

Graphics with Processing



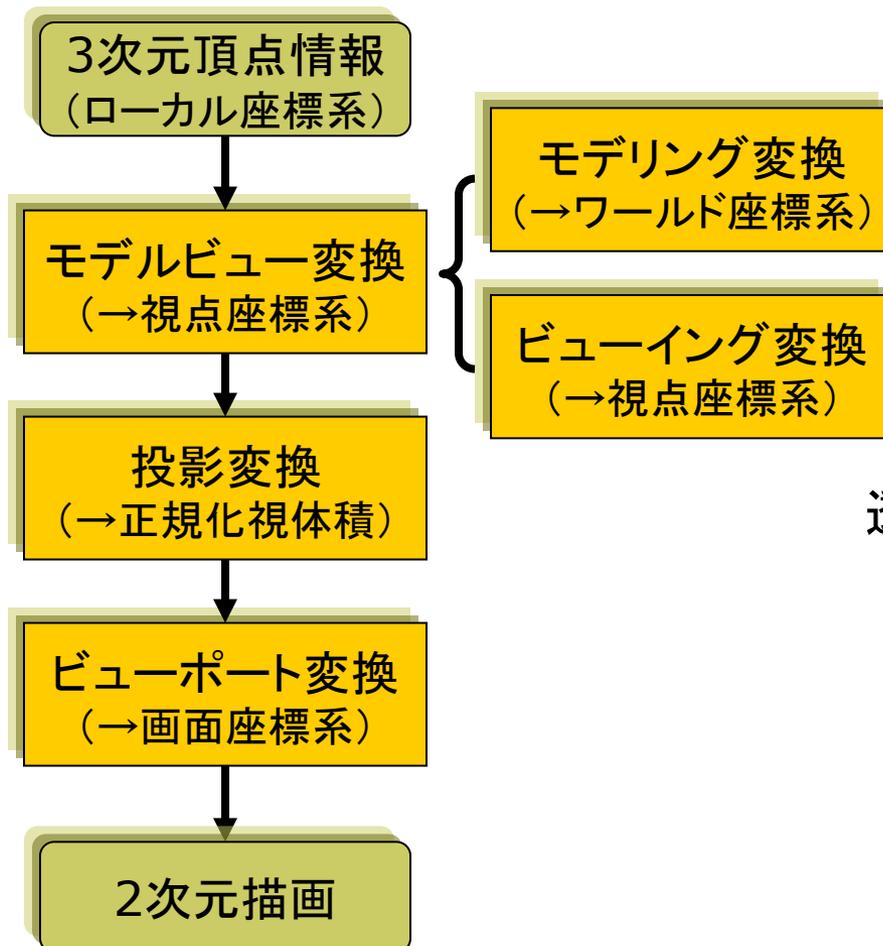
2010-09 投影変換と隠面消去

<http://vilab.org>

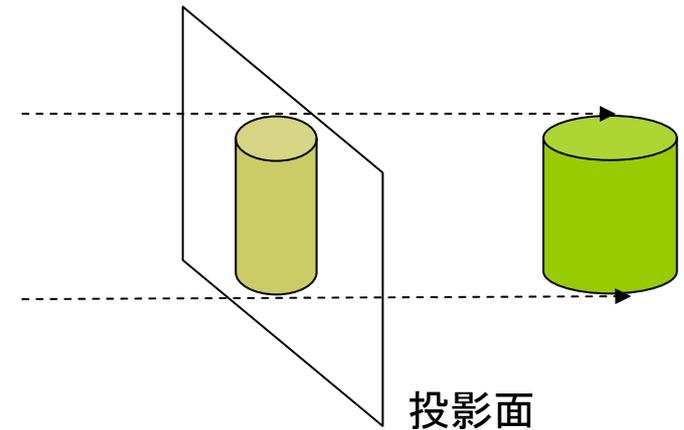
塩澤秀和

9.1 投影変換 (p.32)

