CG 基礎 課題 7

情報科学類二年 江畑 拓哉 (201611350)

1 動作環境の説明

- OS
  Manjaro Linux 17.0.6 Gellivara

- コンパイル
  g++ (GCC) 7.2.0
  Copyright (C) 2017 Free Software Foundation, Inc.

- コーディング
  Spacemacs 0.200.9 (Emacs25.3.1)

2 ソースコード

はじめに今回の課題で用いたソースコードを示す。変数 flag などを変更することでそれぞれの課題の実行が行われている。

```
#include <GL/glut.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <algorithm> // 小さい方の値を返す std::min 関数を使うため

class Vector3d {
public:
  double x, y, z;
  Vector3d() { x = y = z = 0; }
  Vector3d(double _x, double _y, double _z) { x = _x; y = _y; z = _z; }
  void set(double _x, double _y, double _z) { x = _x; y = _y; z = _z; }
```

```
13
14    // 長さを 1 に正規化する
15    void normalize() {
16      double len = length();
17      x /= len; y /= len; z /= len;
18    }
19
20    // 長さを返す
21    double length() { return sqrt(x * x + y * y + z * z); }
22
23    // s 倍する
24    void scale(const double s) { x *= s; y *= s; z *= s; }
25
26    // 加算の定義
27    Vector3d operator+(Vector3d v) { return Vector3d(x + v.x, y + v.y, z + v.z); }
28
29    // 減算の定義
30    Vector3d operator-(Vector3d v) { return Vector3d(x - v.x, y - v.y, z - v.z); }
31
32    // 内積の定義
33    double operator*(Vector3d v) { return x * v.x + y* v.y + z * v.z; }
34
35    // 外積の定義
36    Vector3d operator%(Vector3d v) { return Vector3d(y * v.z - z * v.y, z * v.x - x * v.z,
37
38    // 代入演算の定義
39    Vector3d& operator=(const Vector3d& v){ x = v.x; y = v.y; z = v.z; return (*this); }
40
41    // 加算代入の定義
42    Vector3d& operator+=(const Vector3d& v) { x += v.x; y += v.y; z += v.z; return (*this)
43
44    // 減算代入の定義
45    Vector3d& operator-=(const Vector3d& v) { x -= v.x; y -= v.y; z -= v.z; return (*this)
46
47    // 値を出力する
48    void print() { printf("Vector3d(%f %f %f)\n", x, y, z); }
49  };
50  // マイナスの符号の付いたベクトルを扱えるようにするための定義 例：b=(-a); のように記述できる
51  Vector3d operator-( const Vector3d& v ) { return( Vector3d( -v.x, -v.y, -v.z ) ); }
52
53  // ベクトルと実数の積を扱えるようにするための定義 例： c=5*a+2*b; c=b*3; のように記述できる
54  Vector3d operator*( const double& k, const Vector3d& v ) { return( Vector3d( k*v.x, k*v.y
55  Vector3d operator*( const Vector3d& v, const double& k ) { return( Vector3d( v.x*k, v.y*k
56
```

```
57  // ベクトルを実数で割る操作を扱えるようにするための定義 例： c=a/2.3; のように記述できる
58  Vector3d operator/( const Vector3d& v, const double& k ) { return( Vector3d( v.x/k, v.y/k
59
60
61  // 球体
62  class Sphere {
63  public:
64    Vector3d center; // 中心座標
65    double radius;   // 半径
66    double cR, cG, cB;  // Red, Green, Blue 値 0.0〜1.0
67
