計算機科学実験及び演習 4
コンピュータグラフィックス
発展課題

工学部情報学科 3 回生 1029255242
勝見久央

作成日: 2015 年 11 月 26 日

1 概要

本レポートは必須課題 1〜3 を終了後に取り組んだ発展課題と、その内容についての概要を記したレポートである.

2 課題 4

本課題ではグーローシェーディングのみを実装した. (なお、フォーンシェーディングとカメラ位置の変更については課題 5 で実装した.)

2.1 プログラム本体

プログラム本体は次のようになった.

リスト 1 kadai04.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <math.h>
5  #include <float.h>
6  #include <ctype.h>
7  #include "vrml.h"
8
9
10
11
12 //=========================================================================
13 //必要なデータ
14 #define MAGICNUM "P3"
15 #define WIDTH 256
16 #define WIDTH_STRING "256"
17 #define HEIGHT 256
18 #define HEIGHT_STRING "256"
19 #define MAX 255
20 #define MAX_STRING "255"
21 #define FOCUS 256.0
22 #define Z_BUF_MAX
23 #define ENV_LIGHT 1.0
24
```

```
25  //diffuseColorを格納する配列
26  double diffuse_color[3];
27  //shinessを格納する変数
28  double shininess;
29  //specularColorを格納する変数
30  double specular_color[3];
31
32  //光源モデルは平行光源
33
34  //光源方向
35  const double light_dir[3] = {-1.0, -1.0, 2.0};
36  //光源明るさ
37  const double light_rgb[3] = {1.0, 1.0, 1.0};
38
39  //カメラ位置は原点であるものとして投影を行う.
40
41  //=====================================================================
42
43
44  //メモリ内に画像の描画領域を確保
45  double image[HEIGHT][WIDTH][3];
46  //zバッファ用の領域を確保
47  double z_buf[HEIGHT][WIDTH];
48
49  //投影された後の2次元平面上の各点の座標を格納する領域
50  //double projected_ver[VER_NUM][2];
51  double projected_ver_buf[3][2];
52
53
54  //2点p、qを結ぶ直線上のy座標がyであるような点のx座標を返す関数
55  //eg)
56  //double p[2] = (1.0, 2.0);
57  double func1(double *p, double *q, double y){
58      double x;
59      if(p[1] > q[1]){
60          x = ((p[0] * (y - q[1])) + (q[0] * (p[1] - y))) / (p[1] - q[1]);
61      }
62      if(p[1] < q[1]){
63          x = ((q[0] * (y - p[1])) + (p[0] * (q[1] - y))) / (q[1] - p[1]);
64      }
65      if(p[1] == q[1]){
66          //解なし
67          printf("\n引数が不正です.\n2点\n(%f,␣%f)\n(%f,␣%f)\nはy座標が同じです.\n"
68                  , p[0], p[1], q[0], q[1]);
69          perror(NULL);
70          return -1;
71      }
72      return x;
73  }
74
75  //3点a[2] = {x, y},,,が1直線上にあるかどうかを判定する関数
76  //1直線上に無ければreturn 0;
77  //1直線上にあればreturn 1;
78  int lineOrNot(double *a, double *b, double *c){
79      if(a[0] == b[0]){
80          if(a[0] == c[0]){
81              return 1;
82          }
83          else{
84              return 0;
85          }
86      }
87      else{
88          if(c[1] == a[1] + ((b[1] - a[1]) / (b[0] - a[0])) * (c[0] - a[0])){
89              return 1;
90          }
91          else{
92              return 0;
93          }
94      }
95  }
96
97
98  //引数は3点の座標とRGBと3点の空間内の座標、3点で形成される空間内の平面の法線ベクトルとする
99  void shading(double *a, double *b, double *c,
100                double *rgb_a, double *rgb_b, double *rgb_c,
101                double *A, double *B, double *C,
102                double *poly_i_n_vec){
103     //3点が1直線上に並んでいるときはシェーディングができない
```

```
104     if(lineOrNot(a, b, c) == 1){}
105     else{
106         //y座標の値が真ん中点をp、その他の点をq、rとする
107         //y座標の大きさはr <= p <= qの順
108         double p[2], q[2], r[2];
109         //法線ベクトルも名前を変更する
110         double rgb_p[3], rgb_q[3], rgb_r[3];
111         //空間内での元の座標についても名前を変更する
112         double P[3], Q[3], R[3];
113 
114         if(b[1] <= a[1] && a[1] <= c[1]){
115             memcpy(p, a, sizeof(double) * 2);
116             memcpy(q, c, sizeof(double) * 2);
117             memcpy(r, b, sizeof(double) * 2);
118 
119             memcpy(rgb_p, rgb_a, sizeof(double) * 3);
120             memcpy(rgb_q, rgb_c, sizeof(double) * 3);
121             memcpy(rgb_r, rgb_b, sizeof(double) * 3);
122 
123             memcpy(P, A, sizeof(double) * 3);
124             memcpy(Q, C, sizeof(double) * 3);
125             memcpy(R, B, sizeof(double) * 3);
126         }
127         else{
128             if(c[1] <= a[1] && a[1] <= b[1]){
129                 memcpy(p, a, sizeof(double) * 2);
130                 memcpy(q, b, sizeof(double) * 2);
131                 memcpy(r, c, sizeof(double) * 2);
132 
133                 memcpy(rgb_p, rgb_a, sizeof(double) * 3);
134                 memcpy(rgb_q, rgb_b, sizeof(double) * 3);
135                 memcpy(rgb_r, rgb_c, sizeof(double) * 3);
136 
137                 memcpy(P, A, sizeof(double) * 3);
138                 memcpy(Q, B, sizeof(double) * 3);
139                 memcpy(R, C, sizeof(double) * 3);
140 
141             }
142             else{
143                 if(a[1] <= b[1] && b[1] <= c[1]){
144                     memcpy(p, b, sizeof(double) * 2);
145                     memcpy(q, c, sizeof(double) * 2);
146                     memcpy(r, a, sizeof(double) * 2);
147 
148                     memcpy(rgb_p, rgb_b, sizeof(double) * 3);
149                     memcpy(rgb_q, rgb_c, sizeof(double) * 3);
150                     memcpy(rgb_r, rgb_a, sizeof(double) * 3);
151 
152                     memcpy(P, B, sizeof(double) * 3);
153                     memcpy(Q, C, sizeof(double) * 3);
154                     memcpy(R, A, sizeof(double) * 3);
155                 }
156                 else{
157                     if(c[1] <= b[1] && b[1] <= a[1]){
158                         memcpy(p, b, sizeof(double) * 2);
159                         memcpy(q, a, sizeof(double) * 2);
160                         memcpy(r, c, sizeof(double) * 2);
161 
162                         memcpy(rgb_p, rgb_b, sizeof(double) * 3);
163                         memcpy(rgb_q, rgb_a, sizeof(double) * 3);
164                         memcpy(rgb_r, rgb_c, sizeof(double) * 3);
165 
166                         memcpy(P, B, sizeof(double) * 3);
167                         memcpy(Q, A, sizeof(double) * 3);
168                         memcpy(R, C, sizeof(double) * 3);
169                     }
170                     else{
171                         if(b[1] <= c[1] && c[1] <= a[1]){
172                             memcpy(p, c, sizeof(double) * 2);
173                             memcpy(q, a, sizeof(double) * 2);
174                             memcpy(r, b, sizeof(double) * 2);
175 
176                             memcpy(rgb_p, rgb_c, sizeof(double) * 3);
177                             memcpy(rgb_q, rgb_a, sizeof(double) * 3);
178                             memcpy(rgb_r, rgb_b, sizeof(double) * 3);
179 
180                             memcpy(P, C, sizeof(double) * 3);
181                             memcpy(Q, A, sizeof(double) * 3);
182                             memcpy(R, B, sizeof(double) * 3);
```

```
                        }
                        else{
                            if(a[1] <= c[1] && c[1] <= b[1]){
                                memcpy(p, c, sizeof(double) * 2);
                                memcpy(q, b, sizeof(double) * 2);
                                memcpy(r, a, sizeof(double) * 2);

                                memcpy(rgb_p, rgb_c, sizeof(double) * 3);
                                memcpy(rgb_q, rgb_b, sizeof(double) * 3);
                                memcpy(rgb_r, rgb_a, sizeof(double) * 3);

                                memcpy(P, C, sizeof(double) * 3);
                                memcpy(Q, B, sizeof(double) * 3);
                                memcpy(R, A, sizeof(double) * 3);

                            }
                            else{
                                printf("エラーat2055\n");
                                printf("\na[1]=%f\tb[1]=%f\tc[1]=%f\n", a[1], b[1], c[1]);
                                perror(NULL);
                            }
                        }
                    }
                }
            }
        }
        //分割可能な三角形かを判定
        if(p[1] == r[1] || p[1] == q[1]){
            //分割できない

            //長さが1の光源方向ベクトルを作成する
            //光源方向ベクトルの長さ
            double length_l =
                sqrt(pow(light_dir[0], 2.0) +
                     pow(light_dir[1], 2.0) +
                     pow(light_dir[2], 2.0));

            double light_dir_vec[3];
            light_dir_vec[0] = light_dir[0] / length_l;
            light_dir_vec[1] = light_dir[1] / length_l;
            light_dir_vec[2] = light_dir[2] / length_l;

            //2パターンの三角形を特定
            //Type 1
            if(p[1] == r[1]){
                //debug
                //printf("\np[1] == r[1]\n");
                //x座標が p <= r となるように調整
                if(r[0] <  p[0]){
                    double temp[2];
                    double temp_rgb[3];
                    memcpy(temp, r, sizeof(double) * 2);
                    memcpy(r, p, sizeof(double) * 2);
                    memcpy(p, temp, sizeof(double) * 2);

                    memcpy(temp_rgb, rgb_r, sizeof(double) * 3);
                    memcpy(rgb_r, rgb_p, sizeof(double) * 3);
                    memcpy(rgb_p, temp_rgb, sizeof(double) * 3);
                }

                //debug
                if(r[0] == p[0]){
                  perror("エラーat958");
                }

                //シェーディング処理
                //シェーディングの際に画面からはみ出した部分をどう扱うか
                //以下の実装はxy座標の範囲を0 <= x, y <= 256として実装している
                //三角形pqrをシェーディング
                //y座標はp <= r
                //debug
                if(r[1] < p[1]){
                    perror("エラーat1855");
                }

                //zバッファを確認しながら3点pqrについて先にシェーディングで色をぬる
                int temp_p0 = ceil(p[0]);
                int temp_p1 = ceil(p[1]);
                if(z_buf[temp_p1][temp_p0] < P[2]){
```

```
                    //描画しない
                }
                else{
                    image[temp_p1][temp_p0][0] = rgb_p[0];
                    image[temp_p1][temp_p0][1] = rgb_p[1];
                    image[temp_p1][temp_p0][2] = rgb_p[2];
                }

                int temp_q0 = ceil(q[0]);
                int temp_q1 = ceil(q[1]);
                if(z_buf[temp_q1][temp_q0] < Q[2]){
                    //描画しない
                }
                else{
                    image[temp_q1][temp_q0][0] = rgb_q[0];
                    image[temp_q1][temp_q0][1] = rgb_q[1];
                    image[temp_q1][temp_q0][2] = rgb_q[2];
                }

                int temp_r0 = ceil(r[0]);
                int temp_r1 = ceil(r[1]);
                if(z_buf[temp_r1][temp_r0] < R[2]){
                    //描画しない
                }
                else{
                    image[temp_r1][temp_r0][0] = rgb_r[0];
                    image[temp_r1][temp_r0][1] = rgb_r[1];
                    image[temp_r1][temp_r0][2] = rgb_r[2];
                }


                int i;
                i = ceil(p[1]);
                for(i;
                    p[1] <= i && i <= q[1];
                    i++){

                    //撮像平面からはみ出ていないかのチェック
                    if(0 <= i
                       &&
                       i <= (HEIGHT - 1)){
                            double x1 = func1(p, q, i);
                            double x2 = func1(r, q, i);
                            int j;
                            j = ceil(x1);

                            for(j;
                                x1 <= j && j <= x2 && 0 <= j && j <= (WIDTH - 1);
                                j++){


                                //=========================================
                                //p[1] == r[1]
                                //描画する点の空間内でのz座標を計算
                                //計算時の法線ベクトルは
                                double p_z =
                                    FOCUS
                                    *
                                    ((poly_i_n_vec[0]*A[0]) +
                                     (poly_i_n_vec[1]*A[1]) +
                                     (poly_i_n_vec[2]*A[2]))
                                    /
                                    ((poly_i_n_vec[0]*(j-(MAX/2))) +
                                     (poly_i_n_vec[1]*(i-(MAX/2))) +
                                     (poly_i_n_vec[2]*FOCUS));

                                //zがzバッファの該当する値より大きければ描画を行わない（何もしない）
                                if(z_buf[i][j] < p_z){}

