シューティングゲーム

シューティングゲームのプログラムを配布する。いくつか習っていないものもあるので、補足の説明を加えていく。

文字列の描画

文字の描画は glutBitmapCharacter()を用いる。これは以下のようにして利用する。

```
int i;
char *str = "Display String";
glColor3f(0.0, 0.0, 0.0);                                   // 色の指定
glRasterPos2f(10, 20);                                      // 座標の指定
for(i = 0; i < strlen(str); i++)                            // 一文字ずつ描画
        glutBitmapCharacter(GLUT_BITMAP_9_BY_15, str[i]);   // 一文字描画
```

座標の指定は glRasterPos2f で行い、ｘ、ｙ座標を指定する。原点はウィンドウの左下である。strlen は文字数をカウントする関数であり、string.h が必要となる。glutBitmapCharacter()は 1 文字を表示する関数であり、繰り返し処理と併せて用いることで文字列を表示する。この関数の第一引数では文字の種類（フォント）を指定する。これには、以下のような種類がある。

フォントの種類 : GLUT_BITMAP_8_BY_13, GLUT_BITMAP_9_BY_15, GLUT_BITMAP_TIMES_ROMAN_10,
GLUT_BITMAP_TIMES_ROMAN_24, GLUT_BITMAP_HELVETICA_10, GLUT_BITMAP_HELVETICA_12,
GLUT_BITMAP_HELVETICA_18

以下にサンプルプログラムを用意したので、確認しておくこと。こちらでは使い勝手がいいように、文字列描画用の関数を定義した。この方法での文字列描画は日本語の表示ができない。OpenGL が多バイト文字に対応していないためである。web で確認したところ、どうやら日本語の表示を行うには OpenGL 以外の別のライブラリと組み合わせて行っているようだ。これについては授業では行わない。

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

#define WINDOW_WIDTH                    500     // ウィンドウサイズX
#define WINDOW_HIGHT                    200     // ウィンドウサイズY

void init_opengl(void);                         // OpenGLの初期化
void display(void);                             // コールバック関数glutDisplayFunc()用
void resize(int w, int h);                      // コールバック関数glutReshapeFunc()用
void display_string(char *str, void *font);     // 文字列描画

int main(int argc, char *argv[])
{
        glutInitWindowPosition(100, 100); // ウィンドウの表示位置の指定
```

```
        glutInitWindowSize(WINDOW_WIDTH, WINDOW_HIGHT);    // 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("string");         // ウィンドウを生成
        glutDisplayFunc(display);           // 描画イベントのコールバック関数の設定
        glutReshapeFunc(resize);            // ウィンドウリサイズイベント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); // ウィンドウを塗りつぶす色を設定
}

void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT);     // ウィンドウをglClearColor()で設定した色で塗りつぶす

        // 文字列の表示
        glColor3f(0.0, 0.0, 0.0);
        glRasterPos2f(10, 200 - 20);
        display_string("1. GLUT_BITMAP_8_BY_13", GLUT_BITMAP_8_BY_13);

        glColor3f(1.0, 0.0, 0.0);
        glRasterPos2f(20, 200 - 40);
        display_string("2. GLUT_BITMAP_9_BY_15", GLUT_BITMAP_9_BY_15);

        glColor3f(0.0, 1.0, 0.0);
        glRasterPos2f(30, 200 - 60);
        display_string("3. GLUT_BITMAP_TIMES_ROMAN_10", GLUT_BITMAP_TIMES_ROMAN_10);

        glColor3f(0.0, 0.0, 1.0);
        glRasterPos2f(40, 200 - 80);
        display_string("4. GLUT_BITMAP_TIMES_ROMAN_24", GLUT_BITMAP_TIMES_ROMAN_24);

        glColor3f(1.0, 1.0, 0.0);
        glRasterPos2f(50, 200 - 100);
        display_string("5. GLUT_BITMAP_HELVETICA_10", GLUT_BITMAP_HELVETICA_10);

        glColor3f(0.0, 1.0, 1.0);
        glRasterPos2f(60, 200 - 120);
        display_string("6. GLUT_BITMAP_HELVETICA_12", GLUT_BITMAP_HELVETICA_12);

        glColor3f(1.0, 0.0, 1.0);
        glRasterPos2f(70, 200 - 140);
        display_string("7. GLUT_BITMAP_HELVETICA_18", GLUT_BITMAP_HELVETICA_18);

