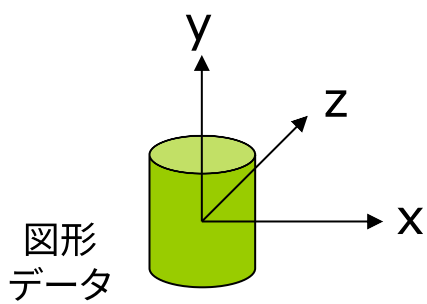
# Graphics with Processing

2019-09 投影変換と隠面消去

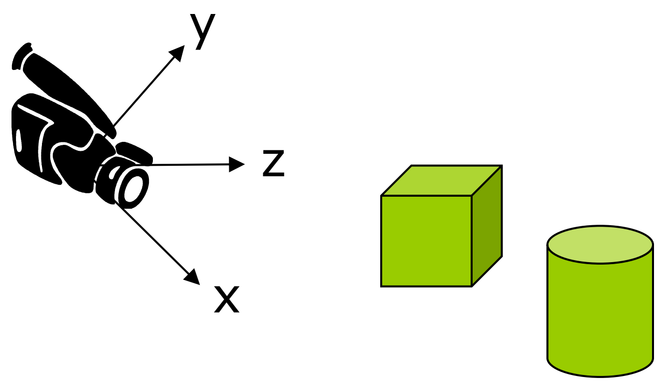
http://vilab.org
塩澤秀和

# 8.1* 3DCGの座標系(p.49)

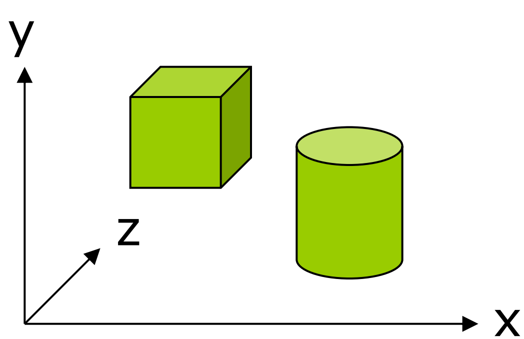
- ローカル(モデリング)座標系
  - オブジェクトの座標系



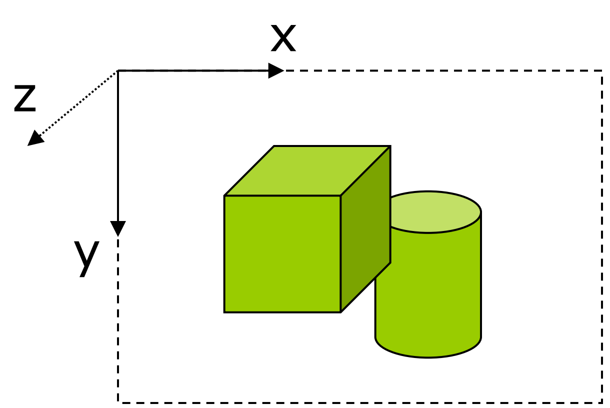

- 視点(カメラ)座標系
  - 遠近感や前後関係を計算




- ワールド座標系
  - 3次元世界の座標系

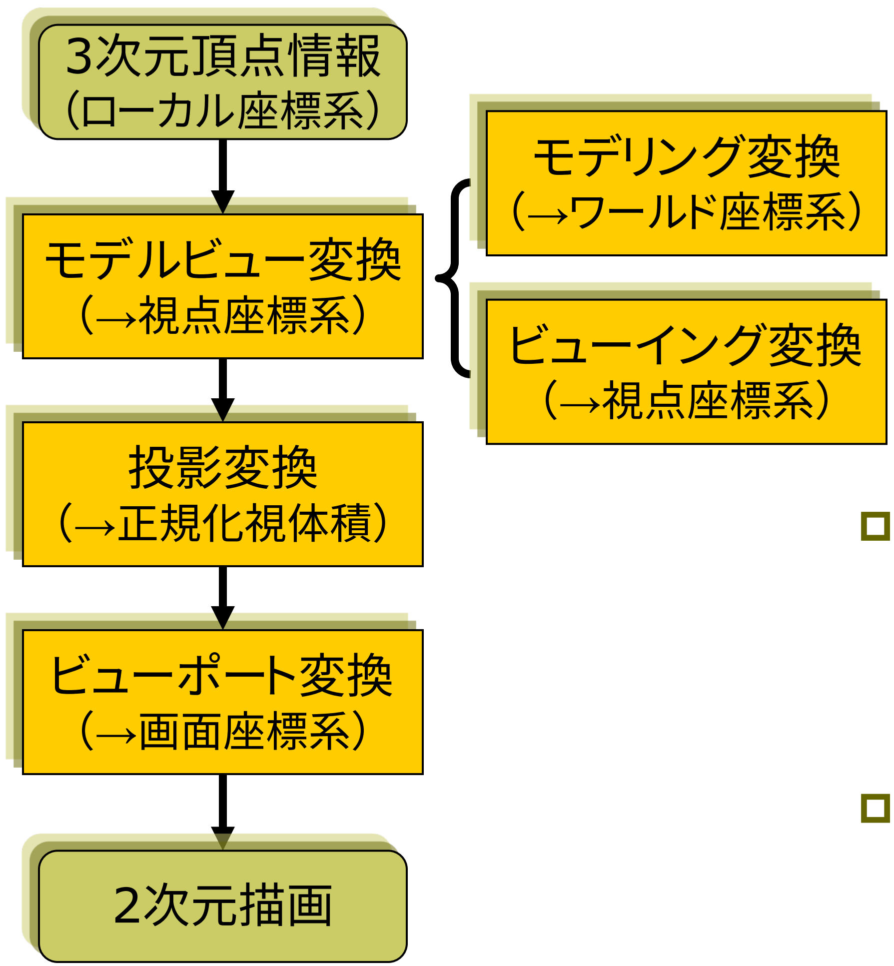



- 画面(デバイス)座標系



# 8.2* 3DCGの座標変換(p.49)

- 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)

