同次座標と行列計算 (第 8 回)

氏名　　入佐 啓士
クラス　　理科 1 類 37 組
学生証番号 J4-220897

□課題8.0 - 11.3節 例 1: 透視投影による立方体の描画 cubeMatrix.py

○プログラムリスト

```
 1  from OpenGL.GL import *
 2  from OpenGL.GLU import *
 3  from OpenGL.GLUT import *
 4  import cubePosition as cp
 5
 6
 7  def reshape(width, height):
 8      fieldOfView, near, far = (25, 1, 20)
 9      aspect = width / height
10      glViewport(0, 0, width, height)
11      glMatrixMode(GL_PROJECTION)
12      glLoadIdentity()
13      gluPerspective(fieldOfView, aspect, near, far)
14      glMatrixMode(GL_MODELVIEW)
15      glLoadIdentity()
16      eyeXZ = (cp.eyex**2 + cp.eyez**2)**0.5
17      eyeXYZ = (eyeXZ**2 + cp.eyey**2)**0.5
18      translate = (1, 0, 0, 0,
19                   0, 1, 0, 0,
20                   0, 0, 1, 0,
21                   0, 0, -eyeXYZ, 1)
22      glMultMatrixd(translate)
23      sinP, cosP = (cp.eyey/eyeXYZ, eyeXZ/eyeXYZ)
24      rotateX = (1, 0, 0, 0,
25                 0, cosP, sinP, 0,
26                 0, -sinP, cosP, 0,
27                 0, 0, 0, 1)
28      glMultMatrixd(rotateX)
29      sinT, cosT = (cp.eyex/eyeXZ, cp.eyez/eyeXZ)
30      rotateY = (cosT, 0, sinT, 0,
31                 0, 1, 0, 0,
32                 -sinT, 0, cosT, 0,
33                 0, 0, 0, 1)
34      glMultMatrixd(rotateY)
35
36
37  def loop():
38      glutReshapeFunc(reshape)
39      glutDisplayFunc(cp.display)
40      glutMainLoop()
41
42
43  def main():
44      cp.window()
45      cp.init()
46      cp.argsInit()
47      loop()
```

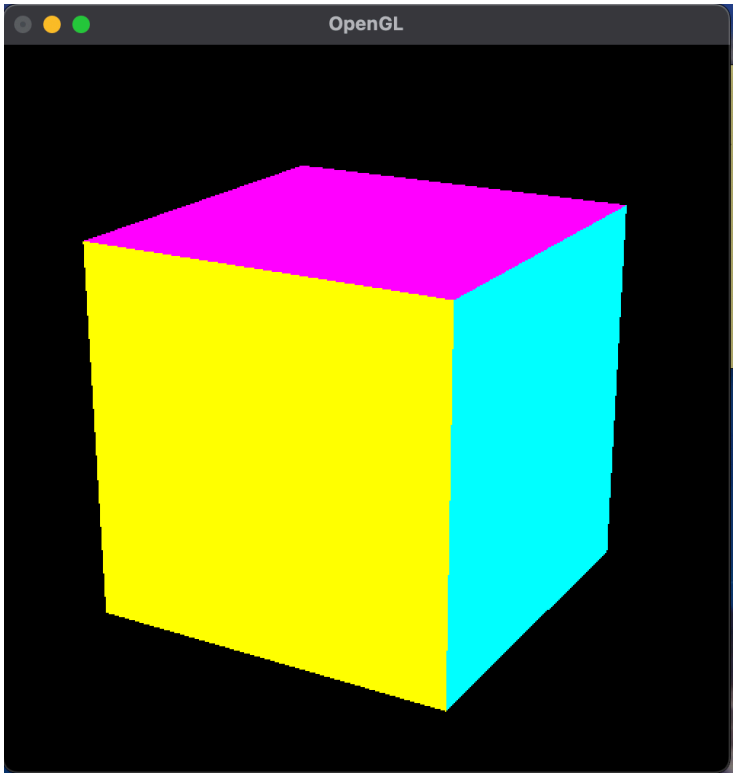
```
48
49
50  if __name__ == '__main__':
51      main()
```

○実行コマンド

```
$ python cubeMatrix.py 4 3 7
```

○実行結果

(文字列の表示なし)



○考察

今回は、GL_MODELVIEW行列を1つずつ計算することで第10章の例1と同じように、シェル引数で指定した視点からの3D立方体を描画するプログラムを走らせた。

モデルの座標系に①y軸に対する回転行列→②x軸に対する回転行列→③並行移動行列の順番で乗じて視点座標系にすることが目標だが、OpenGLではモデルに適用するのとは逆の順序で乗じていくことに注意して16行目から34行目を記述した。

