自己評価Sの演習

◇演習 4. 拡大縮小するウィンドウ： ウィンドウサイズを 2 倍にしたら，ウィンドウ中の図形が 2 倍に拡大されて見えるようなリサイズコールバック関数を作成せよ．ただし，ウィンドウの縦横比が変化しても，描画される図形の縦横比が変わらないようにせよ．例えばウィンドウサイズが 300×200 画素のときは，WCS の $(-1.5, -1.0) \sim (1.5, 1.0)$ の範囲が描画され，ウィンドウサイズが 200×400 画素のときは，WCS の $(-1.0, -2.0) \sim (1.0, 2.0)$ の範囲が描画されるような変換を考えればよい．

作成したリサイズコールバック関数をリスト 1 に示す．この関数はワールド座標系の矩形領域 (WCS_L,WCS_B)∼(WCS_R,WCS_T) ウィンドウにギリギリ収まるように表示する．ワールド座標系の表示する領域を設定できるため，汎用性が高く他のプログラムに応用しやすくなっている．縦に余裕があるときの描画例と横に余裕があるときの描画例を図 に示す．灰色の線は表示したい領域を示している．ウィンドウにギリギリ収まるように描画できていることがわかる．

リスト 1　リサイズコールバック関数（resize）

```
1  void resize(int w, int h)
2  {
3      GLdouble l, r, b, t, tmp1, tmp2;
4      GLdouble wcs_w = WCS_R - WCS_L, wcs_h = WCS_T - WCS_B;
5      glViewport(0, 0, w, h);
6      glMatrixMode(GL_PROJECTION);
7      glLoadIdentity();
8      if ((wcs_h * w) < (wcs_w * h))
9      {
10         tmp1 = WCS_B + WCS_T;
11         tmp2 = wcs_w * h / w;
12         l = WCS_L;
13         r = WCS_R;
14         b = (tmp1 - tmp2) / 2;
15         t = (tmp1 + tmp2) / 2;
16     }
17     else
18     {
19         tmp1 = WCS_L + WCS_R;
20         tmp2 = wcs_h * w / h;
21         l = (tmp1 - tmp2) / 2;
22         r = (tmp1 + tmp2) / 2;
23         b = WCS_B;
24         t = WCS_T;
25     }
26     gluOrtho2D(l, r, b, t);
27 }
```

(a)　縦に余裕があるとき　　(b)　横に余裕があるとき

図 1　ウィンドウサイズ別の描画例

課題

課題 I

1. **プログラミングツールのツール：**BCC には MAKE ユーティリティの他，grep や touch などプログラム開発に行う際に役に立つコマンドラインツールが含まれている．それぞれの機能や利用法について学習せよ．

1.1. MAKE ユーティリティ

Make は，プログラムのビルドを自動化するために用いられるツールである．ビルド作業が複雑になるときに特に役に立つ．例えば，複雑に関連したソースファイルなどをビルドするとき，長いコマンドや複数のコマンドが必要になる．Make によりそれらを一つにまとめビルド作業を簡単化できる．make コマンドを使うために Makefile を規則に従って作る必要がある．コマンドは以下のように使用する．[2]

```
make target
```

1.2. grep

Grep はテキストファイルから指定されたパターン（文字列）を 1 つまたは複数検索する．コマンドを使うときにオプションを指定することでアルファベットの大文字小文字の区別などを設定できる．コマンドは以下のように使用する．[3]

```
grep [option] pattern file
```

1.3. touch

touch は指定したファイルのアクセス時間（atime）や修正時間（mtime）を書き換える．作成時間（ctime）は書き換えない．コマンドは以下のように使用する．[4]

```
touch [option] file
```

2. **C 言語：**変数の「有効範囲（Scope)」,「記憶域期間（記憶寿命)」について学習し，一般的な（ローカル）変数とグローバル変数，スタティック変数の違いを整理せよ．

2.1. 有効範囲（Scope）

「有効範囲（Scope)」には，ブロック内を有効範囲（ブロック有効範囲）とする場合とファイル内を有効範囲（ファイル有効範囲）とする場合がある．ブロック内とは{}で囲まれた部分のことで，ファイル内とはファイル全ての範囲のことである．[5]

2.2. 記憶域期間（記憶寿命）

「記憶域期間（記憶寿命)」には，自動記憶域期間と静的記憶域期間がある．自動記憶域期間とは auto [*1]を使用して宣言したオブジェクト（変数）が持つ記憶域期間のことで，オブジェクトは宣言したブロックに入ったときから終了するまで生存する．静的記憶域期間とは，ファイル有効範囲のオブジェクトか static を使用して宣言したときにオブジェクトが持つ記憶域期間のことで，プログラムの開始から終了するまで生存する．[6]

