

第2～4週：OpenGL ライブラリを使った3次元CGプログラミング

立命館大学 情報理工学部 メディア情報学科

## 1. 実験の目的と手順

本実験は、(1)OpenGLライブラリを用いたCGプログラミングの手法を学ぶ、(2)プログラミングを通して、CGの基礎技術を体験的に学ぶ、(3)インタラクティブな3次元CGアニメーションを作成する方法について学ぶ、ことを目的としている。

テキスト中に課題があるので、その指示にしたがってプログラムを作成・実行し、実行結果を確認すること。実行結果が確認できたら、TAのチェックを受けること。課題用ソースコードは、下記教材フォルダにあるので、最初にコピーして利用すること。

教材フォルダ：/kyozai/mediajikken\_ohshima\_CG\_sample

## 2. OpenGL

### 2.1 OpenGL とは

アプリケーションプログラムにおいて3次元CGの機能を実装するには、グラフィックスライブラリあるいはグラフィックAPI (Application Programming Interface)が使用される。グラフィックAPIとしてOpenGL (OpenGL ARB)とDirectX (Microsoft社)が広く普及している。前者はOS非依存のマルチプラットフォーム対応であり、科学技術における可視化や産業分野のCADシステムに、また後者はMS-Windows OS環境においてパフォーマンスを重視したゲームやマルチメディア用途を中心に利用されている。

本実験では、学習のし易さなどの観点からOpenGLを用いる。OpenGLの特長を以下に示す。

- UNIX, Linux, MS-Windows, MacOSなどの主要なプラットフォーム全てにライセンスフリーで対応しており、移植性の高いグラフィック・アプリケーションを開発することができる。
- 多くのビデオカードがハードウェア機能によってOpenGLコードを高速に処理することができる。
- 3次元CGの高度な描画機能を有する。
- 簡潔でわかりやすいコマンド体系を有する。
- 参考文献やサンプルコードの入手が容易である。
- 標準C/C++言語の初歩的な知識のみでプログラミングが可能である。

### 2.2 OpenGL の構成

以下の3種類のライブラリをプログラムにリンクして実行する。

(1) OpenGL ライブラリ：点・線・多角形の描画、色やパターンの制御、光源設定、幾何変換や投影変換、テクスチャマッピング、シェーディング処理などの基本的な描画機能を備えている。OpenGLや次に述べるGLU, GLUTのヘッダファイルは、標準includeパスの下のGLというフォルダに格納されている。

ヘッダファイル：#include<GL/gl.h>, リンクオプション：-lGL

(2) OpenGL ユーティリティライブラリ (以下GLUと略す)：OpenGLの上位ライブラリであり、OpenGLを組み合わせてアプリケーション開発に便利な機能を提供する。NURBS曲線と曲面、簡易な投影変換設定、高度なテクスチャ処理などのコマンドを有する。

ヘッダファイル：#include<GL/glu.h>, リンクオプション：-lGLU

(3) GLUT ライブラリ (以下GLUTと略す) 主としてウィンドウ管理、入力イベント処理、プリミティブ

(球体, 円柱などの基本立体), OpenGL や GLU を組み合わせた高度な描画を行う. OpenGL は CG の描画機能しか有していないため, ウィンドウ制御やイベント処理の部分は, OS とウィンドウシステムに依存したものとなる. しかしながら GLUT は, UNIX, Linux, MS-Windows, MacOS など各 OS のウィンドウシステムに対応しており, OpenGL と併せて利用することによって, プラットホームに依存しない移植性の高いアプリケーション開発が可能である.

ヘッダファイル: #include<GL/glut.h>, リンクオプション: -lglut

## 2.3 コンパイル方法

ソースファイル名を test.c, 実行ファイル名を test とした場合のコンパイル方法は以下の通りである.

```
% cc -o test test.c -O2 -Wall -L/usr/X11R6/lib -lm -lX11 (注: 改行しないで続けて書く)
-lGL -lGLU -lglut -lXext -lXmu -lXi
```

注1) make を利用することで, 上記コンパイルを簡便に行うことができる. 本テキストの末尾にある Appendix A を参考にすること. 課題1のみ, 上記直接 cc でコンパイルを実行してみよ. 再コンパイルの手間を大幅に軽減するので, 課題2以降, make を用いてコンパイルすることを強く推奨する.

注2) 自宅のコンピュータに cygwin をインストールしている場合, コンパイル方法は以下の通り.

```
% gcc -o test test.c -lglut32 -lglu32 -lopengl32 -lm
```

Cygwin の入手およびインストール方法は, 本テキストの末尾にある Appendix B を参考にすること.

## 2.4 プログラムの構成

プログラムの実行中に, 操作者がキーボードのキーを押したり, マウス操作によりシステムに対して行う入力のことをイベントと呼ぶ. GLUT を使ったプログラムでは, 対話処理を行うためにイベント駆動方式を採用している. 実行中の GLUT コード内部では, 次に示すように常にイベントを監視する無限ループに入っており, 発生したイベントに応じて実行状態を変化させる.

```
for( ;; ) {
    イベント発生
    switch(イベントの種類) {
    case イベント1:
        関数1を実行
        break;
    case イベント2:
        関数2を実行
        break;
        .....
    }
}
```

イベントに応じて呼び出される関数をコールバック関数と呼ぶ. 特に, 画面の描画を行うための関数をディスプレイコールバック関数と呼び, ユーザの操作によるイベントのほかに, ウィンドウ同士の重なりなど, 再描画する必要性が生じた場合に自動的に実行される.

## 3.3 次元 CG プログラミング

### 3.1 ウィンドウを開く

**課題1** sample1.c は, 500×500 ドットで, 背景が黒のウィンドウを画面左上にオープンするプログラムである (太字の行は主な新しい学習内容を示す). sample1.c をコンパイルして, 実行しなさい. また, ウィンド

ウサイズ、表示位置や背景色を任意に変更したプログラム kadail.c を作成せよ (※提出不要)。

```
1  /* sample1.c */
2  #include <GL/glut.h>          /* glut ヘッダファイルのインクルード */
3
4  void display(void)
5  {
6      glClear(GL_COLOR_BUFFER_BIT); /* ウィンドウの背景を指定された色で塗りつぶす */
7      glFlush();                  /* OpenGL コマンドを強制的に実行 (描画実行) */
8  }
9
10 void init(char *winname)
11 {
12     glutInitWindowPosition(0, 0); /* ウィンドウの左上の位置を(0,0)とする */
13     glutInitWindowSize(500, 500); /* ウィンドウのサイズを 500×500 ドットとする */
14     glutInitDisplayMode(GLUT_RGBA); /* 色の指定に RGBA モードを用いる */
15     glutCreateWindow(winname);     /* winname で指定された名前でウィンドウを開く */
16
17     glClearColor(0.0, 0.0, 0.0, 1.0); /* ウィンドウの背景色の指定 */
18                                         /* R(赤),G(緑),B(青),A(透明度)の順で指定 */
19 }
20
21 int main(int argc, char *argv[])
22 {
23     glutInit(&argc, argv);          /* glut の初期化 */
24     init(argv[0]);
25     glutDisplayFunc(display);      /* ディスプレイコールバック関数の指定 */
26     glutMainLoop();               /* イベント待ちの無限ループへ入る */
27     return 0;                     /* ループが終わったら 0 を返して終了 */
28 }
```

## 3.2 図形の描画

### 3.2.1 図形の定義

まず、点、線、多角形 (ポリゴン) を描画する方法について説明する。OpenGL では以下に示すように、頂点座標を基に基本図形を定義する。

```
glBegin(図形のタイプ);          /* 図形定義の開始 */
    glVertex3f(x0, y0, z0);      /* 頂点の設定 */
    glVertex3f(x1, y1, z1);      /* 頂点の設定 */
    .....
    glVertex3f(xn, yn, zn);      /* 頂点の設定 */
glEnd();                          /* 図形定義の終了 */
```

glBegin()の引数に指定できる図形のタイプには図1のようなものがある。OpenGL を処理するハードウェアの多くは、3 角形を塗り潰す処理しかできないため、GL\_POLYGON の場合は、多角形を 3 角形に分割してから処理される。従って、描画速度は GL\_TRIANGLE\_STRIP や GL\_TRIANGLE\_FAN の方が GL\_POLYGON よりも高速となる。GL\_QUADS も GL\_POLYGON より高速である。

```
GL_POINTS : 点を打つ
GL_LINES : 点を対にして、その間を直線で結ぶ
GL_LINE_STRIP : 折れ線を描く
GL_LINE_LOOP : 折れ線を描くが、始点と終点の間も結ばれる
GL_TRIANGLES/GL_QUADS : 3/4 点を組にして、三角形/四角形を描く
GL_TRIANGLE_STRIP/GL_QUAD_STRIP : 一辺を共有しながら帯状に三角形/四角形を描く
```

GL\_TRIANGLE\_FAN : 一边を共有しながら扇状に三角形を描く

GL\_POLYGON : 凸多角形を描く

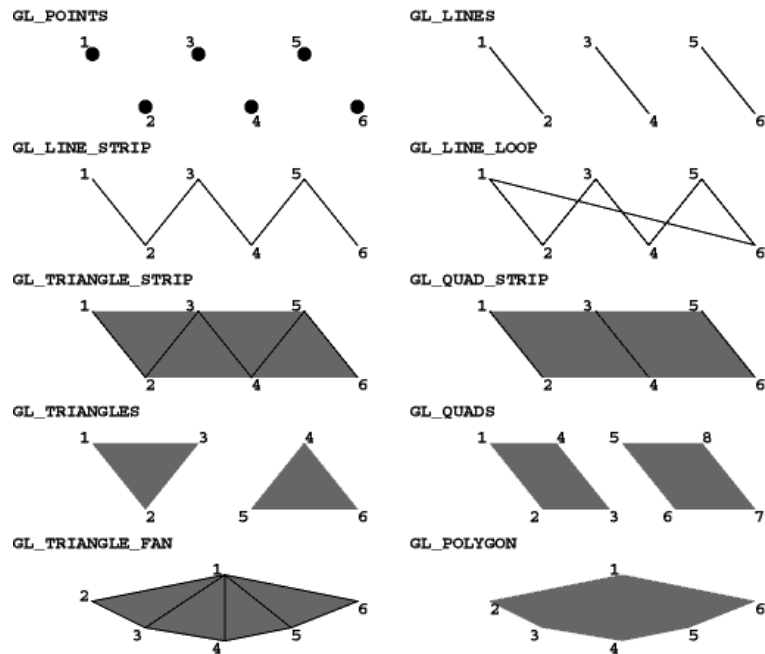


図 1 OpenGL の基本図形

### 3.2.2 ポリゴンの頂点の順番と表面の定義

OpenGL でポリゴンを描画する場合、頂点を定義する順番によって裏表を設定することができる。頂点の順に右ねじを回したときにねじが進む方向の面を表面となる。すなわち、表面側からポリゴンの頂点を見ると、反時計回りの順番で定義される (図 2)。

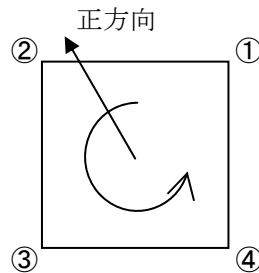
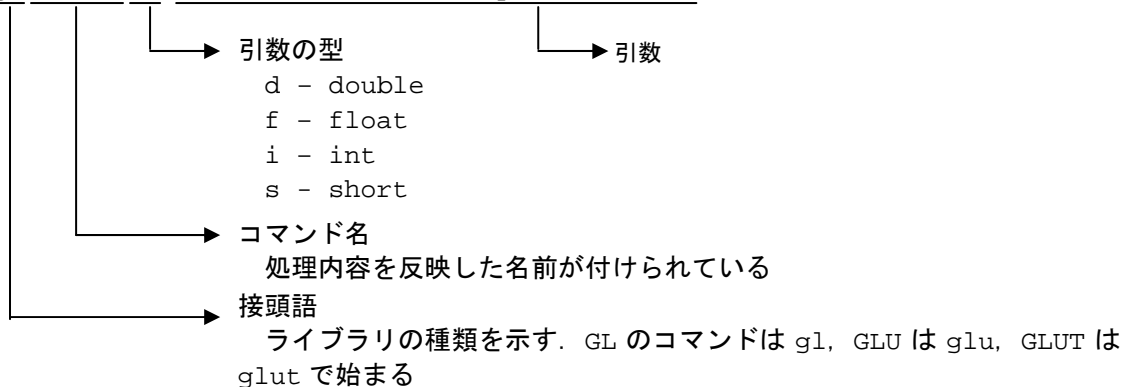


