Chapter 11

同次座標と行列計算

11.1 同次座標と射影変換

第 5 章では，2 次元グラフィクスにおけるアフィン変換の利用法について説明した．3 次元グラフィクスでもモデル変換や投影変換などの，様々な座標変換が行なわれる．ただし，3 次元グラフィクスでは，透視投影などの処理も統一的に処理できる同次座標表現と，その行列計算が多用される．OpenGL においても，同次座標表現が利用されている．

同次座標 (homogeneous coordinates) とは，実座標空間の次元を 1 つ上げて，同値関係を導入した座標表現である．3 次元グラフィクスの対象となる 3 次元空間 $\boldsymbol{p}^{(3)} = [x^{(3)}\ y^{(3)}\ z^{(3)}]^{\mathsf{T}}$ の場合には，4 次元座標 $\boldsymbol{p} = [x\ \ y\ \ z\ \ w]^{\mathsf{T}}$ と以下の同値関係 $\sim$ とで構成される．

$$\boldsymbol{p}^{(3)} \sim \boldsymbol{p} \Leftrightarrow x^{(3)} = \frac{x}{w},\ y^{(3)} = \frac{y}{w},\ z^{(3)} = \frac{z}{w},\quad \text{ただし } w \neq 0$$

すなわち，2 つの同次座標 $[x\ \ y\ \ z\ \ w]^{\mathsf{T}}$ と $[\frac{x}{w}\ \ \frac{y}{w}\ \ \frac{z}{w}\ \ 1]^{\mathsf{T}}$ とは同値であり，3 次元実空間の座標 $[x^{(3)}\ y^{(3)}\ z^{(3)}]^{\mathsf{T}}$ に対応する．

同次座標を用いると，スケール変換 (拡大縮小) や回転変換だけでなく，平行移動や透視投影も 4×4 行列で表現可能である．同次座標を利用した 4×4 行列による変換は，**射影変換** (projective transformation) と呼ばれる変換に相当する．射影変換の基本変換には，アフィン変換の基本変換であるスケール変換と回転変換，平行移動に，透視投影 (perspective projection) が加わる．これらの基本変換の表現は，次のようになる．

- スケール変換：　x, y, z の各成分ごとに，s_x, s_y, s_z 倍する．

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

- 回転変換：　各座標軸ごとに θ 回転する．右手座標系の場合には，軸の正の側から見て反時計回りが正の回転となる．

x 軸回り

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

$$y\text{ 軸回り}\qquad \boldsymbol{p}'=\begin{bmatrix}x'\\y'\\z'\\1\end{bmatrix}=M\boldsymbol{p}=\begin{bmatrix}\cos\theta & 0 & \sin\theta & 0\\0 & 1 & 0 & 0\\-\sin\theta & 0 & \cos\theta & 0\\0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}=\begin{bmatrix}x\cos\theta+z\sin\theta\\y\\-x\sin\theta+z\cos\theta\\1\end{bmatrix}$$

$$z\text{ 軸回り}\qquad \boldsymbol{p}'=\begin{bmatrix}x'\\y'\\z'\\1\end{bmatrix}=M\boldsymbol{p}=\begin{bmatrix}\cos\theta & -\sin\theta & 0 & 0\\\sin\theta & \cos\theta & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}=\begin{bmatrix}x\cos\theta-y\sin\theta\\x\sin\theta+y\cos\theta\\z\\1\end{bmatrix}$$

- 平行移動： x, y, z の各成分ごとに，t_x, t_y, t_z だけ平行移動する．

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

- 透視投影： 特定の軸についてスケールを変える．通常は，z 座標に応じて x 座標と y 座標のスケールを反比例させる．

$$z\text{ 軸方向}\qquad \boldsymbol{p}'=\begin{bmatrix}x'\\y'\\z'\\1\end{bmatrix}=M\boldsymbol{p}=\begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 1 & -\frac{1}{d}\\0 & 0 & \frac{1}{d} & 0\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}=\begin{bmatrix}x\\y\\z-\frac{1}{d}\\\frac{z}{d}\end{bmatrix}\sim\begin{bmatrix}\frac{xd}{z}\\\frac{yd}{z}\\d-\frac{1}{z}\\1\end{bmatrix}$$