                                //Type 1
                                else{
                                    image[i][j][0] =
                                        (
                                         ((x2-j) / (x2-x1))
                                         *
                                         ((rgb_p[0]*(q[1]-i) + rgb_q[0]*(i-p[1])) / (q[1]-p[1]))
                                         )
                                        +
                                        (
```

```
                                    ((j-x1) / (x2-x1))
                                    *
                                    ((rgb_r[0]*(q[1]-i) + rgb_q[0]*(i-r[1])) / (q[1]-r[1]))
                                    );

                                image[i][j][1] =
                                    (
                                    ((x2-j) / (x2-x1))
                                    *
                                    ((rgb_p[1]*(q[1]-i) + rgb_q[1]*(i-p[1])) / (q[1]-p[1]))
                                    )
                                   +
                                   (
                                    ((j-x1) / (x2-x1))
                                    *
                                    ((rgb_r[1]*(q[1]-i) + rgb_q[1]*(i-r[1])) / (q[1]-r[1]))
                                    );

                                image[i][j][2] =
                                   (
                                    ((x2-j) / (x2-x1))
                                    *
                                    ((rgb_p[2]*(q[1]-i) + rgb_q[2]*(i-p[1])) / (q[1]-p[1]))
                                    )
                                   +
                                   (
                                    ((j-x1) / (x2-x1))
                                    *
                                    ((rgb_r[2]*(q[1]-i) + rgb_q[2]*(i-r[1])) / (q[1]-r[1]))
                                    );

                                //zバッファの更新
                                z_buf[i][j] = p_z;
                            }
                        }
                    }
                    //はみ出ている場合は描画しない
                    else{}
                }

            }

            if(p[1] == q[1]){
                //x座標が p < q となるように調整
                if(q[0] <  p[0]){
                    double temp[2];
                    double temp_rgb[3];
                    memcpy(temp, q, sizeof(double) * 2);
                    memcpy(q, p, sizeof(double) * 2);
                    memcpy(p, temp, sizeof(double) * 2);

                    memcpy(temp_rgb, rgb_q, sizeof(double) * 3);
                    memcpy(rgb_q, rgb_p, sizeof(double) * 3);
                    memcpy(rgb_p, temp_rgb, sizeof(double) * 3);
                }

                //debug
                if(q[0] == p[0]){
                    perror("エラーat1011");
                }

                //シェーディング処理
                //三角形pqrをシェーディング
                //y座標はp <= q

                //debug
                if(q[1] < p[1]){
                    perror("エラーat1856");
                }

                //zバッファを確認しながら3点pqrについて先にシェーディングで色をぬる
                int temp_p0 = ceil(p[0]);
                int temp_p1 = ceil(p[1]);
                if(z_buf[temp_p1][temp_p0] < P[2]){
                    //描画しない
                }
                else{
                    image[temp_p1][temp_p0][0] = rgb_p[0];
                    image[temp_p1][temp_p0][1] = rgb_p[1];
```

```
                    image[temp_p1][temp_p0][2] = rgb_p[2];
                }

                int temp_q0 = ceil(q[0]);
                int temp_q1 = ceil(q[1]);
                if(z_buf[temp_q1][temp_q0] < Q[2]){
                    //描画しない
                }
                else{
                    image[temp_q1][temp_q0][0] = rgb_q[0];
                    image[temp_q1][temp_q0][1] = rgb_q[1];
                    image[temp_q1][temp_q0][2] = rgb_q[2];
                }

                int temp_r0 = ceil(r[0]);
                int temp_r1 = ceil(r[1]);
                if(z_buf[temp_r1][temp_r0] < R[2]){
                    //描画しない
                }
                else{
                    image[temp_r1][temp_r0][0] = rgb_r[0];
                    image[temp_r1][temp_r0][1] = rgb_r[1];
                    image[temp_r1][temp_r0][2] = rgb_r[2];
                }

                int i;
                i = ceil(r[1]);
                for(i;
                    r[1] <= i && i <= p[1];
                    i++){

                    //撮像部分からはみ出ていないかのチェック
                    if( 0 <= i &&
                        i <= (HEIGHT - 1)){
                        double x1 = func1(p, r, i);
                        double x2 = func1(q, r, i);

                        int j;
                        j = ceil(x1);

                        for(j;
                            x1 <= j && j <= x2 && 0 <= j && j <= (WIDTH - 1);
                            j++){

                            //============================================================
                            double p_z =
                                FOCUS
                                *
                                ((poly_i_n_vec[0]*A[0]) +
                                 (poly_i_n_vec[1]*A[1]) +
                                 (poly_i_n_vec[2]*A[2]))
                                /
                                ((poly_i_n_vec[0]*(j-(MAX/2))) +
                                 (poly_i_n_vec[1]*(i-(MAX/2))) +
                                  poly_i_n_vec[2]*FOCUS);

                            //zがzバッファの該当する値より大きければ描画を行わない（何もしない）
                            if(z_buf[i][j] < p_z){}

                            else{
                                //Type 2
                                image[i][j][0] =
                                    (
                                     ((x2-j) / (x2-x1))
                                     *
                                     ((rgb_p[0]*(i-r[1]) + rgb_r[0]*(p[1]-i)) / (p[1]-r[1]))
                                     )
                                    +
                                    (
                                     ((j-x1) / (x2-x1))
                                     *
                                     ((rgb_r[0]*(q[1]-i) + rgb_q[0]*(i-r[1])) / (q[1]-r[1]))
                                     );

                                image[i][j][1] =
                                    (
                                     ((x2-j) / (x2-x1))
                                     *
                                     ((rgb_p[1]*(i-r[1]) + rgb_r[1]*(p[1]-i)) / (p[1]-r[1]))
```

```
                                    )
                                   +
                                   (
                                    ((j-x1) / (x2-x1))
                                    *
                                    ((rgb_r[1]*(q[1]-i) + rgb_q[1]*(i-r[1])) / (q[1]-r[1]))
                                    );

                               image[i][j][2] =
                                   (
                                    ((x2-j) / (x2-x1))
                                    *
                                    ((rgb_p[2]*(i-r[1]) + rgb_r[2]*(p[1]-i)) / (p[1]-r[1]))
                                    )
                                   +
                                   (
                                    ((j-x1) / (x2-x1))
                                    *
                                    ((rgb_r[2]*(q[1]-i) + rgb_q[2]*(i-r[1])) / (q[1]-r[1]))
                                    );
                               //zバッファの更新
                               z_buf[i][j] = p_z;
                           }
                       }
                   }
                   //撮像平面からはみ出る部分は描画しない
                   else{}
               }
           }

       }
       //分割できる
       //分割してそれぞれ再帰的に処理
       //分割後の三角形はpp2qとpp2r
       else{
           double p2[2];

           p2[0] = func1(q, r, p[1]);
           p2[1] = p[1];

           double P2[3];
           P2[0] =
               (poly_i_n_vec[0]*(p2[0]-(MAX/2)))
               *
               ((poly_i_n_vec[0]*A[0]) +
                (poly_i_n_vec[1]*A[1]) +
                (poly_i_n_vec[2]*A[2]))
               /
               ((poly_i_n_vec[0]*(p2[0]-(MAX/2))) +
                (poly_i_n_vec[1]*(p2[1]-(MAX/2))) +
                poly_i_n_vec[2]*FOCUS);

           P2[1] =
               (poly_i_n_vec[1]*(p2[1]-(MAX/2)))
               *
               ((poly_i_n_vec[0]*A[0]) +
                (poly_i_n_vec[1]*A[1]) +
                (poly_i_n_vec[2]*A[2]))
               /
               ((poly_i_n_vec[0]*(p2[0]-(MAX/2))) +
                (poly_i_n_vec[1]*(p2[1]-(MAX/2))) +
                poly_i_n_vec[2]*FOCUS);

           P2[2] =
               FOCUS
               *
               ((poly_i_n_vec[0]*A[0]) +
                (poly_i_n_vec[1]*A[1]) +
                (poly_i_n_vec[2]*A[2]))
               /
               ((poly_i_n_vec[0]*(p2[0]-(MAX/2))) +
                (poly_i_n_vec[1]*(p2[1]-(MAX/2))) +
                poly_i_n_vec[2]*FOCUS);


           double rgb_p2[3];
           for(int i = 0; i < 3; i++){
               rgb_p2[i]
                   =
```

```
                        rgb_q[i] * ((p[1]-r[1])/(q[1]-r[1]))
                        +
                        rgb_r[i] * ((q[1]-p[1])/(q[1]-r[1]));
            }



            //p2のほうがpのx座標より大きくなるようにする
            if(p2[0] < p[0]){
                double temp[2];
                double temp_rgb[3];

                memcpy(temp, p2, sizeof(double) * 2);
                memcpy(p2, p, sizeof(double) * 2);
                memcpy(p, temp, sizeof(double) * 2);

                memcpy(temp_rgb, rgb_p2, sizeof(double) * 2);
                memcpy(rgb_p2, rgb_p, sizeof(double) * 2);
                memcpy(rgb_p, temp_rgb, sizeof(double) * 2);
            }
            //分割しても同一平面上なので法線ベクトルと
            //平面上の任意の点は同じものを使える.
            //求める必要があるのはrgb_p2とP2

            shading(p, p2, q, rgb_p, rgb_p2, rgb_q, P, P2, Q, poly_i_n_vec);
            shading(p, p2, r, rgb_p, rgb_p2, rgb_r, P, P2, R, poly_i_n_vec);
        }
    }
}

/*
//////////////////////////////////////////////////////////////////
*/
#define MWS 256

static int strindex( char *s, char *t)
{
    int         i, j, k;

    for (i = 0; s[i] != '\0'; i++) {
        for (j = i, k = 0;  t[k] != '\0' && s[j] == t[k]; j++, k++) ;
        if (k > 0 && t[k] == '\0')
            return i;
    }
    return -1;
}

static int getword(
                    FILE *fp,
                    char word[],
                    int sl)
{
    int i,c;

    while ( (c = fgetc(fp)) != EOF && ( isspace(c) || c == '#')) {
        if ( c == '#' ) {
            while ( (c = fgetc(fp)) != EOF && c != '\n') ;
            if ( c == EOF ) return (0);
        }
    }
    if ( c == EOF )
        return (0);
    ungetc(c,fp);

    for ( i = 0 ; i < sl - 1 ; i++) {
        word[i] = fgetc(fp);
        if ( isspace(word[i]) )
            break;
    }
    word[i] = '\0';

    return i;
}

static int read_material(
                        FILE *fp,
                        Surface *surface,
                        char *b)
{
```

```
657     while (getword(fp,b,MWS)>0) {
658         if      (strindex(b,"}")>=0) break;
659         else if (strindex(b,"diffuseColor") >= 0) {
660             getword(fp,b,MWS);
661             surface->diff[0] = atof(b);
662             getword(fp,b,MWS);
663             surface->diff[1] = atof(b);
664             getword(fp,b,MWS);
665             surface->diff[2] = atof(b);
666         }
667         else if (strindex(b,"ambientIntensity") >= 0) {
668             getword(fp,b,MWS);
669             surface->ambi = atof(b);
670         }
671         else if (strindex(b,"specularColor") >= 0) {
672             getword(fp,b,MWS);
673             surface->spec[0] = atof(b);
674             getword(fp,b,MWS);
675             surface->spec[1] = atof(b);
676             getword(fp,b,MWS);
677             surface->spec[2] = atof(b);
678         }
679         else if (strindex(b,"shininess") >= 0) {
680             getword(fp,b,MWS);
681             surface->shine = atof(b);
682         }
683     }
684     return 1;
685 }
686 
687 static int count_point(
688                         FILE *fp,
689                         char *b)
690 {
691     int num=0;
692     while (getword(fp,b,MWS)>0) {
693         if (strindex(b,"[")>=0) break;
694     }
695     while (getword(fp,b,MWS)>0) {
696         if (strindex(b,"]")>=0) break;
697         else {
698             num++;
699         }
700     }
701     if ( num %3 != 0 ) {
702         fprintf(stderr,"invalid␣file␣type[number␣of␣points␣mismatch]\n");
703     }
704     return num/3;
705 }
706 
707 static int read_point(
708                       FILE *fp,
709                       Polygon *polygon,
710                       char *b)
711 {
712     int num=0;
713     while (getword(fp,b,MWS)>0) {
714         if (strindex(b,"[")>=0) break;
715     }
716     while (getword(fp,b,MWS)>0) {
717         if (strindex(b,"]")>=0) break;
718         else {
719             polygon->vtx[num++] = atof(b);
720         }
721     }
722     return num/3;
723 }
724 
725 static int count_index(
726                         FILE *fp,
727                         char *b)
728 {
729     int num=0;
730     while (getword(fp,b,MWS)>0) {
731         if (strindex(b,"[")>=0) break;
732     }
733     while (getword(fp,b,MWS)>0) {
734         if (strindex(b,"]")>=0) break;
735         else {
```

