1　課題１

1.1　課題 1-1

1.1.1　MAKE ユーティリティ

ウェブサイト [1] によれば，MAKE は，大規模プログラムのコンパイルの簡略化を行うツールである。プログラムが大規模になると，ファイル間の親子関係が複雑になってしまい，コンパイル作業が煩雑になる。そこで，MAKEFILE にその親子関係を記述しておくと，各ファイルの更新を取得して必要なものだけをコンパイルできるようになる。以下のコマンドから，予め作った MAKE ファイルを用いてコンパイルができる。

```
1 make target
```

1.1.2　grep

ウェブサイト [2] によれば，grep は，ファイルの中の文字列を検索するコマンドである。オプションを指定することで大文字小文字の区別をするかの指定や，検索結果の表示内容の変更などができる。以下のように使用する。

```
1 grep オプション検索する文字列検索されるファイル
```

1.1.3　touch

ウェブサイト [3] によれば，touch はファイルのタイムスタンプのうちの，修正日時 (mtime) とアクセス時間 (atime) を書き換えるコマンドである。以下のように使用する。

```
1 touch オプション　ファイル名
```

1.2　課題 1-2

C 言語では，変数は，その宣言を書く場所によって有効範囲が異なる。この有効範囲をスコープという。変数のスコープ外からは，その変数を参照できない。一方，どの関数にも属さない場所で宣言された変数をグローバル変数と呼び，プログラム内のどこからでもアクセスが可能である。記憶域期間には自動記憶域期間と静的記憶域期間がある。自動記憶域期間を持つ変数は auto を使用して宣言されるが，auto は省略できる。この変数の生存期間は宣言した関数内に入った時から出る時までである。このように，関数内で宣言され，自動記憶域期間をもつ変数をローカル変数という。一方，static を使用して宣言された変数は静的記憶域期間を持ち，生存期間はプログラムの開始から終了までとなる。このような変数をスタティック変数と呼ぶ。

1.3　課題 1-3

星形正多角形を描き，回転させるプログラムの主要部をリスト 1 に示す。

Listing 1　星形正多角形を描画するプログラム (主要部)

```
1 #define STAR 5.0
```

```
2
3 double rotAng=0.0;
4
5 void display(void){
6         int i;
7         double theta,dt,x,y;
8         glClear(GL_COLOR_BUFFER_BIT);
9         glColor3d(1.0,0.0,0.0);
10        dt=2.0*M_PI/STAR *2.0;
11        theta=rotAng;
12        glBegin(GL_LINES);
13        for (i=0;i<(int)STAR*2;i++){
14                x=cos(theta);
15                y=sin(theta);
16                glVertex2d(x,y);
17                theta+=dt;
18        }
19        glEnd();
20        glutSwapBuffers();
21        rotAng+=3.0*M_PI/180.0;
22 }
```

定数 STAR の値を変更することで，角の数を変更できる。このプログラムでは星形正五角形を描いている。タイマコールバック関数によって，100 ミリ秒毎に処理が行われる。

鋭角となる点を一つ飛ばしに結ぶことで星形正多角形は描くことができる。一つ飛ばしの 2 点と図形の中心点がなす角度を計算して dt に格納し，これを用いて theta をインクリメントすることで鋭角となる点を求める。また関数が呼び出されるたびに rotAng の値を増やすことで，図形を回転させている。

1.4 課題 1-4

完全グラフを描画するプログラムの主要部をリスト 2 に示す。

Listing 2　完全グラフを描画するプログラム (主要部)

```
1 #define STAR 7.0
2
3 double rotAng=0.0;
4
5 void display(void){
6         int i;
7         double j;
8         double theta,dt,x,y;
9         glClear(GL_COLOR_BUFFER_BIT);
10        glColor3d(1.0,0.0,0.0);
11        glBegin(GL_LINES);
12        for (j=1.0;j<STAR/2.0;j++){
13                theta=rotAng;
```

```
14                  dt=2.0*M_PI/STAR *j;
15                  for (i=0;i<(int)STAR*2;i++){
16                          x=cos(theta);
17                          y=sin(theta);
18                          glVertex2d(x,y);
19                          theta+=dt;
20                  }
21          }
22          glEnd();
23          glutSwapBuffers();
24          rotAng+=3.0*M_PI/180.0;
25  }
```

