コンピュータグラフィックス基礎

第3回 3次元の座標変換

金森 由博

学習の目標

- 3次元での座標変換について理解する
- 3次元空間の物体を2次元のスクリーンに投影するための透視投影変換を理解する（行列を含む）
- 3次元物体をスクリーンに表示するプログラムを作成できるようになる

座標変換の式（1/2）

- 拡大縮小

$$x' = s_x x$$
$$y' = s_y y$$
$$z' = s_z z$$

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

2次元の時と同様に、1つ次数の多いベクトルと行列の演算で表現する (**同次座標**または**斉次座標**)

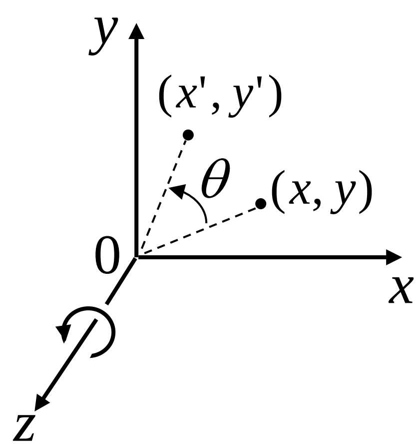
座標変換の式(2/2)

- 平行移動

$$
\begin{aligned}
x' &= x + t_x \\
y' &= y + t_y \\
z' &= z + t_z
\end{aligned}
$$

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

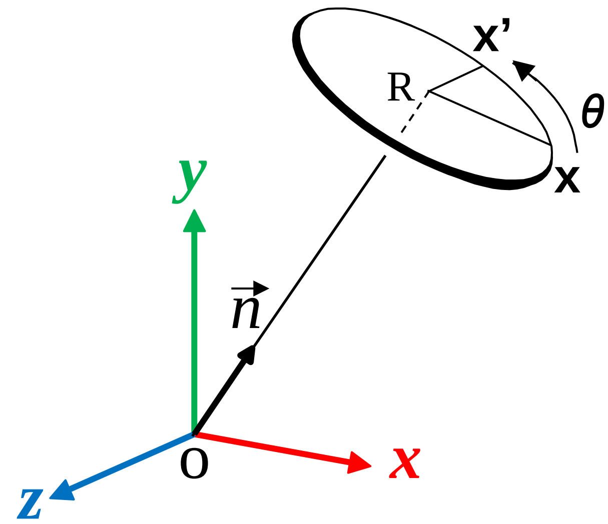
座標軸周りの回転



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

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

参考：任意軸周りの回転



ロドリゲスの公式で回転行列 **R** を計算

$$\mathbf{R} = \mathbf{I} + \sin\theta\,\mathbf{K} + (1-\cos\theta)\,\mathbf{K}^2 \qquad \mathbf{K} = \begin{pmatrix} 0 & -nz & n_y \\ n_z & 0 & -nx \\ -ny & n_x & 0 \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} n_x^{\,2}(1-\cos\theta)+\cos\theta & n_xn_y(1-\cos\theta)-n_z\sin\theta & n_xn_z(1-\cos\theta)+n_y\sin\theta \\ n_xn_y(1-\cos\theta)+n_z\sin\theta & n_y^{\,2}(1-\cos\theta)+\cos\theta & n_yn_z(1-\cos\theta)-n_x\sin\theta \\ n_xn_z(1-\cos\theta)-n_y\sin\theta & n_yn_z(1-\cos\theta)+n_x\sin\theta & n_z^{\,2}(1-\cos\theta)+\cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- 3次元の座標変換は4x4 の変換行列で表現される

