CG基礎 課題9

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.

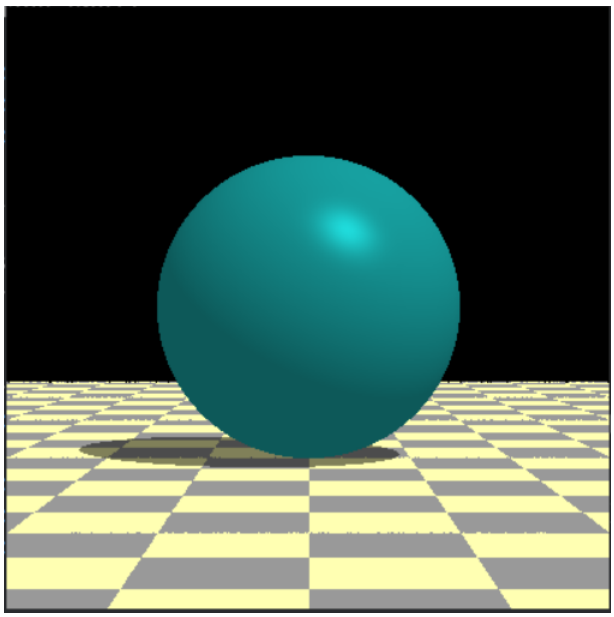
- コーディング
  Spacemacs 0.200.9 (Emacs25.3.1)

2 課題1〜3

1. 球と床板をレンダリングしなさい。

2. 床に格子模様を付加しなさい。

3. 床に球体の影が表示されるようにしなさい。



2.1 ソースコード

編集部分はコメントで囲ってある。

```
#include <GL/glut.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <algorithm>

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; }
  void normalize() {
    double len = length();
    x /= len; y /= len; z /= len;
  }
  double length() { return sqrt(x * x + y * y + z * z); }
  void scale(const double s) { x *= s; y *= s; z *= s; }
  Vector3d operator+(Vector3d v) { return Vector3d(x + v.x, y + v.y, z + v.z); }
  Vector3d operator-(Vector3d v) { return Vector3d(x - v.x, y - v.y, z - v.z); }
  double operator*(Vector3d v) { return x * v.x + y* v.y + z * v.z; }
```

```
22    Vector3d operator%(Vector3d v) { return Vector3d(y * v.z - z * v.y, z * v.x - x * v.z,
23    Vector3d& operator=(const Vector3d& v){ x = v.x; y = v.y; z = v.z; return (*this); }
24    Vector3d& operator+=(const Vector3d& v) { x += v.x; y += v.y; z += v.z; return (*this)
25    Vector3d& operator-=(const Vector3d& v) { x -= v.x; y -= v.y; z -= v.z; return (*this)
26    void print() { printf("Vector3d(%f %f %f)\n", x, y, z); }
27  };
28  Vector3d operator-( const Vector3d& v ) { return( Vector3d( -v.x, -v.y, -v.z ) ); }
29  Vector3d operator*( const double& k, const Vector3d& v ) { return( Vector3d( k*v.x, k*v.y
30  Vector3d operator*( const Vector3d& v, const double& k ) { return( Vector3d( v.x*k, v.y*k
31  Vector3d operator/( const Vector3d& v, const double& k ) { return( Vector3d( v.x/k, v.y/k
32
33
34  // 球体
35  class Sphere {
36  public:
37    Vector3d center; // 中心座標
38    double radius;   // 半径
39    double cR, cG, cB;  // Red, Green, Blue 値 0.0〜1.0
40
41    Sphere(double x, double y, double z, double r,
42      double cr, double cg, double cb) {
43        center.x = x;
44        center.y = y;
45        center.z = z;
46        radius = r;
47        cR = cr;
48        cG = cg;
49        cB = cb;
50    }
51
52    // 点 p を通り、v 方向の Ray との交わりを判定する。
53    // 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
54    double getIntersec(Vector3d &p, Vector3d &v) {
55      // A*t^2 + B*t + C = 0 の形で表す
56      double A = v.x * v.x + v.y * v.y + v.z * v.z;
57      double B = 2.0 * (p.x * v.x - v.x * center.x +
58        p.y * v.y - v.y * center.y +
59        p.z * v.z- v.z * center.z);
60      double C = p.x * p.x - 2 * p.x * center.x + center.x * center.x +
61        p.y * p.y - 2 * p.y * center.y + center.y * center.y +
62        p.z * p.z - 2 * p.z * center.z + center.z * center.z -
63        radius * radius;
64      double D = B * B - 4 * A * C; // 判別式
65
```

```
66      if (D >= 0) { // 交わる
67        double t1 = (-B - sqrt(D)) / (2.0 * A);
68        double t2 = (-B + sqrt(D)) / (2.0 * A);
69        return t1 < t2 ? t1 : t2; // 小さいほうの t の値を返す
70      } else { // 交わらない
71        return -1.0;
72      }
73    }
74  };
75
76  // 板。xz 平面に平行な面とする
77  class Board {
78  public:
79    double y; // y 座標値
80
81    Board(double _y) {
82      y = _y;
83    }
84
85    // 点 p を通り、v 方向の Ray との交わりを判定する。
86    // 交点が p+tv として表せる場合の t の値を返す。交わらない場合は負の値を返す
87    double getIntersec(Vector3d &p, Vector3d &v) {
88      if(fabs(v.y) < 1.0e-10) return -1; // 水平な Ray は交わらない
89
90      double t = -1;
91      // ★ここで t の値を計算する
92      // ★ただし z 座標が-3000 より小さいなら交わらないものとする
93      // ------------------------------------------------------------------------
94      t = (y + p.y) / v.y;
95      if ((t < 0) || ((p + t * v).z < -3000)) {
96        return -1;
97      }
98      // ------------------------------------------------------------------------
99      return t;
100   }
101
102   // x と z の値から床の色を返す（格子模様になるように）
103   Vector3d getColorVec(double x, double z) {
104     // ★ x, z の値によって (1.0, 1.0, 0.7) または (0.6, 0.6, 0.6) のどちらかの色を返すようにす
105     int flag = 1;
106     if (x > 0) {
107       if (((int)x) % 200 > 100) flag *= -1;
108     } else {
109       if (((int) (-1 * x)) % 200 < 100) flag *= -1;
```

```
110       }
111       if (z > 0) {
112         if (((int)z) % 200 > 100) flag *= -1;
113       } else {
114         if (((int) (-1 * z)) % 200 < 100) flag *= -1;
115       }
116       if (flag > 0) {
117         return Vector3d(0.6, 0.6, 0.6);
118       } else {
119         return Vector3d(1.0, 1.0, 0.7);
120       }
121     }
122   };
123
124   int halfWidth;    // 描画領域の横幅/2
125   int halfHeight;   // 描画領域の縦幅/2
126   double screen_z = -1000;  // 投影面の z 座標
127   double Kd = 0.8;  // 拡散反射定数
128   double Ks = 0.8;  // 鏡面反射定数
129   double Iin = 0.5; // 入射光の強さ
130   double Ia  = 0.5; // 環境光
131   Vector3d viewPosition(0, 0, 0); // 原点=視点
132   Vector3d lightDirection(-2, -4, -2); // 入射光の進行方向
133
134
135   // レンダリングする球体
136   Sphere sphere(0.0, 0.0, -1500, // 中心座標
137           150.0,           // 半径
138           0.1, 0.7, 0.7);  // RGB 値
139
140   // 球体の置かれている床
141   Board board(-150); // y 座標値を -150 にする。(球と接するようにする)
142
143   // 2つのベクトルの成す角の cos 値を計算する
144   double getCos(Vector3d &v0, Vector3d &v1) {
145     return (v0.x * v1.x + v0.y * v1.y + v0.z * v1.z ) / (v0.length() * v1.length());
146   }
147
148   // x, y で指定されたスクリーン座標での色 (RGB) を colorVec の xyz の値に格納する
149   void getPixelColor(double x, double y, Vector3d &colorVec) {
150     // 原点からスクリーン上のピクセルへ飛ばすレイの方向
151     Vector3d ray(x - viewPosition.x, y - viewPosition.y, screen_z - viewPosition.z);
152     ray.normalize(); // レイの長さの正規化
153     // レイを飛ばして球と交差するか求める
```