[*1] auto は省略することができるので一般に明示はしない．

2.3. 各変数の有効範囲と記憶域期間

ローカル変数，グローバル変数，スタティック変数の有効範囲と記憶域期間をまとめたものを表 1 に示す．

表 1　各変数の有効範囲と記憶域期間

変数	有効範囲	記憶域期間
ローカル変数	ブロック有効範囲	自動記憶域期間
グローバル変数	ファイル有効範囲	静的記憶域期間
スタティック変数	ブロック有効範囲	静的記憶域期間

3. テキストの図 1，図 2 のような図形（星型正多角形）を描き，回転させるプログラムを作成せよ．

3.1. 星型正多角形

星型正多角形とは，正多角形と同様に全ての辺の長さが等しく全ての内角の大きさが等しいが正多角形とは異なり辺同士が交わり合っている図形である．注意として六芒星[*2]のような幾つかの正多角形に分解できるものは星型正多角形に含まれない．つまり，星型正多角形は一筆書きで書くことができる．[7]

n 本の辺からなり元の位置に戻るために m 周する星型正多角形について m は多角形の密度と呼ばれ，星型正多角形になるための条件は，n と m の最大公約数が 1 かつ，$n > 2m$ である．外角の和は m 周することから $2\pi m$ となり，外角は $2\pi m/n = 2\pi/(n/m)$ [*3]で求めることができる．このことから正多角形を星型正多角形に拡張して，正 n/m 角形と呼ばれる．また，シュレーフリ記号（Schläfli symbol）では $\{n/m\}$ と表記される．

3.2. プログラム

演習 3 で作成した正多角形を描画するプログラムを書き換えて星型正多角形（正 n/m 角形）を描画するプログラムを作成する．作成した星型正多角形を描画する関数（dispStarPolygon）をリスト 2 に示す．この関数は順に n，m，θ を引数としている．θ は基準となる角度で，値を変えることで星型正多角形を回転させることができる．関数 dispStarPolygon を使い正 n/m 角形を回転させるためのプログラムをリスト 3 に示す．リスト 3 は 1，2 行目で正 7/3 角形に設定していて，テキストのリスト 11 のタイマーコルバック関数を用いて，100 [ms] 経つたびに rotAng を $\pi/180$ 増加させ回転させている．このプログラムの出力結果を図 2 に示す．

リスト 2　星型正多角形を描画する関数（dispStarPolygon）

```
1  void dispStarPolygon(int n, int m, double theta)
2  {
3      int i;
4      double dt;
5      dt = 2.0 * M_PI * m / n;
6      glLineWidth(4);
7      glBegin(GL_LINE_LOOP);
8      for (i = 0; i < n; i++)
9      {
10         glVertex2d(cos(theta), sin(theta));
11         theta += dt;
12     }
13     glEnd();
14 }
```

[*2] 2 つの正三角形が重なってできている図形．

[*3] 正多角形の場合も $m = 1$ となることから，この式は使える．

リスト 3　星型正多角形を回転させるプログラム（主要部）

```
#define N 7
#define M 3

void display(void)
{
    static double rotAng = 0.0;
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3d(0.0, 0.0, 0.0);
    dispStarPolygon(N, M, rotAng);
    glFlush();
    rotAng += M_PI / 180;
}
```

4.　テキストの図 3，図 4 のような図形（完全グラフ）を描き，回転させるプログラムを作成せよ．

4.1.　完全グラフ

完全グラフとは任意の 2 頂点間に必ずグラフがあるグラフのことをいう．頂点が n 個の完全グラフを K_n と表記し，辺の数は頂点の組み合わせの数になるので $n(n-1)/2$ となる．[8]

4.2.　プログラム

頂点の数が n の完全グラフ（K_n）を描画するプログラムを作成する．作成した完全グラフを描画する関数（dispCompleteGraph）をリスト 4 に示す．この関数は順に n，θ を引数としている．まず全ての頂点の座標を配列に格納し，その後全ての辺を描いている．関数 dispCompleteGraph を使い K_n を回転させるためのプログラムはリスト 3 の 9 行目のリスト 5 に書き換えたものとなる．$n = 7$ としたとき出力結果を図 3 に示す．

リスト 4　完全グラフを描画する関数（dispCompleteGraph）

```
void dispCompleteGraph(int n, double theta)
{
    int i, j;
    double dt;
    GLdouble p[n][2];
    dt = 2.0 * M_PI / n;
    for (i = 0; i < n; i++)
    {
        p[i][0] = cos(theta);
        p[i][1] = sin(theta);
        theta += dt;
    }
    glLineWidth(4);
    glBegin(GL_LINES);
    for (i = 0; i < n - 1; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            glVertex2dv(p[i]);
            glVertex2dv(p[j]);
        }
    }
    glEnd();
}
```

