コンピュータグラフィックス

コンピュータグラフィックス（以下、CG）を用いたプログラミングついて学ぶ。本日の目標は「OpenGL を使った簡単な 2 次元グラフィックスが描ける」である。

OpenGL

OpenGL というライブラリを使って CG のプログラミングについて学ぶ。OpenGL はシリコングラフィックス社によって開発された CG 用の API（Application Program interface）である。OS に依存しないオープンソースのライブラリで、比較的簡単に３次元のグラフィックスを取り扱うことができる。OS に依存しないということはキーボードやマウスでの入力やディスプレイへの出力に関しては守備範囲外になるので、この部分については別のライブラリが受け持つことになる。授業では GLUT（OpenGL Utility Toolkit）を使う。

簡単な OpenGL プログラムの例

以下の例は白で塗りつぶされたウィンドウが表示されるだけの簡単なプログラムである。

プログラム例１　ウィンドウを表示する

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

void init_opengl(void);                                 // OpenGLの初期化
void display(void);                                     // コールバック関数glutDisplayFunc()用

int main(int argc, char *argv[])
{
        glutInitWindowPosition(100, 100);               // ウィンドウの表示位置の指定
        glutInitWindowSize(200, 200);                   // ウィンドウサイズの指定
        glutInit(&argc, argv);                          // GLUTの初期化
        glutInitDisplayMode(GLUT_RGBA);                 // 表示モードの指定
        glutCreateWindow("create window");              // ウィンドウを生成
        glutDisplayFunc(display);                       // 描画イベント時のコールバック関数の設定

        init_opengl();                                  // OpenGLに関する初期化　一度だけ呼ばれる

        glutMainLoop();                                 // GLUTに関する無限ループ

        return 0;
}

void init_opengl(void)
{
        glClearColor(1.0, 1.0, 1.0, 1.0); // RGBと不透明度の設定　この場合は白となる
}

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

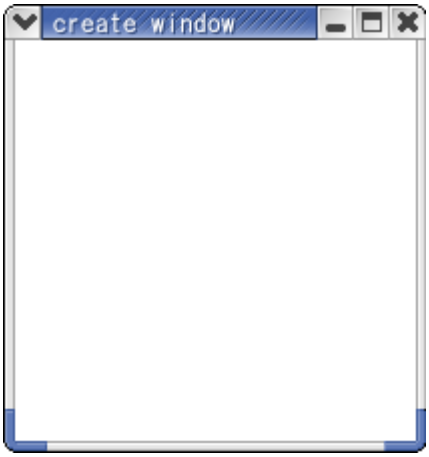


図１　実行結果

GLUT を使ったプログラムはイベント駆動型のプログラムである。イベント駆動型のプログラムはイベントが発生するのを監視して、イベントが発生した場合に記述した処理が実行される仕組みになっている。ここでいうイベントとはキーボードからの入力やマウスのクリックといった操作などのことである。プログラム例１の void display の関数の中で、glFlush();の後に以下のプログラムを追加する。

```
printf("display()が呼び出されました");
```

コンパイルして実行し、表示されたウィンドウを拡大・縮小したり、別のウィンドウを重ねてから元に戻したりするたびにターミナルに以下のように表示されていくことが確認できる。

```
display()が呼び出されました
display()が呼び出されました
display()が呼び出されました
        :
        :
```

これはコールバック関数の display()が描画に関するイベントが発生するたびに呼び出された結果である。描画に関するイベントのコールバック関数の設定は GLUT の関数である glutDisplayFunc()の引数として指定する。この他にもキーボードやマウスなどのイベントの場合に呼び出されるコールバック関数を設定する GLUT の関数があり、必要なものを記述して、glutMainLoop()によって、ループを回し続け、各種のイベントの発生を監視し続ける仕組みとなっている。

２次元グラフィックス

２次元で図を描くプログラムを示す。作図に関する関数はいろいろあるが、ここではいくつかの例を示す。

折れ線による描画

GL_LINE_STRIP を用いると頂点（vertex）を直線で結んだ図形を描くことができる。プログラム例を以下に示す。基本的な使い方は glBegin()と glEnd()の間に頂点を指定していくことで利用する。頂点の指定は glVertex()を用いる。これはウィンドウ上の x 座標と y 座標を引数として与える。ウィンドウの左下は(-1,-1)であり、右上は(1,1)である。