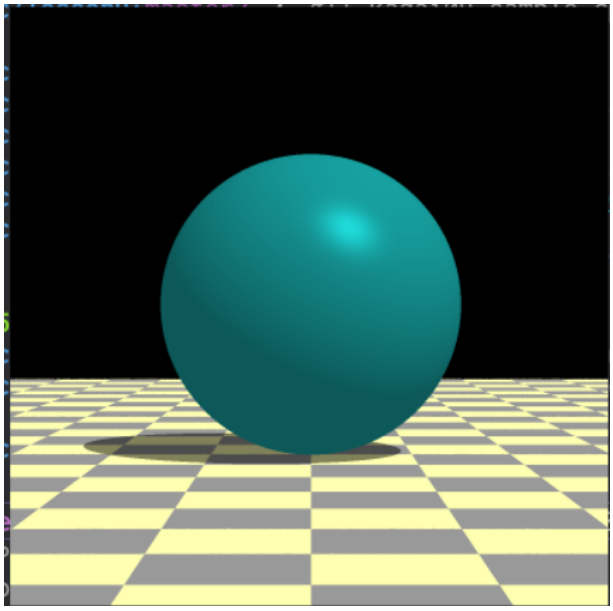
```
154     double t_sphere = sphere.getIntersec(viewPosition, ray);
155     if(t_sphere > 0) { // 球との交点がある
156       // ★前回の課題を参考に、球体の表面の色を計算で求め、colorVec に設定する
157       double r = 0;
158       double g = 0;
159       double b = 0;
160       // ------------------------------------------------------------------------------
161       double Id, Is, cos_Id, cos_Is, a, I;
162       Is = Id = I = 0;
163       Vector3d P = viewPosition + t_sphere * ray;
164       Vector3d N = P - sphere.center;
165       N.normalize();
166       cos_Id = N * (-1 * lightDirection);
167       if (cos_Id > 0) {
168         Id = Iin * Kd * cos_Id;
169       }
170       int n = 20;
171       a = -1 * (lightDirection * N);
172       Vector3d R = lightDirection + 2 * a * N;
173       Vector3d V = P - viewPosition;
174       R.normalize();
175       V.normalize();
176       cos_Is = -1 * R * V;
177       if (cos_Is > 0) {
178         Is = Iin * Ks * pow(cos_Is, n);
179       }
180       I = Id + Is + Ia;
181       r = std::min(I * sphere.cR, 1.0);
182       g = std::min(I * sphere.cG, 1.0);
183       b = std::min(I * sphere.cB, 1.0);
184       // ------------------------------------------------------------------------------
185       colorVec.set(r, g, b);
186       return;
187     }
188
189     // レイを飛ばして床と交差するか求める
190     double t_board = board.getIntersec(viewPosition, ray);
191
192     if(t_board > 0) { // 床との交点がある
193       // ★床の表面の色を設定する
194       // ★球の影になる場合は、RGB の値をそれぞれ 0.5 倍する
195       double r = 0;
196       double g = 0;
197       double b = 0;
```

```
198       // ------------------------------------------------------------------------------
199       double x, z;
200       Vector3d board_pos = viewPosition + t_board * ray;
201       Vector3d colors;
202       x = board_pos.x;
203       z = board_pos.z;
204       colors = board.getColorVec(x, z);
205       r = colors.x;
206       g = colors.y;
207       b = colors.z;
208       if (sphere.getIntersec(board_pos, lightDirection) != -1.0) {
209         r *= 0.5;
210         g *= 0.5;
211         b *= 0.5;
212       }
213       // ------------------------------------------------------------------------------
214       colorVec.set(r, g, b);
215       return;
216     }
217
218     // 何とも交差しない
219     colorVec.set(0, 0, 0); // 背景色（黒）を設定する
220   }
221
222   // 描画を行う
223   void display(void) {
224
225     glClear(GL_COLOR_BUFFER_BIT); // 描画内容のクリア
226
227     // ピクセル単位で描画色を決定するループ処理
228     for(int y = (-halfHeight); y <= halfHeight; y++ ) {
229       for(int x = (-halfWidth); x <= halfWidth; x++ ) {
230
231         Vector3d colorVec;
232
233         // x, y 座標の色を取得する
234         getPixelColor(x, y , colorVec);
235
236         //取得した色で、描画色を設定する
237         glColor3d(colorVec.x, colorVec.y, colorVec.z);
238
239         // (x, y) の画素を描画
240         glBegin(GL_POINTS);
241         glVertex2i( x, y );
```

```
242         glEnd();
243       }
244     }
245     glFlush();
246 }
247
248 void resizeWindow(int w, int h) {
249     h = (h == 0) ? 1 : h;
250     glViewport(0, 0, w, h);
251     halfWidth = w/2;
252     halfHeight = h/2;
253     glMatrixMode(GL_PROJECTION);
254     glLoadIdentity();
255
256     // ウィンドウ内の座標系設定
257     glOrtho( -halfWidth, halfWidth, -halfHeight, halfHeight, 0.0, 1.0);
258     glMatrixMode(GL_MODELVIEW);
259 }
260
261 void keyboard(unsigned char key, int x, int y) {
262     switch (key) {
263       case 27: exit(0);  /* ESC code */
264     }
265     glutPostRedisplay();
266 }
267
268 int main(int argc, char** argv) {
269     lightDirection.normalize();
270
271     glutInit(&argc, argv);
272     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
273     glutInitWindowSize(400,400);
274     glutInitWindowPosition(180,10);
275     glutCreateWindow(argv[0]);
276     glClearColor(1.0, 1.0, 1.0, 1.0);
277     glShadeModel(GL_FLAT);
278
279     glutDisplayFunc(display);
280     glutReshapeFunc(resizeWindow);
281     glutKeyboardFunc(keyboard);
282     glutMainLoop();
283
284     return 0;
```

