

# Graphics with Processing

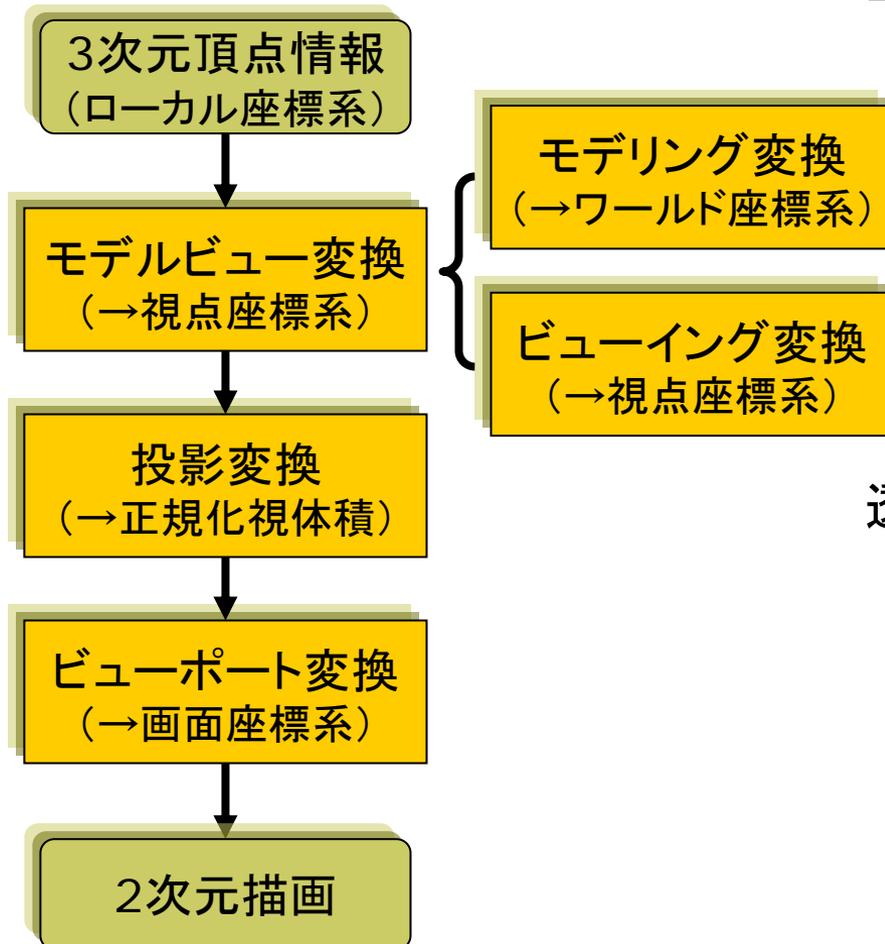


2007-09 投影変換とポリゴン描画

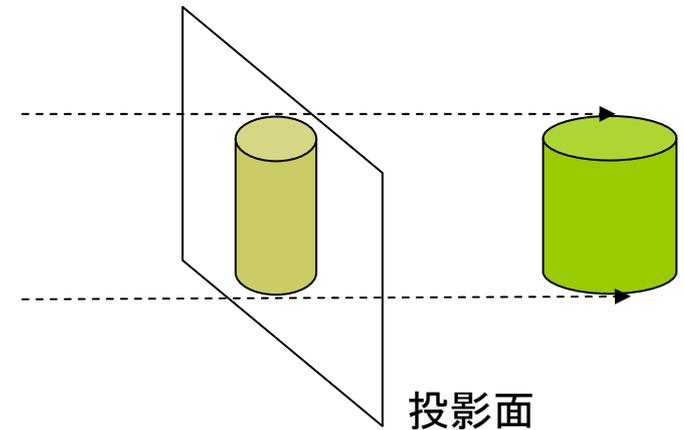
<http://vilab.org>

塩澤秀和

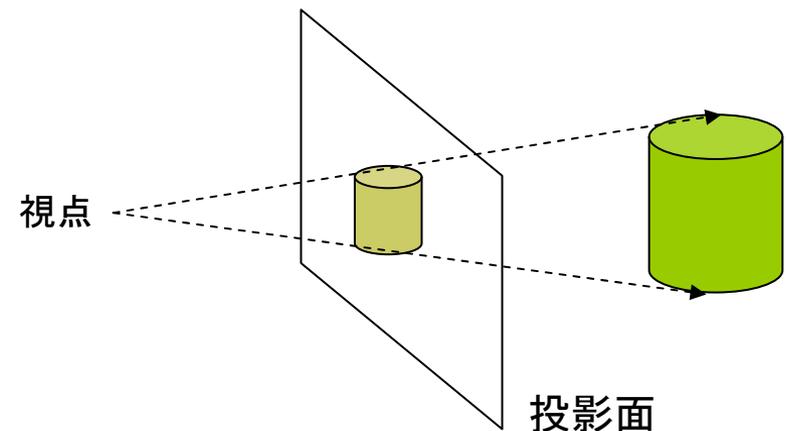
# 9.1 投影変換



## 平行投影



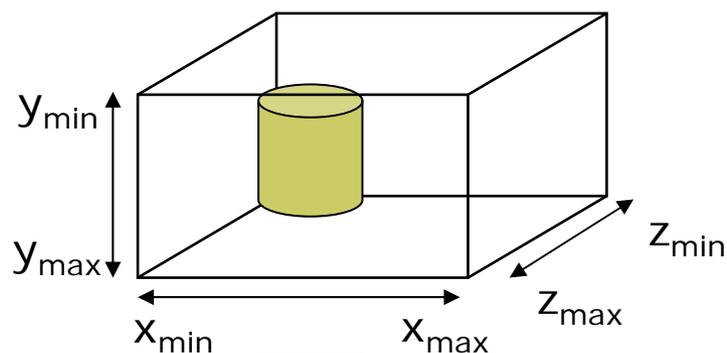
## 透視投影



## 9.2 平行投影とビューポート変換

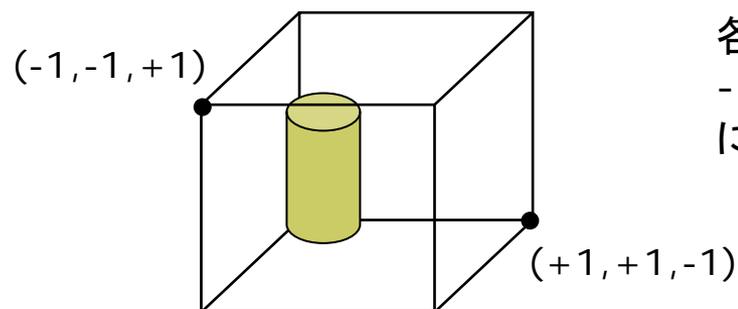
### 平行投影

- 視体積＝「見える領域」



平行投影の  
視体積は直方体

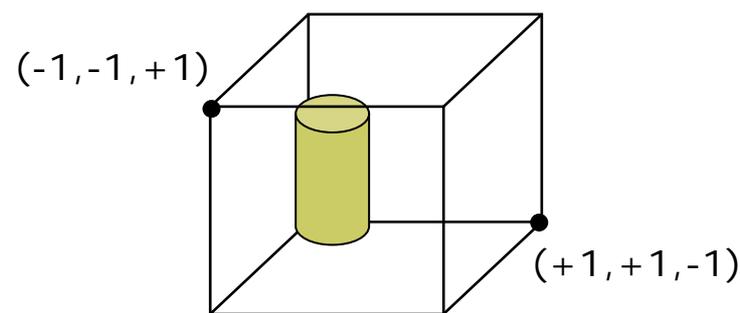
- 正規化視体積