同次座標を利用することにより，いかなる射影変換も 4×4 行列 M で表現できる．さらに，行列計算には結合法則が成り立つことから，行列の並ぶ順序さえ変えなければ，どこから計算しても結果は変わらない．すなわち，実際に変換する座標値が判明する前に，予め行列部分を計算してもよい．後述するように，OpenGL では行列の積を計算して，合成後の行列のみを保持する．

11.2 PyOpenGL における行列計算

OpenGL の行列計算は，個々の変換行列の積として構成される．ここで，行列計算の最初は**恒等行列** (identity matrix) となる．恒等行列は，対角成分のみがすべて 1 で，残りの成分がすべて 0 となる行列である．恒等行列と任意の行列 (ベクトル) との積は，以下に示すように当該の行列 (ベクトル) そのものとなる．恒等行列は積に関する単位元となることから，単位行列と呼ぶこともある．

$$I\boldsymbol{p}=\begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}=\begin{bmatrix}x\\y\\z\\1\end{bmatrix}=\boldsymbol{p}$$

第 10 章で説明したように，OpenGL では 2 種類の行列 (`GL_PROJECTION` 行列と `GL_MODELVIEW` 行列) しか持たない．すなわち，点 $\boldsymbol{p}$ に対して M_1, M_2, M_3 の順に 3 つの変換を施すような行列 M は，

$$M_3(M_2(M_1\boldsymbol{p}))=(M_3M_2M_1)\boldsymbol{p}=M\boldsymbol{p}$$

であるので，結合法則 (associative property) を用いることで，

$$M=IM_3M_2M_1$$

となる．このとき，OpenGL では行列の計算を上の式の左から右へと順に実行するので，最初に行列 M を恒等行列 I としてから，M_3, M_2, M_1 というように，点に施すのとは逆の順序で積をとっていく．

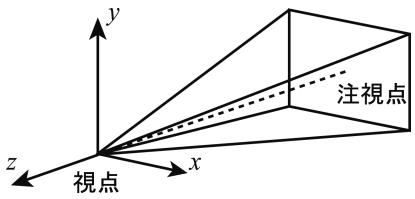

Figure 11.1: 視点座標系

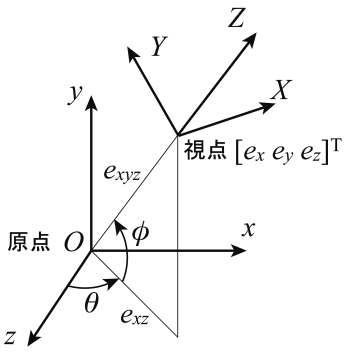

Figure 11.2: 視点座標系への変換

GL ライブラリ： OpenGL 命令のインタフェース

glMultMatrixd　行列の乗算関数
引数として double 型の 1 次元配列で行列を与え，設定対象の行列との積をとって行列を更新する．1 次元配列の順序は列方向が優先されるため，プログラムでは行列が転置した形式で記述されることになる．

glTranslated　平行移動関数
設定対象行列に double 型の 3 引数（平行移動ベクトルの x, y, z 成分）の平行移動を適用する．

glRotated　回転関数
設定対象行列に指定軸回りの回転変換を適用する．回転角の 1 引数（度単位）と回転軸ベクトルの 3 引数（x, y, z 成分）が すべて double 型で与えられる．

