＜おまけ＞
OpenGLで絵を書こう.

GLUTによる「手抜き」OpenGL入門

http://www.wakayama-u.ac.jp/~tokoi/opengl/libglut.html#1

から

空のウィンドウを作成

```
#include <GLUT/glut.h>

void display(void)
{
}

int main(int argc, char *argv[])
{
  glutInit(&argc, argv);
  glutCreateWindow(argv[0]);
  glutDisplayFunc(display);
  glutMainLoop();
  return 0;
}
```

OpenGLを使うためのヘッダ

void glutInit(int *argcp, char **argv)

GLUT および OpenGL 環境を初期化します. 引数には main の引数をそのまま渡します.

int glutCreateWindow(char *name)

ウィンドウを開きます. 引数 name はそのウィンドウの名前の文字列で, タイトルバーなどに表示されます. 以降の OpenGL による図形の描画等は, 開いたウィンドウに対して行われる.

void glutDisplayFunc(void (*func)(void))

引数 func は開いたウィンドウ内に描画する関数へのポインタ. ウィンドウを再描画する必要があるときに, この関数が実行される. この関数内で図形表示を行います.

void glutMainLoop(void)

これは無限ループです. この関数を呼び出すことで, プログラムはイベントの待ち受け状態になります.

コンパイルと実行，終了

- コンパイル

  gcc -framework GLUT –framework OpenGL –o **** ****.c

  ****のところにはファイル名をいれる.

- 実行

  ./*****　今までといっしょ.　（****はファイル名）

- 終了

  Ctrl+C でウィンドウを閉じる.

プログラムの流れ

１．初期化して,
２．ウィンドウを開いて,
３．そのウィンドウ内に絵を描く関数を決めて,
４．何かことが起こるのを待つ.

最初に display() が実行されるのは, 初めてウィンドウが開いたとき, すなわち, glutMainLoop() が glutCreateWindow() の指示を受けてウィンドウの生成を完了したとき. ウィンドウの再描画が必要になったときに実行される.

ウィンドウを塗りつぶす(1)

```
#include <GLUT/glut.h>

void display(void){
  glClear(GL_COLOR_BUFFER_BIT);
  glFlush();
}

void init(void){
  glClearColor(0.0, 0.0, 1.0, 1.0);
}

int main(int argc, char *argv[]){
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGBA);
  glutCreateWindow(argv[0]);
  glutDisplayFunc(display);
  init();
  glutMainLoop();
  return 0;
}
```

void glClear(GLbitfield mask)

ウィンドウを塗りつぶします. mask には塗りつぶすバッファを指定します. OpenGL が管理する画面上のバッファ (メモリ) には, 色を格納するカラーバッファの他, いくつかのものがあり, これらが一つのウィンドウに重なって存在しています. mask に GL_COLOR_BUFFER_BIT を指定したときは, カラーバッファだけが塗りつぶされます.

glFlush(void)

glFlush() はまだ実行されていない OpenGL の命令を全部実行します. OpenGL は関数呼び出しによって生成される OpenGL の命令をその都度実行するのではなく, いくつか溜め込んでおいてまとめて実行します. ひんぱんに glFlush() を呼び出すと, かえって描画速度が低下します.

ウィンドウを塗りつぶす(2)

```
#include <GLUT/glut.h>

void display(void){
  glClear(GL_COLOR_BUFFER_BIT);
  glFlush();
}

void init(void){
  glClearColor(0.0, 0.0, 1.0, 1.0);
}

int main(int argc, char *argv[]){
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGBA);
  glutCreateWindow(argv[0]);
  glutDisplayFunc(display);
  init();
  glutMainLoop();
  return 0;
}
```

void glClearColor(GLclampf R, GLclampf G, GLclampf B, GLclampf A)

glClear(GL_COLOR_BUFFER_BIT) でウィンドウを塗りつぶす際の色を指定します. R, G, B はそれぞれ赤, 緑, 青色の成分の強さを示す値で, 0〜1 の間の値を持ちます. 最後の A はα値と呼ばれ, OpenGL では不透明度として扱われます (0 で透明, 1 で不透明).

void glutInitDisplayMode(unsigned int mode)

ディスプレイの表示モードを設定します. mode に GLUT_RGBA を指定した場合は, 色の指定を RGB (赤緑青, 光の三原色) で行えるようにします. 他にインデックスカラーモード (GLUT_INDEX) も指定できます.

