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
課題 3

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

本実験課題では、3D ポリゴンデータを透視投影によって投影した PPM 画像を生成するプログラムを、課題 2 のプログラム kadai02.c を拡張させる形で C 言語で作成した. したがって、基本的な仕様は前回の ReportForKadai02.pdf に準拠し、本文では変更点に焦点を当てて言及することとする.

2 要求仕様

作成したプログラムが満たす仕様は以下の通りである.

- ポリゴンデータは VRML 形式（拡張子 wrl）のファイルで取り込む形式とした.
- 光源方向は (x,y,z) = (-1.0, -1.0, 2.0) とした.
- 光源の明るさは (r,g,b) = (1.0, 1.0, 1.0) とした.
- 光源モデルは平行光源を採用した.
- カメラ位置は (x,y,z) = (0.0, 0.0, 0.0) とした.
- カメラ方向は (x,y,z) = (0.0, 0.0, 1.0) とした.
- カメラ焦点距離は 256.0 とした.
- ポリゴンには拡散反射に加えて鏡面反射を施した.
- コンスタントシェーディングによりポリゴンを描画した.
- z バッファによる隠面処理を行った.

3 プログラムの仕様

3.1 留意点

本課題では課題 2 と同様 VRML ファイルの読み込みに与えられたルーチンを使用した. なお、主な課題 2 からの変更点については次に示す.

-ADDED! double shininess
-ADDED! double specular_color[3]
-MODIFIED! main(int argc, char *argv[])
-MODIFIED! void shading(double *a, double *b, double *c, double *n, double *A)
-MODIFIED! main(int argc, char *argv[])

3.2 各種定数

プログラム内部で使用した重要な定数について以下に挙げておく.

3.2.1 ppm

次の定数は ppm ファイル生成のための定数である. kadai02.c と同一のものを使用した.

- MAGICNUM
  ppm ファルのヘッダに記述する識別子. P3 を使用.
- WIDTH, HEIGHT, WIDTH_STRING, HEIGHT_STRING
  出力画像の幅、高さ. ともに 256 とする. STRING は文字列として使用するためのマクロ. 以降も同様.
- MAX, MAX_STRING
  RGB の最大値. 255 を使用.

3.2.2 環境設定

次の定数は光源モデルなどの外部環境を特定する定数である.

- FOCUS
  カメラの焦点距離. 256.0 と指定.
- light_dir[3]
  光源方向ベクトル.doubel 型配列.
- light_rgb[3]
  光源の明るさを正規化した RGB 値にして配列に格納したもの. double 型配列.

3.2.3 その他

- image[HEIGHT][WIDTH]
  描画した画像の各点の画素値を格納するための領域. 領域確保のみで初期化は関数内で行う. double 型の 2 次元配列.
- z_buf[HEIGHT][WIDTH]
  z バッファを格納するための領域. 全ての頂点分の z バッファを格納する. 初期化は main 関数内で行う. なお、初期化時の最大値としては double 型の最大値 DBL_MAX を使用した. double 型 2 次元配列.
- projected_ver_buf[3][2]
  ポリゴンを形成する 3 点に対して透視投影を施した結果の座標を保存しておくためのバッファ. なお、課題 02 で作成した kadai02.c の内部ではグローバル変数としていたが、kadai03.c ではバグを避けるた

め main 内部で宣言する変数とした. double 型 2 次元配列.

- double shininess
  鏡面反射強度を格納する double 型変数.
- double specular_color[3]
  鏡面反射係数を格納する doubel 型配列.

3.3 関数外部仕様

3.3.1 double func1(double *p, double *q, double y)

kadai02.c と同一. double 型 2 次元配列で表された 2 点 p、q の座標と double 型の値 y を引数に取り、直線 pq と直線 $y =$ (引数 y の値) の交点の x 座標を double 型で返す関数. ラスタライズの計算を簡素化するために三角形を分割する際に主に用いる.

3.3.2 int lineOrNot(double *a, double *b, double *c)

kadai02.c と同一. double 型 2 次元配列で表された 3 点 a、b、c が一直線上にあるかどうかを判別する関数. 一直線上にある場合は int 型 1 を返し、それ以外のときは int 型 0 を返す. 後述の関数 shading の中で用いる.

3.3.3 void shading(double *a, double *b, double *c, double *n, double *A)

内部で変更があるが、外部仕様としては kadai02.c と同一. 画像平面上に投影された duble 型 2 次元配列で与えられた 3 点 a、b、c に対してシェーディングを行う関数.