3 課題 4

1. 鏡面を滑らかに表現するアンチエイリアシングを行うことを考える。1 ピクセルに 3 * 3 に 9 分割してサンプリングを行い、その 9 つの値の平均値でピクセルの色を決定しなさい。



3.1 ソースコード

```
#include <GL/glut.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <algorithm>

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; }
  void normalize() {
    double len = length();
    x /= len; y /= len; z /= len;
  }
  double length() { return sqrt(x * x + y * y + z * z); }
```

```
18    void scale(const double s) { x *= s; y *= s; z *= s; }
19    Vector3d operator+(Vector3d v) { return Vector3d(x + v.x, y + v.y, z + v.z); }
20    Vector3d operator-(Vector3d v) { return Vector3d(x - v.x, y - v.y, z - v.z); }
21    double operator*(Vector3d v) { return x * v.x + y* v.y + z * v.z; }
22    Vector3d operator%(Vector3d v) { return Vector3d(y * v.z - z * v.y, z * v.x - x * v.z,
23    Vector3d& operator=(const Vector3d& v){ x = v.x; y = v.y; z = v.z; return (*this); }
24    Vector3d& operator+=(const Vector3d& v) { x += v.x; y += v.y; z += v.z; return (*this)
25    Vector3d& operator-=(const Vector3d& v) { x -= v.x; y -= v.y; z -= v.z; return (*this)
26    void print() { printf("Vector3d(%f %f %f)\n", x, y, z); }
27  };
28  Vector3d operator-( const Vector3d& v ) { return( Vector3d( -v.x, -v.y, -v.z ) ); }
29  Vector3d operator*( const double& k, const Vector3d& v ) { return( Vector3d( k*v.x, k*v.y
30  Vector3d operator*( const Vector3d& v, const double& k ) { return( Vector3d( v.x*k, v.y*k
31  Vector3d operator/( const Vector3d& v, const double& k ) { return( Vector3d( v.x/k, v.y/k
32
33
34  // 球体
35  class Sphere {
36  public:
37    Vector3d center; // 中心座標
38    double radius;   // 半径
39    double cR, cG, cB;  // Red, Green, Blue 値 0.0〜1.0
40
41    Sphere(double x, double y, double z, double r,
42      double cr, double cg, double cb) {
43        center.x = x;
44        center.y = y;
45        center.z = z;
46        radius = r;
47        cR = cr;
48        cG = cg;
49        cB = cb;
50    }
51
52    // 点 p を通り、v 方向の Ray との交わりを判定する。
53    // 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
54    double getIntersec(Vector3d &p, Vector3d &v) {
55      // A*t^2 + B*t + C = 0 の形で表す
56      double A = v.x * v.x + v.y * v.y + v.z * v.z;
57      double B = 2.0 * (p.x * v.x - v.x * center.x +
58        p.y * v.y - v.y * center.y +
59        p.z * v.z- v.z * center.z);
60      double C = p.x * p.x - 2 * p.x * center.x + center.x * center.x +
61        p.y * p.y - 2 * p.y * center.y + center.y * center.y +
```

```
62        p.z * p.z - 2 * p.z * center.z + center.z * center.z -
63        radius * radius;
64      double D = B * B - 4 * A * C; // 判別式
65
66      if (D >= 0) { // 交わる
67        double t1 = (-B - sqrt(D)) / (2.0 * A);
68        double t2 = (-B + sqrt(D)) / (2.0 * A);
69        return t1 < t2 ? t1 : t2; // 小さいほうの t の値を返す
70      } else { // 交わらない
71        return -1.0;
72      }
73    }
74  };
75
76  // 板。xz 平面に平行な面とする
77  class Board {
78  public:
79    double y; // y 座標値
80
81    Board(double _y) {
82      y = _y;
83    }
84
85    // 点 p を通り、v 方向の Ray との交わりを判定する。
86    // 交点が p+tv として表せる場合の t の値を返す。交わらない場合は負の値を返す
87    double getIntersec(Vector3d &p, Vector3d &v) {
88      if(fabs(v.y) < 1.0e-10) return -1; // 水平な Ray は交わらない
89
90      double t = -1;
91      // ★ここで t の値を計算する
92      // ★ただし z 座標が-3000 より小さいなら交わらないものとする
93      // ------------------------------------------------------------------------
94      t = (y + p.y) / v.y;
95      if ((t < 0) || ((p + t * v).z < -3000)) {
96        return -1;
97      }
98      // ------------------------------------------------------------------------
99      return t;
100   }
101
102   // x と z の値から床の色を返す（格子模様になるように）
103   Vector3d getColorVec(double x, double z) {
104     // ★ x, z の値によって (1.0, 1.0, 0.7) または (0.6, 0.6, 0.6) のどちらかの色を返すようにす
105     int flag = 1;
```

```
106       if (x > 0) {
107         if (((int)x) % 200 > 100) flag *= -1;
108       } else {
109         if (((int) (-1 * x)) % 200 < 100) flag *= -1;
110       }
111       if (z > 0) {
112         if (((int)z) % 200 > 100) flag *= -1;
113       } else {
114         if (((int) (-1 * z)) % 200 < 100) flag *= -1;
115       }
116       if (flag > 0) {
117         return Vector3d(0.6, 0.6, 0.6);
118       } else {
119         return Vector3d(1.0, 1.0, 0.7);
120       }
121     }
122   };
123
124   int halfWidth;     // 描画領域の横幅/2
125   int halfHeight;    // 描画領域の縦幅/2
126   double screen_z = -1000;  // 投影面の z 座標
127   double Kd = 0.8;  // 拡散反射定数
128   double Ks = 0.8;  // 鏡面反射定数
129   double Iin = 0.5; // 入射光の強さ
130   double Ia  = 0.5; // 環境光
131   Vector3d viewPosition(0, 0, 0); // 原点=視点
132   Vector3d lightDirection(-2, -4, -2); // 入射光の進行方向
133
134
135   // レンダリングする球体
136   Sphere sphere(0.0, 0.0, -1500, // 中心座標
137           150.0,           // 半径
138           0.1, 0.7, 0.7);  // RGB 値
139
140   // 球体の置かれている床
141   Board board(-150); // y 座標値を -150 にする。(球と接するようにする)
142
143   // 2つのベクトルの成す角の cos 値を計算する
144   double getCos(Vector3d &v0, Vector3d &v1) {
145     return (v0.x * v1.x + v0.y * v1.y + v0.z * v1.z ) / (v0.length() * v1.length());
146   }
147
148   // x, y で指定されたスクリーン座標での色 (RGB) を colorVec の xyz の値に格納する
149   void getPixelColor(double x, double y, Vector3d &colorVec) {
```

```
150     // 原点からスクリーン上のピクセルへ飛ばすレイの方向
151     Vector3d ray(x - viewPosition.x, y - viewPosition.y, screen_z - viewPosition.z);
152
153     ray.normalize(); // レイの長さの正規化
154
155     // レイを飛ばして球と交差するか求める
156     double t_sphere = sphere.getIntersec(viewPosition, ray);
157     if(t_sphere > 0) { // 球との交点がある
158       // ★前回の課題を参考に、球体の表面の色を計算で求め、colorVecに設定する
159       double r = 0;
160       double g = 0;
161       double b = 0;
162       // ------------------------------------------------------------------------------
163       double Id, Is, cos_Id, cos_Is, a, I;
164       Is = Id = I = 0;
165       Vector3d P = viewPosition + t_sphere * ray;
166       Vector3d N = P - sphere.center;
167       N.normalize();
168       cos_Id = N * (-1 * lightDirection);
169       if (cos_Id > 0) {
170         Id = Iin * Kd * cos_Id;
171       }
172       int n = 20;
173       a = -1 * (lightDirection * N);
174       Vector3d R = lightDirection + 2 * a * N;
175       Vector3d V = P - viewPosition;
176       R.normalize();
177       V.normalize();
178       cos_Is = -1 * R * V;
179       if (cos_Is > 0) {
180         Is = Iin * Ks * pow(cos_Is, n);
181       }
182       I = Id + Is + Ia;
183       r = std::min(I * sphere.cR, 1.0);
184       g = std::min(I * sphere.cG, 1.0);
185       b = std::min(I * sphere.cB, 1.0);
186       // ------------------------------------------------------------------------------
187       colorVec.set(r, g, b);
188       return;
189     }
190
191     // レイを飛ばして床と交差するか求める
192     double t_board = board.getIntersec(viewPosition, ray);
193
```

