情報デザイン専攻

画像情報処理論及び演習I

-デジタル画像の表現と応用-
その2:アフィン変換と画素値の補間

第３回講義
水曜日１限
教室6218情報処理実習室

吉澤　信
shin@riken.jp, 非常勤講師
大妻女子大学　社会情報学部

独立行政法人 理化学研究所

Shin Yoshizawa: shin@riken.jp

今日の授業内容

www.riken.jp/brict/Yoshizawa/Lectures/index.html

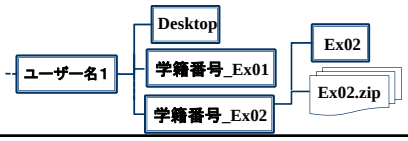
① 復習
② 講義・演習第３回:
　　アフィン変換と補間.
✓ 第一回レポートはこの内容なのでよく聞いてくださいねー
③ レポートについて、

Shin Yoshizawa: shin@riken.jp

復習:講義・演習資料・準備

www.riken.jp/brict/Yoshizawa/Lectures/Lec02_ex02.pdf
www.riken.jp/brict/Yoshizawa/Lectures/Ex02.zip
www.riken.jp/brict/Yoshizawa/Lectures/index.html

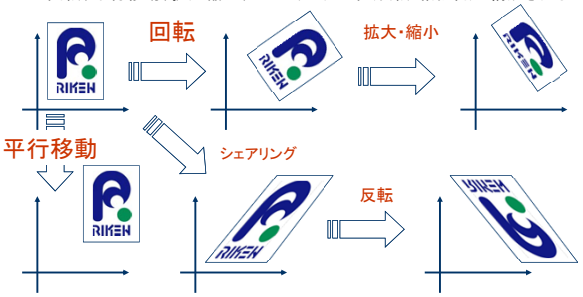
1. 第2回演習用のディレクトリーを作ってください
「mkdir 学籍番号_Ex02」.
2. firefoxを使ってEx02.zipを作ったディレクトリーにダウンロードして端末にて「unzip Ex02.zip」で展開してください.
3. 端末にて「cd Ex02」,「emacs ex02_0.cxx &」



Shin Yoshizawa: shin@riken.jp

アフィン変換とは？

✓ Affine Transformationは既知の行列 A とベクトル $\mathbf{t}$ を用いて**線形変換** $\mathbf{y} = f(\mathbf{x}) = A\mathbf{x} + \mathbf{t}$ により点 $\mathbf{x}$ を点 $\mathbf{y}$ に写像する.
- ✓ Collinearityを保存:直線は必ず直線に写像される.
- ✓ 回転、平行移動、拡大縮小、シェアリングの組(反転・相似含)で構成される.



Shin Yoshizawa: shin@riken.jp

アフィン変換とは？

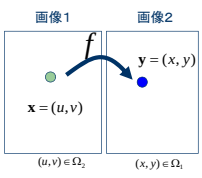
✓ Affine Transformationは既知の行列 A とベクトル $\mathbf{t}$ を用いて**線形変換** $\mathbf{y} = f(\mathbf{x}) = A\mathbf{x} + \mathbf{t}$ により点 $\mathbf{x}$ を点 $\mathbf{y}$ に写像する.
✓ 2次元では、

$$\mathbf{y} = (x, y), \mathbf{x} = (u, v), \mathbf{t} = (b_1, b_2), A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix},$$

とすると、アフィン変換は以下の様に書ける.

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

$$f: \quad (u, v) \rightarrow (x, y)$$



Shin Yoshizawa: shin@riken.jp

演習:アフィン変換

✓ アフィン変換をする関数を作ってみよう！

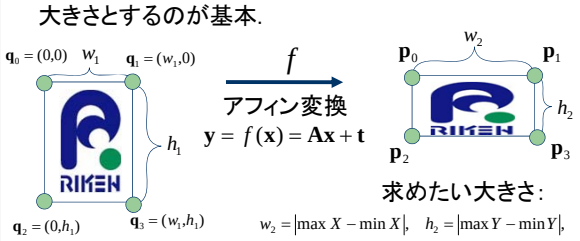
$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

```
void AffineTransformation_FW(double inX[2],double A[2][2],
                             double t[2], double outX[2]){
    outX[0] = A[0][0]*inX[0]+A[0][1]*inX[1]+t[0];
    outX[1] = A[1][0]*inX[0]+A[1][1]*inX[1]+t[1];
}
```

Shin Yoshizawa: shin@riken.jp

重要: 変換後の画像の大きさ

✓ 変換後の画像の大きさは、まず入力画像の四隅の座標をアフィン変換してその横幅と縦の高さを新たな画像の大きさとするのが基本.



求めたい大きさ:

$w_2=|\max X-\min X|, \quad h_2=|\max Y-\min Y|,$

max,min: 4つのpの中で最も大きな/小さなX or Y座標.

$\mathbf{p}_0=f(\mathbf{q}_0),\ \mathbf{p}_1=f(\mathbf{q}_1),\ \mathbf{p}_2=f(\mathbf{q}_2),\ \mathbf{p}_3=f(\mathbf{q}_3).$

Shin Yoshizawa: shin@riken.jp

変換後の画像の大きさ

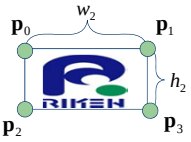
max,min: 4つのpの中で最も大きな/小さなX or Y座標.