11.3 PyOpenGL による変換行列の計算プログラム

例１： 透視投影による立方体の描画 — cubeMatrix.py

第 10 章の例 1 のプログラム cubePosition を拡張しており，実際に実行結果はまったく同じになるが，GL_MODELVIEW 行列を 1 つずつ計算している．視点座標系では，図 11.1 のように，投影面を xy 平面として視線と逆の向きに z 軸が延びている．したがって，図 11.2 に示すようなモデルの座標系 $[x \;\; y \;\; z]^\mathsf{T}$ から視点座標系 $[X \;\; Y \;\; Z]^\mathsf{T}$ への変換は，視点と原点を結ぶ直線に z 軸を合わせる回転変換を施したうえで，視点と原点との距離 e_{xyz} だけの平行移動を行なうことになる．ここで，z 軸を視線の方向に合わせるためには，y 軸に関して角 θ だけ正の向きに回転してから，x 軸に関して角 ϕ だけ負の向きに回転することになる．ただし，OpenGL では，モデルに適用するのとは逆の順序，つまり後から適用される変換行列を先に乗じていくので，プログラム上では最初に平行移動，次に x 軸に関する回転，最後に y 軸に関する回転の順で行列の積がとられている．

回転行列の成分となる θ, ϕ の正弦および余弦の値は，視点の座標 $[e_x \;\; e_y \;\; e_z]^\mathsf{T}$ によって，次のように計算される．

$$\sin\theta = \frac{e_x}{e_{xz}}, \quad \cos\theta = \frac{e_z}{e_{xz}}, \quad \sin\phi = \frac{e_y}{e_{xyz}}, \quad \cos\phi = \frac{e_{xz}}{e_{xyz}},$$
$$\text{ただし}, \quad e_{xz} = \sqrt{{e_x}^2 + {e_z}^2}, \quad e_{xyz} = \sqrt{{e_x}^2 + {e_y}^2 + {e_z}^2}$$

ここで注意すべき点として，2 つの事項が挙げられる．1 つは座標系の回転を考えたことである．座標系に対する変換は，対象に対する変換とは逆になるので，本来は回転の向きを逆にとらなくてはならない．つまり，図 11.3 のように視点が y 軸に関して正の向きに回転することは，視点を固定して立体を y 軸に関して負の向きに回転することに等しい．もう 1 つは，glMultMatrixd の引数が列優先であることである．このためプログラム中では行列が転置したように見える．回転行列の転置行列は，逆回転の行列となる．したがって，これらの 2 つが互いに打ち消しあって，プログラムの上では y 軸に関しては正の向きに角 θ の回転行列，x 軸に関しては負の向きに角 ϕ の回転行列を与えているように見える．10.3 節の例 2「多角形としての立方体の描画」（cubePolygon.py）と同様に，新たに定義した reshape 関