68    Sphere(double x, double y, double z, double r,
69      double cr, double cg, double cb) {
70        center.x = x;
71        center.y = y;
72        center.z = z;
73        radius = r;
74        cR = cr;
75        cG = cg;
76        cB = cb;
77    }
78
79    // 点 p を通り、v 方向の Ray との交わりを判定する。
80    // 交点が p+tv として表せる場合の t の値を返す。交わらない場合は-1 を返す
81    double getIntersec(Vector3d &p, Vector3d &v) {
82      // A*t^2 + B*t + C = 0 の形で表す
83      double A = v.x * v.x + v.y * v.y + v.z * v.z;
84      double B = 2.0 * (p.x * v.x - v.x * center.x +
85        p.y * v.y - v.y * center.y +
86        p.z * v.z- v.z * center.z);
87      double C = p.x * p.x - 2 * p.x * center.x + center.x * center.x +
88        p.y * p.y - 2 * p.y * center.y + center.y * center.y +
89        p.z * p.z - 2 * p.z * center.z + center.z * center.z -
90        radius * radius;
91      double D = B * B - 4 * A * C; // 判別式
92
93      if (D >= 0) { // 交わる
94        double t1 = (-B - sqrt(D)) / (2.0 * A);
95        double t2 = (-B + sqrt(D)) / (2.0 * A);
96        return t1 < t2 ? t1 : t2; // 小さいほうの t の値を返す
97      } else { // 交わらない
98        return -1.0;
99      }
100   }
```

```
101 };
102
103
104 int halfWidth;    // 描画領域の横幅/2
105 int halfHeight;   // 描画領域の縦幅/2
106
107 // 各種定数
108 double d = 1000;  // 視点と投影面との距離
109 double Kd = 0.8;  // 拡散反射定数
110 // double Kd = 0.6;  // 8-3-8
111 // double Kd = 1.0;  // 8-3-9
112 double Ks = 0.8;  // 鏡面反射定数
113 // double Ks = 0.6;  // 8-3-10
114 // double Ks = 1.0;  // 8-3-11
115 // double Ks = 4.0;  // 8-3-12
116 double Iin = 1.0; // 入射光の強さ
117 // double Iin = 0.5; // 8-3-6
118 // double Iin = 2.0; // 8-3-7
119 double Ia  = 0.2; // 環境光
120 // double Ia  = 0.1; // 8-3-4
121 // double Ia  = 0.4; // 8-3-5
122
123 Vector3d viewPosition(0, 0, 0); // 視点位置
124 Vector3d lightDirection(-2, -4, -2); // 入射光の進行方向
125 // Vector3d lightDirection(0, -4, -2); // 8-3-1
126 // Vector3d lightDirection(0, -4, -2); // 8-3-2
127 // Vector3d lightDirection(-2, 0, -2); // 8-3-3
128
129 // レンダリングする球体
130 Sphere sphere(0.0, 0.0, -1500, // 中心座標
131         150.0,           // 半径
132         0.2, 0.9, 0.9);  // RGB値
133
134
135 // 描画を行う
136 void display(void) {
137
138   glClear(GL_COLOR_BUFFER_BIT); // 描画内容のクリア
139
140   // ピクセル単位で描画色を決定するループ処理
141   for(int y = (-halfHeight); y <= halfHeight; y++ ) {
142     for(int x = (-halfWidth); x <= halfWidth; x++ ) {
143
144       Vector3d ray(x - viewPosition.x, y - viewPosition.y, -d - viewPosition.z); // 原点か
```