図 2 ポリゴンの頂点の順番と表方向の定義

### 3.2.3 OpenGL の関数名

OpenGL 関連のライブラリ関数は、接頭語、コマンド名、接尾語、引数からなる。glVertex3f(x, y, z) を例に示す。

`glVertex3f(GLfloat x, GLfloat y, GLfloat z)`



### 3.2.4 図形描画のプログラム

**課題 2** sample2.c は、sample1.c に、四角形を描くコードを追加したものである。sample2.c を完成させて実行しなさい。sample2.c を基にして、中心(0.0, 0.0)、半径 0.5 の円を描くプログラム kadai2.c を作成しなさい。sample2.c 中、『sample1.c と同じ』は、sample1.c と同じ内容で置き換えるべき部分を示す。

- 8~13 行目 (glBegin~glEnd) を変更する。
- 一般に CG の世界では、円や球は多角形（三角形を組み合わせたもの）として描かれる。n を分割数、r を半径、PI を円周率とすると、円周上の i 番目の点は次式で表される。  
x = r \* cos(2.0 \* PI \* (float)i / (float)n);  
y = r \* sin(2.0 \* PI \* (float)i / (float)n);  
z = 0.0;
- glBegin(GL\_TRIANGLE\_FAN) もしくは、glBegin(GL\_POLYGON) を用いる。
- 円周率の定義として、#define PAI 3.14 をプログラムの先頭に入れる。
- 三角関数を使うので、math.h をインクルードする。
- n=10 とした場合と、n=100 とした場合を比較せよ。※提出ソースコードはどちらか一方でよい。

```
1  /* sample2.c */
2  #include <GL/glut.h>
3
4  void display(void)
5  {
6      glClear(GL_COLOR_BUFFER_BIT);
7      glColor3f(1.0, 1.0, 1.0);      /* 描画する図形の色を白に指定 */
8      glBegin(GL_QUADS);            /* 四角形を描画 */
9      glVertex3f(-0.5, -0.5, 0.0);
10     glVertex3f(0.5, -0.5, 0.0);
11     glVertex3f(0.5, 0.5, 0.0);
12     glVertex3f(-0.5, 0.5, 0.0);
13     glEnd();
14     glFlush();                    /* OpenGL コマンドを強制的に実行 (描画実行) */
15 }
16
17 void init(char *winname)
18 {
19     /* ■sample1.c と同じ■ */
20 }
21
22 int main(int argc, char *argv[])
23 {
24     /* ■sample1.c と同じ■ */
25 }
```

## 3.3 座標系と投影法

### 3.3.1 ワールド座標系とスクリーン座標系

コンピュータ内部の 3 次元座標系を **ワールド座標系**、ディスプレイ上の 2 次元平面の座標系を **スクリーン座標系** と呼ぶ (図 3)。

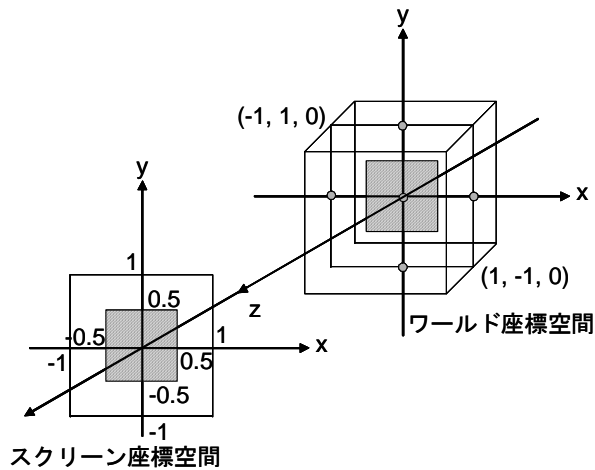


図3 ワールド座標系とスクリーン座標系

### 3.3.2 投影法

ワールド座標系の3次元物体をスクリーン座標系に映すことを**投影**と呼ぶ。投影法は多数あるがCGでは図4に示す**平行投影**と**透視投影**が主に用いられる。

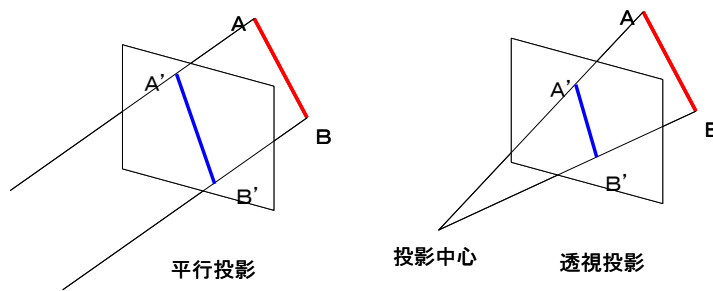


図4 平行投影と透視投影

平行投影では、投影線が平行であるために、投影面を垂直に取ると物体の寸法を正確に表すことができる。しかし、平行投影では遠くの物体も近くの物体も同じ寸法に表示されてしまう。

透視投影では、投影点が視点（投影中心）に収束するように投影される。我々の視覚系と同様に遠近感が得られるため、現実に近い画像生成が可能である。

### 3.3.3 投影法とビューボリューム

一般にCGでは、変換の対象となる3次元空間を限定することによって、計算の効率化を図る。まず、投影面にウィンドウを設定し、視点から見える視野の範囲を限定する。このウィンドウ外の物体の投影を除く作業を**クリッピング**という。このような作業を経て、投影によりウィンドウに表示される領域を**ビューボリューム**と呼ぶ。平行投影と透視投影におけるビューボリュームを図5、図6に示す。OpenGLでは、以下の関数を用いて投影の方法と座標系を指定する。ビューボリュームの外に描かれた物体は表示されないので注意すること。

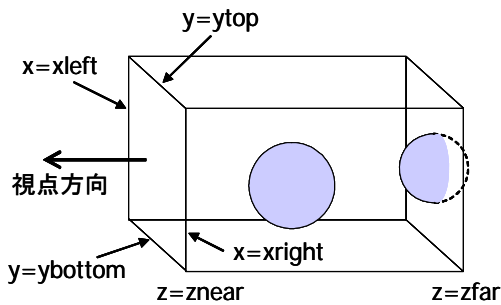


図5 平行投影のビューボリューム

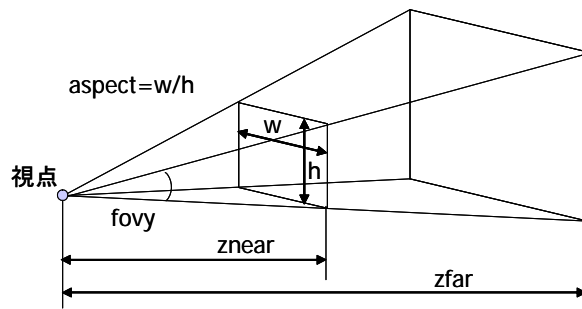


図6 透視投影のビューボリューム

(1)平行投影

```
glOrtho(xleft, xright, ybottom, ytop, znear, zfar)
```

引数は、ビューボリュームの左右、下上、前後の各面を表す x, y, z 座標のペアである。

(2)透視投影

```
gluPerspective(fovy, aspect, znear, zfar)
```

fovy は、ビューボリュームの上下の開き角 (°), aspect は断面の幅/高さの比, znear と zfar は、視点から頂面までと底面までの距離である。

(3)視点位置と視線方向

```
gluLookAt(ex, ey, ez, cx, cy, cz, ux, uy, uz)
```

gluLookAt を定義しない場合は、原点を視点として z 軸の負の方向を、y 軸の正の方向を真上として見る設定となっている。gluLookAt を定義することで、最初の3つの引数 ex, ey, ez を視点の位置、次の3つの引数 cx, cy, cz を目標の位置、最後の3つの引数 ux, uy, uz を、ウィンドウに表示される画像の「上」の方向を示すベクトルとして設定することができる。

OpenGL では、座標変換を伴う全ての処理が1つの変換行列を用いて行われる。初期設定時には、この変換行列を初期化するために、glLoadIdentity() という関数を呼び出し、変換行列に単位行列を代入する。平行投影 glOrtho(), 透視投影 gluPerspective(), 視点位置と視線方向を設定する gluLookAt() などを実行すると、この変換行列にそれぞれの座標変換を行う行列が乗算される。

3.3.4 平行投影と透視投影のプログラム

**課題3** sample3.c は、sample2.c を基に、ビューボリュームが  $-2.0 < x < 2.0$ ,  $-2.0 < y < 2.0$ ,  $-1.0 < z < 1.0$  の平行投影となるように設定したものである。sample3.c を完成させて実行しなさい。

また、19行目を以下のような透視投影となるように変更したプログラム kadai3.c を作成せよ。

- ビューボリューム：上下の開き角 fovy=30.0, aspect=1.0, znear=1.0, zfar=10
- 視点の位置:y軸を上にして((ux,uy,uz)=(0.0, 1.0, 0.0)), (3.0, 4.0, 5.0)から(0.0, 0.0, 0.0)の方向を見る

```
1 /* sample3.c */
2 #include <GL/glut.h>
3
4 void display(void)
5 {
6     /* ■sample2.c と同じ■ */
7 }
8
9 void init(char *winname)
10 {
```

```

11  glutInitWindowPosition(0, 0);
12  glutInitWindowSize(500, 500);
13  glutInitDisplayMode(GLUT_RGBA);
14  glutCreateWindow(winname);
15
16  glClearColor(0.2, 0.4, 0.4, 1.0);
17
18  glLoadIdentity();          /* 変換行列の初期化 */
19  glOrtho(-2.0, 2.0, -2.0, 2.0, -1.0, 1.0); /* 平行投影の設定 */
20  }
21
22  int main(int argc, char *argv[])
23  {
24  /* ■sample2.c と同じ■ */
25  }

```

### 3.4 幾何変換

#### 3.4.1 幾何変換

幾何変換とは、ワールド座標空間において、物体の位置や姿勢を変えたり、拡大・縮小の変形を加える操作を指している。OpenGL（および多くのCGライブラリ）では、この幾何変換も、3.3.3で投影法を設定したときと同じ「変換行列」を利用して、座標系全体の変換として実行される。

##### (1) 平行移動

```
glTranslatef(tx, ty, tz)
```

変換行列に移動の行列を乗じる。tx, ty, tz は各軸方向への移動量。

##### (2) 回転

```
glRotatef(theta, x, y, z)
```

変換行列に回転の行列を乗じる。theta は回転角度 (°), x, y, z は回転軸のベクトル。

##### (3) 拡大・縮小

```
glScalef(sx, sy, sz)
```

変換行列に拡大・縮小の行列を乗じる。sx, sy, sz は各軸方向への拡大量。マイナスの値で座標系が反転する。

これらの関数により、座標系全体が変換され、その後に設定される図形は、変換後の座標軸で定義されることになる。また、これらの関数を組み合わせることで、任意の位置に物体を動かすことができる。プログラムを書くときには、最後に呼び出された変換が頂点に対して最初に適用されるので、例えば、

```

glTranslatef(...);
glRotatef(...);
glBegin(...);

```

とすると回転を行った後で平行移動が実行され（図7左）、

```

glRotatef(...);
glTranslatef(...);
glBegin(...);