線を引く

```
#include <GLUT/glut.h>

void display(void){
  glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_LINE_LOOP);
   glVertex2d(-0.9,-0.9);
   glVertex2d(0.9,-0.9);
   glVertex2d(0.9,0.9);
   glVertex2d(-0.9,0.9);
   glEnd();
  glFlush();
}

void init(void){
    /* 変更なし */
}

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

void glBegin(GLnum mode)

void glEnd(void)

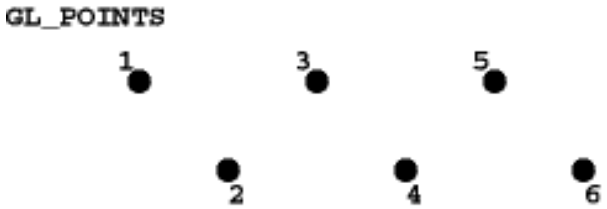
図形を描くには, glBegin()〜glEnd() の間にその図形の各頂点の座標値を設定する関数を置きます. glBegin() の引数 mode には描画する図形のタイプを指定します.

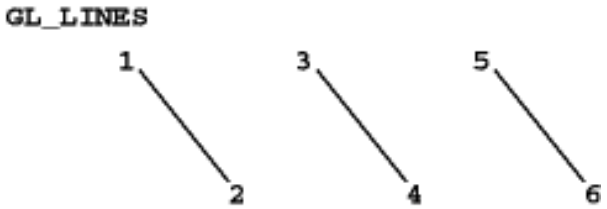
void glVertex2d(GLdouble x, GLdouble y)

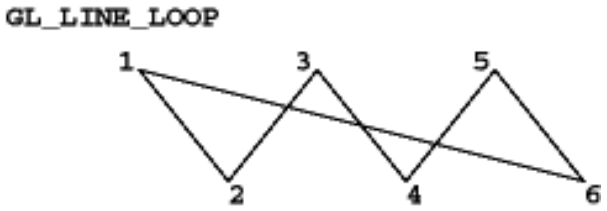
glVertex2d() は二次元の座標値を設定するのに使います. 引数の型は GLdouble (double と等価) です. 引数が float 型のときは glVertex2f(), int 型のときは glVertex2i() を使います.

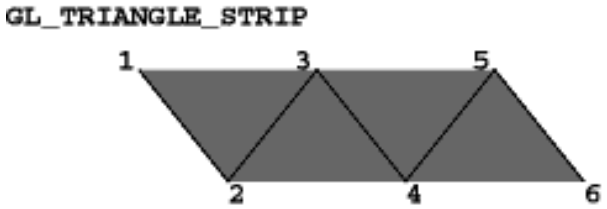
今は，ウィンドウの x 軸と y 軸の範囲が, ともに [-1, 1] に固定されている.

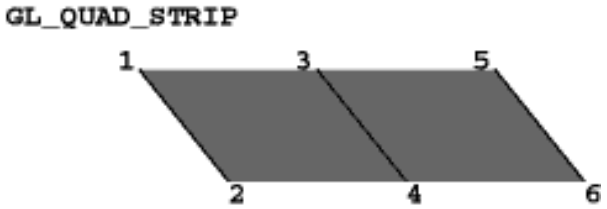
OpenGLでの図形の種類

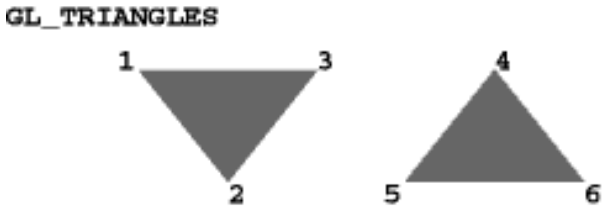


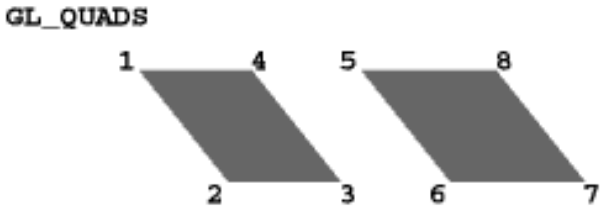


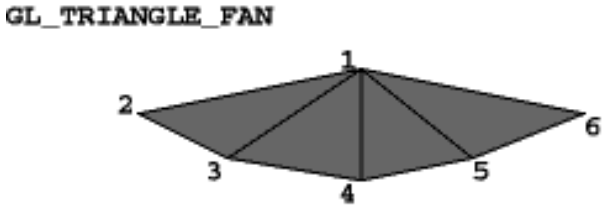


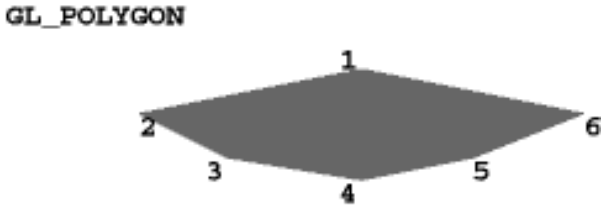








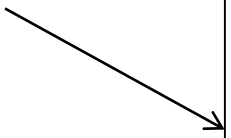




OpenGL を処理するハードウェアは, 実際には三角形しか塗り潰すことができません (モノによっては四角形もできるものもあります). このため GL_POLYGON の場合は, 多角形を三角形に分割してから処理します.

線に色をつける