リスト 5　関数 dispCompleteGraph の呼び出し部分

```
    dispCompleteGraph(N, rotAng);
```

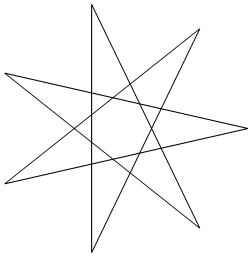


図 2　正 7/3 角形の出力結果

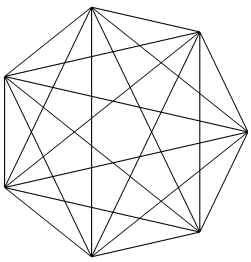


図 3　K_7 の出力結果

5. テキストのリスト 12 を解析し，数学関数のグラフを描くプログラムを作成せよ．例えば θ を 0 から 2π まで変化させながら，次の関数をプロットするとどんな図形が描かれるだろうか（コラム参照）．余力がある学生は，座標軸に目盛り（文字）を加えてみよ．

(a) カージオイド（cardioid）

$$x = [1 + \cos(\theta)]\cos(\theta) \tag{1}$$
$$y = [1 + \cos(\theta)]\sin(\theta) \tag{2}$$

(b) サイクロイド（cycloid）

$$x = \theta - \sin(\theta) \tag{3}$$
$$y = 1 - \cos(\theta) \tag{4}$$

(c) 4 尖点の内サイクロイド（hypocycloid）

$$x = \cos^3(\theta) \tag{5}$$
$$y = \sin^3(\theta) \tag{6}$$

5.1. プログラム

リスト 6 にグラフを描画するために作成した関数のプロトタイプ宣言を示す．関数 `funcX`，`funcY` に描くグラフの関数を設定する．関数 `setRange`，`setXtics`，`setYtics`，`setRatio` はグラフの設定パラメータを格納しているグローバル変数の書き換えをする関数で，2 つの軸の範囲，x 軸の目盛り，y 軸の目盛り，グラフの縦横比を設定する．関数 `dispGraph` はグラフを描くための関数で，関数内に目盛りを描くための関数 `drawXtics`，`drawYtics` を使用する．関数 `dispGraph` をリスト 7 に示す．5 から 20 行目でグラフの軸と目盛りを描画し，22 から 28 行目でグラフを描画している．関数 `setXtics`，`setYtics` をリスト 8，リスト 9 に示す．これらの関数はどちらも 7 から 15 行で $2l$ の長さの目盛りを引いている．17 から 29 行目で，`setXtics` は表示する数字を目盛りの下になるようにして `setYtics` は表示する数字を目盛りの左になるようにしている．

リスト 6　関数のプロトタイプ宣言部分

```
1 double funcX(double theta);
2 double funcY(double theta);
3 void dispGraph(void);
4 void drawXtics(double left, double right, double interval);
5 void drawYtics(double bottom, double top, double interval);
6 void setRange(double left, double right, double bottom, double top);
7 void setXtics(double left, double right, double interval);
8 void setYtics(double bottom, double top, double interval);
9 void setRatio(double w, double h);
```

リスト 7　グラフ描画関数（dispGraph）

```
void dispGraph(void)
{
    double x, y, theta;

    glLineWidth(1);
    glBegin(GL_LINES);
    glVertex2d(g_xmin, 0.0);
    glVertex2d(g_xmax, 0.0);
    glVertex2d(0.0, g_ymin);
    glVertex2d(0.0, g_ymax);
    glEnd();

    glRasterPos2d(g_xmax, 0.0);
    glutBitmapCharacter(GLUT_BITMAP_9_BY_15, 'X');
    glRasterPos2d(0.0, g_ymax);
    glutBitmapCharacter(GLUT_BITMAP_9_BY_15, 'Y');
    glEnd();

    drawXtics(g_xtics[0], g_xtics[1], g_xtics[2]);
    drawYtics(g_ytics[0], g_ytics[1], g_ytics[2]);

    glLineWidth(2);
    glBegin(GL_LINE_STRIP);
    for (theta = 0; theta < 2 * M_PI; theta += 0.05)
    {
        glVertex2d(funcX(theta), funcY(theta));
    }
    glEnd();
}
```

リスト 8　x 軸の目盛り描画関数（drawXtics）

```
void drawXtics(double left, double right, double interval)
{
    int num, i;
    double x, l;
    char str[16];

    glLineWidth(1);
    glBegin(GL_LINES);
    l = 8 * g_ysize / glutGet(GLUT_WINDOW_HEIGHT);
    for (x = left; x <= right; x += interval)
    {
        glVertex2d(x, -l);
        glVertex2d(x, l);
    }
    glEnd();

    for (x = left; x <= right; x += interval)
    {
        num = sprintf(str, "%6.2f", x);
        glRasterPos2d(
            x - 4 * num * g_xsize / glutGet(GLUT_WINDOW_WIDTH),
            -8 * g_ysize / glutGet(GLUT_WINDOW_HEIGHT) - l
        );
        for (i = 0; i < num; i++)
        {
            glutBitmapCharacter(GLUT_BITMAP_8_BY_13, str[i]);
        }
    }
    glEnd();
}
```

