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 12 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.

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      //printf("check x = %f\n", x);
73      //printf("check p[0] = %f\n", p[0]);
74      return x;
75  }
76
77  //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
78  //1直線上に無ければreturn 0;
79  //1直線上にあればreturn 1;
80  int lineOrNot(double *a, double *b, double *c){
81      if(a[0] == b[0]){
82          if(a[0] == c[0]){
83              return 1;
84          }
85          else{
86              return 0;
87          }
88      }
89      else{
90          if(c[1] == a[1] + ((b[1] - a[1]) / (b[0] - a[0])) * (c[0] - a[0])){
91              return 1;
92          }
93          else{
94              return 0;
95          }
96      }
97  }
98
99
100 //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
101 void shading(double *a, double *b, double *c,
102              double *rgb_a, double *rgb_b, double *rgb_c,
103              double *A, double *B, double *C,
104              double *poly_i_n_vec){
105     //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
```

```
106     if(lineOrNot(a, b, c) == 1){}
107     else{
108         //y座標の値が真ん中点をp、その他の点をq、rとする
109         //y座標の大きさはr <= p <= qの順
110         double p[2], q[2], r[2];
111         //法線ベクトルも名前を変更する
112         double rgb_p[3], rgb_q[3], rgb_r[3];
113         //空間内での元の座標についても名前を変更する
114         double P[3], Q[3], R[3];
115 
116         if(b[1] <= a[1] && a[1] <= c[1]){
117             memcpy(p, a, sizeof(double) * 2);
118             memcpy(q, c, sizeof(double) * 2);
119             memcpy(r, b, sizeof(double) * 2);
120 
121             memcpy(rgb_p, rgb_a, sizeof(double) * 3);
122             memcpy(rgb_q, rgb_c, sizeof(double) * 3);
123             memcpy(rgb_r, rgb_b, sizeof(double) * 3);
124 
125             memcpy(P, A, sizeof(double) * 3);
126             memcpy(Q, C, sizeof(double) * 3);
127             memcpy(R, B, sizeof(double) * 3);
128         }
129         else{
130             if(c[1] <= a[1] && a[1] <= b[1]){
131                 memcpy(p, a, sizeof(double) * 2);
132                 memcpy(q, b, sizeof(double) * 2);
133                 memcpy(r, c, sizeof(double) * 2);
134 
135                 memcpy(rgb_p, rgb_a, sizeof(double) * 3);
136                 memcpy(rgb_q, rgb_b, sizeof(double) * 3);
137                 memcpy(rgb_r, rgb_c, sizeof(double) * 3);
138 
139                 memcpy(P, A, sizeof(double) * 3);
140                 memcpy(Q, B, sizeof(double) * 3);
141                 memcpy(R, C, sizeof(double) * 3);
142 
143             }
144             else{
145                 if(a[1] <= b[1] && b[1] <= c[1]){
146                     memcpy(p, b, sizeof(double) * 2);
147                     memcpy(q, c, sizeof(double) * 2);
148                     memcpy(r, a, sizeof(double) * 2);
149 
150                     memcpy(rgb_p, rgb_b, sizeof(double) * 3);
151                     memcpy(rgb_q, rgb_c, sizeof(double) * 3);
152                     memcpy(rgb_r, rgb_a, sizeof(double) * 3);
153 
154                     memcpy(P, B, sizeof(double) * 3);
155                     memcpy(Q, C, sizeof(double) * 3);
156                     memcpy(R, A, sizeof(double) * 3);
157                 }
158                 else{
159                     if(c[1] <= b[1] && b[1] <= a[1]){
160                         memcpy(p, b, sizeof(double) * 2);
161                         memcpy(q, a, sizeof(double) * 2);
162                         memcpy(r, c, sizeof(double) * 2);
163 
164                         memcpy(rgb_p, rgb_b, sizeof(double) * 3);
165                         memcpy(rgb_q, rgb_a, sizeof(double) * 3);
166                         memcpy(rgb_r, rgb_c, sizeof(double) * 3);
167 
168                         memcpy(P, B, sizeof(double) * 3);
169                         memcpy(Q, A, sizeof(double) * 3);
170                         memcpy(R, C, sizeof(double) * 3);
171                     }
172                     else{
173                         if(b[1] <= c[1] && c[1] <= a[1]){
174                             memcpy(p, c, sizeof(double) * 2);
175                             memcpy(q, a, sizeof(double) * 2);
176                             memcpy(r, b, sizeof(double) * 2);
177 
178                             memcpy(rgb_p, rgb_c, sizeof(double) * 3);
179                             memcpy(rgb_q, rgb_a, sizeof(double) * 3);
180                             memcpy(rgb_r, rgb_b, sizeof(double) * 3);
181 