- OpenGLでの座標変換は4x4の行列で扱われる

- 以下の関数を使用すれば行列を直接指定しなくてよい

拡大縮小　`glScaled( sx, sy, sz);`

回転移動　`glRotated( theta, nx, ny, nz);`

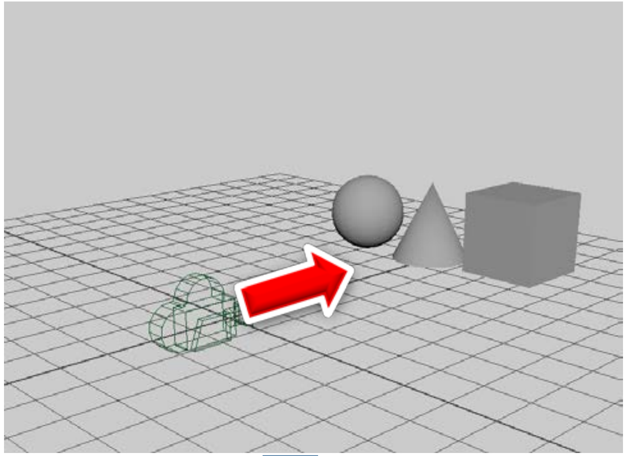
回転角度（0～360度）　回転軸ベクトル

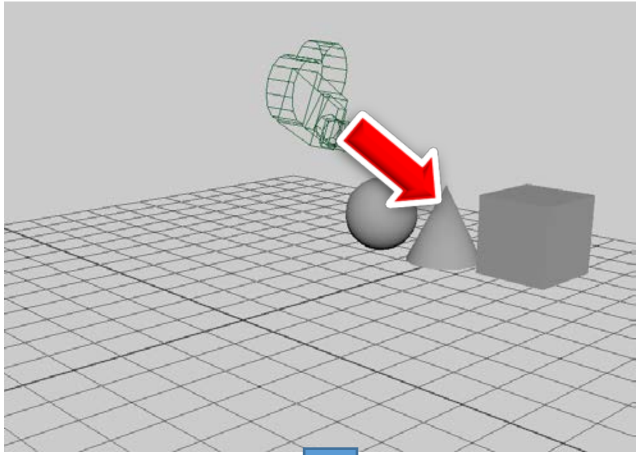
平行移動　`glTranslated( tx, ty, tz);`

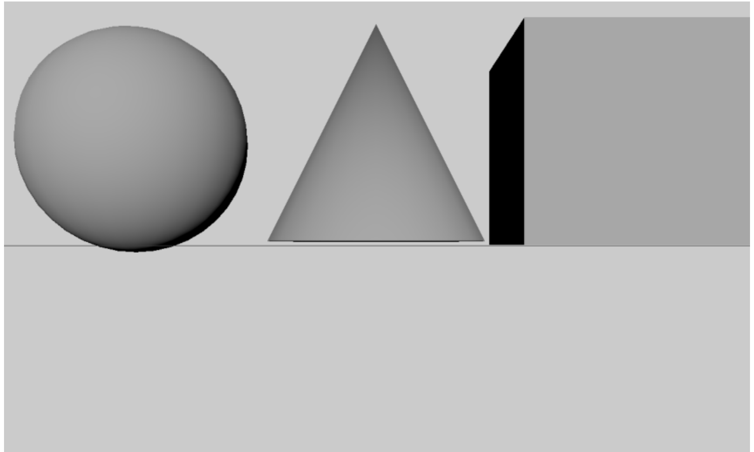
座標変換

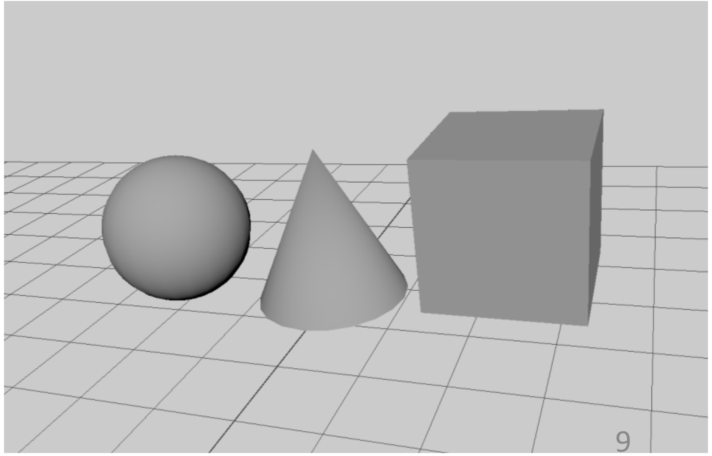
「どこから見るか」

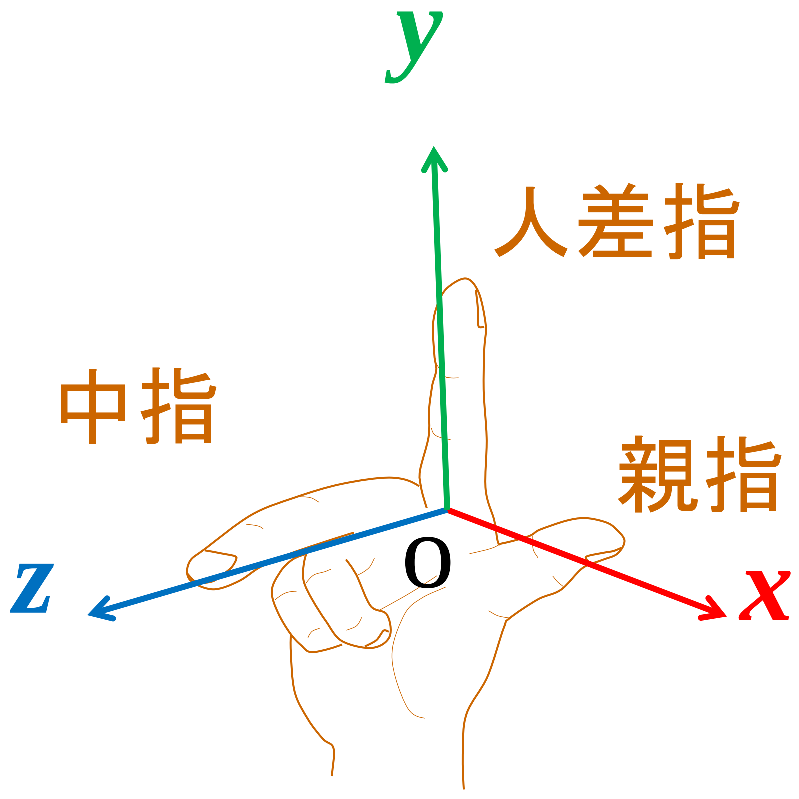
- 視点 (カメラ) の位置・姿勢で見え方が異なる







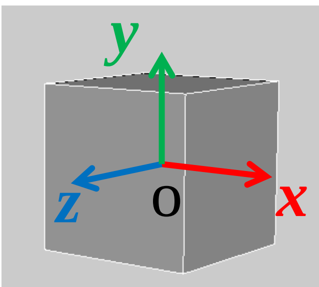




右手座標系

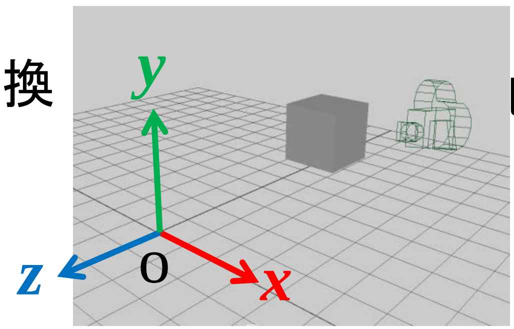
座標系の変換

モデル座標系
(ローカル座標系)



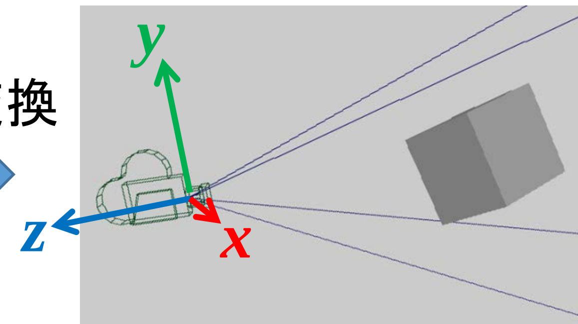
モデル変換

ワールド座標系



ビュー変換

ビュー座標系
（カメラ座標系）



物体ごとの座標系
<u>物体の基準点が原点</u>

物体とカメラの
<u>共通の基準点が原点</u>

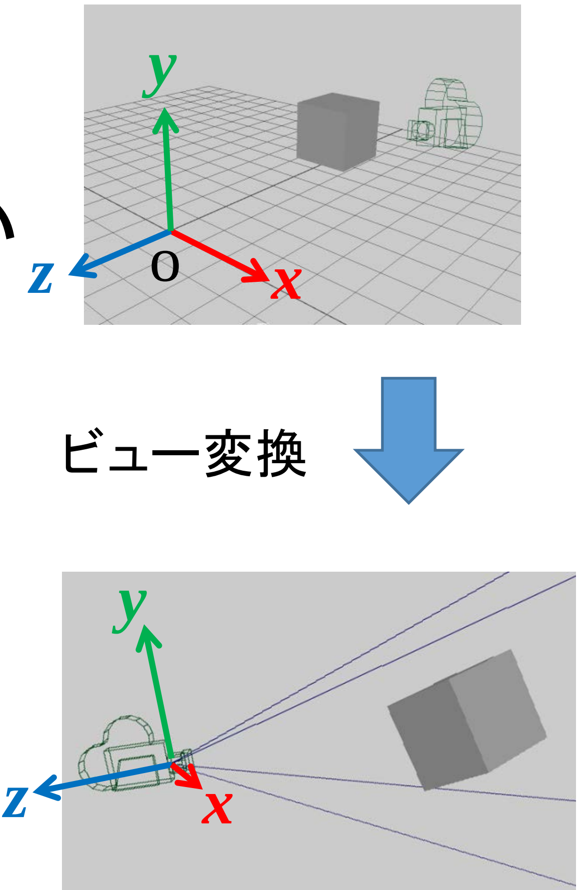
<u>カメラの位置が原点</u>
カメラの向きは $-z$ 方向

モデルビュー変換

座標系の変換は4x4の行列を掛けることで実現される

ビュー変換 (Viewing Transform)

- ワールド座標系で表される物体の位置をカメラ座標系（カメラの位置が原点、カメラの向きが-z方向）で表す
- 次のような4x4 行列 M を掛ければよい
  行列 M を掛けるとカメラの位置が原点に、カメラの向きが -z 方向になる
- このような行列を自分で計算するのは大変なので gluLookAt() が便利

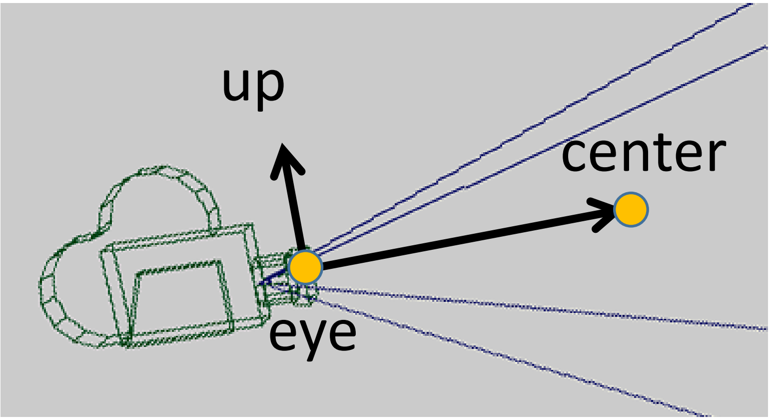


gluLookAt によるビュー変換行列

すべて
ワールド座標

```
gluLookAt(eyex, eyey, eyez,                 // 視点位置
          centerx, centery, centerz,  // 注視点
          upx, upy, upz)                      // up ベクトル
```