リスト 9　y 軸の目盛り描画関数（drawYtics）

```
1  void drawYtics(double bottom, double top, double interval)
2  {
3      int num, i;
4      double y, l;
5      char str[16];
6
7      glLineWidth(1);
8      glBegin(GL_LINES);
9      l = 8 * g_xsize / glutGet(GLUT_WINDOW_WIDTH);
10     for (y = bottom; y <= top; y += interval)
11     {
12         glVertex2d(-l, y);
13         glVertex2d(l, y);
14     }
15     glEnd();
16
17     for (y = bottom; y <= top; y += interval)
18     {
19         num = sprintf(str, "%6.2f", y);
20         glRasterPos2d(
21             -8 * num * g_xsize / glutGet(GLUT_WINDOW_WIDTH) - l,
22             y - 4 * g_ysize / glutGet(GLUT_WINDOW_HEIGHT)
23         );
24         for (i = 0; i < num; i++)
25         {
26             glutBitmapCharacter(GLUT_BITMAP_8_BY_13, str[i]);
27         }
28     }
29     glEnd();
30 }
```

5.2.　各関数の出力結果

この関数でカージオイド，サイクロイド，4 尖点の内サイクロイドを適切に軸の範囲や目盛りを設定して描画したグラフを図 4，図 5，図 6 に示す．

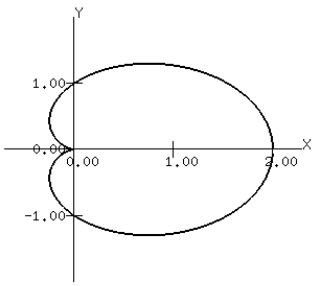


図 4　カージオイド

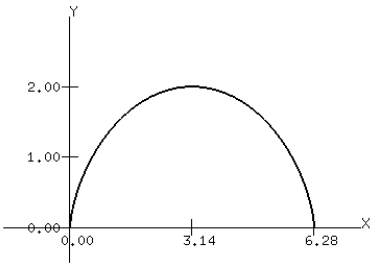


図 5　サイクロイド

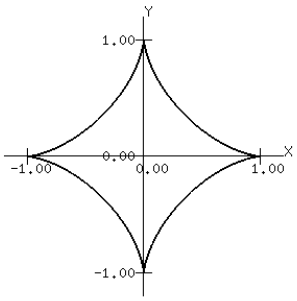


図 6　4 尖点の内サイクロイド

課題 II

1. テキストの図 6 に展開図を示す正四面体 ABCD [*4]について，△BCD の重心 G を原点 O と一致させ，正四面体の一辺の長さを w とするとき，各頂点の 3 次元座標を導出せよ．

△BCD の高さは一辺が w の正三角形なので，$\sqrt{3}w/2$ となる．△GBD が二等辺三角形となる．三平方の定理を用いて △GBD の等辺の長さは $w/\sqrt{3}$ となる．これらのことから正四面体の高さは三平方の定理を用いて，$\sqrt{2/3}w$ となる．よって各頂点の位置は次となる．

$$\boldsymbol{a} = \left(\sqrt{\frac{2}{3}}w, 0, 0\right) \qquad \boldsymbol{b} = \left(0, \frac{w}{\sqrt{3}}, 0\right) \qquad \boldsymbol{c} = \left(0, -\frac{w}{2\sqrt{3}}, \frac{w}{2}\right) \qquad \boldsymbol{d} = \left(0, -\frac{w}{2\sqrt{3}}, -\frac{w}{2}\right)$$

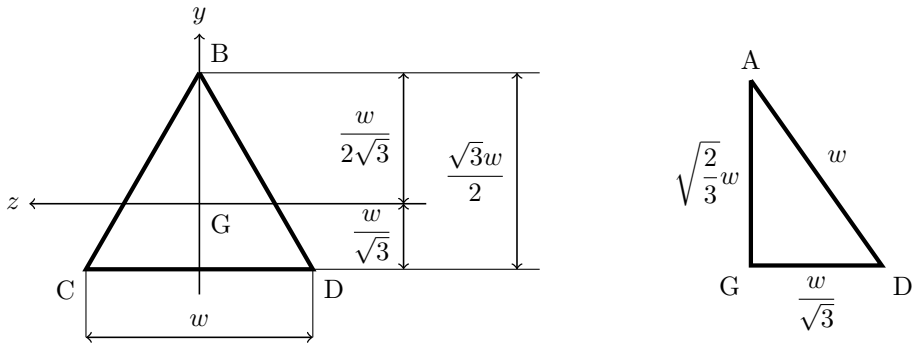


図 7　正四面体の一部の大きさ

2. 前問で求めた結果をもとに，原点 O を中心とする半径 1 の球に内接する正四面体の頂点がテキストのリスト 14 のように定まることを導出せよ．

正四面体の重心 $\boldsymbol{g}'$ が原点と一致させるために前問の正四面体を移動させる．重心 $\boldsymbol{g}'$ は，

$$\boldsymbol{g}' = \frac{\boldsymbol{a}+\boldsymbol{b}+\boldsymbol{c}+\boldsymbol{d}}{4} = \left(\frac{1}{4}\sqrt{\frac{2}{3}}w, 0, 0\right)$$

