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します。このプログラムには、まだ学習していない処理も含まれているため、以下の通り、補足説明を加えます。

文字列の描画

文字列の描画は glutBitmapCharacter()を用います。これは以下のようにして利用します。

```
int i, len = (int)strlen(str);
char *str = "Display String";
glColor4d(0.0, 0.0, 0.0, 1.0);                          // 色の指定
glRasterPos2d(10, 20);                                  // 座標の指定
for(i = 0; i < len; i++)                                // 一文字ずつ描画
        glutBitmapCharacter(GLUT_BITMAP_9_BY_15, str[i]);   // 一文字描画
```

表示位置は glRasterPos2d()で指定し、現在の座標系の x, y 座標を指定します。座標系は、gluOrtho2D(0, WINDOW_WIDTH, 0, WINDOW_HEIGHT); によって設定されており、この設定により、x 方向は 0～WINDOW_WIDTH、y 方向は 0～WINDOW_HEIGHT の範囲となり、原点はウィンドウの左下です。その結果、ウィンドウサイズと同じスケールの 2 次元座標系が構築されます。

strlen()は文字列の長さを求める関数であり、使用するには string.h のインクルードが必要です。glutBitmapCharacter()は 1 文字ずつ描画する関数であり、繰り返し処理と組み合わせることで文字列を表示できます。この関数の第一引数では文字の種類（フォント）を指定します。これには、以下のような種類があります。

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、確認してください。使い勝手がいいように、文字列描画用の関数を用意しました。ただし、この方法では日本語を表示できません。日本語を表示するには OpenGL 以外の別のライブラリと組み合わせる必要があります。

```
// 文字列の描画
#include <stdio.h>
#include <string.h>
#include <GL/freeglut.h>                        // freeglutのインクルード

#define WINDOW_WIDTH    500                     // ウィンドウ幅
#define WINDOW_HEIGHT   200                     // ウィンドウ高さ

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

```
void display_string(const char* str, void* font);   // 文字列描画

int main(int argc, char* argv[])
{
    glutInit(&argc, argv);                                  // GLUTの初期化
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);     // 表示モードの指定
    glutInitWindowPosition(100, 100);                       // ウィンドウの表示位置の指定
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT); // ウィンドウサイズの指定
    glutCreateWindow("OpenGLプログラミング");          // ウィンドウを生成

    init_opengl();                                          // OpenGLの初期化
    glutDisplayFunc(display);           // 描画イベント時のコールバック関数の設定
    glutReshapeFunc(reshape);           // ウィンドウサイズ変更時のコールバック関数の設定

    glutPostRedisplay();                                    // ウィンドウの再描画をリクエスト
    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);                 // 背景色のRGBAの設定（この場合は白）

    glMatrixMode(GL_PROJECTION);                        // 行列モードを投影行列に設定
    glLoadIdentity();                                   // 単位行列で初期化
    gluOrtho2D(0, WINDOW_WIDTH, 0, WINDOW_HEIGHT);  // 表示する範囲を設定

    glMatrixMode(GL_MODELVIEW);                         // 行列モードをモデルビュー行列に設定
    glLoadIdentity();                                   // 単位行列で初期化
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);        // glClearColorで設定した色で画面を消去

    // 文字列の表示
    glColor4d(0.0, 0.0, 0.0, 1.0);
    glRasterPos2d(10, 180);
    display_string("1. GLUT_BITMAP_8_BY_13", GLUT_BITMAP_8_BY_13);

    glColor4d(1.0, 0.0, 0.0, 1.0);
    glRasterPos2d(20, 160);
    display_string("2. GLUT_BITMAP_9_BY_15", GLUT_BITMAP_9_BY_15);

    glColor4d(0.0, 1.0, 0.0, 1.0);
    glRasterPos2d(30, 140);
    display_string("3. GLUT_BITMAP_TIMES_ROMAN_10", GLUT_BITMAP_TIMES_ROMAN_10);

    glColor4d(0.0, 0.0, 1.0, 1.0);
    glRasterPos2d(40, 120);
    display_string("4. GLUT_BITMAP_TIMES_ROMAN_24", GLUT_BITMAP_TIMES_ROMAN_24);
```