```
            num++;
        }
    }
    if ( num %4 != 0 ) {
        fprintf(stderr,"invalid␣file␣type[number␣of␣indices␣mismatch]\n");
    }
    return num/4;
}

static int read_index(
                      FILE *fp,
                      Polygon *polygon,
                      char *b)
{
    int num=0;
    while (getword(fp,b,MWS)>0) {
        if (strindex(b,"[")>=0) break;
    }
    while (getword(fp,b,MWS)>0) {
        if (strindex(b,"]")>=0) break;
        else {
            polygon->idx[num++] = atoi(b);
            if (num%3 == 0) getword(fp,b,MWS);
        }
    }
    return num/3;
}

int read_one_obj(
                 FILE *fp,
                 Polygon *poly,
                 Surface *surface)
{
    char b[MWS];
    int flag_material = 0;
    int flag_point = 0;
    int flag_index = 0;

    /* initialize surface */
    surface->diff[0] = 1.0;
    surface->diff[1] = 1.0;
    surface->diff[2] = 1.0;
    surface->spec[0] = 0.0;
    surface->spec[1] = 0.0;
    surface->spec[2] = 0.0;
    surface->ambi = 0.0;
    surface->shine = 0.2;

    if ( getword(fp,b,MWS) <= 0) return 0;

    poly->vtx_num = 0;
    poly->idx_num = 0;

    while (flag_material==0 || flag_point==0 || flag_index==0) {
        if      (strindex(b,"Material")>=0) {
            getword(fp,b,MWS);
            flag_material = 1;
        }
        else if (strindex(b,"point")>=0) {
            fprintf(stderr,"Counting...␣[point]\n");
            poly->vtx_num = count_point(fp, b);
            flag_point = 1;
        }
        else if (strindex(b,"coordIndex")>=0) {
            fprintf(stderr,"Counting...␣[coordIndex]\n");
            poly->idx_num = count_index(fp, b);
            flag_index = 1;
        }
        else if (getword(fp,b,MWS) <= 0) return 0;
    }

    flag_material = 0;
    flag_point = 0;
    flag_index = 0;

    fseek(fp, 0, SEEK_SET);
    poly->vtx = (double *)malloc(sizeof(double)*3*poly->vtx_num);
    poly->idx = (int *)malloc(sizeof(int)*3*poly->idx_num);
    while (flag_material==0 || flag_point==0 || flag_index==0) {
```

```
        if      (strindex(b,"Material")>=0) {
            fprintf(stderr,"Reading... [Material]\n");
            read_material(fp,surface,b);
            flag_material = 1;
        }
        else if (strindex(b,"point")>=0) {
            fprintf(stderr,"Reading... [point]\n");
            read_point(fp,poly,b);
            flag_point = 1;
        }
        else if (strindex(b,"coordIndex")>=0) {
            fprintf(stderr,"Reading... [coordIndex]\n");
            read_index(fp,poly,b);
            flag_index = 1;
        }
        else if (getword(fp,b,MWS) <= 0) return 0;
    }

    return 1;
}


int main (int argc, char *argv[])
{
    int i;
    FILE *fp;
    Polygon poly;
    Surface surface;

    fp = fopen(argv[1], "r");
    read_one_obj(fp, &poly, &surface);

    fprintf(stderr,"%d vertice are found.(poly.vtx_num)\n",poly.vtx_num);
    fprintf(stderr,"%d triangles are found.(poly.idx_num)\n",poly.idx_num);

    //i th vertex
    printf("\npoly.vtx[i*3+0,1,2]\n");
    for ( i = 0 ; i < poly.vtx_num ; i++ ) {
        fprintf(stdout,"%f %f %f # %d th vertex\n",
                poly.vtx[i*3+0], poly.vtx[i*3+1], poly.vtx[i*3+2],
                i);
    }

    //i th triangle
    printf("\npoly.idx[i*3+0,1,2]\n");
    for ( i = 0 ; i < poly.idx_num ; i++ ) {
        fprintf(stdout,"%d %d %d # %d th triangle\n",
                poly.idx[i*3+0], poly.idx[i*3+1], poly.idx[i*3+2],
                i);
    }

    /* material info */
    fprintf(stderr, "diffuseColor %f %f %f\n", surface.diff[0], surface.diff[1], surface.diff[2]);
    fprintf(stderr, "specularColor %f %f %f\n", surface.spec[0], surface.spec[1], surface.spec[2]);
    fprintf(stderr, "ambientIntensity %f\n", surface.ambi);
    fprintf(stderr, "shininess %f\n", surface.shine);

    //=====================================================================

    FILE *fp_ppm;
    char *fname = argv[2];


    fp_ppm = fopen( fname, "w" );
    //ファイルが開けなかったとき
    if( fp_ppm == NULL ){
        printf("%sファイルが開けません.\n", fname);
        return -1;
    }

    //ファイルが開けたとき
    else{
        //描画領域を初期化
        for(int i = 0; i < 256; i++){
            for(int j = 0; j < 256; j++){
                image[i][j][0] = 0.0 * MAX;
                image[i][j][1] = 0.0 * MAX;
                image[i][j][2] = 0.0 * MAX;
            }
```

```
894         }
895
896          //zバッファを初期化
897         for(int i = 0; i < 256; i++){
898             for(int j = 0; j < 256; j++){
899                 z_buf[i][j] = DBL_MAX;
900             }
901         }
902
903         //diffuse_colorの格納
904         diffuse_color[0] = surface.diff[0];
905         diffuse_color[1] = surface.diff[1];
906         diffuse_color[2] = surface.diff[2];
907
908         //shininessの格納
909         //!!!!!!!!!!!!!!!注意!!!!!!!!!!!!!!!!!
910         //(実験ページの追加情報を参照)
911         //各ファイルのshininessの値は
912         //av4 0.5
913         //av5 0.5
914         //iiyama1997 1.0
915         //aa053 1.0
916         //av007 0.34
917
918         shininess = surface.shine * 128;
919
920         //speculorColorの格納
921         specular_color[0] = surface.spec[0];
922         specular_color[1] = surface.spec[1];
923         specular_color[2] = surface.spec[2];
924
925         //各頂点の法線ベクトルを求める
926         //三角形iの法線ベクトルを求めて配列に格納する(グローバル領域に保存)
927         double poly_n[poly.idx_num * 3];
928
929
930         //=============================================================================
931         //三角形iは3点A、B、Cからなる
932         //この3点で形成される三角形の法線ベクトルを求めてpoly_nに格納していく
933         for(int i = 0; i < poly.idx_num; i++){
934             //三角形iの各頂点の座標
935             double A[3], B[3], C[3];
936             A[0] = poly.vtx[(poly.idx[i*3+0])*3 + 0];
937             A[1] = poly.vtx[(poly.idx[i*3+0])*3 + 1];
938             A[2] = poly.vtx[(poly.idx[i*3+0])*3 + 2];
939
940             B[0] = poly.vtx[(poly.idx[i*3+1])*3 + 0];
941             B[1] = poly.vtx[(poly.idx[i*3+1])*3 + 1];
942             B[2] = poly.vtx[(poly.idx[i*3+1])*3 + 2];
943
944             C[0] = poly.vtx[(poly.idx[i*3+2])*3 + 0];
945             C[1] = poly.vtx[(poly.idx[i*3+2])*3 + 1];
946             C[2] = poly.vtx[(poly.idx[i*3+2])*3 + 2];
947
948             //ベクトルAB, ACから外積を計算して
949             //法線ベクトルnを求める
950             double AB[3], AC[3], n[3];
951             AB[0] = B[0] - A[0];
952             AB[1] = B[1] - A[1];
953             AB[2] = B[2] - A[2];
954
955             AC[0] = C[0] - A[0];
956             AC[1] = C[1] - A[1];
957             AC[2] = C[2] - A[2];
958
959             n[0] = (AB[1] * AC[2]) - (AB[2] * AC[1]);
960             n[1] = (AB[2] * AC[0]) - (AB[0] * AC[2]);
961             n[2] = (AB[0] * AC[1]) - (AB[1] * AC[0]);
962
963             //長さを1に調整
964             double length_n =
965                 sqrt(pow(n[0], 2.0) +
966                      pow(n[1], 2.0) +
967                      pow(n[2], 2.0));
968
969             n[0] = n[0] / length_n;
970             n[1] = n[1] / length_n;
971             n[2] = n[2] / length_n;
972
```

```
            poly_n[i*3 + 0] = n[0];
            poly_n[i*3 + 1] = n[1];
            poly_n[i*3 + 2] = n[2];
        }
        //=====================================================================


        //三角形 i の法線ベクトルが poly_n に格納された.
        //debug
        printf("\npoly_n\n");
        for(int i = 0 ; i < poly.idx_num ; i++){
            fprintf(stdout,"%f %f %f # %d th triangle\n",
                    poly_n[i*3+0], poly_n[i*3+1], poly_n[i*3+2],
                    i);
        }

        //各点の平均、正規化した法線ベクトルを求める=================================================
        //点 i の法線ベクトルをもとめて専用の配列に格納する
        //頂点 i の法線ベクトルは
        //(poly_ave_i[i*3+0], poly_ave_i[i*3+1], poly_ave_i[i*3+2])
        double poly_ave_i[poly.vtx_num];
        //点 i が隣接する平面を探索
        for(int i = 0; i < poly.vtx_num; i++){
            double sum_vec[3] = {0.0, 0.0, 0.0};
            int count = 0;
            //三角形 j の中に頂点 i が含まれるかを判定
            for(int j = 0; j < poly.idx_num; j++){
                //プログラムの可読性を保つためバラして書く
                if(poly.idx[j*3+0] == i||
                   poly.idx[j*3+1] == i||
                   poly.idx[j*3+2] == i){
                    sum_vec[0] =  sum_vec[0] + poly_n[j*3+0];
                    sum_vec[1] =  sum_vec[1] + poly_n[j*3+1];
                    sum_vec[2] =  sum_vec[2] + poly_n[j*3+2];
                    count++;
                }
            }
            //点 i の法線ベクトルを隣接平面の法線ベクトルの平均を正規化して計算する
            double ni_vec[3];
            if(count == 0){
                printf("\n warning!! 1128\n");
                printf("\n i = %d\n" ,i);
                exit(0);
            }
            ni_vec[0] = sum_vec[0] / count;
            ni_vec[1] = sum_vec[1] / count;
            ni_vec[2] = sum_vec[2] / count;

            double length_ni_vec =
                sqrt(pow(ni_vec[0], 2.0)+
                     pow(ni_vec[1], 2.0)+
                     pow(ni_vec[2], 2.0));
            if(length_ni_vec == 0){
                printf("\n warning!! 1129\n");
                exit(0);
            }
            ni_vec[0] = ni_vec[0] / length_ni_vec;
            ni_vec[1] = ni_vec[1] / length_ni_vec;
            ni_vec[2] = ni_vec[2] / length_ni_vec;

            //頂点 i の法線ベクトルを格納
            poly_ave_i[i*3+0] = ni_vec[0];
            poly_ave_i[i*3+1] = ni_vec[1];
            poly_ave_i[i*3+2] = ni_vec[2];

            //debug
            double length_ply_ave =
                sqrt(pow(poly_ave_i[i*3+0], 2.0)+
                     pow(poly_ave_i[i*3+1], 2.0)+
                     pow(poly_ave_i[i*3+2], 2.0));
            if(length_ply_ave == 0){
                printf("\n warning!! 1151 \n");
                exit(0);
            }
        }
        //=============================================================================================


        //点 i の法線ベクトルが poly_ave_i に格納された.
```

```
        //debug
        printf("\npoly_ave_i\n");
        for(int i = 0 ; i < poly.idx_num ; i++){
            fprintf(stdout,"%f␣%f␣%f␣#␣%d␣th␣vertex\n",
                    poly_ave_i[i*3+0], poly_ave_i[i*3+1], poly_ave_i[i*3+2],
                    i);
        }



        //各点の輝度値を決定する=========================================================================
        //点iの輝度値を専用の配列に格納
        double rgb_i[poly.vtx_num*3];
        //点iの輝度値は
        //(rgb_i[i*3+0], rgb_i[i*3+0], rgb_i[i*3+0],)

        //点iのiベクトルは平行光源を使うと全ての点において
        //同じになるので予め用意する==============
        double i_vec[3];
        i_vec[0] = light_dir[0];
        i_vec[1] = light_dir[1];
        i_vec[2] = light_dir[2];
        double length_i =
            sqrt(pow(i_vec[0], 2.0) + pow(i_vec[1], 2.0) + pow(i_vec[2], 2.0));
        if(length_i == 0){
                printf("\n␣warning!␣11403\n");
                exit(0);
            }
        i_vec[0] = (i_vec[0] / length_i);
        i_vec[1] = (i_vec[1] / length_i);
        i_vec[2] = (i_vec[2] / length_i);
        //====================================

        for(int i = 0; i < poly.vtx_num; i++){
            //eベクトル=========================
            double e[3];
            e[0] = -1 * poly.vtx[i*3+0];
            e[1] = -1 * poly.vtx[i*3+1];
            e[2] = -1 * poly.vtx[i*3+2];
            double length_e =
                sqrt(pow(e[0], 2.0) + pow(e[1], 2.0) + pow(e[2], 2.0));
            if(length_e == 0){
                printf("\n␣warning!␣11400\n");
                exit(0);
            }
            e[0] = (e[0] / length_e);
            e[1] = (e[1] / length_e);
            e[2] = (e[2] / length_e);
            //==================================