        glFlush();
}

void resize(int w, int h)
```

```
{
        glLoadIdentity();                   // 変換行列を単位行列に設定

        // 描画するワールド座標系の範囲を指定
        // この場合にはxが0～WINDOW_WIDTH、yが0～WINDOW_HIGHT
        gluOrtho2D(0, WINDOW_WIDTH, 0, WINDOW_HIGHT);

        glViewport(0, 0, w, h);             // ウィンドウの描画領域を指定
}

//=============================================================================
// 文字列描画：strが指す文字列を表示する
//
// 使用例:display_string(str, GLUT_BITMAP_9_BY_15);
//
// フォントの種類:GLUT_BITMAP_8_BY_13, GLUT_BITMAP_9_BY_15,
// GLUT_BITMAP_TIMES_ROMAN_10, GLUT_BITMAP_TIMES_ROMAN_24,
// GLUT_BITMAP_HELVETICA_10, GLUT_BITMAP_HELVETICA_12,
// GLUT_BITMAP_HELVETICA_18
//=============================================================================
void display_string(char *str, void *font)
{
        unsigned int i;

        for(i = 0; i < strlen(str); i++)                                    // 一文字ずつ描画
                glutBitmapCharacter(font, str[i]);                          // 一文字描画
}
```

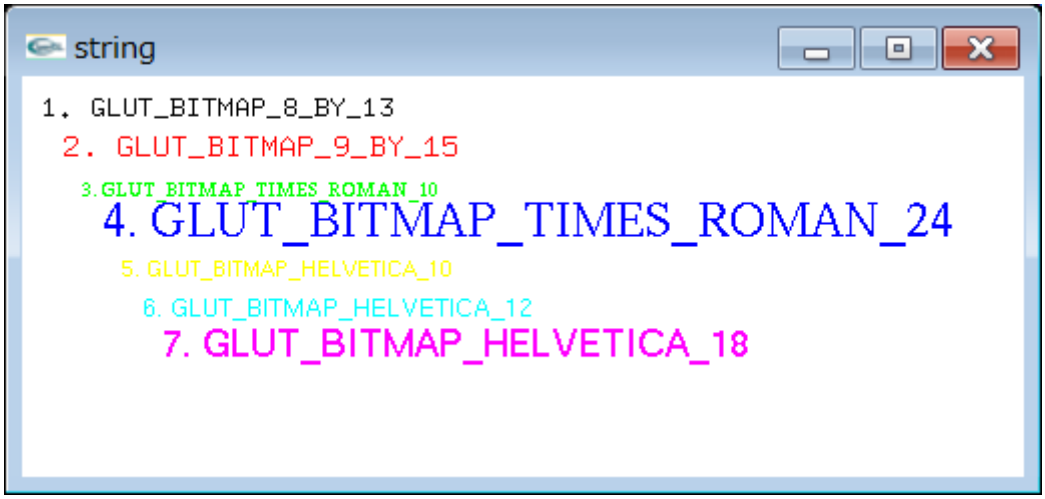

図１　文字列の描画

半透明描画

半透明描画の行い方を示す。次のプログラム例において、半透明描画に関する処理について太字で示した。個別の処理の説明は以下である。

表示モードを半透明描画に設定

