

# Graphics with Processing



2010-08 モデルビュー変換

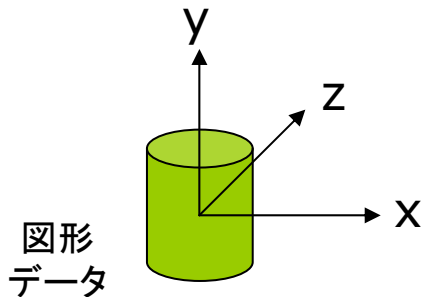
<http://vilab.org>

塩澤秀和

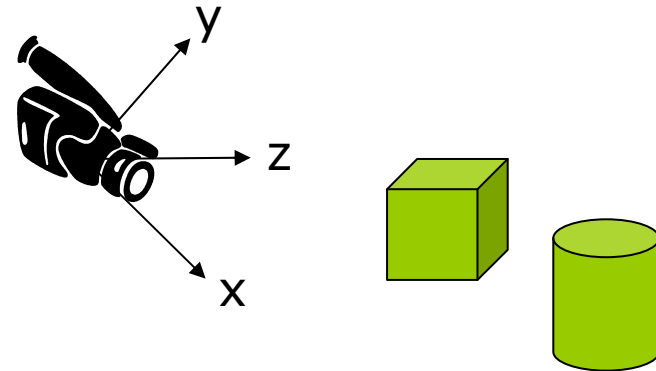
# 8.1 3DCGの座標系 (p.41)