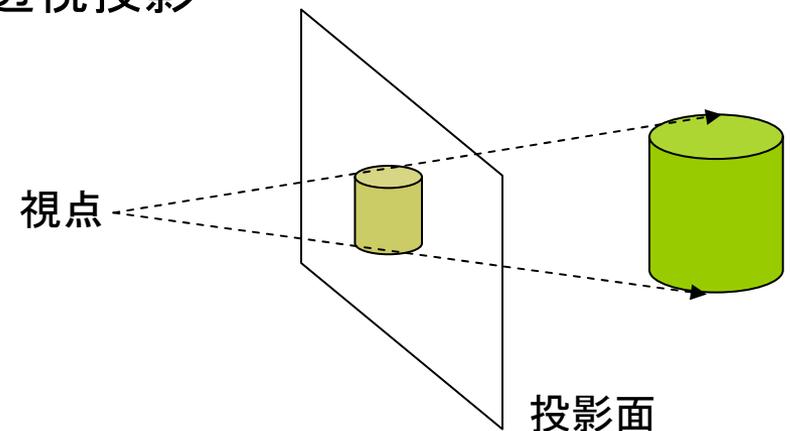
ビューイングパイプライン



平行投影



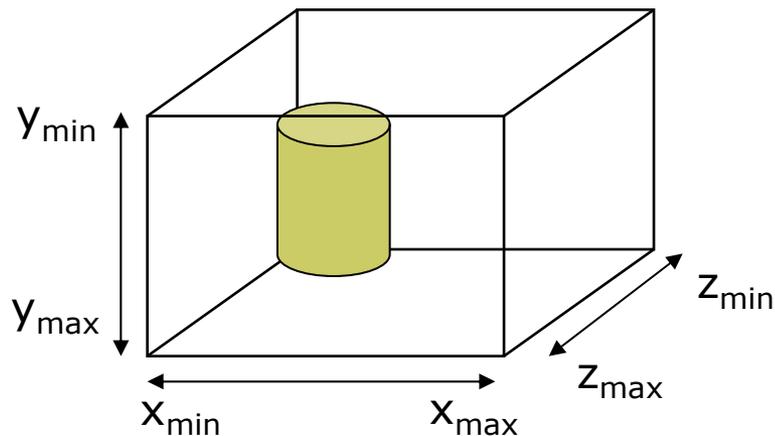
透視投影



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

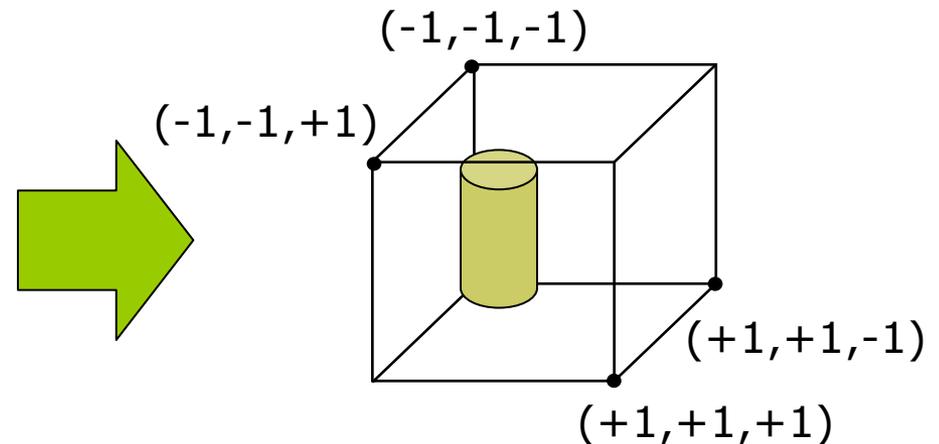


平行投影関数

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

□ 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



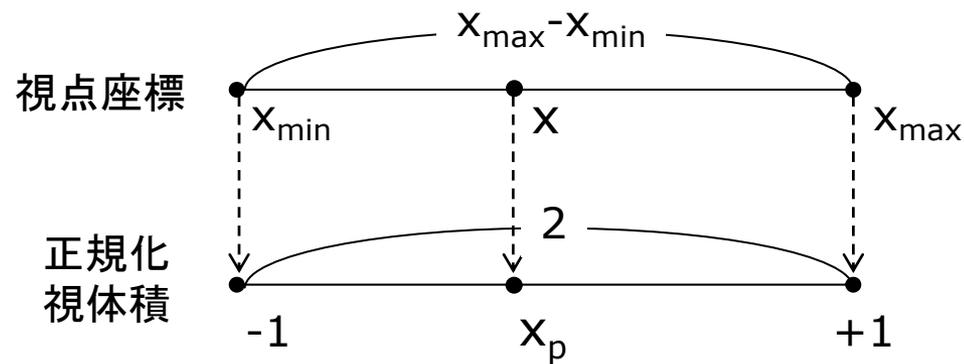
□ 計算式

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

y, z も同様(教科書は z は0~1)