182                             memcpy(P, C, sizeof(double) * 3);
183                             memcpy(Q, A, sizeof(double) * 3);
184                             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]){
```

```
264                     //描画しない
265                 }
266                 else{
267                     image[temp_p1][temp_p0][0] = rgb_p[0];
268                     image[temp_p1][temp_p0][1] = rgb_p[1];
269                     image[temp_p1][temp_p0][2] = rgb_p[2];
270                 }
271
272                 int temp_q0 = ceil(q[0]);
273                 int temp_q1 = ceil(q[1]);
274                 if(z_buf[temp_q1][temp_q0] < Q[2]){
275                     //描画しない
276                 }
277                 else{
278                     image[temp_q1][temp_q0][0] = rgb_q[0];
279                     image[temp_q1][temp_q0][1] = rgb_q[1];
280                     image[temp_q1][temp_q0][2] = rgb_q[2];
281                 }
282
283                 int temp_r0 = ceil(r[0]);
284                 int temp_r1 = ceil(r[1]);
285                 if(z_buf[temp_r1][temp_r0] < R[2]){
286                     //描画しない
287                 }
288                 else{
289                     image[temp_r1][temp_r0][0] = rgb_r[0];
290                     image[temp_r1][temp_r0][1] = rgb_r[1];
291                     image[temp_r1][temp_r0][2] = rgb_r[2];
292                 }
293
294
295                 int i;
296                 i = ceil(p[1]);
297                 for(i;
298                     p[1] <= i && i <= q[1];
299                     i++){
300
301                     //撮像平面からはみ出ていないかのチェック
302                     if(0 <= i
303                        &&
304                        i <= (HEIGHT - 1)){
305                             double x1 = func1(p, q, i);
306                             double x2 = func1(r, q, i);
307                             int j;
308                             j = ceil(x1);
309
310                             for(j;
311                                 x1 <= j && j <= x2 && 0 <= j && j <= (WIDTH - 1);
312                                 j++){
313
314
315                                 //==========================================
316                                 //p[1] == r[1]
317                                 //描画する点の空間内でのz座標を計算
318                                 //計算時の法線ベクトルは
319                                 double p_z =
320                                     FOCUS
321                                     *
322                                     ((poly_i_n_vec[0]*A[0]) +
323                                      (poly_i_n_vec[1]*A[1]) +
324                                      (poly_i_n_vec[2]*A[2]))
325                                     /
326                                     ((poly_i_n_vec[0]*(j-(MAX/2))) +
327                                      (poly_i_n_vec[1]*(i-(MAX/2))) +
328                                      (poly_i_n_vec[2]*FOCUS));
329
330                                 //printf("\np_z = %f\n", p_z);
331
332
333                                 //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）
334                                 if(z_buf[i][j] < p_z){}
335
336                                 //Type 1
337                                 else{
338                                     image[i][j][0] =
339                                         (
340                                          ((x2-j) / (x2-x1))
341                                          *
342                                          ((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];
```

```
580             for(int i = 0; i < 3; i++){
581                 rgb_p2[i]
582                     =
583                     rgb_q[i] * ((p[1]-r[1])/(q[1]-r[1]))
584                     +
585                     rgb_r[i] * ((q[1]-p[1])/(q[1]-r[1]));
586             }
587
588
589
590             //p2のほうがpのx座標より大きくなるようにする
591             if(p2[0] < p[0]){
592                 double temp[2];
593                 double temp_rgb[3];
594
595                 memcpy(temp, p2, sizeof(double) * 2);
596                 memcpy(p2, p, sizeof(double) * 2);
597                 memcpy(p, temp, sizeof(double) * 2);
598
599                 memcpy(temp_rgb, rgb_p2, sizeof(double) * 2);
600                 memcpy(rgb_p2, rgb_p, sizeof(double) * 2);
601                 memcpy(rgb_p, temp_rgb, sizeof(double) * 2);
602             }
603             //分割しても同一平面上なので法線ベクトルと
604             //平面上の任意の点は同じものを使える.