## □ ローカル(モデリング)座標系

### ■ オブジェクトの座標系

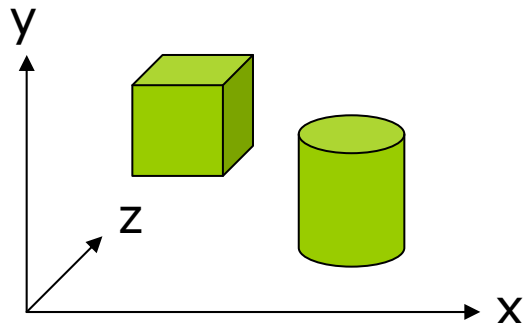


## □ 視点(カメラ)座標系

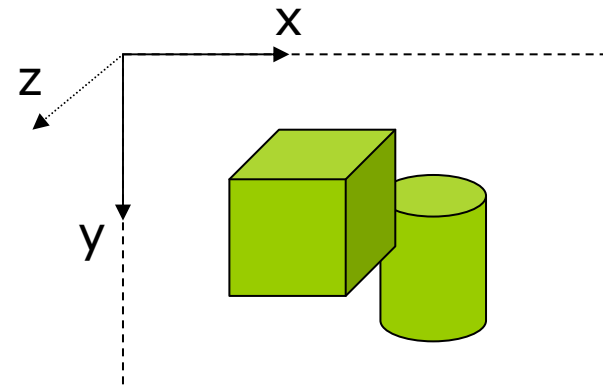


## □ ワールド座標系

### ■ 3次元世界の座標系

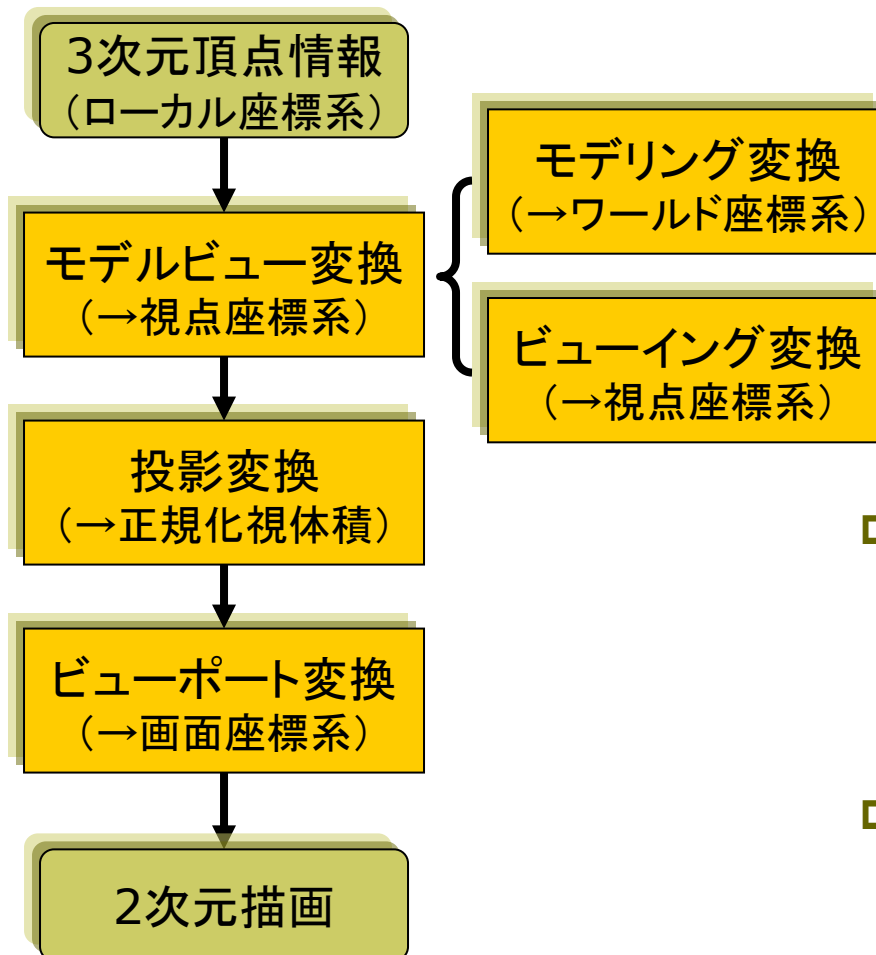


## □ 画面(デバイス)座標系



## 8.2 3DCGの座標変換(p.41)

### □ ビューイングパイプライン



### □ モデルビュー変換

- オブジェクト(図形・物体)と視点(カメラ)の位置関係の設定
- モデリング変換:  
オブジェクトの配置
- ビューイング変換(視野変換):  
視点の位置設定
- `translate()`, `scale()`,  
`rotate{X,Y,Z}()`, `camera()`

### □ 投影変換(次回)

- 投影面へ(正規化視体積へ)
- 平行投影: `ortho()`
- 透視投影: `perspective()`

### □ ビューポート変換

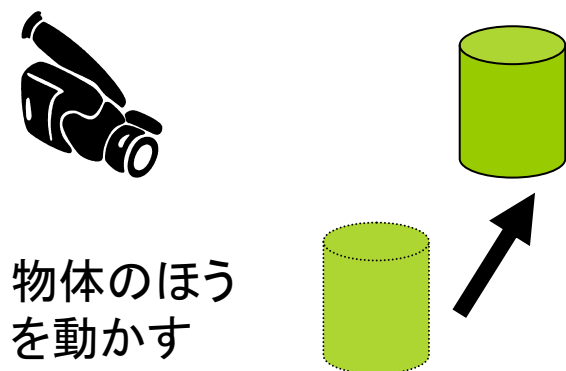
- 正規化視体積から画面座標へ(自動)