プログラム例２　２次元の図形を表示する　直線で結ぶ

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

void init_opengl(void);                         // OpenGLの初期化
void display(void);                             // コールバック関数glutDisplayFunc()用

int main(int argc, char *argv[])
{
        glutInitWindowPosition(100, 100);       // ウィンドウの表示位置の指定
        glutInitWindowSize(200, 200);           // ウィンドウサイズの指定
        glutInit(&argc, argv);                  // GLUTの初期化
        glutInitDisplayMode(GLUT_RGBA);         // 表示モードの指定
        glutCreateWindow("2D oekaki");          // ウィンドウを生成
        glutDisplayFunc(display);               // 描画イベント時のコールバック関数の設定

        init_opengl();                          // OpenGLに関する初期化　一度だけ呼ばれる

        glutMainLoop();                         // GLUTに関する無限ループ

        return 0;
}

void init_opengl(void)
{
        glClearColor(1.0, 1.0, 1.0, 1.0);       // ウィンドウを白で描画
}

void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT);

        glColor3f(1.0, 0.0, 0.0);               // 色をRGBで指定　この場合は赤
        glBegin(GL_LINE_STRIP);                 // 開始　頂点を直線で結ぶ
        glVertex2f(-0.9, -0.9);                 // 頂点を指定
        glVertex2f( 0.9,  0.9);
        glVertex2f( 0.9, -0.9);
        glVertex2f(-0.9,  0.9);
        glEnd();                                // 終了

        glFlush();
}
```

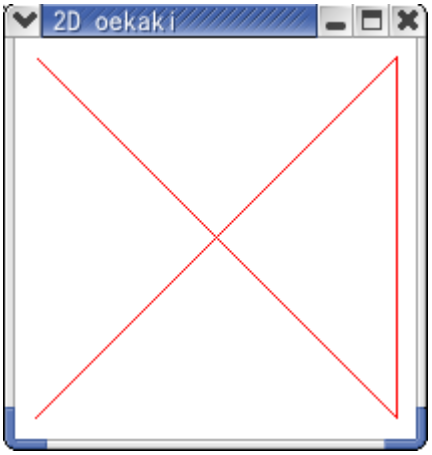


図２　折れ線による図形の描画

複数の図形を描画する

描く図形の数だけ glBegin()と glEnd()を記述することで、複数の図形を描画する。

プログラム例３　２次元の図形を表示する三角と四角

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

void init_opengl(void);                         // OpenGLの初期化
void display(void);                             // コールバック関数glutDisplayFunc()用

int main(int argc, char *argv[])
{
        glutInitWindowPosition(100, 100);       // ウィンドウの表示位置の指定
        glutInitWindowSize(200, 200);           // ウィンドウサイズの指定
        glutInit(&argc, argv);                  // GLUTの初期化
        glutInitDisplayMode(GLUT_RGBA);         // 表示モードの指定
        glutCreateWindow("2D oekaki");          // ウィンドウを生成
        glutDisplayFunc(display);               // 描画イベント時のコールバック関数の設定

        init_opengl();                          // OpenGLに関する初期化　一度だけ呼ばれる

        glutMainLoop();                         // GLUTに関する無限ループ

        return 0;
}

void init_opengl(void)
{
        glClearColor(1.0, 1.0, 1.0, 1.0);       // ウィンドウを白で描画
}

void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT);
```

```
        glColor3f(0.0, 0.0, 1.0);                // 色をRGBで指定　この場合は青
        glBegin(GL_TRIANGLES);                   // 開始　三角形を描く
        glVertex2f(-0.9, -0.9);                  // 頂点を指定
        glVertex2f( 0.9, -0.9);
        glVertex2f(-0.9,  0.9);
        glEnd();                                 // 終了

        glColor3f(1.0, 1.0, 0.0);                // 色をRGBで指定　この場合は黄色
        glBegin(GL_QUADS);                       // 開始　四角形を描く
        glVertex2f( 0.2,  0.2);                  // 頂点を指定
        glVertex2f( 0.2,  0.6);
        glVertex2f( 0.6,  0.6);
        glVertex2f( 0.6,  0.2);
        glEnd();                                 // 終了

        glFlush();
}
```

