プログラミング演習 II 課題レポート

Ec5 24 番 平田 蓮

課題 I

1.

grep　ファイル内の文字列を検索するコマンド。[1]

用例: `grep 検索対象の文字列 検索するファイル`

テキスト [2] のリスト 6 から “`glVertex`” を検索した結果をリスト 1 に示す。

touch　ファイルのタイムスタンプを書き換えるコマンド。[3]

用例: `touch オプション ファイル名`

オプションを指定しない場合、指定したファイルのタイムスタンプが現在に更新される。また、指定したファイルが存在しない場合、ファイルを新規作成する。一般的にはこの機能を用いて、新規ファイルを作成するコマンドとして用いられる。オプションを以下に示す。

-d　日時を指定する。

-c　ファイルの作成を行わない。

-r　指定したファイルのタイムスタンプを別のファイルのそれとあわせる。

リスト 1　grep コマンドの出力

```
1     glVertex2d(0, 1);
2     glVertex2d(0, 0);
3     glVertex2d(1, 0);
```

2.

有効範囲　C 言語のソースコードにおいて、{}で囲まれた範囲のことを有効範囲と呼ぶ。変数は、自身が宣言された有効範囲内でのみ使用することができる。特にソースコードの一番外側の有効範囲（`main()` の外側）で宣言された変数はグローバル変数と呼ぶ。

記憶期間　C 言語において変数宣言をする際は、`auto`, `static` などのキーワードを用いることができる。（用例:`auto int a=100;static double b=4.6;`）デフォルト値は `auto` で、これは省略可能である。`auto` を使用して宣言された変数は自動記憶域期間を持つ。これはプログラムが実行される中で記述通りのタイミングで生成され、有効範囲の実行が完了した時点で遺棄される。また、初期値の指定がない場合は不定値となる。一方 `static` を使用すると変数は静的記憶域期間を持つ。これはプログラムの開始時に生成され、プログラムの終了時まで保持される。さらに、初期値が指定されない場合は自動的に

0 で初期化される。

3.

プログラムを一部抜粋してリスト 2 に示す。

リスト 2　星型正多角形を描くプログラム

```
#define VERTEX_NUM 7
#define ROTATION_SPEED M_PI / 50.0

void display() {
    static double rotation, dt = 4.0 * M_PI / (double)VERTEX_NUM;
    double x, y, theta = rotation;
    int i;

    glClear(GL_COLOR_BUFFER_BIT);

    glColor3d(0, 0, 0);

    glBegin(GL_LINES);
    for (i = 0; i < VERTEX_NUM * 2; i++, theta += dt) {
        x = cos(theta);
        y = sin(theta);

        glVertex2d(x, y);
    }

    glEnd();
    glutSwapBuffers();

    rotation += ROTATION_SPEED;
}
```

VERTEX_NUM, ROTATION_SPEED はそれぞれ頂点数、回転速度である。頂点を一つ飛ばしに結ぶことで図形を描いている。

4.

プログラムを一部抜粋してリスト 3 に示す。

リスト 3　完全グラフを描くプログラム

```
#define VERTEX_NUM 7
#define ROTATION_SPEED M_PI / 50.0

void display() {
    static double rotation;
```

```
    double x, y, theta, dt;
    int i, j;

    glClear(GL_COLOR_BUFFER_BIT);

    glColor3d(0, 0, 0);

    glBegin(GL_LINES);
    for (i = 1, theta = rotation; i < (double)VERTEX_NUM / 2.0; i++) {
        dt = 2.0 * i * M_PI / (double)VERTEX_NUM;
        for (j = 0; j < VERTEX_NUM * 2; j++, theta += dt) {
            x = cos(theta);
            y = sin(theta);

            glVertex2d(x, y);
        }
    }

    glEnd();
    glutSwapBuffers();

    rotation += ROTATION_SPEED;
}
```

星型正多角形は頂点を一つ飛ばしで結んだが、0 個飛ばしから頂点数の半分個飛ばしまでそれぞれでできる図形を重ねることで完全グラフが描ける。

5.