実行結果が画像の通りで第10章例1と変わらない実行結果が得られた。今回の実行結果を踏まえて、前回のプログラムで実行した視点を変化させるプログラムの中でOpenGLがどのように変換行列を作用させているのかがより理解できた。

□課題8.0 - 11.3節 例 2: 角度指定による立方体の描画 cubeAngle.py

○プログラムリスト

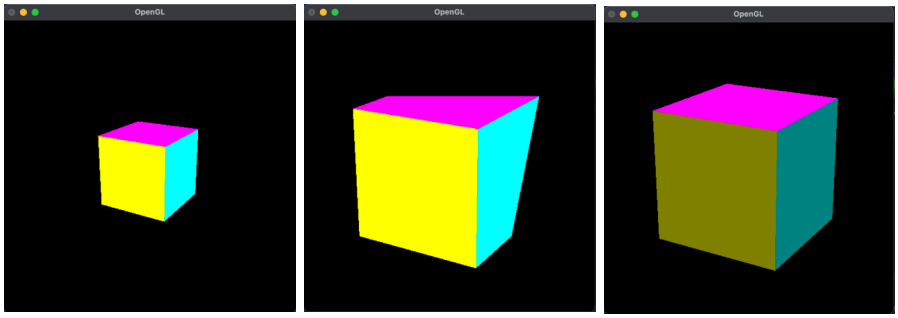
(例題のため省略)

○実行コマンド

```
$ python cubeAngle.py 45
$ python cubeAngle.py 1 10
$ python cubeAngle.py 20 150 0
```

○実行結果

(文字列の表示なし)



○考察

今回は、モデル座標系と視点座標系が必ず距離10だけ離れている立方体をシェル引数で画角、クリッピング面の位置、視線方向の角度のいずれかを指定して描画するプログラムを走らせた。

一つ目の実行結果は画像の1枚目で、はシェル引数の長さを1にして画角を25度から45度に変更した。画像の通り視野が広がったため立方体が小さく見えた。

二つ目の実行結果は画像の2枚目で、シェル引数として1 10を持たせてクリッピング面の位置を変更した。画像の通り立方体の後方が途切れたものが描画できた。これは視点からの距離が10より遠い位置にある部分は描画がストップされるからである。

三つ目の実行結果は画像の3枚目で、シェル引数として20 150 0をを持たせて、視線方向の角度を変更した。前回までのプログラムでは視点の位置を注視点を原点とする座標でシェル引数に指定していたが、今回はモデル座標系のx,y,z軸に対しての回転角度によって指定した。実行結果は画像の通りで、y軸周りの回転角をシェル引数に何も指定しない時から180度回転させた角度にしたため、真反対からの立方体が描画できた。

画角とクリッピング面の位置による立方体の描画の変化は非常に興味深かった。クリッピング面の指定が実際にどのように技術に生かされているのかが気になりまし

た。私は、自動運転などでクリッピングにより近くの物体だけを認識して危険度を測るというような使用方法を考えました。

□課題8.0 - 章末課題: 視点位置 cubeAngle.py

○プログラムリスト

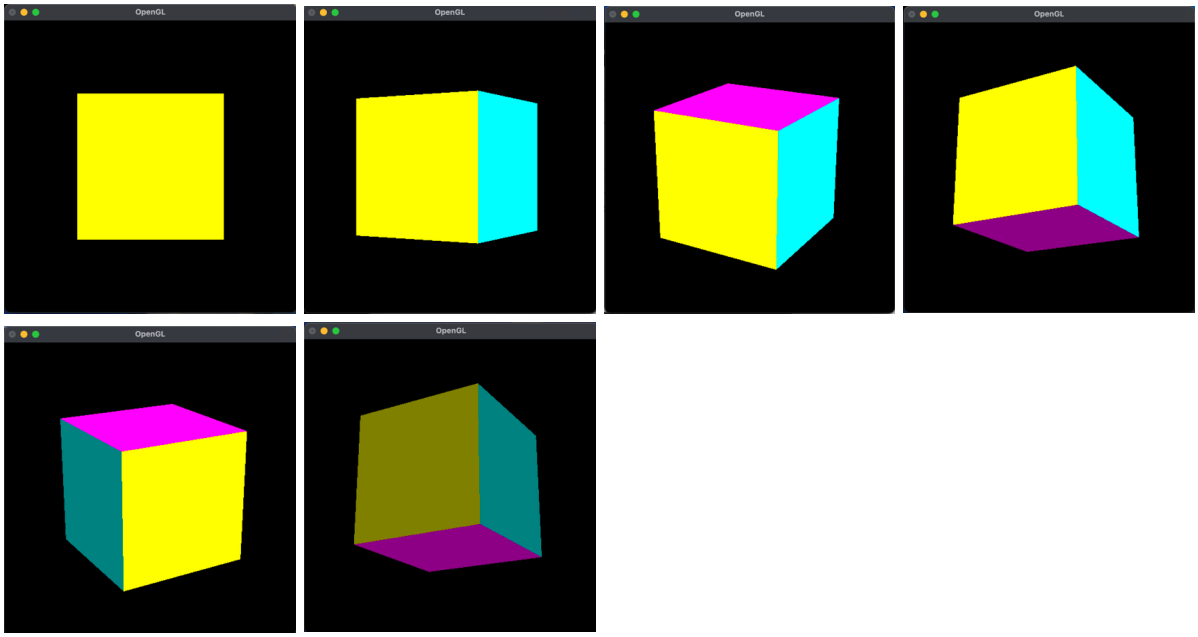
(例題と同じため省略)