```
    glColor4d(1.0, 1.0, 0.0, 1.0);
    glRasterPos2d(50, 100);
    display_string("5. GLUT_BITMAP_HELVETICA_10", GLUT_BITMAP_HELVETICA_10);

    glColor4d(0.0, 1.0, 1.0, 1.0);
    glRasterPos2d(60, 80);
    display_string("6. GLUT_BITMAP_HELVETICA_12", GLUT_BITMAP_HELVETICA_12);

    glColor4d(1.0, 0.0, 1.0, 1.0);
    glRasterPos2d(70, 60);
    display_string("7. GLUT_BITMAP_HELVETICA_18", GLUT_BITMAP_HELVETICA_18);

    glutSwapBuffers();                                  // 描画バッファの入れ替え
}

void reshape(int w, int h)
{
    glViewport(0, 0, w, h);                             // ウィンドウ全体を描画範囲にする
}

//=============================================================================
// 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(const char* str, void* font)
{
    int i, len = (int)strlen(str);

    for (i = 0; i < len; i++)
        glutBitmapCharacter(font, str[i]);
}
```

図１　プログラム例１（文字列の描画）

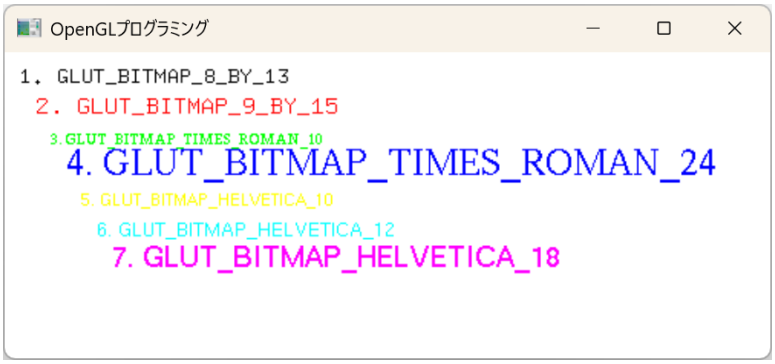

図２　実行結果

半透明描画

半透明描画を扱うプログラム例を示します。半透明描画に関する処理については太字で示しました。

```
// 半透明描画
#include <stdio.h>
#include <GL/freeglut.h>                                    // freeglutのインクルード

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

int main(int argc, char* argv[])
{
    glutInit(&argc, argv);                                  // GLUTの初期化
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);           // 表示モードの指定
    glutInitWindowPosition(100, 100);                       // ウィンドウの表示位置の指定
    glutInitWindowSize(300, 300);                           // ウィンドウサイズの指定
    glutCreateWindow("OpenGLプログラミング");                // ウィンドウを生成

    init_opengl();                                          // OpenGLの初期化
    glutDisplayFunc(display);               // 描画イベント時のコールバック関数の設定
    glutReshapeFunc(reshape);               // ウィンドウサイズ変更時のコールバック関数の設定

    glutPostRedisplay();                                    // ウィンドウの再描画をリクエスト
    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);                       // 背景色のRGBAの設定（この場合は白）

    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);                           // glClearColorで設定した色で画面を消去

    glBegin(GL_QUADS);                                      // 四角形を描画
    glColor4d(0.0, 1.0, 0.0, 0.8);                          // 色をRGBAで指定（この場合は緑）
    glVertex2d( 0.1, -0.1);
    glVertex2d( 0.1,  0.5);
    glVertex2d(-0.5,  0.5);
    glVertex2d(-0.5, -0.1);

    glColor4d(1.0, 0.0, 0.0, 0.8);                          // 色をRGBAで指定（この場合は赤）
    glVertex2d(-0.1,  0.1);
    glVertex2d(-0.1, -0.5);
```