```

とすると平行移動を行った後で回転が実行される（図7右）。



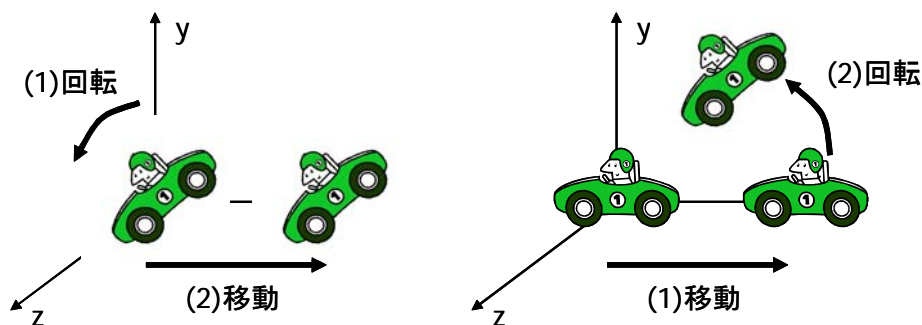


図7 幾何変換

座標変換のプロセスは、以下に示す4つのプロセスから成る。

- (1)図形の空間中での位置を決める「モデリング変換」
- (2)その空間を視点から見た空間に直す「ビューイング（視野）変換」
- (3)その空間をコンピュータ内の空間にあるスクリーンに投影する「投影変換」
- (4)スクリーン上の図形をディスプレイ上の表示領域に切り出す「ビューポート変換」

3.3.3でも触れたが、OpenGLでは1つの「変換行列」に、(1)~(4)の投影を行う行列式を掛け合わせて座標変換を行う。変換行列は、描画の度に設定し直されるが、プログラムの実行中に投影変換を変更することはあまりなく、図形だけを動かす場合はモデリング変換の行列だけを変更すればよい。そこで、OpenGLでは、「モデリング変換-ビューイング変換」の変換行列（モデルビュー変換行列）と、「投影変換」の変換行列（投影変換行列）を独立して取り扱う手段が提供されている。モデルビュー変換行列を設定する場合は

```
glMatrixMode(GL_MODELVIEW),
```

投影変換行列を設定する場合は

```
glMatrixMode(GL_PROJECTION)
```

を実行する。

### 3.4.2 幾何変換のプログラム

**課題4** sample4.c は、kadai3.cを基にx方向に-1.0平行移動するように変更したものである。kadai3.cを利用する未完箇所があるので注意すること。このままコンパイルするとエラーとなる。①sample4.cを完成させてkadai4-1.cとし、kadai3と比較しなさい。②また、上記変換の代わりに、y軸周りに60°回転させてからz方向に-3.0平行移動するように変更したkadai4-2.cを作成しなさい。変換を記述する順序に注意すること。

```

1  /* sample4.c */
2  #include <GL/glut.h>
3
4  float vertex[][3] = { /* 頂点を配列で宣言する方法 */
5    {-0.5,-0.5, 0.0},
6    { 0.5,-0.5, 0.0},
7    { 0.5, 0.5, 0.0},
8    {-0.5, 0.5, 0.0},
9  };
10
11 void display(void)
12 {
13     int i;
14

```

```

15  glClear(GL_COLOR_BUFFER_BIT);
16  glTranslatef(-1.0, 0.0, 0.0); /* 座標系を x 方向に-1.0 平行移動する */
17  glColor3f(1.0, 1.0, 1.0);
18  glBegin(GL_QUADS);
19  for( i=0; i<4; i++)
20      glVertex3f(vertex[i][0],vertex[i][1],vertex[i][2]);
21  glEnd();
22  glFlush();
23  }
24
25  void init(char *winname)
26  {
27      glutInitWindowPosition(0, 0);
28      glutInitWindowSize(500, 500);
29      glutInitDisplayMode(GLUT_RGBA);
30      glutCreateWindow(winname);
31
32      glClearColor(0.2, 0.4, 0.4, 1.0);
33
34      glMatrixMode(GL_PROJECTION); /* 投影法に関する座標変換を開始 */
35      glLoadIdentity();
36      gluPerspective( /*kadai3.c に適用した設定を利用*/ );
37      glMatrixMode(GL_MODELVIEW); /* 視点の移動やモデルの移動など
38                                     投影法以外の座標変換を開始 */
39      gluLookAt( /*kadai3.c に適用した設定を利用*/ );
40  }
41
42  int main(int argc, char *argv[])
43  {
44      /*kadai3.c と同じ*/
45  }

```

## 3.5 アニメーション

### 3.5.1 アニメーションの設定

アニメーションを行うには、頻繁に画面の書き換えを行う必要がある。しかし `glutMainLoop()` は無限ループであり、`glutDisplayFunc()` で指定された関数は、ウィンドウを再描画するイベントが発生したときにしか呼び出されない。したがってアニメーションを実現するには、このウィンドウの再描画イベントを連続的に発生させる必要がある。プログラム中でウィンドウの再描画イベントを発生させるには、**`glutPostRedisplay()`** 関数を用いる。これをプログラムが「暇なとき」(アイドルング時)に繰り返し呼び出すことで、アニメーションが実現できる。プログラムが暇になったときに実行する関数は、**`glutIdleFunc()`** で以下のように指定する。

```

static int param = 0;
void idle ( void ) {
    param += d; /* アニメーションで物体を動かす際のパラメータ param を d 増加させる */
    glutPostRedisplay(); /* ディスプレイコールバック関数 (サンプルプログラムでは、
                           display 関数) を呼び出す */
}
int main () {
    ...
    glutIdleFunc ( idle ); /* アイドルコールバック関数の指定 */
}

```

但し、ウィンドウの再描画イベントを連続的に発生させるだけでは、毎回画面を全部書き換えるため、表示がちらついてしまう。これを防ぐために、**ダブルバッファリング**という方法を用いる。これは画面を2つに分け、一方を表示している間に（見えないところで）もう一方に図形を描き、それが完了したらこの2つの画面を入れ換えるという方法である（図8）。

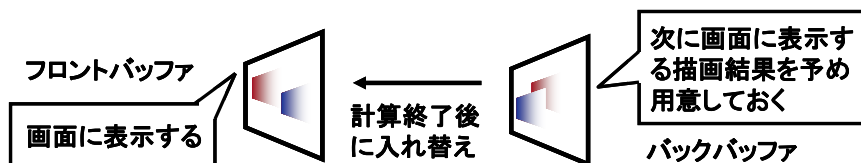


図8 ダブルバッファ

ダブルバッファの設定は、以下のように行う。

```
glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGBA )
```

OpenGL のディスプレイモードを設定する。RGBA モードで色を表現することに加えて、ダブルバッファとして分割して利用することを宣言する。

```
glutSwapBuffers()
```

バッファを入れ替え、バックバッファを空にする。この関数は、描画の最後で `glFlush()` の代わりに記述する。この関数の中で `glFlush()` が呼ばれている。

最後に、ここでは `glutPostRedisplay` 関数を実行して、`display` 関数を何度も呼び出すことで、アニメーション（移動した物体の再描画）を行っている。しかし、`display` 関数内で `glTranslatef()` や `glRotatef()` を実行している場合、`display` 関数を呼ぶたびに、変換行列にこれらのモデル変換行列が掛け合わさるという問題が発生する。例えば、`sample4.c` のように `glTranslatef(-1.0, 0.0, 0.0)` という変換を行っている場合、`display` 関数が実行されるたびに、オブジェクトが x 軸方向に -1.0 ずつ移動してしまう。そこで、OpenGL では、`glTranslatef()` や `glRoatatef()` を使う前に、最初の時点での座標系（変換行列の内容）を一旦保存しておき、後でもとの座標系に戻すことで、この問題を解決している。この処理には `glPushMatrix()` と `glPopMatrix()` という関数を使う。使い方をサンプルプログラム `sample5.c` に示す。`glPushMatrix` と `glPopMatrix` とは必ずペアで使用すること。

### 3.5.2 アニメーションのプログラミング

**課題 5** `sample5.c` は、四角形のポリゴンが y 軸周りに回転するアニメーションを描画するプログラムである。このプログラムを実行しなさい。①次に、物体が z 軸周りに回転するようにプログラムを変更し `kadai5-1.c` を作成しなさい。②また、`kadai5-1.c` の 10, 19 行目をコメントアウトし、25 行目で 0.001 度ずつ増加するように変更して `kadai5-2.c` を作成し、どのようになるか実行して確認しなさい。

```
1  /* sample5.c */
2  #include <GL/glut.h>
3  #include <math.h>                /* math.h をインクルード */
4
5  float param = 0.0;
6
7  void display(void)
8  {
9      glClear(GL_COLOR_BUFFER_BIT);
10     glPushMatrix();                /* この時点での座標系（変換行列）を保存 */
11     glRotatef(param, 0.0, 1.0, 0.0); /* y 軸を中心に param 度回転 */
12     glBegin(GL_QUADS);
13     glColor3f(1.0, 1.0, 1.0);
14     glVertex3f(-0.5, -0.5, 0.0);
```

```

15  glVertex3f(0.5, -0.5, 0.0);
16  glVertex3f(0.5, 0.5, 0.0);
17  glVertex3f(-0.5, 0.5, 0.0);
18  glEnd();
19  glPopMatrix();          /* 9行目の変換系に戻る */
20  glutSwapBuffers();
21  }
22
23 void idle(void)
24 {
25     param = fmod(param+0.1, 360.0); /* 回転角を0~360度まで, 0.1度ずつ増加 */
26     glutPostRedisplay();          /* ディスプレイコールバック関数(display)の実行 */
27 }
28
29 void init(char *winname)
30 {
31     glutInitWindowPosition(0, 0);
32     glutInitWindowSize(500, 500);
33     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA); /* ダブルバッファの宣言 */
34     glutCreateWindow(winname);
35
36     /* ■kadai4-1.cと同じ■ */
37 }
38
39 int main(int argc, char *argv[])
40 {
41     glutInit(&argc, argv);
42     init(argv[0]);
43     glutDisplayFunc(display);
44     glutIdleFunc(idle);          /* アイドルコールバック関数の指定 */
45     glutMainLoop();
46     return 0;
47 }

```

### 3.6 キーボード、マウスによる入力

2.4節で説明した通り、OpenGLはイベント駆動型のプログラムであり、キーボードからの入力や、マウスによる入力など、入力されたイベントに応じてあらかじめ登録されたコールバック関数が実行される。

#### 3.6.1 キーボード入力のプログラム

キーボードからの入力を利用するためには、まず main 関数内で、**glutKeyboardFunc**(コールバック関数名)を実行し、キーボード機能呼び出すコールバック関数を指定する必要がある。こうすることで、キーボードからの入力があった場合に、押されたキーの種類が取得されコールバック関数に渡される。

Sample6-1.cにサンプルプログラムを示す。

**課題 6-1** sample6-1.c は、キーボードで”q”と打つと、プログラムが終了するように sample5.c のプログラムを変更したものである。このプログラムを完成して実行しなさい。次に、プログラムを実行すると、(1)最初は物体は静止していて、キーボードで”r”と打つたびに、物体が y 軸中心に 30 度ずつ回転し、(2)”c”と打つたびに色が白→赤→緑→青→白に変わる、ようにプログラムを変更して kadai6-1.c を作成しなさい。

- 物体の色を変えるには、display 関数の glColor3f(1.0, 1.0, 1.0)を変更すればよい。
- 赤は(1.0, 0.0, 0.0), 緑は(0.0, 1.0, 0.0), 青は(0.0, 0.0, 1.0)である。

```

1  /* sample6-1.c */
2  #include <GL/glut.h>
3  #include <math.h>
4  #include <stdlib.h>
5

```

```

6 float param = 0.0;
7
8 void display(void)
9 {
10  /*■sample5.cと同じ■*/
11 }
12
13 void idle(void)
14 {
15  /*■sample5.cと同じ■*/
16 }
17
18 void keyboard(unsigned char key, int x, int y)
19 /*引数 key にはタイプされたキーの ASCII コードが,
20  x と y にはキーがタイプされたときのマウスの位置が渡される */
21 {
22  switch (key) {
23  case 'q':          /* q が入力されたら, プログラムを終了する */
24    exit(0);
25  default:
26    break;
27  }
28 }
29
30 void init(char *winname)
31 {
32  /*■sample5.cと同じ■*/
33 }
34
35 int main(int argc, char *argv[])
36 {
37  glutInit(&argc, argv);
38  init(argv[0]);
39  glutDisplayFunc(display);
40  glutKeyboardFunc(keyboard); /* キーボード入力のコールバック関数 keyboard の指定 */
41  glutIdleFunc(idle);
42  glutMainLoop();
43  return 0;
44 }