$\mathbf{p}_0=f(\mathbf{q}_0),\ \mathbf{p}_1=f(\mathbf{q}_1),\ \mathbf{p}_2=f(\mathbf{q}_2),\ \mathbf{p}_3=f(\mathbf{q}_3).$

$\mathbf{p}_j=(x_j,y_j), \quad j=0,1,2,3.$ とすると、

$\max X=\max(x_j), \quad j=0,1,2,3.$
$\min X=\min(x_j,0), \quad j=0,1,2,3.$

$\max Y=\max(y_j), \quad j=0,1,2,3.$
$\min Y=\min(y_j,0), \quad j=0,1,2,3.$



変換後の画像の大きさ:

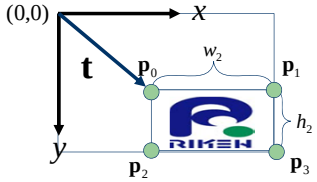
$w_2=|\max X-\min X|,$
$h_2=|\max Y-\min Y|.$

Shin Yoshizawa: shin@riken.jp

変換後の画像の大きさ

✓ 平行移動も入っているときは平行移動ベクトルの大きさを最後に足す事に注意.

$\mathbf{t}=(b_1,b_2)$



最終的な画像の大きさ:

(w_2+b_1,h_2+b_2)

変換後の画像の大きさ:

$w_2=|\max X-\min X|,$
$h_2=|\max Y-\min Y|.$

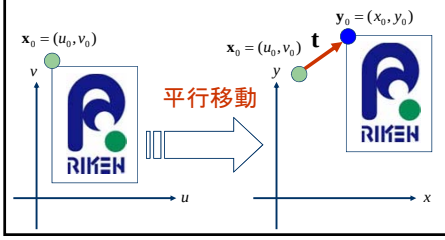
✓ Ex02/ex02_0.cxx のsetNewSize()関数にて計算可能.

Shin Yoshizawa: shin@riken.jp

平行移動: Translation

$$\begin{pmatrix}x\\y\end{pmatrix}=\begin{pmatrix}a_{11}&a_{12}\\a_{21}&a_{22}\end{pmatrix}\begin{pmatrix}u\\v\end{pmatrix}+\begin{pmatrix}b_1\\b_2\end{pmatrix}\Rightarrow\begin{pmatrix}x\\y\end{pmatrix}=\begin{pmatrix}u\\v\end{pmatrix}+\begin{pmatrix}b_1\\b_2\end{pmatrix}$$

✓ 平行移動ベクトル: $\mathbf{t}=(b_1,b_2)=\mathbf{y}_0-\mathbf{x}_0=(x_0-u_0,y_0-v_0),$



✓ 変換行列Aは単位行列になる:

$$A=\begin{pmatrix}1&0\\0&1\end{pmatrix}.$$

Shin Yoshizawa: shin@riken.jp

平行移動: 演習

✓ 平行移動する関数を作ってみよう！

$$\begin{pmatrix}x\\y\end{pmatrix}=\begin{pmatrix}u\\v\end{pmatrix}+\begin{pmatrix}b_1\\b_2\end{pmatrix}$$

```
void Translate(double inX[2],double t[2], double outX[2]){
    outX[0] = inX[0]+t[0];
    outX[1] = inX[1]+t[1];
}
```

Shin Yoshizawa: shin@riken.jp

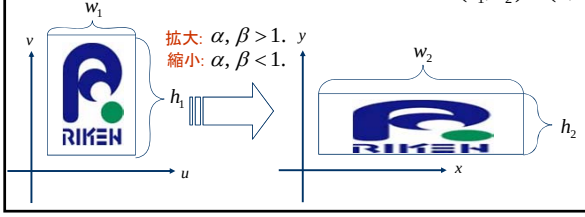
拡大・縮小1: Scaling

$$\begin{pmatrix}x\\y\end{pmatrix}=\begin{pmatrix}a_{11}&a_{12}\\a_{21}&a_{22}\end{pmatrix}\begin{pmatrix}u\\v\end{pmatrix}+\begin{pmatrix}b_1\\b_2\end{pmatrix}\Rightarrow\begin{pmatrix}x\\y\end{pmatrix}=\begin{pmatrix}\alpha&0\\0&\beta\end{pmatrix}\begin{pmatrix}u\\v\end{pmatrix}$$

✓ X-スケーリング・ファクター: $\alpha=w_2/w_1.$

✓ Y-スケーリング・ファクター: $\beta=h_2/h_1.$

✓ 平行移動ベクトルはゼロベクトルになる: $\mathbf{t}=(b_1,b_2)=(0,0),$



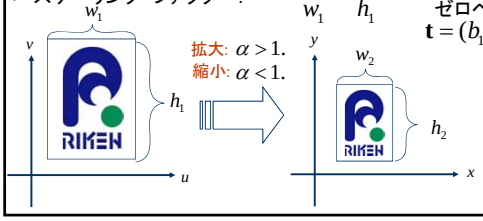
Shin Yoshizawa: shin@riken.jp

拡大・縮小2：Scaling

$\alpha = \beta$ のとき縦横比(Aspect Ratio)は保存される:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \alpha & 0 \\ 0 & \alpha \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$$

- スケーリング・ファクター: $\alpha = \frac{w_2}{w_1} = \frac{h_2}{h_1}.$
- 平行移動ベクトルはゼロベクトルになる: $\mathbf{t} = (b_1, b_2) = (0,0),$



