エルゴノミクスコンピューティング実習

02 基礎知識 - 2D座標変換

人間システム工学科 井村 誠孝
m.imura@kwansei.ac.jp

本節の内容

- 座標変換を適切に用いて描画する練習をする.
- 3次元空間の位置姿勢情報を適切に扱う下準備となる.

サンプル: ウィンドウを開くだけ

```
void setup() {
  size(512, 512);
  frameRate(60);
}

void draw() {
  background(255);
}
```

関数 setup()
実行開始後に一度だけ実行

size(x, y)
ウィンドウサイズを指定

frameRate(f)
フレームレートを指定

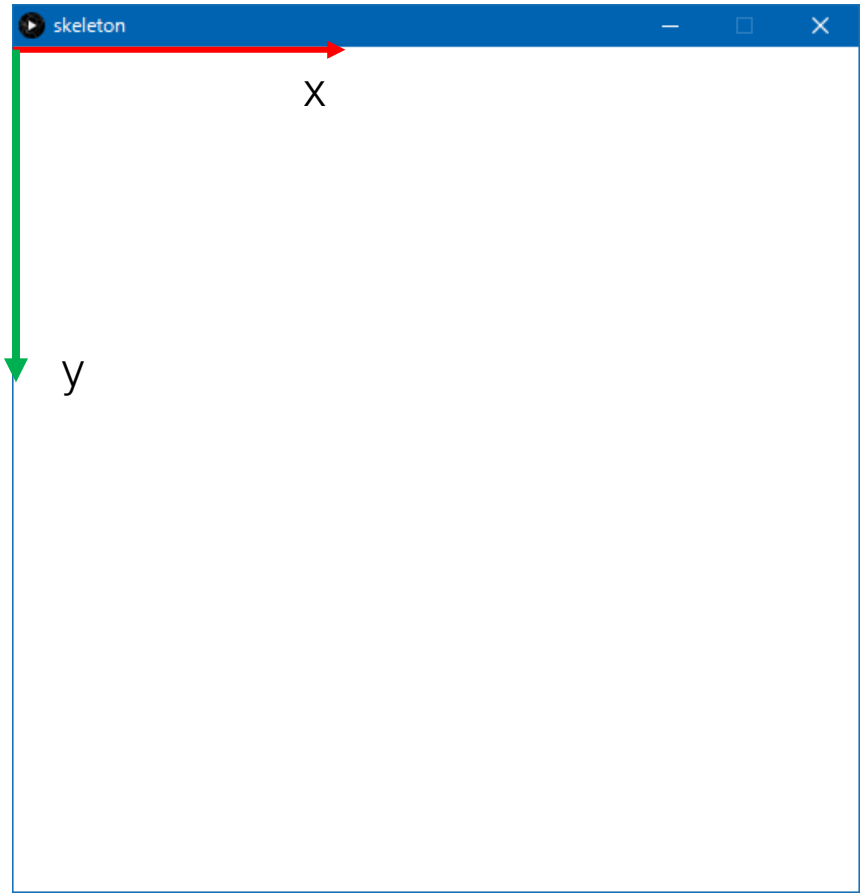
関数 draw()
くりかえし実行される

background(c)
背景を指定色でクリア

実行順:
関数の外側→setup()→draw()→draw()→draw()→

Processingの基本的な座標系

- ウィンドウの左上が原点
- 右方向がx座標 正，下方向がy座標 正



ウィンドウの中央に，矩形を描く

```
void setup() {
  size(512, 512);
  frameRate(60);
}

void draw() {
  background(255);

  int size = 200;
  stroke(128, 192, 255);
  fill(192, 255, 255);
  rect(width / 2 - size / 2, height / 2- size / 2, size, size);
}
```