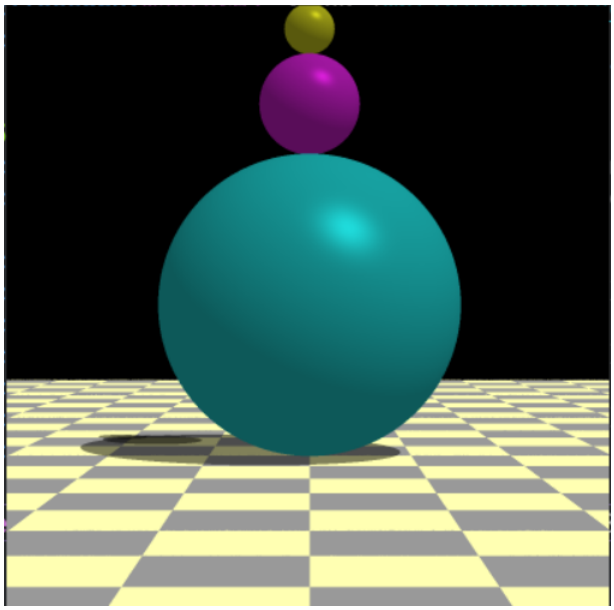
```
194   if(t_board > 0) { // 床との交点がある
195     // ★床の表面の色を設定する
196     // ★球の影になる場合は、RGB の値をそれぞれ 0.5 倍する
197     double r = 0;
198     double g = 0;
199     double b = 0;
200     // ----------------------------------------------------------------------------
201     double x, z;
202     Vector3d board_pos = viewPosition + t_board * ray;
203     Vector3d colors;
204     x = board_pos.x;
205     z = board_pos.z;
206     colors = board.getColorVec(x, z);
207     r = colors.x;
208     g = colors.y;
209     b = colors.z;
210     if (sphere.getIntersec(board_pos, lightDirection) != -1.0) {
211       r *= 0.5;
212       g *= 0.5;
213       b *= 0.5;
214     }
215     // ----------------------------------------------------------------------------
216     colorVec.set(r, g, b);
217     return;
218   }
219
220   // 何とも交差しない
221   colorVec.set(0, 0, 0); // 背景色（黒）を設定する
222 }
223
224 // 描画を行う
225 void display(void) {
226
227   glClear(GL_COLOR_BUFFER_BIT); // 描画内容のクリア
228
229   // ピクセル単位で描画色を決定するループ処理
230   for(int y = (-halfHeight); y <= halfHeight; y++ ) {
231     for(int x = (-halfWidth); x <= halfWidth; x++ ) {
232       // ----------------------------------------------------------------------------
233       int i, j;
234       Vector3d colorVecs[9];
235       Vector3d colorVec;
236       for (i = 1; i < 4; ++i) {
237         for (j = 1; j < 4; ++j) {
```

```
238             getPixelColor(x - i / 3.0, y - j / 3.0, colorVecs[(i - 1) * 3 + (j - 1)]);
239           }
240         }
241         for (i = 0; i < 9; ++i) {
242           colorVec += colorVecs[i];
243         }
244         //取得した色で、描画色を設定する
245         glColor3d(colorVec.x / 9.0, colorVec.y / 9.0, colorVec.z / 9.0);
246         // ------------------------------------------------------------------------------
247 
248         // (x, y) の画素を描画
249         glBegin(GL_POINTS);
250         glVertex2i( x, y );
251         glEnd();
252       }
253     }
254     glFlush();
255   }
256 
257   void resizeWindow(int w, int h) {
258     h = (h == 0) ? 1 : h;
259     glViewport(0, 0, w, h);
260     halfWidth = w/2;
261     halfHeight = h/2;
262     glMatrixMode(GL_PROJECTION);
263     glLoadIdentity();
264 
265     // ウィンドウ内の座標系設定
266     glOrtho( -halfWidth, halfWidth, -halfHeight, halfHeight, 0.0, 1.0);
267     glMatrixMode(GL_MODELVIEW);
268   }
269 
270   void keyboard(unsigned char key, int x, int y) {
271     switch (key) {
272       case 27: exit(0);  /* ESC code */
273     }
274     glutPostRedisplay();
275   }
276 
277   int main(int argc, char** argv) {
278     lightDirection.normalize();
279 
280     glutInit(&argc, argv);
281     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
```

```
282    glutInitWindowSize(400,400);
283    glutInitWindowPosition(180,10);
284    glutCreateWindow(argv[0]);
285    glClearColor(1.0, 1.0, 1.0, 1.0);
286    glShadeModel(GL_FLAT);
287
288    glutDisplayFunc(display);
289    glutReshapeFunc(resizeWindow);
290    glutKeyboardFunc(keyboard);
291    glutMainLoop();
292
293    return 0;
294  }
```