```

### 3.6.2 マウス入力 of プログラム

マウス操作に関するイベント処理は、ボタンのオン・オフ（クリック）とマウスの移動（ドラッグ）の 2 種類の設定が可能であるが、ここでは、クリックについてのみ説明する。マウスボタンのクリックによる入力を利用するためには、main 関数内で `glutMouseFunc`（コールバック関数名）を実行し、マウス機能を呼び出すコールバック関数を指定する必要がある。

sample6-2.c にサンプルプログラムを示す。

**課題 6-2** sample6-2.c は、ウィンドウ内をマウスで左クリックすると、物体が y 軸中心に一回転するように `kadai6-1.c` を変更したものである。このプログラムを完成させ、実行しなさい。次に、ウィンドウ内をマウスで左クリックすると、回転が始まり、右クリックすると、回転が止まるようにプログラムを変更した `kadai6-2.c` を作成しなさい。

```

1  /* sample6-2.c */
2  #include <GL/glut.h>
3  #include <math.h>
4  #include <stdlib.h>
5

```

```

6 float param = 0.0;
7 int flag = 0.0;
8
9 void display(void)
10 {
11     /*■kadai6-1.cと同じ■*/
12 }
13
14 void idle(void)
15 {
16     if(flag == 1){
17         param = fmod(param+1.0, 360.0); /* 回転角を 0~360 度まで, 1 度ずつ増加 */
18         if(param == 0.0) flag = 0; /* 一周したら, flag を 0 にして回転を止める */
19     }
20     glutPostRedisplay();
21 }
22
23 void mouse(int button, int state, int x, int y)
24 {
25     if(state == GLUT_DOWN){ /* マウスがクリックされて */
26         switch(button){
27             case GLUT_LEFT_BUTTON: /*クリックされたのが左ボタンだったら */
28                 flag = 1; /* 左ボタンがクリックされたときに行う処理 */
29                 break;
30             case GLUT_RIGHT_BUTTON: /*クリックされたのが右ボタンだったら */
31                 /* 右ボタンがクリックされたときに行う処理 */
32                 break;
33         }
34     }
35 }
36
37 void keyboard(unsigned char key, int x, int y)
38 {
39     /*■kadai6-1.cと同じ■*/
40 }
41
42 void init(char *winname)
43 {
44     /*■kadai6-1.cと同じ■*/
45 }
46
47 int main(int argc, char *argv[])
48 {
49     glutInit(&argc, argv);
50     init(argv[0]);
51     glutDisplayFunc(display);
52     glutKeyboardFunc(keyboard);
53     glutMouseFunc(mouse); /* マウス入力のコールバック関数 mouse の指定 */
54     glutIdleFunc(idle);
55     glutMainLoop();
56     return 0;
57 }

```

### 3.7 陰面処理とシェーディング

#### 3.7.1 隠面処理

複数の面（ポリゴン）から構成される立体形状を描画する場合に、あるポリゴンが他のポリゴンに隠されて見えなくなる状況が発生する。このような場合、全てのポリゴンを無条件で描画してしまうと、立体形状が正しく表現できない。このように、隠された面を描画しない処理を「**隠面処理**」という。OpenGL プログ

ラム内部では、隠面処理は、各画素ごとに、視点から物体までの投影変換後の奥行き値（z 値※）を比較して、最も近い可視面だけを描画する。これは、以下の手順で行われる。（※z 座標値ではないことに注意）

- (1) Zバッファ（z 値を格納する記憶領域）を最大値でクリア
- (2) 描画物体の各画素に対応する z 値を比較
- (3) z 値が小さい画素のみ描画を行い、対応する z 値を書き換える
- (4) 物体を描画するたびに(2)(3)を繰り返す。

OpenGL では、以下のようにして、隠面処理の設定を行う。

- (1) z バッファ利用の宣言

```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
```

z バッファの記憶領域を確保するために、フレームバッファを設定する命令 `glutInitDisplayMode` の引数に `GLUT_DEPTH` をビット論理和で加える。縦棒（|）はビット論理和を示す。

- (2) z バッファを初期化

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Z バッファを初期化する。 `glClear` は、ウィンドウの背景を塗りつぶす命令であるが、引数に `GL_DEPTH_BUFFER_BIT` を加えることにより、z バッファの記憶領域が最大値で初期化される。

- (3) 隠面処理の有効範囲の指定

```
glEnable( GL_DEPTH_TEST);  
.....  
glDisable( GL_DEPTH_TEST);
```

`glEnable` と `glDisable` で囲まれた範囲の描画計算に Z バッファ法による隠面処理が適用される。

### 3.7.2 シェーディングの原理

**シェーディング**とは、陰影付け処理であり、物体表面で反射して視点に到達する光の強度を計算することにより、物体の色の濃淡と質感を再現する。物体表面で反射する光の成分は、おおむね 3 つの成分であらわすことができる。シェーディングでは、視点に到達するこれらの成分の強度を別々に計算し、重ね合わせることで物体の一点の輝度を求める。

- (1) 拡散反射成分

拡散反射成分とは、物体表面の非常に薄い層で起こる光の反射である。拡散反射光は、すべての方向に一樣に散乱されるため、入射光の方向と強度および表面の向きのみで計算され、視点の方向には依存しない。拡散反射成分は、物体の材質が持つ色を反映しており、面の傾きにより生成される濃淡の違いが陰影として知覚される。

- (2) 鏡面反射成分

物体表面に細かな凹凸がなく滑らかであると、入射光は特定の方向に強く反射され、ハイライトと呼ばれる輝点を生ずる。このような反射を鏡面反射と呼び、物体表面に光沢感を与える。鏡面反射の方向は、面の法線方向に対して入射光の方向と対称となる角度（正反射方向）付近に分布している。したがって、ハイライトの強度は視点位置によって異なり、視点方向が正反射方向と一致したときに最大となる。

- (3) 環境光反射成分

われわれの周囲の空間は、光源からの直接光の他に、空気による散乱光や他の物体表面からの副次的な反射によりすべての方向から弱く照らされている。このため、物体の影の部分もある程度の明るさを持つ。CG

では、このような間接光の成分を**環境光**と呼び、それによる反射成分を考慮する。環境光成分を考慮しない画像は、宇宙空間で撮影した写真のように影の部分が真っ黒になり、不自然な印象を与える。

### 3.7.3 シェーディングの設定

シェーディング計算の設定方法は、以下のような手順で行う。

- (1)光源を有効にする
- (2)ポリゴンの頂点に法線ベクトル（光が反射する方向）を設定
- (3)シェーディングを行う範囲の指定

次に、上記(1)~(3)の設定方法について説明する。

#### (1)光源を有効にする

```
glEnable( GL_LIGHT0 );
```

初期設定において、利用する光源を登録する。GL\_LIGHT0 はデフォルトの光源で、z 軸のマイナス方向を照らす白色の平行光源である。GL\_LIGHT0~GL\_LIGHT6 の7個の光源を同時に利用することが可能である。光源には、平行光源、点光源、スポット光源、環境光などいろいろな種類があるが、本実験ではそれらの設定方法は省略する。

#### (2)ポリゴンの頂点に法線ベクトルを設定

```
glNormal3f(x, y, z);
```

x, y, z は法線ベクトルの成分。長さが1の単位ベクトル ( $x^2+y^2+z^2=1$ ) にする必要がある。ポリゴンの頂点の宣言、すなわち glVertex3f の前に glColor3f() の代わりに記述する。

#### (3)シェーディングを行う範囲の指定

```
glEnable(GL_LIGHTING);  
...  
glDisable(GL_LIGHTING);
```

2つの関数の間で描画される図形には、シェーディング処理が施される。

以上の手順のみの場合、複数のポリゴンの連続で構成される物体のシェーディングを行うと、同一のポリゴン内部は、シェーディング計算の結果ほぼ同じ色に塗られるため、ポリゴンで構成されたモデルであること（ポリゴンの境界線）がはっきりとわかってしまう。このようなシェーディングを、「**フラットシェーディング**」と呼ぶ。そこで、陰影変化を平滑化して、見た目には曲面のようにみせる「**スムーズシェーディング**」が用いられる。フラットシェーディングとスムーズシェーディングの設定は、以下のように行う。

```
glShadeModel(GL_FLAT);
```

最初の頂点の色をポリゴン全体に適用する。

```
glShadeModel(GL_SMOOTH);
```

ポリゴン内部の各点について、各頂点からの距離の比により頂点の色を混合して適用する。

### 3.7.4 表面属性の設定

陰影付けが有効になっているときは、glColor3f() による色指定は無視され、物体表面の持つ光の反射特性を設定することで色を指定することができる。この反射特性を表面属性と呼ぶ。OpenGL では、以下のコマンドを用いて、表面属性の設定を行う。

```
glMaterialf*(材質が設定される面, 材質の特性, 設定値);
```



このコマンドは、物体の描画前に設定する。引き数の指定方法は次の通り。

(1)材質が設定されている面

GL\_FRONT : ポリゴンの表面のみに設定

GL\_BACK : ポリゴンの裏面のみに設定

GL\_FRONT\_AND\_BACK : ポリゴンの両面に設定

ポリゴンの裏表の指定は、p.4 の 3.2.2 図 2 で述べたとおり。

(2)材質の特性

GL\_DIFFUSE : 拡散反射成分の設定

設定値は R, G, B, A 強度を 0.0~1.0 で指定

GL\_SPECULAR : 鏡面反射成分の設定

設定値は R, G, B, A 強度を 0.0~1.0 で指定

GL\_AMBIENT : 環境光反射成分の設定

設定値は R, G, B, A 強度を 0.0~1.0 で指定

GL\_EMISSIVE : 発光物体の設定

設定値は R, G, B, A 強度を 0.0~1.0 で指定

GL\_SHININESS : 鏡面反射光の鋭さの設定

鏡面反射光の鋭さを 0.0~128.0 までの数値で設定

(3)設定値

- ・材質特性に、GL\_DIFFUSE, GL\_SPECULAR, GL\_AMBIENT, GL\_EMISSIVE を設定する場合、RGBA 強度の値をあらかじめ定義しておき、glMaterialfv を用いる。

```
float diffuse[ ] = {1.0, 1.0, 1.0, 1.0};
float specular[ ] = {1.0, 1.0, 1.0, 1.0};
float ambient[ ] = {0.1, 0.1, 0.1, 1.0};
.....
glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, specular);
glMaterialfv(GL_FRONT, GL_AMBIENT, ambient);
```

配列の第 1~4 パラメータは、R, G, B, A (Alpha: 透明度) の各値 (0.0~1.0) である。

- ・材質特性に、GL\_SHININESS を設定する場合、glMaterialf を用いる。

```
glMaterialf(GL_FRONT, GL_SHININESS, 128.0);
```

### 3.7.5 陰面処理とシェーディングのプログラミング

**課題 7** sample7.c は、kadai6-2.c に光源、物体の表面属性の設定を行ったものである。sample7.c を完成させて実行しなさい。次に下記のヒントを参考にして、(1)陰面処理を行って、描画物体を球に変更し、(2)“s”キーを押すたびに、シェーディングの種類がフラットシェーディングとスムーズシェーディングで切り替わり、(3)“c”キーを押すたびに、球の色が白→赤→緑→青→白に変わるようにして kadai7.c を作成しなさい。

- ・ 忘れずに陰面処理を行うこと : glutInitDisplayMode に GLUT\_DEPTH を論理輪で追加し、GL\_DEPTH\_TEST を glEnable する。また、glClear に GL\_DEPTH\_BUFFER\_BIT を論理輪で追加する (詳細は、3.7.1 の説明を参照せよ)。
- ・ GLUT には基本物体として球を描画する関数が用意されている。その他の基本物体は、3.8 節で述べる。

```
glutSolidSphere(double radius, int slices, int stacks);
```

(radius は球の半径, slices と stacks は緯度と経度方向の分割数)

法線の設定は, glutSolidSphere 関数の中で行われている. プログラム中では, 27~33 行目までのポリゴンを描画する部分を変更し, 半径 1.0, 経度・緯度の分割数 10 の球を設定する.