それぞれの図形を描画するプログラムをリスト 4, 5, 6 に示す。

リスト 4　カージオイドを描くプログラム

```
void drawAxis() {
    double x, y;

    glBegin(GL_LINES);
    glVertex2d(-0.5, 0);
    glVertex2d(3, 0);
    glVertex2d(0, -1.5);
    glVertex2d(0, 1.5);
    for (x = 0; x < 3; x++) {
        glVertex2d(x, -0.05);
        glVertex2d(x, 0.05);
    }
    for (y = -1; y < 1.5; y += 0.5) {
        glVertex2d(-0.05, y);
```

```
15          glVertex2d(0.05, y);
16      }
17      glEnd();
18  }
19
20  void drawCardioid() {
21      double x, y, theta;
22
23      glBegin(GL_LINE_STRIP);
24      for (theta = 0; theta < 2 * M_PI; theta += 0.01) {
25          x = cos(theta) * (1 + cos(theta));
26          y = sin(theta) * (1 + cos(theta));
27
28          glVertex2d(x, y);
29      }
30      glEnd();
31  }
32
33  void display() {
34      glClear(GL_COLOR_BUFFER_BIT);
35
36      glColor3d(0, 0, 0);
37
38      drawAxis();
39      drawCardioid();
40
41      glutSwapBuffers();
42  }
```

リスト 5　サイクロイドを描くプログラム

```
1   void drawAxis() {
2       double x, y;
3
4       glBegin(GL_LINES);
5       glVertex2d(-0.5, 0);
6       glVertex2d(2 * M_PI + 0.5, 0);
7       glVertex2d(0, -0.5);
8       glVertex2d(0, 2.5);
9       for (x = 0; x < 2.5; x += 0.5) {
10          glVertex2d(x * M_PI, -0.05);
11          glVertex2d(x * M_PI, 0.05);
12      }
13      for (y = 0; y < 3; y ++) {
14          glVertex2d(-0.05, y);
15          glVertex2d(0.05, y);
16      }
```

```
17      glEnd();
18  }
19
20  void drawCycloid() {
21      double x, y, theta;
22
23      glBegin(GL_LINE_STRIP);
24      for (theta = 0; theta < 2 * M_PI; theta += 0.01) {
25          x = theta - sin(theta);
26          y = 1 - cos(theta);
27
28          glVertex2d(x, y);
29      }
30      glEnd();
31  }
32
33  void display() {
34      glClear(GL_COLOR_BUFFER_BIT);
35
36      glColor3d(0, 0, 0);
37
38      drawAxis();
39      drawCycloid();
40
41      glutSwapBuffers();
42  }
```

リスト 6　4 尖点の内サイクロイドを描くプログラム

```
1   void drawAxis() {
2       double x, y;
3
4       glBegin(GL_LINES);
5       glVertex2d(-1.5, 0);
6       glVertex2d(1.5, 0);
7       glVertex2d(0, -1.5);
8       glVertex2d(0, 1.5);
9       for (x = -1; x < 1.5; x += 0.5) {
10          glVertex2d(x, -0.05);
11          glVertex2d(x, 0.05);
12
13          y = x;
14          glVertex2d(-0.05, y);
15          glVertex2d(0.05, y);
16      }
17      glEnd();
18  }
```

```
19
20 void drawHypocycloid() {
21     double x, y, theta;
22
23     glBegin(GL_LINE_STRIP);
24     for (theta = 0; theta < 2 * M_PI; theta += 0.01) {
25         x = cos(theta);
26         x = x * x * x;
27         y = sin(theta);
28         y = y * y * y;
29
30         glVertex2d(x, y);
31     }
32     glEnd();
33 }
34
35 void display() {
36     glClear(GL_COLOR_BUFFER_BIT);
37
38     glColor3d(0, 0, 0);
39
40     drawAxis();
41     drawHypocycloid();
42
43     glutSwapBuffers();
44 }
```

描画結果を図 1, 2, 3 に示す。

課題 II

1.