4 発展課題 1

1. 自由にプログラムコードを変更してサンプル以外の結果を出力する。



4.1 ソースコード

```
1  #include <GL/glut.h>
2  #include <math.h>
3  #include <stdlib.h>
4  #include <stdio.h>
```

```
#include <algorithm>

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; }
  void normalize() {
    double len = length();
    x /= len; y /= len; z /= len;
  }
  double length() { return sqrt(x * x + y * y + z * z); }
  void scale(const double s) { x *= s; y *= s; z *= s; }
  Vector3d operator+(Vector3d v) { return Vector3d(x + v.x, y + v.y, z + v.z); }
  Vector3d operator-(Vector3d v) { return Vector3d(x - v.x, y - v.y, z - v.z); }
  double operator*(Vector3d v) { return x * v.x + y* v.y + z * v.z; }
  Vector3d operator%(Vector3d v) { return Vector3d(y * v.z - z * v.y, z * v.x - x * v.z,
  Vector3d& operator=(const Vector3d& v){ x = v.x; y = v.y; z = v.z; return (*this); }
  Vector3d& operator+=(const Vector3d& v) { x += v.x; y += v.y; z += v.z; return (*this)
  Vector3d& operator-=(const Vector3d& v) { x -= v.x; y -= v.y; z -= v.z; return (*this)
  void print() { printf("Vector3d(%f %f %f)\n", x, y, z); }
};
Vector3d operator-( const Vector3d& v ) { return( Vector3d( -v.x, -v.y, -v.z ) ); }
Vector3d operator*( const double& k, const Vector3d& v ) { return( Vector3d( k*v.x, k*v.y
Vector3d operator*( const Vector3d& v, const double& k ) { return( Vector3d( v.x*k, v.y*k
Vector3d operator/( const Vector3d& v, const double& k ) { return( Vector3d( v.x/k, v.y/k


// 球体
class Sphere {
public:
  Vector3d center; // 中心座標
  double radius;   // 半径
  double cR, cG, cB;  // Red, Green, Blue 値 0.0～1.0

  Sphere(double x, double y, double z, double r,
    double cr, double cg, double cb) {
      center.x = x;
      center.y = y;
      center.z = z;
      radius = r;
      cR = cr;
      cG = cg;
```

```
49        cB = cb;
50    }
51
52    // 点 p を通り、v 方向の Ray との交わりを判定する。
53    // 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
54    double getIntersec(Vector3d &p, Vector3d &v) {
55      // A*t^2 + B*t + C = 0 の形で表す
56      double A = v.x * v.x + v.y * v.y + v.z * v.z;
57      double B = 2.0 * (p.x * v.x - v.x * center.x +
58        p.y * v.y - v.y * center.y +
59        p.z * v.z- v.z * center.z);
60      double C = p.x * p.x - 2 * p.x * center.x + center.x * center.x +
61        p.y * p.y - 2 * p.y * center.y + center.y * center.y +
62        p.z * p.z - 2 * p.z * center.z + center.z * center.z -
63        radius * radius;
64      double D = B * B - 4 * A * C; // 判別式
65
66      if (D >= 0) { // 交わる
67        double t1 = (-B - sqrt(D)) / (2.0 * A);
68        double t2 = (-B + sqrt(D)) / (2.0 * A);
69        return t1 < t2 ? t1 : t2; // 小さいほうの t の値を返す
70      } else { // 交わらない
71        return -1.0;
72      }
73    }
74  };
75
76  // 板。xz 平面に平行な面とする
77  class Board {
78  public:
79    double y; // y 座標値
80
81    Board(double _y) {
82      y = _y;
83    }
84
85    // 点 p を通り、v 方向の Ray との交わりを判定する。
86    // 交点が p+tv として表せる場合の t の値を返す。交わらない場合は負の値を返す
87    double getIntersec(Vector3d &p, Vector3d &v) {
88      if(fabs(v.y) < 1.0e-10) return -1; // 水平な Ray は交わらない
89
90      double t = -1;
91      // ★ここで t の値を計算する
92      // ★ただし z 座標が-3000 より小さいなら交わらないものとする
```

```
93      // ------------------------------------------------------------------------
94      t = (y + p.y) / v.y;
95      if ((t < 0) || ((p + t * v).z < -3000)) {
96        return -1;
97      }
98      // ------------------------------------------------------------------------
99      return t;
100   }
101
102   // x と z の値から床の色を返す（格子模様になるように）
103   Vector3d getColorVec(double x, double z) {
104     // ★ x, z の値によって (1.0, 1.0, 0.7) または (0.6, 0.6, 0.6) のどちらかの色を返すようにす
105     int flag = 1;
106     if (x > 0) {
107       if (((int)x) % 200 > 100) flag *= -1;
108     } else {
109       if (((int) (-1 * x)) % 200 < 100) flag *= -1;
110     }
111     if (z > 0) {
112       if (((int)z) % 200 > 100) flag *= -1;
113     } else {
114       if (((int) (-1 * z)) % 200 < 100) flag *= -1;
115     }
116     if (flag > 0) {
117       return Vector3d(0.6, 0.6, 0.6);
118     } else {
119       return Vector3d(1.0, 1.0, 0.7);
120     }
121   }
122 };
123
124 int halfWidth;    // 描画領域の横幅/2
125 int halfHeight;   // 描画領域の縦幅/2
126 double screen_z = -1000;  // 投影面の z 座標
127 double Kd = 0.8;  // 拡散反射定数
128 double Ks = 0.8;  // 鏡面反射定数
129 double Iin = 0.5; // 入射光の強さ
130 double Ia  = 0.5; // 環境光
131 Vector3d viewPosition(0, 0, 0); // 原点=視点
132 Vector3d lightDirection(-2, -4, -2); // 入射光の進行方向
133
134
135 // レンダリングする球体
136 Sphere sphere(0.0, 0.0, -1500, // 中心座標
```

```
137         150.0,            // 半径
138         0.1, 0.7, 0.7);   // RGB値
139
140 // ----------------------------------------
141 Sphere spheres[2] = {
142   Sphere(0.0, 200, -1500,
143           50.0,
144           0.7, 0.1, 0.7),
145   Sphere(0.0, 275, -1500,
146          25.0,
147          0.7, 0.7, 0.1)};
148 // ----------------------------------------
149 // 球体の置かれている床
150 Board board(-150); // y座標値を -150 にする。(球と接するようにする)
151
152 // 2つのベクトルの成す角の cos 値を計算する
153 double getCos(Vector3d &v0, Vector3d &v1) {
154   return (v0.x * v1.x + v0.y * v1.y + v0.z * v1.z ) / (v0.length() * v1.length());
155 }
156
157 // x, y で指定されたスクリーン座標での色 (RGB) を colorVec の xyz の値に格納する
158 void getPixelColor(double x, double y, Vector3d &colorVec) {
159   // 原点からスクリーン上のピクセルへ飛ばすレイの方向
160   Vector3d ray(x - viewPosition.x, y - viewPosition.y, screen_z - viewPosition.z);
161
162   ray.normalize(); // レイの長さの正規化
163
164   // レイを飛ばして球と交差するか求める
165   double t_sphere = sphere.getIntersec(viewPosition, ray);
166   if(t_sphere > 0) { // 球との交点がある
167     // ★前回の課題を参考に、球体の表面の色を計算で求め、colorVec に設定する
168     double r = 0;
169     double g = 0;
170     double b = 0;
171     // ------------------------------------------------------------------------------
172     double Id, Is, cos_Id, cos_Is, a, I;
173     Is = Id = I = 0;
174     Vector3d P = viewPosition + t_sphere * ray;
175     Vector3d N = P - sphere.center;
176     N.normalize();
177     cos_Id = N * (-1 * lightDirection);
178     if (cos_Id > 0) {
179       Id = Iin * Kd * cos_Id;
180     }
```

```
181       int n = 20;
182       a = -1 * (lightDirection * N);
183       Vector3d R = lightDirection + 2 * a * N;
184       Vector3d V = P - viewPosition;
185       R.normalize();
186       V.normalize();
187       cos_Is = -1 * R * V;
188       if (cos_Is > 0) {
189         Is = Iin * Ks * pow(cos_Is, n);
190       }
191       I = Id + Is + Ia;
192       r = std::min(I * sphere.cR, 1.0);
193       g = std::min(I * sphere.cG, 1.0);
194       b = std::min(I * sphere.cB, 1.0);
195       // ------------------------------------------------------------------------------
196       colorVec.set(r, g, b);
197       return;
198     }
199     // ------------------------------------------------------------------------------
200     for (int i = 0; i < 2; ++i) {
201       t_sphere = spheres[i].getIntersec(viewPosition, ray);
202       if(t_sphere > 0) {
203         double r = 0;
204         double g = 0;
205         double b = 0;
206         double Id, Is, cos_Id, cos_Is, a, I;
207         Is = Id = I = 0;
208         Vector3d P = viewPosition + t_sphere * ray;
209         Vector3d N = P - spheres[i].center;
210         N.normalize();
211         cos_Id = N * (-1 * lightDirection);
212         if (cos_Id > 0) {
213           Id = Iin * Kd * cos_Id;
214         }
215         int n = 20;
216         a = -1 * (lightDirection * N);
217         Vector3d R = lightDirection + 2 * a * N;
218         Vector3d V = P - viewPosition;
219         R.normalize();
220         V.normalize();
221         cos_Is = -1 * R * V;
222         if (cos_Is > 0) {
223           Is = Iin * Ks * pow(cos_Is, n);
224         }
```