9.3 平行投影行列 (p.37参考)

□ 平行投影の計算



$$\begin{aligned}
 x_p &= \frac{x - x_{\min}}{x_{\max} - x_{\min}} \times 2 - 1 \\
 &= \frac{2x - (x_{\max} + x_{\min})}{x_{\max} - x_{\min}} \\
 &= \frac{2x}{x_{\max} - x_{\min}} - \frac{x_{\max} + x_{\min}}{x_{\max} - x_{\min}}
 \end{aligned}$$

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

9.4 透視投影 (p.35)

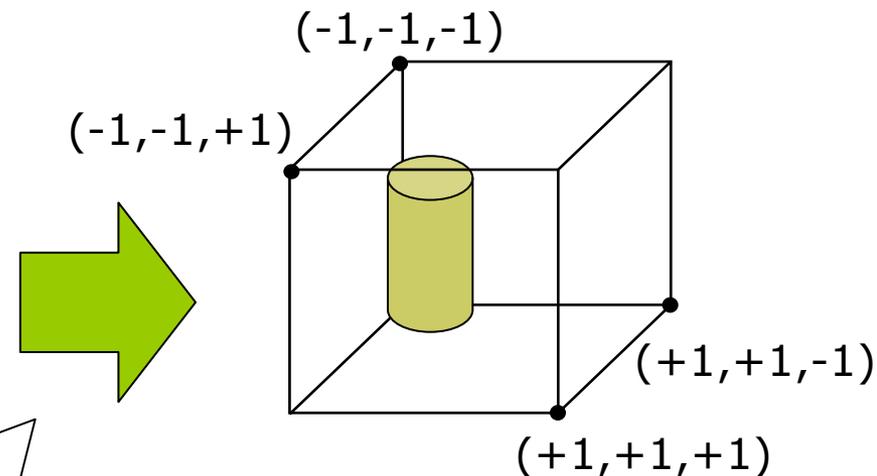
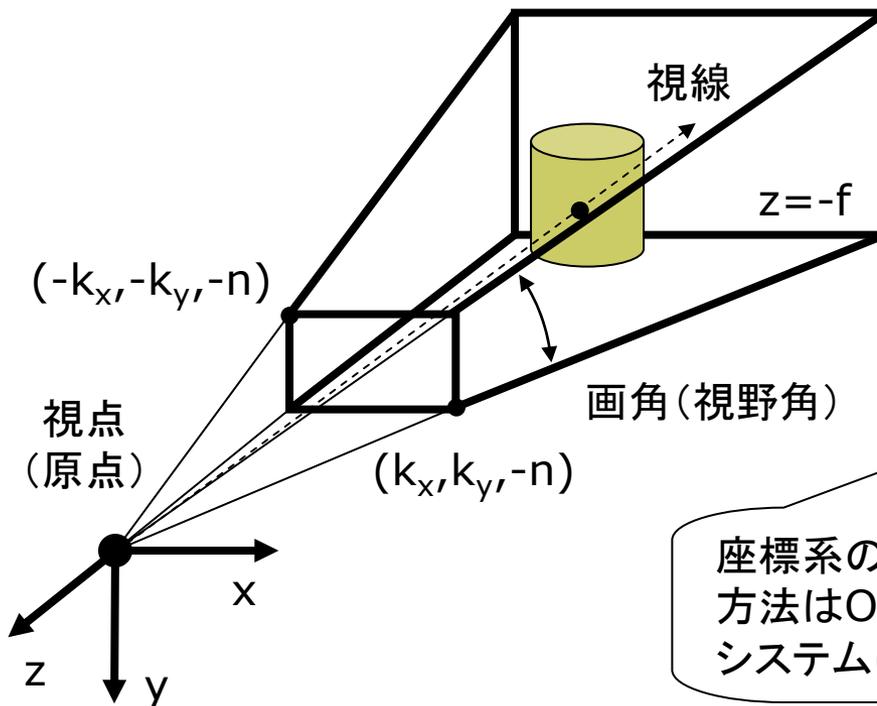
透視投影