- keyboard 関数の中で glShadeModel(GL\_FLAT); と glShadeModel(GL\_SMOOTH); を切り替える.
- keyboard 関数の中で diffuse[] の中身を変更する. 色の指定は, 課題 6-1 のヒントを参考にすること.

```
1  /* sample7.c */
2  #include <GL/glut.h>
3  #include <math.h>
4  #include <stdlib.h>
5
6  float param = 0.0;
7  int flag = 0.0
8
9  float diffuse[] = {0.8, 0.8, 0.8, 1.0}; /* 拡散反射成分の反射強度*/
10 float specular[] = {1.0, 1.0, 1.0, 1.0}; /* 鏡面反射成分の反射強度 */
11 float ambient[] = {0.2, 0.2, 0.2, 1.0}; /* 環境光成分の反射強度*/
12 float shininess = 128.0;
13
14 void display(void)
15 {
16     glClear(GL_COLOR_BUFFER_BIT);
17
18     glPushMatrix();
19     glRotatef(param, 0.0, 1.0, 0.0);
20
21     glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse); /* 拡散反射の設定 */
22     glMaterialfv(GL_FRONT, GL_SPECULAR, specular); /* 鏡面反射の設定 */
23     glMaterialfv(GL_FRONT, GL_AMBIENT, ambient); /* 環境光の設定 */
24     glMaterialf(GL_FRONT, GL_SHININESS, shininess); /* 鏡面反射の鋭さの設定 */
25
26     glEnable(GL_LIGHTING); /* シェーディング計算の開始 */
27     glBegin(GL_QUADS);
28     glNormal3f(0.0, 0.0, 1.0); /* 法線ベクトルの設定 */
29     glVertex3f(-0.5, -0.5, 0.);
30     glVertex3f(0.5, -0.5, 0.);
31     glVertex3f(0.5, 0.5, 0.);
32     glVertex3f(-0.5, 0.5, 0.);
33     glEnd();
34     glDisable(GL_LIGHTING);
35
36     glPopMatrix();
37     glutSwapBuffers();
38 }
39
40 void idle(void)
41 {
42     /*■kadai6-2.c と同じ■*/
43 }
44
45 void mouse(int button, int state, int x, int y)
46 {
47     /*■kadai6-2.c と同じ■*/
48 }
49
50 void keyboard(unsigned char key, int x, int y)
51 {
52     /*■kadai6-2.c と同じ■*/
```

```

53 }
54
55 void init(char *winname)
56 {
57     /*■kadai6-2.cと同じ■*/
58
59     glShadeModel(GL_FLAT);    /* シェーディングの種類をフラットシェーディングに設定 */
60     glEnable(GL_LIGHT0);    /* LIGHT0 の光源を有効にする */
61 }
62
63 int main(int argc, char *argv[])
64 {
65     /*■kadai6-2.cと同じ■*/
66 }

```

### 3.8 基本形状の組み合わせによる形状表現

#### 3.8.1 プリミティブの描画

任意の形状を表現しようとする場合、OpenGL 自体にはモデリングという概念はない。OpenGL は描画のための機能を提供するだけである。したがって、OpenGL で任意の形状を表現することとは、形状モデルデータを作成することではなく、形状を描画する命令系列をプログラムとして記述することである。これまで頂点を一つ一つ設定し、ポリゴンを描画する方法を学んだが、これでは多数のポリゴンから構成される複雑な形状を表現するのに莫大な労力が必要となる。そのため、GLUT ライブラリでは、基本的な図形の描画を簡単に行えるように球や円錐などの基本形状（プリミティブ）の描画を行う関数が提供されている。課題 7 で利用した球を描画する関数もその 1 つである。ここでは、まずこのプリミティブ関数を紹介する。

##### ・球の描画

```

glutWireSphere(double radius, int slices, int stacks); /* 線だけで描画 */
glutSolidSphere(double radius, int slices, int stacks); /* 面を描画 */

```

radius は球の半径, slices と stacks は緯度と経度方向の分割数。

##### ・立方体の描画

```

glutWireCube(double size);
glutSolidCube(double size);

```

size は立方体の大きさ。

##### ・円錐の描画

```

glutWireCone(double radius, double height, int slices, int stacks);
glutSolidCone(double radius, double height, int slices, int stacks);

```

radius は円錐底面の半径, height は高さ, slices と stacks は緯度と経度方向の分割数。

##### ・輪（ドーナツ）の描画

```

glutWireTorus(double innerR, double outerR, int nsizes, int rings);
glutSolidTorus(double innerR, double outerR, int nsizes, int rings);

```

innerR は輪の内側の半径, outerR は輪の外側の半径, nsizes と rings は分割数。

##### ・参考：ティーポットの描画（注意：課題ではティーポットを使用しないこと）

```

glutSolidTeapot(double size);

```

size はティーポットの大きさ。

これらのプリミティブ関数は、法線の設定が関数内で行われているので、別途設定する必要はない。

##### ・円柱の描画は、GLUT では提供されていないので、サンプルプログラムの myShape.h を利用すること。

myShape.h の使い方は、サンプルプログラムの /lesson/CG/sample-cylinder.c を自分のディレクトリにコピーして参考にしなさい。

```
myWireCylinder(float radius, float height, int slices);
mySolidCylinder(float radius, float height, int slices);
```

radius は円柱底面の半径, height は高さ, slices は高さの分割数。

これらのプリミティブは、それぞれ固有の局所的な座標系（ローカル座標系）において定義されている。p.9 3.4.1 で述べた「モデリング変換」をプリミティブの描画に適用することにより拡大縮小・移動・回転し、ワールド座標空間に任意に配置することができる。したがって、モデリング変換を施した複数のプリミティブを定義することにより、プリミティブをブロックのように組み合わせて、より複雑な形状を定義することが出来る。

図 8 に座標系の関係を示す。最終的には、各物体の全頂点座標がそれぞれモデリング変換によりワールド座標空間で表現され、投影変換によって 2 次元スクリーン座標に投影される。

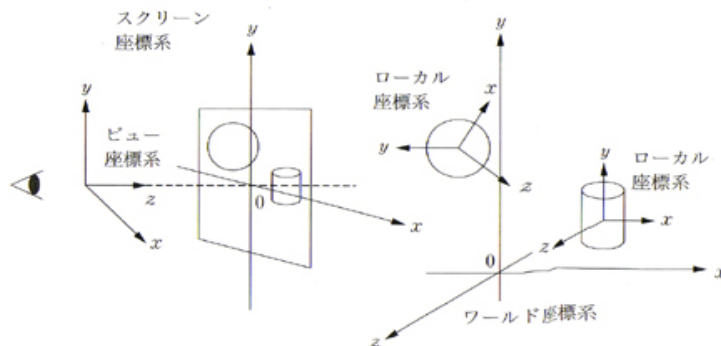


図 8 座標系の関係

### 3.8.2 プリミティブの組み合わせ

立方体と球を組み合わせるとロボットの顔を作る。図 9 に示すようなロボットの顔をモデリングするために、まず図 10 のような設計図を作る。

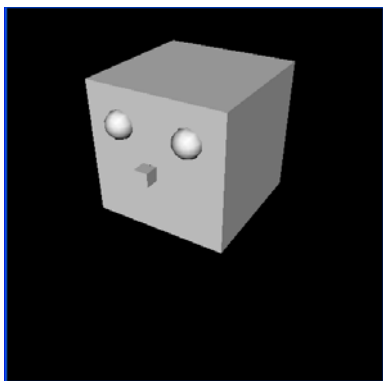


図 9 実行画面

(黒い背景色は一例で実際とは異なる)

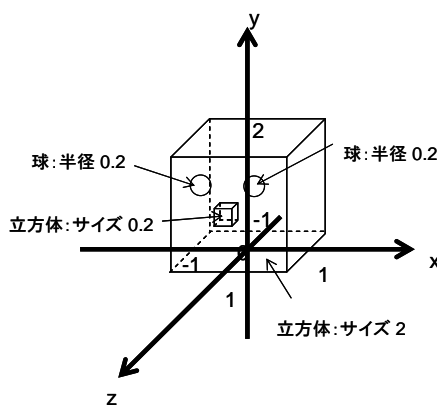
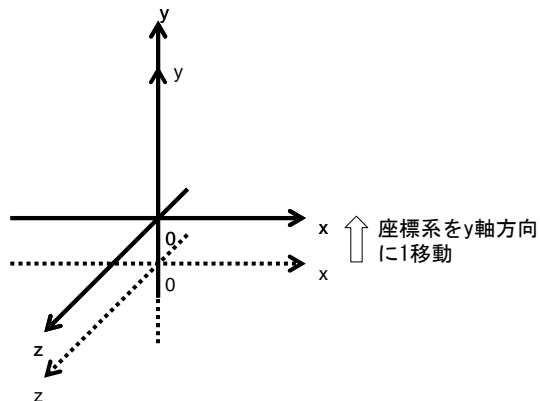
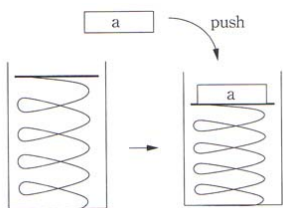


図 10 設計図

以下に、p.11 3.5.1 で述べた `glPushMatrix()` と `glPopMatrix()` を利用して、図 10 のように複数の形状を組み合わせてモデリングを行う方法について説明する。

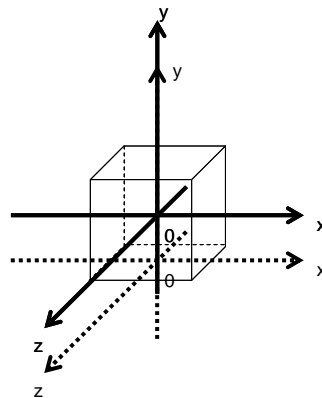
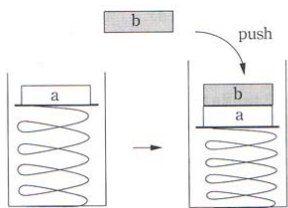
手順 1 :

```
glPushMatrix(); /*最初の座標系を保存*/  
glTranslatef(0.0, 1.0, 0.0); /*座標系を移動*/
```



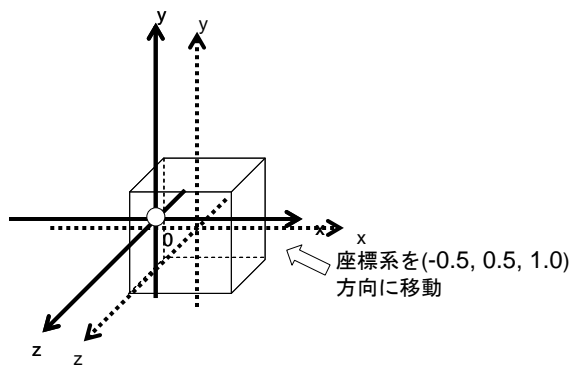
手順 2 :

```
/*移動後の座標系に対して*/  
glutSolidCube(2.0); /*立方体を描画して*/  
glPushMatrix(); /*このときの座標系を保存*/
```



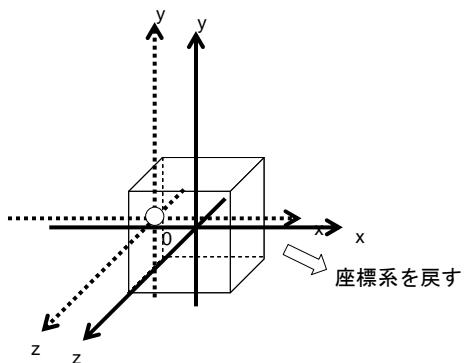
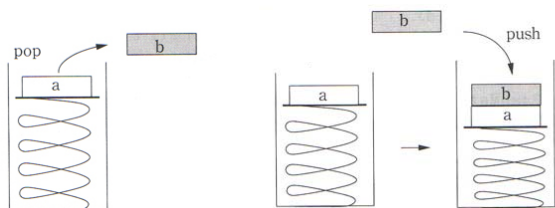
手順 3 :