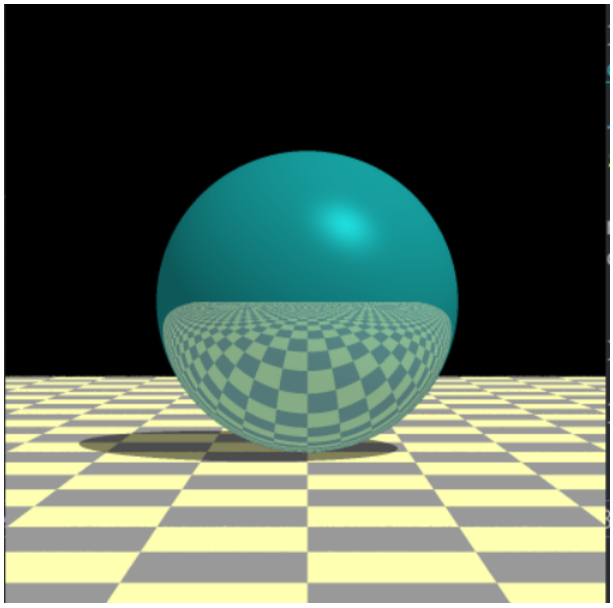
```
225         I = Id + Is + Ia;
226         r = std::min(I * spheres[i].cR, 1.0);
227         g = std::min(I * spheres[i].cG, 1.0);
228         b = std::min(I * spheres[i].cB, 1.0);
229         colorVec.set(r, g, b);
230         return;
231       }
232     }
233
234     // ----------------------------------------------------------------------------------------
235
236     // レイを飛ばして床と交差するか求める
237     double t_board = board.getIntersec(viewPosition, ray);
238
239     if(t_board > 0) { // 床との交点がある
240       // ★床の表面の色を設定する
241       // ★球の影になる場合は、RGBの値をそれぞれ0.5倍する
242       double r = 0;
243       double g = 0;
244       double b = 0;
245       // ----------------------------------------------------------------------------------------
246       double x, z;
247       Vector3d board_pos = viewPosition + t_board * ray;
248       Vector3d colors;
249       Vector3d d_lightDirection = -1 * lightDirection;
250       d_lightDirection.normalize();
251       x = board_pos.x;
252       z = board_pos.z;
253       colors = board.getColorVec(x, z);
254       r = colors.x;
255       g = colors.y;
256       b = colors.z;
257       for (int i = 0; i < 2; ++i) {
258         if (spheres[i].getIntersec(board_pos, d_lightDirection) > 0.0) {
259           r *= 0.5;
260           g *= 0.5;
261           b *= 0.5;
262         }
263       }
264       if (sphere.getIntersec(board_pos, d_lightDirection) > 0.0) {
265         r *= 0.5;
266         g *= 0.5;
267         b *= 0.5;
268       }
```

```
269       // ------------------------------------------------------------------------------
270       colorVec.set(r, g, b);
271       return;
272     }
273
274     // 何とも交差しない
275     colorVec.set(0, 0, 0); // 背景色（黒）を設定する
276   }
277
278   // 描画を行う
279   void display(void) {
280
281     glClear(GL_COLOR_BUFFER_BIT); // 描画内容のクリア
282
283     // ピクセル単位で描画色を決定するループ処理
284     for(int y = (-halfHeight); y <= halfHeight; y++ ) {
285       for(int x = (-halfWidth); x <= halfWidth; x++ ) {
286         // ------------------------------------------------------------------------------
287         int i, j;
288         Vector3d colorVecs[9];
289         Vector3d colorVec;
290         for (i = 1; i < 4; ++i) {
291           for (j = 1; j < 4; ++j) {
292             getPixelColor(x - i / 3.0, y - j / 3.0, colorVecs[(i - 1) * 3 + (j - 1)]);
293           }
294         }
295         for (i = 0; i < 9; ++i) {
296           colorVec += colorVecs[i];
297         }
298         //取得した色で、描画色を設定する
299         glColor3d(colorVec.x / 9.0, colorVec.y / 9.0, colorVec.z / 9.0);
300         // ------------------------------------------------------------------------------
301
302         // (x, y) の画素を描画
303         glBegin(GL_POINTS);
304         glVertex2i( x, y );
305         glEnd();
306       }
307     }
308     glFlush();
309   }
310
311   void resizeWindow(int w, int h) {
312     h = (h == 0) ? 1 : h;
```

```
313   glViewport(0, 0, w, h);
314   halfWidth = w/2;
315   halfHeight = h/2;
316   glMatrixMode(GL_PROJECTION);
317   glLoadIdentity();
318
319   // ウィンドウ内の座標系設定
320   glOrtho( -halfWidth, halfWidth, -halfHeight, halfHeight, 0.0, 1.0);
321   glMatrixMode(GL_MODELVIEW);
322 }
323
324 void keyboard(unsigned char key, int x, int y) {
325   switch (key) {
326     case 27: exit(0);  /* ESC code */
327   }
328   glutPostRedisplay();
329 }
330
331 int main(int argc, char** argv) {
332   lightDirection.normalize();
333
334   glutInit(&argc, argv);
335   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
336   glutInitWindowSize(400,400);
337   glutInitWindowPosition(180,10);
338   glutCreateWindow(argv[0]);
339   glClearColor(1.0, 1.0, 1.0, 1.0);
340   glShadeModel(GL_FLAT);
341
342   glutDisplayFunc(display);
343   glutReshapeFunc(resizeWindow);
344   glutKeyboardFunc(keyboard);
345   glutMainLoop();
346
347   return 0;
348 }
```

