情報工学実験2

グラフィックプログラミング基礎

担当教員名：赤嶺有平

実験日 2008/12/08
提出日 2008/01/16

075730G：澤岻千明

課題 1

直線やポリゴンを用いて、オリジナルの図形を描画するプログラムを作成せよ。

project draw の main.cpp のソースと、それを実行した図 1 を以下に示す。

draw の main.cpp のソース

```
/*
 *  main.cpp
 *  draw
 *
 *  Created by C-T on 08/12/08.
 *  Copyright 2008 __MyCompanyName__. All rights reserved.
 *
 */

#include <glut/glut.h>
#include <math.h>

GLubyte mask[128];

//描画イベントハンドラ
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLE_FAN);
    glColor3d(0.5,0.5,0.5); // 灰
    glVertex2d(0, 0);
    glVertex2d(0, 0.875);
    glVertex2d(0.3, 0.8);
    glVertex2d(0.6, 0.6);
    glVertex2d(0.8, 0.3);
    glVertex2d(0.875, 0);
    glVertex2d(0.8, -0.3);
    glVertex2d(0.6, -0.6);
    glVertex2d(0.3, -0.8);
    glVertex2d(0, -0.875);
    glVertex2d(-0.3, -0.8);
    glVertex2d(-0.6, -0.6);
    glVertex2d(-0.8, -0.3);
    glVertex2d(-0.875, 0);
    glVertex2d(-0.8, 0.3);
    glVertex2d(-0.6, 0.6);
```

```
    glVertex2d(-0.3, 0.8);
    glVertex2d(0, 0.875);
    glEnd();

    glBegin(GL_QUADS);
    glColor3d(0,0,0); // 黒
    glVertex2f(0, 0.8);
    glVertex2f(0.8, 0);
    glVertex2f(0, -0.8);
    glVertex2f(-0.8, 0);
    glEnd();

    glEnable(GL_POLYGON_STIPPLE);
    glPolygonStipple(mask);
    glBegin(GL_POLYGON);
    glColor3d(0,0,0); // 黒
    glVertex2f(-1, 1);
    glVertex2f(1, 1);
    glVertex2f(1, -1);
    glVertex2f(-1, -1);
    glEnd();

    glBegin(GL_QUADS);
    glColor3d(0.7,0.7,0.7); // 白
    glVertex2f(0.5, -0.5);
    glVertex2f(0.5, 0.5);
    glVertex2f(-0.5, 0.5);
    glVertex2f(-0.5, -0.5);
    glEnd();

    glFlush();
}
```

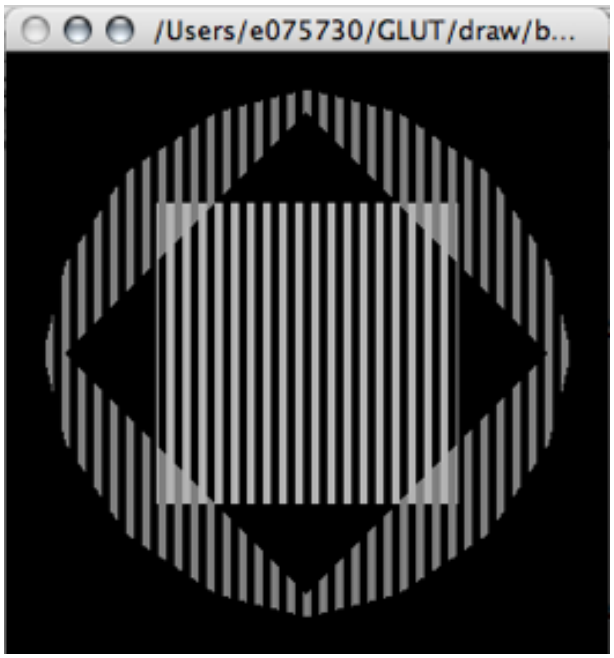

図 1: draw の main.cpp 実行結果

課題 2

基本図形、直線、ポリゴンなどを用いて 3 D モデルを作成せよ。何らかのアニメーション（回転、移動）をつけること。

project 3D の main.cpp の display 関数ソースと、これを実行した際の図 2 を以下に示す。

3D の main.cpp のソース

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

    /* モデルビュー変換行列の初期化 */
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    /* 原点の移動（物体を奥に移動）*/
    glTranslated(0.0, 0.0, -7.0);

    glRotatef(sRotationY,0,1,0); //キーボード操作による回転

    glPushMatrix();// 変換を保存
```

```
        glRotatef((sTime/400.f)*20, 0,0,1); //一回転

        glutWireTorus(1.5, 1, 8, 12); // 回し車

        glPushMatrix();  // 小さい球体の動き -ここから
            glRotatef(sin(sTime/400.f)*20, 0,1,1); //その場での運動
            glTranslatef(0.3,0.3,0);    //原点を移動
            glutSolidSphere(0.1,8,8); //原点に球体をおく
        glPopMatrix();   // 小さい球体の動き -ここまで
    glPopMatrix(); // 直前に保存した変換を復元
    //glPushMatrix() と glPopMatrix() で挟まれている変換命令は外側に影響を与えない