□ 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

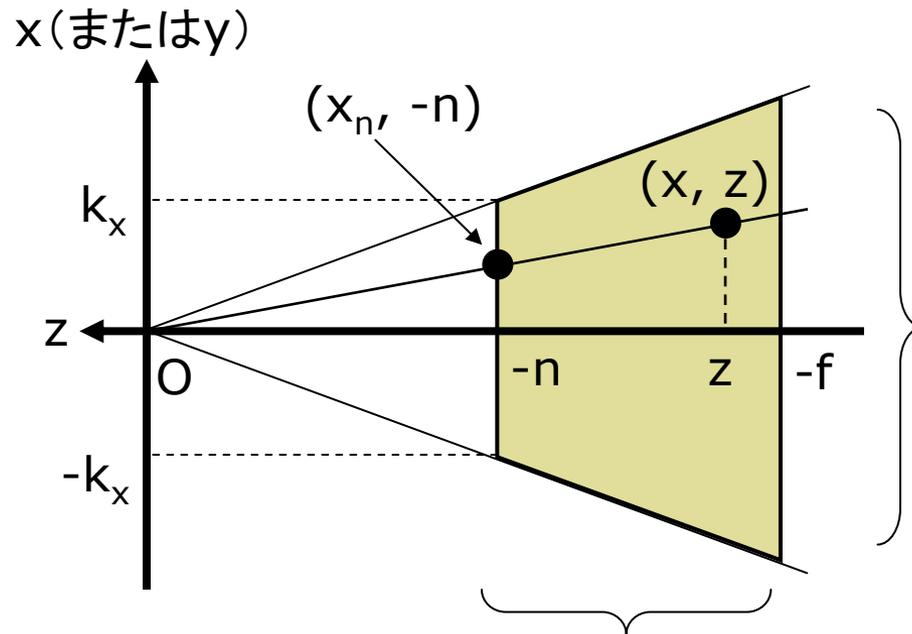
- 平行変換と同様に, x, y, z 座標の値を $-1 \sim +1$ の範囲に収める
- 四角錐台 → 立方体
- z 座標は $0 \sim 1$ にする方式もある



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

9.5 透視投影の計算 (p.36参考)

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 $z_p = +1.0$
 $z = -f$ のとき $z_p = -1.0$

OpenGL/Processingの計算式 \Rightarrow
 (教科書的方式は, $0.0 \sim 1.0$)

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

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

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

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

9.6 透視投影行列 (p.36参考)

