

Graphics with Processing



2010-06 座標変換と同次座標系

<http://vilab.org>

塩澤秀和

6.1 幾何変換 (p.16)

幾何変換とアフィン変換

□ 座標変換

$$\begin{pmatrix} x \\ y \end{pmatrix} \xrightarrow{f} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

□ 幾何変換 (幾何学的変換)

- 平行移動
- 拡大・縮小
- 回転

□ アフィン変換

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

幾何変換はどんな組合せでも
アフィン変換で表現することができる

幾何変換関数

□ translate(x_0, y_0)

- 描画座標系を平行移動
- x軸方向に x_0 移動
- y軸方向に y_0 移動
- Processingではy軸は下向き

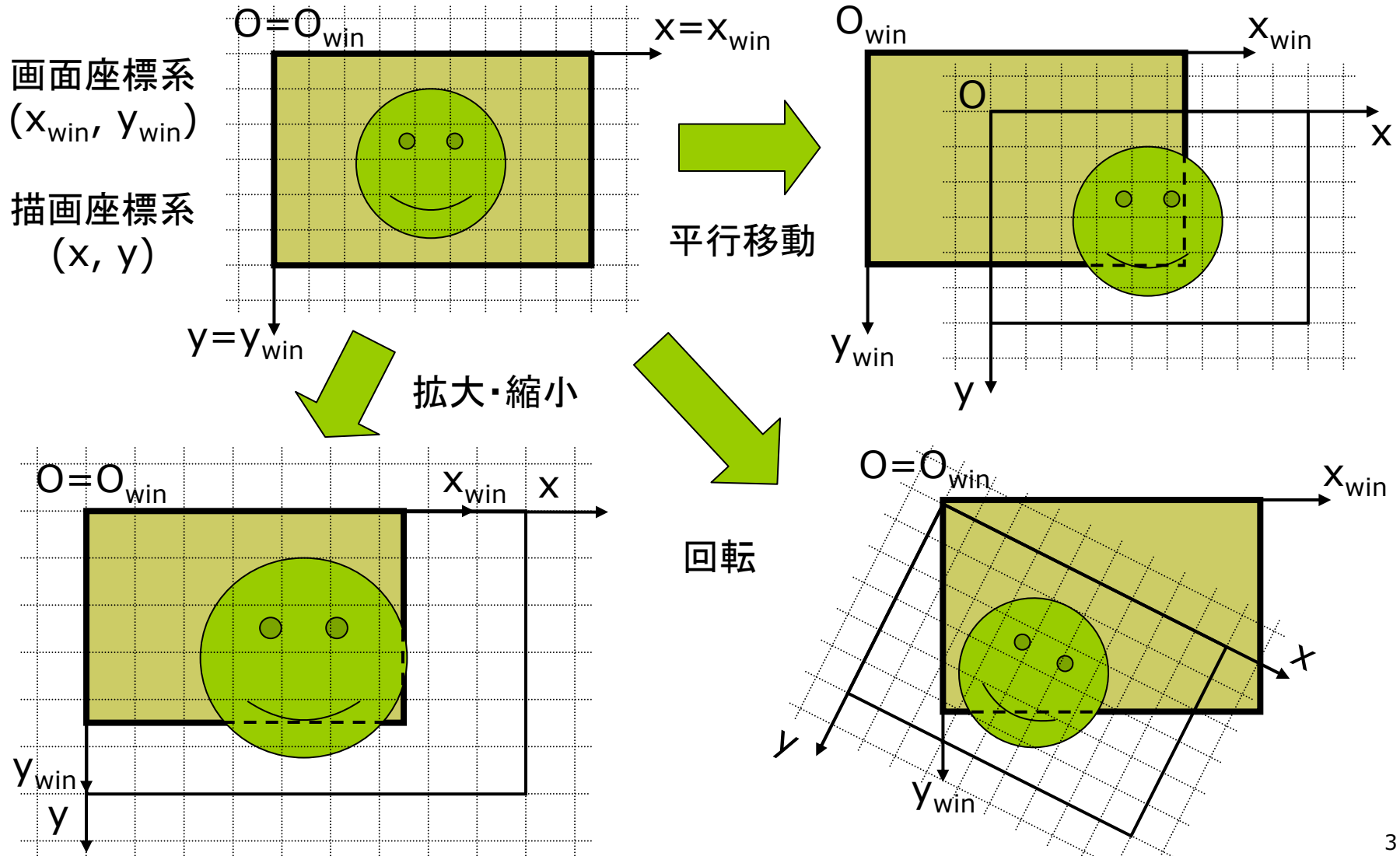
□ scale(α, β)