    glRotatef(sin(sTime/400.f)*15, 0,0,1); //その場での動き

    glPushMatrix(); // 上の球体
        glTranslatef(1,2.6,0);    //原点を y 軸方向（画面の縦方向）に移動
        glutSolidSphere(0.5,8,8); //原点に球体をおく
    glPopMatrix();

    glPushMatrix(); // 目
        glTranslatef(0.6,2.7,0.2);
        glutSolidSphere(0.1,8,8);

        glTranslatef(0,0,-0.4);
        glutSolidSphere(0.1,8,8);
    glPopMatrix();

    glutSwapBuffers();
}
```

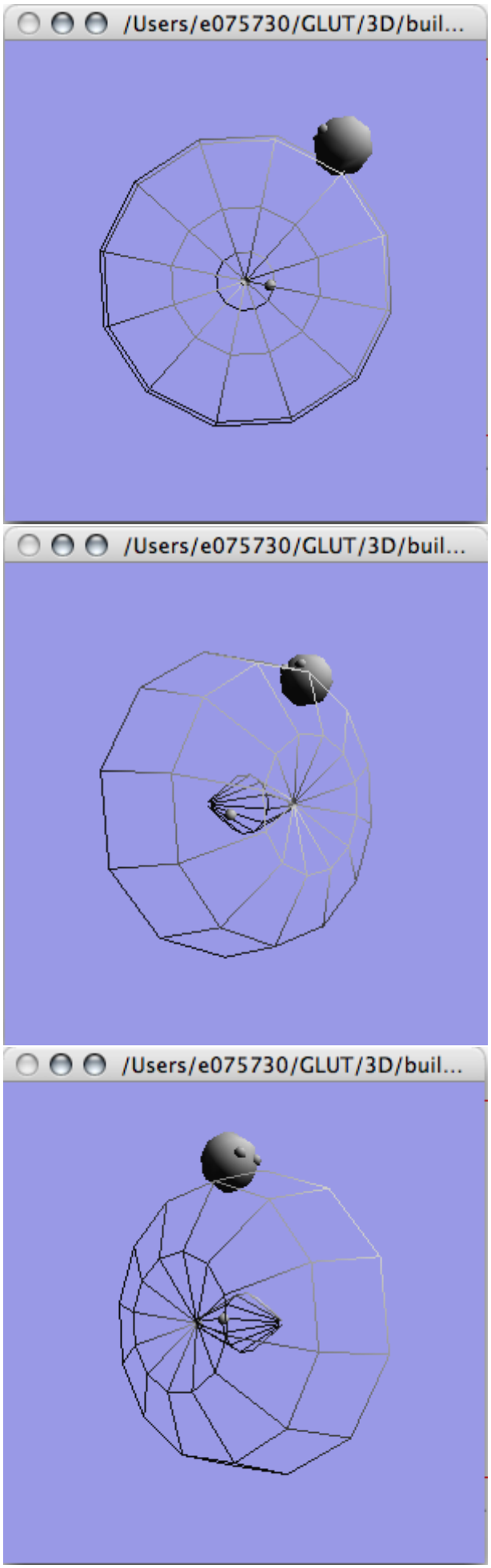

図 2: 3D の main.cpp のスナップショット

課題 3

ラインアートの頂点が z 軸方向にも動く様に改良せよ。x、y と同様に跳ね返るように実装する。

ソースコードの変更点のみ抜粋。

lines.cpp のソース

```
//POINT オブジェクトを移動する
void Point::move()
{
    pos.add(vel); //pos に vel を加算する

    if(pos.x < -1 || pos.x > 1)
        vel.x = -vel.x;
    if(pos.y < -1 || pos.y > 1)
        vel.y = -vel.y;
    if(pos.z < -1 || pos.z > 1) // 新たに z 座標の計算も入れる
        vel.z = -vel.z;
}

  -- 省略 --

//POINT オブジェクトを描画する
void Point::draw()
{
    glColor3f(col.r, col.g, col.b);
    // 3 つの引数なので 2 から 3 に変更、z 座標の位置を追加
    glVertex3f(pos.x, pos.y, pos.z);
}

  -- 省略 --

//LINES オブジェクトを生成する (num:生成する頂点数)
Lines::Lines(int num)
{
    points = new Point[num];
    num_points = num;

    int i;
    //LINES オブジェクトを初期化する
    for(i=0; i<numPoints(); ++i) {
        //各頂点を適当な位置，速度，色に設定する.
        // Vector3 に変更　つまり、引数が 3 つになるように追加
```