```
    glVertex2d( 0.5, -0.5);
    glVertex2d( 0.5,  0.1);
    glEnd();                                    // 終了

    glutSwapBuffers();                          // 描画バッファの入れ替え
}

void reshape(int w, int h)
{
    glViewport(0, 0, w, h);                     // ウィンドウ全体を描画範囲にする
}
```

図３　プログラム例２（半透明描画）

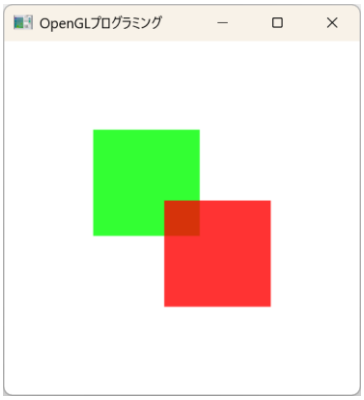

図４　実行結果

以下は半透明描画に関連する処理の説明です。

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

GLUT_RGBA を設定することで、RGBA（赤・緑・青・α）の４成分で色を扱えるようになります。α（アルファ値）は、後述するブレンド設定によって不透明度として扱われます。glColor4d()を用いた α 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ります。

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

半透明描画は、すでに描画されている画素の色と、これから描画する色を混合することで実現されます。この関数でその混合の割合を決めています。上記はよく使われる設定です。他にも種類がありますが、これについては各自の調査にまかせます。

```
glEnable(GL_BLEND);
```

この関数を用いて混合処理を有効にします。無効にする場合は glDisable(GL_BLEND)を使います。必要に応じて、これらを使い分けて描画尾を行います。

```
glColor4d(0.0, 1.0, 0.0, 0.8);
```

4 つ目の引数で、0.0～1.0 の範囲で不透明度を指定します。

ビットマップファイルの利用

ビットマップファイルを読み込んで OpenGL で描画するプログラム例を示します。関連する処理は太字にしました。画像ファイルには他に jpeg や gif といった形式があります。いろいろな形式に対応させるにはライブラリを使うと良く、Nuget に対応しているものとしては native.freeimage などがあります。一方で、ビットマップファイルは一般に無圧縮で扱われるため、読み込むプログラムを作成することは比較的容易です。ファイル構造や読み込み処理を理解することは学習に有用であるため、授業では自作プログラムを提供することとしました。基本的な使用方法は以下となります。

インクルード

use_bitmap.hをインクルードします。このファイルはインベーダーのプログラムの一部として配布します。

ビットマップファイル用構造体

BitmapFileData bmp_file1;のようにして、ビットマップファイルを扱うための構造体の変数を宣言します。

読み込み

read_bitmap("image1.bmp", &bmp_file1);のように利用します。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します。ファイルは実行するディレクトリ内になければ読み込みに失敗します。ビットマップファイル「image1.bmp」はインベーダーのサンプルプログラム中のものを使ってください。

描画

glRasterPos2d(50, 120);で描画する座標を指定します。

glDrawPixels(FIGURE_WIDTH, FIGURE_HIGHT, GL_RGBA, GL_UNSIGNED_BYTE, bmp_file1.image_data);で描画します。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、最後の引数は描画に使用する画素データへのポインタです。第3、4引数には他にも種類がありますが、これについては各自の調査に任せます。

メモリ解放

void free_bitmap(&bmp_file1);のように利用します。read_bitmap()内で calloc()により確保されたメモリにビットマップファイルのデータを格納しています。そのため、データの使用後には free_bitmap()内の free()で解放します。

```
// ビットマップファイルの利用
```

```
#define _CRT_SECURE_NO_WARNINGS             // scanfやfopenに対するコンパイルエラーを抑制する

#include <stdio.h>
#include <GL/freeglut.h>                    // freeglutのインクルード
#include "use_bitmap.h"                     // ビットマップファイル読み込み用ヘッダファイルのインクルード