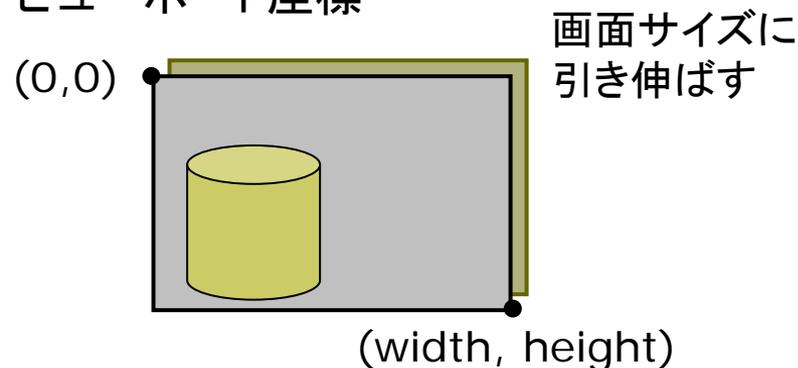
各座標を  
-1～+1  
に正規化

### ビューポート変換

- 正規化視体積



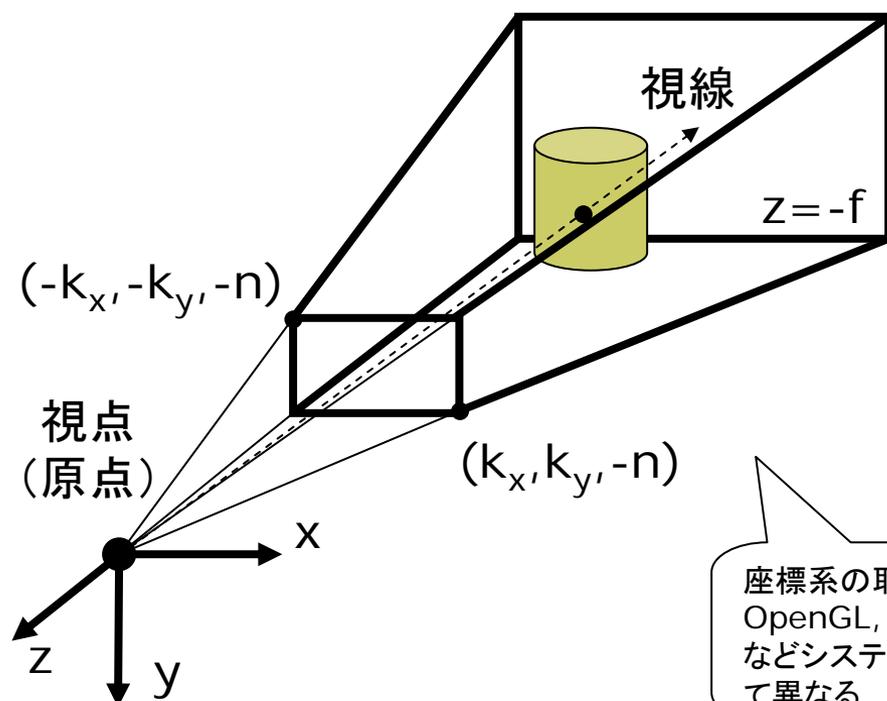
- ビューポート座標



# 9.3 透視投影

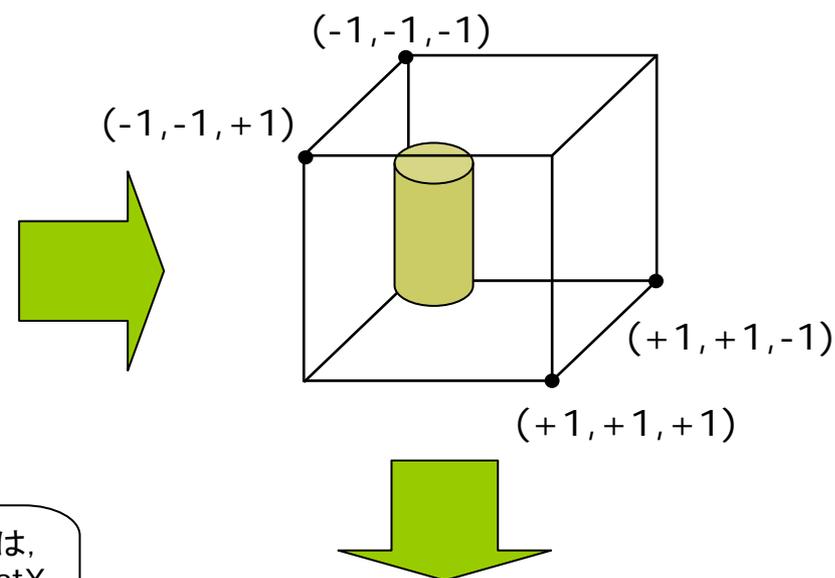
## 視体積

- 透視投影の視体積は四角錐台
  - Processingでは y軸が下向き
  - 視線は -z方向



## 正規化視体積

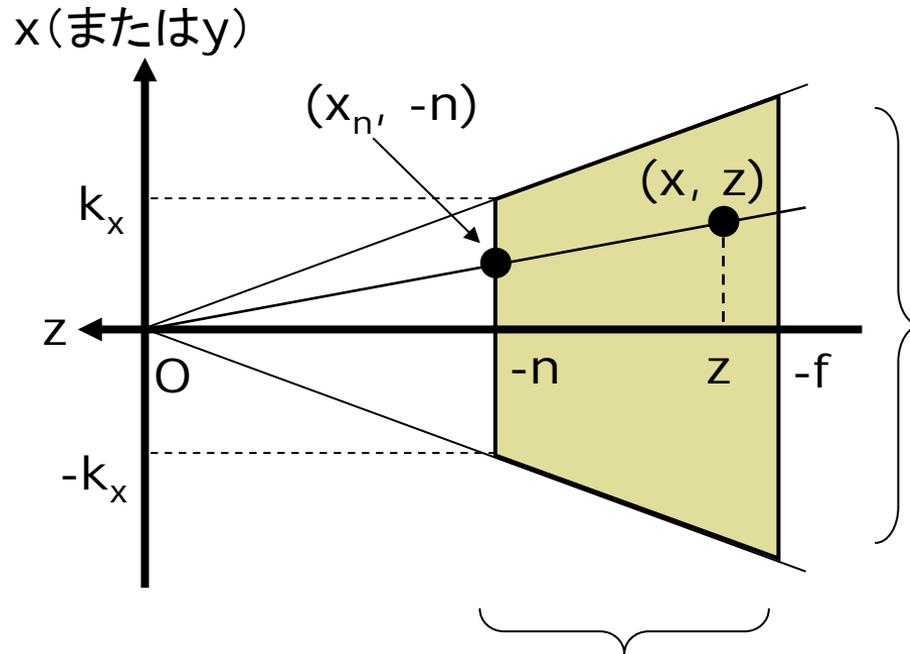
- 正規化
  - 平行変換と同様に, x, y, z座標の値を-1~+1の範囲に収める
  - 四角錐台 → 立方体



ビューポート変換(平行投影と同様)

# 9.4 透視投影の計算

## 視体積の正規化



z座標も,  $-1 \sim +1$  の範囲に収める