Shin Yoshizawa: shin@riken.jp

拡大縮小：演習

- 2次元配列A[2][2]にScaling行列をセットする関数を作ってみよう！

$$A = \begin{pmatrix} \alpha & 0 \\ 0 & \beta \end{pmatrix}$$

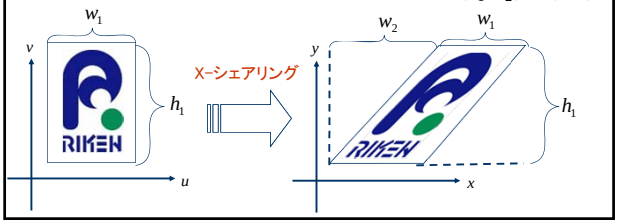
```
void Scaling(double A[2][2],double xfac, double yfac){
    A[0][0] = xfac;
    A[0][1] = A[1][0] = 0.0;
    A[1][1] = yfac;
}
```

Shin Yoshizawa: shin@riken.jp

シェアリング1：X-Shearing

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 & \alpha_s \\ 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$$

- X-シェア・ファクター: $\alpha_s = w_2 / h_1.$
- 平行移動ベクトルはゼロベクトルになる: $\mathbf{t} = (b_1, b_2) = (0,0),$

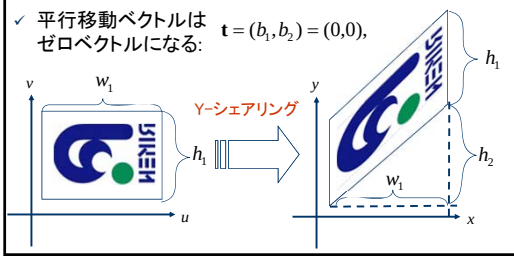


Shin Yoshizawa: shin@riken.jp

シェアリング2：Y-Shearing

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \beta_s & 1 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$$

- Y-シェア・ファクター: $\beta_s = h_2 / w_1.$
- 平行移動ベクトルはゼロベクトルになる: $\mathbf{t} = (b_1, b_2) = (0,0),$

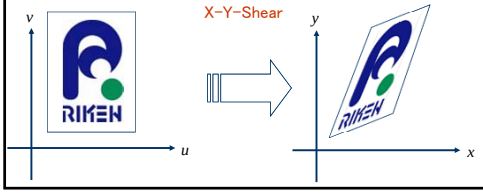


Shin Yoshizawa: shin@riken.jp

シェアリング3：Shearing

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 & \alpha_s \\ \beta_s & 1 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$$

- X-シェア・ファクター: $\alpha_s.$
- Y-シェア・ファクター: $\beta_s.$
- 平行移動ベクトルはゼロベクトルになる: $\mathbf{t} = (b_1, b_2) = (0,0),$



Shin Yoshizawa: shin@riken.jp

シェアリング：演習

- 2次元配列A[2][2]にShearing行列をセットする関数を作ってみよう！

$$A = \begin{pmatrix} 1 & \alpha_s \\ \beta_s & 1 \end{pmatrix}$$

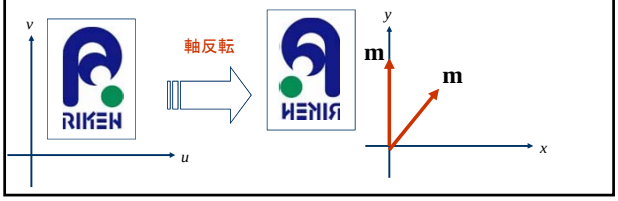
```
void Shearing(double A[2][2],double xfac, double yfac){
    A[0][0] = A[1][1] = 1.0;
    A[0][1] = xfac;
    A[1][0] = yfac;
}
```

Shin Yoshizawa: shin@riken.jp

反転

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{|\mathbf{m}|^2} \begin{pmatrix} m_x^2 - m_y^2 & 2m_x m_y \\ 2m_x m_y & m_y^2 - m_x^2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$$

- 原点を通る反転軸の方向ベクトル：$\mathbf{m} = (m_x, m_y)$.
- 平行移動ベクトルはゼロベクトルになる: $\mathbf{t} = (b_1, b_2) = (0,0)$.



Shin Yoshizawa: shin@riken.jp

反転：演習

- 2次元配列A[2][2]に反転行列をセットする関数を作ってみよう！

$$A = \frac{1}{|\mathbf{m}|^2} \begin{pmatrix} m_x^2 - m_y^2 & 2m_x m_y \\ 2m_x m_y & m_y^2 - m_x^2 \end{pmatrix}$$

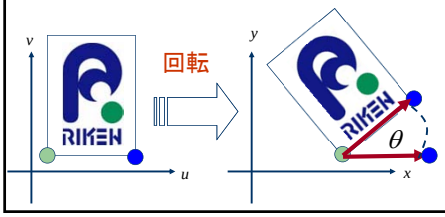
```
void Line_Inversion(double A[2][2],double mxy[2]){
    double x2=mxy[0]*mxy[0];
    double y2=mxy[1]*mxy[1];
    double msize = x2+y2;
    A[0][0] = (x2-y2)/msize;
    A[0][1] = A[1][0] = (2.0*mxy[0]*mxy[1])/msize;
    A[1][1] = -A[0][0];
}
```

Shin Yoshizawa: shin@riken.jp

回転1：Rotation

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$$

- 回転行列：$A = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$,
- 平行移動ベクトルはゼロベクトルになる: $\mathbf{t} = (b_1, b_2) = (0,0)$,



- 回転角：θ,

Shin Yoshizawa: shin@riken.jp

回転：演習

- 2次元配列A[2][2]に回転行列をセットする関数を作ってみよう！

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

```
void Rotation(double A[2][2],double theta){
    A[0][0] = A[1][1] = cos(theta);
    A[0][1] = -sin(theta);
    A[1][0] = -A[0][1];
}
```