5 発展課題 2

1. 飛ばしたレイを球面で反射させ、鏡面反射を実現してみる。



5.1 ソースコード

```
#include <GL/glut.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <algorithm>

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; }
  void normalize() {
    double len = length();
    x /= len; y /= len; z /= len;
  }
  double length() { return sqrt(x * x + y * y + z * z); }
  void scale(const double s) { x *= s; y *= s; z *= s; }
  Vector3d operator+(Vector3d v) { return Vector3d(x + v.x, y + v.y, z + v.z); }
  Vector3d operator-(Vector3d v) { return Vector3d(x - v.x, y - v.y, z - v.z); }
  double operator*(Vector3d v) { return x * v.x + y* v.y + z * v.z; }
  Vector3d operator%(Vector3d v) { return Vector3d(y * v.z - z * v.y, z * v.x - x * v.z,
  Vector3d& operator=(const Vector3d& v){ x = v.x; y = v.y; z = v.z; return (*this); }
```

```
24    Vector3d& operator+=(const Vector3d& v) { x += v.x; y += v.y; z += v.z; return (*this)
25    Vector3d& operator-=(const Vector3d& v) { x -= v.x; y -= v.y; z -= v.z; return (*this)
26    void print() { printf("Vector3d(%f %f %f)\n", x, y, z); }
27  };
28  Vector3d operator-( const Vector3d& v ) { return( Vector3d( -v.x, -v.y, -v.z ) ); }
29  Vector3d operator*( const double& k, const Vector3d& v ) { return( Vector3d( k*v.x, k*v.y
30  Vector3d operator*( const Vector3d& v, const double& k ) { return( Vector3d( v.x*k, v.y*k
31  Vector3d operator/( const Vector3d& v, const double& k ) { return( Vector3d( v.x/k, v.y/k
32
33
34  // 球体
35  class Sphere {
36  public:
37    Vector3d center; // 中心座標
38    double radius;   // 半径
39    double cR, cG, cB;  // Red, Green, Blue 値 0.0〜1.0
40
41    Sphere(double x, double y, double z, double r,
42      double cr, double cg, double cb) {
43        center.x = x;
44        center.y = y;
45        center.z = z;
46        radius = r;
47        cR = cr;
48        cG = cg;
49        cB = cb;
50    }
51
52    // 点 p を通り、v 方向の Ray との交わりを判定する。
53    // 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
54    double getIntersec(Vector3d &p, Vector3d &v) {
55      // A*t^2 + B*t + C = 0 の形で表す
56      double A = v.x * v.x + v.y * v.y + v.z * v.z;
57      double B = 2.0 * (p.x * v.x - v.x * center.x +
58        p.y * v.y - v.y * center.y +
59        p.z * v.z- v.z * center.z);
60      double C = p.x * p.x - 2 * p.x * center.x + center.x * center.x +
61        p.y * p.y - 2 * p.y * center.y + center.y * center.y +
62        p.z * p.z - 2 * p.z * center.z + center.z * center.z -
63        radius * radius;
64      double D = B * B - 4 * A * C; // 判別式
65
66      if (D >= 0) { // 交わる
67        double t1 = (-B - sqrt(D)) / (2.0 * A);
```

```
68        double t2 = (-B + sqrt(D)) / (2.0 * A);
69        return t1 < t2 ? t1 : t2; // 小さいほうの t の値を返す
70      } else { // 交わらない
71        return -1.0;
72      }
73    }
74  };
75
76  // 板。xz 平面に平行な面とする
77  class Board {
78  public:
79    double y; // y 座標値
80
81    Board(double _y) {
82      y = _y;
83    }
84
85    // 点 p を通り、v 方向の Ray との交わりを判定する。
86    // 交点が p+tv として表せる場合の t の値を返す。交わらない場合は負の値を返す
87    double getIntersec(Vector3d &p, Vector3d &v) {
88      if(fabs(v.y) < 1.0e-10) return -1; // 水平な Ray は交わらない
89
90      double t = -1;
91      // ★ここで t の値を計算する
92      // ★ただし z 座標が-3000 より小さいなら交わらないものとする
93      // ------------------------------------------------------------------------
94      t = (y + p.y) / v.y;
95      if ((t < 0) || ((p + t * v).z < -3000)) {
96        return -1;
97      }
98      // ------------------------------------------------------------------------
99      return t;
100   }
101
102   // x と z の値から床の色を返す（格子模様になるように）
103   Vector3d getColorVec(double x, double z) {
104     // ★ x, z の値によって (1.0, 1.0, 0.7) または (0.6, 0.6, 0.6) のどちらかの色を返すようにす
105     int flag = 1;
106     if (x > 0) {
107       if (((int)x) % 200 > 100) flag *= -1;
108     } else {
109       if (((int) (-1 * x)) % 200 < 100) flag *= -1;
110     }
111     if (z > 0) {
```

```
112         if (((int)z) % 200 > 100) flag *= -1;
113       } else {
114         if (((int) (-1 * z)) % 200 < 100) flag *= -1;
115       }
116       if (flag > 0) {
117         return Vector3d(0.6, 0.6, 0.6);
118       } else {
119         return Vector3d(1.0, 1.0, 0.7);
120       }
121     }
122   };
123
124   int halfWidth;    // 描画領域の横幅/2
125   int halfHeight;   // 描画領域の縦幅/2
126   double screen_z = -1000;  // 投影面の z 座標
127   double Kd = 0.8;  // 拡散反射定数
128   double Ks = 0.8;  // 鏡面反射定数
129   double Iin = 0.5; // 入射光の強さ
130   double Ia  = 0.5; // 環境光
131   Vector3d viewPosition(0, 0, 0); // 原点=視点
132   Vector3d lightDirection(-2, -4, -2); // 入射光の進行方向
133
134
135   // レンダリングする球体
136   Sphere sphere(0.0, 0.0, -1500, // 中心座標
137           150.0,           // 半径
138           0.1, 0.7, 0.7);  // RGB 値
139
140   // 球体の置かれている床
141   Board board(-150); // y 座標値を -150 にする。(球と接するようにする)
142
143   // 2 つのベクトルの成す角の cos 値を計算する
144   double getCos(Vector3d &v0, Vector3d &v1) {
145     return (v0.x * v1.x + v0.y * v1.y + v0.z * v1.z ) / (v0.length() * v1.length());
146   }
147
148   // x, y で指定されたスクリーン座標での色 (RGB) を colorVec の xyz の値に格納する
149   void getPixelColor(double x, double y, Vector3d &colorVec) {