#define WINDOW_WIDTH    200                 // ウィンドウ幅
#define WINDOW_HEIGHT   200                 // ウィンドウ高さ

void init_opengl(void);                     // OpenGLの初期化
void display(void);                         // コールバック関数glutDisplayFunc()用
void reshape(int w, int h);                 // コールバック関数glutReshapeFunc()用
void keyboard(unsigned char key, int x, int y);     // コールバック関数glutKeyboardFunc()用

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

int main(int argc, char* argv[])
{
    read_bitmap("image1.bmp", &bmp_file1);          // ビットマップファイルの読み込み

    glutInit(&argc, argv);                          // GLUTの初期化
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);   // 表示モードの指定
    glutInitWindowPosition(100, 100);               // ウィンドウの表示位置の指定
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);// ウィンドウサイズの指定
    glutCreateWindow("OpenGLプログラミング");       // ウィンドウを生成

    init_opengl();                          // OpenGLの初期化
    glutDisplayFunc(display);               // 描画イベント時のコールバック関数の設定
    glutReshapeFunc(reshape);               // ウィンドウサイズ変更時のコールバック関数の設定
    glutKeyboardFunc(keyboard);             // キーボードイベント時のコールバック関数の設定

    glutPostRedisplay();                            // ウィンドウの再描画をリクエスト
    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);               // 背景色のRGBAの設定（この場合は白）

    glMatrixMode(GL_PROJECTION);                    // 行列モードを投影行列に設定
    glLoadIdentity();                               // 単位行列で初期化
    gluOrtho2D(0, WINDOW_WIDTH, 0, WINDOW_HEIGHT);  // 表示する範囲を設定

    glMatrixMode(GL_MODELVIEW);                     // 行列モードをモデルビュー行列に設定
    glLoadIdentity();                               // 単位行列で初期化

    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);        // glClearColorで設定した色で画面を消去

    glRasterPos2d(50, 120);          // 描画位置の設定
    glDrawPixels(bmp_file1.info.width, bmp_file1.info.height, GL_RGBA,
                 GL_UNSIGNED_BYTE, bmp_file1.image_data);

    glutSwapBuffers();                                          // 描画バッファの入れ替え
}

void reshape(int w, int h)
{
    glViewport(0, 0, w, h);                          // ウィンドウ全体を描画範囲にする
}

void keyboard(unsigned char key, int x, int y)
{
    if (key == 0x1b)    // Escキー
    {
        free_bitmap(&bmp_file1);
        exit(0);
    }
}
```

図５　プログラム例３（ビットマップファイルの利用）

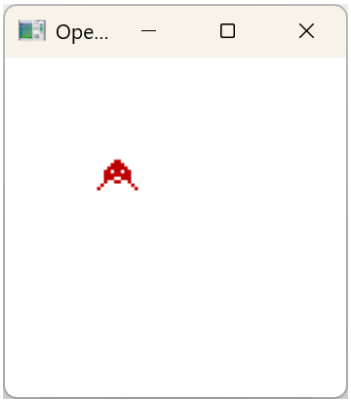

図６　実行結果

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不透明度を 0 に設定して、描画しないようにしています。この処理は use_bitmap.h において、色を読み込む部分で行っています。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、すべてのビットマップファイルの形式に対応しているわけではありません。読み込めるビットマップファイルは Windows 形式で、色数が 24 ビット、非圧縮のものだけに対応します。

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ます。例えば、シューティングゲームで敵

がランダムにビームを発射したり、RPG のダメージ量をある程度の幅を持たせてランダムにしたりできます。疑似乱数を用いたプログラムは以下のようになります。

```
#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;

    if (a < 6)
        printf("6割の確率でこの文が表示されています¥n");
    else
        printf("4割の確率でこの文が表示されています¥n");

    return 0;
}
```

rand()関数は 0 から最大値 RAND_MAX（マクロ定数）までの範囲の疑似乱数を生成します。srand()は乱数生成の開始値を設定する関数です。time()関数は現在時刻を取得します。上記のプログラムによって、実行ごとに違う乱数を生成することができます。さらに、得られた乱数を%の演算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とができます。