stroke(c)
線の色を指定

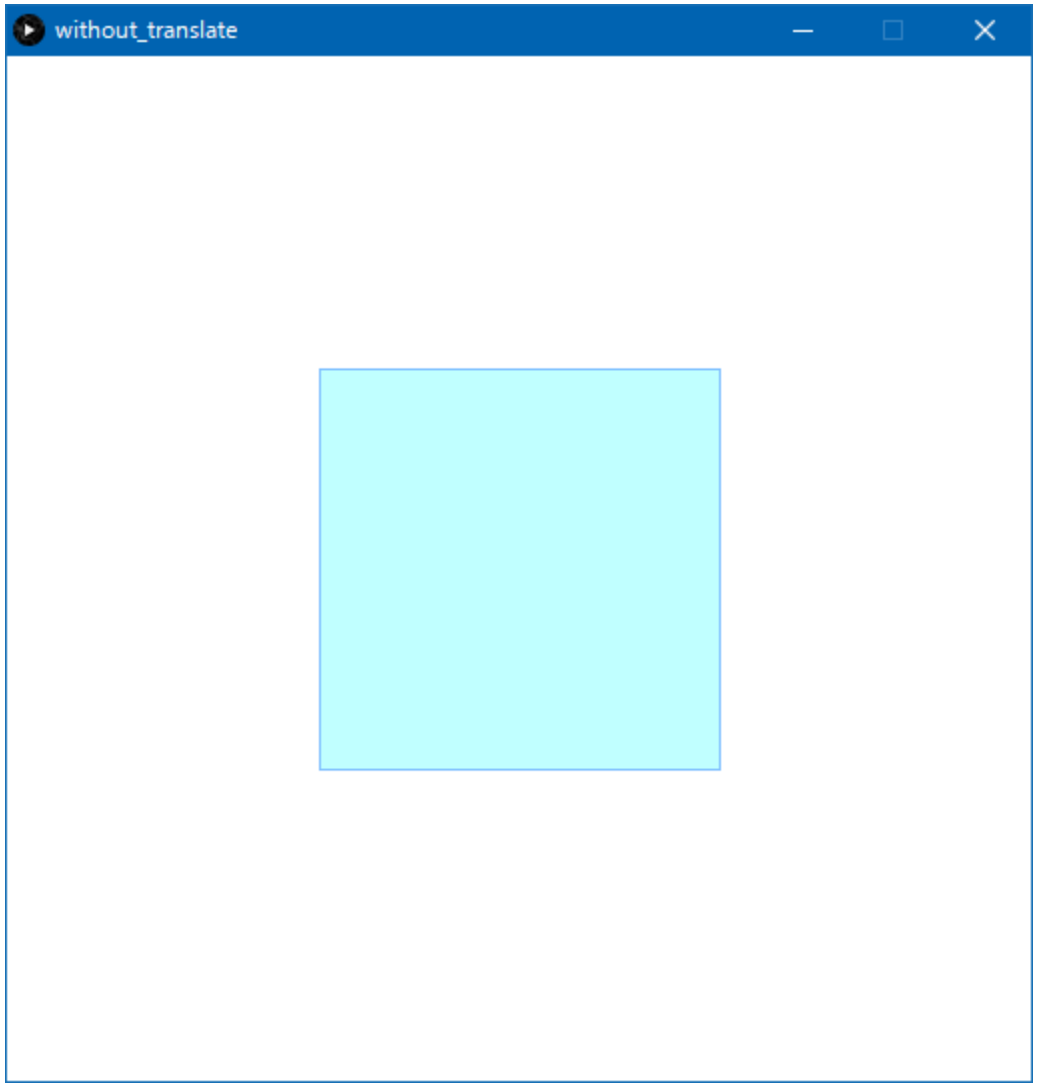
fill(c)
塗りの色を指定

width, height
ウィンドウのサイズ

rect(x, y, w, h)
矩形を描画

http://j.mp/kg-ergo-p

実行例



まとめ: 描画色の指定

- 数字1個: グレイスケール
- 数字3個: RGB
  - R: 赤
  - G: 緑
  - B: 青
- 数字4個: RGBA
  - A: 透明度 (アルファチャンネル)

- デフォルトでは0-255だが変更も可能
- HSB(HSV)にも切り替え可能

- 線の色: `stroke(c)`
  - 線を描きたくない場合は `noStroke()`
- 塗りの色: `fill(c)`
  - 塗り潰したくない場合は `noFill()`