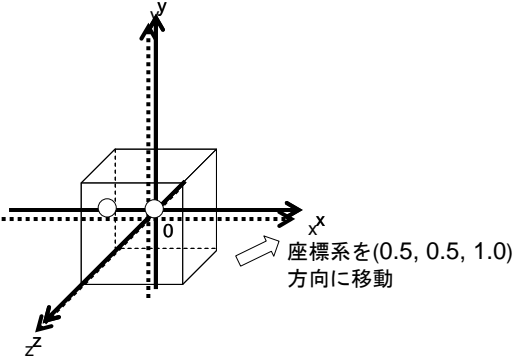
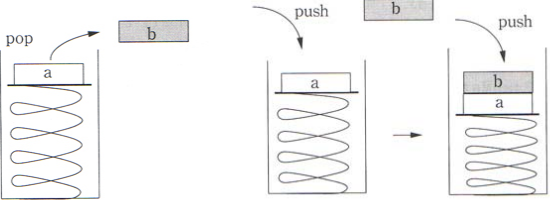
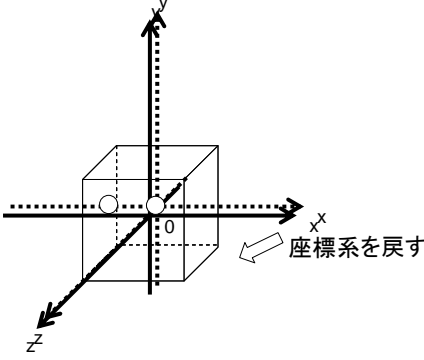
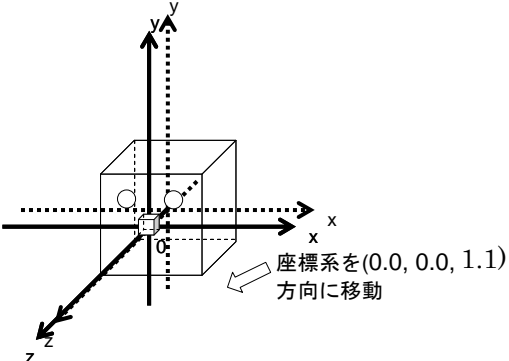
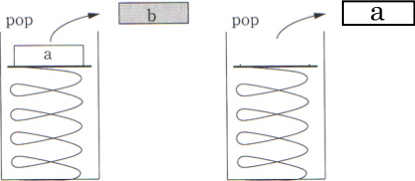
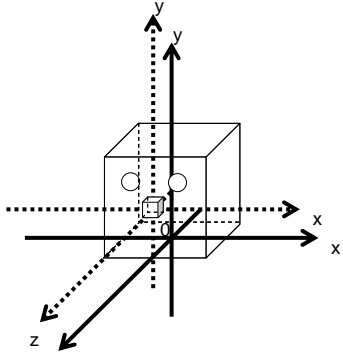
```
/*手順 2 で移動した座標系に対してさらに*/  
glTranslatef(-0.5, 0.5, 1.0); /*座標系を移動*/  
glutSolidSphere(0.2, 10, 10); /*球を描画*/
```



手順 4 :

```
/*手順 2 で保存した座標系に戻す*/  
glPopMatrix();  
glPushMatrix(); /*このときの座標系を保存*/
```



<p>手順 5:</p> <pre>/*手順 4 で戻した座標系に対してさらに*/ glTranslatef(0.5, 0.5, 1.0); /*座標系を移動*/ glutSolidSphere(0.2, 10, 10); /*球を描画*/</pre>	
<p>手順 6:</p> <pre>/*手順 4 で保存した座標系に戻す*/ glPopMatrix(); glPushMatrix(); /*このときの座標系を保存*/</pre> 	
<p>手順 5:</p> <pre>/*手順 6 で戻した座標系に対してさらに*/ glTranslatef(0.0, 0.0, 1.1); /*座標系を移動*/ glutSolidCube(0.2); /*立方体を描画*/</pre>	
<p>手順 7:</p> <pre>/*手順 4 で保存した座標系に戻す*/ glPopMatrix();  /*手順 1 で保存した座標系に戻す*/ glPopMatrix();</pre> 	

### 3.8.3 プリミティブの組み合わせのプログラミング

**課題 8** sample8.c は、kadai7.c を基に前節 3.8.2 で設計したロボットの顔を描画するプログラムである。このプログラムを完成させて実行しなさい。次に、(1)ロボットに耳をつける、(2)マウスを左クリックするとロボットの顔が y 軸周りに回転する、(3)ビューボリュームの上下の開き角を fovy=45 にする、ようにプログラムを変更し kadai8.c を作成しなさい。ロボットの耳は任意のオブジェクトで良い。顔をより複雑に改良しても構わない。

```
1  /* sample8.c */
2  #include <GL/glut.h>
3  #include <math.h>
4  #include <stdlib.h>
5
6  /*■kadai7.c と同じ■*/
7
8  void display(void)
9  {
10     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); /* Zバッファを初期化 */
11
12     glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse);
13     glMaterialfv(GL_FRONT, GL_SPECULAR, specular);
14     glMaterialfv(GL_FRONT, GL_AMBIENT, ambient);
15     glMaterialf(GL_FRONT, GL_SHININESS, shininess);
16     glEnable(GL_LIGHTING);
17
18     glPushMatrix();
19     glTranslatef(0.0, 1.0, 0.0);
20     glutSolidCube(2.0);
21
22     glPushMatrix();
23     glTranslatef(-0.5, 0.5, 1.0);
24     glutSolidSphere(0.2, 10, 10);
25     glPopMatrix();
26
27     glPushMatrix();
28     glTranslatef(0.5, 0.5, 1.0);
29     glutSolidSphere(0.2, 10, 10);
30     glPopMatrix();
31
32     glPushMatrix();
33     glTranslatef(0.0, 0.0, 1.1);
34     glutSolidCube(0.2);
35     glPopMatrix();
36
37     glPopMatrix();
38
39     glDisable(GL_LIGHTING);
40     glutSwapBuffers();
41 }
42
43 void idle(void)
44 {
45     /*■kadai7.c と同じ■*/
46 }
47
48 void mouse(int button, int state, int x, int y)
49 {
50     /*■kadai7.c と同じ■*/
```

```

51 }
52
53 void keyboard(unsigned char key, int x, int y)
54 {
55     /* ■kadai7.c と同じ ■ */
56 }
57
58 void init(char *winname)
59 {
60     glutInitWindowPosition(0, 0);
61     glutInitWindowSize(500, 500);
62     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH); /* Zバッファ利用の宣言 */
63     glutCreateWindow(winname);
64
65     /* kadai7.c と同じ */
66
67     glEnable(GL_DEPTH_TEST); /* 隠面処理の開始を宣言 */
68 }
69
70 int main(int argc, char *argv[])
71 {
72     /* ■kadai7.c と同じ ■ */
73 }

```

(次ページ総合課題につづく)



## 4. 総合課題（課題 9~13）

これまで課題 1~8 で学んだ方法を利用して、以下の 3 次元 CG プログラムを作成せよ。余裕があれば、各課題で追加の機能改良を行っても良い。キー割り当てやマウスなどの操作方法を明記すること。課題の文章は短い、レポートにおいて重要視される部分である。レポートには、プログラム作成にあたっての仕様設計とその意図、考察を充実させて記載すること。

### ■課題 9

頭（課題 8 で作成したもので良い）、胴体、腕 2 本、脚 2 本からなるロボットを作る。各パーツをどのプリミティブ（「ティーポット」の使用は禁止する）で表現するかは各自の選択に任せる。手足を複数のプリミティブの組み合わせで表現しても良い。上記機能を持つプログラム `kadai9.c` を作成せよ。手足は増やして良い。

### ■課題 10

キーボードから `h` を入力すると、ロボットの頭が y 軸を中心に 1 回転するプログラム `kadai10.c` を作成せよ。

### ■課題 11

キーボードから `j` を入力すると、ロボットがジャンプするようなプログラム `kadai11.c` を作成せよ。ジャンプ動作をどのようにモデル化したかについて説明し、その妥当性について考察せよ。

### ■課題 12

マウスを操作してロボットを歩かせる、あるいは移動させるプログラム `kadai12.c` を作成せよ。マウスを使ったロボット移動操作機能をどのような意図に基づいて、どのようにモデル化し実現したか、どのような操作インタフェースとしたか、および、それらの根拠について説明せよ。「特に理由なし」などは無記入とみなす。また、結果の妥当性について考察せよ。操作方法は、前述のマウスボタンイベントのコールバック関数でも良いが、Appendix C にマウスドラッグによる対話操作の参考資料を添付しているので、これも利用するのも良い。その場合、教材フォルダに `glutMotionFunc` を利用したサンプルプログラム `samplex.c` を参考にせよ。

### ■課題 13

一つ以上の新規パーツを任意に追加すると共に、一つ以上の新規対話操作機能をロボットに付加する。対話操作機能は新規パーツに適用しても良いし、ロボット本体に適用しても良い。上記機能を持つプログラム `kadai13.c` を作成せよ。機能内容と操作インタフェースについて詳細に説明し、その目的や意図を述べよ。「特に理由なし」などは無記入とみなす。また、結果の妥当性について考察せよ。なお、例えば「目玉が飛び出す」あるいは「バラバラになる」のような創意工夫のないものは、概ね低い評価とせざるを得ない。本課題は、メディア情報学実験 I 「CG」の集大成であるので、労力を惜しまないこと。

（以上）

## 参考文献・ホームページ

今回の実験で解説した機能の他にも、OpenGL には様々な機能があるので、下記参考文献やホームページを利用して、さらに学習することを推奨する。

- [1] 林武文, 加藤清敬 : OpenGL による 3 次元 CG プログラミング, コロナ社 (2003)
- [2] OpenGL プログラミング ガイド 第 2 版 日本語版, ピアソン・エデュケーション (2002)
- [3] The OpenGL WEB Site : <http://www.opengl.org/>
- [4] GLUT - The OpenGL Utility Toolkit のページ : <http://www.opengl.org/resources/libraries/glut.html>
- [5] GLUT による「手抜き」OpenGL 入門 : <http://www.wakayama-u.ac.jp/~tokoi/opengl/libglut.html>

## Appendix A : Make を使ったコンパイル方法

1. CGの実験用に作成したフォルダに移動(cd コマンドを利用)
2. ls コマンドを実行して, フォルダの中に Makefile というファイルがあるか確認
3. Makefile がなければ以下のコマンドを実行して, /lesson/CG から Makefile をコピーする.  
**cp /lesson/CG/Makefile ./**
4. 次に, emacs で Makefile の中身を見てみましょう.

```
all: sample1 sample2 sample3 sample4

sample1: sample1.c
        cc -o sample1 sample1.c -O2 -Wall -L/usr/X11R6/lib -lm -lX11 -IGL
-IGLU -lglut -lXext -lXmu -lXi

sample2: sample2.c
        cc -o sample2 sample2.c -O2 -Wall -L/usr/X11R6/lib -lm -lX11 -IGL
-IGLU -lglut -lXext -lXmu -lXi

sample3: sample3.c
        cc -o sample3 sample3.c -O2 -Wall -L/usr/X11R6/lib -lm -lX11 -IGL
-IGLU -lglut -lXext -lXmu -lXi

sample4: sample4.c
        cc -o sample4 sample4.c -O2 -Wall -L/usr/X11R6/lib -lm -lX11 -IGL
-IGLU -lglut -lXext -lXmu -lXi
```

5. 中身を見ると, コンパイルのコマンドが順番に書かれていることがわかります.
6. 以下のコマンドを実行してみる.  
**make sample1**  
すると, sample1 がコンパイルされる.
7. また, 引数なしで, 以下のコマンドを実行してみる.  
**make**  
すると, 一行目に書かれたプログラムが全て, すなわち sample1 から sample4 まで,  
次々コンパイルされる. ソースプログラムが存在しない場合にはエラーが出る.
8. 課題を進めて, 新しいプログラムを作ったときには, Makefile に同様にして行を追加する.
9. make コマンドは, 実行ファイルと.c ファイルのどちらが新しいかを調べて, 実行ファイルの方が新しければ, そのプログラムは既にコンパイル済みと判断し, 無駄なコンパイルを行いません. 逆に, .c ファイルの方が新しければ, プログラムが変更されたと判断して, コンパイルを実行します.