□ 同次座標で表現

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

9.5の式を同次座標で表す
このとき $w'_p = -z$ とすると便利

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

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

$$w'_p = -z$$

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

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



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

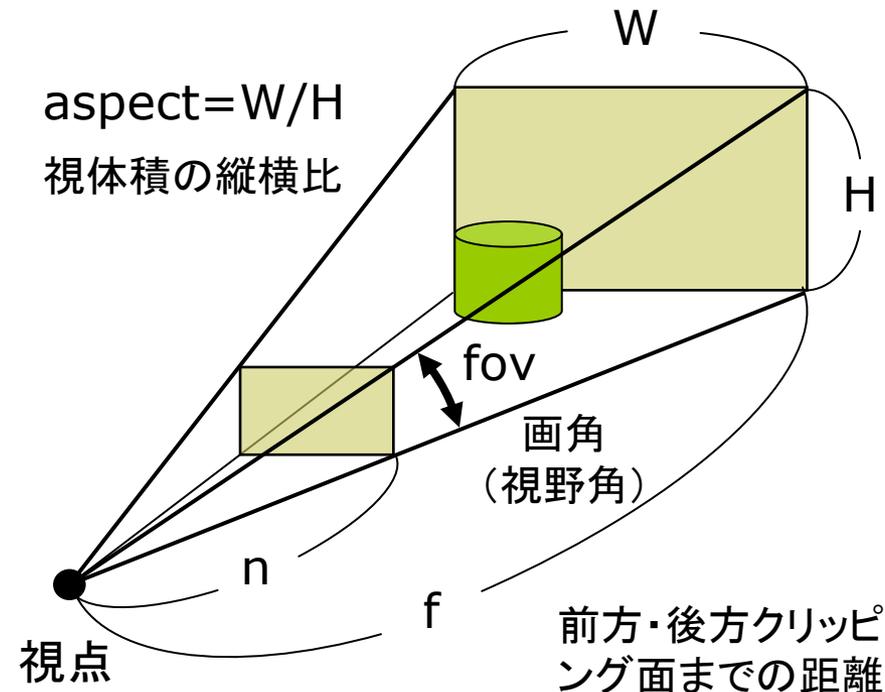
□ ここまでの座標変換の合成

$$P_{proj} = M_{proj} M_{view} M_{model} P_{local} \quad 7$$

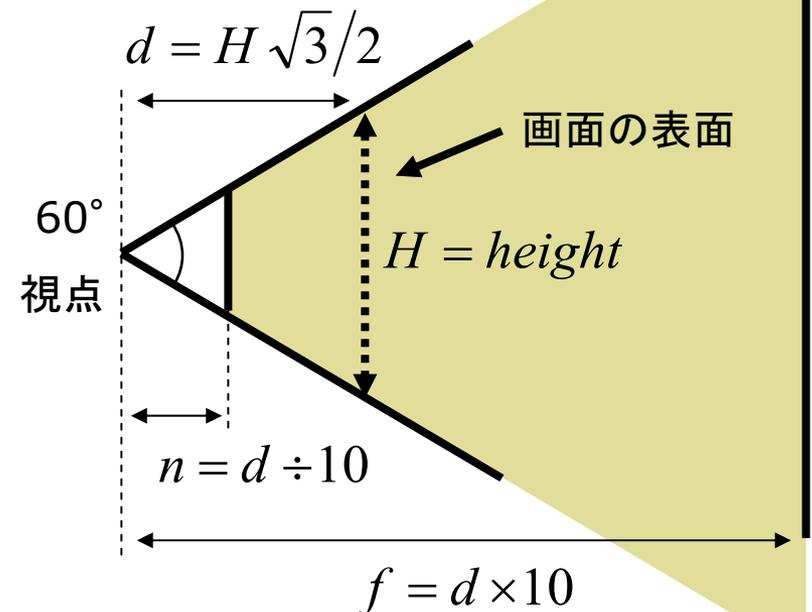
9.7 透視投影関数

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



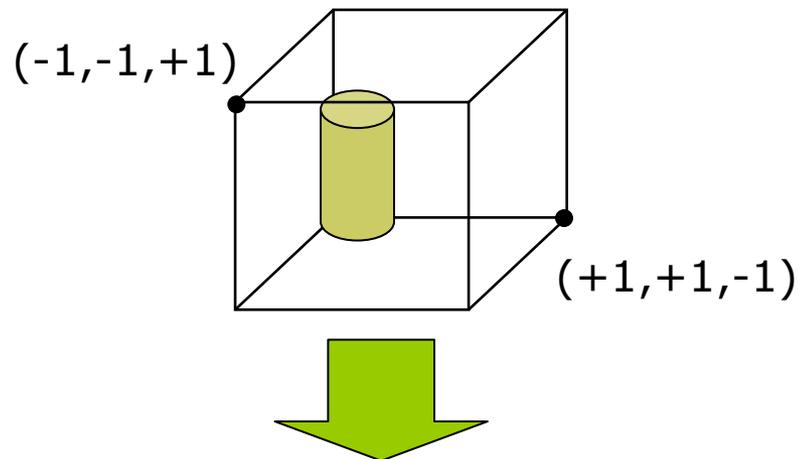
- Processingのデフォルト設定
 - perspective()を呼ばない場合
 - または、引数なしで呼んだ場合
 - 画角(視野角) = 60° ($\pi / 3$)
 - aspect = width / height



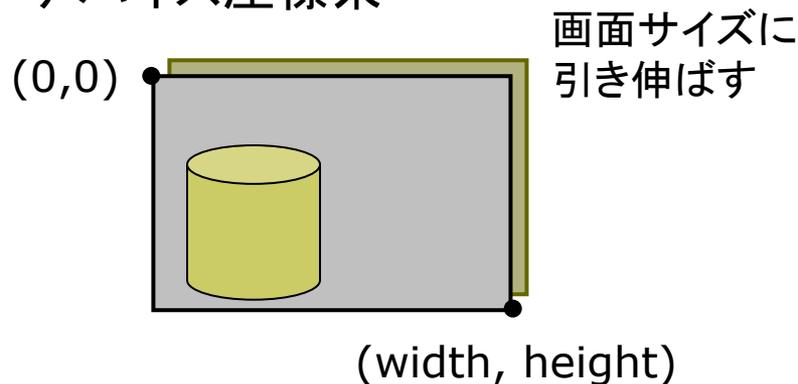
9.8 ビューポート変換とクリッピング

ビューポート変換 (p.42)

