値が存在する場所

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

補間法 (1D)

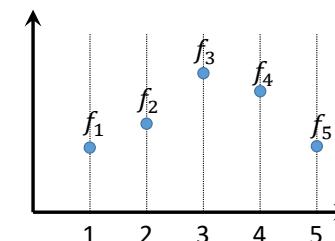


入力 : 画素値 f_i
 x が整数の位置のみに値が存在

出力 : f_i を補間した関数 $g(x)$

※修正: 連続関数 $g(x)$ → 関数 $g(x)$

補間法 (1D) : Nearest Neighbor

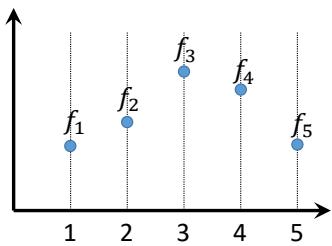


入力 : 画素値 f_i

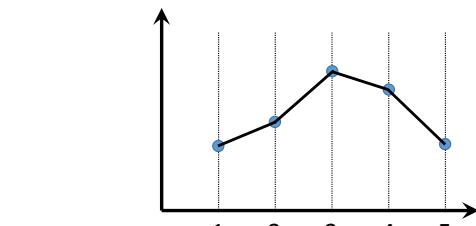
最近傍画素の値を使う
 $g(x) = f_{\lfloor x+0.5 \rfloor}$

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

補間法 (1D) : Linear Interpolation



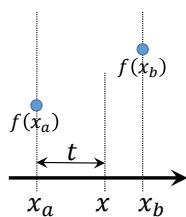
入力 : 画素値 f_i



前後 2 画素を線形に補間する

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

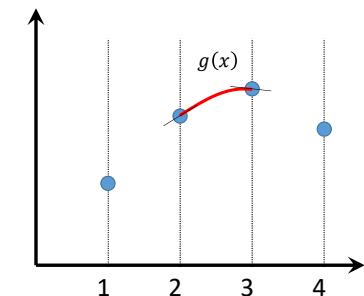
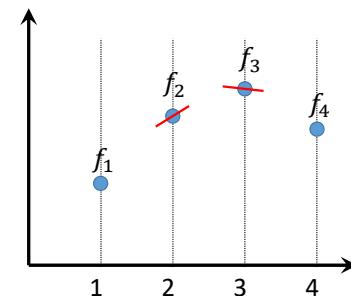
$$\begin{aligned} x_a &= \lfloor x \rfloor, \\ x_b &= \lfloor x \rfloor + 1, \\ t &= x - \lfloor x \rfloor \end{aligned}$$



補間法 (1D) : Hermite Cubic Spline Interpolation

区間[2,3]を補間するとき

- 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



補間法 (1D) : Hermite Cubic Spline Interpolation

$g(x)$ は3次の関数であるとする,

$$g(x) = ax^3 + bx^2 + cx + d \quad \text{for } x \in [0,1] \quad \dots (1)$$

境界において画素値を満たすため,

$$g(0) = f_0, \quad g(1) = f_1 \quad \dots (2)$$

境界における勾配を4点を用いて指定

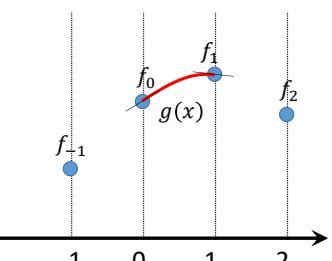
$$g'(0) = \frac{1}{2}(f_1 - f_{-1}), \quad g'(1) = \frac{1}{2}(f_2 - f_0) \quad \dots (3)$$

式(1)(2)(3)より $g(x)$ が求まる

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

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



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

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

$g(x)$ を 4 つの画素値の重み付け和で表現する

$$g(x) = h(t_{-1})f_{-1} + h(t_0)f_0 + h(t_1)f_1 + h(t_2)f_2$$

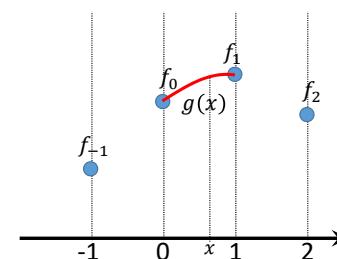
ただし, t_i は, x から画素までの距離

$$t_{-1} = x + 1, \quad t_0 = x, \quad t_1 = 1 - x, \quad t_2 = 2 - x$$

重み関数 $h(\cdot)$ は以下の通り定義される [1]

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

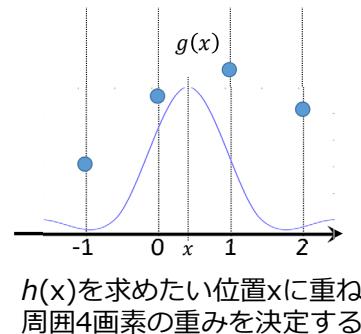
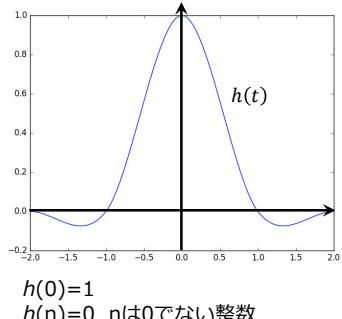
a はユーザが決める変数, $a=-0.5$ などとするとよい [1]