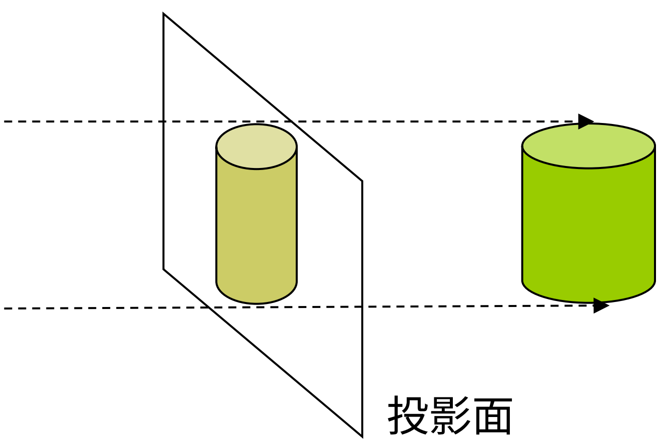
# 9.1* 平行投影と透視投影(p.38)

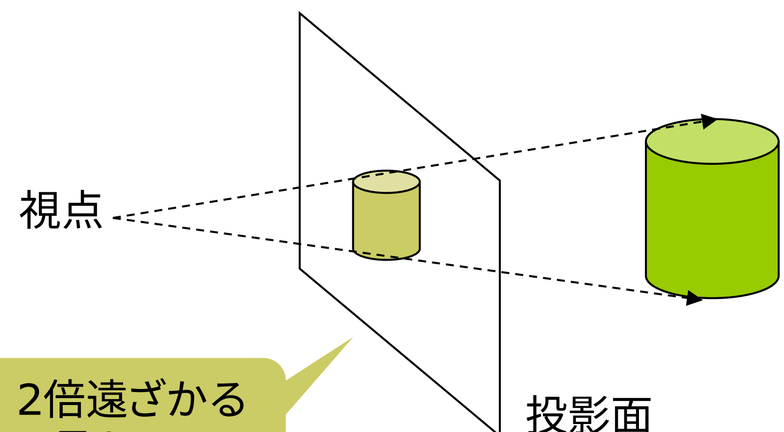
平行投影(直交投影)

- ortho($x_{min}$, $x_{max}$, $y_{min}$, $y_{max}$, $z_{min}$, $z_{max}$)
  - 遠近感をつけない投影方法
  - 画面に表示するx, y, z座標の範囲(視体積)を設定
- サンプル
  - Basics (3D) → Camera

透視投影(透視図法)

- pserspective()
  - 人が実際に見るように,遠いものほど小さく描画する(遠近法)
  - 投影面に映る大きさを計算
- perspective(fov, aspect, zNear, zFar)
  - 視野角(画角)などを指定