$z = -n$  のとき  $z_p = +1$

$z = -f$  のとき  $z_p = -1$

OpenGLの採用している計算式 →

三角形の相似より ( $z < 0$  に注意)

$$x_n : n = x : -z \quad (\text{y軸も同様})$$

$$\therefore x_n = x \cdot \frac{n}{-z}, \quad y_n = y \cdot \frac{n}{-z}$$

$x, y$ 座標を  $-1 \sim +1$  の範囲に収める

$$x_p = \frac{x_n}{k_x} = \frac{n}{k_x} \cdot \frac{x}{-z}$$

$$y_p = \frac{n}{k_y} \cdot \frac{y}{-z}$$

$$z_p = -\frac{z(f+n) + 2fn}{-z(f-n)}$$

# 9.5 透視投影行列

## 透視投影行列の算出

- 正規化視体積での座標  
(=正規化デバイス座標)

$$x_p = \frac{n}{k_x} \cdot \frac{x}{-z}, \quad y_p = \frac{n}{k_y} \cdot \frac{y}{-z}$$

$$z_p = -\frac{z(f+n) + 2fn}{-z(f-n)}$$

- 同次座標に変換
  - $x, y$ 座標の都合で  $w' = -z$  とする

$$\begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix} \Leftrightarrow \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} \quad \begin{aligned} x' &= w' x_p \\ y' &= w' y_p \\ z' &= w' z_p \end{aligned}$$