Shin Yoshizawa: shin@riken.jp

回転：演習

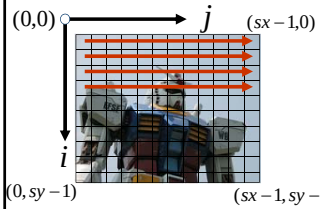
- 注意1：cos(), sin()はmath.hをインクルードしてコンパイルのときには最後に「-lm」が必要.絶対値を取る関数fabs(), abs()も同様.
- 注意2：math.hに入っている三角関数は弧度法(ラジアン)なので…

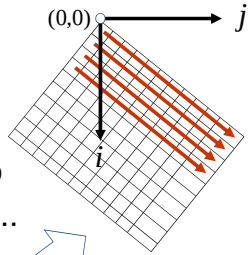
```
void Rotation(double A[2][2],double theta){
    double rad = (theta*PI)/180.0;
    A[0][0] = A[1][1] = cos(rad);
    A[0][1] = -sin(rad);
    A[1][0] = -A[0][1];
}
```

Shin Yoshizawa: shin@riken.jp

回転2：Rotation

- 画像の座標系は…





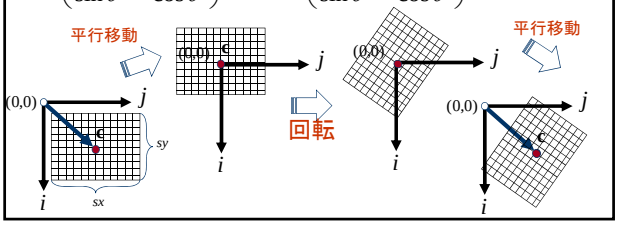
- なので普通に回転すると…

Shin Yoshizawa: shin@riken.jp

回転3：Rotation

- 画像の中心を原点とする座標系で回転させたい場合は一回、中心に平行移動させて回転後に逆向きの平行移動をする：　$\mathbf{c} = (sx/2, sy/2)$

$$\mathbf{y} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}\mathbf{x} \Rightarrow \mathbf{y} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}(\mathbf{x}-\mathbf{c})+\mathbf{c}$$

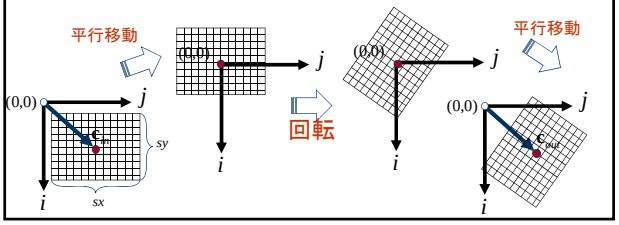


Shin Yoshizawa: shin@riken.jp

回転4：Rotation

- 入力と出力の画像サイズが違うとき：

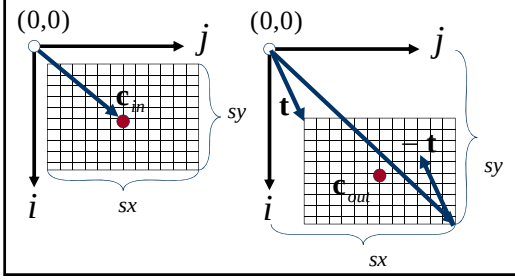
$$\mathbf{y} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}(\mathbf{x}-\mathbf{c}_{in})+\mathbf{c}_{out}$$



Shin Yoshizawa: shin@riken.jp

回転5：Rotation

- 平行移動も入っているときは出力画像の中心位置$\mathbf{c}_{out}$が：(画像サイズ－平行移動ベクトル)の半分になる事に注意.　$\mathbf{c}_{out} = ((sx, sy)-\mathbf{t})/2.$



Shin Yoshizawa: shin@riken.jp

演習：アフィン変換2

- 中心で変換をする関数を作ってみよう！

$$\mathbf{y} = A(\mathbf{x}-\mathbf{c}_{in})+\mathbf{c}_{out}+\mathbf{t}$$

```
void AffineTransformation_FW(double inX[2],double A[2][2],
      double ci[2],double co[2], double t[2], double outX[2]){
   double dx = inX[0]−ci[0];
   double dy = inX[1]−ci[1];
   outX[0] = A[0][0]*dx+A[0][1]*dy+co[0]+t[0];
   outX[1] = A[1][0]*dx+A[1][1]*dy+co[1]+t[1];
}
```

Shin Yoshizawa: shin@riken.jp

写像の合成：複数の変換転

- 回転の後に拡大・縮小等、複数変換を行う場合は、画像を変換毎にセーブするのではなく、写像を合成して変換を行うのが基本:

$$\mathbf{y} = f(\mathbf{x}) = \mathbf{A}\mathbf{x}+\mathbf{t} \qquad f(\mathbf{x}) = f_1 \circ f_2 \circ \ldots \circ f_n(\mathbf{x})$$

$$A = \mu \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}\begin{pmatrix} 1 & 0 \\ \gamma & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{\lambda_x} \end{pmatrix}\begin{pmatrix} \lambda_x & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & \gamma \\ 0 & 1 \end{pmatrix}$$