通常 (0, 1, 0) を指定

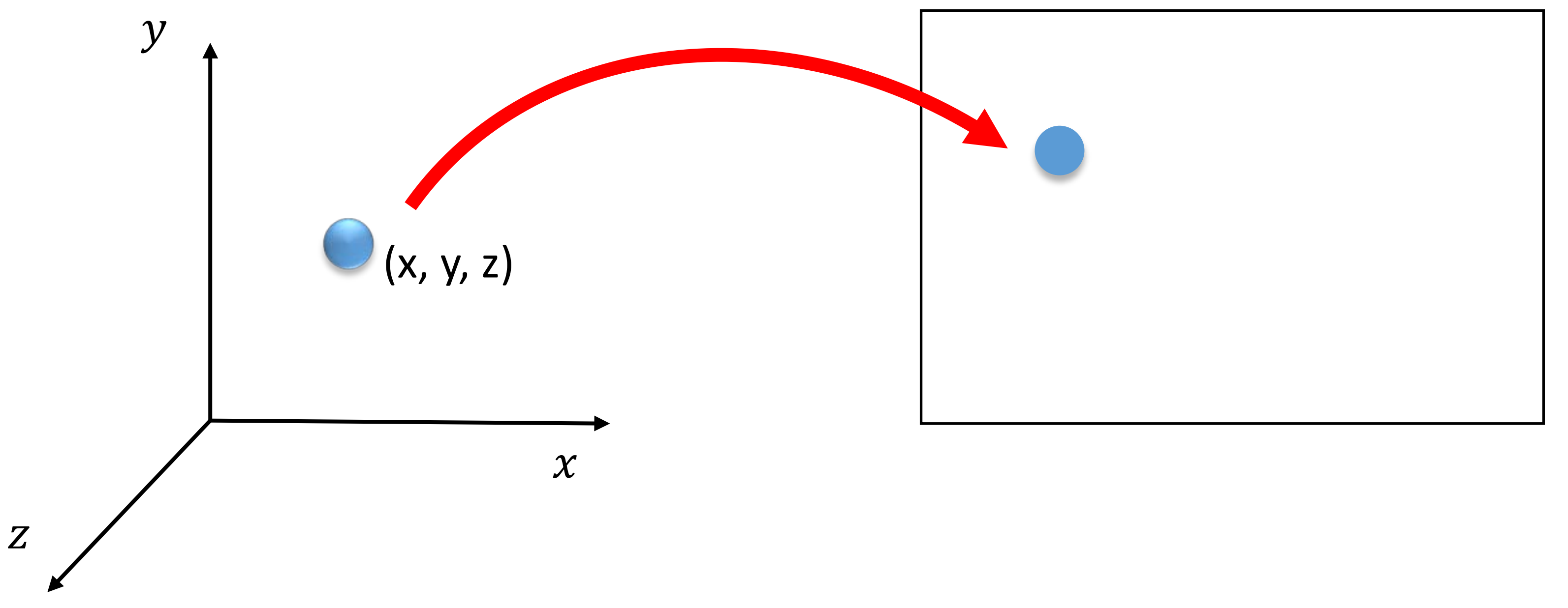


投影変換

立体をどのように平面スクリーンに投影するか

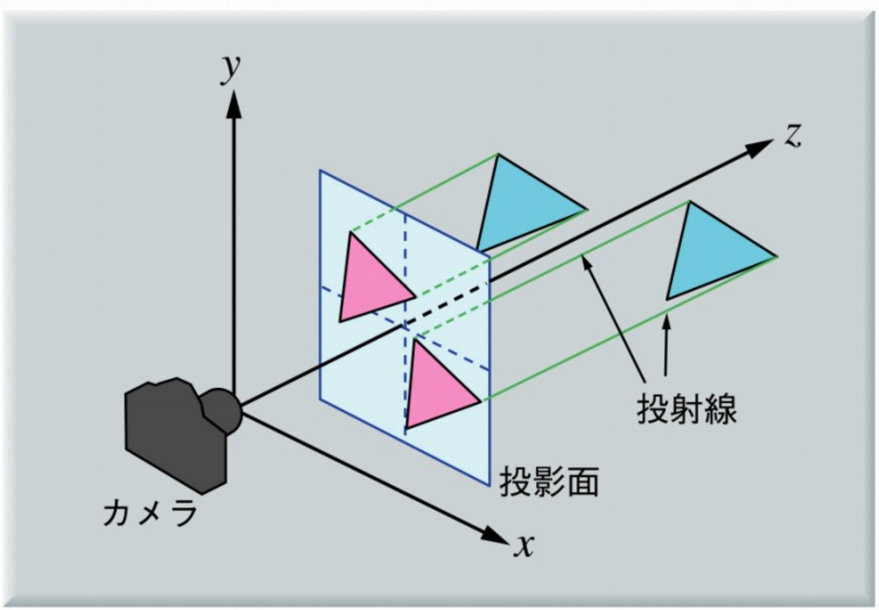
投影変換

点(x, y, z) が、スクリーン上のどこに投影されるか？



平行投影

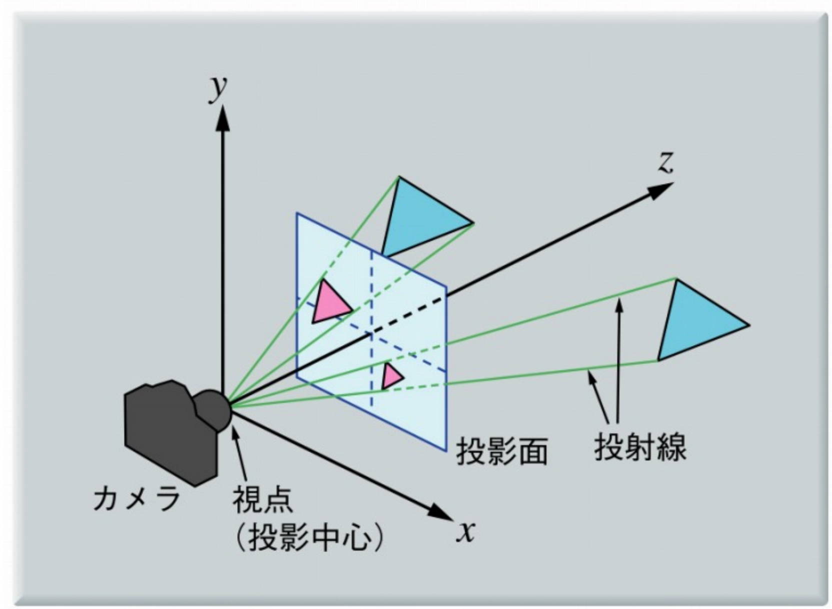
■図2.29——平行投影の原理



「コンピュータグラフィックス」2004年 / 財団法人画像情報教育振興協会（CG-ARTS協会）

透視投影

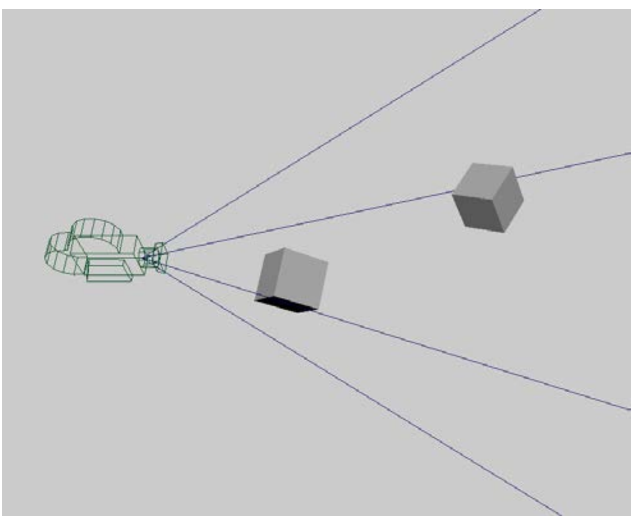
■図2.27——透視投影の原理

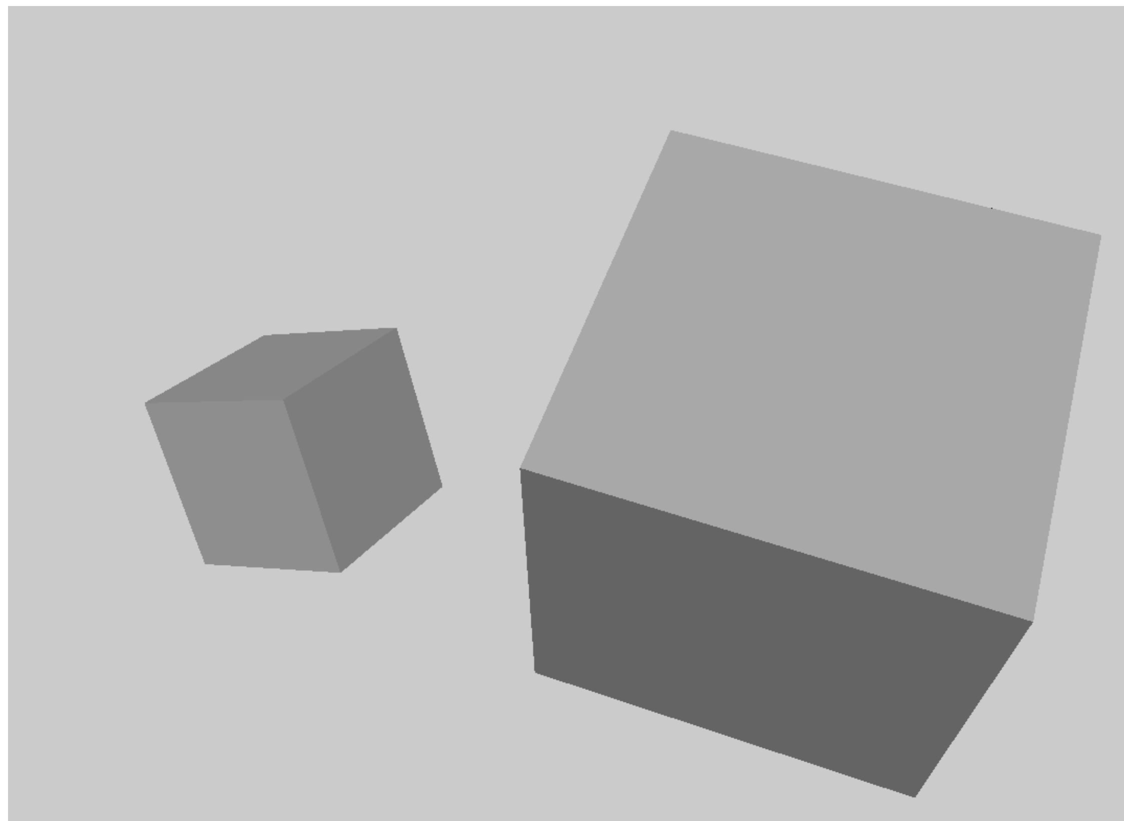


「コンピュータグラフィックス」2004年 / 財団法人画像情報教育振興協会（CG-ARTS協会）

透視投影 (Perspective Projection)

