Graphics with Processing

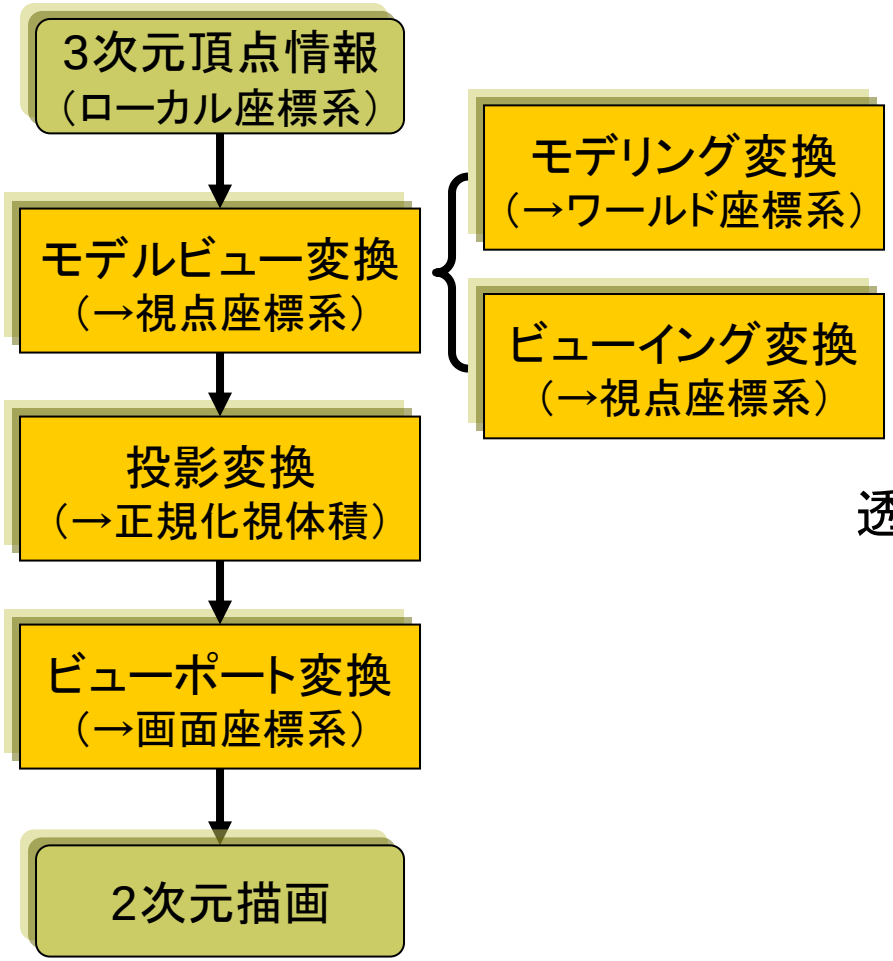
2009-09 投影変換

http://vilab.org

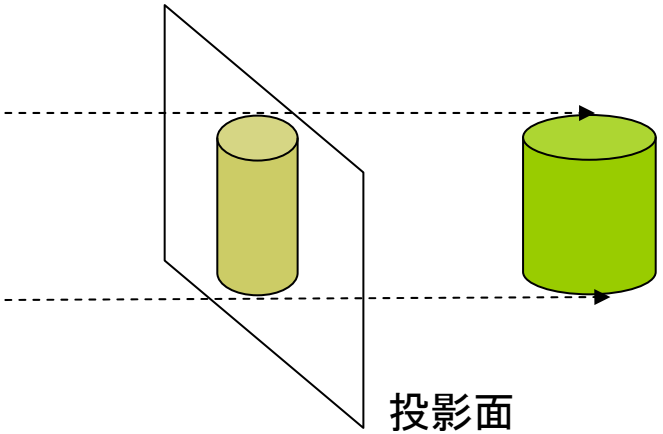
塩澤秀和

9.1 投影変換(p.32)