移動後の各頂点を $\boldsymbol{a}'$，$\boldsymbol{b}'$，$\boldsymbol{c}'$，$\boldsymbol{d}'$ とすると各頂点の位置は，

$$\boldsymbol{a}' = \boldsymbol{a} - \boldsymbol{g}' = \left(\frac{1}{2}\sqrt{\frac{3}{2}}w, 0, 0\right) \qquad \boldsymbol{b}' = \boldsymbol{b} - \boldsymbol{g}' = \left(-\frac{1}{4}\sqrt{\frac{2}{3}}w, \frac{w}{\sqrt{3}}, 0\right)$$

$$\boldsymbol{c}' = \boldsymbol{c} - \boldsymbol{g}' = \left(-\frac{1}{4}\sqrt{\frac{2}{3}}w, -\frac{w}{2\sqrt{3}}, \frac{w}{2}\right) \qquad \boldsymbol{d}' = \boldsymbol{d} - \boldsymbol{g}' = \left(-\frac{1}{4}\sqrt{\frac{2}{3}}w, -\frac{w}{2\sqrt{3}}, -\frac{w}{2}\right)$$

半径 1 の球に内接するとき $\boldsymbol{a}' = (1, 0, 0)$ となるので $w = 2\sqrt{2/3}$ となる．よって各頂点の位置は，

$$\boldsymbol{a}' = (1, 0, 0) \qquad \boldsymbol{b}' = \left(-\frac{1}{3}, \frac{2\sqrt{2}}{3}, 0\right) \qquad \boldsymbol{c}' = \left(-\frac{1}{3}, -\frac{\sqrt{2}}{3}, \frac{\sqrt{6}}{3}\right) \qquad \boldsymbol{d}' = \left(-\frac{1}{3}, -\frac{\sqrt{2}}{3}, -\frac{\sqrt{6}}{3}\right)$$

この位置から各頂点の値がテキストのリスト 14 のように定まることがわかる．

[*4] 本報告書では原点 O から $\boldsymbol{a}$ にある点を A と表記する．

3. テキストの **9.2** のアームロボットについて，キー入力で台座の回転と，各関節の傾斜角を制御できるような対話プログラムを作成してみよう．

アームロボットの各パーツの登録部分をリスト 10 に示す．ここでは台座，下腕，上腕の登録をしている．[*5] キーボードコールバック関数をリスト 11 に示す．押されたキーに対応したパーツの角度を変え再描画している．そして，テキストのリスト 20 と同様の関数 display を用いてアームロボットを描画する．

リスト 10　各パーツの登録（init の一部）

```
1  GLUquadric *g_quad;
2
3  void init(void)
4  {
5      glClearColor(0.0, 0.0, 0.0, 1.0);
6      g_quad = gluNewQuadric();
7      gluQuadricDrawStyle(g_quad, GLU_LINE);
8      glNewList(ID_B, GL_COMPILE);
9      glColor3f(1.0, 0.0, 0.0);
10     glPushMatrix();
11     glTranslatef(0.0, HEIGHT_B, 0.0);
12     glScalef(RADIUS_B, HEIGHT_B, RADIUS_B);
13     glRotatef(90, 1, 0, 0);
14     gluCylinder(g_quad, 1, 1, 1, 12, 6);
15     glPopMatrix();
16     glEndList();
17     glNewList(ID_L, GL_COMPILE);
18     glColor3f(0.0, 1.0, 0.0);
19     glPushMatrix();
20     glTranslatef(0.0, 0.5*HEIGHT_L, 0.0);
21     glScalef(WIDTH_L, HEIGHT_L, WIDTH_L);
22     glutWireCube(1.0);
23     glPopMatrix();
24     glEndList();
25     glNewList(ID_U, GL_COMPILE);
26     glColor3f(0.0, 0.0, 1.0);
27     glPushMatrix();
28     glTranslatef(0.0, 0.5*HEIGHT_U, 0.0);
29     glScalef(WIDTH_U, HEIGHT_U, WIDTH_U);
30     glutWireCube(1.0);
31     glPopMatrix();
32     glEndList();
33 }
```

リスト 11　キーボードコールバック関数（keyin）

```
1  GLfloat g_rotAng[3] = {0.0};
2
3  void keyin(unsigned char key, int x, int y)
4  {
5      switch(key)
6      {
7          case 'b': g_rotAng[0] += 1.0; glutPostRedisplay(); break;
8          case 'B': g_rotAng[0] -= 1.0; glutPostRedisplay(); break;
9          case 'l': g_rotAng[1] += 1.0; glutPostRedisplay(); break;
10         case 'L': g_rotAng[1] -= 1.0; glutPostRedisplay(); break;
11         case 'u': g_rotAng[2] += 1.0; glutPostRedisplay(); break;
12         case 'U': g_rotAng[2] -= 1.0; glutPostRedisplay(); break;
13         default: break;
14     }
15 }
```