定数 STAR の値を変更することで，頂点の数を変更できる。このプログラムでは K_7 を描いている。タイマコールバック関数によって，100 ミリ秒毎に処理が行われる。

すべての点同士をそれぞれ線分で結べばよい。結ぶ 2 点と図形の中心点がなす角度を計算して dt に格納し，これを用いて theta をインクリメントすることで結んでいく点を求める。この処理を，結ぶ点の間隔が隣り合っている場合から 1 点飛ばしの場合，2 点飛ばしの場合，・・・と順番に行っていく。また関数が呼び出されるたびに rotAng の値を増やすことで，図形を回転させていることは前節と同様である。

1.5 課題 1-5

カージオイド，サイクロイド，4 尖点の内サイクロイドを描画するプログラムについて，display 関数のみ，それぞれリスト 3,4,5 に示す。サイクロイドについては，画面に収めるために y 軸方向にグラフを 0.5 倍に圧縮している。

Listing 3　カージオイドを描画するプログラム (主要部)

```
1  void display(void){
2          double theta,x,y;
3          glClear(GL_COLOR_BUFFER_BIT);
4          glColor3d(0.0,0.0,0.0);
5          glBegin(GL_LINES);
6          glVertex2d(-0.5,0.0);
7          glVertex2d(3.2,0.0);
8          glVertex2d(0.0,-M_PI);
9          glVertex2d(0.0,M_PI);
10         for(x=0;x<=3;x+=1){
11                 glVertex2d(x,-0.05);
12                 glVertex2d(x,0.05);
13         }
14         for(y=-1.5;y<=1.5;y+=0.5){
15                 glVertex2d(-0.05,y);
16                 glVertex2d(0.05,y);
17         }
18         glEnd();
```

```
19          glColor3d(0.0,0.0,0.0);
20          glBegin(GL_LINE_STRIP);
21          for(theta=0;theta<2*M_PI;theta+=0.02*M_PI){
22                  x=cos(theta)*(1+cos(theta));
23                  y=sin(theta)*(1+cos(theta));
24                  glVertex2d(x,y);
25          }
26          glEnd();
27          glFlush();
28  }
```

Listing 4　サイクロイドを描画するプログラム (主要部)

```
1  void display(void){
2          double theta,x,y;
3          glClear(GL_COLOR_BUFFER_BIT);
4          glColor3d(0.0,0.0,0.0);
5          glBegin(GL_LINES);
6          glVertex2d(-0.5,0.0);
7          glVertex2d(2*M_PI+0.5,0.0);
8          glVertex2d(0.0,-0.5);
9          glVertex2d(0.0,M_PI);
10         for(x=0;x<=M_PI*4;x+=M_PI){
11                 glVertex2d(x/2,-0.05);
12                 glVertex2d(x/2,0.05);
13         }
14         for(y=0;y<=1.0;y+=0.5){
15                 glVertex2d(-0.05,y);
16                 glVertex2d(0.05,y);
17         }
18         glEnd();
19         glColor3d(0.0,0.0,0.0);
20         glBegin(GL_LINE_STRIP);
21         for(theta=0;theta<2*M_PI;theta+=0.02*M_PI){
22                 x=theta-sin(theta);
23                 y=1-cos(theta);
24                 y/=2.0;
25                 glVertex2d(x,y);
26         }
27         glEnd();
28         glFlush();
29 }
```

Listing 5　4 尖点の内サイクロイドを描画するプログラム (主要部)

```
1  void display(void){
2          double theta,x,y;
```

```
3         glClear(GL_COLOR_BUFFER_BIT);
4         glColor3d(0.0,0.0,0.0);
5         glBegin(GL_LINES);
6         glVertex2d(-1.2,0.0);
7         glVertex2d(1.2,0.0);
8         glVertex2d(0.0,-1.2);
9         glVertex2d(0.0,1.2);
10        for(x=-1.0;x<=1.0;x+=0.5){
11                glVertex2d(x,-0.05);
12                glVertex2d(x,0.05);
13        }
14        for(y=-1.0;y<=1.0;y+=0.5){
15                glVertex2d(-0.05,y);
16                glVertex2d(0.05,y);
17        }
18        glEnd();
19        glColor3d(0.0,0.0,0.0);
20        glBegin(GL_LINE_STRIP);
21        for(theta=0;theta<2*M_PI;theta+=0.02*M_PI){
22                x=cos(theta)*cos(theta)*cos(theta);
23                y=sin(theta)*sin(theta)*sin(theta);
24                glVertex2d(x,y);
25        }
26        glEnd();
27        glFlush();
28 }
```