```
#include <GLUT/glut.h>

void display(void){
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3d(1.0, 0.0, 0.0);
  glBegin(GL_LINE_LOOP);
  glVertex2d(-0.9,-0.9);
  glVertex2d(0.9,-0.9);
  glVertex2d(0.9,0.9);
  glVertex2d(-0.9,0.9);
  glEnd();
  glFlush();
}

void init(void){
    /* 変更なし */
}

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



void glColor3d(GLdouble r, GLdouble g, GLdouble b)

glColor3d() はこれから描画するものの色を指定します. 引数の型は GLdouble 型 (double と等価) で, r, g, b にはそれぞれ赤, 緑, 青の強さを 0〜1 の範囲で指定します. 引数が float 型のときは glColor3f(), int 型のときは glColor3i() を使います.

図形を塗りつぶす(1)

```
#include <GLUT/glut.h>

void display(void){
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3d(1.0, 0.0, 0.0);
  glBegin(GL_POLYGON);
  glVertex2d(-0.9,-0.9);
  glVertex2d(0.9,-0.9);
  glVertex2d(0.9,0.9);
  glVertex2d(-0.9,0.9);
  glEnd();
  glFlush();
}

void init(void){
   glClearColor(1.0, 1.0, 1.0, 1.0);
}

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

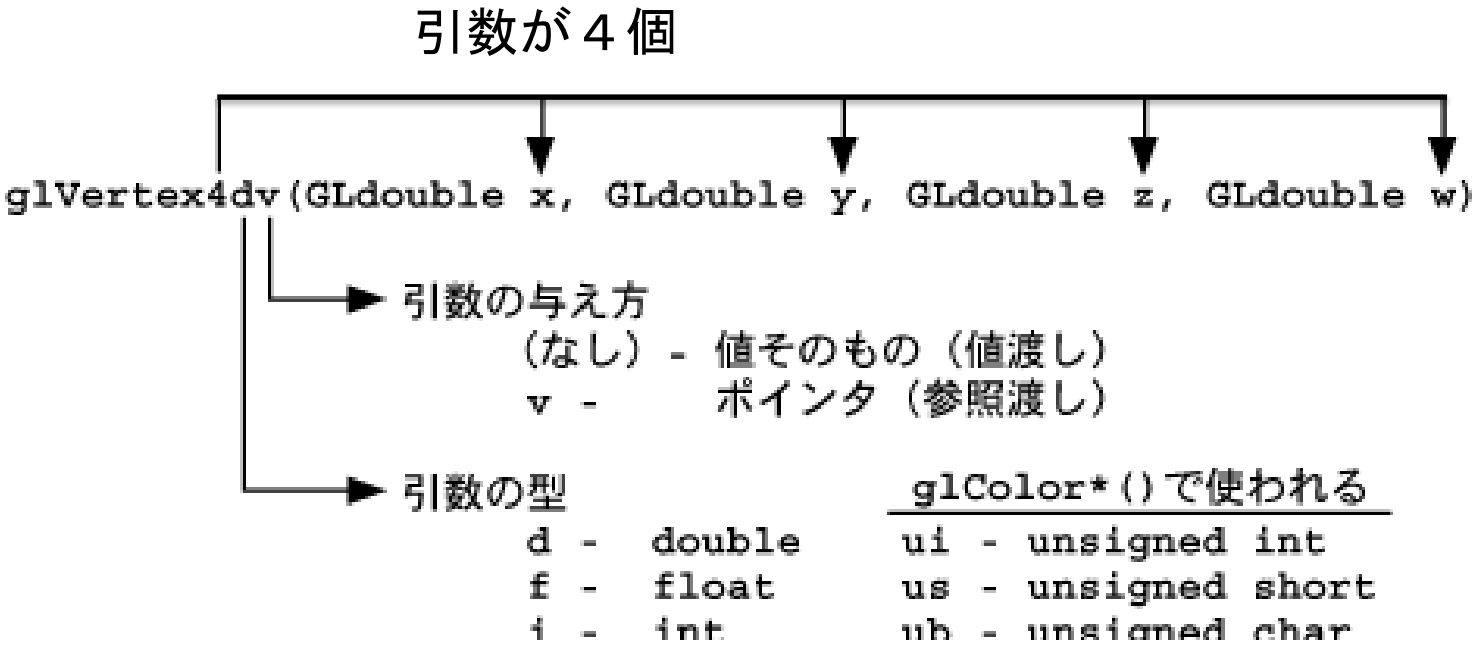
GL_LINE_LOOP を GL_POLYGON に変更し, ついでに背景も白色に変更しましょう.

図形を塗りつぶす(2)