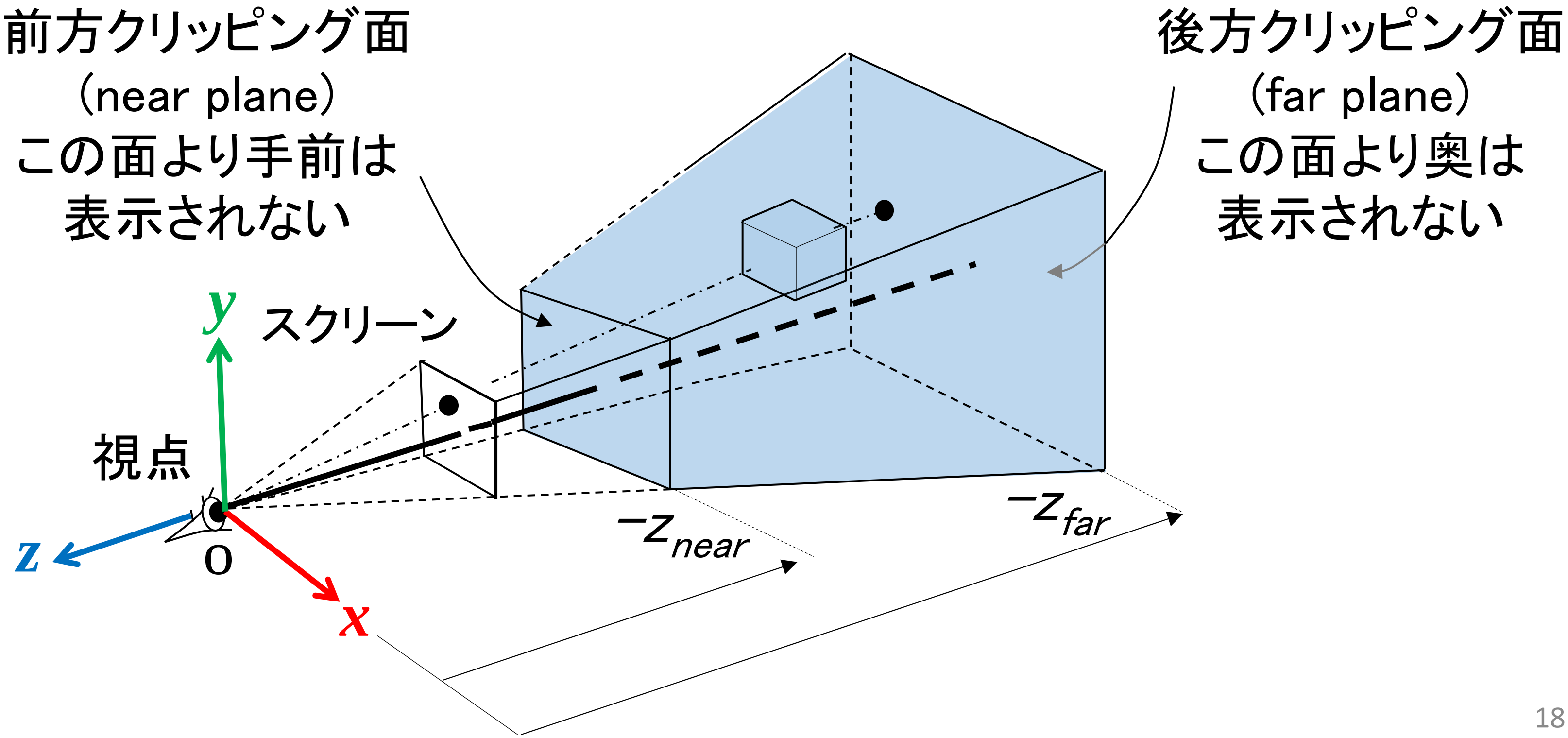
- 透視投影の性質
  - 遠くのものは小さく、手前のものは大きく表示される
  - 直線は直線のまま保たれる



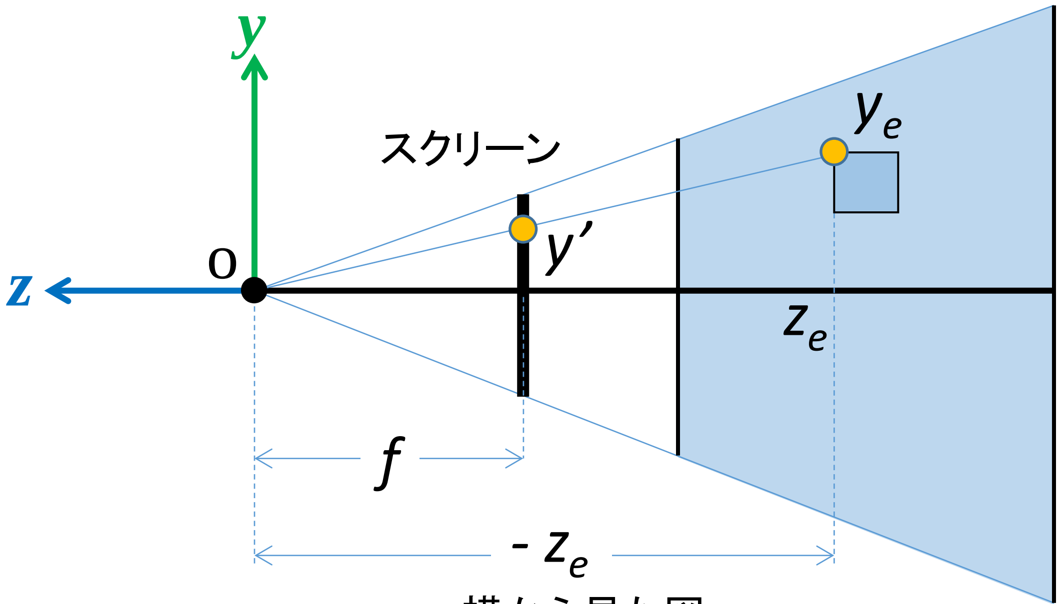


ビューボリューム (Viewing Volume)

- スクリーンに映る範囲を指定
- ビューボリューム（視野錘台）という四角錐台の形をした空間のみが最終的に画面に表示される



カメラ (原点) からスクリーンまでの距離を f として
ビュー座標 $(x_e, y_e, z_e, 1)^T$ の投影位置を考える



横から見た図

(z_e は負の値)

$$f : (-z_e) = y' : y_e \quad \Rightarrow \quad y' = -f\frac{y_e}{z_e}$$

x 座標についても同様にして

$$x' = -f\frac{x_e}{z_e}$$

$-z_e$ で除算

透視投影の計算

- 単に 4×4 行列を掛けても $-z$ での除算は表せない (非線形な操作)
- 同次座標の導入
- 同次座標では w 座標で割る前/割った後を同一であると見なす（w=0で無限遠点を表現できる）

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \equiv \begin{pmatrix} x/w \\ y/w \\ z/w \\ 1 \end{pmatrix}$$

例えば、同次座標を用いた次のような演算で、非線形な座標変換が行える

実際の値は、もっと複雑な形になる（のちほど説明）

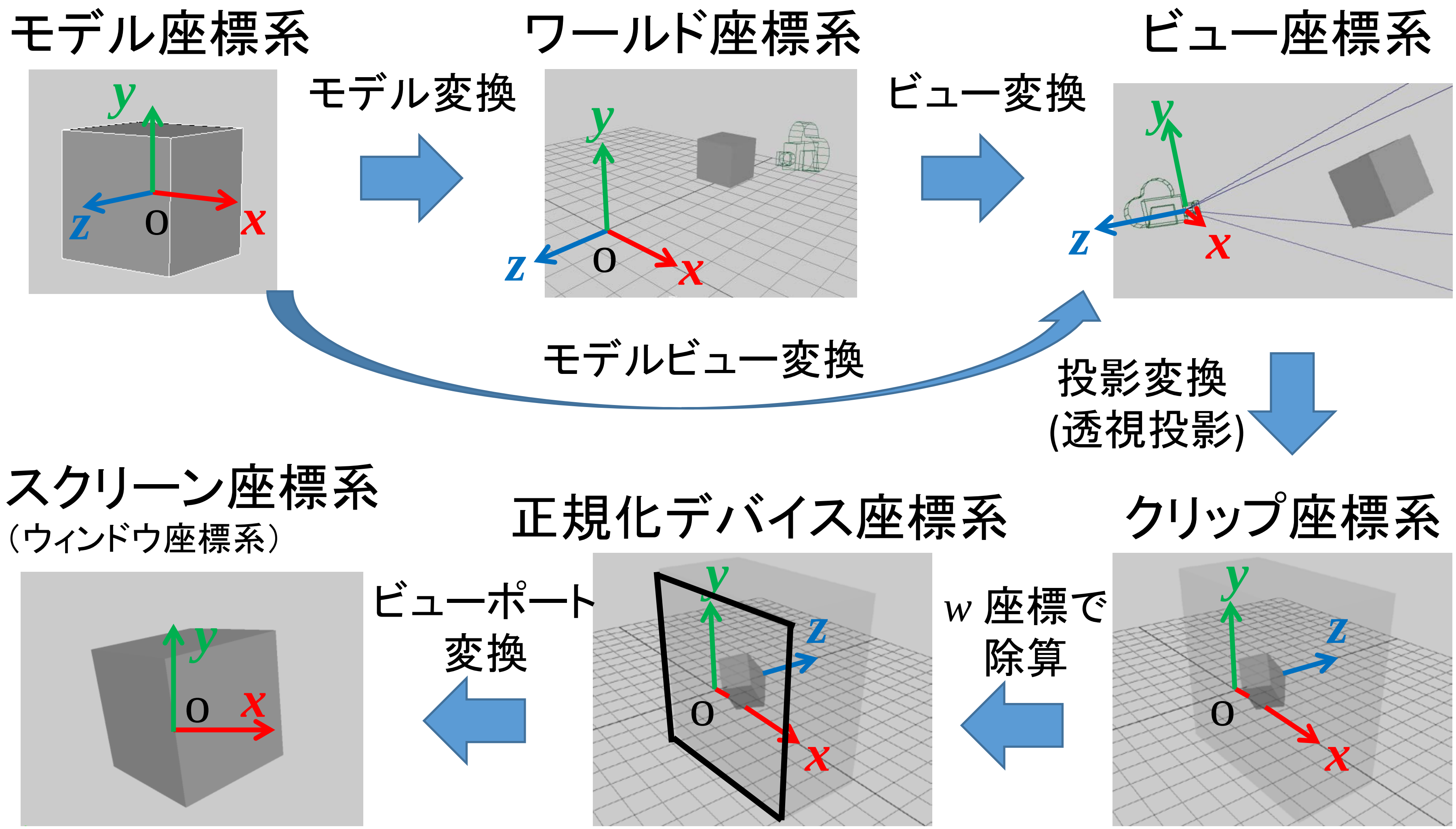
$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ 1 \end{pmatrix} = \begin{pmatrix} x_e \\ y_e \\ z_e + 1 \\ z_e \end{pmatrix} \equiv \begin{pmatrix} x_e/z_e \\ y_e/z_e \\ 1 + 1/z_e \\ 1 \end{pmatrix}$$