描画する図形によって軸の長さなどを変更している。

描画したグラフを図 1,3,2 に示す。カージオイドの x 軸，y 軸の目盛の間隔は 1,0.5 であり，サイクロイドの x 軸，y 軸の目盛の間隔は $\frac{\pi}{2}$,0.5，4 尖点の内サイクロイドの x 軸，y 軸の目盛の間隔は 0.5,0.5 である。

2 課題 2

2.1 課題 2-1

テキストの展開図のように A, B, C, D, H, I, J, G を定める。$\triangle BCD$ が正三角形であることから，$HB = IC = JD = \frac{\sqrt{3}}{2}w$ である。三角形の重心は三角形の頂点と対辺の 2 等分点を結ぶ線分を 2 : 1 に内分するから，$GB = GC = GD = \frac{1}{\sqrt{3}}w$,$GH = GI = GJ = \frac{1}{2\sqrt{3}}w$ となる。正四面体の高さ AG は，三平方の定理より $AG = \sqrt{IA^2 - IG^2} = \sqrt{\frac{2}{3}}w$ となる。よって各頂点の位置は，

$$A : (\sqrt{\frac{2}{3}}w, 0, 0)$$
$$B : (0, \frac{1}{\sqrt{3}}w, 0)$$
$$C : (0, -\frac{1}{2\sqrt{3}}w, \frac{w}{2})$$