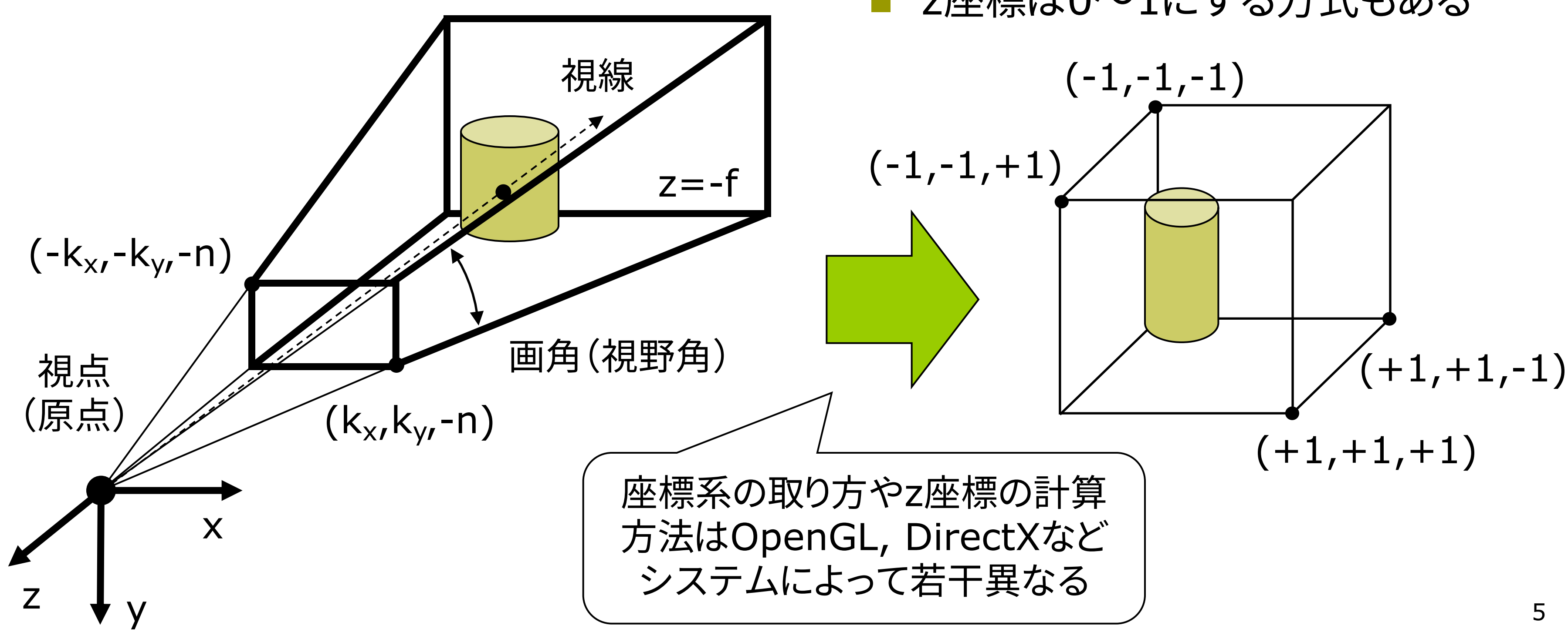

2倍遠ざかると見た目は?

# 9.2* 透視投影(p.39)

透視投影

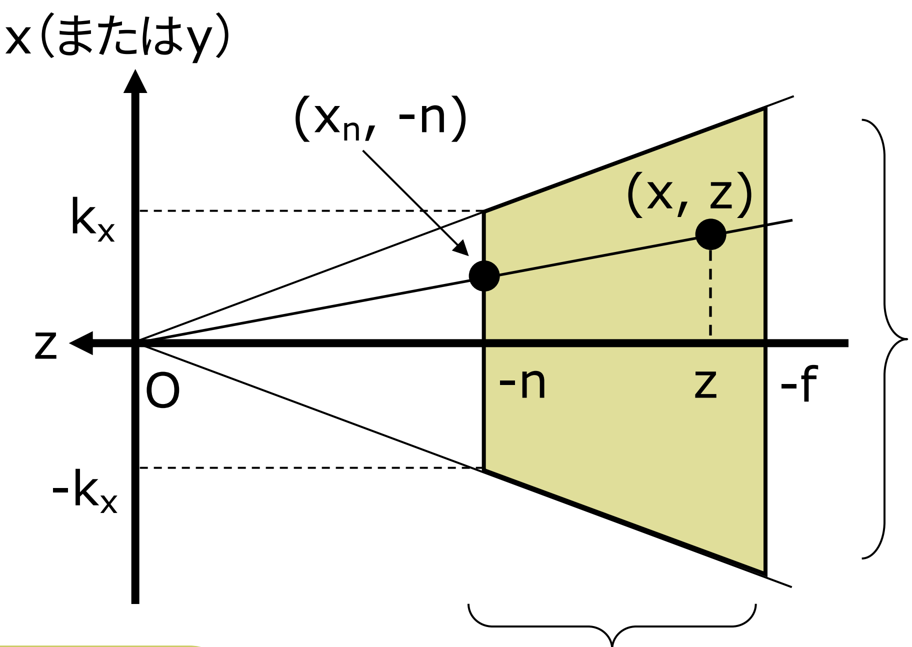
- 視体積(ビューボリューム)
  - 画角(視野角) ⇒ 見える範囲
  - 画角大＝広角,画角小＝望遠
  - 透視投影の視体積は四角錘台

- 正規化視体積
  - 各座標の値を-1～+1に正規化
  - 四角錐台 → 立方体
  - 空間が歪み,視点から遠いものほど大きく縮む ⇒ 遠近感
  - z座標は0～1にする方式もある