拡大・縮小　回転　Y−シェア　拡大・縮小　拡大・X−シェア
Y−Dilatation　縮小
Y−Dilatation

Shin Yoshizawa: shin@riken.jp

写像の分解1

- 行列分解の種類だけ分解可能：アフィン変換の行列は以下の9種類に分類される：

LDU分解：

Type 1: $A = \mu \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}\begin{pmatrix} 1 & 0 \\ \gamma & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{\lambda_x} \end{pmatrix}\begin{pmatrix} \lambda_x & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & \gamma \\ 0 & 1 \end{pmatrix}$

Type 2: $A = \mu \begin{pmatrix} 1 & 0 \\ \gamma & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{\lambda_x} \end{pmatrix}\begin{pmatrix} \lambda_x & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & \gamma \\ 0 & 1 \end{pmatrix}\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$

Type 3: $A = \mu \begin{pmatrix} 1 & 0 \\ \gamma_y & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{\lambda_x} \end{pmatrix}\begin{pmatrix} \lambda_x & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & \gamma_x \\ 0 & 1 \end{pmatrix}$

Type 4: $A = \mu \begin{pmatrix} 1 & \gamma_x \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{\lambda_x} \end{pmatrix}\begin{pmatrix} \lambda_x & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ \gamma_y & 1 \end{pmatrix}$

Shin Yoshizawa: shin@riken.jp

写像の分解2

- 行列分解の種類だけ分解可能：アフィン変換の行列は以下の9種類に分類される：

QR分解：
- Type 5: $A=\mu\begin{pmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & \frac{1}{\lambda_x}\end{pmatrix}\begin{pmatrix}\lambda_x & 0\\ 0 & 1\end{pmatrix}\begin{pmatrix}1 & \gamma\\ 0 & 1\end{pmatrix}$
- Type 6: $A=\mu\begin{pmatrix}1 & 0\\ 0 & \frac{1}{\lambda_x}\end{pmatrix}\begin{pmatrix}\lambda_x & 0\\ 0 & 1\end{pmatrix}\begin{pmatrix}1 & \gamma\\ 0 & 1\end{pmatrix}\begin{pmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{pmatrix}$

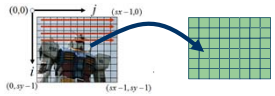
QL分解：
- Type 7: $A=\mu\begin{pmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & \frac{1}{\lambda_x}\end{pmatrix}\begin{pmatrix}\lambda_x & 0\\ 0 & 1\end{pmatrix}\begin{pmatrix}1 & 0\\ \gamma & 1\end{pmatrix}$
- Type 8: $A=\mu\begin{pmatrix}1 & 0\\ 0 & \frac{1}{\lambda_x}\end{pmatrix}\begin{pmatrix}\lambda_x & 0\\ 0 & 1\end{pmatrix}\begin{pmatrix}1 & 0\\ \gamma & 1\end{pmatrix}\begin{pmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{pmatrix}$

特異値分解：
- Type 9: $A=\mu\begin{pmatrix}\cos\theta_1 & -\sin\theta_1\\ \sin\theta_1 & \cos\theta_1\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & \frac{1}{\lambda_x}\end{pmatrix}\begin{pmatrix}\lambda_x & 0\\ 0 & 1\end{pmatrix}\begin{pmatrix}\cos\theta_2 & -\sin\theta_2\\ \sin\theta_2 & \cos\theta_2\end{pmatrix}$

Shin Yoshizawa: shin@riken.jp

演習：実際にアフィン変換をしてみる！

- まずは、ex02_0.cxxのmain内の/* Forward Mapping */の下に変換のループを書く：




```
/* Forward Mapping */
double uv[2],xy[2];
for(i=0;i<in->sy;i++){ uv[1] = ((double)(i));
    for(j=0;j<in->sx;j++){ uv[0] = ((double)(j));
    AffineTransformation_FW(uv,A,t,centerIn,centerOut,xy);
    int out_i = getImage_Coordinate(out->sy,xy[1]);
    int out_j = getImage_Coordinate(out->sx,xy[0]);
    out->img[out_i][out_j] = in->img[i][j];
}
/* end main processing */
```

Shin Yoshizawa: shin@riken.jp

演習：実際にアフィン変換をしてみる！

- 回転してみよう！：ex02_0.cxxのmain内の上の方でdouble A[2][2];にA[1][0] = 1.0…と入れている下にで、作った回転行列関数Rotation()を用いて回転行列を作ってみる.

```
double A[2][2];
A[0][0]=1.0; A[0][1] = 0.0;
A[1][0] = 0.0; A[1][1] = 1.0;

Rotation(A,45.0);
```

- ex02_0.cxxをセーブしてコンパイル

「g++ -o ex02_0 ex02_0.cxx -lm」後に実行してみよう

「./ex02_0 lena.pgm test_R45.pgm」,「display test_R45.pgm」

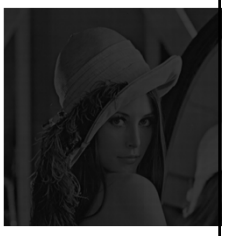
Shin Yoshizawa: shin@riken.jp

演習：実際にアフィン変換をしてみる！

- 拡大してみよう！：回転と同様にやってみてください.

```
double A[2][2];
A[0][0]=1.0; A[0][1] = 0.0;
A[1][0] = 0.0; A[1][1] = 1.0;

//Rotation(A,45.0);
Scaling(A,2.0,2.0);
```