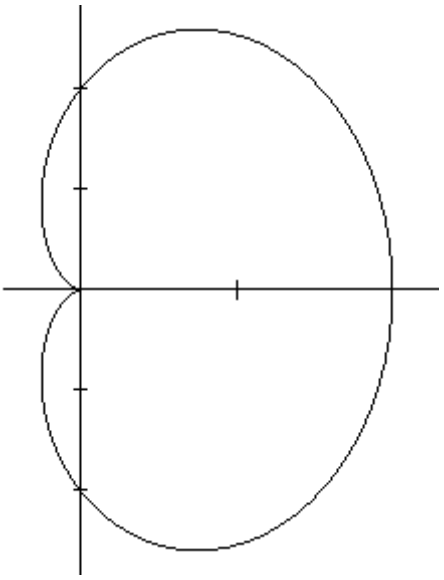

図 1　カージオイド

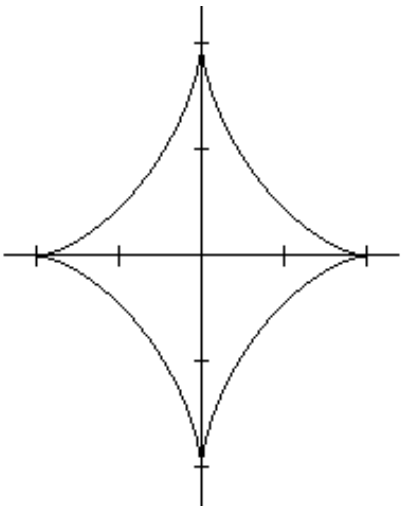

図 2　4 尖点の内サイクロイド

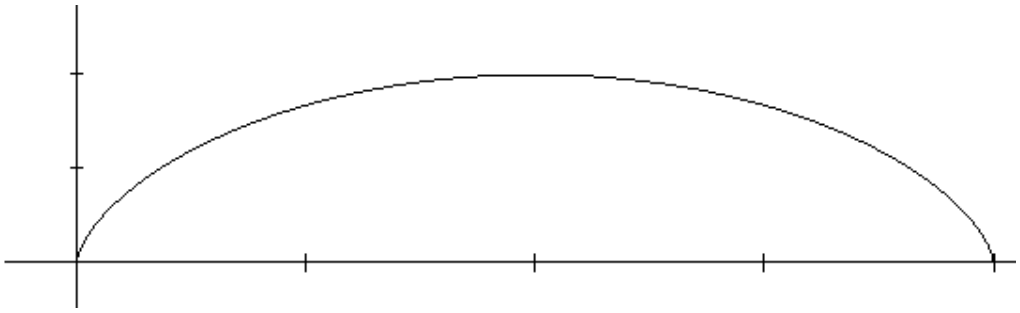

図 3　サイクロイド

$$D : (0, -\frac{1}{2\sqrt{3}}w, -\frac{w}{2})$$

となる。

2.2　課題 2-2

正四面体の重心は,

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

重心が原点 O に一致するように各頂点を平行移動すると,

$$A' : (\frac{1}{2}\sqrt{\frac{3}{2}}w, 0, 0)$$

$$B' : (-\frac{1}{4}\sqrt{\frac{2}{3}}w, \frac{1}{\sqrt{3}}w, 0)$$
$$C' : (-\frac{1}{4}\sqrt{\frac{2}{3}}w, -\frac{1}{2\sqrt{3}}w, \frac{w}{2})$$
$$D' : (-\frac{1}{4}\sqrt{\frac{2}{3}}w, -\frac{1}{2\sqrt{3}}w, -\frac{w}{2})$$

ここで，半径 1 の球に内接する正四面体を考えているから，$A' : (1, 0, 0)$ である。よって，$w = 2\sqrt{\frac{2}{3}}$ となる。これを代入して，正四面体の頂点を求めると以下のようになる。

$$A' : (1, 0, 0)$$
$$B' : (-\frac{1}{3}, \frac{2\sqrt{2}}{3}, 0)$$
$$C' : (-\frac{1}{3}, -\frac{\sqrt{2}}{3}, \frac{\sqrt{6}}{3})$$
$$D' : (-\frac{1}{3}, -\frac{\sqrt{2}}{3}, -\frac{\sqrt{6}}{3})$$

2.3 課題 2-3

ロボットアームの各パーツの登録部分をリスト 6 に示す。

Listing 6　ロボットアームの各パーツの登録部分

```
1  void init(void){
2          myQuad=gluNewQuadric();
3          gluQuadricDrawStyle(myQuad,GLU_LINE);
4          glClearColor(1.0,1.0,1.0,1.0);
5          gluLookAt(1,1,1,0,0,0,0,1,0);
6
7          glNewList(ID_B,GL_COMPILE);
8          glColor3f(0.0,0.0,0.0);
9          glPushMatrix();
10         glTranslatef(0.0, HEIGHT_B, 0.0);
11         glScalef(0.5,HEIGHT_B,0.5);
12         glRotated(90.0,1.0,0.0,0.0);
13         gluCylinder(myQuad,1,1,2,50,4);
14         glPopMatrix();
15         glEndList();
16
17         glNewList(ID_L,GL_COMPILE);
18         glColor3f(0.0,0.0,0.0);
19         glPushMatrix();
20         glTranslatef(0.0,0.5*HEIGHT_L,WIDTH_L);
21         glScalef(WIDTH_L,HEIGHT_L,WIDTH_L);
22         glutWireCube(1.0);
23         glPopMatrix();
```

```
24          glEndList();
25
26          glNewList(ID_U,GL_COMPILE);
27          glColor3f(0.0,0.0,0.0);
28          glPushMatrix();
29          glTranslatef(0.0,0.5*HEIGHT_U,WIDTH_U);
30          glScalef(WIDTH_U,HEIGHT_U,WIDTH_U);
31          glutWireCube(1.0);
32          glPopMatrix();
33          glEndList();
34  }
```

次に，キーボードコールバック関数をリスト 7 に示す。SPEED は定数マクロである。

Listing 7　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 'a':
9                  rotAng[0]+=SPEED;
10                 break;
11                 case 's':
12                 rotAng[0]-=SPEED;
13                 break;
14                 case 'd':
15                 rotAng[1]+=SPEED;
16                 break;
17                 case 'f':
18                 rotAng[1]-=SPEED;
19                 break;
20                 case 'h':
21                 rotAng[2]+=SPEED;
22                 break;
23                 case 'j':
24                 rotAng[2]-=SPEED;
25                 break;
26         }
27 }
```

このキーボードコールバック関数を通じて各パーツの角度を変更し，テキストのリスト 20 のディスプレイ関数によって描画処理を行う。

描画されたロボットアームを図 4 に示す。

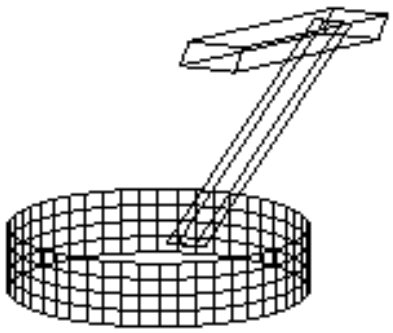

図 4　描画されたロボットアーム

2.4　課題 2-4

実装したタイマコールバック関数をリスト 8 に示す。

Listing 8　タイマコールバック関数