# 9.3* 透視投影の計算（p.43参考）

視体積の正規化




三角形の相似より（z<0に注意）

$$x_n : n = x : -z$$ （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〜+1の範囲に収める

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

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

視点からの距離に反比例

4×4の行列で表す必要から

$$z_p = \frac{a}{z} + b$$

に代入して$a, b$を求める

z座標も,-1〜+1の範囲に収める

$z=-n$ のとき $z_p=+1.0$

$z=-f$ のとき $z_p= -1.0$

OpenGL/Processingの計算式 ⇒

（教科書の方式は,0.0〜1.0）

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

# 9.4 透視投影行列（p.43参考）

- 同次座標で表現

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

9.3の式を同次座標で表す
分母にzがあるので $w'_p = -z$ で対応

$$x'_p = \frac{n}{k_x} x \qquad y'_p = \frac{n}{k_y} y$$

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

$$w'_p = -z$$

$w'_p$に視点からの
奥行情報が残る

- 透視投影行列
  - OpenGL/Processingの方式

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

$$\Updownarrow$$

$$P_{proj} = M_{proj} P_{view}$$

- ここまでの座標変換の合成

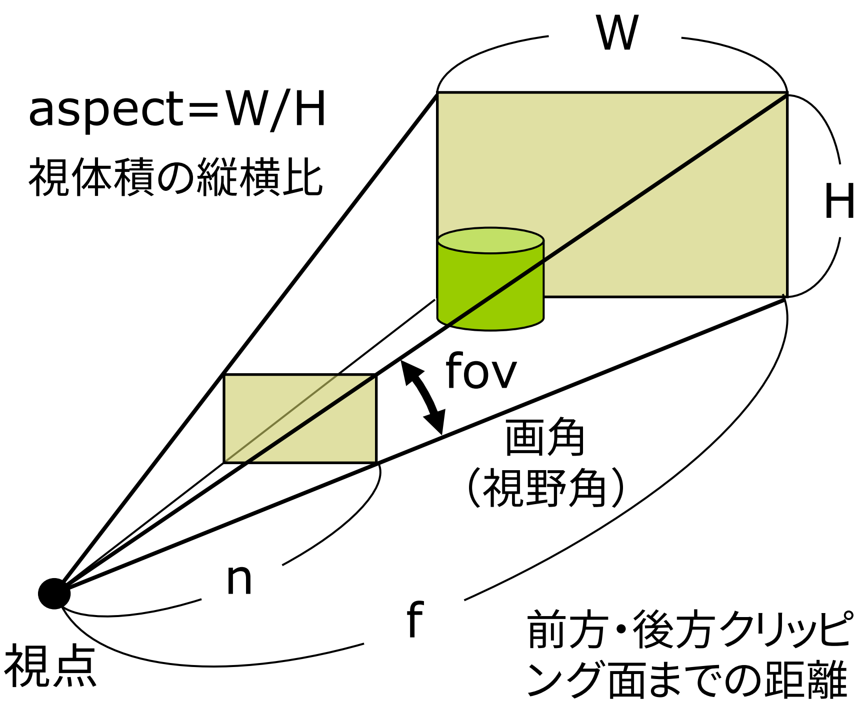
$$P_{proj} = M_{proj} M_{view} M_{model} P_{local}$$

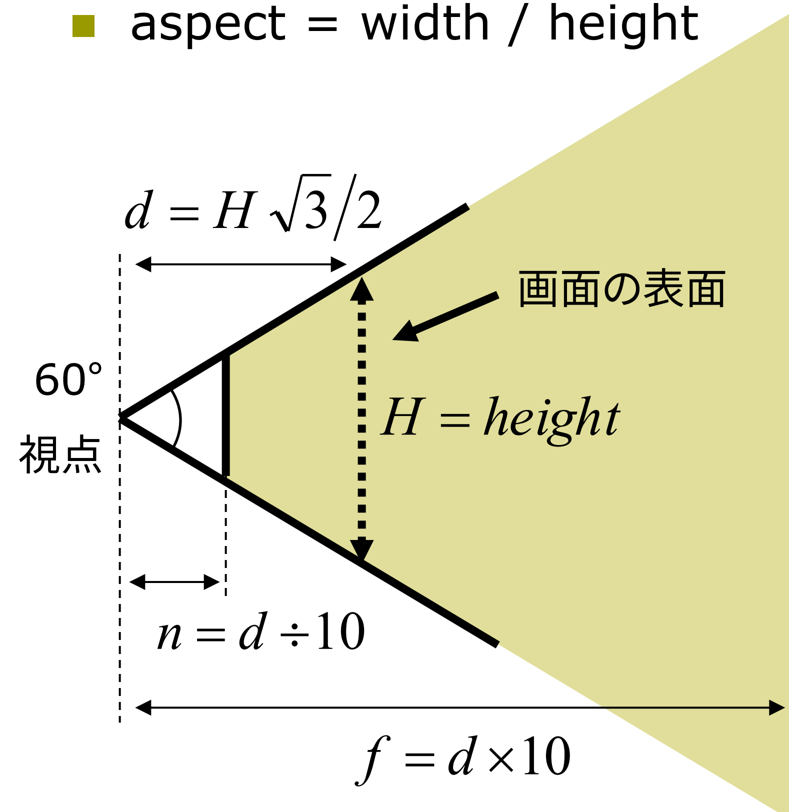
# 9.5 透視投影関数

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
  - fov（視野角）は縦方向で指定
  - aspectは，floatで計算すること