605             //求める必要があるのはrgb_p2とP2
606
607             shading(p, p2, q, rgb_p, rgb_p2, rgb_q, P, P2, Q, poly_i_n_vec);
608             shading(p, p2, r, rgb_p, rgb_p2, rgb_r, P, P2, R, poly_i_n_vec);
609         }
610     }
611 }
612
613 /*
614 ////////////////////////////////////////////////////////////////
615 */
616 #define MWS 256
617
618 static int strindex( char *s, char *t)
619 {
620     int         i, j, k;
621
622     for (i = 0; s[i] != '\0'; i++) {
623         for (j = i, k = 0;  t[k] != '\0' && s[j] == t[k]; j++, k++) ;
624         if (k > 0 && t[k] == '\0')
625             return i;
626     }
627     return -1;
628 }
629
630 static int getword(
631                     FILE *fp,
632                     char word[],
633                     int sl)
634 {
635     int i,c;
636
637     while ( (c = fgetc(fp)) != EOF && ( isspace(c) || c == '#')) {
638         if ( c == '#' ) {
639             while ( (c = fgetc(fp)) != EOF && c != '\n') ;
640             if ( c == EOF ) return (0);
641         }
642     }
643     if ( c == EOF )
644         return (0);
645     ungetc(c,fp);
646
647     for ( i = 0 ; i < sl - 1 ; i++) {
648         word[i] = fgetc(fp);
649         if ( isspace(word[i]) )
650             break;
651     }
652     word[i] = '\0';
653
654     return i;
655 }
656
657 static int read_material(
658                         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;
        else {
            num++;
        }
    }
    if ( num %3 != 0 ) {
        fprintf(stderr,"invalid␣file␣type[number␣of␣points␣mismatch]\n");
    }
    return num/3;
}

static int read_point(
                       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->vtx[num++] = atof(b);
        }
    }
    return num/3;
}

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

```
738     while (getword(fp,b,MWS)>0) {
739         if (strindex(b,"]")>=0) break;
740         else {
741             num++;
742         }
743     }
744     if ( num %4 != 0 ) {
745         fprintf(stderr,"invalid␣file␣type[number␣of␣indices␣mismatch]\n");
746     }
747     return num/4;
748 }
749 
750 static int read_index(
751                       FILE *fp,
752                       Polygon *polygon,
753                       char *b)
754 {
755     int num=0;
756     while (getword(fp,b,MWS)>0) {
757         if (strindex(b,"[")>=0) break;
758     }
759     while (getword(fp,b,MWS)>0) {
760         if (strindex(b,"]")>=0) break;
761         else {
762             polygon->idx[num++] = atoi(b);
763             if (num%3 == 0) getword(fp,b,MWS);
764         }
765     }
766     return num/3;
767 }
768 
769 int read_one_obj(
770                  FILE *fp,
771                  Polygon *poly,
772                  Surface *surface)
773 {
774     char b[MWS];
775     int flag_material = 0;
776     int flag_point = 0;
777     int flag_index = 0;
778 
779     /* initialize surface */
780     surface->diff[0] = 1.0;
781     surface->diff[1] = 1.0;
782     surface->diff[2] = 1.0;
783     surface->spec[0] = 0.0;
784     surface->spec[1] = 0.0;
785     surface->spec[2] = 0.0;
786     surface->ambi = 0.0;
787     surface->shine = 0.2;
788 
789     if ( getword(fp,b,MWS) <= 0) return 0;
790 
791     poly->vtx_num = 0;
792     poly->idx_num = 0;
793 
794     while (flag_material==0 || flag_point==0 || flag_index==0) {
795         if      (strindex(b,"Material")>=0) {
796             getword(fp,b,MWS);
797             flag_material = 1;
798         }
799         else if (strindex(b,"point")>=0) {
800             fprintf(stderr,"Counting...␣[point]\n");
801             poly->vtx_num = count_point(fp, b);
802             flag_point = 1;
803         }