- ex02_0.cxxをセーブしてコンパイル

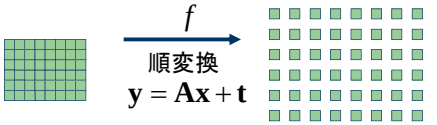
「g++ -o ex02_0 ex02_0.cxx -lm」後に実行してみよう

「./ex02_0 lena.pgm test_S2.pgm」

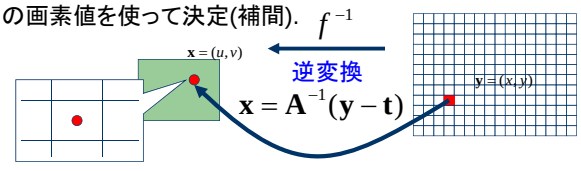
Shin Yoshizawa: shin@riken.jp

重要：画像では？ 順変換と逆変換1

- 逆変換 ＋ 画素値の補間によるアフィン変換：
  - 順写像(Forward Mapping) VS 逆写像(Backward Mapping):

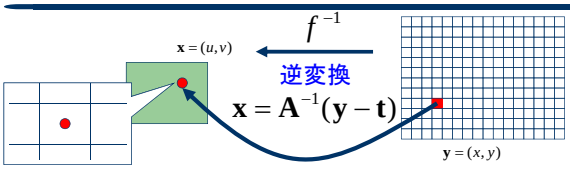


逆変換した位置の画素値を周りの画素値を使って決定(補間).



Shin Yoshizawa: shin@riken.jp

順変換と逆変換2



$$\begin{pmatrix}x\\ y\end{pmatrix}=\begin{pmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{pmatrix}\begin{pmatrix}u\\ v\end{pmatrix}+\begin{pmatrix}b_1\\ b_2\end{pmatrix}$$

- ad−bc=0になる様な変換は作ってはいけない.
- ↓なので、

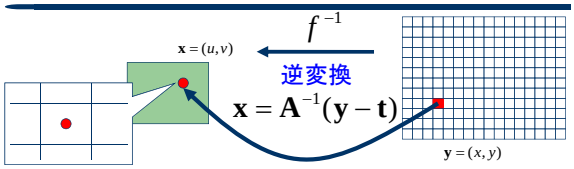
$$A=\begin{pmatrix}a & b\\ c & d\end{pmatrix}\Leftrightarrow A^{-1}=\frac{1}{ad-bc}\begin{pmatrix}d & -b\\ -c & a\end{pmatrix}$$

逆変換：

$$\begin{pmatrix}u\\ v\end{pmatrix}=\frac{1}{a_{11}a_{22}-a_{12}a_{21}}\begin{pmatrix}a_{22} & -a_{12}\\ -a_{21} & a_{11}\end{pmatrix}\begin{pmatrix}x-b_1\\ y-b_2\end{pmatrix}$$

Shin Yoshizawa: shin@riken.jp

順変換と逆変換3



画像の中心を原点とする座標系では:

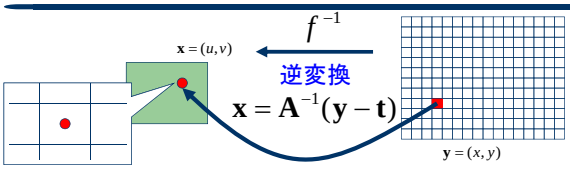
$$\mathbf{x}=\mathbf{A}^{-1}(\mathbf{y}-\mathbf{c}-\mathbf{t})+\mathbf{c},\quad \mathbf{c}=(c_x,c_y)=(sx/2,sy/2),$$

逆変換:

$$\begin{pmatrix}u\\v\end{pmatrix}=\frac{1}{a_{11}a_{22}-a_{12}a_{21}}\begin{pmatrix}a_{22}&-a_{12}\\-a_{21}&a_{11}\end{pmatrix}\begin{pmatrix}x-c_x-b_1\\y-c_y-b_2\end{pmatrix}+\begin{pmatrix}c_x\\c_y\end{pmatrix}$$

Shin Yoshizawa: shin@riken.jp

順変換と逆変換4



✓ 入力と出力の画像サイズが違うとき:

$$\mathbf{x}=\mathbf{A}^{-1}(\mathbf{y}-\mathbf{c}_{out}-\mathbf{t})+\mathbf{c}_{in},$$

逆変換:

$$\begin{pmatrix}u\\v\end{pmatrix}=\frac{1}{a_{11}a_{22}-a_{12}a_{21}}\begin{pmatrix}a_{22}&-a_{12}\\-a_{21}&a_{11}\end{pmatrix}\begin{pmatrix}x-c^{out}{}_x-b_1\\y-c^{out}{}_y-b_2\end{pmatrix}+\begin{pmatrix}c^{in}{}_x\\c^{in}{}_y\end{pmatrix}$$

Shin Yoshizawa: shin@riken.jp

演習:アフィン変換3

✓ 中心で逆変換をする関数を作ってみよう!

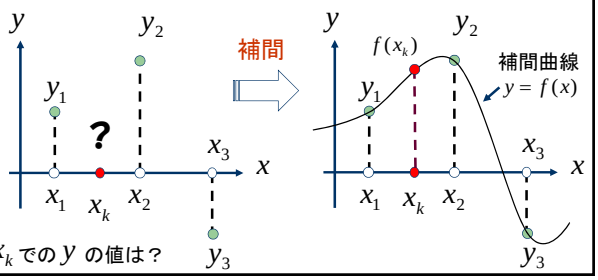
$$\mathbf{x}=\mathbf{A}^{-1}(\mathbf{y}-\mathbf{c}_{out}-\mathbf{t})+\mathbf{c}_{in},$$

```
void AffineTransformation_BW(double inX[2],double A[2][2],
      double ci[2],double co[2], double t[2], double outX[2]){
   double tmp1[2],tmp2[2],tmp3[2];
   double mt[2]; mt[0] = -(co[0]+t[0]); mt[1] = -(co[1]+t[1]);
  Translate(inX,mt,tmp1);
   Vector_Matrix_Multiplication_Inverse(tmp1,A,tmp2);
   Translate(tmp2,ci,outX);
}
```

Shin Yoshizawa: shin@riken.jp

補間(Interpolation)とは?