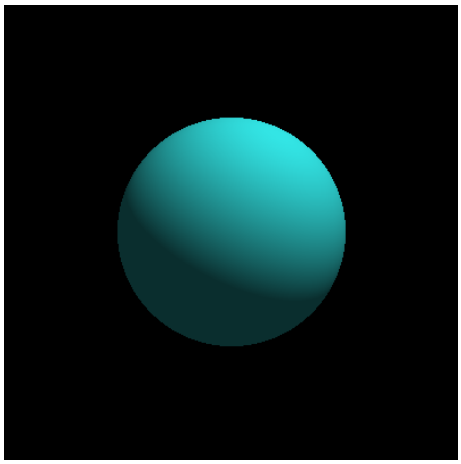
```
145         ray.normalize(); // レイの長さの正規化
146 
147         // レイを飛ばして球との交点を求める
148         double t = sphere.getIntersec(viewPosition, ray);
149 
150         if(t > 0) { // 交点がある
151           double Is = 0; // 鏡面反射光
152           double Id = 0; // 拡散反射光
153           // -------------------------------------------------------------------------------
154           // ★ここで Is および Id の値を計算する
155           int flag = 1; // 0: 拡散反射光 <=> 1: 鏡面反射光
156           Vector3d P = viewPosition + t * ray;
157           Vector3d N = P - sphere.center;
158           N.normalize();
159           if (flag >= 0) {
160             double cos_Id = N * (-1 * lightDirection);
161             if(cos_Id > 0) {
162               Id = Iin * Kd * cos_Id;
163             }
164           } if(flag >= 1) {
165             int n = 20;
166             // int n = 2; // 8-3-13
167             // int n = 40; // 8-3-14
168             double a = -1 * (lightDirection * N);
169             Vector3d R = lightDirection + 2 * a * N;
170             Vector3d V = P - viewPosition;
171             R.normalize();
172             V.normalize();
173             double cos_Is = -1 * R * V;
174             if (cos_Is > 0) {
175               Is = Iin * Ks * pow(cos_Is, n);
176             }
177           }
178           // -------------------------------------------------------------------------------
179           double I = Id + Is + Ia;
180           double r = std::min(I * sphere.cR, 1.0); // 1.0 を超えないようにする
181           double g = std::min(I * sphere.cG, 1.0); // 1.0 を超えないようにする
182           double b = std::min(I * sphere.cB, 1.0); // 1.0 を超えないようにする
183 
184           // 描画色の設定
185           glColor3d(r, g, b);
186 
187         } else { // 交点が無い
188 
```

```
189           // 描画色を黒にする
190           glColor3f(0.0f, 0.0f, 0.0f);
191         }
192
193         // (x, y) の画素を描画
194         glBegin(GL_POINTS);
195         glVertex2i(x, y);
196         glEnd();
197       }
198     }
199     glFlush();
200   }
201
202   void resize(int w, int h) {
203     if (h < 1) return;
204     glViewport(0, 0, w, h);
205     halfWidth = w/2;
206     halfHeight = h/2;
207     glMatrixMode(GL_PROJECTION);
208     glLoadIdentity();
209
210     // ウィンドウ内の座標系設定
211     glOrtho( -halfWidth, halfWidth, -halfHeight, halfHeight, 0.0, 1.0);
212     glMatrixMode(GL_MODELVIEW);
213   }
214
215   void keyboard(unsigned char key, int x, int y) {
216     switch (key) {
217       case 27: exit(0);  /* ESC code */
218     }
219     glutPostRedisplay();
220   }
221
222   int main(int argc, char** argv) {
223     lightDirection.normalize();
224
225     glutInit(&argc, argv);
226     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
227     glutInitWindowSize(400,400);
228     glutCreateWindow(argv[0]);
229     glClearColor(1.0, 1.0, 1.0, 1.0);
230     glShadeModel(GL_FLAT);
231
232     glutDisplayFunc(display);
```

```
233     glutReshapeFunc(resize);
234     glutKeyboardFunc(keyboard);
235     glutMainLoop();
236 
237     return 0;
238 
239 }
```