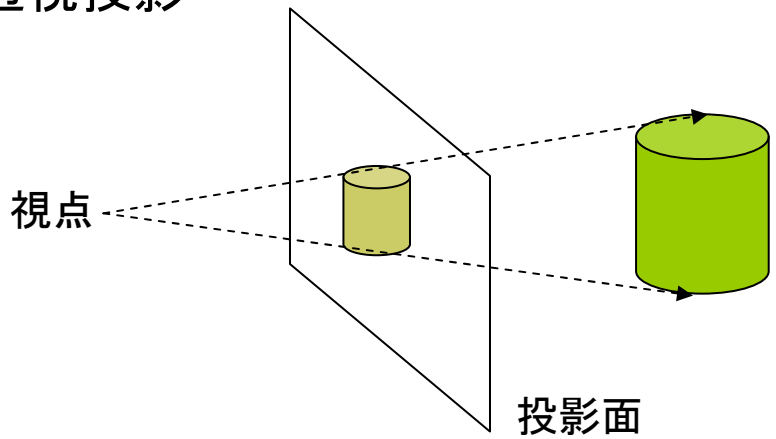
ビューイングパイプライン



平行投影



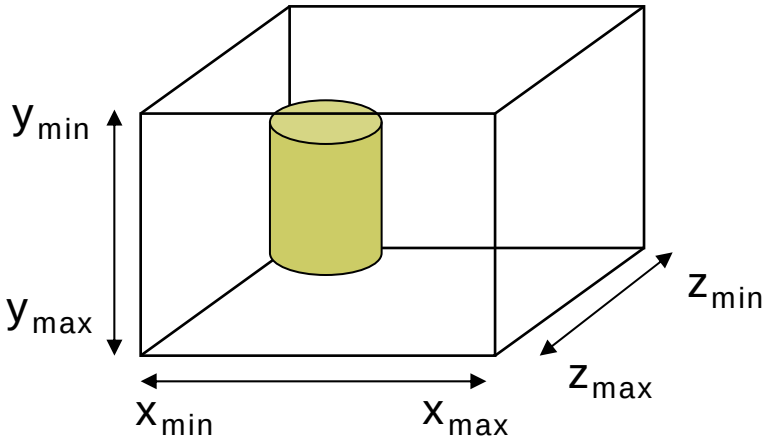
透視投影



9.2 平行投影(p.37)

平行投影(直交投影)

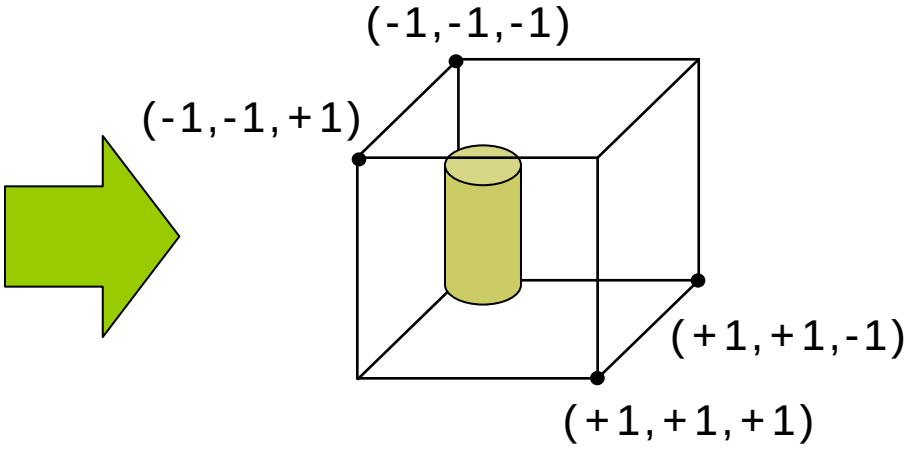
- 視体積(ビューボリューム)
  - 視体積＝「見える領域」
  - 平行投影の視体積は直方体