## 8.3 モデルビュー変換

### モデリング変換

#### □ オブジェクトの位置設定

- 目的: ワールド座標系に個々の3Dモデルを配置する
- 変換前: ローカル座標系
- 変換後: ワールド座標系

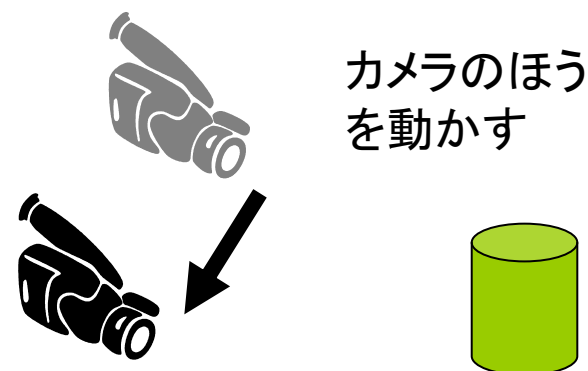


$$P_{world} = M_{model} P_{local}$$

### ビューイング変換(視野変換)

#### □ 視点(カメラ)の位置設定

- 目的: 投影計算のために, 座標の原点を視点に移動する
- 変換前: ワールド座標系
- 変換後: 視点座標系

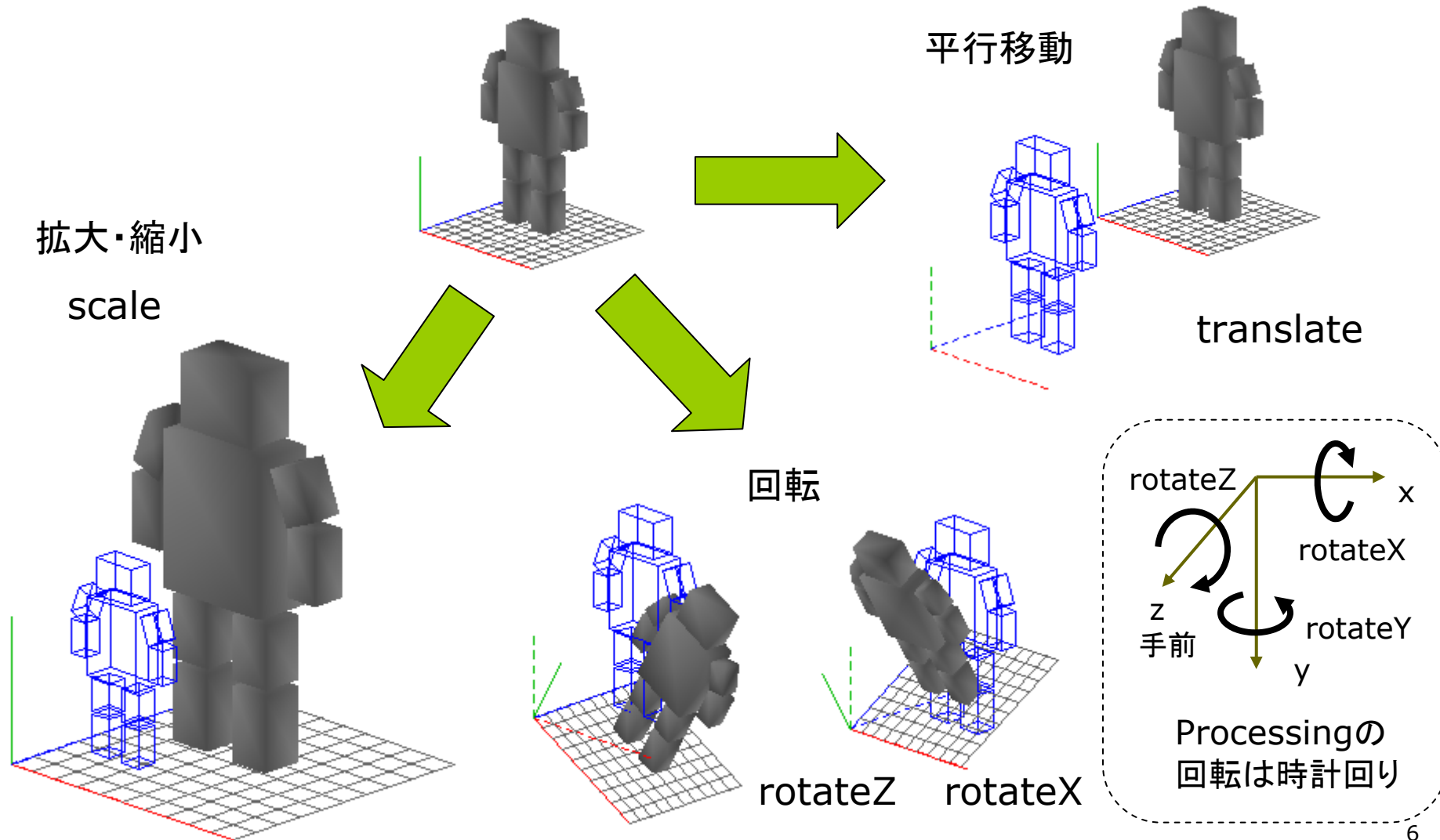


$$P_{view} = M_{view} P_{world}$$

数学的には同じ形式  
一連の「モデルビュー変換」

$$P_{view} = (M_{view} M_{model}) P_{local}$$

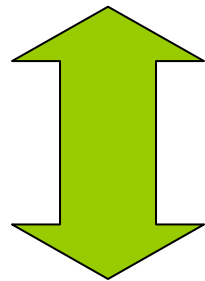
## 8.4 3次元幾何変換(p.28)



## 8.5 3次元同次座標 (p.28)

### 3次元同次座標

$$(x, y, z, w)$$



同次座標

直交座標

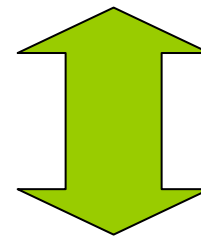
$$\left( \frac{x}{w}, \frac{y}{w}, \frac{z}{w} \right)$$