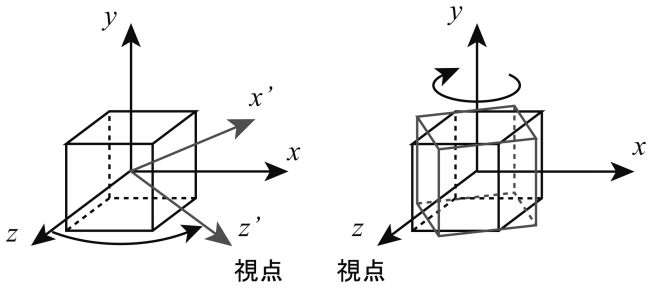

Figure 11.3: 座標系の変換

数を呼び出すためには，reshape 関数を呼び出している loop 関数も同一ファイル内に持つ必要がある（cubePosition モジュールの loop 関数は利用できない）.

```
from OpenGL.GL import *                    # GL モジュールの import
from OpenGL.GLU import *                   # GLU モジュールの import
from OpenGL.GLUT import *                  # GLUT モジュールの import
import cubePosition as cp                  # cubePosition モジュールの import

def reshape(width, height):                # ウィンドウのサイズ変更に伴うコールバック関数
  '''
  width  - 変更後の OpenGL ウィンドウの幅
  height - 変更後の OpenGL ウィンドウの高さ
  ウィンドウサイズ変更に伴う処理を行う
  '''
  fieldOfView, near, far = (25, 1, 20)   # カメラの設定 画角, 前方/後方クリッピング面
  aspect = width / height                # スクリーン面の縦横比
  glViewport(0, 0, width, height)        # ビューポートの設定
  glMatrixMode(GL_PROJECTION)            # 投影変換行列の設定開始
  glLoadIdentity()                       # 恒等行列での初期化
  gluPerspective(fieldOfView, aspect, near, far)
  glMatrixMode(GL_MODELVIEW)             # モデル変換行列の設定開始
  glLoadIdentity()                       # 恒等行列での初期化
  eyeXZ  = (cp.eyeX**2 + cp.eyeZ**2)**0.5 # xz 平面内での視点と注視点との距離
  eyeXYZ = (eyeXZ**2 + cp.eyeY**2)**0.5  # 視点と注視点との距離
  translate = (1,    0,    0,    0,
               0,    1,    0,    0,
               0,    0,    1,    0,
               0,    0,-eyeXYZ,  1)      # 平行移動行列 (z 軸)
  glMultMatrixd(translate)               # 平行移動行列の適用
  sinP, cosP = (cp.eyeY/eyeXYZ, eyeXZ/eyeXYZ) # sin(phi), cos(phi)
  rotateX   = (1,    0,    0,    0,
               0, cosP, sinP,    0,
               0,-sinP, cosP,    0,
               0,    0,    0,    1)      # 回転行列 (x 軸)
  glMultMatrixd(rotateX)                 # 回転行列の適用
  sinT, cosT = (cp.eyeX/eyeXZ, cp.eyeZ/eyeXZ) # sin(theta), cos(theta)
  rotateY   = (cosT,    0, sinT,   0,
                  0,    1,    0,   0,
              -sinT,    0, cosT,   0,
                  0,    0,    0,   1)    # 回転行列 (y 軸)
  glMultMatrixd(rotateY)                 # 回転行列の適用

def loop():                                # コールバック関数の設定とループ起動
  '''
  reshape と display コールバック関数を設定し，ループを起動する
  '''
  glutReshapeFunc(reshape)               # reshape コールバック関数の登録
  glutDisplayFunc(cp.display)            # display コールバック関数の登録
  glutMainLoop()                         # GLUT のメインループ起動

def main():                                # main 関数
```

```
  cp.window()                              # ウィンドウの作成
  cp.init()                                # OpenGL の初期化
  cp.argsInit()                            # シェル引数などの初期化
  loop()                                   # コールバック関数の設定とループ起動

if __name__ == '__main__':                 # 起動の確認（コマンドラインからの起動）
  main()                                   # main 関数の呼出
```

例２： 角度指定による立方体の描画 — cubeAngle.py

例 1 のプログラムと似ているが，必ず距離 10 だけ離れるものとし，画角，奥行き方向のクリッピング面の位置，視線方向の角度，のいずれかを，シェル引数ないしキーボード入力で与えている (図 11.4)．文字列を getArgs 関数で取得し，得られた文字列は argsInit 関数で変換されている．文字列による指定がない場合には，画角は 25°，クリッピング面は前方と後方がそれぞれ視点から 1 と 20 の距離，視線方向の角度は x, y, z 軸回りにそれぞれ 20°，−30°，0° となっている．