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

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

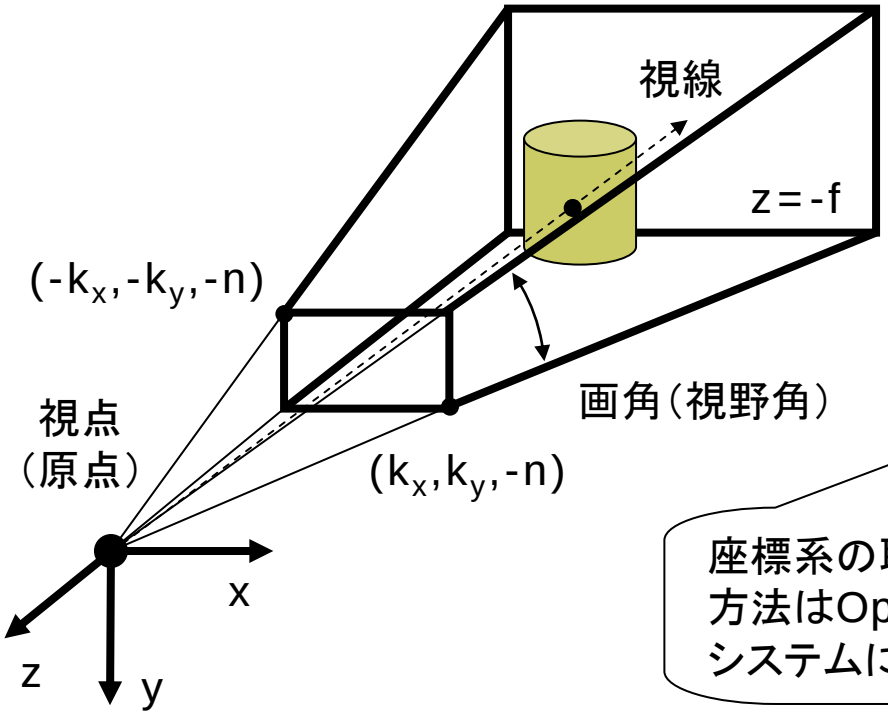
y, z も同様(教科書はzは0～1)

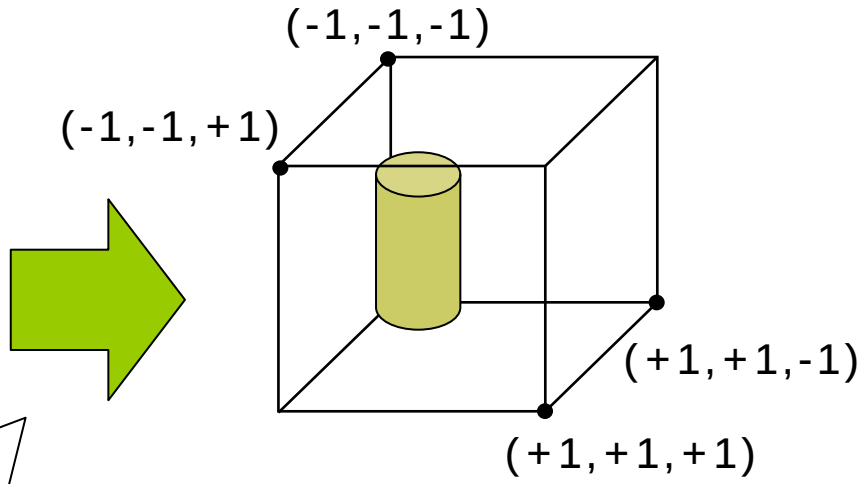
9.3 透視投影(p.35)

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錐台

- 正規化視体積
  - 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
  - z座標は0～1にする方式もある