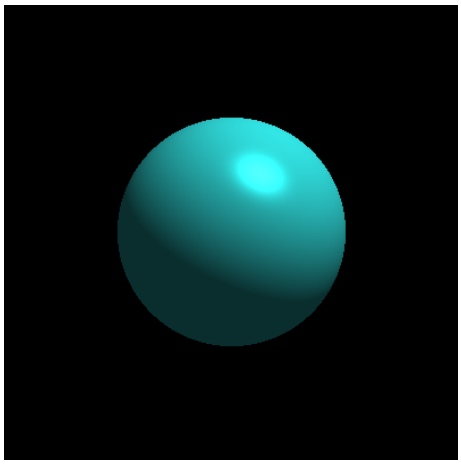
3 課題 1

153 行目の変数 flag を 0 にすることで実行可能である。



4 課題 2

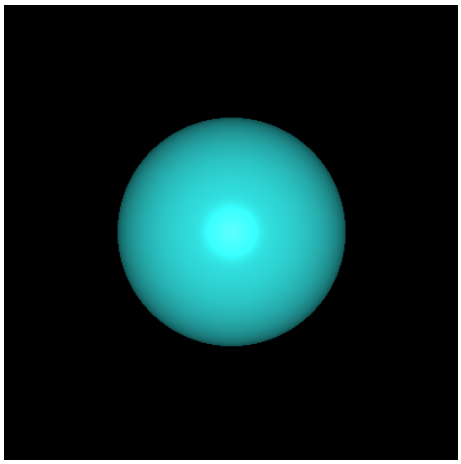
153 行目の変数 flag を 1 にすることで実行可能である。



5 課題 3

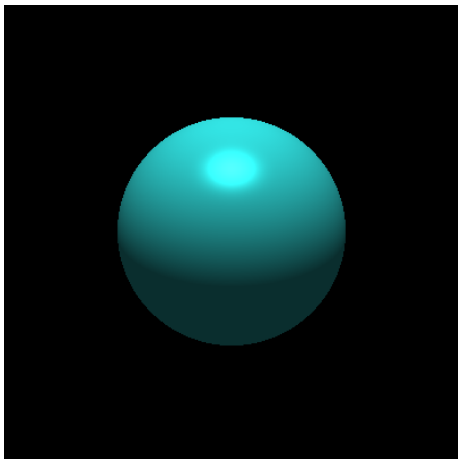
5.1 1

125 行目 `Vector3d lightDirection(0, 0, -2); // 8-3-1` を有効にした場合、入射光が手前からのものになった。



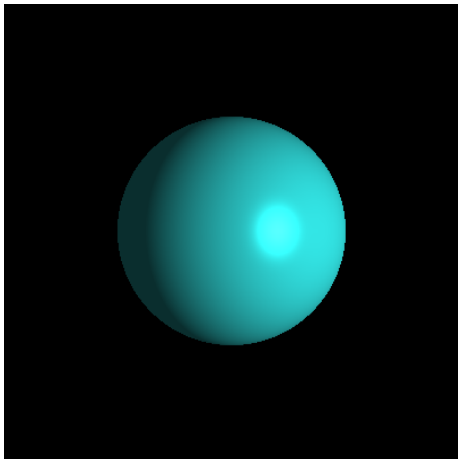
5.2 2

126 行目 `Vector3d lightDirection(0, -4, -2); // 8-3-2` を有効にした場合、入射光が上からのものになった。