```
        glutInitDisplayMode(GLUT_RGBA);
```

ここまでの授業での半透明ではない描画にも同じ設定を用いてきた。これは色の指定に赤緑青αの値を用いるという設定である。αは次で示す「半透明描画の設定」によって不透明度に扱われる。つまり、glColor**4**f()を用いた色の指定に、1.0 を設定すると完全な不透明になり、0.0 を設定すると完全な透明になる。

半透明描画の設定

glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

blend の名前の通り、半透明描画は書き込む場所の色と書き込む図形の色の混合で実現される。この関数で混合の割合を決めている。実際には引数の設定で使い分けていくことになるが、慣れないうちはここで示した例に従ってプログラムを作成していけば良いだろう。その他の使用方法については各自の調査にまかせる。

半透明描画を有効にする

glEnable(GL_BLEND);

この関数を用いて混合処理を有効にする。無効にする関数には glDisable(GL_BLEND)がある。

色の指定

glColor**4**f(0.0, 1.0, 0.0, **0.8**);

不透明度を含めた指定になる。今までは glColor**3**f で RGB を指定したが、この場合には 4 つ目の引数が増えた。0.0～1.0 の値を指定すればよい。

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

#define WINDOW_WIDTH                    200     // ウィンドウサイズX
#define WINDOW_HIGHT                    200     // ウィンドウサイズY

void init_opengl(void);                         // OpenGLの初期化
void display(void);                             // コールバック関数glutDisplayFunc()用
void resize(int w, int h);                      // コールバック関数glutReshapeFunc()用

int main(int argc, char *argv[])
{
        glutInitWindowPosition(100, 100);               // ウィンドウの表示位置の指定
        glutInitWindowSize(WINDOW_WIDTH, WINDOW_HIGHT);     // 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("blend");                      // ウィンドウを生成
        glutDisplayFunc(display);         // 描画イベントのコールバック関数の設定
        glutReshapeFunc(resize);          // ウィンドウリサイズイベント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);                    // ウィンドウを塗りつぶす色を設定
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA); // 半透明描画の設定
        glEnable(GL_BLEND);                                  // 半透明描画を有効にする
}

void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT);

        glBegin(GL_QUADS);
        glColor4f(0.0, 1.0, 0.0, 0.8);                       // 色をRGBAで指定　この場合は緑
        glVertex2f( 0.1, -0.1);
        glVertex2f( 0.1,  0.5);
        glVertex2f(-0.5,  0.5);
        glVertex2f(-0.5, -0.1);

        glColor4f(1.0, 0.0, 0.0, 0.8);                       // 色をRGBAで指定　この場合は赤
        glVertex2f(-0.1,  0.1);
        glVertex2f(-0.1, -0.5);
        glVertex2f( 0.5, -0.5);
        glVertex2f( 0.5,  0.1);
        glEnd();                                             // 終了

        glFlush();
}

void resize(int w, int h)
{
        glLoadIdentity();
        gluOrtho2D(-w / 200.0, w / 200.0, -h / 200.0, h / 200.0);
        glViewport(0, 0, w, h);
}
```

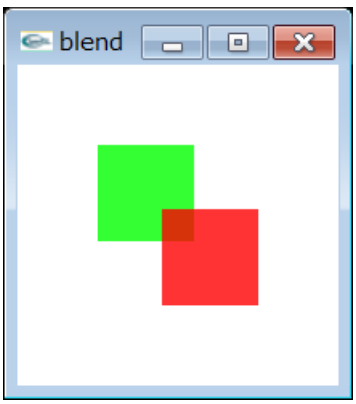

図２　半透明描画

ビットマップ画像の読み込み

ビットマップ画像を読み込んで OpenGL で描画するプログラムについて示す。画像ファイルには他に有名な

ものとして、jpeg や gif といった形式がある。ビットマップファイルは普通、無圧縮で使うため、読み込むプログラムを作成するのは比較的楽である。こちらで提供するヘッダファイルを用いたプログラム例を次に示す。基本的な使用方法は以下である。

インクルード

use_bitmap.hをインクルードする。これをインクルードする前にstdlib.hをインクルードしておく仕様。ヘッダファイル内で動的にメモリを確保するcalloc()関数を利用しているため。ちなみにstdlib.hはglut.hの前に読み込まなければならない。これは両方のヘッダファイルにexit()関数が定義されているため。ヘッダファイル「use_bitmap.h」はインベーダーもどきのサンプルプログラム中のものを使う。

ビットマップファイル用構造体

BitmapFileData bmp_file1;のようにして、ビットマップを扱うための構造体の変数を宣言する。

読み込み

read_bitmap("image1.bmp", &bmp_file1);のように利用する。第1引数は読み込みたいビットマップファイル名、第2引数はビットマップ構造体へのアドレスを指定する。指定するファイル名は実行するディレクトリ内になければ読み込みに失敗する。ビットマップファイル「image1.bmp」はインベーダーもどきのサンプルプログラム中のものを使う。

描画

glRasterPos2i(50, 120);で描画する座標を指定する。

```
glDrawPixels(FIGURE_WIDTH, FIGURE_HIGHT, GL_RGBA, GL_UNSIGNED_BYTE,
        bmp_file1.image_data);
```