```
1  static void timer(int dummy){
2          glutTimerFunc(100,timer,0);
3          glMatrixMode(GL_MODELVIEW);
4          if (isAnime==1){
5                  rotAng[0]+=2.0;
6                  rotAng[1]=sin(speed)*10.0;
7                  rotAng[2]=sin(speed*4)*10.0;
8                  speed+=0.2;
9          }
10         glutPostRedisplay();
11 }
```

上腕を下腕より速く揺らすことによって，手を振って踊っているように見えると考えた。揺らす処理には sin 関数を用いた。また，台座も一定速度で回転させてみた。前節のキーボードコールバック関数の代わりに，タイマコールバック関数を呼び出すことで，踊っているような処理を実現した。

3　課題 3

3.1　課題 3-1

2 つのベクトルの外積は，そのベクトルがなす平面の法線ベクトルになる。すなわち，3 点 P_1, P_2, P_3 を含む平面の法線ベクトルは，$\overrightarrow{P_1P_2} \times \overrightarrow{P_1P_3}$ によって求められる。よって，求めるすべての単位法線ベクトルは，

$$\pm \frac{\overrightarrow{P_1P_2} \times \overrightarrow{P_1P_3}}{|\overrightarrow{P_1P_2} \times \overrightarrow{P_1P_3}|}$$

となる。

3.2 課題 3-2

右手系の外積で，$\overrightarrow{P_1P_2} \times \overrightarrow{P_1P_3}$ が正となるとき表で，負となるとき裏であると判定できる。

3.3 課題 3-3

ウェブサイト [4] によれば，Phong の照光モデルは Phong の反射モデルとも呼ばれる，3 次元コンピュータグラフィックスにおいて，モデリングされた面上の点に影をつけるための照明と陰影 (シェーディング) モデルである。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”Illumination for Computer Generated Pictures”の題で学位論文として発表された。

現実世界では，物体は光源からの光のほか，反射光などにより様々な方向から光が当たる。しかし，これらすべてをそのままプログラミングすることは難しいため，Phong の照光モデルによって単純化を行う。このモデルでは，面上の点における陰影を決定する際に，次のような単純化ができる利点がある。

- このモデルは，「局所的な」反射モデルである。すなわち，ラジオシティのようなレイトレーシングで行うような二次反射を計算する必要はない。反射した光の減衰を補正するために，外部の「環境光」(ambient) の項をレンダリングする際に加えている。
- 表面からの反射を 3 つの項目, すなわち「鏡面反射」(specular reflection),「拡散反射」(diffuse reflection) と「環境反射」(ambient reflection) に分けている。

Phong の反射モデルの式は以下のようになる。

$$I_p = k_a i_a + \sum_{\text{lights}} (k_d(\boldsymbol{L} \cdot \boldsymbol{N}) i_d + k_s(\boldsymbol{R} \cdot \boldsymbol{V})^\alpha i_s)$$

各文字の定義を以下に示す。

I_p : 表面上の各点における光の強度
k_s : 鏡面反射係数。入射光に対する鏡面反射率
k_d : 拡散反射係数。入射光に対する拡散反射率
k_a : 環境反射係数。シーン全体を照らす環境光の反射率
i_a : 環境光の照度
i_d : 入射光の拡散反射成分
i_s : 入射光の鏡面反射成分
α : 材質の光沢度。光沢のある点から反射する光がどのくらい均等に反射するか
$\boldsymbol{L}$: 物体表面上の点からそれぞれの光源 (light) への方向ベクトル
$\boldsymbol{N}$: 表面上の点における法線
$\boldsymbol{R}$: 面上のその点において光線が完全に反射 (reflect) される方向
$\boldsymbol{V}$: 視点に向かう方向

総和記号は光源についてすべての和をとることを意味する。総和記号の外にある項が環境光の反射による強度についての項，総和記号の中の 1 項目が光源からの光の拡散反射に関する項，2 項目が鏡面反射に関する項である。

3.4 課題 3-4

頂点の法線ベクトルは，重心からその頂点に向かう向きとなる。よって，原点を中心とする半径 1 の円に内接する正多面体の頂点の法線ベクトルは，その頂点の位置ベクトルと一致する。正四面体，正六面体，正八面体を描画するプログラムの主要部をリスト 9,10,11 にそれぞれ示す。

Listing 9　正四面体を描画するプログラム (主要部)