□ 正規化視体積



□ デバイス座標系



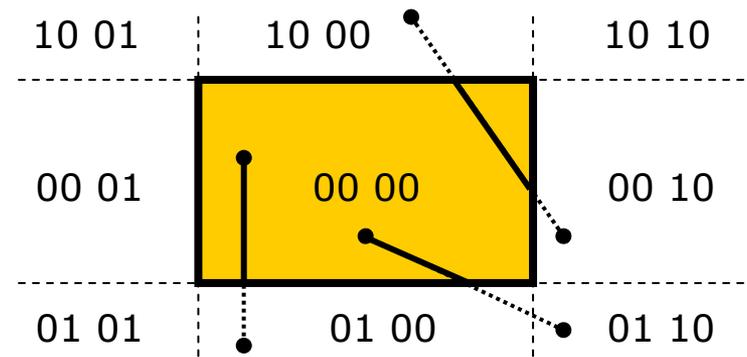
クリッピング (p.44)

□ 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

9.9 隠面消去(1)

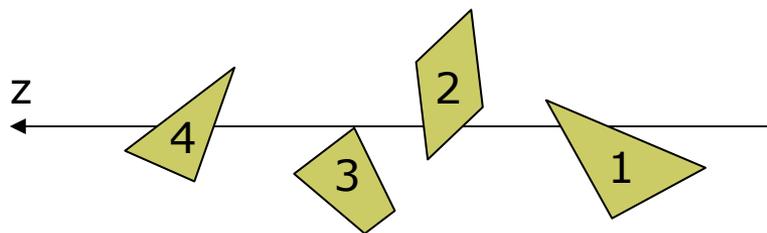
隠面消去(隠線・隠面処理)

□ 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

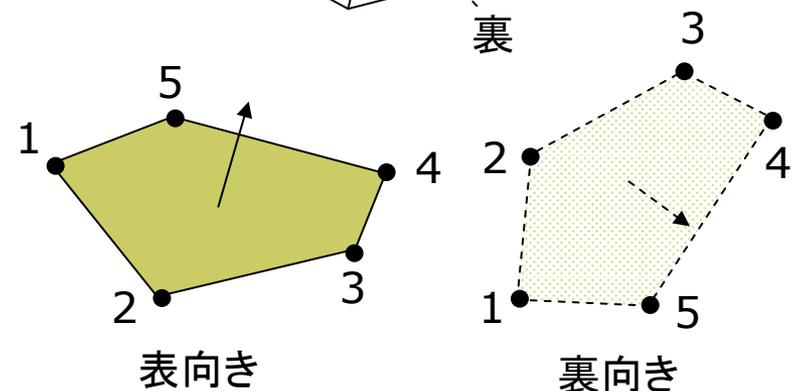
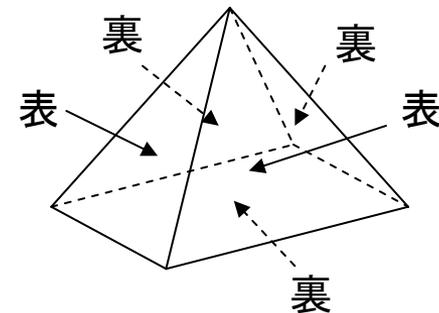
□ 奥行きソート法(p.102)

- 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.100)

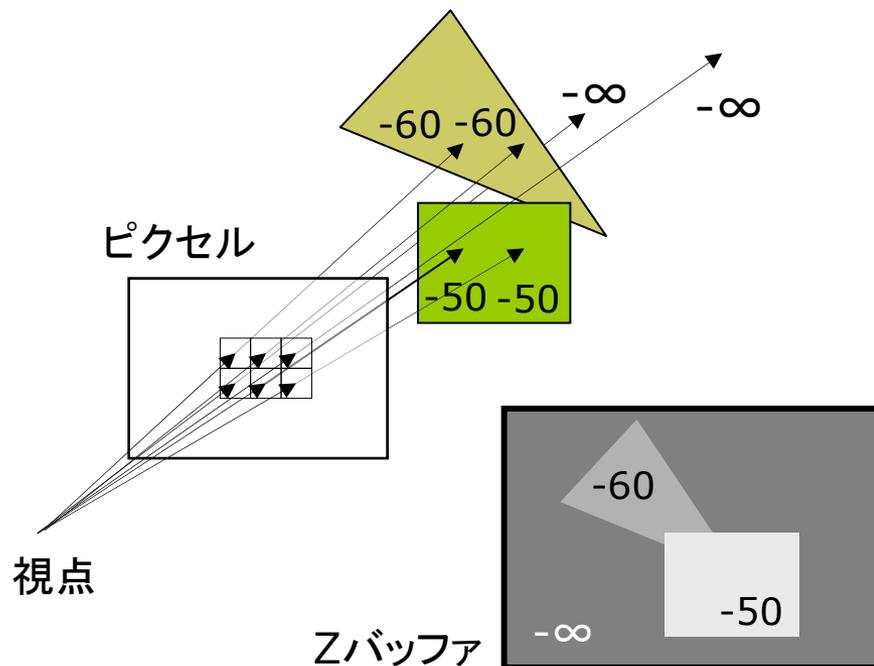
- ポリゴンに表裏を設定し, 裏側を向いているポリゴンを描画しない
- 表裏はポリゴン作成時の頂点の順序(右回り・左回り)で指定
- 1つの凸多面体上の隠面消去