○実行コマンド

```
$ python cubeAngle.py 0 0 0
$ python cubeAngle.py 0 -30 0
$ python cubeAngle.py 20 -30 0
$ python cubeAngle.py -20 -30 0
$ python cubeAngle.py 20 30 0
$ python cubeAngle.py -20 150 0
```

○実行結果

(文字列の表示なし)



○考察

今回は、例題2のプログラムのシェル引数にさまざまな回転角を指定することで、様々な方向からの画像を描画するプログラムを走らせた。

1つ目は(0, 0, 0)を指定し、モデル座標系から少しも回転しなかったため真正面から立方体を眺めた1番目の画像の通りの実行結果となった。

2つ目は(0 -30 0)を指定し、視点をy軸周りに-30度回転させたため視点を固定して水

平方向に30度回転させた2番目の画像のような実行結果となった。

3つ目は(20 -30 0)を指定し、視点をx軸周りに20度回転させたため視点から見る物体の位置はx軸周りに-20度回転して立方体の上部が見える3枚目の画像の通りになった。

4つ目は(-20 -30 0)を指定し、2つ目の立方体を視点を固定して、20度x軸周りに回転させた立方体を描画する。そのため実行結果は4枚目の画像の通りで、立方体の下部が見える結果となった。

5つ目は(20 30 0)を指定し、3つ目の立方体を視点を固定して、-60度y軸周りに回転させた立方体を描画する。そのため実行結果は5枚目の画像の通り3では見えなかった立方体の左の面が見える結果となった。

6つ目は(-20 150 0)を指定し、4つ目の立方体を視点を固定して、180度y軸周りに回転させた立方体を描画する。そのため実行結果は6枚目の画像の通り、4では見えなかった正反対の部分が見える結果となった。

▫課題8.0 - 章末課題: 画角とクリッピング面の効果 cubeAngle.py

○プログラムリスト

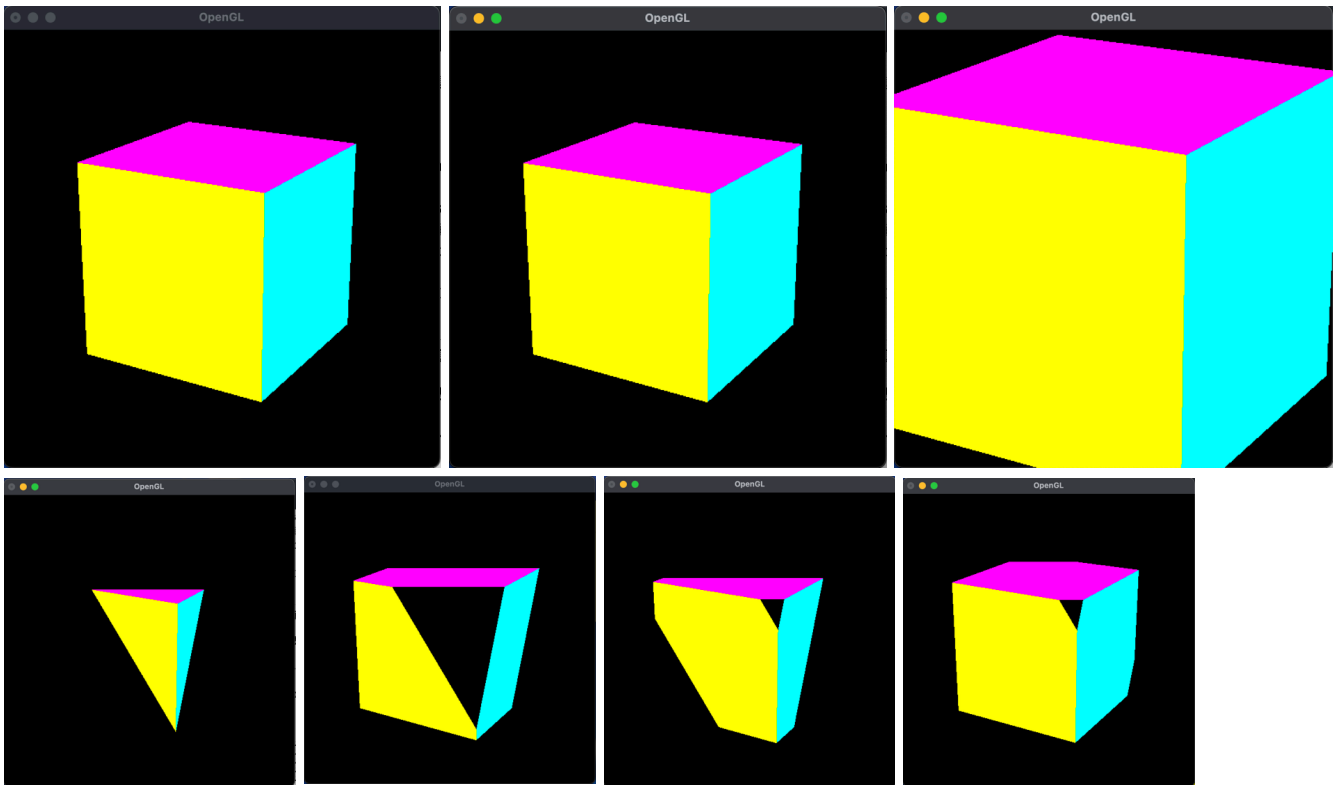
(例題と同じ省略)