$\triangle BGI$ に注目すると、$\angle GBI = \frac{\pi}{3}, BI = \frac{w}{2}$ なので、$GB = \frac{w}{\sqrt{3}}, GI = \frac{w}{2\sqrt{3}}$ である。同様に、$GH = \frac{w}{2\sqrt{3}}, HC = HD = \frac{w}{2}$ である。

また、正四面体状での $\triangle AGI$ に注目すると、三平方の定理より、

$$GA = \sqrt{(AI)^2 - (GI)^2} = \sqrt{(GI + GB)^2 - (GI)^2} = \sqrt{\frac{2}{3}}w$$

となる。

以上により、各頂点の座標は、

$$A = \left(\sqrt{\frac{2}{3}}w, 0, 0\right), B = \left(0, \frac{w}{\sqrt{3}}, 0\right), C = \left(0, -\frac{w}{2\sqrt{3}}, \pm\frac{w}{2}\right), D = \left(0, -\frac{w}{2\sqrt{3}}, \mp\frac{w}{2}\right)$$

である。(C, D の z 成分は複号同順で、z 軸の取り方に依る。)

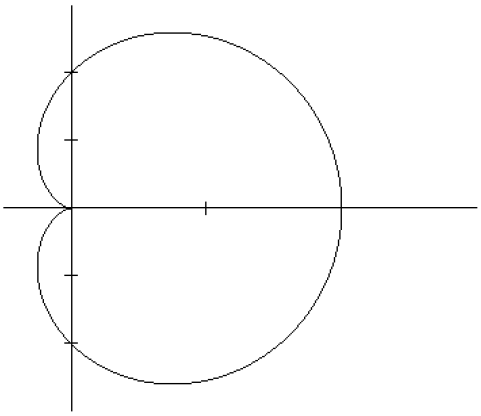

図 1　カージオイド

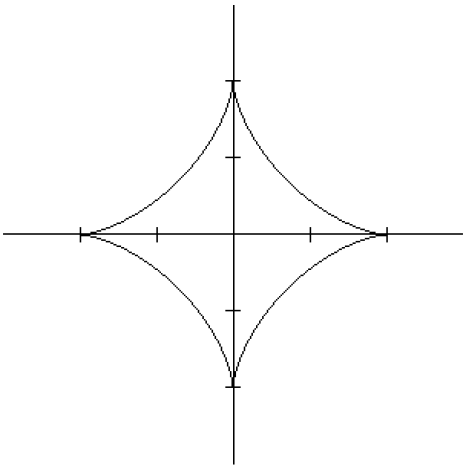

図 2　4 尖点の内サイクロイド

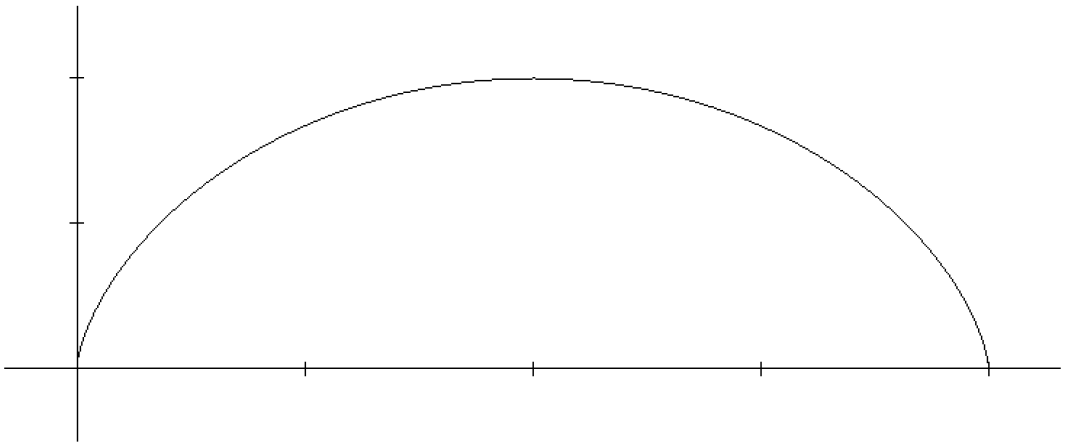

図 3　サイクロイド

2.