804         else if (strindex(b,"coordIndex")>=0) {
805             fprintf(stderr,"Counting...␣[coordIndex]\n");
806             poly->idx_num = count_index(fp, b);
807             flag_index = 1;
808         }
809         else if (getword(fp,b,MWS) <= 0) return 0;
810     }
811 
812     flag_material = 0;
813     flag_point = 0;
814     flag_index = 0;
815 
816     fseek(fp, 0, SEEK_SET);
```

```
817     poly->vtx = (double *)malloc(sizeof(double)*3*poly->vtx_num);
818     poly->idx = (int *)malloc(sizeof(int)*3*poly->idx_num);
819     while (flag_material==0 || flag_point==0 || flag_index==0) {
820         if      (strindex(b,"Material")>=0) {
821             fprintf(stderr,"Reading...␣[Material]\n");
822             read_material(fp,surface,b);
823             flag_material = 1;
824         }
825         else if (strindex(b,"point")>=0) {
826             fprintf(stderr,"Reading...␣[point]\n");
827             read_point(fp,poly,b);
828             flag_point = 1;
829         }
830         else if (strindex(b,"coordIndex")>=0) {
831             fprintf(stderr,"Reading...␣[coordIndex]\n");
832             read_index(fp,poly,b);
833             flag_index = 1;
834         }
835         else if (getword(fp,b,MWS) <= 0) return 0;
836     }
837
838     return 1;
839 }
840
841
842 int main (int argc, char *argv[])
843 {
844     int i;
845     FILE *fp;
846     Polygon poly;
847     Surface surface;
848
849     fp = fopen(argv[1], "r");
850     read_one_obj(fp, &poly, &surface);
851
852     fprintf(stderr,"%d␣vertice␣are␣found.(poly.vtx_num)\n",poly.vtx_num);
853     fprintf(stderr,"%d␣triangles␣are␣found.(poly.idx_num)\n",poly.idx_num);
854
855     //i th vertex
856     printf("\npoly.vtx[i*3+0,1,2]\n");
857     for ( i = 0 ; i < poly.vtx_num ; i++ ) {
858         fprintf(stdout,"%f␣%f␣%f␣#␣%d␣th␣vertex\n",
859                 poly.vtx[i*3+0], poly.vtx[i*3+1], poly.vtx[i*3+2],
860                 i);
861     }
862
863     //i th triangle
864     printf("\npoly.idx[i*3+0,1,2]\n");
865     for ( i = 0 ; i < poly.idx_num ; i++ ) {
866         fprintf(stdout,"%d␣%d␣%d␣#␣%d␣th␣triangle\n",
867                 poly.idx[i*3+0], poly.idx[i*3+1], poly.idx[i*3+2],
868                 i);
869     }
870
871     /* material info */
872     fprintf(stderr, "diffuseColor␣%f␣%f␣%f\n", surface.diff[0], surface.diff[1], surface.diff[2]);
873     fprintf(stderr, "specularColor␣%f␣%f␣%f\n", surface.spec[0], surface.spec[1], surface.spec[2]);
874     fprintf(stderr, "ambientIntensity␣%f\n", surface.ambi);
875     fprintf(stderr, "shininess␣%f\n", surface.shine);
876
877     //=====================================================================
878
879     FILE *fp_ppm;
880     char *fname = argv[2];
881
882
883     fp_ppm = fopen( fname, "w" );
884     //ファイルが開けなかったとき
885     if( fp_ppm == NULL ){
886         printf("%sファイルが開けません.\n", fname);
887         return -1;
888     }
889
890     //ファイルが開けたとき
891     else{
892         //描画領域を初期化
893         for(int i = 0; i < 256; i++){
894             for(int j = 0; j < 256; j++){
895                 image[i][j][0] = 0.0 * MAX;
```

```
896                 image[i][j][1] = 0.0 * MAX;
897                 image[i][j][2] = 0.0 * MAX;
898             }
899         }
900 
901          //zバッファを初期化
902         for(int i = 0; i < 256; i++){
903             for(int j = 0; j < 256; j++){
904                 z_buf[i][j] = DBL_MAX;
905             }
906         }
907 
908         //diffuse_colorの格納
909         diffuse_color[0] = surface.diff[0];
910         diffuse_color[1] = surface.diff[1];
911         diffuse_color[2] = surface.diff[2];
912 
913         //shininessの格納
914         //!!!!!!!!!!!!!!!!注意!!!!!!!!!!!!!!!!!