- 描画座標系を拡大・縮小
- x軸方向(左右)に α 倍
- y軸方向(上下)に β 倍
- 原点が中心に全体が拡大

□ rotate(θ)

- 描画座標系を回転
- 原点中心に θ 回転
- Processingで+方向は時計回り

6.2 幾何変換の効果



6.3 幾何変換の数学表現

数式による表現

- 描画座標系から画面座標系へ

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} x' \\ y' \end{pmatrix} \rightarrow \dots \rightarrow \begin{pmatrix} x_{win} \\ y_{win} \end{pmatrix}$$

- 平行移動と拡大・縮小

$$x' = x + x_0 \quad x' = \alpha x$$

$$y' = y + y_0 \quad y' = \beta y$$

- 回転

$$x' = x \cos \theta - y \sin \theta$$

$$y' = x \sin \theta + y \cos \theta$$

ベクトルと行列による表現

- 平行移動

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

- 拡大・縮小

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- 回転

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

6.4 同次座標系 (p.19)

同次座標系

- 座標計算をしやすい表現

2次元直交座標 2次元同次座標

$$(x, y) \Leftrightarrow (x, y, 1)$$

同じ座標

- 同次座標系による変換行列

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & e \\ c & d & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

行列1つですべてのアフィン変換を表せる

同次座標系による表現

- 平行移動

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

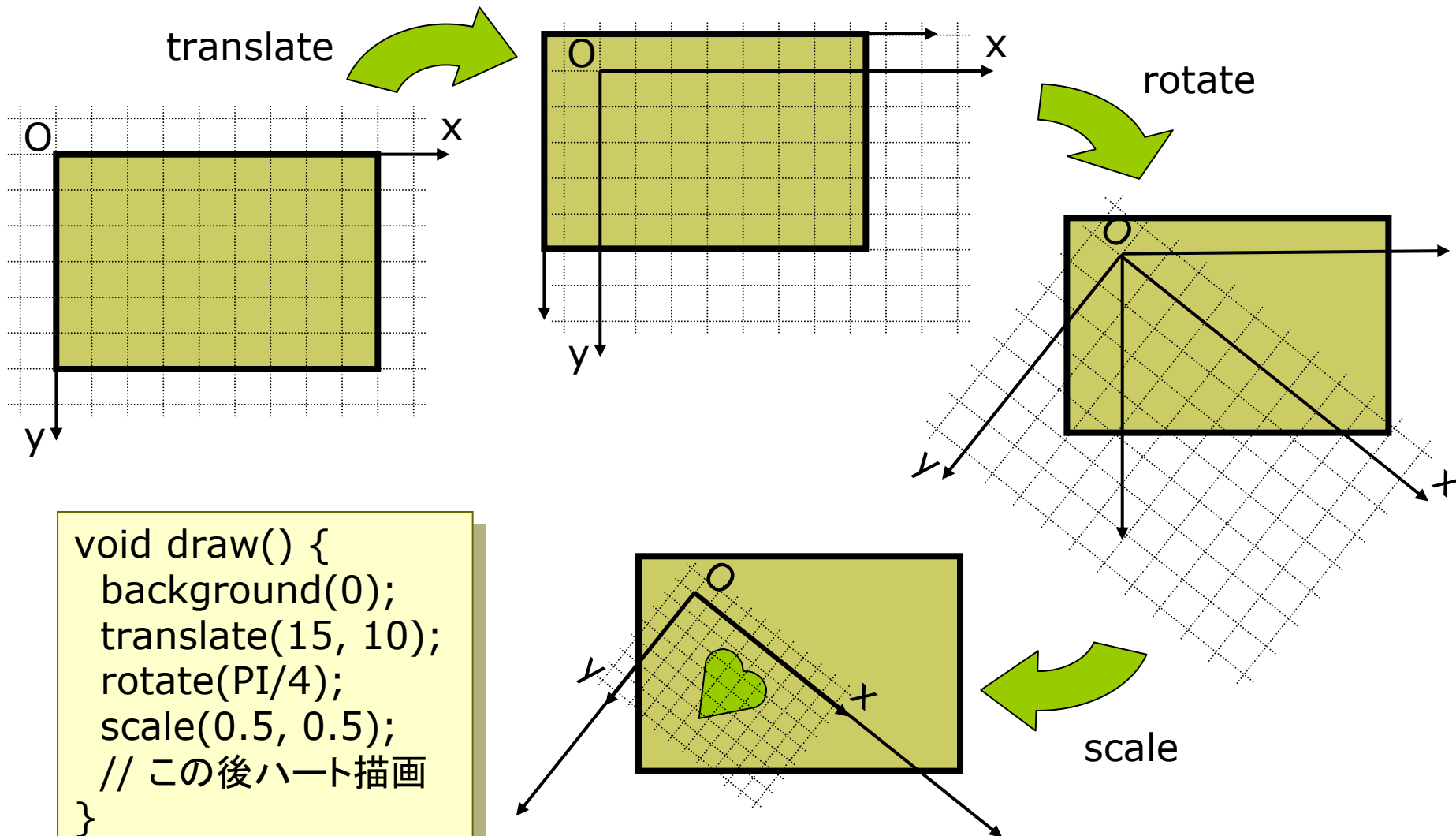
- 拡大縮小

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- 回転

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

6.5 幾何変換の合成 (p.22)



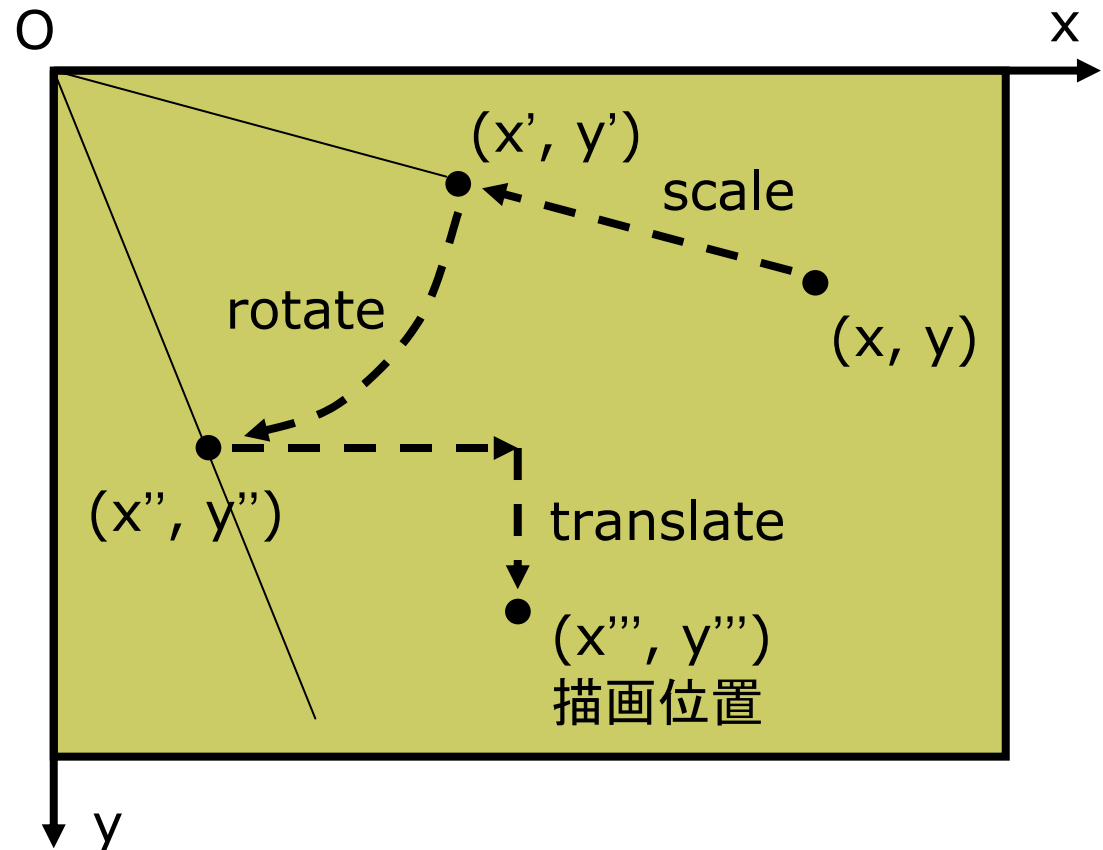
6.6 図形移動での考え方 (p.23)