透視投影のための要素

すべての要素をz_eで割る

※ スクリーン上での座標は(x’ y’) がわかればよいが、
奥行き情報が必要なため（最も手前の物体だけが見える） z’ の計算も必要になる

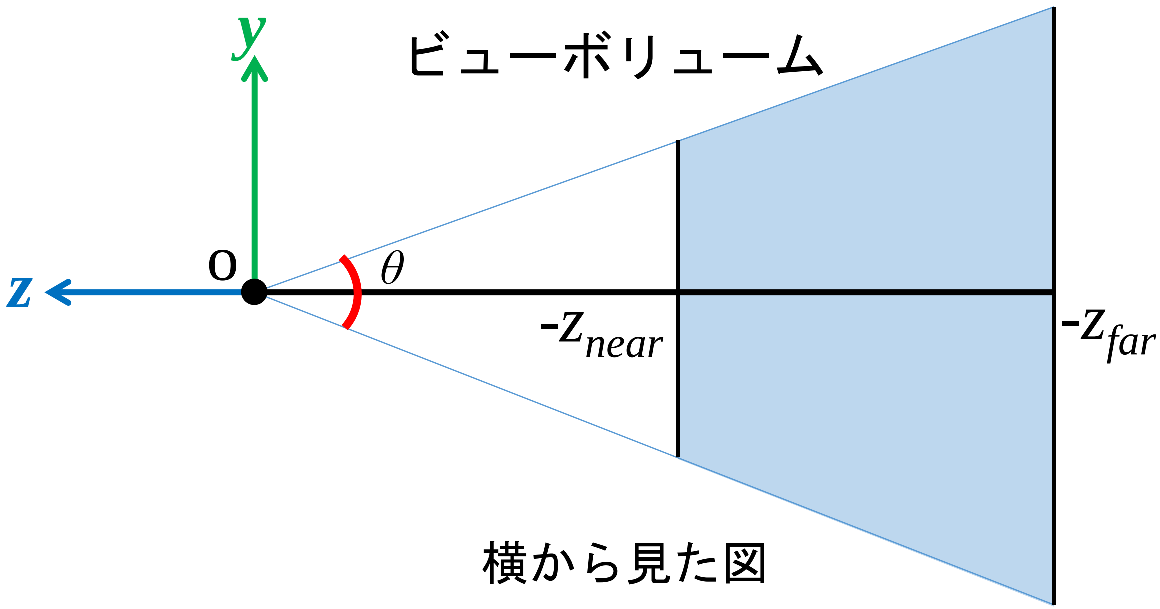
OpenGL の座標系と座標変換



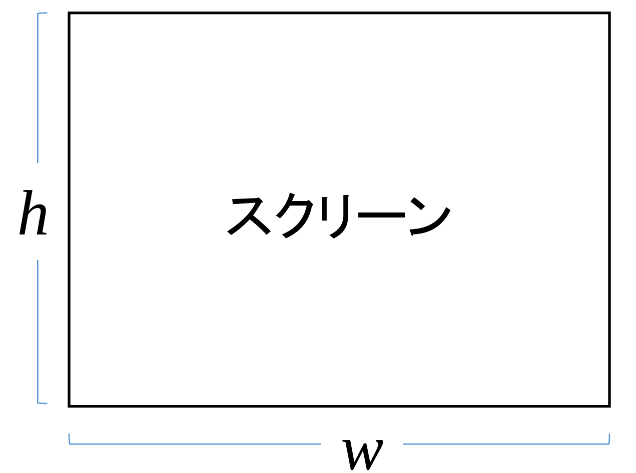
OpenGL での投影行列の指定

- 透視投影には gluPerspective() が便利

```
gluPerspective(fovy,     // 垂直方向の視野角 θ (度数で指定)
               aspect,   // アスペクト比 (スクリーンの縦横比)
               znear,    // near plane の z 座標 (正の値)
               zfar)     // far plane の z 座標 (正の値)
```



横から見た図

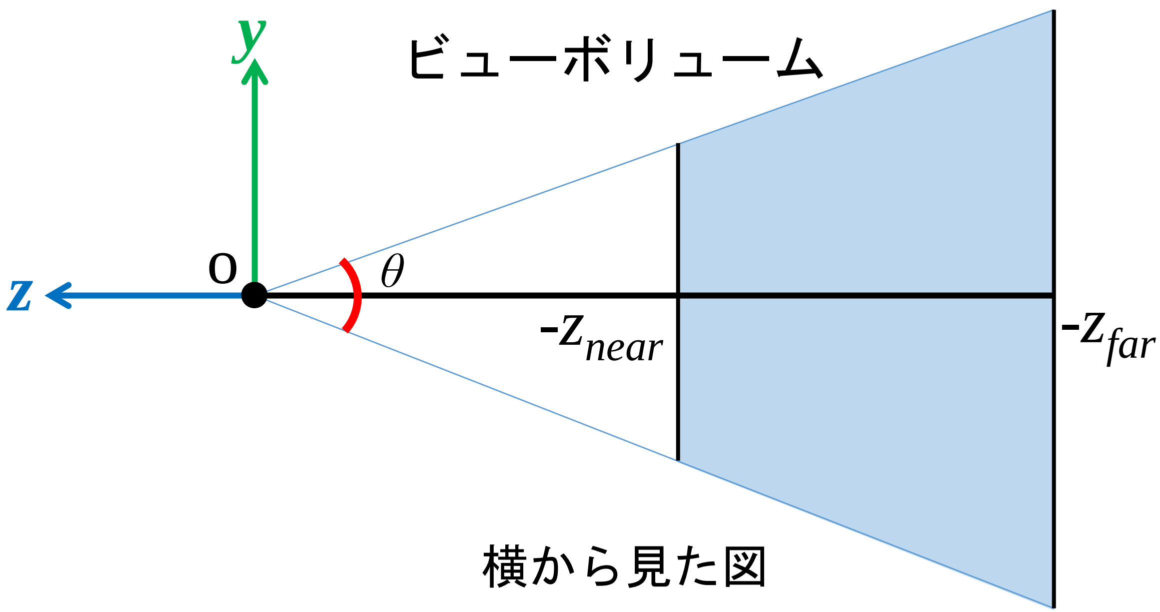


(アスペクト比) = w / h

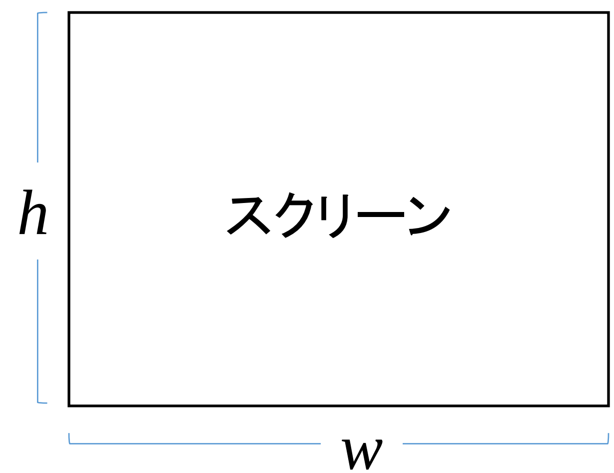
参考: OpenGLでの透視投影行列

- gluPerspective() で指定される透視投影の行列 P

$$P = \begin{pmatrix} \frac{2\, z_{near}}{w} & 0 & 0 & 0 \\ 0 & \frac{2\, z_{near}}{h} & 0 & 0 \\ 0 & 0 & -\frac{z_{far} + z_{near}}{z_{far} - z_{near}} & -\frac{2\, z_{far}\, z_{near}}{z_{far} - z_{near}} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