915         //(実験ページの追加情報を参照)
916         //各ファイルのshininessの値は
917         //av4 0.5
918         //av5 0.5
919         //iiyama1997 1.0
920         //aa053 1.0
921         //av007 0.34
922 
923         shininess = surface.shine * 128;
924 
925         //speculorColorの格納
926         specular_color[0] = surface.spec[0];
927         specular_color[1] = surface.spec[1];
928         specular_color[2] = surface.spec[2];
929 
930         //各頂点の法線ベクトルを求める
931         //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)
932         double poly_n[poly.idx_num * 3];
933 
934 
935         //=============================================================================
936         //三角形iは3点A、B、Cからなる
937         //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
938         for(int i = 0; i < poly.idx_num; i++){
939             //三角形iの各頂点の座標
940             double A[3], B[3], C[3];
941             A[0] = poly.vtx[(poly.idx[i*3+0])*3 + 0];
942             A[1] = poly.vtx[(poly.idx[i*3+0])*3 + 1];
943             A[2] = poly.vtx[(poly.idx[i*3+0])*3 + 2];
944 
945             B[0] = poly.vtx[(poly.idx[i*3+1])*3 + 0];
946             B[1] = poly.vtx[(poly.idx[i*3+1])*3 + 1];
947             B[2] = poly.vtx[(poly.idx[i*3+1])*3 + 2];
948 
949             C[0] = poly.vtx[(poly.idx[i*3+2])*3 + 0];
950             C[1] = poly.vtx[(poly.idx[i*3+2])*3 + 1];
951             C[2] = poly.vtx[(poly.idx[i*3+2])*3 + 2];
952 
953             //ベクトルAB, ACから外積を計算して
954             //法線ベクトルnを求める
955             double AB[3], AC[3], n[3];
956             AB[0] = B[0] - A[0];
957             AB[1] = B[1] - A[1];
958             AB[2] = B[2] - A[2];
959 
960             AC[0] = C[0] - A[0];
961             AC[1] = C[1] - A[1];
962             AC[2] = C[2] - A[2];
963 
964             n[0] = (AB[1] * AC[2]) - (AB[2] * AC[1]);
965             n[1] = (AB[2] * AC[0]) - (AB[0] * AC[2]);
966             n[2] = (AB[0] * AC[1]) - (AB[1] * AC[0]);
967 
968             //長さを1に調整
969             double length_n =
970                 sqrt(pow(n[0], 2.0) +
971                      pow(n[1], 2.0) +
972                      pow(n[2], 2.0));
973 
974             n[0] = n[0] / length_n;
```

```
975             n[1] = n[1] / length_n;
976             n[2] = n[2] / length_n;
977 
978             poly_n[i*3 + 0] = n[0];
979             poly_n[i*3 + 1] = n[1];
980             poly_n[i*3 + 2] = n[2];
981         }
982         //=========================================================================
983 
984 
985         //三角形 i の法線ベクトルが poly_n に格納された .