```
#include <GLUT/glut.h>

void display(void){
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_POLYGON);
  glColor3d(1.0, 0.0, 0.0);
  glVertex2d(-0.9,-0.9);
  glColor3d(0.0, 1.0, 0.0);
  glVertex2d(0.9,-0.9);
  glColor3d(0.0, 0.0, 1.0);
  glVertex2d(0.9,0.9);
  glColor3d(1.0, 1.0, 0.0);
  glVertex2d(-0.9,0.9);
  glEnd();
  glFlush();
}

void init(void){ /* 変更なし */ }

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

どうなるか？

関数名



glVertex*() や glColor*() のような関数の * の部分は, 引数の型や数などを示しています. 詳しくは man glVertex2d や man glColor3d を参照してください.

座標軸とビューポート

- ウィンドウ内に表示する図形の座標軸は，そのウィンドウ自体の大きさと図形表示を行う“空間”との関係で決定される．
- 次の例では，表示内容の大きさを変えずに表示領域のみを広げるようにしてみる．

座標軸とビューポート(2)

```
#include <GLUT/glut.h>

void display(void){ /* 変更なし */ }
void resize(int w, int h)
{
  glViewport(0, 0, w, h);     /* ウィンドウ全体をビューポートにする */
  glLoadIdentity(); /* 変換行列の初期化 */
  glOrtho(-w/200.0, w/200.0, -h/200.0, h/200.0, -1.0, 1.0);
  /* スクリーン上の表示領域をビューポートの大きさに比例させる */
}

void init(void){ /* 変更なし */ }

int main(int argc, char *argv[]){
  …
   /* init( ) の前にいれる.    */
  glutReshapeFunc(resize);
  ….
}
```

void glViewport(GLint x, GLint y, GLsizei w, GLsizei h)

ビューポートを設定. ビューポートとは, 開いたウィンドウの中で, 実際に描画が行われる領域. 最初の二つの引数 x, y にはその領域の左下隅の位置, w には幅, h には高さをデバイス座標系での値, すなわちディスプレイ上の画素数で指定. 関数 resize() の引数 w, h にはそれぞれウィンドウの幅と高さが入っているので, glViewport(0, 0, w, h) はリサイズ後のウィンドウの全面が表示領域.

座標軸とビューポート(2)

```
void resize(int w, int h)
{
  glViewport(0, 0, w, h);     /* ウィンドウ全体をビューポートにする */
  glLoadIdentity(); /* 変換行列の初期化 */
  glOrtho(-w/200.0, w/200.0, -h/200.0, h/200.0, -1.0, 1.0);
  /* スクリーン上の表示領域をビューポートの大きさに比例させる */
}
```

void glLoadIdentity(void)

変換行列を初期化. 座標変換の合成は行列の積で表されるので, 変換行列には初期値として単位行列を設定.

glutReshapeFunc(void (*func)(int w, int h))

引数 func には, ウィンドウがリサイズされたときに実行する関数のポインタを与える. 引数はリサイズ後のウィンドウの幅と高さが渡される.

void glOrtho(GLdouble l, GLdouble r, GLdouble b, GLdouble t, GLdouble n, GLdouble f)

ワールド座標系を正規化デバイス座標系に平行投影する行列を変換行列に乗じる. 引数には左から, l に表示領域の左端 (left) の位置, r に右端 (right) の位置, b に下端 (bottom) の位置, t に上端 (top) の位置, n に前方面 (near) の位置, f に後方面 (far) の位置を指定します. ビューポートに表示される空間の座標軸を設定.

- ワールド座標系：もともと図形のある座標系.
  - ここから，(l,b) (r,t)を対角とする矩形領域を抜き出す.
- 正規化デバイス座標系：
  - ２点（-1, -1） (1, 1)を対角とする矩形領域．矩形の中心が原点
  - この領域がデバイス座標系（ディスプレイ上）のビューポートに表示される.
  - 結果として，ワールド座標系からglOrtho()で指定した領域を切り取って表示することと同じ.