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

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

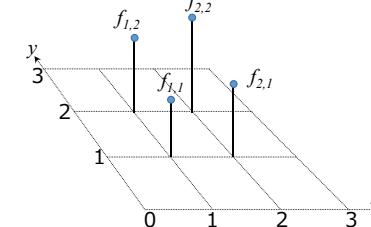
$$h(t) = \begin{cases} (a+2)|t|^3 - (a+3)|t|^2 + 1 & \text{if } |t| \leq 1 \\ a|t|^3 - 5a|t|^2 + 8a|t| - 4a & \text{if } 0 \leq |t| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$



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

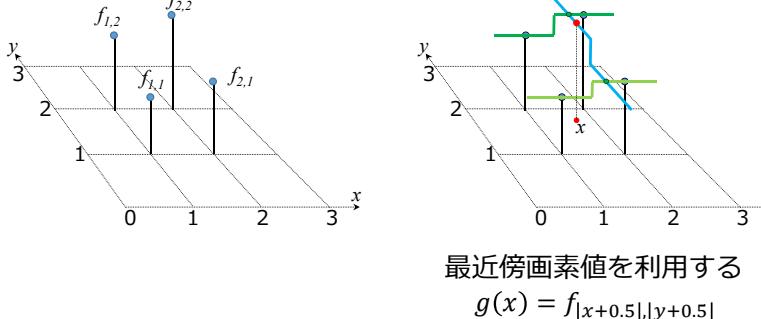
補間法 (2D)

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

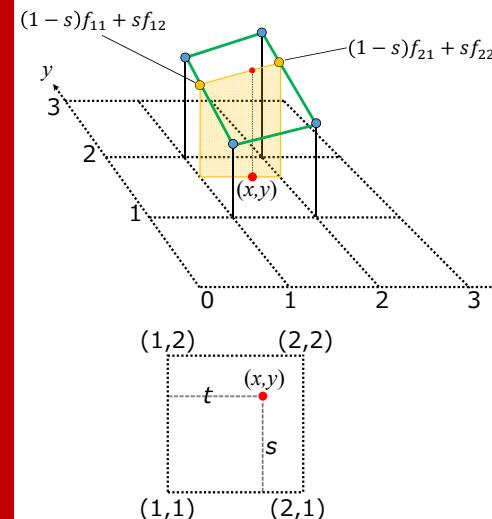


この図では、破線の交差部分に
画素中心があるとする

補間法 (2D) : Nearest neighbor



補間法 (2D) : Linear Interpolation



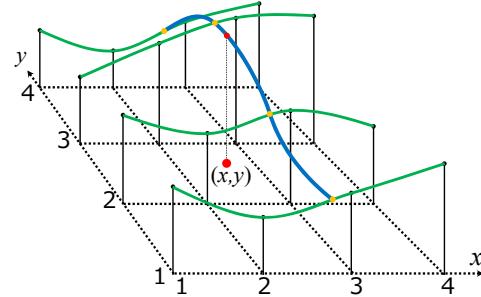
$x \in [1,2], y \in [1,2]$ の範囲を
画素 $f_{11}, f_{12}, f_{21}, f_{22}$ より補間する

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

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

上式はなにをしているのか?
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)
 $(1-t)((1-s)f_{11} + sf_{12}) + t((1-s)f_{21} + sf_{22})$

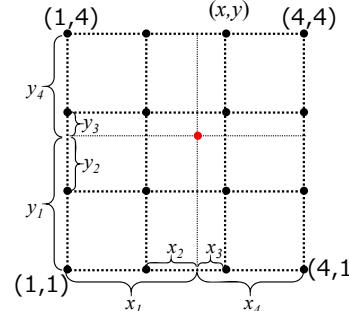
補間法 (2D) : Bicubic Convolution Interpolation



$x \in [1,2], y \in [1,2]$ 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

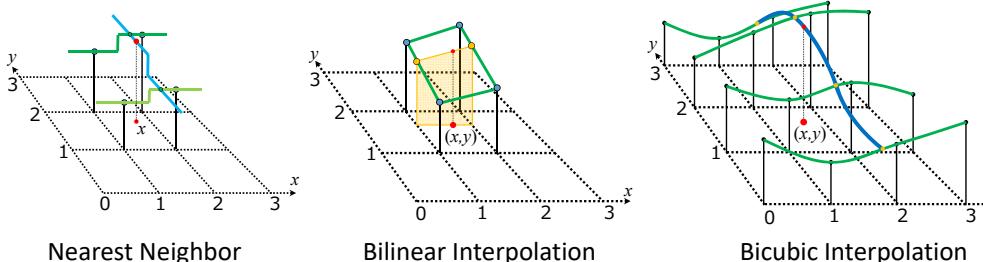
画像の補間法 : 例



教科書 p171 の図参照

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