            //sベクトル=========================
            double s[3];
            s[0] = e[0] - i_vec[0];
            s[1] = e[1] - i_vec[1];
            s[2] = e[2] - i_vec[2];
            double length_s =
                sqrt(pow(s[0], 2.0) + pow(s[1], 2.0) + pow(s[2], 2.0));
            if(length_s == 0){
                printf("\n␣warning!␣11401\n");
                exit(0);
            }
            s[0] = (s[0] / length_s);
            s[1] = (s[1] / length_s);
            s[2] = (s[2] / length_s);
            //==================================


            //iベクトルとnベクトルの内積を計算
            double ip =
                (poly_ave_i[i*3+0] * i_vec[0]) +
                (poly_ave_i[i*3+1] * i_vec[1]) +
                (poly_ave_i[i*3+2] * i_vec[2]);

            if(0 <= ip){
                ip = 0;
            }

            //内積sn
```

```
            double sn
                = ((s[0] * poly_ave_i[i*3+0]) +
                   (s[1] * poly_ave_i[i*3+1]) +
                   (s[2] * poly_ave_i[i*3+2]));
            if(sn <= 0){
                sn = 0;
            }

            //頂点 i の 輝 度 値 を 計 算
            rgb_i[i*3+0] =
                //拡散反射
                (-1 * ip * diffuse_color[0] * light_rgb[0] * MAX)
                //鏡面反射
                + (pow(sn, shininess) * specular_color[0] * light_rgb[0] * MAX)
                //環境反射
                + surface.ambi * ENV_LIGHT * MAX
                ;

            rgb_i[i*3+1] =
                //拡散反射
                (-1 * ip * diffuse_color[1] * light_rgb[1] * MAX)
                //鏡面反射
                + (pow(sn, shininess) * specular_color[1] * light_rgb[1] * MAX)
                //環境反射
                + surface.ambi * ENV_LIGHT * MAX
                ;

            rgb_i[i*3+2] =
                //拡散反射
                (-1 * ip * diffuse_color[2] * light_rgb[2] * MAX)
                //鏡面反射
                + (pow(sn, shininess) * specular_color[2] * light_rgb[2] * MAX)
                //環境反射
                + surface.ambi * ENV_LIGHT * MAX
                ;


            //debug
            printf("\nrgb_i=(%f\t%f\t%f),␣i=%d\n", rgb_i[i*3+0], rgb_i[i*3+1], rgb_i[i*3+2], i);
        }
        //=============================================================================================



        //シェーディング
        //ポリゴン i を シ ェ ー デ ィ ン グ ================================================

        for(int i = 0; i < poly.idx_num; i++){
            //三角形の各点の透視投影処理==========================================
            for(int j = 0; j < 3; j++){
                double xp = poly.vtx[(poly.idx[i*3+j])*3 + 0];
                double yp = poly.vtx[(poly.idx[i*3+j])*3 + 1];
                double zp = poly.vtx[(poly.idx[i*3+j])*3 + 2];
                double zi = FOCUS;

                //debug
                //printf("\nxp = %f\typ = %f\tzp = %f\n", xp, yp, zp);

                //debug
                if(zp == 0){
                    printf("\n(%f\t%f\t%f)␣i=%d,␣j=%d\n", xp, yp, zp, i, j);
                    perror("\nエラー0934\n");
                    exit(0);
                    //break;
                }

                double xp2 = xp * (zi / zp);
                double yp2 = yp * (zi / zp);
                double zp2 = zi;

                //座標軸を平行移動
                projected_ver_buf[j][0] = (MAX / 2) + xp2;
                projected_ver_buf[j][1] = (MAX / 2) + yp2;
            }


            double a[2], b[2], c[2];
            a[0] = projected_ver_buf[0][0];
```

```
1210            a[1] = projected_ver_buf[0][1];
1211            b[0] = projected_ver_buf[1][0];
1212            b[1] = projected_ver_buf[1][1];
1213            c[0] = projected_ver_buf[2][0];
1214            c[1] = projected_ver_buf[2][1];
1215            //================================================================
1216
1217            //点a、b、cがそれぞれ何番目の頂点かを参照
1218
1219            int index_a = poly.idx[i*3+0];
1220            int index_b = poly.idx[i*3+1];
1221            int index_c = poly.idx[i*3+2];
1222
1223            //点iの輝度値を参照する
1224            double rgb_a[3], rgb_b[3], rgb_c[3];
1225            rgb_a[0] = rgb_i[index_a*3+0];
1226            rgb_a[1] = rgb_i[index_a*3+1];
1227            rgb_a[2] = rgb_i[index_a*3+2];
1228
1229            rgb_b[0] = rgb_i[index_b*3+0];
1230            rgb_b[1] = rgb_i[index_b*3+1];
1231            rgb_b[2] = rgb_i[index_b*3+2];
1232
1233            rgb_c[0] = rgb_i[index_c*3+0];
1234            rgb_c[1] = rgb_i[index_c*3+1];
1235            rgb_c[2] = rgb_i[index_c*3+2];
1236
1237            //関数shadingの中では3点の空間内での座標も必要
1238            double A[3], B[3], C[3];
1239            A[0] = poly.vtx[index_a*3 + 0];
1240            A[1] = poly.vtx[index_a*3 + 1];
1241            A[2] = poly.vtx[index_a*3 + 2];
1242
1243            B[0] = poly.vtx[index_b*3 + 0];
1244            B[1] = poly.vtx[index_b*3 + 1];
1245            B[2] = poly.vtx[index_b*3 + 2];
1246
1247            C[0] = poly.vtx[index_c*3 + 0];
1248            C[1] = poly.vtx[index_c*3 + 1];
1249            C[2] = poly.vtx[index_c*3 + 2];
1250            //三角形iのシェーディングを行う
1251
1252            //三角形iの（本来の）法線ベクトルは
1253            //(poly_n[i*3+0], poly_n[i*3+1], poly_n[i*3+2])
1254            double poly_i_n_vec[3]
1255                = {poly_n[i*3+0], poly_n[i*3+1], poly_n[i*3+2]};
1256
1257            shading(a, b, c, rgb_a, rgb_b, rgb_c, A, B, C, poly_i_n_vec);
1258        }
1259
1260
1261
1262
1263        //ヘッダー出力
1264        fputs(MAGICNUM, fp_ppm);
1265        fputs("\n", fp_ppm);
1266        fputs(WIDTH_STRING, fp_ppm);
1267        fputs("␣", fp_ppm);
1268        fputs(HEIGHT_STRING, fp_ppm);
1269        fputs("\n", fp_ppm);
1270        fputs(MAX_STRING, fp_ppm);
1271        fputs("\n" ,fp_ppm);
1272
1273        //imageの出力
1274        for(int i = 0; i < 256; i++){
1275            for(int j = 0; j < 256; j++){
1276                char r[256];
1277                char g[256];
1278                char b[256];
1279                char str[1024];
1280                sprintf(r, "%d", (int)round(image[i][j][0]));
1281                sprintf(g, "%d", (int)round(image[i][j][1]));
1282                sprintf(b, "%d", (int)round(image[i][j][2]));
1283                sprintf(str, "%s\t%s\t%s\n", r, g, b);
1284                fputs(str, fp_ppm);
1285            }
1286        }
1287    }
1288    fclose(fp_ppm);
```

```
1289      fclose(fp);
1290
1291      printf("\nppmファイル␣%s␣の作成が完了しました.\n", fname );
1292      return 1;
1293  }
```

3 実行例

kadai04.c と同一のディレクトリに次のプログラムを置き、

リスト 2 EvalKadai04.sh

```
1  #!/bin/sh
2  SRC=kadai04.c
3  WRL=sample/av5.wrl
4  PPM=Kadai04ForAv5.ppm
5
6  gcc -Wall $SRC
7  ./a.out $WRL $PPM
8  open $PPM
9  echo completed!! "\xF0\x9f\x8d\xbb"
```

さらに同一ディレクトリ内のディレクトリ sample の中に対象とする VRML ファイルを置いて、

```
1  $ sh EvalKadai04.sh
```

を実行した. 出力画像は図 1 のようになった.

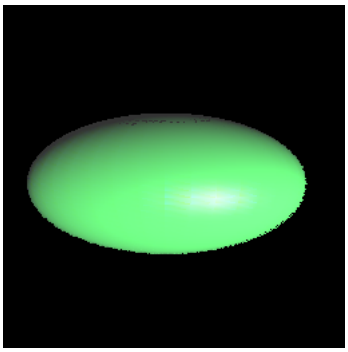

図 1 av5.wrl の出力結果

4 課題 5

本課題ではまず環境マッピングをフォーンシェーディングによって実装した. さらに課題 4 では実装しなかったカメラ位置の変更機能をプログラムに加えた. なお、便宜上、カメラ位置についてはコマンドライン変数より x 座標のみを指定する仕様としている. また、環境マップのパスはプログラム内のマクロで