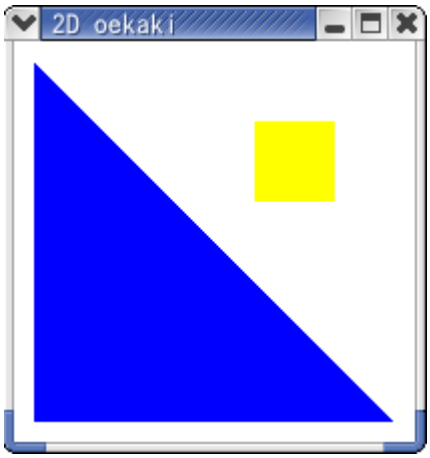


図３　複数の図形の描画

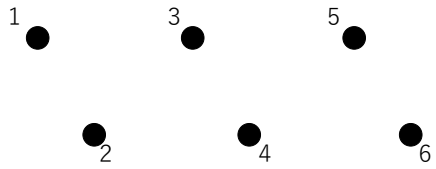
演習

演習１　プログラム例１について、イベント駆動型のプログラムの動作を確認する。テキストで説明した「display()が呼び出されました」という表示を行うプログラムに変更して動作を確認しなさい。

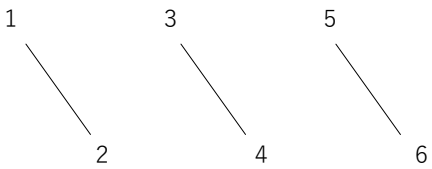
演習２　以下のタイプの図形を作図しなさい。

- GL_POINTS:点を打つ
- GL_LINES：2 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　最後の点は最初の点と結ばれる
- GL_TRIANGLES：3 点を組として、三角形を描く
- GL_QUADS：4 点を組として、四角形を描く
- GL_TRIANGLE_STRIP：一辺を共有しながら帯状に三角形を描く
- GL_QUAD_STRIP:一辺を共有しながら帯状に四角形を描く
- GL_TRIANGLE_FAN:一辺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

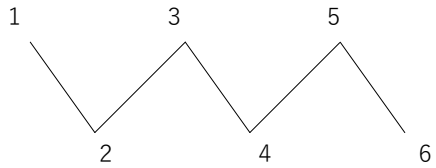
GL_POINTS



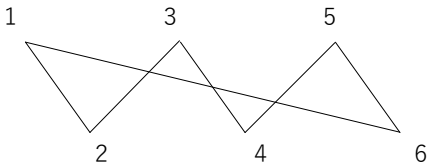
GL_LINES



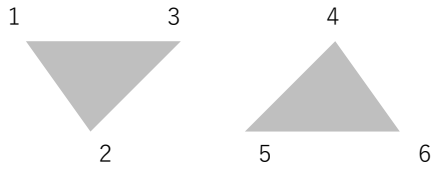
GL_LINE_STRIP



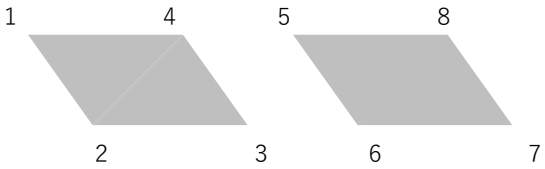
GL_LINE_LOOP



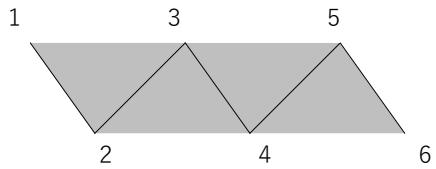
GL_TRIANGLES



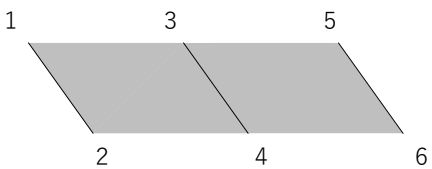
GL_QUADS



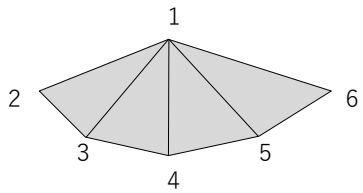
GL_TRIANGLE_STRIP



GL_QUAD_STRIP



GL_TRIANGLE_FAN



GL_POLYGON