第 1、2 引数は読み込む画像の幅と高さの指定、第 3 引数は描画モードの指定、第 4 引数は描画データ形式の指定、最後の引数は実際に描画を行うデータ列へのポインタである。第 3、4 引数には他にも種類がある。このまま利用するか、あるいは各自の調査に任せる。

```
#include <stdio.h>
#include <stdlib.h>              // glut.hとuse_bitmap.hのインクルードの前に必要
#include <GL/glut.h>

#include "use_bitmap.h"

#define WINDOW_WIDTH             200          // ウィンドウサイズX
#define WINDOW_HIGHT             200          // ウィンドウサイズY
#define FIGURE_WIDTH             32           // キャラクタの図の横サイズ
#define FIGURE_HIGHT             20           // キャラクタの図の縦サイズ

void init_setting(void);                      // 初期化処理

void init_opengl(void);                       // OpenGLの初期化
```

```
void display(void);                                 // コールバック関数glutDisplayFunc()用
void resize(int w, int h);                          // コールバック関数glutReshapeFunc()用

BitmapFileData bmp_file1;                           // ビットマップの構造体

int main(int argc, char *argv[])
{
        init_setting();                             // 初期化

        glutInitWindowPosition(100, 100);           // ウィンドウの表示位置の指定
        glutInitWindowSize(WINDOW_WIDTH, WINDOW_HIGHT);    // 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("invader modoki");                 // ウィンドウを生成
        glutDisplayFunc(display);          // 描画イベントのコールバック関数の設定
        glutReshapeFunc(resize);           // ウィンドウリサイズイベント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_setting(void)
{
        read_bitmap("image1.bmp", &bmp_file1);      // ビットマップファイルの読み込み
}

void init_opengl(void)
{
        glClearColor(1.0, 1.0, 1.0, 1.0);           // ウィンドウを塗りつぶす色を設定
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA); // 半透明描画の設定
        glEnable(GL_BLEND);                                 // 半透明描画を有効にする
}

void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT);

        glRasterPos2i(50, 120);
        glDrawPixels(FIGURE_WIDTH, FIGURE_HIGHT, GL_RGBA, GL_UNSIGNED_BYTE,
                bmp_file1.image_data);

        glFlush();
}

void resize(int w, int h)
{
        glLoadIdentity();
        gluOrtho2D(0, WINDOW_WIDTH, 0, WINDOW_HIGHT);
        glViewport(0, 0, w, h);
}
```

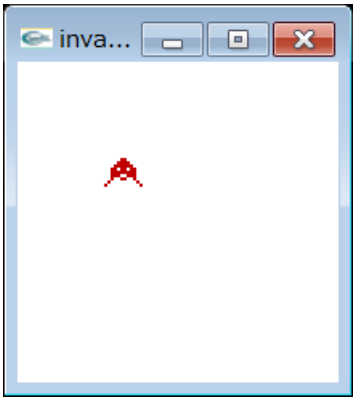

図３　ビットマップファイルを利用した描画

ビットマップファイルのピクセルの白い部分については背景色と同じ色になる。この部分については不透明度を 0 に設定して、描画しないようにしている。use_bitmap.h のファイルの色の読み込む部分でこれを設定しているので、確認しておくこと。

ビットマップファイルには様々な設定が可能であり、このヘッダファイルのプログラムは完全に対応したものではない。このプログラムでは読み込めるビットマップファイルは Windows 形式、色数が 24 ビット、非圧縮のものだけに限定している。

乱数の利用方法

疑似乱数を生成する rand()関数はゲーム制作に良く使われる。例えばある確率で敵がビームを発射したり、RPG で敵から受けたり、敵に与えたりするダメージの量をある程度の幅を持たせてランダムにするといった場合に利用できる。疑似乱数を用いたプログラムは以下のようになる。

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
	int a;

	a = rand() % 10; // aには0～9までの数値が格納される

	if(a < 6)	// aが0～5までの6通りのどれかならば
		printf("6割の確率でこの文が表示されています¥n");
	else		// aが6～9までの4通りのどれかならば
		printf("4割の確率でこの文が表示されています¥n");

	return 0;
}
```

関数 rand()は 0 から最大値 RAND_MAX（#define された定数）までの範囲の値について、関数を実行するたびに違う値を生成する。この得られた値について%を用いて場合分けすることで、確率をプログラムに導入する。この関数は擬似的な乱数を生成するが、上記のような使い方をする上では大きな問題にならない。問題はこのプログラムを数回実行すると確認できるが、毎回同じ結果になってしまうことである。疑似乱数の生

成を行うこの関数は特定のルールに従って、決められた処理を行っているため、このような結果になる。プログラムを実行するたびに結果を変えるためには、次のプログラムのようにする必要がある。

このプログラムでは乱数生成に用いて初期値を設定している。その値は time()関数によって、現在時間を取得しているものを用いた。この方法を使えば、プログラムを実行するたびに異なる結果が得られる。ちなみに srand をプログラム中で何度も呼び出す必要はない。一度呼び出せば十分である。

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
        int a;

        srand((unsigned int)time(NULL));          // 乱数の初期化
        a = rand() % 10; // aには～までの数値が格納される

        if(a < 6)       // aが0～5までの6通りの内ならば
                printf("6割の確率でこの文が表示されています¥n");
        else            // aが6～9までの4通りの内ならば
                printf("4割の確率でこの文が表示されています¥n");

        return 0;
}
```