- Processingのデフォルト設定
  - perspective()を呼ばない場合
  - または，引数なしで呼んだ場合
  - 画角（視野角） = 60°（π/3）
  - aspect = width / height

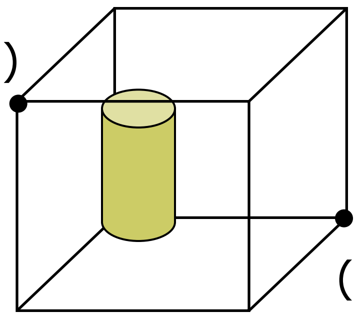


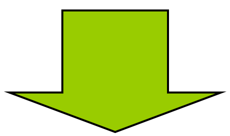



# 9.6 ビューポート変換とクリッピング

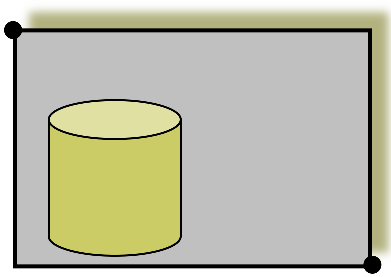
ビューポート変換（p.50）

- 正規化視体積



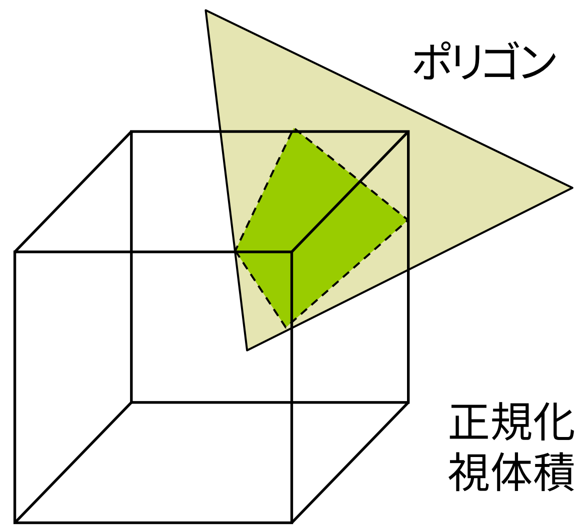



- デバイス座標系




3次元クリッピング（p.53）

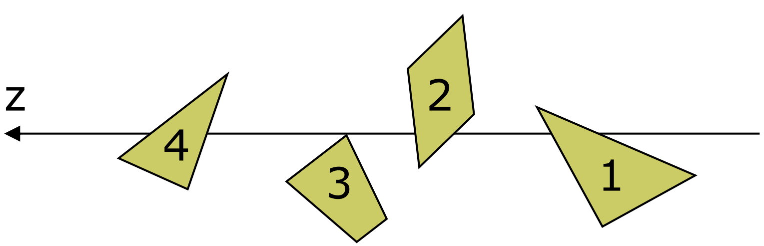
- 線分のクリッピング
  - コーエン・サザランド法（4.6）
  - z座標を加えた6ビットコード
- ポリゴンのクリッピング
  - ポリゴンの形状が変わるので，分割処理等が必要になる
  - 特に三角形しか扱えない場合



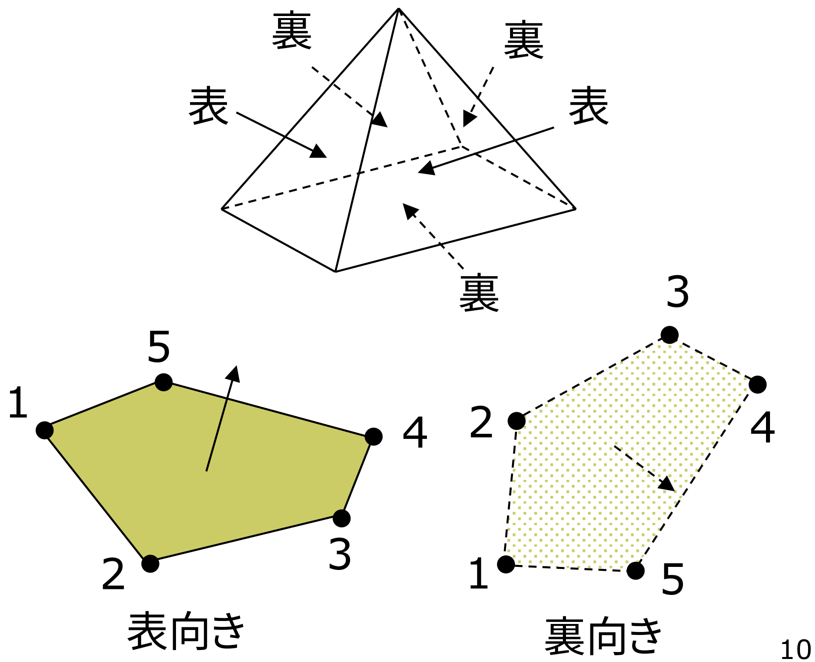
# 9.7* 隠面消去(1)