[*5] ID_X や HEIGHT_X などはテキストのリスト 27 のようにマクロ定義してある．

4. タイマーコールバックを使って，アームロボットを制御するようなプログラムを作成してみよう．アームロボットが踊っているかのように見せるには，どんな工夫ができるだろう．

4.1. 踊ってるように見せるために

下腕と上腕を滑らかに手を振っているようにするには各角度 θ_{L}，θ_{U} を式 (7)，式 (8) とすることでできると考えた．（時刻を t とする．）

$$\theta_{\mathrm{L}} = A\sin[B\sin(2t)] \tag{7}$$

$$\theta_{\mathrm{U}} = A\sin[2B\sin(2t)] \quad （A\text{と}B\text{は任意定数}） \tag{8}$$

A と B を変えることで動きの速さを変えることができる．下腕と上腕が揺れているところをいろいろな角度から見れるように，台座を式 (9) で回転させる．

$$\theta_{\mathrm{B}} = Ct \quad （C\text{は任意定数}） \tag{9}$$

4.2. プログラム

$A = 45$，$B = 1.5$，$C = 5$ に設定して式 (7)，式 (8)，式 (9) をタイマーコールバック関数に実装する．作成したタイマーコールバック関数をリスト 12 に示す．これを全問のキーボードコールバック関数の代わりに使いアームロボットを踊らせる．

リスト 12　タイマーコールバック関数（timer）

```
1  #define DT 100
2  GLfloat g_time = 0.0;
3
4  static void timer(int dummy)
5  {
6      GLfloat tmp;
7      glutTimerFunc(DT, timer, 0);
8      glutPostRedisplay();
9      tmp = 1.5 * sin(2 * g_time);
10     g_rotAng[0] = 5 * g_time;
11     g_rotAng[1] = 45 * sin(tmp);
12     g_rotAng[2] = 45 * sin(2 * tmp);
13     g_time += DT / 1000.0;
14 }
```

課題 III

1. 空間中の任意の 3 点 $\mathrm{P}_i(x_i, y_i, z_i)$，$i = 1, 2, 3$ が与えられたとき，$\triangle \mathrm{P}_1\mathrm{P}_2\mathrm{P}_3$ の単位法線ベクトルをすべて求める式を導出せよ．

$\triangle \mathrm{P}_1\mathrm{P}_2\mathrm{P}_3$ を含む平面上の互いに平行でない 2 つのベクトルの外積は，その平面の法線ベクトルになる．ここで 2 つのベクトルを $\overrightarrow{\mathrm{P}_1\mathrm{P}_2}$ と $\overrightarrow{\mathrm{P}_1\mathrm{P}_3}$ とすると，$\triangle \mathrm{P}_1\mathrm{P}_2\mathrm{P}_3$ のすべての単位法線ベクトルは，

$$\pm \frac{\overrightarrow{\mathrm{P}_1\mathrm{P}_2} \times \overrightarrow{\mathrm{P}_1\mathrm{P}_3}}{\left|\overrightarrow{\mathrm{P}_1\mathrm{P}_2} \times \overrightarrow{\mathrm{P}_1\mathrm{P}_3}\right|}$$

となる．

2. 空間中の任意の 3 点 $\mathrm{P}_i(x_i, y_i, z_i)$，$i = 1, 2, 3$ が与えられたとき，$\triangle \mathrm{P}_1\mathrm{P}_2\mathrm{P}_3$ のどちらが表（CCW）で，どちらが裏（CW）と判定できるか，判定方法を導出せよ．

右手系の外積で，$\overrightarrow{\mathrm{P}_1\mathrm{P}_2} \times \overrightarrow{\mathrm{P}_1\mathrm{P}_3}$ が正となるとき表になる．負のとき裏になる．

3. Phong の照光モデルについて，詳しく調べよ．

3.1. 発表

Phong の照光モデル[*6]（Phong reflection model）はユタ大学の理学博士である Bui Tuong Phong によって開発され，1973 年に学位論文 [9]，1975 年に論文（Illumination for Computer Generated Pictures）[10] として発表された．

3.2. 概要

現実の光源では反射現象により物体に複数の方向からいろいろな光が当たる．コンピュータ上でこれらを全て扱うことは非常に困難であるため，単純化を行う必要がある．ここで反射現象を単純化するために Phong の照光モデルを用いる．このモデルは物体の表面からの反射光を環境光の反射光[*7]，直接光の拡散反射光[*8]，直接光の鏡面反射光[*9]の 3 つの反射光の和で近似する．