正四面体が球に内接しているため、その重心は原点にあると考えられる。前節より、正四面体の重心は、

$$\frac{1}{4}\left(\sqrt{\frac{2}{3}}w, \frac{w}{\sqrt{3}} - \frac{w}{2\sqrt{3}} - \frac{w}{2\sqrt{3}}, \frac{w}{2} - \frac{w}{2}\right) = \left(\frac{w}{2\sqrt{6}}, 0, 0\right)$$

である。各点を平行移動させると、

$$A = \left(\frac{\sqrt{6}}{4}w, 0, 0\right), B = \left(-\frac{w}{2\sqrt{6}}, \frac{w}{\sqrt{3}}, 0\right), C = \left(-\frac{w}{2\sqrt{6}}, -\frac{w}{2\sqrt{3}}, \pm\frac{w}{2}\right), D = \left(-\frac{w}{2\sqrt{6}}, -\frac{w}{2\sqrt{3}}, \mp\frac{w}{2}\right)$$

となり、$A = (1, 0, 0)$ であるので、$w = 2\sqrt{\frac{2}{3}}$ である。これを上に代入して、

$$A = (1, 0, 0), B = \left(-\frac{1}{3}, \frac{2\sqrt{2}}{3}, 0\right), C = \left(-\frac{1}{3}, -\frac{\sqrt{2}}{3}, \pm\sqrt{\frac{2}{3}}\right), D = \left(-\frac{1}{3}, -\frac{\sqrt{2}}{3}, \mp\sqrt{\frac{2}{3}}\right)$$

を得る。これは、テキストのリスト 14 と一致している。

3.

台座、下腕、上腕それぞれの登録部を抜粋してそれぞれリスト 7, 8, 9 に示す。

リスト 7　台座の登録部

```
glNewList(ID_B, GL_COMPILE);
glColor3d(0, 1, 0);
glPushMatrix();
glRotated(90, 1, 0, 0);
GLUquadric *quad = gluNewQuadric();
gluQuadricDrawStyle(quad, GLU_LINE);
gluCylinder(quad, RADIUS_B, RADIUS_B, HEIGHT_B, SLICES_B, STACKS_B);
glPopMatrix();
glEndList();
```

リスト 8　下腕の登録部

```
glNewList(ID_L, GL_COMPILE);
glColor3d(0, 1, 0);
glPushMatrix();
glScalef(WIDTH_L, HEIGHT_L, WIDTH_L);
glutWireCube(1);
glPopMatrix();
glEndList();
```

リスト 9　上腕の登録部

```
glNewList(ID_U, GL_COMPILE);
glColor3d(0, 1, 0);
glPushMatrix();
glTranslatef(-0.5 * HEIGHT_U, 0, 0);
glScalef(WIDTH_U, HEIGHT_U, WIDTH_U);
glutWireCube(1);
glPopMatrix();
glEndList();
```

次に、キーボードコールバック関数をリスト 10 に示す。ROTATION_SPEED は回転速度で、マクロ定義をしている。

リスト 10　キーボードコールバック関数

```
1  void keyin(unsigned char key, int x, int y) {
2      switch (key) {
3          case '\033':
4          case 'q':
5          case 'Q':
6              exit(0);
7              break;
8          case 'w':
9          case 'W':
10             rotAng[0] += ROTATION_SPEED;
11             break;
12         case 's':
13         case 'S':
14             rotAng[0] -= ROTATION_SPEED;
15             break;
16         case 'e':
17         case 'E':
18             rotAng[1] += ROTATION_SPEED;
19             break;
20         case 'd':
21         case 'D':
22             rotAng[1] -= ROTATION_SPEED;
23             break;
24         case 'r':
25         case 'R':
26             rotAng[2] += ROTATION_SPEED;
27             break;
28         case 'f':
29         case 'F':
30             rotAng[2] -= ROTATION_SPEED;
31             break;
32     }
33 }
```

最後に、描画されたアームロボットを図 4 に示す。

4.