隠面消去(隠線・隠面処理)

- 隠面消去とは
  - 他の物体などに**隠れて見えない**物体(の全部または一部)を描画しない処理
  - 弱点を補い合ういくつかの手法を組み合わせることもある
- 奥行きソート法(p.127)
  - ポリゴンをz座標(視点座標)で並び替え,遠くから順に描画
  - 細長いポリゴンで問題が生じる

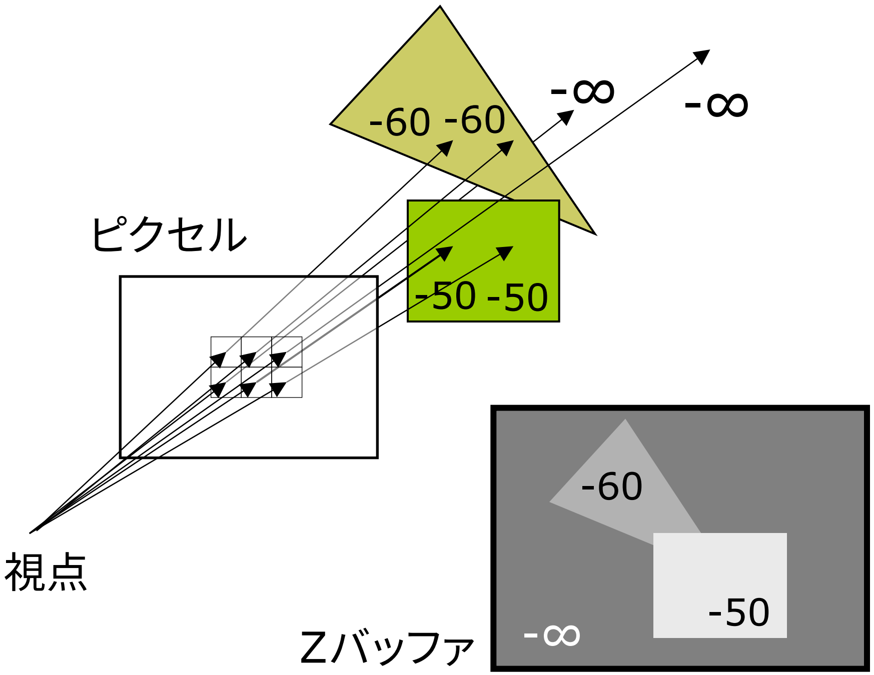


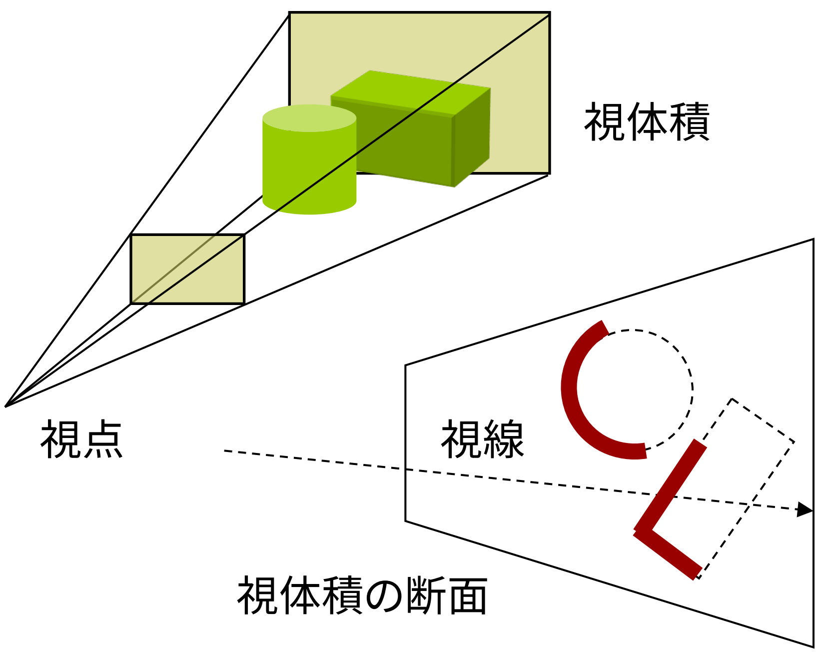

- バックフェースカリング(p.126)
  - ポリゴンに表裏を設定し,裏側を向いているポリゴンを描画しない
  - 表裏はポリゴン作成時の頂点の順序(右回り・左回り)で指定
  - 各凸多面体での隠面消去