通常,  $w=1$  で用いる

$$(x, y, z)$$

### 3次元アフィン変換

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



同次座標による表現

直交座標による表現

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

## 8.6 3次元幾何変換(1)

### 3次元幾何変換

#### □ 平行移動

$$x' = x + t_x$$

$$y' = y + t_y$$

$$z' = z + t_z$$

#### □ 拡大・縮小

$$x' = s_x x$$

$$y' = s_y y$$

$$z' = s_z z$$

### 同次座標を用いた表現

#### □ 平行移動

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

#### □ 拡大・縮小

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

## 8.7 3次元幾何変換(2)

### □ z軸まわりの回転

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

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

$$z' = z$$

### □ x軸まわりの回転

$$x' = x$$

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

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

### □ y軸まわりの回転

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

$$y' = y$$

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

### □ z軸まわりの回転

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

### □ x軸まわりの回転

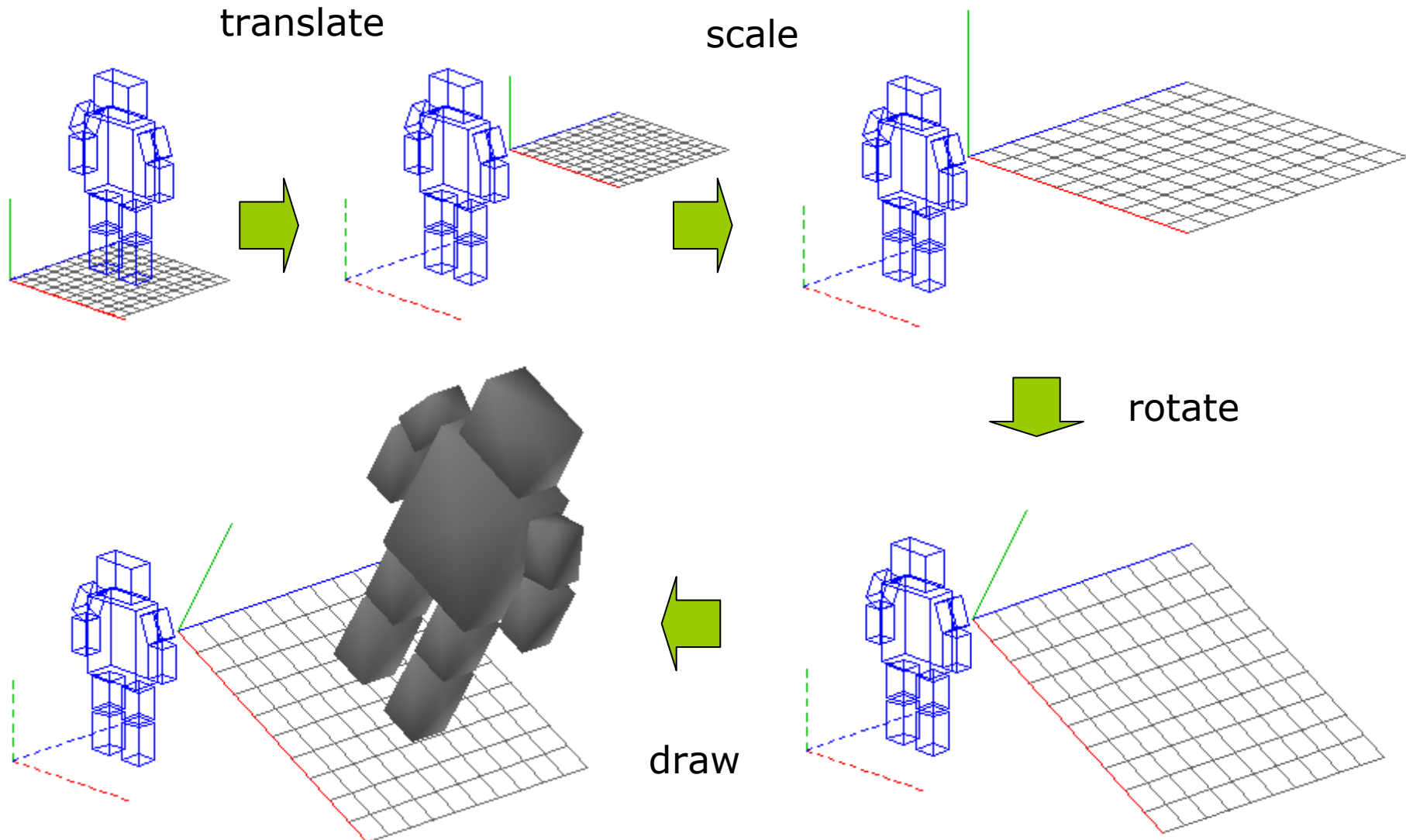
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