3.4 各関数のアルゴリズムの概要

3.4.1 double func1(double *p, double *q, double y)

kadai02.c と同一. 2 点 p、q を通る直線の方程式を求めて、直線 $y =$ (引数 y の値) との交点を計算する. なお直線 pq が x 軸に平行の時はエラーが発生する.

3.4.2 int lineOrNot(double *a, double *b, double *c)

kadai02.c と同一. まず最初に 3 点 a、b、c の x 座標が全て同じであるかどうかを判定し、同じであれば一直線上にあると判定する. 同じでなければ、次に点 c の座標を直線 ab の方程式に代入し、等号が成立するかどうかで一直線上にあるかどうかを判定する。

3.4.3 void shading(double *a, double *b, double *c, double *n)

kadai02.c のものを拡張、変更した. 変更点は、ラスタ走査でシェーディングを行う際に、描画中の点に対して鏡面反射を施すために必要な視線方向ベクトルを計算しながらシェーディングを行うという点である. これにはテキストから抜粋した図 3.4.3 における、ベクトル e を求める計算が必要であるが、投影平面上の点 (x_p, y_p) の三次元空間内での座標は、カメラ位置（原点）と投影平面上の点を結ぶ直線と、xyz 空間内の元の三角形 ABC を含む平面との交点を求める形で算出でき、元の三角形の法線ベクトルを (n_x, n_y, n_z)、点 A で

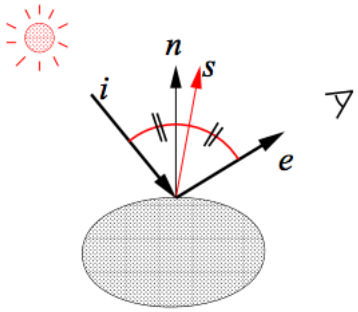

図 1　鏡面反射（テキストより）

の座標を (x_A, y_A, z_A)、投影平面の z 座標を z_p とすると、

$$(\frac{x_p(n_x x_A + n_y y_A + n_z z_A)}{n_x x_p + n_y y_p + n_z z_p}, \frac{y_p(n_x x_A + n_y y_A + n_z z_A)}{n_x x_p + n_y y_p + n_z z_p}, \frac{z_p(n_x x_A + n_y y_A + n_z z_A)}{n_x x_p + n_y y_p + n_z z_p}) \tag{1}$$

として表されるから、視線方向のベクトルはこの座標と視点座標から容易に求まる.

3.4.4　int main(int argc, char *argv[])

kadai02.c のものを変更、拡張. 読み込む VRML が記述されたファイル名（*.wrl）と出力画像を書き込む ppm ファイル名（*.ppm）をコマンドライン引数として取得する. kadai02.c からの変更点としては、main 関数内部でグローバル領域に確保した鏡面反射係数を格納する配列と、鏡面反射強度を格納する変数を初期化しする点である. なお、鏡面反射強度についてはテキストの注意事項に記載のあった通り、VRML に記載されている値を 128 倍して使用することとした.

4　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　kadai03.c

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

//diffuseColorを格納する配列
double diffuse_color[3];
//shinessを格納する変数
```

```
25  double shininess;
26  //specularColorを格納する変数
27  double specular_color[3];
28
29  //光源モデルは平行光源
30
31  //光源方向
32  const double light_dir[3] = {-1.0, -1.0, 2.0};
33  //光源明るさ
34  const double light_rgb[3] = {1.0, 1.0, 1.0};
35  //カメラ位置は原点であるものとして投影を行う.
36  //========================================================================
37  //メモリ内に画像の描画領域を確保
38  double image[HEIGHT][WIDTH][3];
39  //zバッファ用の領域を確保
40  double z_buf[HEIGHT][WIDTH];
41
42
43  //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
44  //eg)
45  //double p[2] = (1.0, 2.0);
46  double func1(double *p, double *q, double y){
47      double x;
48      if(p[1] > q[1]){
49          x = ((p[0] * (y - q[1])) + (q[0] * (p[1] - y))) / (p[1] - q[1]);
50      }
51      if(p[1] < q[1]){
52          x = ((q[0] * (y - p[1])) + (p[0] * (q[1] - y))) / (q[1] - p[1]);
53      }
54      if(p[1] == q[1]){
55          //解なし
56          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"
57                  , p[0], p[1], q[0], q[1]);
58          perror(NULL);
59          return -1;
60      }
61      return x;
62  }
63
64  //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
65  //1直線上に無ければreturn 0;
66  //1直線上にあればreturn 1;
67  int lineOrNot(double *a, double *b, double *c){
68      if(a[0] == b[0]){
69          if(a[0] == c[0]){
70              return 1;
71          }
72          else{
73              return 0;
74          }
75      }
76      else{
77          if(c[1] == a[1] + ((b[1] - a[1]) / (b[0] - a[0])) * (c[0] - a[0])){
78              return 1;
79          }
80          else{
81              return 0;
82          }
83      }
84  }
85
86  //投影された三角形abcにラスタライズ、クリッピングでシェーディングを行う関数
87  //引数a, b, cは投影平面上の3点
88  //eg)
89  //double a = {1.0, 2.0};
90  //nは法線ベクトル
91  //Aは投影前の3点からなる三角形平面上の任意の点の座標.
92  //(3点A、B、Cのうちいずれでも良いがmain関数内のAを使うものとする.)
93  void shading(double *a, double *b, double *c, double *n, double *A){
94      //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
95      if(lineOrNot(a, b, c) == 1){
96          //塗りつぶす点が無いので何もしない.
97      }
98      else{
99          //y座標の値が真ん中点をp、その他の点をq、rとする
100         //y座標の大きさはr <= p <= qの順
101         double p[2], q[2], r[2];
102         if(b[1] <= a[1] && a[1] <= c[1]){
103             memcpy(p, a, sizeof(double) * 2);
```