# 9.8* 隠面消去(2)

- Zバッファ法(p.133)
  - 画面上の全ピクセルに z座標を持たせ,1点1点描画するときに遠近関係をチェックする
  - 単純&高速 ⇒ ハードウェア化
  - 半透明の重なりの処理に難点
- スキャンライン法(p.130)
  - ピクセル横1行(スキャンライン)ごとにポリゴンの断面の重なりを数学的に計算し,描画する
  - 計算は複雑だが,使用メモリが少ない



# 9.9* 演習課題

課題

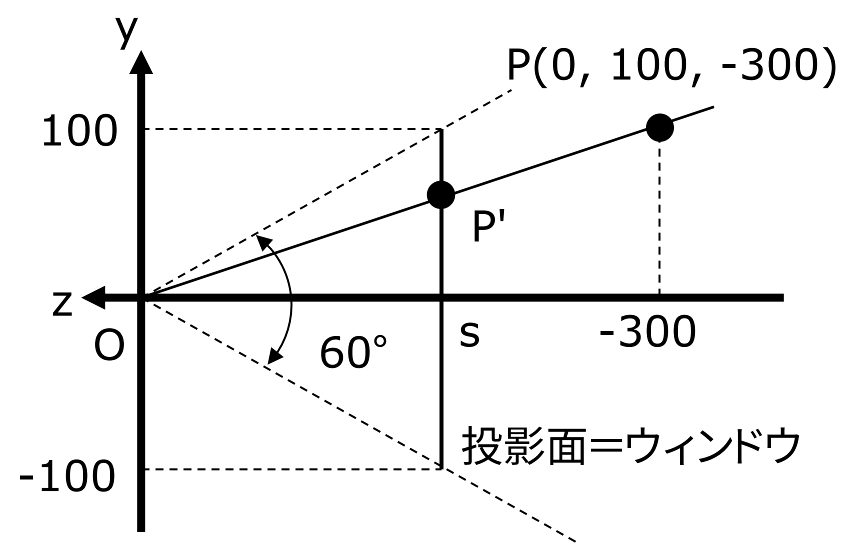
問1) 9.10のプログラムに適切な処理を補って，実行してみなさい

- 適当なsetup関数を補う

1. 紙飛行機が遠くから手前に近づいてきて，カメラの横を飛び去っていくようにしなさい
   - 飛び去ったら，元の位置に戻って繰り返すようにしなさい
   - ヒント： translate
2. カメラの向きを紙飛行機をずっと追跡するようにしなさい
   - ヒント： camera
3. マウスのボタンでカメラを望遠に切り替えられるようにしなさい
   - ヒント： perspective

問2) 下図は投影変換の原理を示したものである（ウィンドウサイズは200×200，画角は60°とする）

1. P’のz座標 s を求めなさい
2. 視点座標系で (0, 100, -300) に変換された点Pが，投影面上に写像される座標P'を求めなさい
   - 次回，**A4レポート用紙**で提出



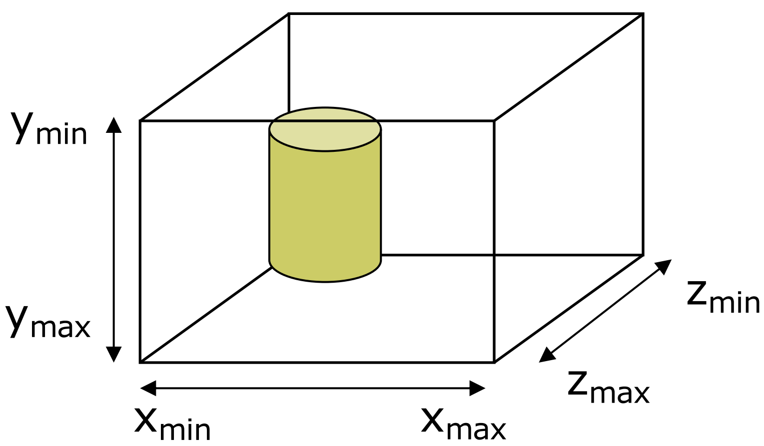
# 9.10 演習課題(続き)

```
void draw() {
  background(50, 50, 100);

  // 画角の設定
  perspective(PI/3, (float) width /
                  height, 10, 10000);

  // カメラの位置と撮影目標の設定
  camera(-150, -500, 1500,
             0, 0, 0,  0, 1, 0) ;

  // 照明の光を上からに変更
  pushMatrix();
    rotateX(PI/2); lights();
  popMatrix();

  fill(255); noStroke();
  pushMatrix();
    translate(0, -300, 1200);
    paperplane();
  popMatrix();

  fill(0, 50, 0); noStroke();
  for (int i = -10; i <= 10; i++) {
    for (int j = -10; j <= 10; j++) {
      pushMatrix();
        translate(i*200, 0, j*200);
        box(180, 10, 180);
      popMatrix();
    }
  }
}

// 紙飛行機のモデル
void paperplane() {
  beginShape(TRIANGLE_FAN);
  vertex(0, 0, 0);
  vertex(-30, 5, -50);
  vertex(-5, 0, -50);
  vertex(0, 20, -50);
  vertex(5, 0, -50);
  vertex(30, 5, -50);
  endShape();
}
```