### □ y軸まわりの回転

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



## 8.8 幾何変換の合成 (p.31)



## 8.9 合成変換行列

### モデリング変換行列

#### □ 同次変換行列の積になる

$$P_{world} = M_1 M_2 M_3 \cdots M_n P_{local}$$

$$M_{model} = M_1 M_2 M_3 \cdots M_n$$

#### □ Processingコード

```
translate(0, 100, 300); // M1
scale(2, 2, 2); // M2
rotateZ(PI/6); // M3
// 図形描画...
```

$$\begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 100 \\ 0 & 0 & 1 & 300 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\pi/6) & -\sin(\pi/6) & 0 & 0 \\ \sin(\pi/6) & \cos(\pi/6) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{local} \\ y_{local} \\ z_{local} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \\ 1 \end{bmatrix} = \begin{bmatrix} \sqrt{3} & -1 & 0 & 0 \\ 1 & \sqrt{3} & 0 & 100 \\ 0 & 0 & 2 & 300 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{local} \\ y_{local} \\ z_{local} \\ 1 \end{bmatrix}$$

$$\therefore M_{model} = \begin{bmatrix} \sqrt{3} & -1 & 0 & 0 \\ 1 & \sqrt{3} & 0 & 100 \\ 0 & 0 & 2 & 300 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## 8.10 変換行列の操作 (p.45)

### 複数オブジェクトの配置

### 行列スタックの操作

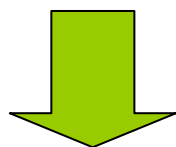
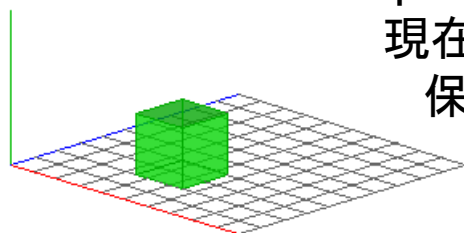
#### □ pushMatrix()

- システム変換行列(座標系)を一時的に退避する
- 階層モデリングに利用される

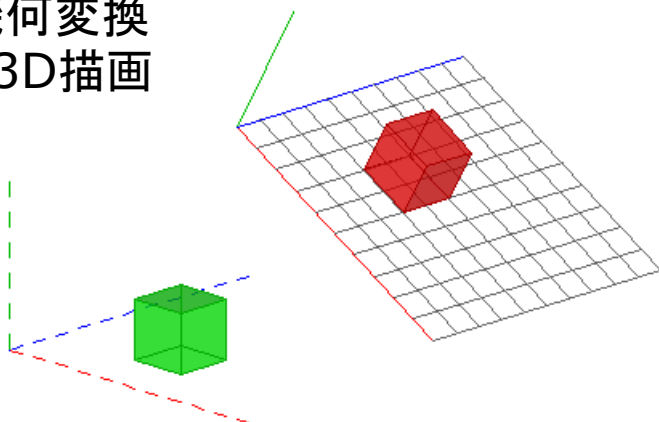
#### □ popMatrix()

- 最近保存した変換行列を戻す
- pushとpopは必ず対にすること

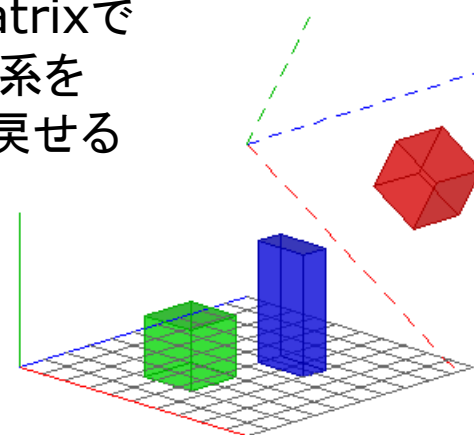
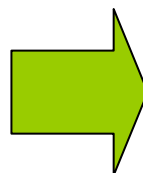
pushMatrixで  
現在の座標系を  
保存しておく