150     // 原点からスクリーン上のピクセルへ飛ばすレイの方向
151     Vector3d ray(x - viewPosition.x, y - viewPosition.y, screen_z - viewPosition.z);
152
153     ray.normalize(); // レイの長さの正規化
154
155     // レイを飛ばして球と交差するか求める
```

```
156     double t_sphere = sphere.getIntersec(viewPosition, ray);
157     if(t_sphere > 0) { // 球との交点がある
158       // ★前回の課題を参考に、球体の表面の色を計算で求め、colorVec に設定する
159       double r = 0;
160       double g = 0;
161       double b = 0;
162       // ------------------------------------------------------------------------------
163       double Id, Is, cos_Id, cos_Is, a, I;
164       Is = Id = I = 0;
165       Vector3d P = viewPosition + t_sphere * ray;
166       Vector3d N = P - sphere.center;
167       N.normalize();
168       cos_Id = N * (-1 * lightDirection);
169       if (cos_Id > 0) {
170         Id = Iin * Kd * cos_Id;
171       }
172       int n = 20;
173       a = -1 * (lightDirection * N);
174       Vector3d R = lightDirection + 2 * a * N;
175       Vector3d V = P - viewPosition;
176       R.normalize();
177       V.normalize();
178       cos_Is = -1 * R * V;
179       if (cos_Is > 0) {
180         Is = Iin * Ks * pow(cos_Is, n);
181       }
182       I = Id + Is + Ia;
183       r = std::min(I * sphere.cR, 1.0);
184       g = std::min(I * sphere.cG, 1.0);
185       b = std::min(I * sphere.cB, 1.0);
186       double a2 = -1 * P * N;
187       Vector3d R2 = P + 2 * a2 * N;
188       double t2 = board.getIntersec(P, R2);
189       if (t2 > 0.0) {
190         Vector3d K = P + t2 * R2;
191         Vector3d colors = board.getColorVec(K.x, K.z);
192         r += colors.x;
193         g += colors.y;
194         b += colors.z;
195         r /= 2;
196         g /= 2;
197         b /= 2;
198       }
199       // ------------------------------------------------------------------------------
```

```
200       colorVec.set(r, g, b);
201       return;
202     }
203
204     // レイを飛ばして床と交差するか求める
205     double t_board = board.getIntersec(viewPosition, ray);
206
207     if(t_board > 0) { // 床との交点がある
208       // ★床の表面の色を設定する
209       // ★球の影になる場合は、RGBの値をそれぞれ0.5倍する
210       double r = 0;
211       double g = 0;
212       double b = 0;
213       // ----------------------------------------------------------------------------------
214       double x, z;
215       Vector3d board_pos = viewPosition + t_board * ray;
216       Vector3d colors;
217       x = board_pos.x;
218       z = board_pos.z;
219       colors = board.getColorVec(x, z);
220       r = colors.x;
221       g = colors.y;
222       b = colors.z;
223       if (sphere.getIntersec(board_pos, lightDirection) != -1.0) {
224         r *= 0.5;
225         g *= 0.5;
226         b *= 0.5;
227       }
228       // ----------------------------------------------------------------------------------
229       colorVec.set(r, g, b);
230       return;
231     }
232
233     // 何とも交差しない
234     colorVec.set(0, 0, 0); // 背景色（黒）を設定する
235   }
236
237   // 描画を行う
238   void display(void) {
239
240     glClear(GL_COLOR_BUFFER_BIT); // 描画内容のクリア
241
242     // ピクセル単位で描画色を決定するループ処理
243     for(int y = (-halfHeight); y <= halfHeight; y++ ) {
```

```
244       for(int x = (-halfWidth); x <= halfWidth; x++ ) {
245         // ------------------------------------------------------------------------
246         int i, j;
247         Vector3d colorVecs[9];
248         Vector3d colorVec;
249         for (i = 1; i < 4; ++i) {
250           for (j = 1; j < 4; ++j) {
251             getPixelColor(x - i / 3.0, y - j / 3.0, colorVecs[(i - 1) * 3 + (j - 1)]);
252           }
253         }
254         for (i = 0; i < 9; ++i) {
255           colorVec += colorVecs[i];
256         }
257         //取得した色で、描画色を設定する
258         glColor3d(colorVec.x / 9.0, colorVec.y / 9.0, colorVec.z / 9.0);
259         // ------------------------------------------------------------------------
260 
261         // (x, y) の画素を描画
262         glBegin(GL_POINTS);
263         glVertex2i( x, y );
264         glEnd();
265       }
266     }
267     glFlush();
268   }
269 
270   void resizeWindow(int w, int h) {
271     h = (h == 0) ? 1 : h;
272     glViewport(0, 0, w, h);
273     halfWidth = w/2;
274     halfHeight = h/2;
275     glMatrixMode(GL_PROJECTION);
276     glLoadIdentity();
277 
278     // ウィンドウ内の座標系設定
279     glOrtho( -halfWidth, halfWidth, -halfHeight, halfHeight, 0.0, 1.0);
280     glMatrixMode(GL_MODELVIEW);
281   }
282 
283   void keyboard(unsigned char key, int x, int y) {
284     switch (key) {
285       case 27: exit(0);  /* ESC code */
286     }
287     glutPostRedisplay();
```

```
288 }
289
290 int main(int argc, char** argv) {
291   lightDirection.normalize();
292
293   glutInit(&argc, argv);
294   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
295   glutInitWindowSize(400,400);
296   glutInitWindowPosition(180,10);
297   glutCreateWindow(argv[0]);
298   glClearColor(1.0, 1.0, 1.0, 1.0);
299   glShadeModel(GL_FLAT);
300
301   glutDisplayFunc(display);
302   glutReshapeFunc(resizeWindow);
303   glutKeyboardFunc(keyboard);
304   glutMainLoop();
305
306   return 0;
307 }
```