3.3. 計算方法

物体表面の輝度を L_r，環境光の反射光の輝度を L_a，直接光の拡散反射光の輝度を L_d，直接光の鏡面反射光の輝度を L_s とすると，L_r は式 (10) となる．

$$L_\mathrm{r} = L_\mathrm{a} + L_\mathrm{d} + L_\mathrm{s} \tag{10}$$

環境光反射係数を k_a，環境光の照度を E_a とすると，L_a は式 (11) となる．

$$L_\mathrm{a} = k_\mathrm{a} E_\mathrm{a} \tag{11}$$

拡散反射係数を k_d，入射光の照度を E_i，物体表面の単位法線ベクトルを $\boldsymbol{n}$，光の入射方向の単位ベクトルを $\boldsymbol{l}$ とすると，L_d は式 (12) となる．

$$L_\mathrm{d} = k_\mathrm{d} E_\mathrm{i} (\boldsymbol{n} \cdot \boldsymbol{l}) \tag{12}$$

鏡面反射係数を k_s，入射光の照度を E_i，視線と逆向きの単位ベクトルを $\boldsymbol{v}$，入射光の正反射単位ベクトルを $\boldsymbol{r}$，光沢度を α とすると，L_s は式 (13) となる．

$$L_\mathrm{s} = k_\mathrm{s} E_\mathrm{i} (\boldsymbol{v} \cdot \boldsymbol{r})^\alpha \tag{13}$$

式 (11)，式 (12)，式 (13) より式 (10) は式 (14) となる．

$$L_\mathrm{r} = k_\mathrm{a} E_\mathrm{a} + k_\mathrm{d} E_\mathrm{i} (\boldsymbol{n} \cdot \boldsymbol{l}) + k_\mathrm{s} E_\mathrm{i} (\boldsymbol{v} \cdot \boldsymbol{r})^\alpha \tag{14}$$

[*6] Phong の反射モデルとも呼ばれる．
[*7] 環境光は間接光とも呼ばれ，別の物体表面での反射を経て物体表面に届く光のこと．
[*8] 拡散反射光は，反射角にほとんど依存せず多様な方向に反射した光のこと．
[*9] 鏡面反射光は反射角が入射角と等しくなるように反射した光のこと．

4. 正四面体，正六面体，正八面体をスムーズシェーディングで表示するために，各頂点の単位法線ベクトルをどのように定めればよいか考え，実装せよ．

4.1. 頂点の単位法線ベクトル

頂点の単位法線ベクトルは重心から頂点に向かうベクトルと同じ方向になる．よって原点を中心とする半径 1 の球に内接する正多面体の頂点ベクトルが頂点の単位法線ベクトルとなる．

4.2. プログラム

テキストのリスト 21 と演習 19 で求めた正四面体，正六面体，正八面体の頂点の座標を使い各正多面体をスムーズシェーディングで表示するプログラムを作成する．正四面体をスムーズシェーディングで表示するプログラムの主要部をリスト 13 に示す．g_vP は頂点の座標，g_tp はトポロジ情報を格納してある．そして関数 display で正四面体を表示している．正六面体，正八面体をスムーズシェーディングで表示するプログラムの主要部をリスト 14，リスト 15 に示す．これらも同様の処理で表示している．スムーズシェーディングで表示された正多面体を図 8 に示す．[*10]

リスト 13　正四面体をスムーズシェーディングで表示するプログラム（主要部）

```
GLdouble g_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 g_tP[4][3] = {
    {0, 1, 2}, {0, 3, 1}, {0, 2, 3}, {1, 3, 2}
};

void display(void)
{
    int i, j;
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glBegin(GL_TRIANGLES);
    for (i = 0; i < 4; i++)
    {
        for (j = 0; j < 3; j++)
        {
            glNormal3dv(g_vP[g_tP[i][j]]);
            glVertex3dv(g_vP[g_tP[i][j]]);
        }
    }
    glEnd();
    glutSwapBuffers();
}
```

リスト 14　正六面体をスムーズシェーディングで表示するプログラム（主要部）

```
GLdouble g_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},
```

[*10] 光源はデフォルトの GL_LIGHT0 に設定してある．

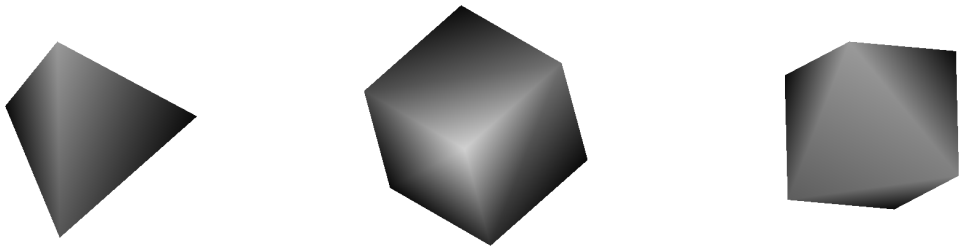
```
    { 0.577,  0.577,  0.577},
    { 0.577,  0.577, -0.577},
    {-0.577,  0.577, -0.577}
};
int g_tP[6][4] = {
    {3, 2, 1, 0}, {0, 1, 5, 4}, {1, 2, 6, 5},
    {4, 5, 6, 7}, {3, 7, 6, 2}, {0, 4, 7, 3}
};

void display(void)
{
    int i, j;
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glBegin(GL_QUADS);
    for (i = 0; i < 6; i++)
    {
        for (j = 0; j < 4; j++)
        {
            glNormal3dv(g_vP[g_tP[i][j]]);
            glVertex3dv(g_vP[g_tP[i][j]]);
        }
    }
    glEnd();
    glutSwapBuffers();
}
```