```
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},{0,3,1},{0,2,3},{1,3,2}};

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

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

}
```

Listing 10　正六面体を描画するプログラム (主要部)

```
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]={{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);

        glBegin(GL_QUADS);
        for(i=0;i<6;i++){
                for(j=0;j<4;j++){
```

```
15                          glNormal3dv(vP[tP[i][j]]);
16                          glVertex3dv(vP[tP[i][j]]);
17                  }
18          }
19          glEnd();
20          glutPostRedisplay();
21          glutSwapBuffers();
22
23  }
```

Listing 11　正八面体を描画するプログラム (主要部)

```
1  GLdouble vP[6][3]={{1,0,0},{0,1,0},{0,0,1},{-1,0,0},{0,-1,0},{0,0,-1}};
2  int 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}};
3
4
5  void display(void){
6          int i,j;
7          glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
8          glMatrixMode(GL_MODELVIEW);
9
10         glBegin(GL_TRIANGLES);
11         for(i=0;i<8;i++){
12                 for(j=0;j<3;j++){
13                         glNormal3dv(vP[tP[i][j]]);
14                         glVertex3dv(vP[tP[i][j]]);
15                 }
16         }
17         glEnd();
18         glutPostRedisplay();
19         glutSwapBuffers();
20
21 }
```

vP に頂点の位置，tP にトポロジ情報を格納している。正六面体，正八面体の頂点の位置，トポロジ情報は，演習 19 で求めたものである。法線ベクトルの指定には，vP をそのまま利用している。描画された正四面体，正六面体，正八面体をそれぞれ図 5,6,7 に示す。

3.5　課題 3-5

各パーツを回転させることができるように各パーツの回転角度を格納する変数 rotAng をグローバル変数として定義し，display 関数内での各パーツの呼び出し前に glRotated 関数を挿入した。他の部分はテキストのリスト 27 と同じである。テキストのリスト 27 との変更部分をリスト 12 に示す。

Listing 12　リアルなロボットアームを描画するプログラム (変更部分)

```
1  GLdouble rotAng[3]={0};
```

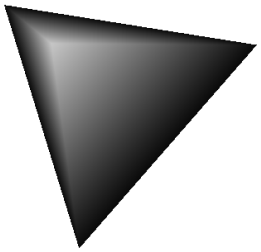

図 5　正四面体

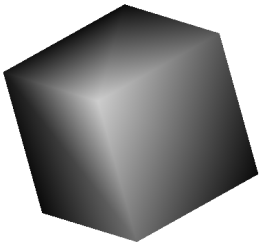

図 6　正六面体

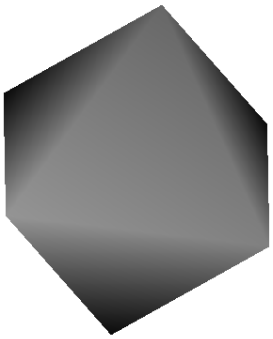

図 7　正八面体

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

描画されたロボットアームを図 8 に示す。

4　感想

これまで履修してきた情報系の授業に比べて，グラフィックスに関する内容が多かったため，実装したことが視覚的に強く反映されたので，楽しかった。一応 C 言語ではあるものの，初めて扱う OpenGL の関数が多

図 8　リアルなロボットアーム (ロケット?)

く出てきて難しかった。今後もプログラミングをする場面がたくさんあると思うので，継続的にプログラミング技術を高めていきたいと思った。

5　改善案

付録として，主要な OpenGL の関数がまとめてあると便利だと思いました。あと，授業内容とは少しずれますが，4 年生も履修可能な科目にしてもいいと思いました。

参考文献

[1] https://qiita.com/keitean/items/2f95dfb2944895f001e7

[2] https://eng-entrance.com/linux-command-grep

[3] https://kazmax.zpp.jp/cmd/t/touch.1.html

[4] https://ja.wikipedia.org/wiki/Phong%E3%81%AE%E5%8F%8D%E5%B0%84%E3%83%A2%E3%83%87%E3%83%AB

[5] 高橋章，R03-Ec5 プログラミング演習 II テキスト