参考 URL:

<http://www.unixuser.org/~euske/doc/makefile/>

# Appendix B : Cygwin インストール・ガイド

2005 年 10 月 12 日作成

2007 年 5 月 20 日 Cygwin バージョン更新(1.5.24-2)

## Cygwin とは

Cygwin とは Microsoft Windows プラットフォーム上で UNIX / Linux 環境を実現するものです .Cygwin をインストールすることで GNU gcc コンパイラや gdb デバッガなどの開発ツールを Windows プラットフォーム上で利用できるようになります .また Cygwin に含まれる Cygwin.dll によって UNIX ライクな様々な API が利用できるようになります .

Cygwin は Linux のディストリビュータとして有名な Redhat によって整備されています .Cygwin についての詳しい情報は , <http://www.redhat.com/software/cygwin/> を参照してください .自宅 Windows を利用できる皆さんは , 以下に説明する手順に従って Cygwin をインストールし , C 言語や TEX などの学習に役立ててください .

## Cygwin のダウンロード

本節では Cygwin DLL release version 1.5.24-2 をインストールする手順を説明しています .まずは Cygwin をダウンロードします .ダウンロード元の URL を以下に示します .

<http://sources.redhat.com/cygwin>

この URL を開くと図 1 示すようなページが現れます .ここで図中に丸で示した “Install or update now!” という部分をクリックしてください .

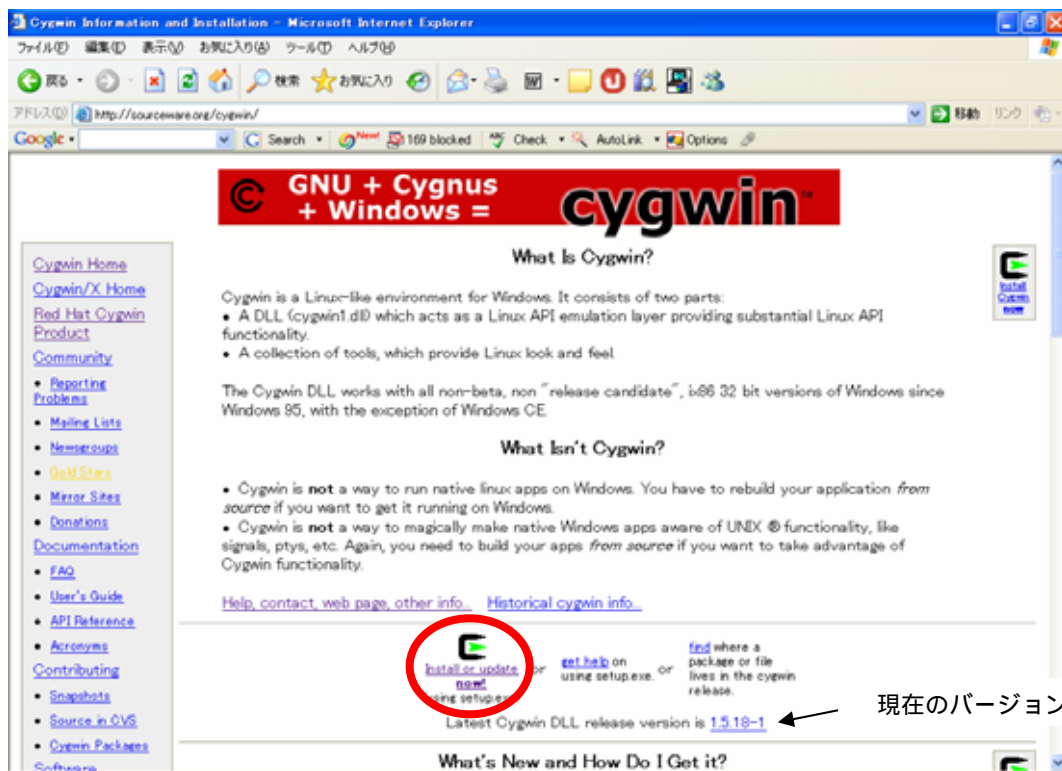


図 1 : Cygwin のダウンロードページ

すると画面に図2に示すようなメッセージが表示されます。各自適当なディレクトリを作成して setup.exe ファイルをダウンロードしてください。ここでは C:\Storage\Develop\Cygwin\1.5.24-2 というフォルダにダウンロードすることとします。

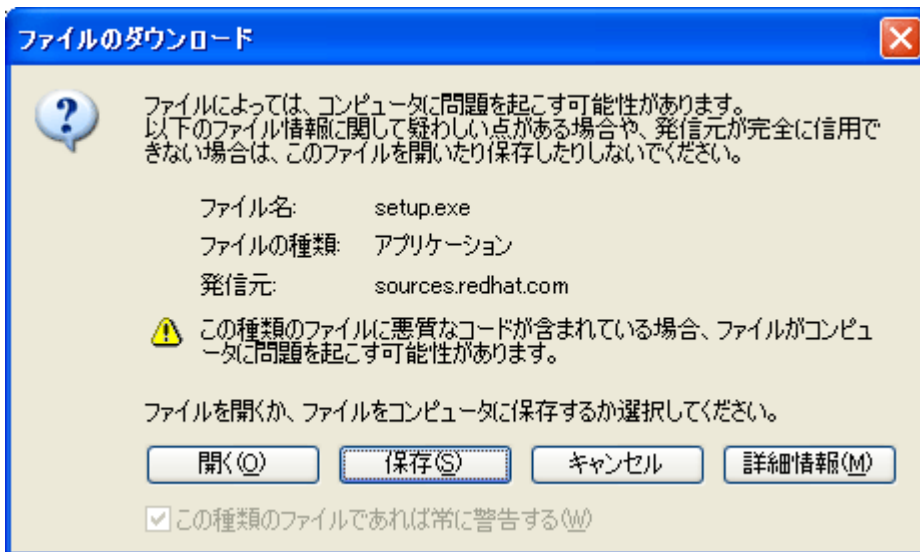


図2：setup.exe ダウンロード時のメッセージ

次にダウンロードした setup.exe をダブルクリックしてセットアップツールを起動します。ツールを起動したら「次へ」を選択して図3に示す状態にします。ここで “Download from Internet” にチェックをつけて「次へ」を選択します。



図3：インストールタイプの選択

パッケージをダウンロードするディレクトリを聞いてくるので先程と同じディレクトリを指定して「次へ」を押します。通常は、同じディレクトリがすでに選ばれているはずですが。続いてインターネットへの接続方法

を選択します。家でインストール作業をする場合は、通常“Direct Connection”や“Use IE5 Setting”問題  
ないはずですが。(もし立命館大学内のネットワーク経由でパッケージをダウンロードする場合は、“Use IE5  
Setting”か“Use HTTP/FTP Proxy”として、“Proxy Host”を“proxy.ritsumei.ac.jp”と設定します)次にダウ  
ンロード先のサイトを選択します。どのサイトでもいいですが、普通は国内のサイト(最後が“jp”となっ  
ている URL)を選んでおけばダウンロードスピードがあがります。ダウンロード先のサイトを選択すると図4に  
示すようなダイアログになります。ここでは矢印で示した部分をクリックして“All”項目の状態を“Install”  
に変更します。なお、初期の状態は“Default”で、状態の変更に時間を要する場合がありますのでクリックし  
た後はしばらく待つようにしてください。

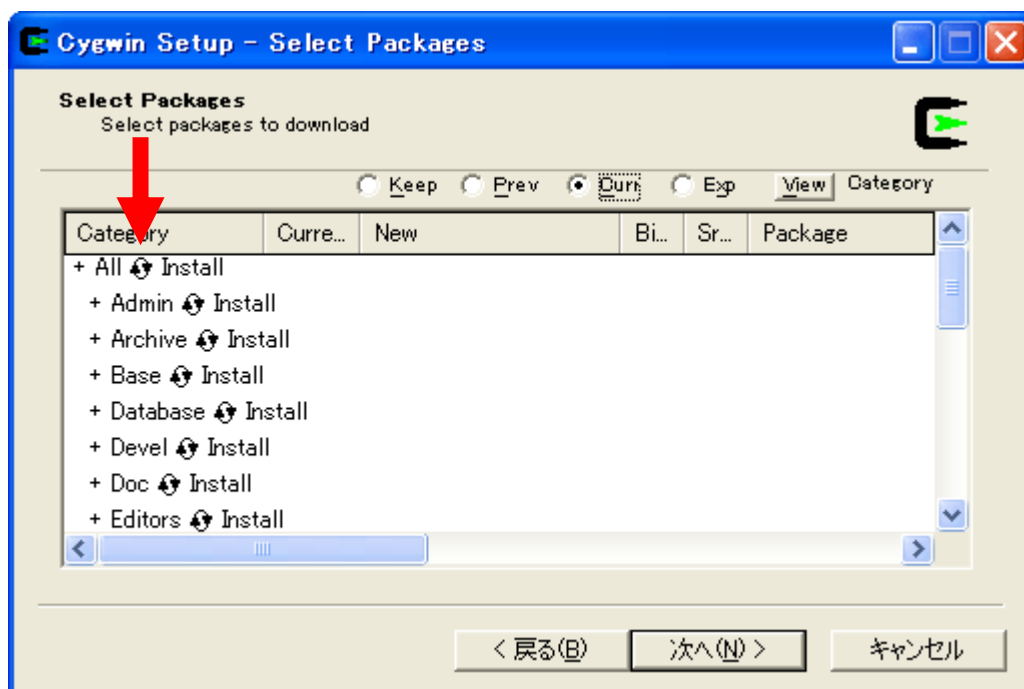


図4：パッケージの選択

ここで「次へ」を押すとインターネットからダウンロードが始まります。このとき、ダウンロードが始まら  
ずに“Dependency”の問題があるという Warning ダイアログが表示されることがありますが、下部のチェ  
ックボックスにチェックを入れて「OK」とすると、ダウンロードが始まります。すべてのダウンロードが終  
了するとセットアップは終了します。ダウンロード容量は約 629MBです。

自宅が電話回線によるダイヤルアップを利用してインターネット接続している場合は、このダウンロード作  
業に非常に時間がかかるので、大学のコンピュータでいったん setup.exe と Cygwin 1.5.24-2 をダウンロード  
し、USB メモリ(1GB 以上)や CD-R などを持ち帰り、自宅では Cygwin のインストールから始めるとよいで  
しょう。

## Cygwin のインストール

次は Cygwin のインストールです。ここでも先程ダウンロードした setup.exe を使います。セットアップツ  
ールを起動して「次へ」を選択し図3に示す状態にします。今度は“Install from Local Directory”にチェッ  
クをつけて「次へ」を選択します。すると図5に示す画面になります。



図 5 : インストール先の選択

ここでは Cygwin のインストール先ルートディレクトリとインストールの対象ユーザ及びデフォルトのテキストファイルの種類を選択します。ここではインストール先ルートディレクトリを “C:\#develop\#cygwin” にすることとし、対象ユーザは “All Users” を、テキストファイルの種類は “Unix” を選択して、「次へ」をクリックします。

次にパッケージの場所を聞いてきますので、上でパッケージをダウンロードしたディレクトリ(実験でインストール用 CD を借りた人は、cygwin\#ftp-file というディレクトリ)を指定して「次へ」をクリックします。するとパッケージの MD5 チェックが始まります。この作業は、ダウンロードしたパッケージが正しいものかどうかを確認する作業です。それが終了すると図 4 に示したパッケージの選択ダイアログが表示されます。ここでも同様に矢印で示した部分をクリックして “All” 項目の状態を “Install” に変更します。そして「次へ」をクリックするとインストール作業が始まります。インストールには 10 分程度の時間がかかります。最後までインストールが終了するとアイコンをどこに作るかを尋ねるダイアログがでます。好きなようにチェック項目を設定して「完了」をクリックすれば Cygwin のインストールは終了です。

## プログラムを書く

図6に示すように、Windowsに標準装備されている”ワードパッド”などのテキストエディタソフトで、プログラムを書きC:\develop\cygwin\homeの中に保存します。テキストエディタは、”ワードパッド”以外のものでも利用できます。保存する際、プログラム名の最後は、必ず”.c”という拡張子をつけるようにします。（例えば、”C:\develop\cygwin\home\ユーザ名”の下に、”Practice”というディレクトリを作り、その中にプログラム”test.c”を保存する）