986         //debug
987         printf("\npoly_n\n");
988         for(int i = 0 ; i < poly.idx_num ; i++){
989             fprintf(stdout,"%f %f %f # %d th triangle\n",
990                     poly_n[i*3+0], poly_n[i*3+1], poly_n[i*3+2],
991                     i);
992         }
993 
994         //各点の平均、正規化した法線ベクトルを求める================================================
995         //点 i の法線ベクトルをもとめて専用の配列に格納する
996         //頂点 i の法線ベクトルは
997         //(poly_ave_i[i*3+0], poly_ave_i[i*3+1], poly_ave_i[i*3+2])
998         double poly_ave_i[poly.vtx_num];
999         //点 i が隣接する平面を探索
1000        for(int i = 0; i < poly.vtx_num; i++){
1001            double sum_vec[3] = {0.0, 0.0, 0.0};
1002            int count = 0;
1003            //三角形 j の中に頂点 i が含まれるかを判定
1004            for(int j = 0; j < poly.idx_num; j++){
1005                //プログラムの可読性を保つためバラして書く
1006                if(poly.idx[j*3+0] == i||
1007                   poly.idx[j*3+1] == i||
1008                   poly.idx[j*3+2] == i){
1009                    sum_vec[0] =  sum_vec[0] + poly_n[j*3+0];
1010                    sum_vec[1] =  sum_vec[1] + poly_n[j*3+1];
1011                    sum_vec[2] =  sum_vec[2] + poly_n[j*3+2];
1012                    count++;
1013                }
1014            }
1015            //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
1016            double ni_vec[3];
1017            if(count == 0){
1018                printf("\n warning!! 1128\n");
1019                printf("\n i = %d\n" ,i);
1020                exit(0);
1021            }
1022            ni_vec[0] = sum_vec[0] / count;
1023            ni_vec[1] = sum_vec[1] / count;
1024            ni_vec[2] = sum_vec[2] / count;
1025 
1026            double length_ni_vec =
1027                sqrt(pow(ni_vec[0], 2.0)+
1028                     pow(ni_vec[1], 2.0)+
1029                     pow(ni_vec[2], 2.0));
1030            if(length_ni_vec == 0){
1031                printf("\n warning!! 1129\n");
1032                exit(0);
1033            }
1034            ni_vec[0] = ni_vec[0] / length_ni_vec;
1035            ni_vec[1] = ni_vec[1] / length_ni_vec;
1036            ni_vec[2] = ni_vec[2] / length_ni_vec;
1037 
1038            //頂点 i の法線ベクトルを格納
1039            poly_ave_i[i*3+0] = ni_vec[0];
1040            poly_ave_i[i*3+1] = ni_vec[1];
1041            poly_ave_i[i*3+2] = ni_vec[2];
1042 
1043            //debug
1044            double length_ply_ave =
1045                sqrt(pow(poly_ave_i[i*3+0], 2.0)+
1046                     pow(poly_ave_i[i*3+1], 2.0)+
1047                     pow(poly_ave_i[i*3+2], 2.0));
1048            if(length_ply_ave == 0){
1049                printf("\n warning!! 1151 \n");
1050                exit(0);
1051            }
1052        }
1053        //=============================================================================================
```

```
1054
1055
1056            //点iの法線ベクトルがpoly_ave_iに格納された.