座標変換を用いて描画する

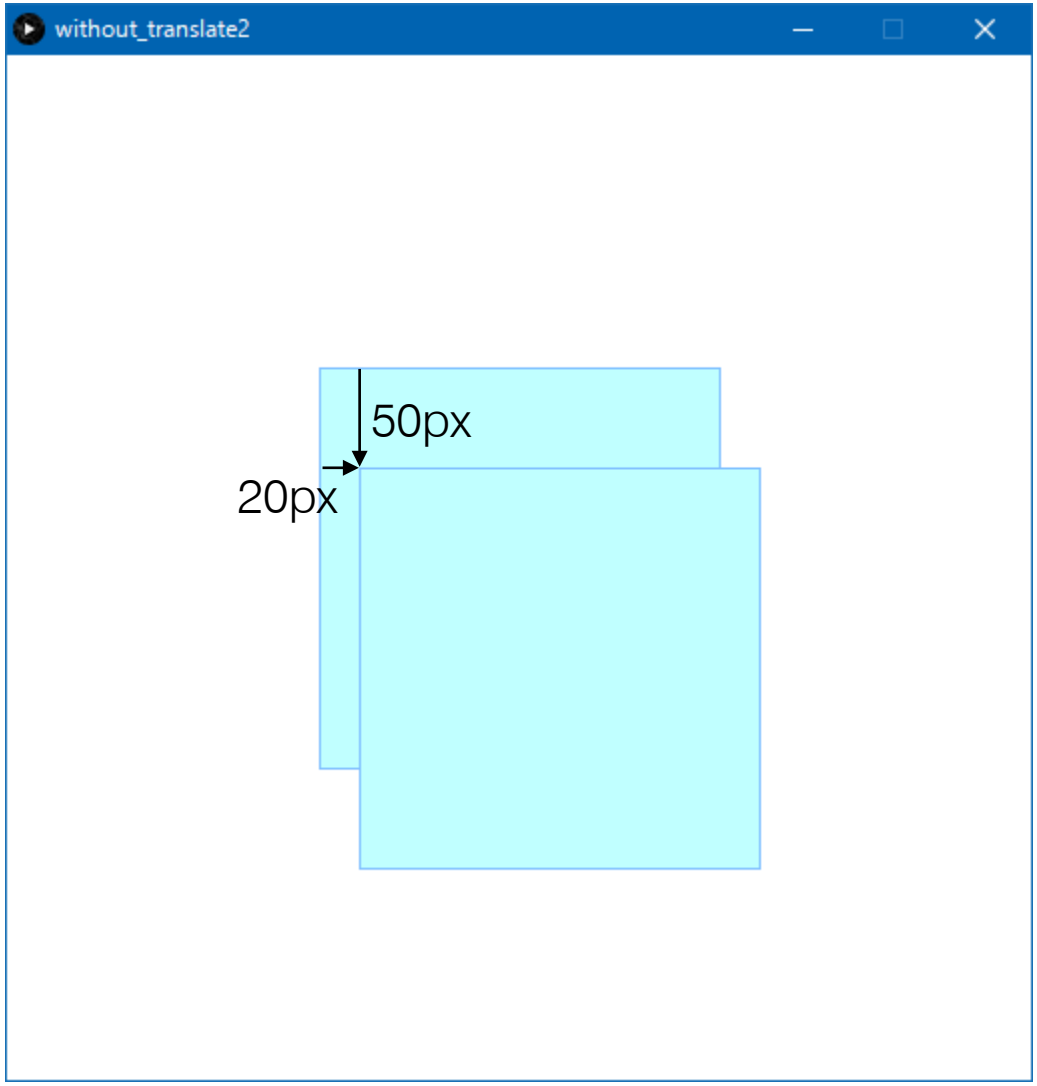
```
void setup() {
  size(512, 512);
  frameRate(60);
}

void draw() {
  background(255);

  int size = 200;
  stroke(128, 192, 255);
  fill(192, 255, 255);
  translate(width / 2 - size / 2, height / 2 - size / 2);
  rect(0, 0, size, size);
}
```