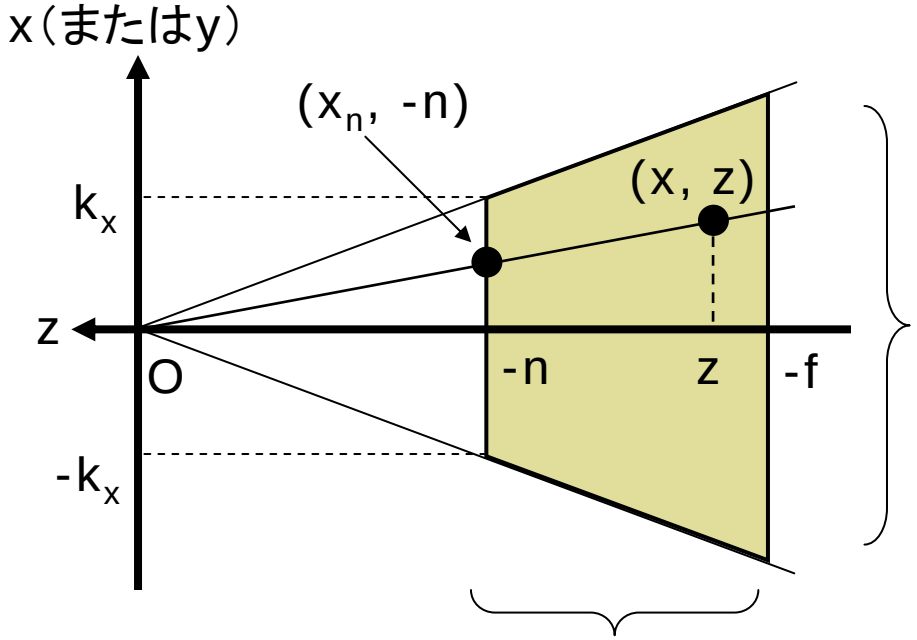




座標系の取り方やz座標の計算方法はOpenGL, DirectXなどシステムによって若干異なる

9.4 透視投影の計算（p.36参考）

視体積の正規化



三角形の相似より（z<0に注意）

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

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

x,y座標を-1～+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）

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

9.5 透視投影行列（p.36参考）

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

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

- 同次座標を計算