```
./sample/spheremap1.ppm
```

と指定した. (当初、フォーンシェーディングではなくグーローシェーディングで実装を行っていたが、それではポリゴン単位で環境マップの輝度値が一様になる傾向があることがわかったため、フォーンシェーディングで実装を行った.)

4.1 プログラム本体

プログラム本体は次のようになった.

リスト 3　kadai05.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <float.h>


//========================================================================
//必要なデータ
#define MAGICNUM "P3"
#define WIDTH 256
#define WIDTH_STRING "256"
#define HEIGHT 256
#define HEIGHT_STRING "256"
#define MAX 255
#define MAX_STRING "255"
#define FOCUS 256.0
#define ENV_LIGHT 1.0
#define MAP_FILENAME "./sample/spheremap1.ppm"

//diffuseColorを格納する配列
double diffuse_color[3];
//shinessを格納する変数
double shininess;
//specularColorを格納する変数
double specular_color[3];
//光源モデルは平行光源
//光源方向
const double light_dir[3] = {-1.0, -1.0, 2.0};
//光源明るさ
const double light_rgb[3] = {1.0, 1.0, 1.0};
//カメラ位置
double camera_xyz[3];
//========================================================================
//メモリ内に画像の描画領域を確保
double image[HEIGHT][WIDTH][3];
//zバッファ用の領域を確保
double z_buf[HEIGHT][WIDTH];
//投影された後の2次元平面上の各点の座標を格納する領域
//double projected_ver[VER_NUM][2];
double projected_ver_buf[3][2];
//環境マップを格納するリストの構造体の定義
struct list{
    int num;
    int index;
    struct list *next;
}list;
typedef struct list LIST;

//環境マッピング用の画像の縦横幅、上限値
int ppm_width, ppm_height, ppm_max;

LIST *add_list(int num, int index, LIST *tail){
  LIST *p;

  /* 記憶領域の確保 */
  if ((p = (LIST *) malloc(sizeof(LIST))) == NULL) {
    printf("malloc␣error\n");
    exit(EXIT_FAILURE);
  }

  /* リストにデータを登録 */
  p->num = num;
  p->index = index;
  /* ポインタのつなぎ換え */
  p->next = NULL;
  tail->next = p;
    //最後尾はpになる
  return p;
```

```
}

//2点 p、 qを結ぶ直線上のy座標がyであるような点のx座標を返す関数
//eg)
//double p[2] = (1.0, 2.0);
double func1(double *p, double *q, double y){
    double x;
    if(p[1] > q[1]){
        x = ((p[0] * (y - q[1])) + (q[0] * (p[1] - y))) / (p[1] - q[1]);
    }
    if(p[1] < q[1]){
        x = ((q[0] * (y - p[1])) + (p[0] * (q[1] - y))) / (q[1] - p[1]);
    }
    if(p[1] == q[1]){
        //解なし
        printf("\n引数が不正です.\n2点\n(%f,␣%f)\n(%f,␣%f)\nはy座標が同じです.\n"
               , p[0], p[1], q[0], q[1]);
        perror(NULL);
        return -1;
    }
    return x;
}

//3点a[2] = {x, y},,,が1直線上にあるかどうかを判定する関数
//1直線上に無ければreturn 0;
//1直線上にあればreturn 1;
int lineOrNot(double *a, double *b, double *c){
    if(a[0] == b[0]){
        if(a[0] == c[0]){
            return 1;
        }
        else{
            return 0;
        }
    }
    else{
        if(c[1] == a[1] + ((b[1] - a[1]) / (b[0] - a[0])) * (c[0] - a[0])){
            return 1;
        }
        else{
            return 0;
        }
    }
}


//引数は3点の座標とRGBと3点の空間内の座標、3点で形成される空間内の平面の法線ベクトルとする
void shading(double *a, double *b, double *c,
             double *n_a, double *n_b, double *n_c,
             double *A, double *B, double *C, double *poly_i_n_vec,
             int input_ppm[ppm_height][ppm_width][3]){
    //3点が1直線上に並んでいるときはシェーディングができない
    if(lineOrNot(a, b, c) == 1){
        //塗りつぶす点が無いので何もしない.
    }
    else{
        //y座標の値が真ん中点をp、その他の点をq、rとする
        //y座標の大きさはr <= p <= qの順
        double p[2], q[2], r[2];
        //法線ベクトルも名前を変更する
        double n_p[3], n_q[3], n_r[3];
        //空間内での元の座標についても名前を変更する
        double P[3], Q[3], R[3];

        if(b[1] <= a[1] && a[1] <= c[1]){
            memcpy(p, a, sizeof(double) * 2);
            memcpy(q, c, sizeof(double) * 2);
            memcpy(r, b, sizeof(double) * 2);

            memcpy(n_p, n_a, sizeof(double) * 3);
            memcpy(n_q, n_c, sizeof(double) * 3);
            memcpy(n_r, n_b, sizeof(double) * 3);

            memcpy(P, A, sizeof(double) * 3);
            memcpy(Q, C, sizeof(double) * 3);
            memcpy(R, B, sizeof(double) * 3);
        }
        else{
            if(c[1] <= a[1] && a[1] <= b[1]){
```

```
149                 memcpy(p, a, sizeof(double) * 2);
150                 memcpy(q, b, sizeof(double) * 2);
151                 memcpy(r, c, sizeof(double) * 2);
152 
153                 memcpy(n_p, n_a, sizeof(double) * 3);
154                 memcpy(n_q, n_b, sizeof(double) * 3);
155                 memcpy(n_r, n_c, sizeof(double) * 3);
156 
157                 memcpy(P, A, sizeof(double) * 3);
158                 memcpy(Q, B, sizeof(double) * 3);
159                 memcpy(R, C, sizeof(double) * 3);
160 
161             }
162             else{
163                 if(a[1] <= b[1] && b[1] <= c[1]){
164                     memcpy(p, b, sizeof(double) * 2);
165                     memcpy(q, c, sizeof(double) * 2);
166                     memcpy(r, a, sizeof(double) * 2);
167 
168                     memcpy(n_p, n_b, sizeof(double) * 3);
169                     memcpy(n_q, n_c, sizeof(double) * 3);
170                     memcpy(n_r, n_a, sizeof(double) * 3);
171 
172                     memcpy(P, B, sizeof(double) * 3);
173                     memcpy(Q, C, sizeof(double) * 3);
174                     memcpy(R, A, sizeof(double) * 3);
175                 }
176                 else{
177                     if(c[1] <= b[1] && b[1] <= a[1]){
178                         memcpy(p, b, sizeof(double) * 2);
179                         memcpy(q, a, sizeof(double) * 2);
180                         memcpy(r, c, sizeof(double) * 2);
181 
182                         memcpy(n_p, n_b, sizeof(double) * 3);
183                         memcpy(n_q, n_a, sizeof(double) * 3);
184                         memcpy(n_r, n_c, sizeof(double) * 3);
185 
186                         memcpy(P, B, sizeof(double) * 3);
187                         memcpy(Q, A, sizeof(double) * 3);
188                         memcpy(R, C, sizeof(double) * 3);
189                     }
190                     else{
191                         if(b[1] <= c[1] && c[1] <= a[1]){
192                             memcpy(p, c, sizeof(double) * 2);
193                             memcpy(q, a, sizeof(double) * 2);
194                             memcpy(r, b, sizeof(double) * 2);
195 
196                             memcpy(n_p, n_c, sizeof(double) * 3);
197                             memcpy(n_q, n_a, sizeof(double) * 3);
198                             memcpy(n_r, n_b, sizeof(double) * 3);
199 
200                             memcpy(P, C, sizeof(double) * 3);
201                             memcpy(Q, A, sizeof(double) * 3);
202                             memcpy(R, B, sizeof(double) * 3);
203                         }
204                         else{
205                             if(a[1] <= c[1] && c[1] <= b[1]){
206                                 memcpy(p, c, sizeof(double) * 2);
207                                 memcpy(q, b, sizeof(double) * 2);
208                                 memcpy(r, a, sizeof(double) * 2);
209 
210                                 memcpy(n_p, n_c, sizeof(double) * 3);
211                                 memcpy(n_q, n_b, sizeof(double) * 3);
212                                 memcpy(n_r, n_a, sizeof(double) * 3);
213 
214                                 memcpy(P, C, sizeof(double) * 3);
215                                 memcpy(Q, B, sizeof(double) * 3);
216                                 memcpy(R, A, sizeof(double) * 3);
217 
218                             }
219                             else{
220                                 printf("エラーat2055\n");
221                                 printf("\na[1]=%f\tb[1]=%f\tc[1]=%f\n", a[1], b[1], c[1]);
222                                 perror(NULL);
223                             }
224                         }
225                     }
226                 }
227             }
```

```
228         }
229         //分割可能な三角形かを判定
230         if(p[1] == r[1] || p[1] == q[1]){
231             //分割できない
232 
233             //長さが1の光源方向ベクトルを作成する
234             //光源方向ベクトルの長さ
235             double length_l =
236                 sqrt(pow(light_dir[0], 2.0) +
237                      pow(light_dir[1], 2.0) +
238                      pow(light_dir[2], 2.0));
239 
240             double light_dir_vec[3];
241             light_dir_vec[0] = light_dir[0] / length_l;
242             light_dir_vec[1] = light_dir[1] / length_l;
243             light_dir_vec[2] = light_dir[2] / length_l;
244 
245             //2パターンの三角形を特定
246             //Type 1
247             if(p[1] == r[1]){
248                 //debug
249                 //printf("\np[1] == r[1]\n");
250                 //x座標が p <= r となるように調整
251                 if(r[0] <  p[0]){
252                     double temp[2];
253                     double temp_n[3];
254                     memcpy(temp, r, sizeof(double) * 2);
255                     memcpy(r, p, sizeof(double) * 2);
256                     memcpy(p, temp, sizeof(double) * 2);
257 
258                     memcpy(temp_n, n_r, sizeof(double) * 3);
259                     memcpy(n_r, n_p, sizeof(double) * 3);
260                     memcpy(n_p, temp_n, sizeof(double) * 3);
261                 }
262 
263                 //debug
264                 if(r[0] == p[0]){
265                   perror("エラーat958");
266                 }
267 
268                 //シェーディング処理
269                 //シェーディングの際に画面からはみ出した部分をどう扱うか
270                 //以下の実装はxy座標の範囲を0 <= x, y <= 256として実装している
271                 //三角形pqrをシェーディング
272                 //y座標はp <= r
273                 //debug
274                 if(r[1] < p[1]){
275                     perror("エラーat1855");
276                 }
277 
278 
279                 /* 点(j, i)のシェーディング================================================================== */
280                 int i;
281                 i = ceil(p[1]);
282                 for(i;
283                     p[1] <= i && i <= q[1];
284                     i++){
285 
286                     //撮像平面からはみ出ていないかのチェック
287                     if(0 <= i
288                        &&
289                        i <= (HEIGHT - 1)){
290                         double x1 = func1(p, q, i);
291                         double x2 = func1(r, q, i);
292                         int j;
293                         j = ceil(x1);
294 
295                         for(j;
296                             x1 <= j && j <= x2 && 0 <= j && j <= (WIDTH - 1);
297                             j++){
298 
299 
300                             /* 点(j, i)の空間座標を求める. */
301                             double p_or[3];
302                             double k =
303                                 ((poly_i_n_vec[0] * (A[0] - camera_xyz[0])) +
304                                  (poly_i_n_vec[1] * (A[1] - camera_xyz[1])) +
305                                  (poly_i_n_vec[2] * (A[2] - camera_xyz[2])))
```

```
306                                 /
307                                 ((poly_i_n_vec[0] * ((j-(WIDTH/2)) - camera_xyz[0])) +
308                                  (poly_i_n_vec[1] * ((i-(HEIGHT/2)) - camera_xyz[1])) +
309                                  (poly_i_n_vec[2] * (FOCUS - camera_xyz[2])));
310
311                             p_or[0] = k * ((j-(WIDTH/2)) - camera_xyz[0]) + camera_xyz[0];
312                             p_or[1] = k * ((i-(HEIGHT/2)) - camera_xyz[1]) + camera_xyz[1];
313                             p_or[2] = k * (FOCUS - camera_xyz[2]) + camera_xyz[2];
314
315
316                             /* 点(J, i)の法線ベクトルを線形補間によって求める */
317                             double n_ji[3];
318                             n_ji[0] =
319                                     (
320                                      ((x2-j) / (x2-x1))
321                                      *
322                                      ((n_p[0]*(q[1]-i) + n_q[0]*(i-p[1])) / (q[1]-p[1]))
323                                      )
324                                     +
325                                     (
326                                      ((j-x1) / (x2-x1))
327                                      *
328                                      ((n_r[0]*(q[1]-i) + n_q[0]*(i-r[1])) / (q[1]-r[1]))
329                                      );
330
331                             n_ji[1] =
332                                      (
333                                      ((x2-j) / (x2-x1))
334                                      *
335                                      ((n_p[1]*(q[1]-i) + n_q[1]*(i-p[1])) / (q[1]-p[1]))
336                                      )
337                                     +
338                                     (
339                                      ((j-x1) / (x2-x1))
340                                      *
341                                      ((n_r[1]*(q[1]-i) + n_q[1]*(i-r[1])) / (q[1]-r[1]))
342                                      );
343
344                             n_ji[2] =
345                                     (
346                                      ((x2-j) / (x2-x1))
347                                      *
348                                      ((n_p[2]*(q[1]-i) + n_q[2]*(i-p[1])) / (q[1]-p[1]))
349                                      )
350                                     +
351                                     (
352                                      ((j-x1) / (x2-x1))
353                                      *
354                                      ((n_r[2]*(q[1]-i) + n_q[2]*(i-r[1])) / (q[1]-r[1]))
355                                      );
356                             double length_n_ji =
357                                 sqrt(pow(n_ji[0], 2.0) +
358                                      pow(n_ji[1], 2.0) +
359                                      pow(n_ji[2], 2.0));
360                             n_ji[0] = (n_ji[0] / length_n_ji);
361                             n_ji[1] = (n_ji[1] / length_n_ji);
362                             n_ji[2] = (n_ji[2] / length_n_ji);
363
364
365                             /* 視線方向ベクトルを求める */
366                             double u[3];
367                             u[0] = p_or[0] - camera_xyz[0];
368                             u[1] = p_or[1] - camera_xyz[1];
369                             u[2] = p_or[2] - camera_xyz[2];
370                             double length_u =
371                                 sqrt(pow(u[0], 2.0) +
372                                      pow(u[1], 2.0) +
373                                      pow(u[2], 2.0));
374                             u[0] = (u[0] / length_u);
375                             u[1] = (u[1] / length_u);
376                             u[2] = (u[2] / length_u);
377
378                             /* 反射ベクトルを求める */
379                             double nu = (n_ji[0]*u[0])+(n_ji[1]*u[1])+(n_ji[2]*u[2]);
380                             double f[3];
381                             f[0] = u[0] - 2*n_ji[0]*nu;
382                             f[1] = u[1] - 2*n_ji[1]*nu;
383                             f[2] = u[2] - 2*n_ji[2]*nu;
384                             double length_f =
```

```
                                sqrt(pow(f[0], 2.0) +
                                     pow(f[1], 2.0) +
                                     pow(f[2], 2.0));
                            f[0] = (f[0] / length_f);
                            f[1] = (f[1] / length_f);
                            f[2] = (f[2] / length_f);

                            double m = 2*sqrt(pow(f[0], 2.0)+
                                              pow(f[1], 2.0)+
                                              pow(f[2], 2.0));

                            int s_x = (int)round((0.5 + (f[0]/m)) * ppm_width);
                            int t_y = (int)round((0.5 - (f[1]/m)) * ppm_height);
                            int env_r, env_g, env_b;
                            //環境マップ外なら描写しない
                            if(s_x < 0 || t_y < 0 ||
                               s_x > (ppm_width-1) || t_y > (ppm_height-1)){
                                env_r = 0;
                                env_g = 0;
                                env_b = 0;
                            }
                            else{
                                env_r = input_ppm[t_y][s_x][0];
                                env_g = input_ppm[t_y][s_x][1];
                                env_b = input_ppm[t_y][s_x][2];
                            }


                            //zがzバッファの該当する値より大きければ描画を行わない（何もしない）
                            if(z_buf[i][j] < p_or[2]){}
                            //小さいとき
                            else{
                                image[i][j][0] =  env_r;

                                image[i][j][1] =  env_g;

                                image[i][j][2] =  env_b;

                                //zバッファの更新
                                z_buf[i][j] = p_or[2];
                            }
                        }
                    }
                    //はみ出ている場合は描画しない
                    else{}
                }
                /* 点(j, i)のシェーディングここまで================================================================================ */

            }

            if(p[1] == q[1]){
                //x座標が p < q となるように調整
                if(q[0] <  p[0]){
                    double temp[2];
                    double temp_n[3];
                    memcpy(temp, q, sizeof(double) * 2);
                    memcpy(q, p, sizeof(double) * 2);
                    memcpy(p, temp, sizeof(double) * 2);

                    memcpy(temp_n, n_q, sizeof(double) * 3);
                    memcpy(n_q, n_p, sizeof(double) * 3);
                    memcpy(n_p, temp_n, sizeof(double) * 3);
                }

                //debug
                if(q[0] == p[0]){
                    perror("エラーat1011");
                }

                //シェーディング処理
                //三角形pqrをシェーディング
                //y座標はp <= q

                //debug
                if(q[1] < p[1]){
                    perror("エラーat1856");
                }