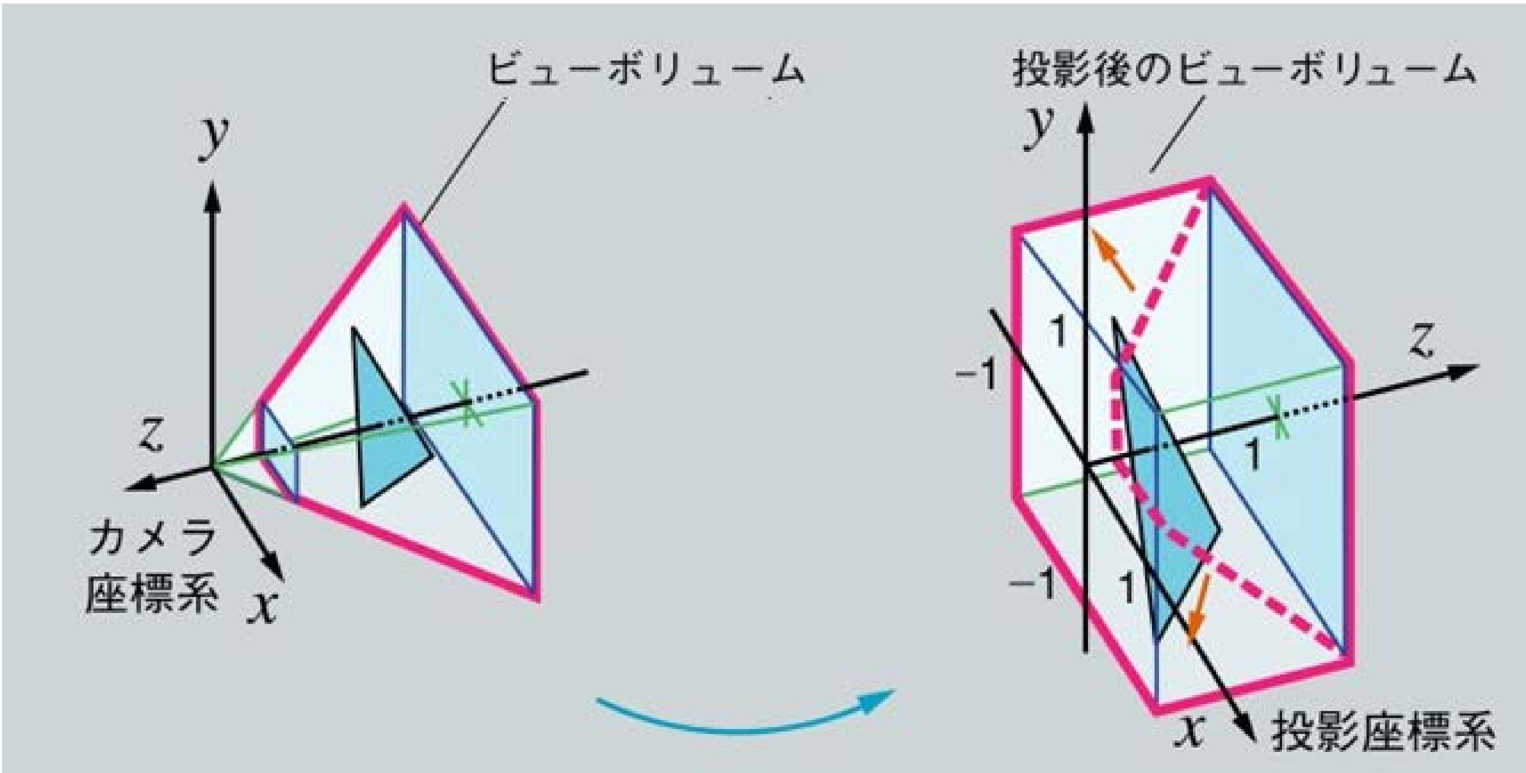


横から見た図



(アスペクト比) = w / h

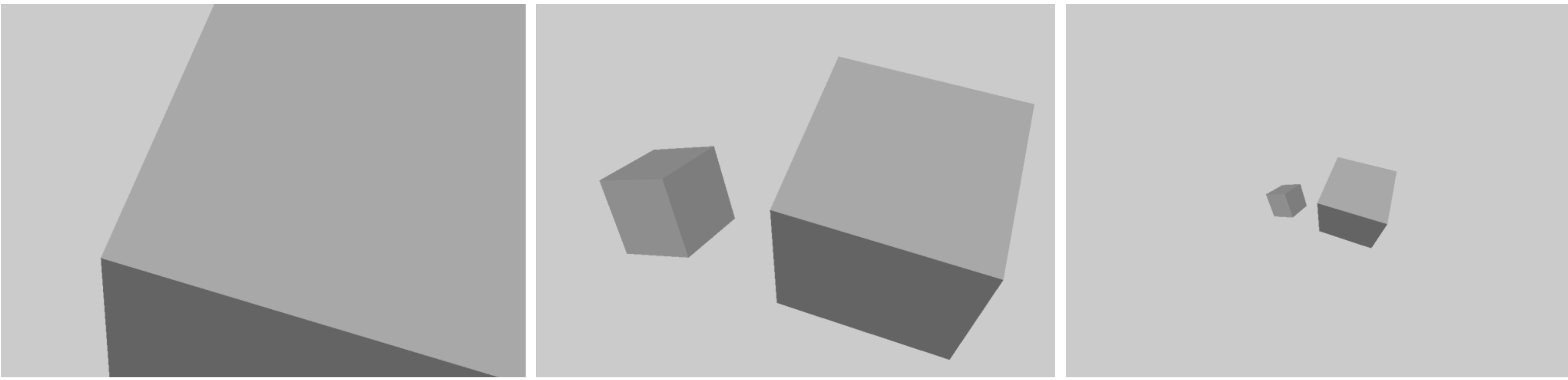
透視投影の計算

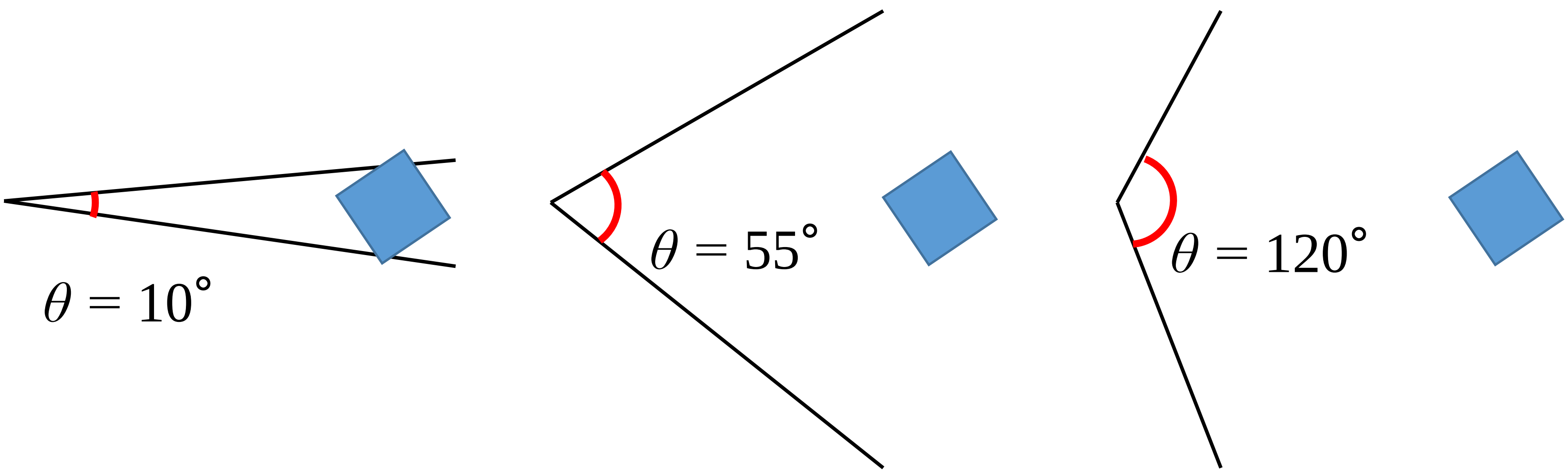


※右手座標系から左手座標系に変わっていることに注意 (-z_e で除算するため)