# 9.11 参考：平行投影(p.45)

平行投影(直交投影)

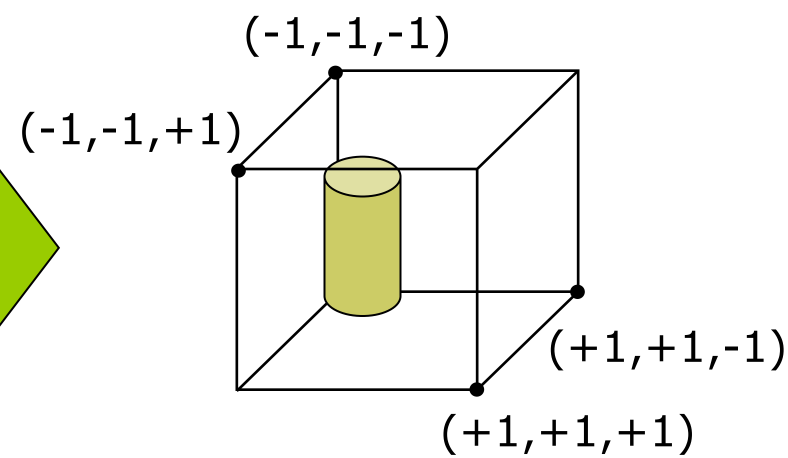
- 視体積(ビューボリューム)
  - 視体積=「見える領域」
  - 平行投影の視体積は直方体




平行投影関数

- ortho($x_{min}$, $x_{max}$, $y_{min}$, $y_{max}$, $z_{min}$, $z_{max}$)

- 正規化視体積
  - 各座標の値を-1～+1に正規化
  - 直方体 → 立方体
  - z座標は0～1にする方式もある



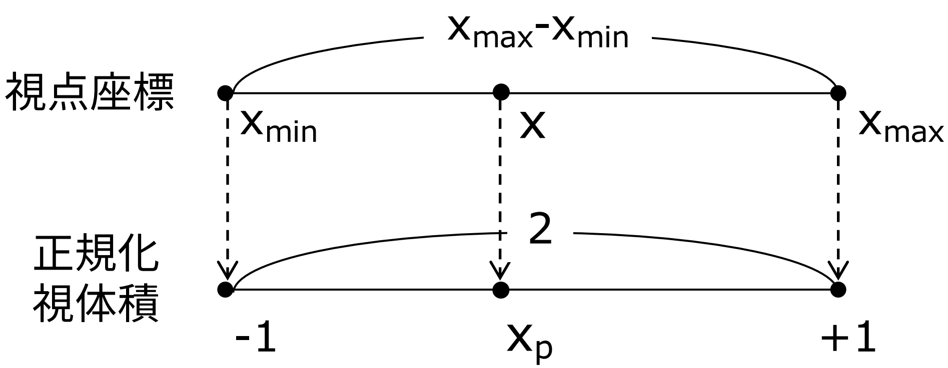

- 計算式

$$x_p = \frac{2x - (x_{max} + x_{min})}{x_{max} - x_{min}}$$

y, z も同様(教科書はzは0～1)

# 9.12 参考：平行投影行列（p.45参考）

- 平行投影の計算




$$x_p = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \times 2 - 1$$

$$= \frac{2x - (x_{\max} + x_{\min})}{x_{\max} - x_{\min}}$$

$$= \frac{2x}{x_{\max} - x_{\min}} - \frac{x_{\max} + x_{\min}}{x_{\max} - x_{\min}}$$

y座標，
z座標
も同様

- 変換行列による表現

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{2}{x_{\max} - x_{\min}} & 0 & 0 & -\frac{x_{\max} + x_{\min}}{x_{\max} - x_{\min}} \\ 0 & \frac{2}{y_{\max} - y_{\min}} & 0 & -\frac{y_{\max} + y_{\min}}{y_{\max} - y_{\min}} \\ 0 & 0 & \frac{2}{z_{\max} - z_{\min}} & -\frac{z_{\max} + z_{\min}}{z_{\max} - z_{\min}} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

正規化
視体積
の座標

視点
座標