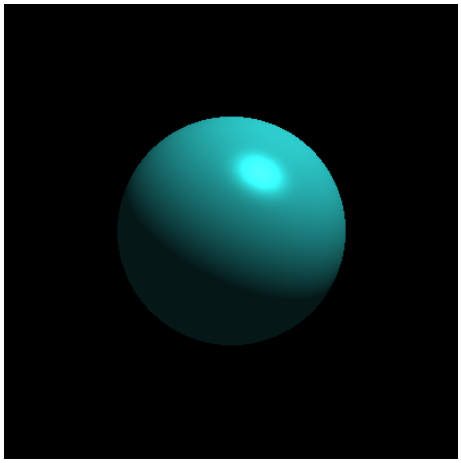
5.3 3

127 行目 `Vector3d lightDirection(-2, 0, -2); // 8-3-3` を有効にした場合、入射光が右からのものになった。



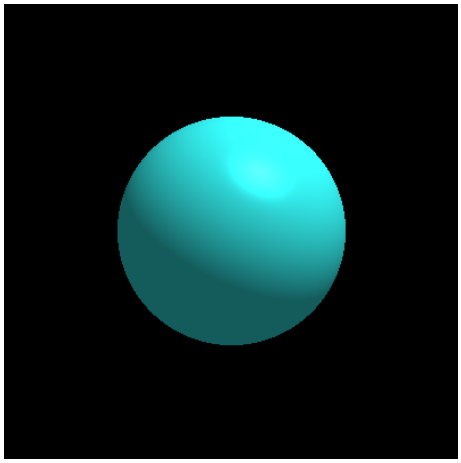
5.4 4

120 行目 `double Ia = 0.1; // 8-3-4` を有効にした場合、全体的に暗くなった。



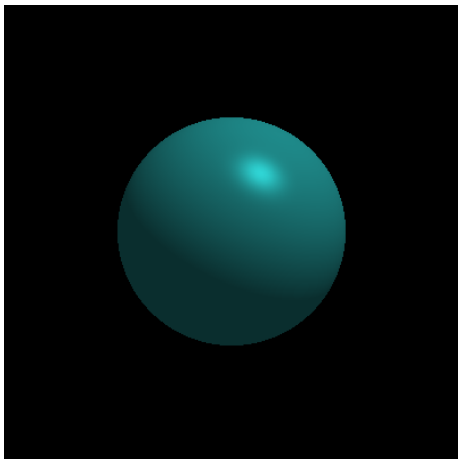
5.5 5

121 行目 `double Ia = 0.4; // 8-3-5` を有効にした場合、全体的に明るくなった。



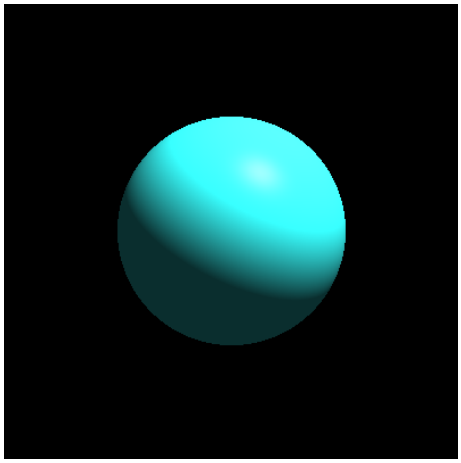
5.6 6

117 行目 `double Iin = 0.5; // 8-3-6` を有効にした場合、全体的に暗くなった。(入射光が暗くなったが、影も当然暗いため全体的に暗く感じた)