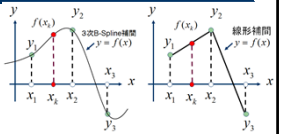
✓ 与えられた既知の数値データ列を基に、そのデータ列の間の数値(又は既知の数値データ列を通る関数)を求める事、内挿とも言う. 与えられたデータ外を考える場合は補外(外挿)と言う.



Shin Yoshizawa: shin@riken.jp

様々な補間法

✓ 多くの方法がある:
- ラグランジュ補間、多項式、
- 線形補間(直線の式)、Sinc関数、
- スプライン補間(B-Spline, etc.)、
- RBF (Radial Basis Function)、エルミート補間等.



✓ 一番基本でよく用いられているのは線形補間と3次スプライン補間(Cubic Spline Interpolation).

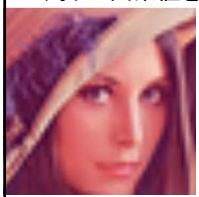
✓ 補間はいろいろ使える.

©Y. Ohtake, 2011. ©S. Yoshizawa et al. ACM SMA, 2003.

Shin Yoshizawa: shin@riken.jp

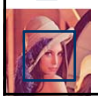
重要:画像では?画素値の補間(2D)

✓ 周りの画素値を使って補間:

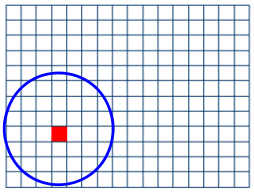


逆変換+線形補間

5倍拡大



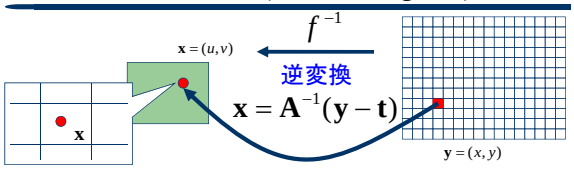
順変換



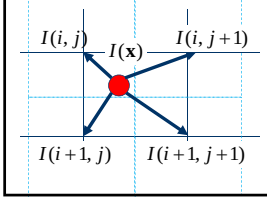
今日は...
最近傍法
線形補間法
3次Spline補間法

Shin Yoshizawa: shin@riken.jp

補間：最近傍法(Nearest Neighbor)



✓ 近傍4つの画素値のうち最も近い画素の値を使う.



$$I(\mathbf{x}) = \arg\min_{I}(|\mathbf{x} - \mathbf{p}_j|), \quad j = 0,1,2,3.$$

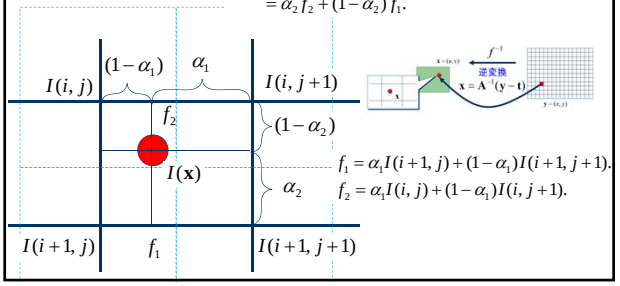
$$\mathbf{p}_0 = (i, j), \quad \mathbf{p}_1 = (i, j+1),$$
$$\mathbf{p}_2 = (i+1, j), \quad \mathbf{p}_3 = (i+1, j+1).$$

Shin Yoshizawa: shin@riken.jp

補間：線形補間法(Linear Interpolation)

✓ 近傍4つの画素値を線形補間する.

$$I(\mathbf{x}) = \alpha_2(\alpha_1 I(i, j) + (1-\alpha_1) I(i, j+1)) + (1-\alpha_2)(\alpha_1 I(i+1, j) + (1-\alpha_1) I(i+1, j+1))$$
$$= \alpha_2 f_2 + (1-\alpha_2) f_1.$$



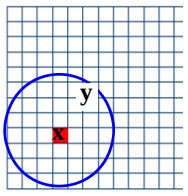
$f_1 = \alpha_1 I(i+1, j) + (1-\alpha_1) I(i+1, j+1).$
$f_2 = \alpha_1 I(i, j) + (1-\alpha_1) I(i, j+1).$

Shin Yoshizawa: shin@riken.jp

補間：線形畳み込み法

✓ 線形補間や3次Spline補間は「線形畳み込み(Linear Convolution)」と呼ばれる重み付和で計算出来る.重みの事を補間関数という.