```

```
463
464                 /* 点(j, i)のシェーディング============================================== */
465                 int i;
466                 i = ceil(r[1]);
467
468                 for(i;
469                     r[1] <= i && i <= p[1];
470                     i++){
471
472                     //撮像部分からはみ出ていないかのチェック
473                     if( 0 <= i &&
474                         i <= (HEIGHT - 1)){
475                         double x1 = func1(p, r, i);
476                         double x2 = func1(q, r, i);
477
478                         int j;
479                         j = ceil(x1);
480
481                         for(j;
482                             x1 <= j && j <= x2 && 0 <= j && j <= (WIDTH - 1);
483                             j++){
484
485                               /* 点(j, i)の空間座標を求める. */
486                                double p_or[3];
487                                double k =
488                                    ((poly_i_n_vec[0] * (A[0] - camera_xyz[0])) +
489                                     (poly_i_n_vec[1] * (A[1] - camera_xyz[1])) +
490                                     (poly_i_n_vec[2] * (A[2] - camera_xyz[2])))
491                                    /
492                                    ((poly_i_n_vec[0] * ((j-(WIDTH/2)) - camera_xyz[0])) +
493                                     (poly_i_n_vec[1] * ((i-(HEIGHT/2)) - camera_xyz[1])) +
494                                     (poly_i_n_vec[2] * (FOCUS - camera_xyz[2])));
495
496                                p_or[0] = k * ((j-(WIDTH/2)) - camera_xyz[0]) + camera_xyz[0];
497                                p_or[1] = k * ((i-(HEIGHT/2)) - camera_xyz[1]) + camera_xyz[1];
498                                p_or[2] = k * (FOCUS - camera_xyz[2]) + camera_xyz[2];
499
500                                /* 点(J, i)の法線ベクトルを線形補間によって求める */
501                                double n_ji[3];
502                                n_ji[0] =
503                                    (
504                                     ((x2-j) / (x2-x1))
505                                     *
506                                     ((n_p[0]*(i-r[1]) + n_r[0]*(p[1]-i)) / (p[1]-r[1]))
507                                     )
508                                    +
509                                    (
510                                     ((j-x1) / (x2-x1))
511                                     *
512                                     ((n_r[0]*(q[1]-i) + n_q[0]*(i-r[1])) / (q[1]-r[1]))
513                                     );
514
515                                n_ji[1] =
516                                    (
517                                     ((x2-j) / (x2-x1))
518                                     *
519                                     ((n_p[1]*(i-r[1]) + n_r[1]*(p[1]-i)) / (p[1]-r[1]))
520                                     )
521                                    +
522                                    (
523                                     ((j-x1) / (x2-x1))
524                                     *
525                                     ((n_r[1]*(q[1]-i) + n_q[1]*(i-r[1])) / (q[1]-r[1]))
526                                     );
527
528                                n_ji[2] =
529                                    (
530                                     ((x2-j) / (x2-x1))
531                                     *
532                                     ((n_p[2]*(i-r[1]) + n_r[2]*(p[1]-i)) / (p[1]-r[1]))
533                                     )
534                                    +
535                                    (
536                                     ((j-x1) / (x2-x1))
537                                     *
538                                     ((n_r[2]*(q[1]-i) + n_q[2]*(i-r[1])) / (q[1]-r[1]))
539                                     );
540                                double length_n_ji =
541                                    sqrt(pow(n_ji[0], 2.0) +
```

```
542                                     pow(n_ji[1], 2.0) +
543                                     pow(n_ji[2], 2.0));
544                                 n_ji[0] = (n_ji[0] / length_n_ji);
545                                 n_ji[1] = (n_ji[1] / length_n_ji);
546                                 n_ji[2] = (n_ji[2] / length_n_ji);
547
548
549                                 /* 視線方向ベクトルを求める */
550                                 double u[3];
551                                 u[0] = p_or[0] - camera_xyz[0];
552                                 u[1] = p_or[1] - camera_xyz[1];
553                                 u[2] = p_or[2] - camera_xyz[2];
554                                 double length_u =
555                                     sqrt(pow(u[0], 2.0) +
556                                          pow(u[1], 2.0) +
557                                          pow(u[2], 2.0));
558                                 u[0] = (u[0] / length_u);
559                                 u[1] = (u[1] / length_u);
560                                 u[2] = (u[2] / length_u);
561
562                                 /* 反射ベクトルを求める */
563                                 double nu =
564                                     (n_ji[0]*u[0])+(n_ji[1]*u[1])+(n_ji[2]*u[2]);
565                                 double f[3];
566                                 f[0] = u[0] - 2*n_ji[0]*nu;
567                                 f[1] = u[1] - 2*n_ji[1]*nu;
568                                 f[2] = u[2] - 2*n_ji[2]*nu;
569                                 double length_f =
570                                     sqrt(pow(f[0], 2.0) +
571                                          pow(f[1], 2.0) +
572                                          pow(f[2], 2.0));
573                                 f[0] = (f[0] / length_f);
574                                 f[1] = (f[1] / length_f);
575                                 f[2] = (f[2] / length_f);
576
577                                 double m = 2*sqrt(pow(f[0], 2.0)+
578                                                   pow(f[1], 2.0)+
579                                                   pow(f[2], 2.0));
580
581
582                                 int s_x = (int)round((0.5 + (f[0]/m)) * ppm_width);
583                                 int t_y = (int)round((0.5 - (f[1]/m)) * ppm_height);
584                                 int env_r, env_g, env_b;
585                                 //環境マップ外なら描写しない
586                                 if(s_x < 0 || t_y < 0 ||
587                                    s_x > (ppm_width-1) || t_y > (ppm_height-1)){
588                                     env_r = 0;
589                                     env_g = 0;
590                                     env_b = 0;
591                                 }
592                                 else{
593                                     env_r = input_ppm[t_y][s_x][0];
594                                     env_g = input_ppm[t_y][s_x][1];
595                                     env_b = input_ppm[t_y][s_x][2];
596                                 }
597
598
599
600
601                                 //zがzバッファの該当する値より大きければ描画を行わない（何もしない）
602                                 if(z_buf[i][j] < p_or[2]){}
603
604                                 else{
605                                     image[i][j][0] =  env_r;
606
607                                     image[i][j][1] =  env_g;
608
609                                     image[i][j][2] =  env_b;
610
611                                     z_buf[i][j] = p_or[2];
612                                 }
613                             }
614                         }
615                         //撮像平面からはみ出る部分は描画しない
616                         else{}
617                     }
618                 }
619
620         }
```

```
621             //分割できる
622             //分割してそれぞれ再帰的に処理
623             //分割後の三角形はpp2qとpp2r
624             else{
625                 double p2[2];
626                 p2[0] = func1(q, r, p[1]);
627                 p2[1] = p[1];
628 
629                 double P2[3];
630                 double k =
631                     ((poly_i_n_vec[0] * (A[0] - camera_xyz[0])) +
632                      (poly_i_n_vec[1] * (A[1] - camera_xyz[1])) +
633                      (poly_i_n_vec[2] * (A[2] - camera_xyz[2])))
634                     /
635                     ((poly_i_n_vec[0] * ((p2[0]-(WIDTH/2)) - camera_xyz[0])) +
636                      (poly_i_n_vec[1] * ((p2[1]-(HEIGHT/2)) - camera_xyz[1])) +
637                      (poly_i_n_vec[2] * (FOCUS - camera_xyz[2])));
638 
639                 P2[0] = k * ((p2[0]-(WIDTH/2)) - camera_xyz[0]) + camera_xyz[0];
640                 P2[1] = k * ((p2[1]-(HEIGHT/2)) - camera_xyz[1]) + camera_xyz[1];
641                 P2[2] = k * (FOCUS - camera_xyz[2]) + camera_xyz[2];
642 
643                 double n_p2[3];
644                 for(int i = 0; i < 3; i++){
645                     n_p2[i]
646                         =
647                         n_q[i] * ((p[1]-r[1])/(q[1]-r[1]))
648                         +
649                         n_r[i] * ((q[1]-p[1])/(q[1]-r[1]));
650                 }
651 
652 
653 
654                 //p2のほうがpのx座標より大きくなるようにする
655                 if(p2[0] < p[0]){
656                     double temp[2];
657                     double temp_n[3];
658 
659                     memcpy(temp, p2, sizeof(double) * 2);
660                     memcpy(p2, p, sizeof(double) * 2);
661                     memcpy(p, temp, sizeof(double) * 2);
662 
663                     memcpy(temp_n, n_p2, sizeof(double) * 2);
664                     memcpy(n_p2, n_p, sizeof(double) * 2);
665                     memcpy(n_p, temp_n, sizeof(double) * 2);
666                 }
667                 shading(p, p2, q, n_p, n_p2, n_q, P, P2, Q, poly_i_n_vec, input_ppm);
668                 shading(p, p2, r, n_p, n_p2, n_r, P, P2, R, poly_i_n_vec, input_ppm);
669             }
670         }
671     }
672 
673 
674 
675 
676 
677 /* VRML 2.0 Reader
678  *
679  * ver1.1 2005/10/06 Masaaki IIYAMA (bug fix)
680  * ver1.0 2005/09/27 Masaaki IIYAMA
681  *
682  */
683 
684 #include <stdio.h>
685 #include <stdlib.h>
686 #include <ctype.h>
687 #include "vrml.h"
688 
689 
690 /*
691 ////////////////////////////////////////////////////////////////
692 */
693 #define MWS 256
694 
695 static int strindex( char *s, char *t)
696 {
697     int         i, j, k;
698 
699     for (i = 0; s[i] != '\0'; i++) {
```

```
        for (j = i, k = 0;  t[k] != '\0' && s[j] == t[k]; j++, k++) ;
        if (k > 0 && t[k] == '\0')
            return i;
    }
    return -1;
}

static int getword(
                    FILE *fp,
                    char word[],
                    int sl)
{
    int i,c;

    while ( (c = fgetc(fp)) != EOF && ( isspace(c) || c == '#')) {
        if ( c == '#' ) {
            while ( (c = fgetc(fp)) != EOF && c != '\n') ;
            if ( c == EOF ) return (0);
        }
    }
    if ( c == EOF )
        return (0);
    ungetc(c,fp);

    for ( i = 0 ; i < sl - 1 ; i++) {
        word[i] = fgetc(fp);
        if ( isspace(word[i]) )
            break;
    }
    word[i] = '\0';

    return i;
}

static int read_material(
                         FILE *fp,
                         Surface *surface,
                         char *b)
{
    while (getword(fp,b,MWS)>0) {
        if      (strindex(b,"}")>=0) break;
        else if (strindex(b,"diffuseColor") >= 0) {
            getword(fp,b,MWS);
            surface->diff[0] = atof(b);
            getword(fp,b,MWS);
            surface->diff[1] = atof(b);
            getword(fp,b,MWS);
            surface->diff[2] = atof(b);
        }
        else if (strindex(b,"ambientIntensity") >= 0) {
            getword(fp,b,MWS);
            surface->ambi = atof(b);
        }
        else if (strindex(b,"specularColor") >= 0) {
            getword(fp,b,MWS);
            surface->spec[0] = atof(b);
            getword(fp,b,MWS);
            surface->spec[1] = atof(b);
            getword(fp,b,MWS);
            surface->spec[2] = atof(b);
        }
        else if (strindex(b,"shininess") >= 0) {
            getword(fp,b,MWS);
            surface->shine = atof(b);
        }
    }
    return 1;
}

static int count_point(
                       FILE *fp,
                       char *b)
{
    int num=0;
    while (getword(fp,b,MWS)>0) {
        if (strindex(b,"[")>=0) break;
    }
    while (getword(fp,b,MWS)>0) {
        if (strindex(b,"]")>=0) break;
```

```
779         else {
780             num++;
781         }
782     }
783     if ( num %3 != 0 ) {
784         fprintf(stderr,"invalid␣file␣type[number␣of␣points␣mismatch]\n");
785     }
786     return num/3;
787 }
788 
789 static int read_point(
790                       FILE *fp,
791                       Polygon *polygon,
792                       char *b)
793 {
794     int num=0;
795     while (getword(fp,b,MWS)>0) {
796         if (strindex(b,"[")>=0) break;
797     }
798     while (getword(fp,b,MWS)>0) {
799         if (strindex(b,"]")>=0) break;
800         else {
801             polygon->vtx[num++] = atof(b);
802         }
803     }
804     return num/3;
805 }
806 
807 static int count_index(
808                        FILE *fp,
809                        char *b)
810 {
811     int num=0;
812     while (getword(fp,b,MWS)>0) {
813         if (strindex(b,"[")>=0) break;
814     }
815     while (getword(fp,b,MWS)>0) {
816         if (strindex(b,"]")>=0) break;
817         else {
818             num++;
819         }
820     }
821     if ( num %4 != 0 ) {
822         fprintf(stderr,"invalid␣file␣type[number␣of␣indices␣mismatch]\n");
823     }
824     return num/4;
825 }
826 
827 static int read_index(
828                       FILE *fp,
829                       Polygon *polygon,
830                       char *b)
831 {
832     int num=0;
833     while (getword(fp,b,MWS)>0) {
834         if (strindex(b,"[")>=0) break;
835     }
836     while (getword(fp,b,MWS)>0) {
837         if (strindex(b,"]")>=0) break;
838         else {
839             polygon->idx[num++] = atoi(b);
840             if (num%3 == 0) getword(fp,b,MWS);
841         }
842     }
843     return num/3;
844 }
845 
846 int read_one_obj(
847                  FILE *fp,
848                  Polygon *poly,
849                  Surface *surface)
850 {
851     char b[MWS];
852     int flag_material = 0;
853     int flag_point = 0;
854     int flag_index = 0;
855 
856     /* initialize surface */
857     surface->diff[0] = 1.0;
```

```
858     surface->diff[1] = 1.0;
859     surface->diff[2] = 1.0;
860     surface->spec[0] = 0.0;
861     surface->spec[1] = 0.0;
862     surface->spec[2] = 0.0;
863     surface->ambi = 0.0;
864     surface->shine = 0.2;
865 
866     if ( getword(fp,b,MWS) <= 0) return 0;
867 
868     poly->vtx_num = 0;
869     poly->idx_num = 0;
870 
871     while (flag_material==0 || flag_point==0 || flag_index==0) {
872         if      (strindex(b,"Material")>=0) {
873             getword(fp,b,MWS);
874             flag_material = 1;
875         }
876         else if (strindex(b,"point")>=0) {
877             fprintf(stderr,"Counting...␣[point]\n");
878             poly->vtx_num = count_point(fp, b);
879             flag_point = 1;
880         }
881         else if (strindex(b,"coordIndex")>=0) {
882             fprintf(stderr,"Counting...␣[coordIndex]\n");
883             poly->idx_num = count_index(fp, b);
884             flag_index = 1;
885         }
886         else if (getword(fp,b,MWS) <= 0) return 0;
887     }
888 
889     flag_material = 0;
890     flag_point = 0;
891     flag_index = 0;
892 
893     fseek(fp, 0, SEEK_SET);
894     poly->vtx = (double *)malloc(sizeof(double)*3*poly->vtx_num);
895     poly->idx = (int *)malloc(sizeof(int)*3*poly->idx_num);
896     while (flag_material==0 || flag_point==0 || flag_index==0) {
897         if      (strindex(b,"Material")>=0) {
898             fprintf(stderr,"Reading...␣[Material]\n");
899             read_material(fp,surface,b);
900             flag_material = 1;
901         }
902         else if (strindex(b,"point")>=0) {
903             fprintf(stderr,"Reading...␣[point]\n");
904             read_point(fp,poly,b);
905             flag_point = 1;
906         }
907         else if (strindex(b,"coordIndex")>=0) {
908             fprintf(stderr,"Reading...␣[coordIndex]\n");
909             read_index(fp,poly,b);
910             flag_index = 1;
911         }
912         else if (getword(fp,b,MWS) <= 0) return 0;
913     }
914 
915     return 1;
916 }
917 
918 
919 int main (int argc, char *argv[]){
920     /* VRML読み込み ============================================================*/
921     int i;
922     FILE *fp;
923     Polygon poly;
924     Surface surface;
925 
926     fp = fopen(argv[1], "r");
927     read_one_obj(fp, &poly, &surface);
928 
929     fprintf(stderr,"%d␣vertice␣are␣found.(poly.vtx_num)\n",poly.vtx_num);
930     fprintf(stderr,"%d␣triangles␣are␣found.(poly.idx_num)\n",poly.idx_num);