1057            //debug
1058            printf("\npoly_ave_i\n");
1059            for(int i = 0 ; i < poly.idx_num ; i++){
1060                fprintf(stdout,"%f␣%f␣%f␣#␣%d␣th␣vertex\n",
1061                        poly_ave_i[i*3+0], poly_ave_i[i*3+1], poly_ave_i[i*3+2],
1062                        i);
1063            }
1064
1065
1066
1067
1068            //各点の輝度値を決定する=========================================================================
1069            //点iの輝度値を専用の配列に格納
1070            double rgb_i[poly.vtx_num*3];
1071            //点iの輝度値は
1072            //(rgb_i[i*3+0], rgb_i[i*3+0], rgb_i[i*3+0],)
1073
1074            //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
1075            //同じになるので予め用意する==============
1076            double i_vec[3];
1077            i_vec[0] = light_dir[0];
1078            i_vec[1] = light_dir[1];
1079            i_vec[2] = light_dir[2];
1080            double length_i =
1081                sqrt(pow(i_vec[0], 2.0) + pow(i_vec[1], 2.0) + pow(i_vec[2], 2.0));
1082            if(length_i == 0){
1083                    printf("\n␣warning!␣11403\n");
1084                    exit(0);
1085                }
1086            i_vec[0] = (i_vec[0] / length_i);
1087            i_vec[1] = (i_vec[1] / length_i);
1088            i_vec[2] = (i_vec[2] / length_i);
1089            //====================================
1090
1091            for(int i = 0; i < poly.vtx_num; i++){
1092                //eベクトル=========================
1093                double e[3];
1094                e[0] = -1 * poly.vtx[i*3+0];
1095                e[1] = -1 * poly.vtx[i*3+1];
1096                e[2] = -1 * poly.vtx[i*3+2];
1097                double length_e =
1098                    sqrt(pow(e[0], 2.0) + pow(e[1], 2.0) + pow(e[2], 2.0));
1099                if(length_e == 0){
1100                    printf("\n␣warning!␣11400\n");
1101                    exit(0);
1102                }
1103                e[0] = (e[0] / length_e);
1104                e[1] = (e[1] / length_e);
1105                e[2] = (e[2] / length_e);
1106                //==================================
1107
1108                //sベクトル=========================
1109                double s[3];
1110                s[0] = e[0] - i_vec[0];
1111                s[1] = e[1] - i_vec[1];
1112                s[2] = e[2] - i_vec[2];
1113                double length_s =
1114                    sqrt(pow(s[0], 2.0) + pow(s[1], 2.0) + pow(s[2], 2.0));
1115                if(length_s == 0){
1116                    printf("\n␣warning!␣11401\n");
1117                    exit(0);
1118                }
1119                s[0] = (s[0] / length_s);
1120                s[1] = (s[1] / length_s);
1121                s[2] = (s[2] / length_s);
1122                //==================================
1123
1124
1125                //iベクトルとnベクトルの内積を計算
1126                double ip =
1127                    (poly_ave_i[i*3+0] * i_vec[0]) +
1128                    (poly_ave_i[i*3+1] * i_vec[1]) +
1129                    (poly_ave_i[i*3+2] * i_vec[2]);
1130
1131                if(0 <= ip){
1132                    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;
            }
```

```
1212
1213            double a[2], b[2], c[2];
1214            a[0] = projected_ver_buf[0][0];
1215            a[1] = projected_ver_buf[0][1];
1216            b[0] = projected_ver_buf[1][0];
1217            b[1] = projected_ver_buf[1][1];
1218            c[0] = projected_ver_buf[2][0];
1219            c[1] = projected_ver_buf[2][1];