作成したタイマーコールバック関数をリスト 11 に示す。

リスト 11　タイマーコールバック関数

```
1  void timer(int dummy) {
```

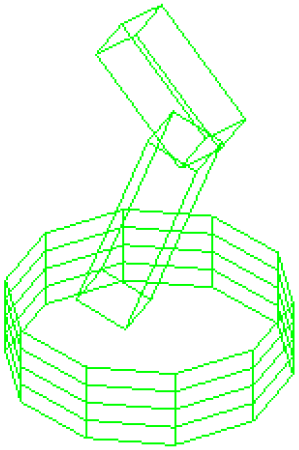

図 4　アームロボット

```
2       glutTimerFunc(15, timer, 0);
3
4       rotAng[0] += BASE_SPEED;
5
6       rotAng[1] = AMP * sin((double)LOWER_SPEED * rotAng[0] / 180.0 * M_PI);
7       rotAng[2] = AMP * sin((double)UPPER_SPEED * rotAng[0] / 180.0 * M_PI);
8
9       glutPostRedisplay();
10  }
```

BASE_SPEED, LOWER_SPEED, UPPER_SPEED, AMP はそれぞれ台座の回転速度、下腕の回転速度、上腕の回転速度、両腕の振幅であり、2 本の腕が異なる速度で振動する。異なる速度で振動させることでアームロボットが踊っているように見えた。

課題 III

1.

P_1 から P_2 のベクトル、P_1 から P_3 のベクトルをそれぞれ $\overrightarrow{P_1P_2}, \overrightarrow{P_1P_3}$ とする。$\triangle P_1P_2P_3$ はこの 2 本のベクトルが成す平面上にあるので、その単位法線ベクトルは、$\pm\dfrac{\overrightarrow{P_1P_2} \times \overrightarrow{P_1P_3}}{|\overrightarrow{P_1P_2} \times \overrightarrow{P_1P_3}|}$ である。

2.

ある点から P_1, P_2 に向かうベクトルを v_1, v_2 とすると、$v_1 \times v_2$ の符号でその点が $\triangle P_1P_2P_3$ のどちらの面にあるか判断できる。

3.

Phong の照光モデルは 1975 年にユタ大学の Bui Tuong Phong によって提案された [4]。このモデルは物体が発する光を環境光、拡散反射光、鏡面反射光の和としている。非常に単純で、計算量も小さいため、従来、標準シェーダーとして用いられてきた。しかし、近似モデルであるため、大域照明を記述することができない。

4.

頂点の法線ベクトルは、図形の中心からその頂点に向かうベクトルとすれば良い。よって、原点を中心とする図形の場合、頂点の位置ベクトルと一致する。三つの図形を表示するプログラムの主要部をリスト 12, 13, 14、三つの図形を 5, 6, 7 に示す。

リスト 12　正四面体を表示するプログラム

```
GLdouble vP[4][3] = {
    { 1.000,  0.000,  0.000},
    {-0.333,  0.943,  0.000},
    {-0.333, -0.471,  0.816},
    {-0.333, -0.471, -0.816}
};
int tP[4][3] = {
    {0, 1, 2},
    {0, 2, 3},
    {0, 3, 1},
    {1, 3, 2}
};

void display() {
    int i,j;
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);

    glBegin(GL_TRIANGLES);
    for (i = 0; i < 4; i++) {
        for (j = 0; j < 3; j++) {
            glColor3d(0, 0, 0);
            glNormal3dv(vP[tP[i][j]]);
            glVertex3dv(vP[tP[i][j]]);
        }
    }
    glEnd();

    glutSwapBuffers();
}
```

リスト 13　正六面体を表示するプログラム

```
GLdouble vP[8][3] = {
    {-0.577, -0.577, 0.577},
    {0.577, -0.577, 0.577},
    {0.577, -0.577, -0.577},
    {-0.577, -0.577, -0.577},
    {-0.577, 0.577, 0.577},
    {0.577, 0.577, 0.577},
    {0.577, 0.577, -0.577},
    {-0.577, 0.577, -0.577}
};
int tP[6][4] = {
    {0, 1, 5, 4},
    {0, 4, 7, 3},
    {1, 2, 6, 5},
    {3, 2, 1, 0},
    {3, 7, 6, 2},
    {4, 5, 6, 7}
};

void display() {
    int i,j;
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);

    glBegin(GL_QUADS);
    for (i = 0; i < 6; i++) {
        for (j = 0; j < 4; j++) {
            glColor3d(0, 0, 0);
            glNormal3dv(vP[tP[i][j]]);
            glVertex3dv(vP[tP[i][j]]);
        }
    }
    glEnd();

    glutSwapBuffers();
}
```