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

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

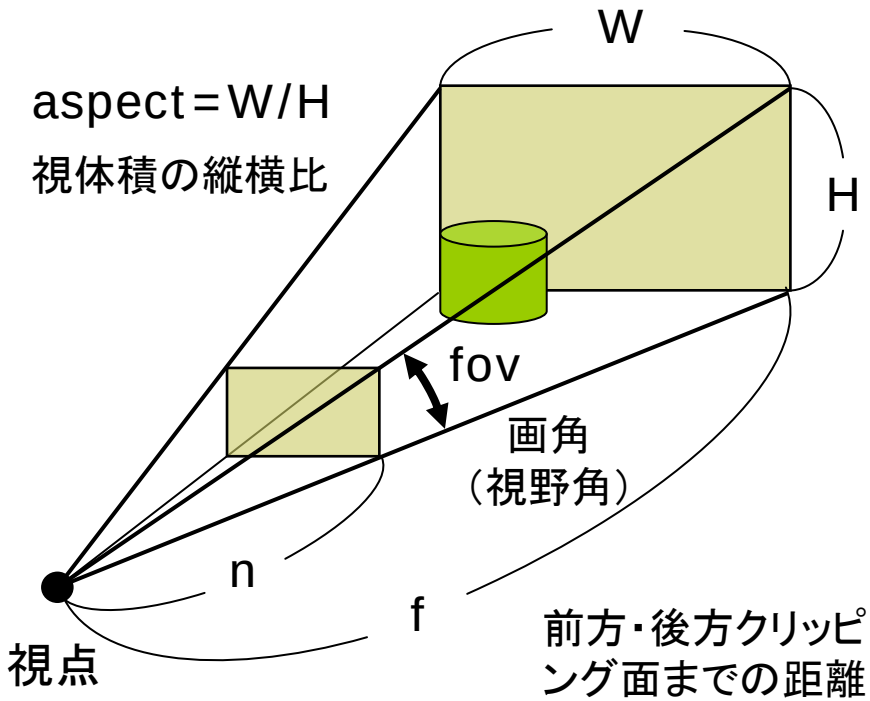
- 透視投影行列
  - OpenGL/Processingの方式

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

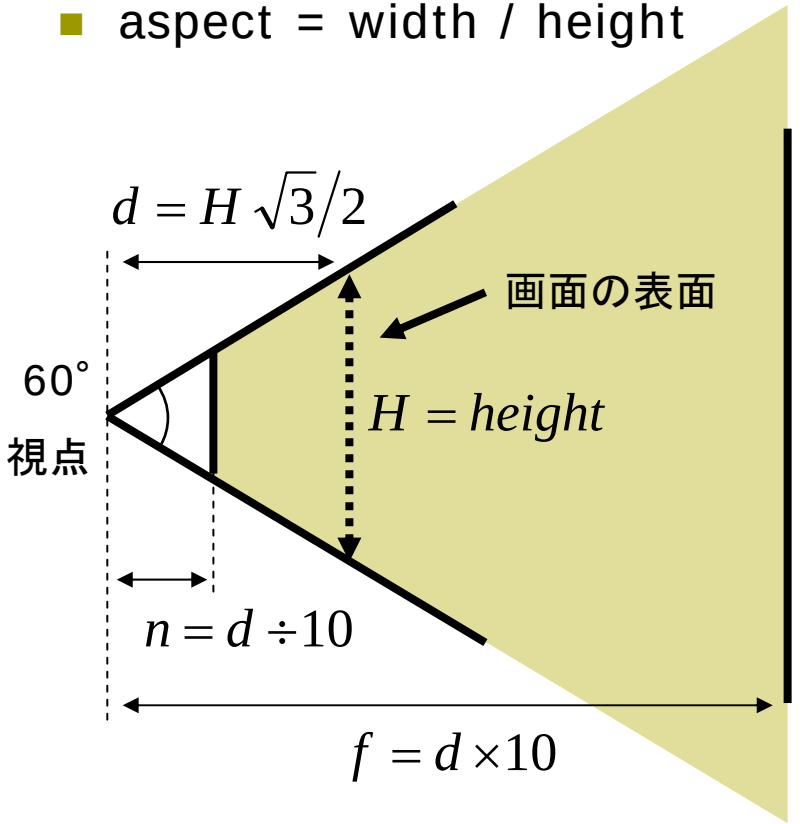
9.6 透視投影関数

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
  - P3DよりOPENGLのほうが正確



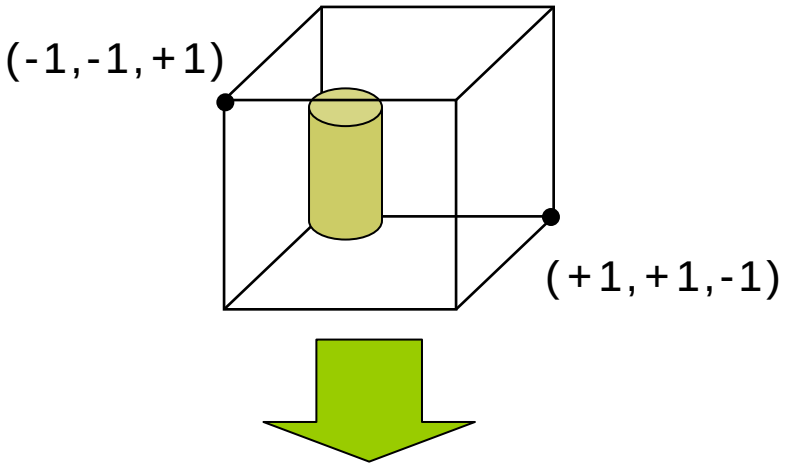
- 無指定時の自動設定
  - perspective()を呼ばない場合
  - または，引数なしで呼んだ場合
  - 画角（視野角）＝ 60°（π / 3）
  - aspect ＝ width / height