translate(x, y)
座標系を平行移動

「特に簡単になったようには，見えませんけど…」

http://j.mp/kg-ergo-p

ではこれはどうか



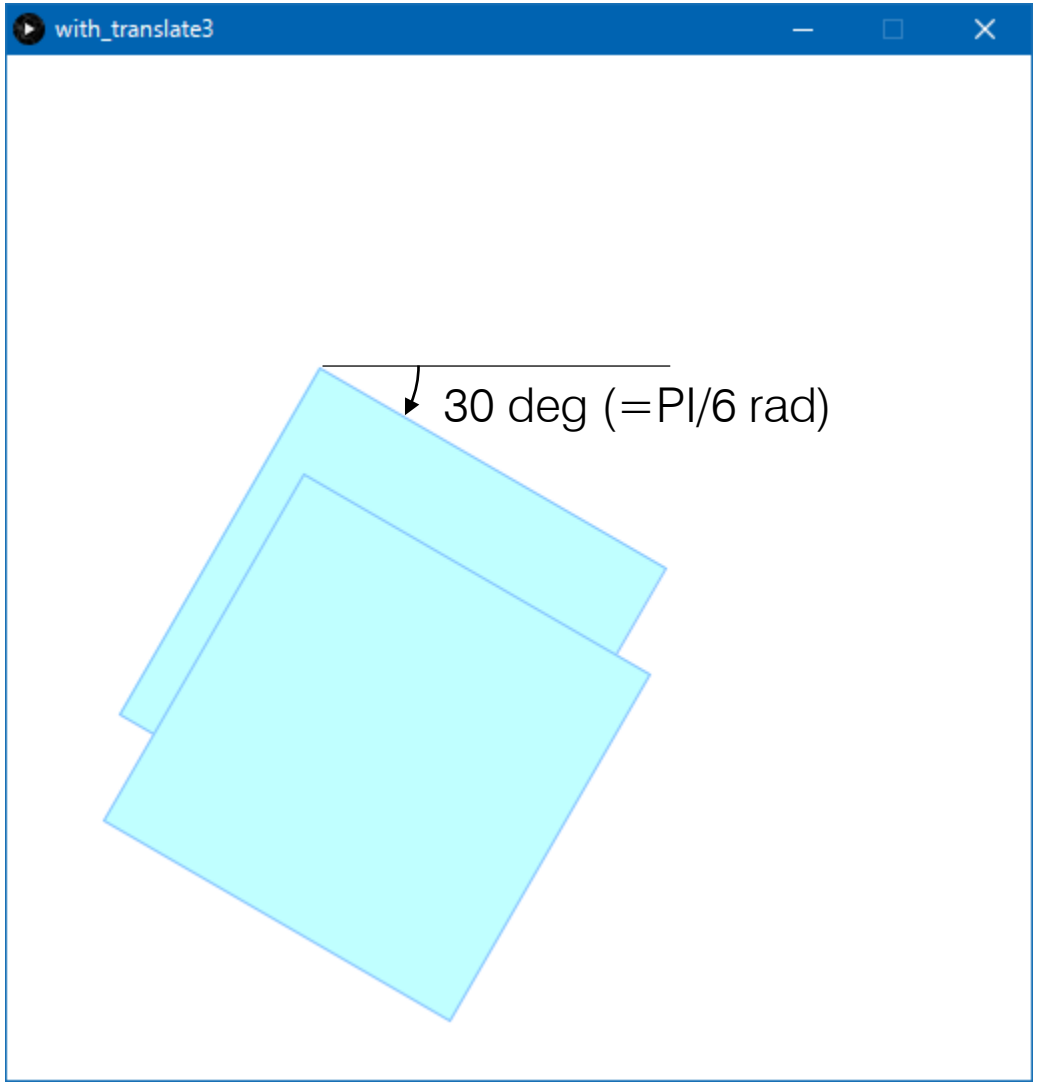
座標値を直接計算する方法だと...

```
void setup() {
  size(512, 512);
  frameRate(60);
}

void draw() {
  background(255);

  int size = 200;
  int dx = 20;
  int dy = 50;
  stroke(128, 192, 255);
  fill(192, 255, 255);
  rect(width / 2 - size / 2, height / 2- size / 2, size, size);
  rect(width / 2 - size / 2 + dx, height / 2- size / 2 + dy,
       size, size);
}
```