GL_MODELVIEW 行列は reshape 関数で z 軸方向に depth 分 (デフォルトでは −10) の平行移動が指定されており，回転変換は display 関数内で毎回施されている．ここでは，毎回，同じ回転をすればよいので，このようなプログラムでなくても構わないが，第 13 章ではマウスでインタラクティブに回転するために，このような書き方が必要となってくる．なお，行列スタックと行列スタックに関する関数 glPushMatrix と glPopMatrix については 12.2 節「ディスプレイリストと行列スタック」で詳しく説明する．

```
import sys                                 # sys モジュールの import
from OpenGL.GL import *                    # GL モジュールの import
from OpenGL.GLU import *                   # GLU モジュールの import
from OpenGL.GLUT import *                  # GLUT モジュールの import
import cubePosition as cp                  # cubePosition モジュールの import

fieldOfView, near, far = (25, 1, 20)       # カメラの設定 画角，前方/後方クリッピング面
depth, rotX, rotY, rotZ = (-10, 20, -30, 0) # 平行移動量，x 軸/y 軸/z 軸回りの回転角

def getArgs():                             # シェル引数/キーボード入力による文字列の取得
  '''
  シェル引数やキーボード入力によって文字列を取得する
  '''
  if len(sys.argv) > 1:                    # シェル引数がある場合
    args = sys.argv[1:]                    # 第 1 引数以降の文字列
  else:                                    # シェル引数がない場合
    args = input('FOV / near far / rotX rotY rotZ / [] -> ').split(' ')
  return args                  # 画角，前方/後方クリッピング面，x 軸/y 軸/z 軸回りの回転角

def argsInit(args):                        # シェル引数などの初期化
  '''
  args - 文字列の配列
  シェル引数やキーボード入力に合わせて初期化する
  '''
  global fieldOfView, near, far            # 大域変数 fieldOfView, near, far
  global rotX, rotY, rotZ                  # 大域変数 rotX, rotY, rotZ
  if len(args) == 1 and args[0] != '':     # 文字列が 1 つの場合
    fieldOfView = float(args[0])           # 画角の設定
  if len(args) == 2:                       # 文字列が 2 つの場合
    near, far = (float(args[0]), float(args[1])) # 前方/後方クリッピング面の設定
  if len(args) == 3:                       # 文字列が 3 つの場合
    rotX, rotY, rotZ = (float(args[0]), float(args[1]), float(args[2]))
                                           # x 軸/y 軸/z 軸回りの回転角の設定

def reshape(width, height):                # ウィンドウのサイズ変更に伴うコールバック関数
  '''
  width  - 変更後の OpenGL ウィンドウの幅
  height - 変更後の OpenGL ウィンドウの高さ
  ウィンドウサイズ変更に伴う処理を行う
  '''
  aspect = width / height                  # スクリーン面の縦横比
  glViewport(0, 0, width, height)          # ビューポートの設定
```

```
  glMatrixMode(GL_PROJECTION)            # 投影変換行列の設定開始
  glLoadIdentity()                       # 恒等行列での初期化
  gluPerspective(fieldOfView, aspect, near, far)
  glMatrixMode(GL_MODELVIEW)             # モデル変換行列の設定開始
  glLoadIdentity()                       # 恒等行列での初期化
  glTranslated(0, 0, depth)              # 平行移動（z 軸）

def display():                           # 描画要求に伴うコールバック関数
  '''
  描画要求に伴う処理を行う（立方体を描画する）
  '''
  glPushMatrix()                         # 行列の複写（待避）
  glRotated(rotX, 1, 0, 0)               # 回転（x 軸）
  glRotated(rotY, 0, 1, 0)               # 回転（y 軸）
  glRotated(rotZ, 0, 0, 1)               # 回転（z 軸）
  cp.display()                           # 立方体描画
  glPopMatrix()                          # 複写行列の回復

def loop():                              # コールバック関数の設定とループ起動
  '''
  reshape と display コールバック関数を設定し，ループを起動する
  '''
  glutReshapeFunc(reshape)               # reshape コールバック関数の登録
  glutDisplayFunc(display)               # display コールバック関数の登録
  glutMainLoop()                         # GLUT のメインループ起動

def main():                              # main 関数
  cp.window()                            # ウィンドウの作成
  cp.init()                              # OpenGL の初期化
  argsInit(getArgs())                    # シェル引数などの初期化
  loop()                                 # コールバック関数の設定とループ起動

if __name__ == '__main__':               # 起動の確認（コマンドラインからの起動）
  main()                                 # main 関数の呼出
```

章末課題

視点位置

例 2「角度指定による立方体の描画」のプログラムで，様々な方向からの画像を作成してみよ．

画角とクリッピング面の効果

例 2「角度指定による立方体の描画」のプログラムにおいて，画角 `fieldOfView` とクリッピング面 `near, far` の指定を変えてみよ．特にクリッピング面については，前方/後方のクリッピング面の位置として 9, 9.5, 10, 10.5 などの値を試してみよ．