垂直視野角 θ の影響

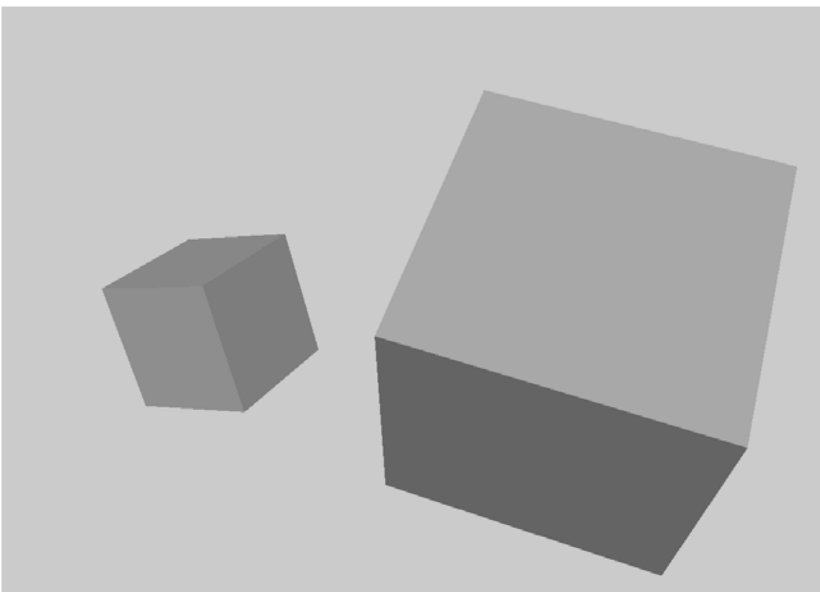
- θ が大きくなるにつれて物体が小さく表示される



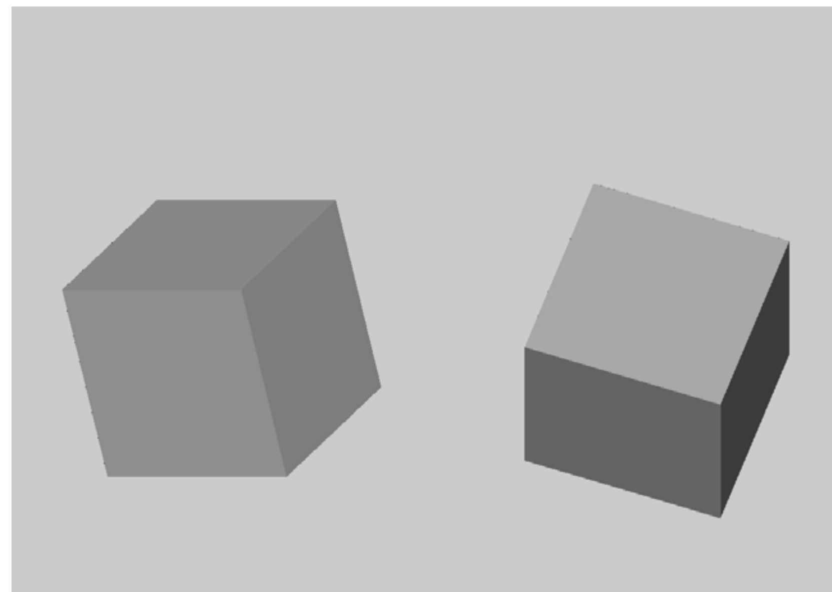


平行投影 (Orthographic Projection)

- 無限遠のカメラで (≒望遠レンズでズームして) 撮影
- 投影された物体の大きさは遠近に依存しない
- OpenGL では glOrtho() や gluOrtho2D() が便利



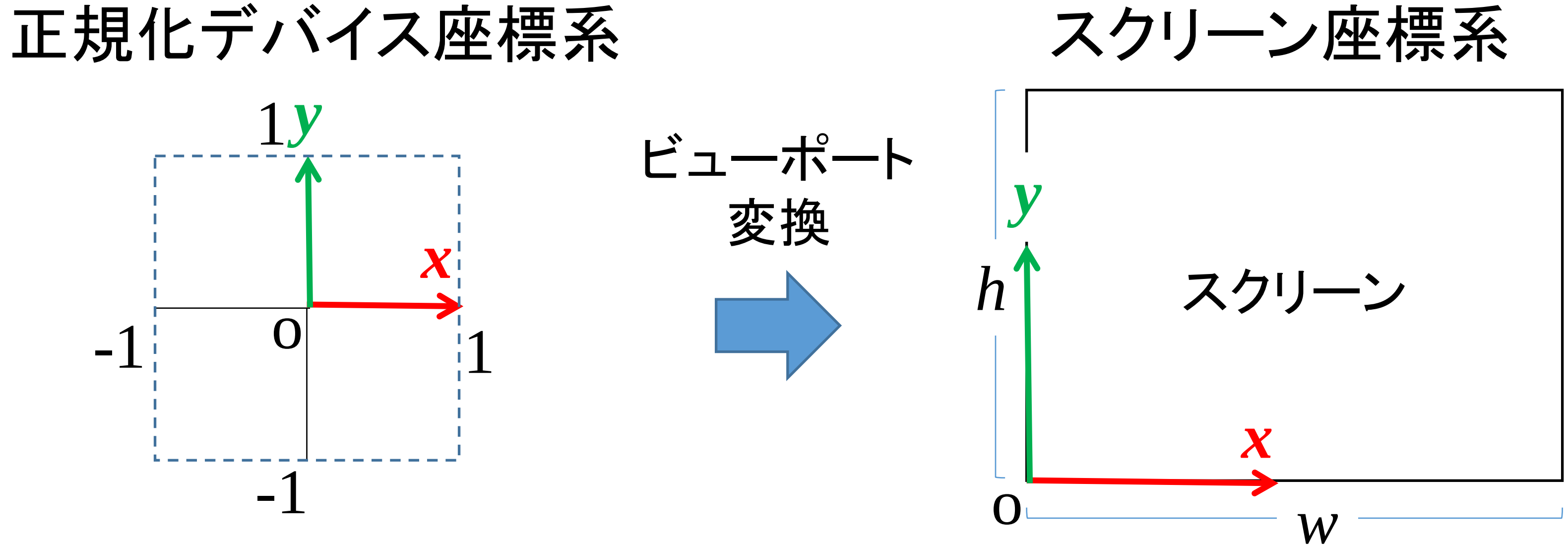
透視投影 (θ = 55°)