- ワールド座標系は，奥行き（z方向）も持っており，glOrtho関数では前方面 n（可視範囲の手前側の限界）と後方面 f（遠方の限界）も指定する必要あり.
- 前方面より手前，後方面より遠方は表示されない.

表示図形のサイズを一定にする.

- glOrtho() で指定するの領域の大きさをビューポートの大きさに比例するように設定する.
- ワールド座標系で，l,b,r,t,n,fが定義されていて，元のウィンドウがWxH，新しいウィンドウがwxhであるとする.
- glOrtho(l*w/W,r*w/W,b*h/H,t*h/H,n,f)とする.

ウィンドウの位置やサイズを指定

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

void display(void) {  /* 変更なし */ }
void resize(int w, int h) {  /* 変更なし */ }
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(resize);
  init();
  glutMainLoop();
  return 0;
}
```

void glutInitWindowSize(int w, int h)

新たに開くウィンドウの幅と高さを指定します. これを指定しないときは, 300×300 のウィンドウを開きます.

void glutInitWindowPosition(int x, int y)

新たに開くウィンドウの位置を指定します. これを指定しないときは, ウィンドウマネージャによってウィンドウを開く位置を決定します.

マウスが押されたことを知る.

```
void mouse(int button, int state, int x, int y)
{
  switch (button) {
  case GLUT_LEFT_BUTTON:
    printf("left");
    break;
  case GLUT_MIDDLE_BUTTON:
    printf("middle");
    break;
  case GLUT_RIGHT_BUTTON:
    printf("right");
    break;
  default:
    break;
  }

  printf(" button is ");
  switch (state) {
  case GLUT_UP:
    printf("up");
    break;
  case GLUT_DOWN:
    printf("down");
    break;
  default:
    break;
  }
  printf(" at (%d, %d)¥n", x, y);
}
```

glutMouseFunc(void (*func)(int button, int state, int x, int y))

引数 func には, マウスのボタンが押され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 button には押されたボタン (GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON), state には「押した (GLUT_DOWN)」のか「離した (GLUT_UP)」のか, x と y にはその位置が渡されます.

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

void display(void){
  glClear(GL_COLOR_BUFFER_BIT);
  glFlush();
}

void resize(int w, int h){
  /* ウィンドウ全体をビューポートにする */
  glViewport(0, 0, w, h);

  /* 変換行列の初期化 */
  glLoadIdentity();
}

void mouse(int button, int state, int x, int y){
    /* 先ほどの中身 */
}
```

```
void init(void){
  glClearColor(0.0, 0.0, 1.0, 1.0);
}

int main(int argc, char *argv[])
{
  glutInitWindowPosition(100, 100);
  glutInitWindowSize(320, 240);
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGBA);
  glutCreateWindow(argv[0]);
  glutDisplayFunc(display);
  glutReshapeFunc(resize);
  glutMouseFunc(mouse);
  init();
  glutMainLoop();
  return 0;
}
```

- 左隅上が原点（0,0）となる．デバイス座標系とは異なることに注意．

マウスで線を描く

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

void display(void){  /* 変更なし */     }
void resize(int w, int h){
  /* ウィンドウ全体をビューポートにする */
  glViewport(0, 0, w, h);

  /* 変換行列の初期化 */
  glLoadIdentity();

  /* スクリーン上座標系をマウス座標系に */
  glOrtho(-0.5, (GLdouble)w - 0.5, (GLdouble)h - 0.5, -0.5, -1.0, 1.0);
}
```

glVertex2i(GLint, GLint)
この関数は glVertex2d() と同様に二次元の座標値を設定しますが, 引数の型が GLint 型 (int 型と等価) です.

```
void mouse(int button, int state, int x, int y)
{
  static int x0, y0;
  switch (button) {
  case GLUT_LEFT_BUTTON:
    if (state == GLUT_UP) {
      /* ボタン位置から離した位置まで線を */
      glColor3d(0.0, 0.0, 0.0);
      glBegin(GL_LINES);
      glVertex2i(x0, y0);
      glVertex2i(x, y);
      glEnd();
      glFlush();
    }
    else {
      /* ボタンを押した位置を覚える */
      x0 = x;
      y0 = y;
    }
```

マウスで線を描く (2)

```
break;
  case GLUT_MIDDLE_BUTTON:
                    /* 何もしない */
                    break;
  case GLUT_RIGHT_BUTTON:
                    /* 何もしない */
                       break;
  default:
    break;
  }
}

void init(void){  /* 変更なし */ }

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