リスト 15　正八面体をスムーズシェーディングで表示するプログラム（主要部）

```
GLdouble g_vP[6][3] = {
    { 1.000,  0.000,  0.000},
    { 0.000,  1.000,  0.000},
    { 0.000,  0.000,  1.000},
    {-1.000,  0.000,  0.000},
    { 0.000, -1.000,  0.000},
    { 0.000,  0.000, -1.000}
};
int g_tP[8][3] = {
    {0, 1, 2}, {5, 1, 0}, {3, 1, 5}, {2, 1, 3},
    {2, 4, 0}, {0, 4, 5}, {5, 4, 3}, {3, 4, 2}
};

void display(void)
{
    int i, j;
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glBegin(GL_TRIANGLES);
    for (i = 0; i < 8; i++)
    {
        for (j = 0; j < 3; j++)
        {
            glNormal3dv(g_vP[g_tP[i][j]]);
            glVertex3dv(g_vP[g_tP[i][j]]);
        }
    }
    glEnd();
    glutSwapBuffers();
}
```



(a)　正四面体　　(b)　正六面体　　(c)　正八面体

図 8　スムーズシェーディングで表示された正多面体

5.　テキストのリスト 27 を参考に，アームロボットをシェーディング表示するものに変更してみよ．

各パーツの角度を格納する g_rotAng をグローバル変数に定義する．そして各パーツを描画する前に関数 glRotated を用いて回転させる．g_rotAng の値を書き換えるためにリスト 11 のキーボードコールバック関数を用いた．ソースコードの他の部分はテキストのリスト 27 と同様である．描画されたアームロボットを図 9 に示す．

リスト 16　アームロボットをシェーディング表示するプログラム（主要部）

```
1  GLfloat g_rotAng[3] = {0.0};
2
3  void display(void)
4  {
5      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
6      glMatrixMode(GL_MODELVIEW);
7      glLoadIdentity();
8      gluLookAt(1.0, 1.0, 1.0, 0.0, 0.5, 0.0, 0.0, 1.0, 0.0);
9      glRotated(g_rotAng[0], 0, 1, 0);
10     glCallList(ID_B);
11     glTranslatef(0.0, HEIGHT_B, 0.0);
12     glRotated(g_rotAng[1], 0, 0, 1);
13     glCallList(ID_L);
14     glTranslatef(0.0, HEIGHT_L, 0.0);
15     glRotated(g_rotAng[2], 0, 0, 1);
16     glCallList(ID_U);
17     glutSwapBuffers();
18 }
```

図 9　シェーディング表示されたアームロボット

感想

課題で調べごとをするときに，できるだけ英語の文献を参考にするようにした．本科目の目的とは関係ないが，英語の苦手意識を少しでも軽減できたり，英語の文献の検索や読む練習になったりするので努力した．また，日本語でまとめているサイトではなく公式のサイトを参考にする場合しっかりとした情報が得られてよかった．これから更に英語の文献に触れる機会が増えると思うので英語に慣れていきたい．演習や課題の手応えはちょうどよくスムーズに進んでよかった．受講して非常に良かった．（受講しない場合は研究室を破門されるが...）

改善案

課題 III-5 はリストを写すだけなので演習にするべきだと思う．関数 abc や abc 関数や abc*の表記の違いがわかりにくいです．違いがないなら統一したほうが良いのではないでしょうか．

参考文献・URL

[1] 高橋章，R01-Ec5 プログラミング演習 IV テキスト
[2] GNU Make，http://www.gnu.org/software/make/
[3] GNU Grep，http://www.gnu.org/software/grep/
[4] touch，
http://www.gnu.org/software/coreutils/manual/html_node/touch-invocation.html
[5] 変数のスコープ，http://www.isl.ne.jp/pcsp/beginC/C_Language_09.html
[6] C における識別子の有効範囲と変数の生存期間，
https://www.atmarkit.co.jp/ait/articles/1104/26/news107.html
[7] Star Polygon，http://mathworld.wolfram.com/StarPolygon.html
[8] Complete Graph，http://mathworld.wolfram.com/CompleteGraph.html
[9] University of Utah School of Computing，https://www.cs.utah.edu/about/history/#phong-ref
[10] Illumination for Computer Generated Pictures，Bui Tuong Phong，
https://users.cs.northwestern.edu/~ago820/cs395/Papers/Phong_1975.pdf
[11] Phong の反射モデル，https://knzw.tech/raytracing/?page_id=240