平行投影

ビューポート変換

- 正規化デバイス座標系の x, y 座標を スクリーンの大きさに合わせて拡大する

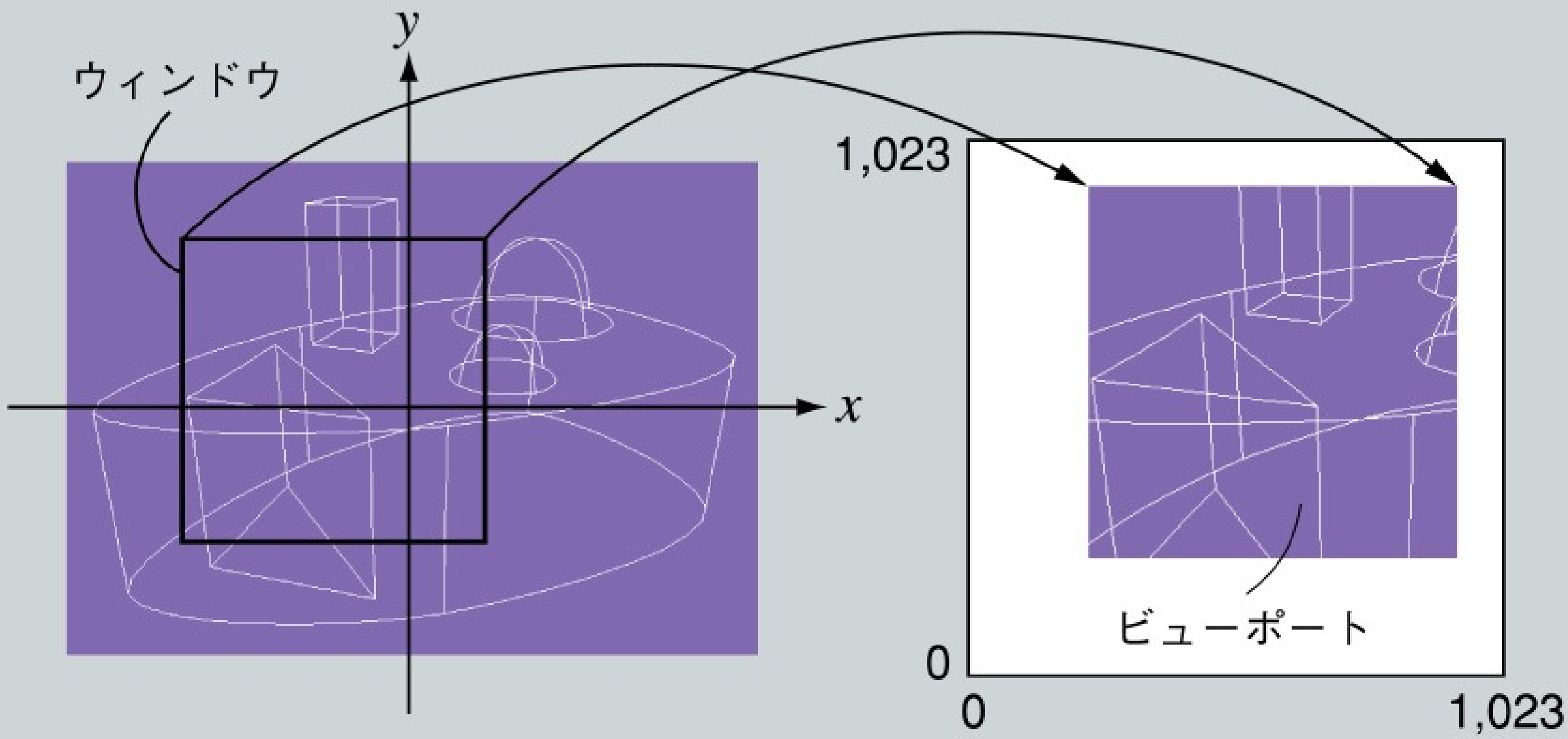


- OpenGL では glViewport() で指定

```
glViewport(x0, y0, w, h)
```

通常はともに 0 を指定

■図2.44——ビューポート変換



[a] 3次元CGでは投影面（xy座標系）
2次元CGではワールド座標系

[b] デバイス座標系（1,024×1,024の場合）

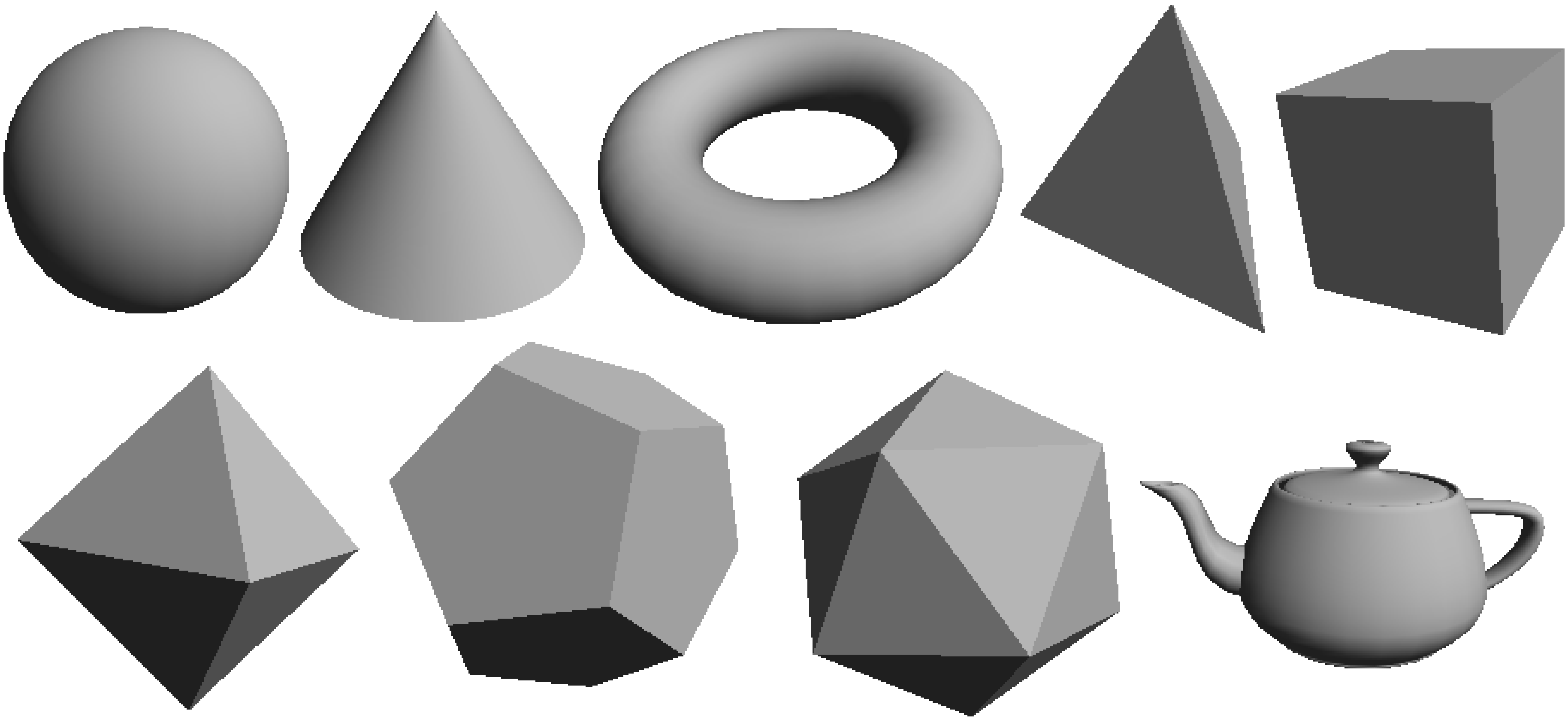
「コンピュータグラフィックス」2004年 / 財団法人画像情報教育振興協会（CG-ARTS協会）

OpenGL での座標変換の指定

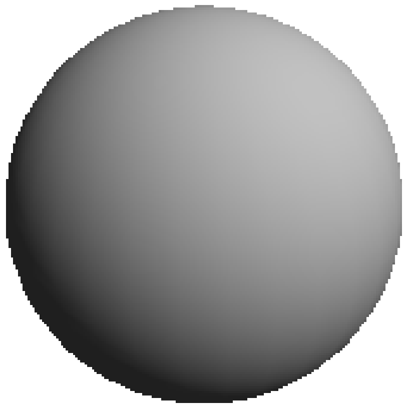
```
glViewport(0, 0, w, h);      // ビューポート変換行列の指定

glMatrixMode(GL_MODELVIEW); // これ以降はモデルビュー変換行列の指定
glLoadIdentity();           // 単位行列を指定
gluLookAt( … );             // カメラの位置・姿勢の行列を乗算

glMatrixMode(GL_PROJECTION); // これ以降は投影変換行列の指定
glLoadIdentity();            // 単位行列を指定
gluPerspective( … );         // 透視投影の行列を乗算

glBegin(GL_TRIANGLES);       // 描画命令を発行
glVertex3d( … );             // ワールド座標系の座標を指定
…
glEnd();
```

GLUT に予め用意されている立体形状

中心が原点で大きさが固定なのでモデル変換が必要



詳しくは http://opengl.jp/glut/section11.html を参照



void glutSolidSphere(
GLdouble radius,
GLint slices, GLint stacks);

void glutSolidTeapot(GLdouble size);

陰影の計算 (Shading / Lighting)

- glColor3d(…) などで色を指定するとベタ塗りになる

陰影計算**なし**　　　陰影計算**あり**

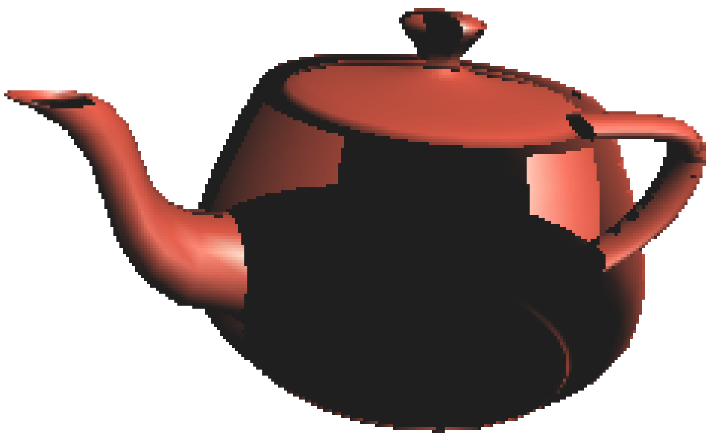
- 法線ベクトルと光源と反射特性の指定が必要
  - GLUT に用意された立体形状なら法線ベクトルは計算済み
- 詳しくは本講義の後半「レンダリング」で学習

陰影の計算 (Shading / Lighting)

```
glEnable(GL_LIGHTING); // 陰影計算を有効化
glEnable(GL_LIGHT0);   // 光源 0 を有効化 (1, 2, …も指定可)
// 以下、光源のパラメータを設定
glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmbient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDiffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, lightSpecular);
glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);

// 以下、物体の反射特性を指定
glMaterialfv(GL_FRONT, GL_AMBIENT, ambientColor);
glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuseColor);
glMaterialfv(GL_FRONT, GL_SPECULAR, specularColor);
glMaterialfv(GL_FRONT, GL_SHININESS, &shininess);

glBegin(GL_TRIANGLES);      // 描画命令を発行 (以下略)
…
```

隠れ面消去 (Hidden Surface Removal)

- 手前の面に隠される奥の面を除外する処理を「隠れ面消去」という



隠れ面消去**なし**

隠れ面消去**あり**

- OpenGL では glEnable(GL_DEPTH_TEST) を指定
- 詳しくは本講義の後半「レンダリング」で学習

課題の説明

- ティーポットメリーゴーラウンド
- ティーポットをゆっくり上下させながら回転させる
- 視点も移動させる