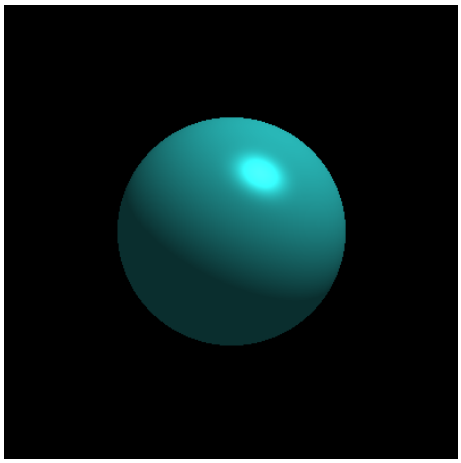
5.7 7

118 行目 `double Iin = 2.0; // 8-3-6` を有効にした場合、入射光が強くなった。



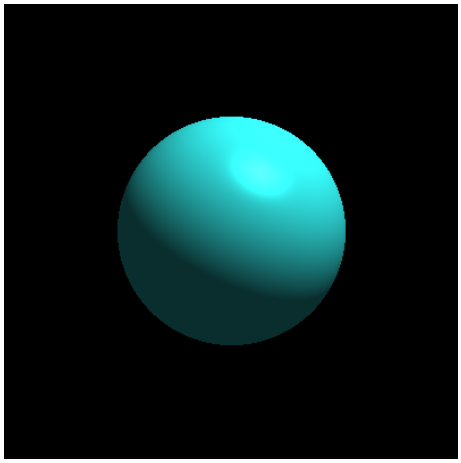
5.8 8

110 行目 `double Kd = 0.6; // 8-3-8` を有効にした場合、鏡面部分を除いて暗くなった。



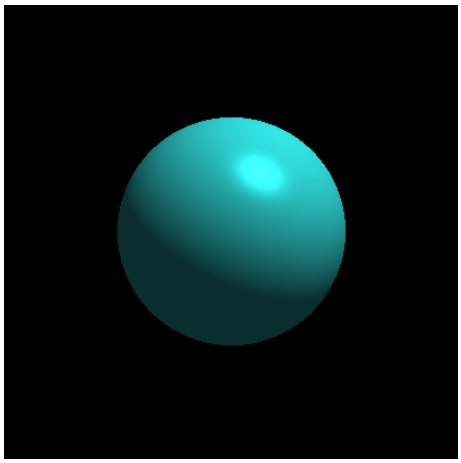
5.9 9

111 行目 `double Kd = 1.0; // 8-3-9` を有効にした場合、鏡面部分を除いて明るくなった。



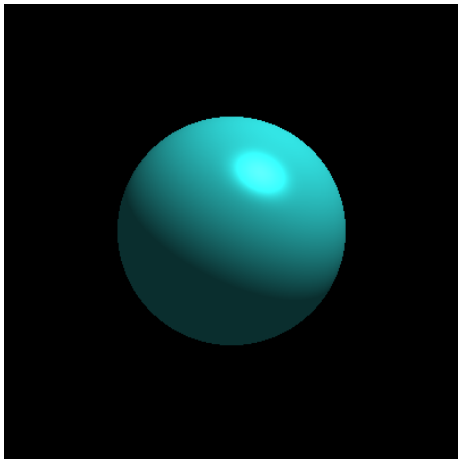
5.10 10

113 行目 `double Ks = 0.6; // 8-3-10` を有効にした場合、鏡面部分がほんのり暗くなった。



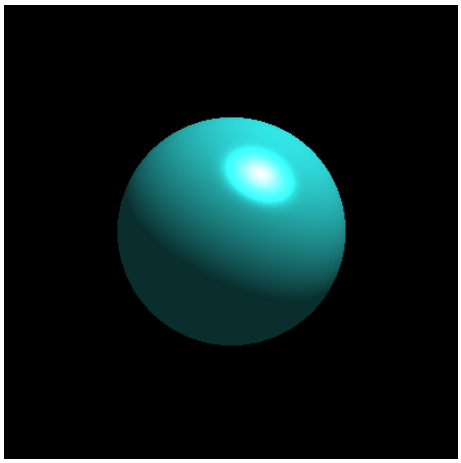
5.11 11

114 行目 `double Ks = 1.0; // 8-3-11` を有効にした場合、鏡面部分がほんのり明るくなった。



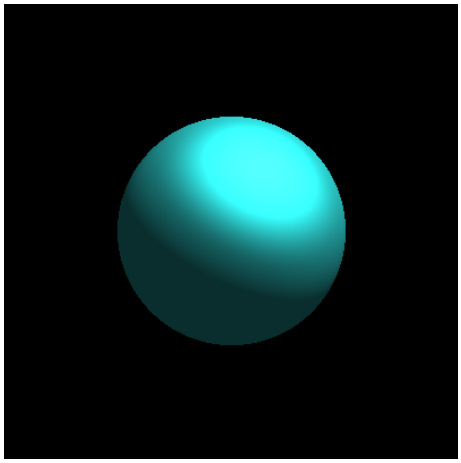
5.12 12

上の変化が少なかったため、追加した。115 行目 `double Ks = 4.0; // 8-3-12` を有効にした場合、鏡面部分がはっきり明るくなった。



5.13 13

166 行目 `int n = 2; // 8-3-13` を有効にした場合、鏡面部分が潰れてぼやけて見えた。



5.14 14

167 行目 `int n = 40; // 8-3-14` を有効にした場合、鏡面部分が鋭くはっきりと見えた。