リスト 14　正八面体を表示するプログラム

```
GLdouble vP[6][3] = {
    { 1,  0,  0},
    { 0,  1,  0},
    { 0,  0,  1},
    {-1,  0,  0},
    { 0, -1,  0},
```

```
7       { 0,  0, -1}
8   };
9   int tP[8][3] = {
10      {0, 1, 2},
11      {0, 4, 5},
12      {2, 1, 3},
13      {2, 4, 0},
14      {3, 4, 2},
15      {3, 1, 5},
16      {5, 1, 0},
17      {5, 4, 3}
18  };
19
20  void display() {
21      int i,j;
22      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
23      glMatrixMode(GL_MODELVIEW);
24
25      glBegin(GL_TRIANGLES);
26      for (i = 0; i < 8; i++) {
27          for (j = 0; j < 3; j++) {
28              glColor3d(0, 0, 0);
29              glNormal3dv(vP[tP[i][j]]);
30              glVertex3dv(vP[tP[i][j]]);
31          }
32      }
33      glEnd();
34
35      glutSwapBuffers();
36  }
```

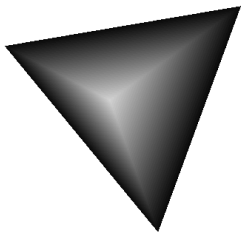

図 5　正四面体

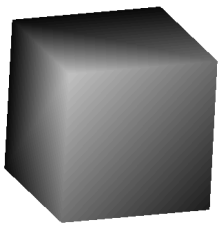

図 6　正六面体

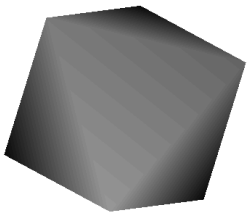

図 7　正八面体

5.

図 8 に、シェーディング表示にしたロボットアームを示す。

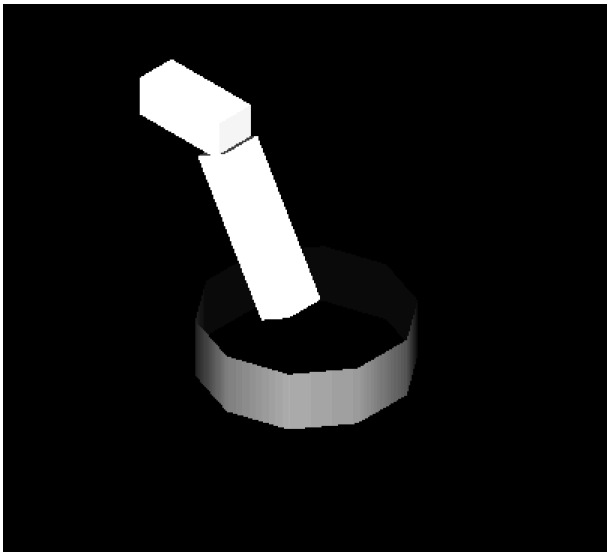

図 8　シェーディング表示のロボットアーム

感想等

身近なソフトウェアに OpenGL はよく使われているが、実際に自分で OpenGL を扱ったことはなかったので、新しい体験ができた。OpenGL はあまり直感的にプログラムを書くのが難しく、今では当たり前になっている GUI の裏ではそこそこ面倒なプログラミングがなされていることを知った。

テキストの改善点として、課題の設問文が少しわかりづらい点をあげる。特に課題 III の 2 など、どのような条件で具体的に何を求めれば良いかがわかりづらかった。

参考文献

[1] “grep コマンド”, IBM, https://www.ibm.com/docs/ja/aix/7.1?topic=grep-command, 2020

[2] “R03-Ec5 プログラミング演習 II テキスト”, 高橋章, 2021

[3] “touch コマンド”, IBM, https://www.ibm.com/docs/ja/aix/7.1?topic=t-touch-command, 2020

[4] “Illumination for computer generated pictures”, Bui Tuong Phong, Communications of the ACM vol. 18, 311-317, 1975