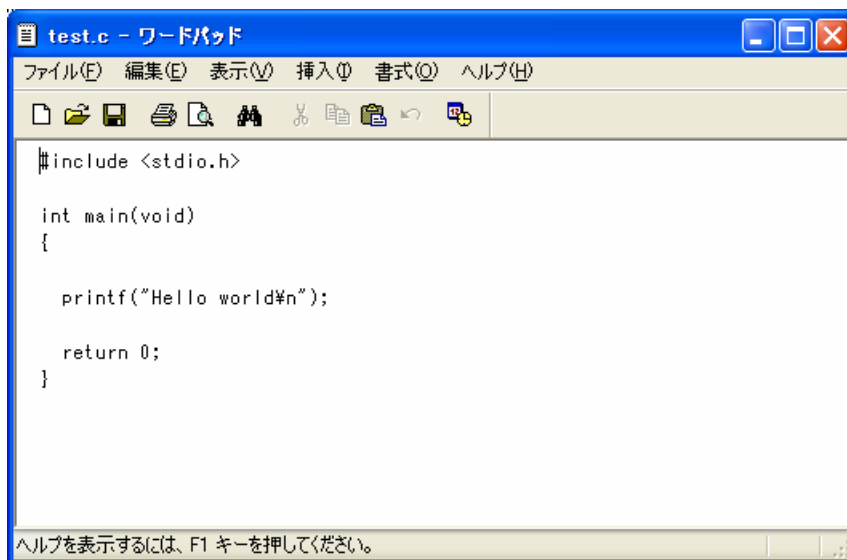


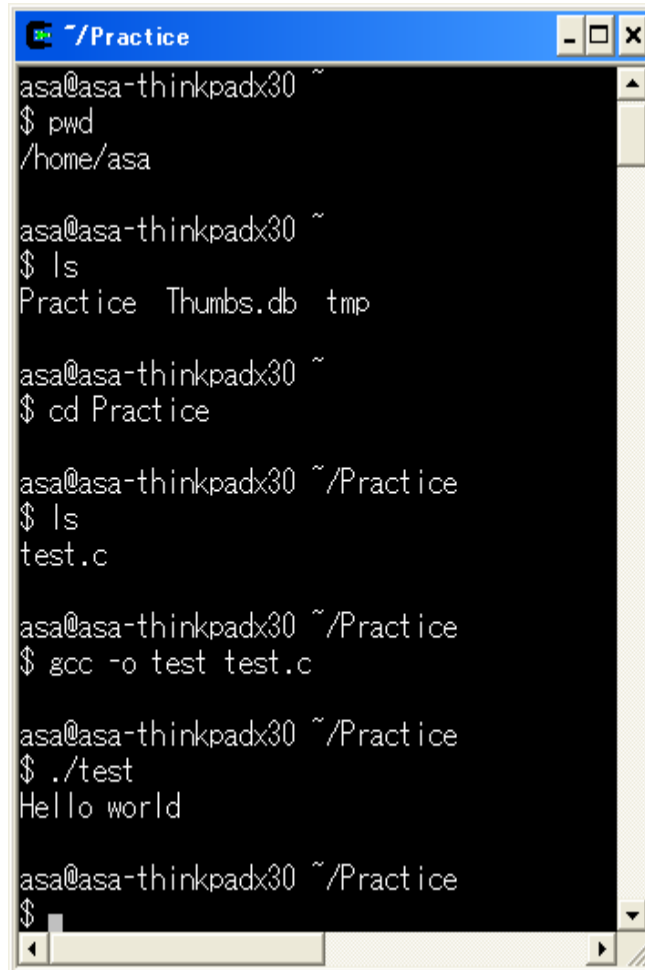
図6：プログラムを書く



## Cygwin の起動とプログラムのコンパイル



Cygwin のアイコン Cygwin.lnk をダブルクリックすると、図 7 に示すウィンドウが表示されます。“pwd”と入力すると、“/home/ユーザ名”が表示されます。(Cygwin では、C:¥develop¥cygwin がディレクトリの一番上の階層となります) 次に、“cd”コマンドを利用して、自分がプログラムを保存したディレクトリに移動します。コンパイルの方法は、授業と同じです。

The image is a screenshot of a Cygwin terminal window. The window title bar shows the path ~/Practice. The terminal content shows a series of commands and their outputs: 'pwd' returns '/home/asa', 'ls' lists 'Practice Thumbs.db tmp', 'cd Practice' changes the directory to ~/Practice, 'ls' lists 'test.c', 'gcc -o test test.c' compiles the program, and './test' outputs 'Hello world'.

```
asa@asa-thinkpadx30 ~  
$ pwd  
/home/asa  
  
asa@asa-thinkpadx30 ~  
$ ls  
Practice Thumbs.db tmp  
  
asa@asa-thinkpadx30 ~  
$ cd Practice  
  
asa@asa-thinkpadx30 ~/Practice  
$ ls  
test.c  
  
asa@asa-thinkpadx30 ~/Practice  
$ gcc -o test test.c  
  
asa@asa-thinkpadx30 ~/Practice  
$ ./test  
Hello world  
  
asa@asa-thinkpadx30 ~/Practice  
$
```

図 7 : Cygwin の画面とコンパイル

## Appendix C: マウスをドラッグする方法

マウスボタンをクリックした状態でマウスポジションを連続的に取得する方法について説明する（下記，参考URLより抜粋）

参考URL:

GLUTによる「手抜き」OpenGL入門

<http://www.wakayama-u.ac.jp/~tokoi/opengl/libglut.html>

```
void init(void)
{
    /* 変更なし */
}

int main(int argc, char *argv[])
{
    /* 変更なし */
}
```

### glVertex2iv(const GLint \*v)

この関数は glVertex2i() と同様に2次元の座標値を設定しますが、引数 v には2個の要素をもつ GLint 型 (int と等価) の配列を指定します。v[0] には x 座標値, v[1] には y 座標値を格納します。この例のように、複数の点の座標を指定する場合に便利です。

## 7.2 マウスをドラッグする

マウスのボタンを押しながらマウスを動かす操作を、**ドラッグ**と言います。ドラッグ中はマウスの位置を継続的に取得する必要がありますが、glutMouseFunc() で指定するハンドラはボタンを押したときにしか実行されないため、この目的には使用できません。

マウスを動かしたときに実行する関数を指定するには、glutMotionFunc() または glutPassiveMotionFunc() を使用します。glutMotionFunc() で指定した関数は、マウスのボタンを押しながらマウスを動かしたときに実行されます。glutPassiveMotionFunc() で指定した関数は、マウスのボタンを押さずにマウスを動かしたときに実行されます。

前のプログラムでは、マウスの左ボタンを押してから離すまでウィンドウには何も表示されませんでした。これを、マウスのドラッグ中は線分をマウスに追従して描くようにします。このような効果を**ラバーバンド** (輪ゴム) と言います。このために glutMotionFunc() を使って、マウスのドラッグ中にラバーバンドを表示するようにします (大川様ご指摘ありがとうございました)。

```
#include <stdio.h>
#include <GL/glut.h>

#define MAXPOINTS 100      /* 記憶する点の数      */
GLint point[MAXPOINTS][2]; /* 座標を記憶する配列 */
int pointnum = 0;         /* 記憶した座標の数   */
int rubberband = 0;       /* ラバーバンドの消去 */

void display(void)
{
    /* 変更なし */
}

void resize(int w, int h)
{
    /* 変更なし */
}

void mouse(int button, int state, int x, int y)
{
    switch (button) {
    case GLUT_LEFT_BUTTON:
        /* ボタンを操作した位置を記録する */
        point[pointnum][0] = x;
        point[pointnum][1] = y;
        if (state == GLUT_UP) {
            /* ボタンを押した位置から離れた位置まで線を引く */
        }
    }
}
```

```
    glColor3d(0.0, 0.0, 0.0);
    glBegin(GL_LINES);
    glVertex2iv(point[pointnum - 1]); /* 一つ前は押した位置 */
    glVertex2iv(point[pointnum]);    /* 今の位置は離れた位置 */
    glEnd();
    glFlush();

    /* 始点ではラバーバンドを描いていないので消さない */
    rubberband = 0;
}
else {
}
if (pointnum < MAXPOINTS) ++pointnum;
break;
case GLUT_MIDDLE_BUTTON:
    break;
case GLUT_RIGHT_BUTTON:
    break;
default:
    break;
}
}

void motion(int x, int y)
{
    static GLint savepoint[2]; /* 以前のラバーバンドの端点 */

    /* 論理演算機能 ON */
    glEnable(GL_COLOR_LOGIC_OP);
    glLogicOp(GL_INVERT);

    glBegin(GL_LINES);
    if (rubberband) {
        /* 以前のラバーバンドを消す */
        glVertex2iv(point[pointnum - 1]);
        glVertex2iv(savepoint);
    }
    /* 新しいラバーバンドを描く */
    glVertex2iv(point[pointnum - 1]);
    glVertex2i(x, y);
    glEnd();

    glFlush();

    /* 論理演算機能 OFF */
    glLogicOp(GL_COPY);
    glDisable(GL_COLOR_LOGIC_OP);

    /* 今描いたラバーバンドの端点を保存 */
    savepoint[0] = x;
    savepoint[1] = y;

    /* 今描いたラバーバンドは次のタイミングで消す */
    rubberband = 1;
}

void init(void)
{
    /* 変更なし */
}

int main(int argc, char *argv[])
{
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(320, 240);
```

```
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_RGBA);
glutCreateWindow(argv[0]);
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutMouseFunc(mouse);
glutMotionFunc(motion);
init();
glutMainLoop();
return 0;
}
```

#### **glEnable(GLenum cap)**

引数 cap に指定した機能を使用可能にします。GL\_LOGIC\_OP もしくは GL\_COLOR\_LOGIC\_OP は、図形の描画の際にウィンドウに既に描かれている内容と、これから描こうとする内容の間で論理演算を行うことができますようにします。

#### **glDisable(GLenum cap)**

引数 cap に指定した機能を使用不可にします。

#### **glLogicOp(GLenum opcode)**

引数 opcode にはウィンドウに描かれている内容と、これから描こうとする内容との間で行う論理演算のタイプを指定します。GL\_COPY はこれから描こうとする内容をそのままウィンドウ内に描きます。GL\_INVERT はウィンドウに描かれている内容の、これから描こうとする図形の領域を反転します。詳しくは `man glLogicOp` を参照してください。

#### **glutMotionFunc(void (\*func)(int x, int y))**

引数 func には、マウスのいずれかのボタンを押しながらマウスを動かしたときに実行する関数のポインタを与えます。この関数の引数 x と y には、現在のマウスの位置が渡されます。この設定を解除するには、引数に 0 (ヌルポインタ) を指定します (stdio.h 等の中で定義されている記号定数 NULL を使用しても良い)。

ラバーバンドを実現する場合、マウスを動かしたときに直前に描いたラバーバンドを消す必要があります。また、ラバーバンドを描いたことによってウィンドウに既に描かれていた内容が壊されてしまうので、その部分をもう一度描き直す必要があります。しかし、そのために画面全体を書き換えるのは、ちょっともったいない気がします。

そこでラバーバンドを描く際には、線を背景とは異なる色で描く代わりに、描こうとする線上の画素の色を反転するようにします。こうすればもう一度同じ線上の画素の色を反転することで、そこに描かれていた以前の線が消えてウィンドウに描かれた図形が元に戻ります。このために `glLogicOp()` を使用します。`glLogicOp()` で指定した論理演算は、`glEnable(GL_LOGIC_OP)` <白黒の場合> あるいは `glEnable(GL_COLOR_LOGIC_OP)` <カラーの場合> で有効になります (陳先生ご指摘ありがとうございました)。

ただし、マウスのボタンを押した直後はまだラバーバンドは描かれていませんから、そのときだけラバーバンドの消去は行わないようにしなければなりません。このため `rubberband` なんていう変数を使ったちょっと泥臭いプログラムになっていますが、我慢してください (もっとエレガントな方法もありますけど...)

`glutMotionFunc()`、`glutPassiveMotionFunc()` で指定した関数は、マウスの移動にともなって頻繁に実行されるので、この関数の中で時間のかかる処理を行うと、マウスの応答が悪くなってしまいます。これを避ける方法は9節以降で解説します。

## 7.3 キーボードから読み込む

OpenGL のアプリケーションプログラムが開いたウィンドウには、ターミナルウィンドウのようにキー