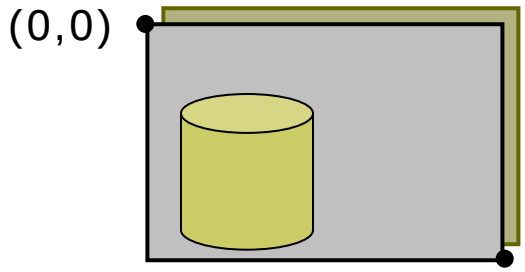
9.7 ビューポート変換とクリッピング

ビューポート変換（p.42）

- 正規化視体積

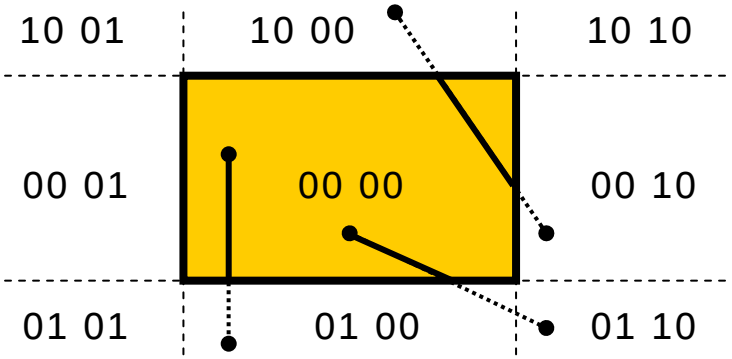


- デバイス座標系



クリッピング（p.44）

- クリッピングとは
  - 図形の表示領域（ビューポート）外の部分を描画しない処理



- 2次元クリッピング
  - コーエン・サザランドの方法
  - 線分端点のxy座標を4ビットでコード化し，ビット演算で表示領域にかかるかどうかを判定する
- 3次元クリッピング
  - z座標を加えた6ビットコード

9.8 演習課題

課題

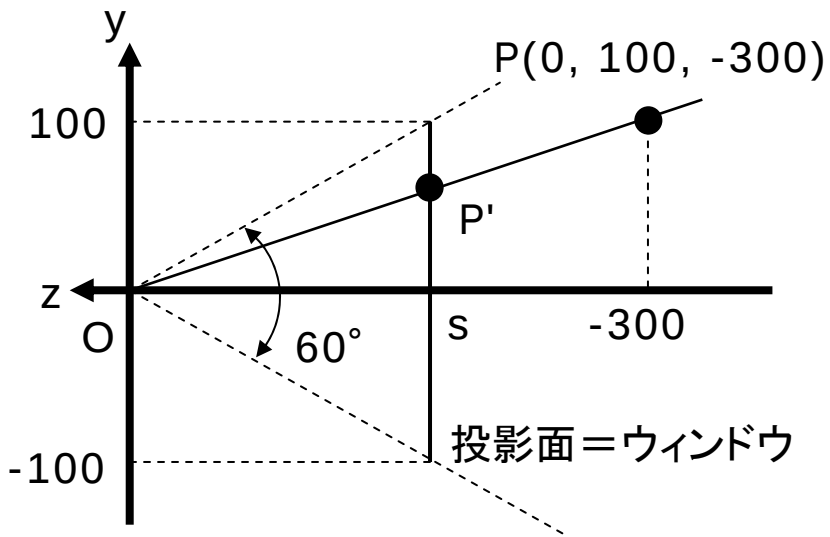
問1) 9.9のプログラムに適切な処理を補って，実行してみなさい

- OPENGLモードを使うこと

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. 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



9.9 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();
}
```