座標変換を用いるならば

```
void setup() {
  size(512, 512);
  frameRate(60);
}

void draw() {
  background(255);

  int size = 200;
  int dx = 20;
  int dy = 50;
  stroke(128, 192, 255);
  fill(192, 255, 255);
  translate(width / 2 - size / 2, height / 2 - size / 2);
  rect(0, 0, size, size);
  translate(dx, dy);
  rect(0, 0, size, size);
}
```

更にこうだとどうか



座標変換を用いて描画する3

```
void setup() { … }

void draw() {
  background(255);

  int size = 200;
  int dx = 20;
  int dy = 50;
  stroke(128, 192, 255);
  fill(192, 255, 255);
  translate(width / 2 - size / 2, height / 2 - size / 2);
  rotate(PI/6);
  rect(0, 0, size, size);
  translate(dx, dy);
  rect(0, 0, size, size);
}
```

PI
定数π(=3.14159265…)
QUARTER_PI, HALF_PI, TWO_PIも用意されている

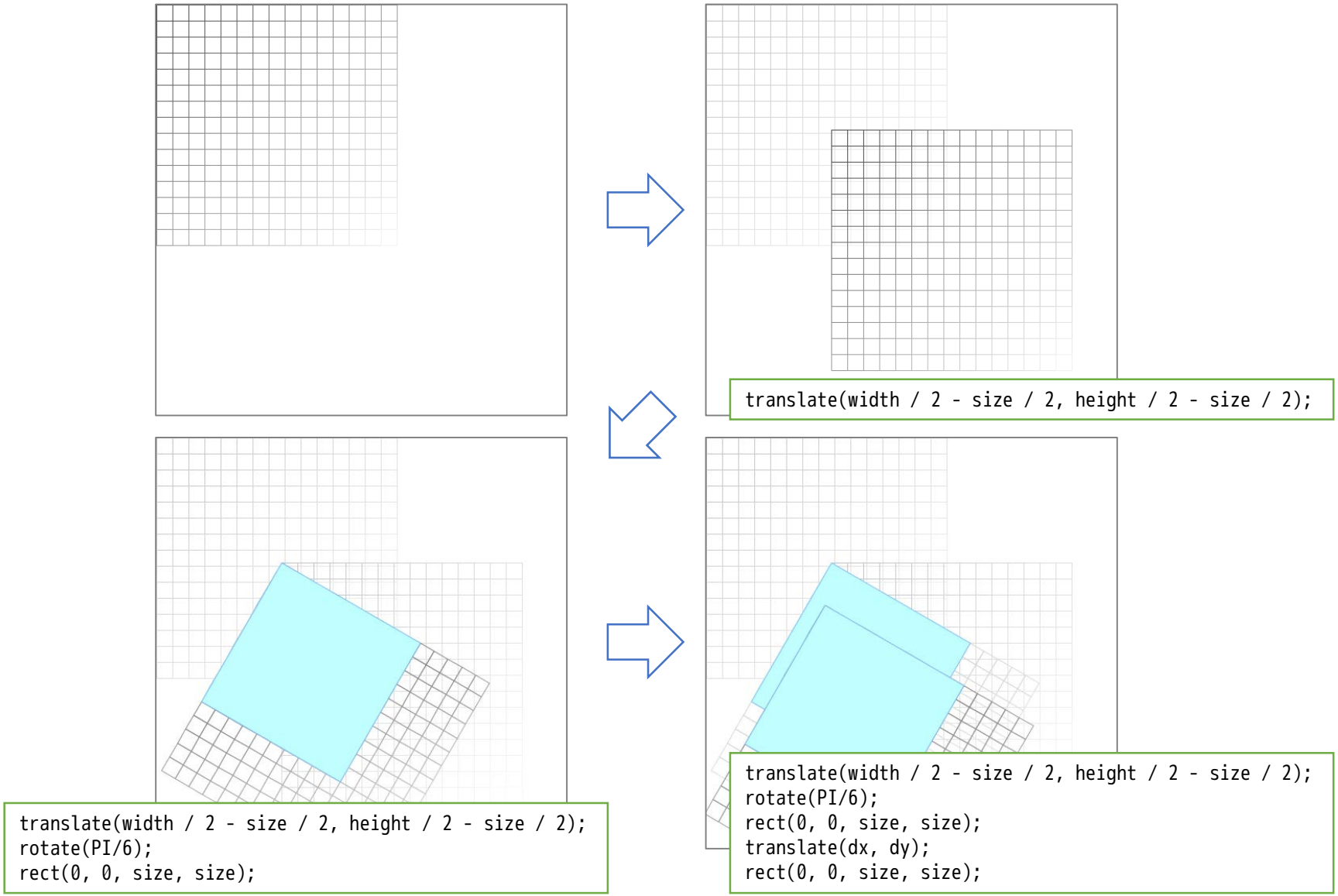
rotate(a)
座標系を回転
時計まわりが正
角度はラジアンで指定

座標変換

- translate() や rotate() は，座標系を移動させる変換であると考える．
- 各座標変換は，カレント座標系を基準にして適用される．
  - カレント座標系: 今，描画を実行したら，座標値はその座標系での値とみなされる座標系
- 座標系を適切に移動させることによって，座標値のややこしい計算を自前で行う必要が無くなる．

http://j.mp/kg-ergo-p

座標変換の過程を可視化



```
translate(width / 2 - size / 2, height / 2 - size / 2);
```

```