✓ この方法は画像等の間隔が同じ格子が並んでいる場合に簡単に適用出来る.



g とIの畳み込み:

補間したい画素の位置 $\mathbf{x}$　　既知の輝度値

$$I^{\text{new}}(\mathbf{x}) = \int g(\mathbf{x} - \mathbf{y}) I(\mathbf{y}) d\mathbf{y}$$

補間された輝度値　　補間関数

$\mathbf{x}$近傍の画素の位置 $\mathbf{y}$

Shin Yoshizawa: shin@riken.jp

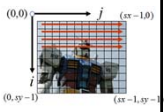
補間：線形畳み込み法

✓ 画像では、

$$I^{\text{new}}(\mathbf{x}) = \int g(\mathbf{x} - \mathbf{y}) I(\mathbf{y}) d\mathbf{y} = \iint g(x-u, y-v) I(u, v) du dv,$$

もしも、　$g(u, v) = g_1(u) g_2(v)$　ならば、

$$I^{\text{new}}(\mathbf{x}) = \iint g(x-u, y-v) I(u, v) du dv$$
$$= \iint g_1(x-u) g_2(y-v) I(u, v) du dv$$
$$= \int g_2(y-v) \left(\int g_1(x-u) I(u, v) du \right) dv$$
$$\approx \sum_i^N g_2(y-i) \left(\sum_j^M g_1(x-j) I[i][j] \right).$$

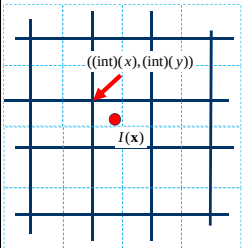


Shin Yoshizawa: shin@riken.jp

補間：3次補間法(Cubic Interpolation)

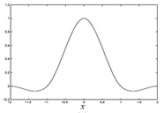
✓ 近傍16個の画素値を3次補間する.

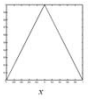
$$I^{\text{new}}(\mathbf{x}) = \sum_{i=(\text{int})(y)-1}^{(\text{int})(y)+2} g(y-i) \left(\sum_{j=(\text{int})(x)-1}^{(\text{int})(x)+2} g(x-j) I(i, j) \right).$$
$\mathbf{x} = (x, y)$



$$g(x) = \begin{cases} (a+2)|x|^3 - (a+3)|x|^2 + 1, & 0 \le |x| < 1 \\ a|x|^3 - 5a|x|^2 + 8a|x| - 4a, & 1 \le |x| < 2 \\ 0 & 2 \le |x| \end{cases}$$

aは−0.5～2がよい.


3次補間関数


線形補間関数

Shin Yoshizawa: shin@riken.jp

重要：補間法実装

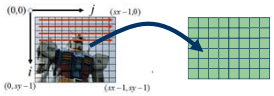
✓ Ex02/interpolation.hに
- 最近傍法：NearestNeighbor(…)
- 線形補間法：LinearInterpolation(…)
- 3次補間法：CubicInterpolation(…)

が実装されているのでよく見ておいてください.

Shin Yoshizawa: shin@riken.jp

演習: 実際に逆変換をしてみる！

- まずは、ex02_0.cxxのmain内の/* Forward Mapping */の下を以下の様に書き換える.




```
/* Forward Mapping */
double uv[2],xy[2];
for(i=0;i<out->sy;i++){ uv[1] = ((double)(i));
    for(j=0;j<out->sx;j++){ uv[0] = ((double)(j));
    AffineTransformation_BW(uv,A,t,centerIn,centerOut,xy);
    out->img[i][j] = CubicInterpolation(in,xy);
}
/* end main processing */
```

Shin Yoshizawa: shin@riken.jp

演習: 実際に逆変換をしてみる！

- 先ほどと同様に45度の回転Rotation(A,45.0);と2倍の拡大を実行してみてください.

Shin Yoshizawa: shin@riken.jp

重要！演習: 実際に逆変換をしてみる！

- Ex02.zipの中は、カラー画像(ppm)をアフィン変換するプログラムです.
  - ex02_1.cxx: 順変換.
  - ex02_2.cxx: 逆変換(補間なし)
  - ex02_3.cxx: 逆変換 (補間あり)
  - 中を見てそれぞれ動かしてみましょう！
  - レポートではpgm, ppm両方について色々なアフィン変換を実行した結果とプログラムを提出してもらうので、今日作ったプログラムと↑のプログラムは動かせる様にしておいてください.
- 以上で第2回演習は終了です.

Shin Yoshizawa: shin@riken.jp

レポートについて

- 中身と提出方法は4/27の授業で説明します.
- 内容はアフィン変換で今日作ったプログラムとex02_3.cxxを少し改造すれば作れます.
- プログラムの他にPDFでレポートを出して頂きます: 作ったプログラムの実験結果や考察.
- レポートで使う画像はみなさんデジカメや携帯カメラで撮ったオリジナルの画像を使ってください.
- どうしても学校のPCへ画像の取り込み方がわからない人はWEBで探した画像でもOK.

Shin Yoshizawa: shin@riken.jp

レポート:補足

- 他の画像フォーマットからppmやpgmにするには、端末で

「convert −quality 100 −depth 8 −compress none AAAAA BBBB」

とする. ここでAAAA, BBBBはそれぞれ入力画像ファイル名と出力画像ファイル名. 例えば、ABC.bmpというファイル名のBMP画像を

  - pgmにする場合は:

「convert −quality 100 −depth 8 −compress none ABC.bmp ABC.pgm」.

  - ppmにする場合は:

「convert −quality 100 −depth 8 −compress none ABC.bmp ABC.ppm」.

- 逆にレポート用にMSワード等で読めるbmpファイルにするには

「convert −quality 100 −depth 8 −compress none ABC.ppm ABC.bmp」

Shin Yoshizawa: shin@riken.jp

講義・演習内容

シラバス

1回	画像とは何か。画像の構成要素、画像の表現
2回	画像とは何か。画像の構成要素、画像の表現
3回	画像とは何か。画像の構成要素、画像の表現

} 基礎

~~第1・2回：画像処理の世界~~
~~第3回：アフィン変換と画素値の補間~~
第4回：画像化・画像処理法・色彩＋
第3回演習の続き・第1回レポート.