○実行コマンド

```
$ python cubeAngle.py
$ python cubeAngle.py 25
$ python cubeAngle.py 15
$ python cubeAngle.py 1 9
$ python cubeAngle.py 9 10
$ python cubeAngle.py 8.5 9.5
$ python cubeAngle.py 8.5 10.5
```

○実行結果

(文字列の表示なし)



○考察

今回は、画角とクリッピング面の指定を変更して様々な立方体を描画するプログラムを走らせた。前半3つは画角を変更したもので、後半3つはクリッピング面を変更したものである

1つ目ではシェル引数を指定しなかったため、プログラムの初期値である画角25度で描画された。これは前回までのプログラムで立方体を描画していた画角と同じであるため、同じ立方体が描画された。

2つ目では画角25度を指定して描画した。しかしこれは初期値と同じ値のため1つ目と全く同じ画像が描画された。

3つ目では画角15度を指定して描画した。画角が先ほどより小さくなったということは、より視点が物体に近づいたということなので、より大きく立方体が描画されると予想した。実行結果は3枚目の画像の通りで、物体に視点が近づきすぎて視野に収まっていないことがわかる。

4つ目ではクリッピング位置を(1 9)と指定して描画した。これは後方のクリッピング面が初期値よりも11小さい。注視点と視点の距離は必ず10のため、切り口は注視点より視点側であり、立方体の後方の大部分が途切れた立方体が描画されると予想した。実行結果は4枚目の画像の通りで立方体の角一つだけが残ったものが描画された。

5つ目ではクリッピング位置を(9 10)と指定して描画した。前方のクリッピング面の

位置が4つ目の後方のクリッピング位置であり、後方のクリッピング位置は例題2の2つ目の実行結果の後方のクリッピング位置と等しいため、例題2の2つ目の実行結果から4つ目の実行結果の部分を除いた立体が描画されると予想した。実行結果は5枚目の画像の通りで、予想通り4つ目の立体を除いた立体が描画された。

6つ目ではクリッピング位置を(8.5 9.5)と指定して描画した。5つ目と比較すると前方の位置が0.5手前、後方が0.5手前なので、5つ目の実行結果の後方が少し途切れて、前方の描画が少し増えた立体が描画されると予想した。実行結果は6枚目の画像の通りで、前方が少し欠け、後方は5つ目より途切れている立体が描画できた。

7つ目ではクリッピング位置を(8.5 10.5)と指定して描画した。6つ目より後方の位置が1大きいため、後方の描画される部分がより大きくなると予想した。実行結果は7つ目の画像の通りで、予想通り後方の大部分が描画されたように感じた。これは立方体の対角線中心付近にクリッピング位置が近づくと、同じ距離の変化でもより多くの面積が描画されるからである。

▫課題や授業に関して

○レポート作成に要した時間

4時間

○特に苦労した点

cubeMatrix.py視点座標系への変換の部分を理解するのに時間がかかった。

○授業についての感想や希望

ちょうど良い分量の課題でした。授業が多面体の描画に入るのが楽しみです。