- 同次座標を計算

$$x' = \frac{n}{k_x} x, \quad y' = \frac{n}{k_y} y$$

$$z' = -\frac{f+n}{f-n} z - \frac{2fn}{f-n}$$

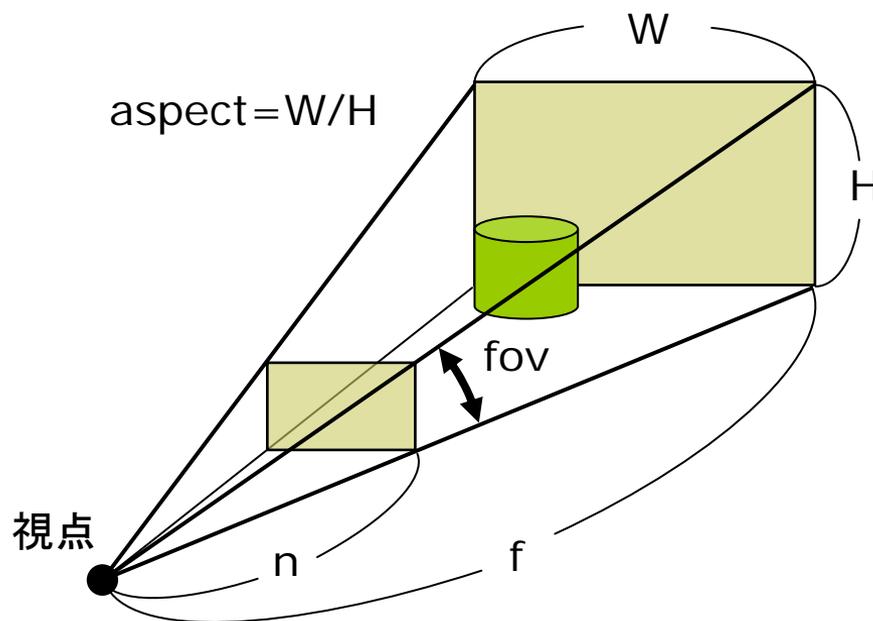
- 透視投影行列
  - 上記の同次行列表現

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} \frac{n}{k_x} & 0 & 0 & 0 \\ 0 & \frac{n}{k_y} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

## 9.6 透視投影関数 (7.3参照)

### 透視投影関数

- perspective(fov, aspect, n, f)
  - ただし、すべての引数はゼロ以外
  - aspectは、floatで計算すること
  - 遠・近クリッピング(n, f)は不正確 (OpenGLウィンドウなら正確)

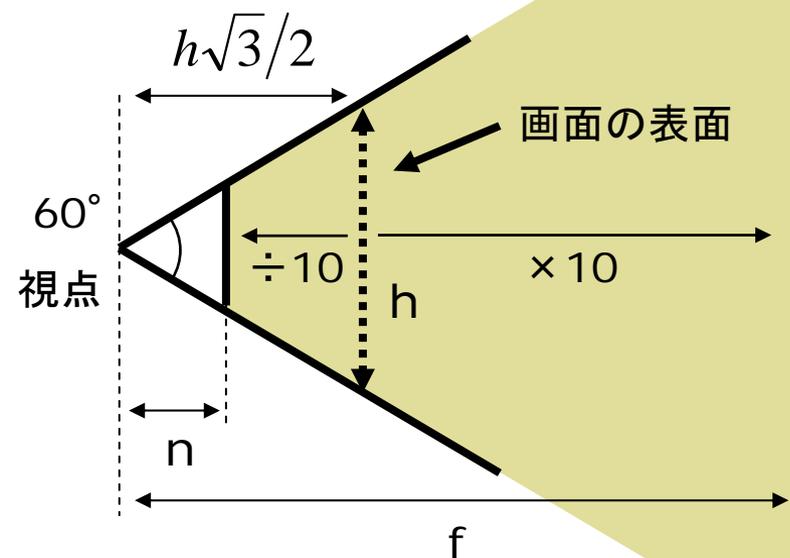


### □ 無指定時の視体積

- perspective()を呼ばない場合
- または、引数なしで呼んだ場合

aspect = ウィンドウ比

h = height  
(ウィンドウの高さ)