座標変換の別の解釈

- 座標系の移動ではなく、同じ画面座標系上での図形の移動として考えることもできる
- その場合、描画からさかのぼって、図形に命令の逆順で変換を作用させる
- 数学的に同じこと
= 結果はどちらも同じ
- 右図の例

```
translate(15, 10);
rotate(PI/4);
scale(0.5, 0.5);
point(x, y);
```

↑
逆順



6.7 合成変換行列 (p.22)

合成変換の数学表現

- 同次変換行列の積になる

$$P_{win} = M_1 M_2 M_3 \cdots M_n P$$

$$M = M_1 M_2 M_3 \cdots M_n$$

- 右上の例の行列表現

$$\begin{bmatrix} x_{win} \\ y_{win} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 15 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\pi/4) & -\sin(\pi/4) & 0 \\ \sin(\pi/4) & \cos(\pi/4) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

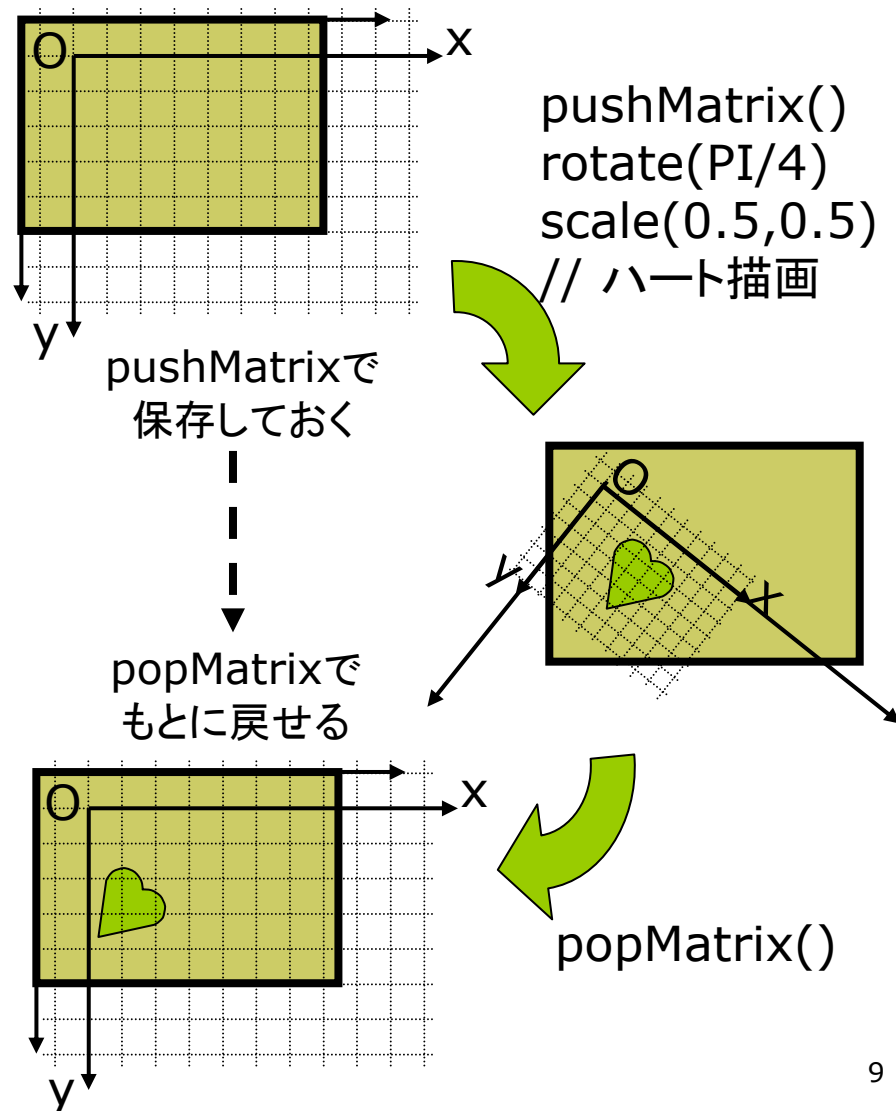
$$\begin{bmatrix} x_{win} \\ y_{win} \\ 1 \end{bmatrix} = \begin{bmatrix} \sqrt{2}/4 & -\sqrt{2}/4 & 15 \\ \sqrt{2}/4 & \sqrt{2}/4 & 10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \therefore M = \begin{bmatrix} \sqrt{2}/4 & -\sqrt{2}/4 & 15 \\ \sqrt{2}/4 & \sqrt{2}/4 & 10 \\ 0 & 0 & 1 \end{bmatrix}$$

```
void draw() {
    background(0);
    translate(15, 10); // 変換 M1
    rotate(PI/4);     // 変換 M2
    scale(0.5, 0.5);  // 変換 M3
    // 図形描画...
}
```