幾何変換  
と3D描画



popMatrixで  
座標系を  
もとに戻せる



## 8.11 ビューイング変換(p.42)

### ビューイング変換(視野変換)

#### □ 視点と視線の設定

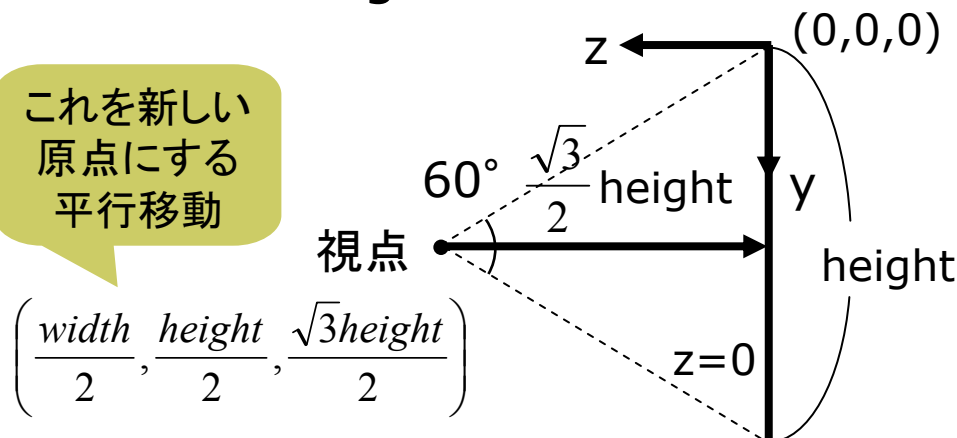
- 全座標を視点を基準にした位置に変換(視点が新しい原点)
- ワールド座標系 → 視点座標系

$$P_{view} = M_{view} P_{world}$$

- 全世界を動かすモデリング変換と同じと考えてもいい

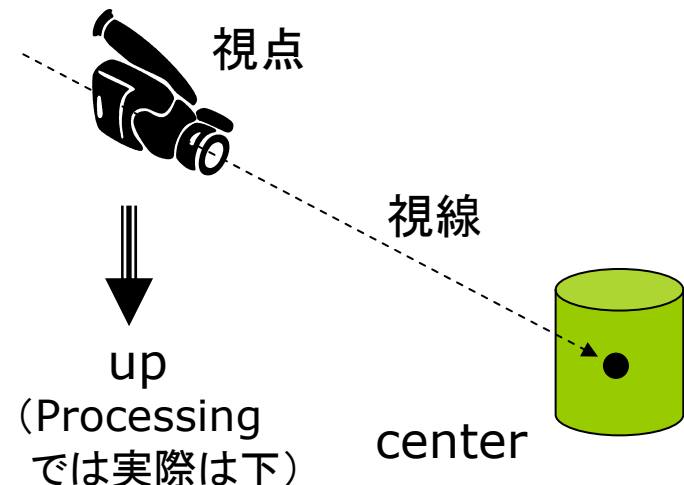
#### □ Processingのデフォルト

これを新しい  
原点にする  
平行移動



### 視点設定関数

- `camera(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ)`
  - eye: カメラ(視点)の座標
  - center: カメラで狙う座標
  - up: 上下方向を示すベクトル
  - **モデリング変換より前に書くこと**



## 8.12 演習課題

### 課題

問1) 右のプログラムのビューイング変換を行列 ( $M_{\text{view}}$ ) で示しなさい

※ デフォルトはビューイング変換は8.11の図を参照

- さらに, ★の箇所でのモデリング変換 ( $M_{\text{model}}$ ) を求めなさい
- 両者を合成してモデルビュー変換  $M_{\text{view}} M_{\text{model}}$  を求め, boxの座標を視点座標系に変換しなさい
- 次回, **A4レポート用紙**で提出

問2) 複数の図形(boxなど)を使って立体の文字やマークを組み立てるプログラムを作成しなさい

- 棒1本で「I」などはダメ
- 回転など動きをつけるとよい

```
// 「I」と書く例
void setup() {
  size(400, 400, P3D);
  noLoop();
}

void draw() {
  background(0);
  lights();
  perspective();
  translate(width/2, height/2, 0);
  rotateY(PI/6);
  fill(#ff5050); noStroke();
  pushMatrix();
    translate(-10, -30, 0);
    rotateZ(-PI/4);
    box(200, 50, 50);
  popMatrix();
  pushMatrix();
    translate(0, 50, 0);
    box(50, 150, 50); // ← ★
  popMatrix();
}
```