931 
932     //i th vertex
933     printf("\npoly.vtx[i*3+0,1,2]\n");
934     for ( i = 0 ; i < poly.vtx_num ; i++ ) {
935         fprintf(stdout,"%f␣%f␣%f␣#␣%d␣th␣vertex\n",
936                 poly.vtx[i*3+0], poly.vtx[i*3+1], poly.vtx[i*3+2],
```

```
937                 i);
938     }
939
940     //i th triangle
941     printf("\npoly.idx[i*3+0,1,2]\n");
942     for ( i = 0 ; i < poly.idx_num ; i++ ) {
943         fprintf(stdout,"%d␣%d␣%d␣#␣%d␣th␣triangle\n",
944                 poly.idx[i*3+0], poly.idx[i*3+1], poly.idx[i*3+2],
945                 i);
946     }
947
948     /* material info */
949     fprintf(stderr, "diffuseColor␣%f␣%f␣%f\n", surface.diff[0], surface.diff[1], surface.diff[2]);
950     fprintf(stderr, "specularColor␣%f␣%f␣%f\n", surface.spec[0], surface.spec[1], surface.spec[2]);
951     fprintf(stderr, "ambientIntensity␣%f\n", surface.ambi);
952     fprintf(stderr, "shininess␣%f\n", surface.shine);
953
954     /* VRML読み込みここまで ========================================================*/
955
956
957     /* 環境マップppmファイルの読み込み ================================ */
958     char *map_fname = MAP_FILENAME;
959     FILE *ip;
960     ip = fopen(map_fname,"r");
961     if(ip == NULL){
962         fprintf(stderr, "%sを正常に開くことが出来ませんでした.\n" ,MAP_FILENAME);
963         exit(1);/*異常終了*/
964     }
965
966     printf("loading␣%s...\n",MAP_FILENAME);
967
968     char buf[MAX];
969     //実装上読み込むppmの形式を以下のように制限する
970     /* P3\n */
971     /* WIDTH HEIGHT\n */ //(空白で区切る)
972     /* 255\n */
973     /* ..... */
974     //と指定
975     //画素値についてはrgbの3つの数値の間に改行を挟むことを許さない
976     //仕様書通り幅と高さは空白含めて70文字までとする
977     //コメントは実装しない
978
979     //マジックナンバーを取得=======================
980     fgets(buf,70,ip);
981     char *magic_num = strtok(buf, "\n");
982     printf("magic␣number␣is␣%s.\n", magic_num);
983     //==========================================
984
985     //WIDTH、HEIGHTを取得 ==========================
986     fgets(buf,70,ip);
987     ppm_width = atoi(strtok(buf, "␣"));
988     printf("width␣is␣%d.\n", ppm_width);
989     ppm_height = atoi(strtok(NULL, "\n"));
990     printf("height␣is␣%d.\n", ppm_height);
991     //==========================================
992
993     //上限値を取得================================
994     fgets(buf,70,ip);
995     ppm_max = atoi(strtok(buf, "\n"));
996     printf("max␣is␣%d.\n", ppm_max);
997     //==========================================
998
999     //リストの先頭ポインタ
1000    LIST *head, *tail;
1001    head = NULL;
1002    tail = NULL;
1003
1004    int num;
1005    int index = 0;
1006    char char_buf[256];
1007    int flag = 0;
1008
1009    while (1){
1010        num = fgetc(ip);
1011        char reset[] = "";
1012        //空白判定
1013        //空白のとき
1014        if(isspace(num) != 0 || num == EOF){
1015            //直前に読み込んだ文字が空白のとき
```

```
1016            if(flag == 1){
1017                /* ループから抜ける */
1018                if(num == EOF){
1019                    break;
1020                }
1021            }
1022            //直前に読み込んだ文字が空白でないとき
1023            else{
1024                //先頭、最後尾をセット
1025                if(index == 0){
1026                    LIST *p;
1027                    /* 記憶領域の確保 */
1028                    if ((p = (LIST *) malloc(sizeof(LIST))) == NULL) {
1029                        printf("malloc␣error\n");
1030                        exit(EXIT_FAILURE);
1031                    }
1032                    /* リストにデータを登録 */
1033                    p->num = atoi(char_buf);
1034                    p->index = index;
1035                    printf("index␣=␣%d␣num␣=␣%d\n", p->index, p->num);
1036
1037                    /* ポインタのつなぎ換え */
1038                    p->next = NULL;
1039                    tail = p;
1040                    head = p;
1041                    /* ループから抜ける */
1042                    if(num == EOF){
1043                        break;
1044                    }
1045                }
1046                else{
1047                    tail = add_list(atoi(char_buf), index, tail);
1048                }
1049                index ++;
1050                memcpy(char_buf, reset, sizeof(char) * 256);
1051                flag = 1;
1052            }
1053        }
1054        //空白以外のとき（数字のはず）
1055        else{
1056            flag = 0;
1057            sprintf(buf, "%c", num);
1058            strcat(char_buf, buf);
1059        }
1060    }
1061    fclose(ip);
1062    printf("completed␣processing␣%s\n",MAP_FILENAME);
1063
1064    //debug
1065    //show_list(head);
1066
1067    //取り込んで環境マッピングに使用するppmの保存領域内を確保
1068    int input_ppm[ppm_height][ppm_width][3];
1069    //LISTを通常の配列に変換 ==================================
1070    LIST *p = head;
1071
1072    while (p->next != NULL) {
1073        //通常の画像viewerはヘッダを見て256*256であれば
1074        //それ以降の余分な数値は無視する
1075        int max_index = (ppm_height*ppm_width*3)-1;
1076        if(max_index < (p->index)){
1077            break;
1078        }
1079        div_t d1 = div(p->index, 3);
1080        div_t d2 = div(d1.quot, ppm_width);
1081
1082        input_ppm[d2.quot][d2.rem][d1.rem] = p->num;
1083        p = p->next;
1084    }
1085    /* 環境マップppmファイルの読み込みここまで ================================== */
1086
1087    FILE *fp_ppm;
1088    char *fname = argv[2];
1089
1090
1091    fp_ppm = fopen( fname, "w" );
1092    //ファイルが開けなかったとき
1093    if( fp_ppm == NULL ){
1094        printf("%sファイルが開けません.\n", fname);
```

```
1095            return -1;
1096        }
1097
1098        //ファイルが開けたとき
1099        else{
1100            //描画領域を初期化
1101            for(int i = 0; i < 256; i++){
1102                for(int j = 0; j < 256; j++){
1103                    image[i][j][0] = 0.0 * MAX;
1104                    image[i][j][1] = 0.0 * MAX;
1105                    image[i][j][2] = 0.0 * MAX;
1106                }
1107            }
1108
1109             //zバッファを初期化
1110            for(int i = 0; i < 256; i++){
1111                for(int j = 0; j < 256; j++){
1112                    z_buf[i][j] = DBL_MAX;
1113                }
1114            }
1115
1116            //diffuse_colorの格納
1117            diffuse_color[0] = surface.diff[0];
1118            diffuse_color[1] = surface.diff[1];
1119            diffuse_color[2] = surface.diff[2];
1120
1121            //shininessの格納
1122            //!!!!!!!!!!!!!!注意!!!!!!!!!!!!!!!!!
1123            //(実験ページの追加情報を参照)
1124            //各ファイルのshininessの値は
1125            //av4 0.5
1126            //av5 0.5
1127            //iiyama1997 1.0
1128            //aa053 1.0
1129            //av007 0.34
1130
1131            shininess = surface.shine * 128;
1132
1133            //speculorColorの格納
1134            specular_color[0] = surface.spec[0];
1135            specular_color[1] = surface.spec[1];
1136            specular_color[2] = surface.spec[2];
1137
1138            //カメラ位置の取り込み
1139            camera_xyz[0] = atoi(argv[3]);
1140            camera_xyz[1] = 0.0;
1141            camera_xyz[2] = 0.0;
1142            printf("camera␣is␣at␣(%f,␣%f,␣%f)\n", camera_xyz[0], camera_xyz[1], camera_xyz[2]);
1143
1144            /* 全三角形の法線ベクトルを格納========================================================== */
1145
1146            //各頂点の法線ベクトルを求める
1147            //三角形iの法線ベクトルを求めて配列に格納する(グローバル領域に保存)
1148            double poly_n[poly.idx_num * 3];
1149            //三角形iは3点A、B、Cからなる
1150            //この3点で形成される三角形の法線ベクトルを求めてpoly_nに格納していく
1151            for(int i = 0; i < poly.idx_num; i++){
1152                //三角形iの各頂点の座標
1153                double A[3], B[3], C[3];
1154                A[0] = poly.vtx[(poly.idx[i*3+0])*3 + 0];
1155                A[1] = poly.vtx[(poly.idx[i*3+0])*3 + 1];
1156                A[2] = poly.vtx[(poly.idx[i*3+0])*3 + 2];
1157
1158                B[0] = poly.vtx[(poly.idx[i*3+1])*3 + 0];
1159                B[1] = poly.vtx[(poly.idx[i*3+1])*3 + 1];
1160                B[2] = poly.vtx[(poly.idx[i*3+1])*3 + 2];
1161
1162                C[0] = poly.vtx[(poly.idx[i*3+2])*3 + 0];
1163                C[1] = poly.vtx[(poly.idx[i*3+2])*3 + 1];
1164                C[2] = poly.vtx[(poly.idx[i*3+2])*3 + 2];
1165
1166                //ベクトルAB, ACから外積を計算して
1167                //法線ベクトルnを求める
1168                double AB[3], AC[3], n[3];
1169                AB[0] = B[0] - A[0];
1170                AB[1] = B[1] - A[1];
1171                AB[2] = B[2] - A[2];
1172
1173                AC[0] = C[0] - A[0];
```

```
1174            AC[1] = C[1] - A[1];
1175            AC[2] = C[2] - A[2];
1176
1177            n[0] = (AB[1] * AC[2]) - (AB[2] * AC[1]);
1178            n[1] = (AB[2] * AC[0]) - (AB[0] * AC[2]);
1179            n[2] = (AB[0] * AC[1]) - (AB[1] * AC[0]);
1180
1181            //長さを1に調整
1182            double length_n =
1183                sqrt(pow(n[0], 2.0) +
1184                     pow(n[1], 2.0) +
1185                     pow(n[2], 2.0));
1186
1187            n[0] = n[0] / length_n;
1188            n[1] = n[1] / length_n;
1189            n[2] = n[2] / length_n;
1190
1191            poly_n[i*3 + 0] = n[0];
1192            poly_n[i*3 + 1] = n[1];
1193            poly_n[i*3 + 2] = n[2];
1194        }
1195
1196        //三角形iの法線ベクトルがpoly_nに格納された.
1197        //debug
1198        printf("\npoly_n\n");
1199        for(int i = 0 ; i < poly.idx_num ; i++){
1200            fprintf(stdout,"%f␣%f␣%f␣#␣%d␣th␣triangle\n",
1201                    poly_n[i*3+0], poly_n[i*3+1], poly_n[i*3+2],
1202                    i);
1203        }
1204        /* ここまで全三角形の法線ベクトルを格
               納========================================================== */
1205
1206
1207
1208
1209        /* 各点の平均、正規化した法線ベクトルを格納================================================
               */
1210        //点iの法線ベクトルをもとめて専用の配列に格納する
1211        //頂点iの法線ベクトルは
1212        //(poly_ave_i[i*3+0], poly_ave_i[i*3+1], poly_ave_i[i*3+2])
1213        double poly_ave_i[poly.vtx_num];
1214        //点iが隣接する平面を探索
1215        for(int i = 0; i < poly.vtx_num; i++){
1216            double sum_vec[3] = {0.0, 0.0, 0.0};
1217            int count = 0;
1218            //三角形jの中に頂点iが含まれるかを判定
1219            for(int j = 0; j < poly.idx_num; j++){
1220                //プログラムの可読性を保つためバラして書く
1221                if(poly.idx[j*3+0] == i||
1222                   poly.idx[j*3+1] == i||
1223                   poly.idx[j*3+2] == i){
1224                    sum_vec[0] =  sum_vec[0] + poly_n[j*3+0];
1225                    sum_vec[1] =  sum_vec[1] + poly_n[j*3+1];
1226                    sum_vec[2] =  sum_vec[2] + poly_n[j*3+2];
1227                    count++;
1228                }
1229            }
1230            //点iの法線ベクトルを隣接平面の法線ベクトルの平均を正規化して計算する
1231            double ni_vec[3];
1232            if(count == 0){
1233                printf("\n␣warning!!␣1128\n");
1234                printf("\n␣i␣=␣%d\n" ,i);
1235                exit(0);
1236            }
1237            ni_vec[0] = sum_vec[0] / count;
1238            ni_vec[1] = sum_vec[1] / count;
1239            ni_vec[2] = sum_vec[2] / count;
1240
1241            double length_ni_vec =
1242                sqrt(pow(ni_vec[0], 2.0)+
1243                     pow(ni_vec[1], 2.0)+
1244                     pow(ni_vec[2], 2.0));
1245            if(length_ni_vec == 0){
1246                printf("\n␣warning!!␣1129\n");
1247                exit(0);
1248            }
1249            ni_vec[0] = ni_vec[0] / length_ni_vec;
1250            ni_vec[1] = ni_vec[1] / length_ni_vec;
```

```
1251            ni_vec[2] = ni_vec[2] / length_ni_vec;
1252
1253            //頂点iの法線ベクトルを格納
1254            poly_ave_i[i*3+0] = ni_vec[0];
1255            poly_ave_i[i*3+1] = ni_vec[1];
1256            poly_ave_i[i*3+2] = ni_vec[2];
1257
1258            //debug
1259            double length_ply_ave =
1260                sqrt(pow(poly_ave_i[i*3+0], 2.0)+
1261                     pow(poly_ave_i[i*3+1], 2.0)+
1262                     pow(poly_ave_i[i*3+2], 2.0));
1263            if(length_ply_ave == 0){
1264                printf("\n␣warning!!␣1151␣\n");
1265                exit(0);
1266            }
1267        }
1268        //点iの法線ベクトルがpoly_ave_iに格納された.
1269        //debug
1270        printf("\npoly_ave_i\n");
1271        for(int i = 0 ; i < poly.idx_num ; i++){
1272            fprintf(stdout,"%f␣%f␣%f␣#␣%d␣th␣vertex\n",
1273                    poly_ave_i[i*3+0], poly_ave_i[i*3+1], poly_ave_i[i*3+2],
1274                    i);
1275        }
1276        /* 各点の平均、正規化した法線ベクトルを格納ここまで==========================================*/
1277
1278        //シェーディング
1279        //ポリゴンiをシェーディング==========================================================================
1280        //  ==================================================================================================
1281
1282        for(int i = 0; i < poly.idx_num; i++){
1283            //三角形の各点の透視投影処理==========================================
1284            for(int j = 0; j < 3; j++){
1285                double xp = poly.vtx[(poly.idx[i*3+j])*3 + 0];
1286                double yp = poly.vtx[(poly.idx[i*3+j])*3 + 1];
1287                double zp = poly.vtx[(poly.idx[i*3+j])*3 + 2];
1288                double zi = FOCUS;
1289                //debug
1290                if(zp == 0){
1291                    printf("\n(%f\t%f\t%f)␣i=%d,␣j=%d\n", xp, yp, zp, i, j);
1292                    perror("\nエラー0934\n");
1293                    exit(0);
1294                    //break;
1295                }
1296
1297                double xp2 = xp * ((zi - camera_xyz[2]) / (zp - camera_xyz[2]));
1298                double yp2 = yp * ((zi - camera_xyz[2]) / (zp - camera_xyz[2]));
1299                double zp2 = zi;
1300
1301                //座標軸を平行移動
1302                projected_ver_buf[j][0] = (WIDTH / 2) + xp2;
1303                projected_ver_buf[j][1] = (HEIGHT / 2) + yp2;
1304            }
1305
1306
1307            double a[2], b[2], c[2];
1308            a[0] = projected_ver_buf[0][0];
1309            a[1] = projected_ver_buf[0][1];
1310            b[0] = projected_ver_buf[1][0];
1311            b[1] = projected_ver_buf[1][1];
1312            c[0] = projected_ver_buf[2][0];
1313            c[1] = projected_ver_buf[2][1];
1314
1315            //点a、b、cがそれぞれ何番目の頂点かを参照
1316
1317            int index_a = poly.idx[i*3+0];
1318            int index_b = poly.idx[i*3+1];
1319            int index_c = poly.idx[i*3+2];
1320
1321            //関数shadingの中では3点の空間内での座標も必要
1322            double A[3], B[3], C[3];
1323            A[0] = poly.vtx[index_a*3 + 0];
1324            A[1] = poly.vtx[index_a*3 + 1];
1325            A[2] = poly.vtx[index_a*3 + 2];
```

```
1326
1327                B[0] = poly.vtx[index_b*3 + 0];
1328                B[1] = poly.vtx[index_b*3 + 1];
1329                B[2] = poly.vtx[index_b*3 + 2];
1330
1331                C[0] = poly.vtx[index_c*3 + 0];
1332                C[1] = poly.vtx[index_c*3 + 1];
1333                C[2] = poly.vtx[index_c*3 + 2];
1334                //三角形 i のシェーディングを行う
1335
1336                double n_a[3], n_b[3], n_c[3];
1337                n_a[0] = poly_ave_i[index_a*3+0];
1338                n_a[1] = poly_ave_i[index_a*3+1];
1339                n_a[2] = poly_ave_i[index_a*3+2];
1340                n_b[0] = poly_ave_i[index_b*3+0];
1341                n_b[1] = poly_ave_i[index_b*3+1];
1342                n_b[2] = poly_ave_i[index_b*3+2];
1343                n_c[0] = poly_ave_i[index_c*3+0];
1344                n_c[1] = poly_ave_i[index_c*3+1];
1345                n_c[2] = poly_ave_i[index_c*3+2];
1346
1347                //三角形 i の（本来の）法線ベクトルは
1348                //(poly_n[i*3+0], poly_n[i*3+1], poly_n[i*3+2])
1349                double poly_i_n_vec[3]
1350                    = {poly_n[i*3+0], poly_n[i*3+1], poly_n[i*3+2]};
1351
1352                shading(a, b, c, n_a, n_b, n_c, A, B, C, poly_i_n_vec, input_ppm);
1353            }
1354
1355            //ヘッダー出力
1356            fputs(MAGICNUM, fp_ppm);
1357            fputs("\n", fp_ppm);
1358            fputs(WIDTH_STRING, fp_ppm);
1359            fputs("␣", fp_ppm);
1360            fputs(HEIGHT_STRING, fp_ppm);
1361            fputs("\n", fp_ppm);
1362            fputs(MAX_STRING, fp_ppm);
1363            fputs("\n" ,fp_ppm);
1364
1365            //imageの出力
1366            for(int i = 0; i < 256; i++){
1367                for(int j = 0; j < 256; j++){
1368                    char r[256];
1369                    char g[256];
1370                    char b[256];
1371                    char str[1024];
1372                    sprintf(r, "%d", (int)round(image[i][j][0]));
1373                    sprintf(g, "%d", (int)round(image[i][j][1]));
1374                    sprintf(b, "%d", (int)round(image[i][j][2]));
1375                    sprintf(str, "%s\t%s\t%s\n", r, g, b);
1376                    fputs(str, fp_ppm);
1377                }
1378            }
1379        }
1380        fclose(fp_ppm);
1381        fclose(fp);
1382
1383        printf("\nppmファイル␣%s␣の作成が完了しました.\n", fname );
1384        return 1;
1385    }
```