9.10 隠面消去(2)

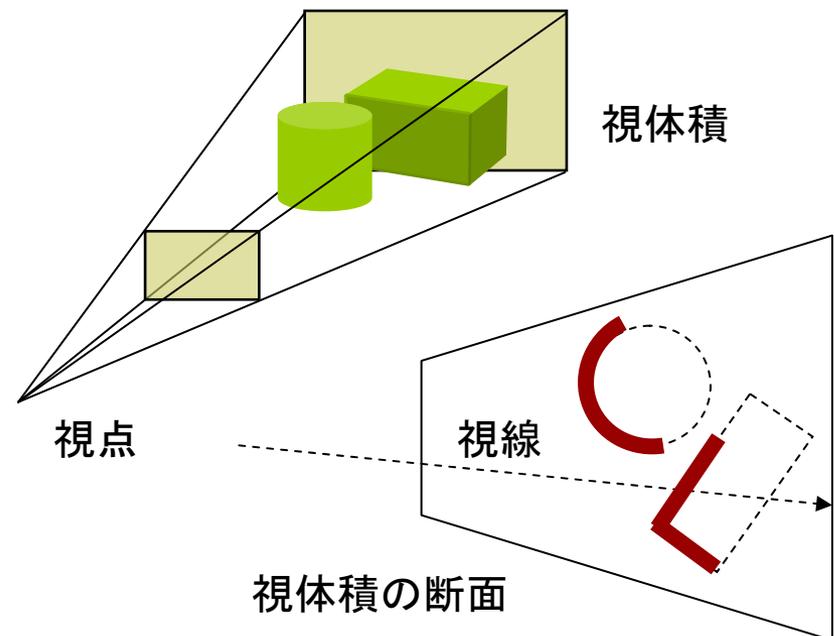
□ Zバッファ法(p.107)

- 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
- 半透明の重なり処理に難点



□ スキャンライン法(p.105)

- 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



9.11 演習課題

課題

問1) 9.12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

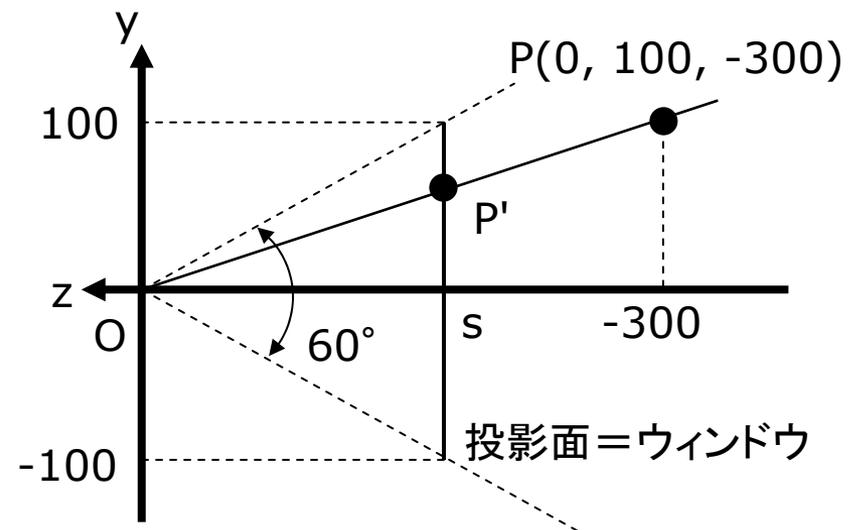
- ヒント: perspective

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



9.12 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();
}
```