```
        points[i] = Point(Vector3(frand(2.0)-1, frand(2.0)-1, frand(2.0)-1)
                        ,Vector3(frand(0.001), frand(0.001), frand(0.001))
                        ,Color((i & 4) / 4, (i & 2) / 2, i & 1));
    }
}
```

lines.h のソース

```
class Point {
public:
   Point() {} // Vector2 から Vector3 に変更
   Point(Vector3 ipos, Vector3 ivel, Color icol) {
       pos = ipos; vel = ivel; col = icol; //初期位置，速度，色を設定する
   }
   void move(); //頂点を移動する
   void draw(); //頂点を描画する
private:
    Vector3 pos; //位置 Vector3
    Vector3 vel; //速度 Vector3
    Color   col; //色
}; //頂点を表すクラス
```

types.h のソース

```
class Vector3 { // 引数３つなので Vector3 に。
public:
    Vector3() {} //引数を指定しない場合は初期化しない
    // float iz ,{ z = iz; } を追加。
    Vector3(float ix, float iy, float iz) { x = ix; y = iy; z = iz; }
    void add(Vector3 rhs) { //ベクトルの加算
        x += rhs.x; y += rhs.y; z += rhs.z; // z += rhs.z; を追加
    }
    float x;
    float y;
    float z; // これを追加
};
```

課題 4.1

大砲プログラムを変更して、ウオークスルー（３次元空間を歩き回る）プログラムを作成せよ。

キーボードからの入力をイベントハンドラで受け取り、前後左右に平行移動できるようにすること（視点の向きは変わらなくてよい）。

変更部を抜粋。

main.cpp のソース

```
void keyboard(unsigned char key, int x, int y)
{
     switch(key) {
         case 'w':// 前に移動
             g_scene.move(VECTOR3(0,0,-0.1));
             break;
         case 's':// 後ろに移動
             g_scene.move(VECTOR3(0,0,0.1));
             break;
         case 'a':// 左に移動
             g_scene.move(VECTOR3(-0.1,0,0));
             break;
         case 'd':// 右に移動
             g_scene.move(VECTOR3(0.1,0,0));
             break;
         }
         g_scene.update();    // 更新
         glutPostRedisplay();
}

void idle(void)
{
    g_scene.update();
    glutPostRedisplay();

    usleep(10000);
}
```

keyboard イベントハンドラを追加し、idle から g_scene.move を取り除いた。
w キーで前に、s キーで後ろへ、a キーで左、d キーで右に移動するようになっている。

課題 4.2

このプログラムでは、ポリモーフィズムの仕組みを用いて複数種類の立体図形を同一の手続きで描画している。具体的にどの部分が該当するか示せ。

ソースの 220 行から 249 行部分の prim と、160 行から 177 行部分の drawPrim() がそれである。

```
void Cube::drawPrim()
{
    glutSolidCube(1);
}

void Sphere::drawPrim()
{
    glutSolidSphere(1,8,8);
}

void Cone::drawPrim()
{
    glPushMatrix();
    glTranslatef(0,-0.5,0);
    glRotatef(-90,1,0,0);
    glutSolidCone(1,2,8,1);
    glPopMatrix();
}

  -- 省略 --

Scene::Scene()
{
    eyePoint = VECTOR3(0,1,50);

    int i;
    for(i=0; i<NUM_SCENE_OBJ; ++i) {
        VECTOR3 pos, zero(0,0,0);

        pos = VECTOR3(frand(100)-50,0.0,frand(100)-50);

        MovableObject* prim;

        switch (rand() % 3) {
        case 0:
            prim = new Cube();
            break;
        case 1:
            prim = new Sphere();
            break;
```

```
        case 2:
            prim = new Cone();
            break;
        default:
            break;
        }
        prim->setPos(pos);
        prim->setVel(zero);
        scene_objects.addObject(prim);
    }
}
```

課題 4.3

このプログラムでは，撃てる大砲の数が限られている。その理由を考察せよ。

ソースの 90 行目から、大砲の処理について記述されているのだが、ここでポインタの配列の宣言がされている。最大のオブジェクト数は 200、フィールド上に作られているオブジェクトの数は 150 と予め与えられている。なので、大砲を 50 以上撃つと与えられた配列よりオブジェクト数が上回るため処理できなくなってしまう。故に、撃てる大砲の数が限られている。

課題 4.4

見かけ上、無制限に大砲を撃てる様にするにはどうすればよいか、解決策を述べよ。

撃った後の大砲の構造体を削除していくようにすればよい。

課題 4.5

4.2 で示した箇所にプロモーフィズムを用いることによる利点を述べよ。

カテゴリに分類してまとめることで、基本的な動作・設計部分を統一することができる。それにより、同じインターフェイスで扱えるようになる。

参考文献

[1] OpenGL 入門 http://wisdom.sakura.ne.jp/system/opengl/

[2] 11 Geometric Object Rendering http://opengl.jp/glut/section11.html