1220            //=====================================================================
1221
1222            //点a、b、cがそれぞれ何番目の頂点かを参照
1223
1224            int index_a = poly.idx[i*3+0];
1225            int index_b = poly.idx[i*3+1];
1226            int index_c = poly.idx[i*3+2];
1227
1228            //点iの輝度値を参照する
1229            double rgb_a[3], rgb_b[3], rgb_c[3];
1230            rgb_a[0] = rgb_i[index_a*3+0];
1231            rgb_a[1] = rgb_i[index_a*3+1];
1232            rgb_a[2] = rgb_i[index_a*3+2];
1233
1234            rgb_b[0] = rgb_i[index_b*3+0];
1235            rgb_b[1] = rgb_i[index_b*3+1];
1236            rgb_b[2] = rgb_i[index_b*3+2];
1237
1238            rgb_c[0] = rgb_i[index_c*3+0];
1239            rgb_c[1] = rgb_i[index_c*3+1];
1240            rgb_c[2] = rgb_i[index_c*3+2];
1241
1242            //関数shadingの中では3点の空間内での座標も必要
1243            double A[3], B[3], C[3];
1244            A[0] = poly.vtx[index_a*3 + 0];
1245            A[1] = poly.vtx[index_a*3 + 1];
1246            A[2] = poly.vtx[index_a*3 + 2];
1247
1248            B[0] = poly.vtx[index_b*3 + 0];
1249            B[1] = poly.vtx[index_b*3 + 1];
1250            B[2] = poly.vtx[index_b*3 + 2];
1251
1252            C[0] = poly.vtx[index_c*3 + 0];
1253            C[1] = poly.vtx[index_c*3 + 1];
1254            C[2] = poly.vtx[index_c*3 + 2];
1255            //三角形iのシェーディングを行う
1256
1257            //三角形iの（本来の）法線ベクトルは
1258            //(poly_n[i*3+0], poly_n[i*3+1], poly_n[i*3+2])
1259            double poly_i_n_vec[3]
1260                = {poly_n[i*3+0], poly_n[i*3+1], poly_n[i*3+2]};
1261
1262            shading(a, b, c, rgb_a, rgb_b, rgb_c, A, B, C, poly_i_n_vec);
1263        }
1264
1265
1266
1267
1268        //ヘッダー出力
1269        fputs(MAGICNUM, fp_ppm);
1270        fputs("\n", fp_ppm);
1271        fputs(WIDTH_STRING, fp_ppm);
1272        fputs("␣", fp_ppm);
1273        fputs(HEIGHT_STRING, fp_ppm);
1274        fputs("\n", fp_ppm);
1275        fputs(MAX_STRING, fp_ppm);
1276        fputs("\n" ,fp_ppm);
1277
1278        //imageの出力
1279        for(int i = 0; i < 256; i++){
1280            for(int j = 0; j < 256; j++){
1281                char r[256];
1282                char g[256];
1283                char b[256];
1284                char str[1024];
1285                sprintf(r, "%d", (int)round(image[i][j][0]));
1286                sprintf(g, "%d", (int)round(image[i][j][1]));
1287                sprintf(b, "%d", (int)round(image[i][j][2]));
1288                sprintf(str, "%s\t%s\t%s\n", r, g, b);
1289                fputs(str, fp_ppm);
1290            }
```

```
1291            }
1292        }
1293        fclose(fp_ppm);
1294        fclose(fp);
1295
1296        printf("\nppmファイル␣%s␣の作成が完了しました.\n", fname );
1297        return 1;
1298    }
```

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.

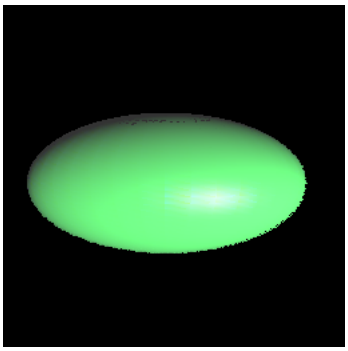

図 1 av5.wrl の出力結果

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.

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 3　EvalKadai05.sh

```
#!/bin/sh
SRC=kadai05New.c

WRL=sample/av5.wrl

PPM0=Kadai05ForAv5-0.ppm
CAMERA0=0.0

PPM1=Kadai05ForAv5-1.ppm
CAMERA1=50.0

PPM2=Kadai05ForAv5-2.ppm
CAMERA2=-50.0

gcc -Wall $SRC
./a.out $WRL $PPM0 $CAMERA0
open $PPM0

./a.out $WRL $PPM1 $CAMERA1
open $PPM1

./a.out $WRL $PPM2 $CAMERA2
open $PPM2
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、

```
$ 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

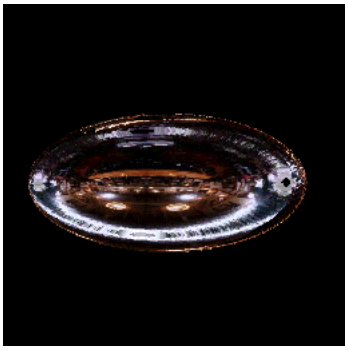

図 4　カメラの x 座標が-50.0 の時の av5 の出力結果