```
104             memcpy(q, c, sizeof(double) * 2);
105             memcpy(r, b, sizeof(double) * 2);
106         }
107         else{
108             if(c[1] <= a[1] && a[1] <= b[1]){
109                 memcpy(p, a, sizeof(double) * 2);
110                 memcpy(q, b, sizeof(double) * 2);
111                 memcpy(r, c, sizeof(double) * 2);
112             }
113             else{
114                 if(a[1] <= b[1] && b[1] <= c[1]){
115                     memcpy(p, b, sizeof(double) * 2);
116                     memcpy(q, c, sizeof(double) * 2);
117                     memcpy(r, a, sizeof(double) * 2);
118                 }
119                 else{
120                     if(c[1] <= b[1] && b[1] <= a[1]){
121                         memcpy(p, b, sizeof(double) * 2);
122                         memcpy(q, a, sizeof(double) * 2);
123                         memcpy(r, c, sizeof(double) * 2);
124                     }
125                     else{
126                         if(b[1] <= c[1] && c[1] <= a[1]){
127                             memcpy(p, c, sizeof(double) * 2);
128                             memcpy(q, a, sizeof(double) * 2);
129                             memcpy(r, b, sizeof(double) * 2);
130                         }
131                         else{
132                             if(a[1] <= c[1] && c[1] <= b[1]){
133                                 memcpy(p, c, sizeof(double) * 2);
134                                 memcpy(q, b, sizeof(double) * 2);
135                                 memcpy(r, a, sizeof(double) * 2);
136                             }
137                             else{
138                                 printf("エラーat2055\n");
139                                 printf("\na[1]=%f\tb[1]=%f\tc[1]=%f\n", a[1], b[1], c[1]);
140                                 perror(NULL);
141                             }
142                         }
143                     }
144                 }
145             }
146         }
147         //分割可能な三角形かを判定
148         if(p[1] == r[1] || p[1] == q[1]){
149             //分割できない
150 
151             //長さが1の光源方向ベクトルを作成する
152             //光源方向ベクトルの長さ
153             double length_l =
154                 sqrt(pow(light_dir[0], 2.0) +
155                      pow(light_dir[1], 2.0) +
156                      pow(light_dir[2], 2.0));
157 
158             double light_dir_vec[3];
159             light_dir_vec[0] = light_dir[0] / length_l;
160             light_dir_vec[1] = light_dir[1] / length_l;
161             light_dir_vec[2] = light_dir[2] / length_l;
162 
163             //2パターンの三角形を特定
164             if(p[1] == r[1]){
165                 //x座標が p <= r となるように調整
166                 if(r[0] <  p[0]){
167                     double temp[2];
168                     memcpy(temp, r, sizeof(double) * 2);
169                     memcpy(r, p, sizeof(double) * 2);
170                     memcpy(p, temp, sizeof(double) * 2);
171                 }
172 
173                 //debug
174                 if(r[0] == p[0]){
175                   perror("エラーat958");
176                 }
177 
178                 //シェーディング処理
179                 //三角形pqrをシェーディング
180                 //y座標はp <= r
181                 //debug
182                 if(r[1] < p[1]){
```