6.8 変換行列の操作 (p.45)

行列スタックの操作

- システム変換行列
 - 現在の描画座標系を示す行列
 - システム変換行列は幾何変換 (translate, rotate, scale) の処理のたびに合成されていく
- pushMatrix()
 - システム変換行列 (描画座標系) を一時待避する
- popMatrix()
 - 最近保存した変換行列を戻す
 - pushMatrix() と必ず対にする
- resetMatrix()
 - 変換行列をリセットする
 - 描画座標系 = 画面座標系の初期状態に戻す



6.9 幾何変換と行列操作の例

```
// 描画原点を移動する例
```

```
float bai = 1.0;
```

```
void setup() {  
    size(400, 400);  
    rectMode(CENTER);  
}
```

```
void draw() {  
    background(255);  
    translate(200, 200);  
    scale(bai);  
  
    rect(0, 0, 50, 50);  
    bai += 0.02;  
    if (bai > 8.0) bai = 1.0;  
}
```

```
// 行列のpushとpopの例
```

```
void setup() {  
    size(600, 400);  
    rectMode(CENTER);  
    noLoop();  
}
```

```
void draw() {  
    background(#8080e0);  
    pushMatrix();  
    translate(200, 200);  
    fill(#ffd0d0); rect(0,0, 50, 50);  
    popMatrix();  
    pushMatrix();  
    translate(400, 200);  
    rotate(radians(45));  
    fill(#ffff00); rect(0,0, 50, 50);  
    popMatrix();  
}
```

6.10 演習課題

課題

- 6.11のプログラムは、スマイリー(にこちゃんマーク)を2つ描画するものである

問1) 中心と外側の顔の描画位置を決めている合成変換行列($M_{\text{中心}}$ と $M_{\text{外側}}$)の両方を求めなさい

- $M_{\text{中心}}$ は右のヒント参照
- 次回**A4レポート用紙**で提出

問2) このプログラムに幾何変換の関数を2つ加えて、外側の顔の大きさを半分にして、顔の向きは回転しないようにしなさい

- ただし、顔を描画する関数は、変更したり追加したりしないこと
- プログラム(.pde)をWeb提出

問1の $M_{\text{中心}}$ のヒント

- $M_{\text{中心}}$ は次の2つの変換の合成
 - $M_1 = \text{translate}(200, 200)$
 - $M_2 = \text{rotate}(-a)$
- それぞれの行列表現は

$$M_1 = \begin{bmatrix} 1 & 0 & 200 \\ 0 & 1 & 200 \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_2 = \begin{bmatrix} \cos(-a) & -\sin(-a) & 0 \\ \sin(-a) & \cos(-a) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- $M_{\text{中心}}$ はこの2つの合成なので

$$M = \begin{bmatrix} 1 & 0 & 200 \\ 0 & 1 & 200 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(-a) & -\sin(-a) & 0 \\ \sin(-a) & \cos(-a) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

6.11 演習課題 (続き)

```
void setup() {
  size(400, 400);
  frameRate(30);
}

void draw_smiley() {
  ellipseMode(CENTER);
  strokeWeight(3);
  stroke(0); fill(#ffff00);
  ellipse(0, 0, 100, 100);
  noStroke(); fill(0);
  ellipse(-15, -15, 12, 12);
  ellipse( 15, -15, 12, 12);
  stroke(#ff0000); noFill();
  bezier(-25, 20, -10, 35,
         10, 35, 25, 20);
}
```

```
void draw() {
  float a = radians(frameCount);
  background(255);
  translate(200, 200); // 原点移動
  // ★
  pushMatrix();
  rotate(-a);
  draw_smiley();
  popMatrix();
  // ★
  pushMatrix();
  rotate(-a);
  translate(130, 0);
  // ここに2つ幾何変換を追加する
  draw_smiley();
  popMatrix();
  // ★
}
```

★のところ
の座標系は
同じになる