5 実行例

kadai05.c と同一のディレクトリに次のプログラムを置き、

リスト 4 EvalKadai05.sh

```
1  #!/bin/sh
2  SRC=kadai05.c
3
4  WRL=sample/av5.wrl
5
6  PPM0=Kadai05ForAv5-0.ppm
7  CAMERA0=0.0
```

```
8
9  PPM1=Kadai05ForAv5-1.ppm
10 CAMERA1=50.0
11
12 PPM2=Kadai05ForAv5-2.ppm
13 CAMERA2=-50.0
14
15 gcc -Wall $SRC
16 ./a.out $WRL $PPM0 $CAMERA0
17 open $PPM0
18
19 ./a.out $WRL $PPM1 $CAMERA1
20 open $PPM1
21
22 ./a.out $WRL $PPM2 $CAMERA2
23 open $PPM2
24 echo completed!! "\xF0\x9f\x8d\xbb"
```

さらに同一ディレクトリ内のディレクトリ sample の中に対象とする VRML ファイル、環境マップを置いて、

```
1 $ sh EvalKadai05.sh
```

を実行した. 出力画像は図 2、図 3、図 4 のようになった.

図 2　カメラの x 座標が 0.0 の時の av5 の出力結果

図 3　カメラの x 座標が 50.0 の時の av5 の出力結果

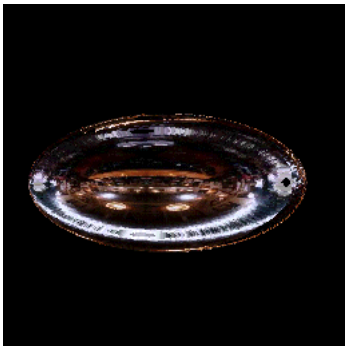

図 4　カメラの x 座標が-50.0 の時の av5 の出力結果