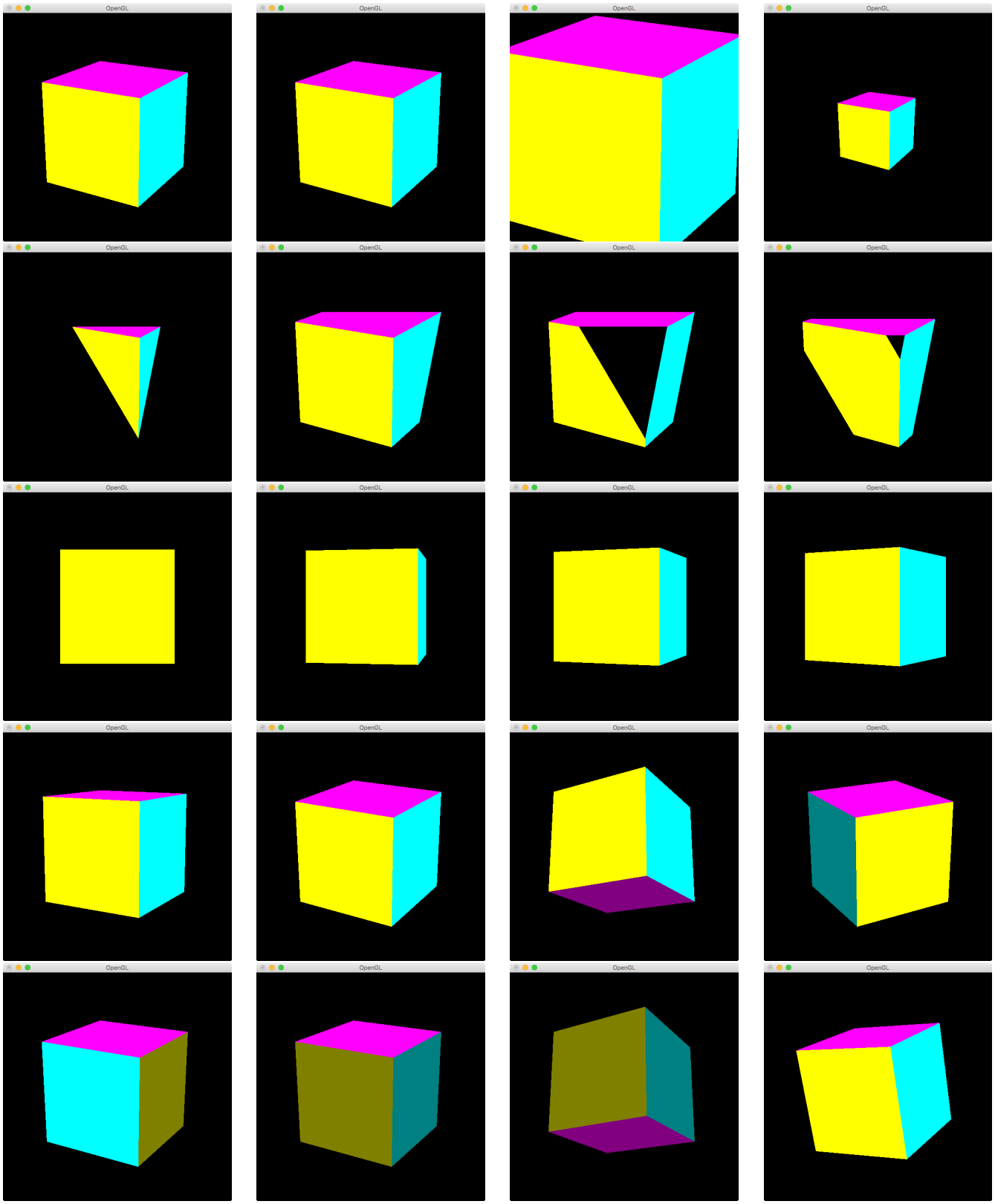

Figure 11.4: 立方体の描画．実行時に与えた引数は，左上から右方向の順に，それぞれ「(なし)」,「25」,「15」,「45」,「1 9」,「1 10」,「9 10」,「8.5 9.5」,「0 0 0」,「0 −10 0」,「0 −20 0」,「0 −30 0」,「10 −30 0」,「20 −30 0」,「−20 −30 0」,「20 30 0」,「20 −120 0」,「20 150 0」,「−20 150 0」,「20 −30 10」