## 9.7 ポリゴン描画

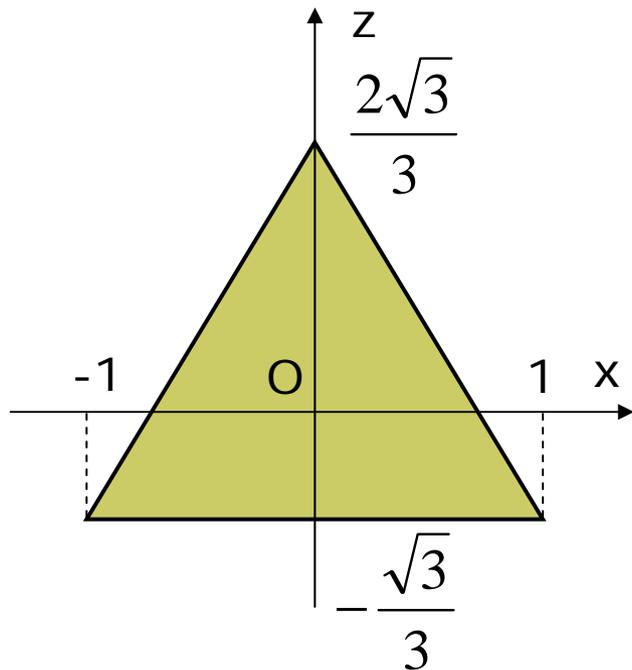
### ポリゴン描画

- box, sphere以外のオブジェクトは、多角形の集合で描画する

### □ 例) 三角柱を描画する

- 幅=2 ( $-1 \leq x \leq 1$ ), 原点に重心,  
高さ=2 ( $-1 \leq y \leq 1$ )

真下から見た図

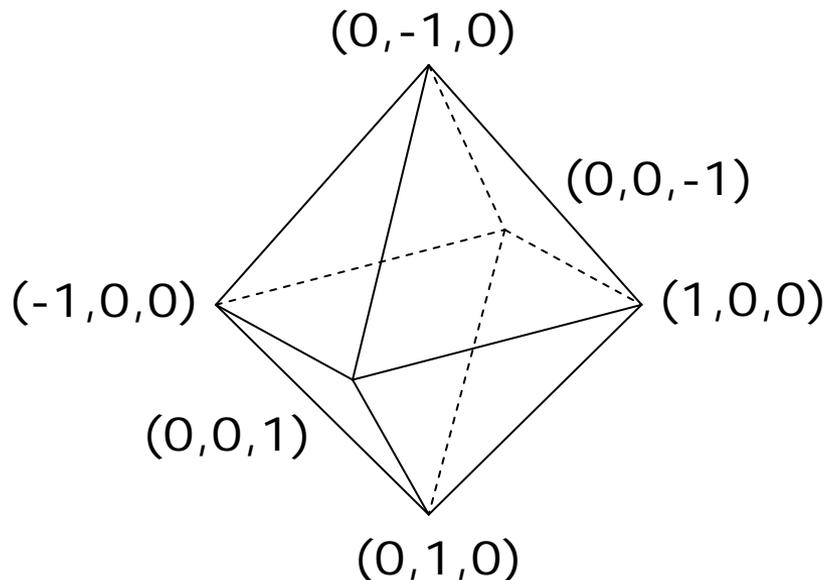


```
void prism3() {
    float g = sqrt(3) / 3.0;
    beginShape(QUAD_STRIP);
        vertex(1, -1, -g);   vertex(1, 1, -g);
        vertex(0, -1, g*2); vertex(0, 1, g*2);
        vertex(-1, -1, -g); vertex(-1, 1, -g);
        vertex(1, -1, -g);   vertex(1, 1, -g);
    endShape();
    beginShape(TRIANGLES);
        vertex(1, -1, -g);   vertex(0, -1, g*2);
        vertex(-1, -1, -g); vertex(1, 1, -g);
        vertex(0, 1, g*2);   vertex(-1, 1, -g);
    endShape();
}
```

## 9.8 演習課題

### 課題

- 正八面体を描画するプログラムを作成しなさい
  - 8枚の正三角形を描画する
  - `beginShape()`でTRIANGLESかTRIANGLE\_FANを用いる
  - 下図に座標値の例を示した



### □ 三角形の描画例

```
scale(適当な倍率)
beginShape(TRIANGLES);
  vertex(1, 0, 0);
  vertex(0, 0, 1);
  vertex(0, 1, 0);
  // さらに必要な枚数の三角形
endShape();
```

### 参考(演習とは無関係)

#### □ OpenGLウィンドウ

- より正確な3D描画
- プログラム冒頭
 

```
import processing.opengl.*;
```
- ウィンドウオープン
 

```
size(幅, 高さ, OPENGL);
```