translate(width / 2 - size / 2, height / 2 - size / 2);
rotate(PI/6);
rect(0, 0, size, size);
```

```
translate(width / 2 - size / 2, height / 2 - size / 2);
rotate(PI/6);
rect(0, 0, size, size);
translate(dx, dy);
rect(0, 0, size, size);
```

内部の処理は…

- 指定された座標値(カレント座標系での値)を,
  ウィンドウ座標系での座標値に変換している.
- 変換は3x3行列で記述される(同次座標系).
  - `printMatrix()` で確認可能(上2行が表示される)

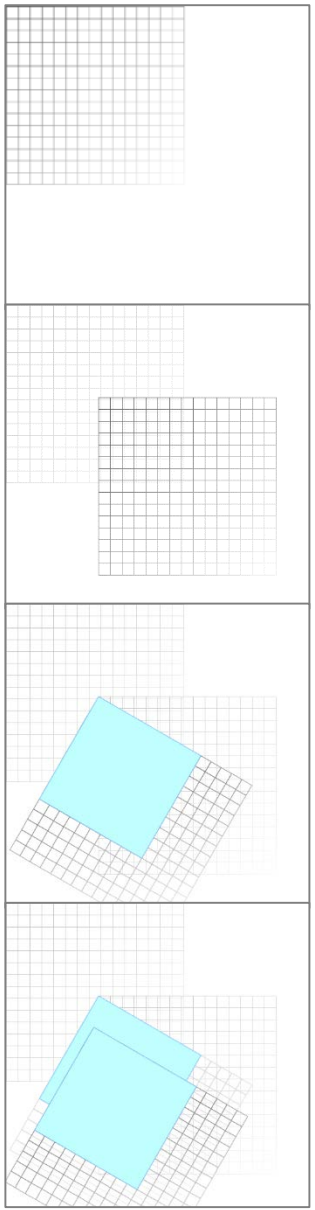
回転(とスケーリング)　平行移動

$$\begin{pmatrix} x_w \\ y_w \\ 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

ウィンドウ座標系　　カレント座標系

http://j.mp/kg-ergo-p

変換行列の具体的な値



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

```
translate(width / 2 - size / 2, height / 2 - size / 2);
```

$$\begin{pmatrix} 1 & 0 & 156 \\ 0 & 1 & 156 \\ 0 & 0 & 1 \end{pmatrix}$$ を右から乗じる

$$\begin{pmatrix} 1 & 0 & 156 \\ 0 & 1 & 156 \\ 0 & 0 & 1 \end{pmatrix}$$

```
rotate(PI/6);
```

$$\begin{pmatrix} 0.866 & -0.5 & 0 \\ 0.5 & 0.866 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$ を右から乗じる

$$\begin{pmatrix} 0.866 & -0.5 & 156 \\ 0.5 & 0.866 & 156 \\ 0 & 0 & 1 \end{pmatrix}$$

```
translate(dx, dy);
```

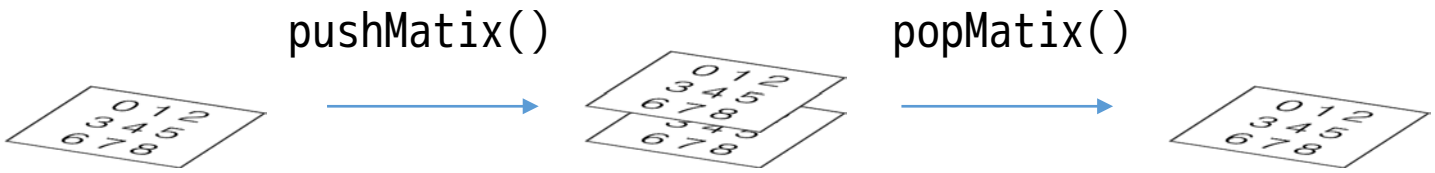
$$\begin{pmatrix} 1 & 0 & 20 \\ 0 & 1 & 50 \\ 0 & 0 & 1 \end{pmatrix}$$ を右から乗じる

$$\begin{pmatrix} 0.866 & -0.5 & 148.3 \\ 0.5 & 0.866 & 209.3 \\ 0 & 0 & 1 \end{pmatrix}$$

実行途中の変換行列を一時保存する

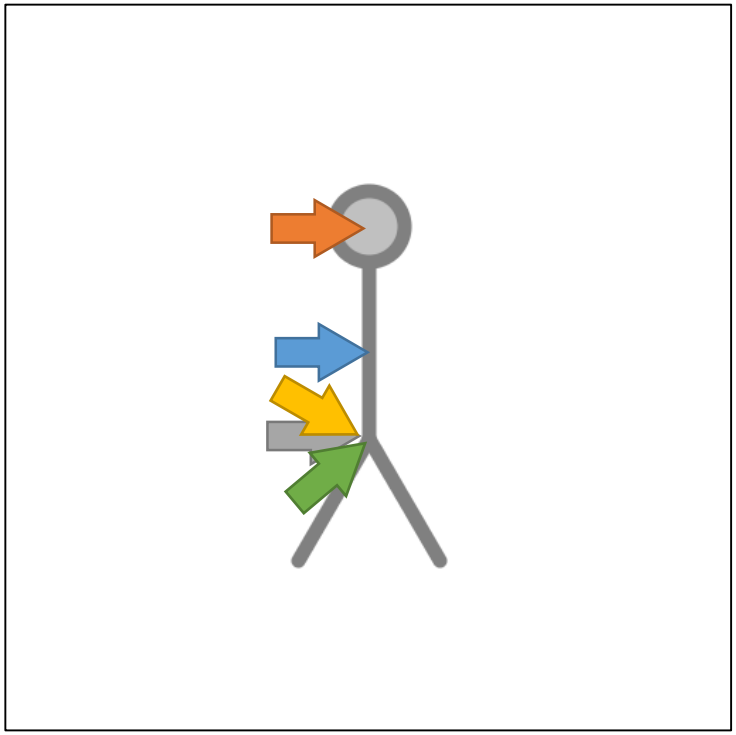
- pushMatix()で一時的にスタックに保存
- popMatrix()でスタックから取り出し

- スタックは知っていますか? 上からしか取り出せない箱のようなデータ構造.



一時保存すると何ができるのか

- リンク構造を持つ物体(例:人体)を容易に描画することができる.



```
translate(width / 2, height / 2); // body
line(0, -100, 0, 50);

pushMatrix();
translate(0, -100); // head
ellipse(0, 0, 50, 50);
popMatrix();

pushMatrix();
translate(0, 50);

pushMatrix();
rotate(PI / 6); // leg1
line(0, 0, 0, 100);
popMatrix();

pushMatrix();
rotate(-PI / 6); // leg2
line(0, 0, 0, 100);
popMatrix();

popMatrix();
```