```
183                     perror("エラーat1855");
184                 }
185
186                 /* 点(j, i)のシェーディング============================================== */
187                 int i;
188                 i = ceil(p[1]);
189                 for(i;
190                     p[1] <= i && i <= q[1];
191                     i++){
192
193                     //撮像平面からはみ出ていないかのチェック
194                     if(0 <= i
195                        &&
196                        i <= (HEIGHT - 1)){
197                             double x1 = func1(p, q, i);
198                             double x2 = func1(r, q, i);
199                             int j;
200                             j = ceil(x1);
201
202                             for(j;
203                                 x1 <= j && j <= x2 && 0 <= j && j <= (WIDTH - 1);
204                                 j++){
205
206
207                                 /* 鏡面反射を計算 */
208
209                                 //描画する点の投影前の空間内の座標.
210                                 double p_or[3];
211
212                                 p_or[0] =
213                                     (j-(WIDTH/2))
214                                     *
215                                     ((n[0]*A[0]) + (n[1]*A[1]) + (n[2]*A[2]))
216                                     /
217                                     ((n[0]*(j-(WIDTH/2))) + (n[1]*(i-(HEIGHT/2))) + n[2]*FOCUS);
218
219                                 p_or[1] =
220                                     (i-(HEIGHT/2))
221                                     *
222                                     ((n[0]*A[0]) + (n[1]*A[1]) + (n[2]*A[2]))
223                                     /
224                                     ((n[0]*(j-(WIDTH/2))) + (n[1]*(i-(HEIGHT/2))) + n[2]*FOCUS);
225
226                                p_or[2] =
227                                     FOCUS
228                                     *
229                                     ((n[0]*A[0]) + (n[1]*A[1]) + (n[2]*A[2]))
230                                     /
231                                     ((n[0]*(j-(WIDTH/2))) + (n[1]*(i-(HEIGHT/2))) + n[2]*FOCUS);
232
233                                 /* eは描画中の点pから視点位置へ向かうベクトルを計算 */
234                                 //視点方向は原点に固定
235                                 double e[3];
236                                 e[0] = -1 * p_or[0];
237                                 e[1] = -1 * p_or[1];
238                                 e[2] = -1 * p_or[2];
239                                 double length_e =
240                                     sqrt(pow(e[0], 2.0) +
241                                          pow(e[1], 2.0) +
242                                          pow(e[2], 2.0));
243                                 e[0] = (e[0] / length_e);
244                                 e[1] = (e[1] / length_e);
245                                 e[2] = (e[2] / length_e);
246
247                                 /* iは光源から描画中の点pへの入射光ベクトルを計算 */
248                                 //平行光源のため光源方向は
249                                 //const double light_dir[3] = {-1.0, -1.0, 2.0};
250                                 //を用いる
251                                 double i_vec[3];
252                                 i_vec[0] = light_dir[0];
253                                 i_vec[1] = light_dir[1];
254                                 i_vec[2] = light_dir[2];
255                                 double length_i =
256                                     sqrt(pow(i_vec[0], 2.0) +
257                                          pow(i_vec[1], 2.0) +
258                                          pow(i_vec[2], 2.0));
259                                 i_vec[0] = (i_vec[0] / length_i);
260                                 i_vec[1] = (i_vec[1] / length_i);
261                                 i_vec[2] = (i_vec[2] / length_i);
```

```
262
263                             /* sベクトルを計算 */
264                             double s[3];
265                             s[0] = e[0] - i_vec[0];
266                             s[1] = e[1] - i_vec[1];
267                             s[2] = e[2] - i_vec[2];
268                             double s_length =
269                                 sqrt(pow(s[0], 2.0) +
270                                      pow(s[1], 2.0) +
271                                      pow(s[2], 2.0));
272                             s[0] = (s[0] / s_length);
273                             s[1] = (s[1] / s_length);
274                             s[2] = (s[2] / s_length);
275
276                             //内積sn
277                             double sn =
278                                 ((s[0] * n[0]) + (s[1] * n[1]) + (s[2] * n[2]));
279
280                             if(sn <= 0){sn = 0;}
281
282                             /* 法線ベクトルnと光源方向ベクトルの内積 */
283                             double ip =
284                                 (n[0] * i_vec[0]) +
285                                 (n[1] * i_vec[1]) +
286                                 (n[2] * i_vec[2]);
287
288                             if(0 <= ip){ip = 0;}
289
290                             //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）
291                             if(z_buf[i][j] < p_or[2]){}
292
293                             else{
294                                 image[i][j][0] =
295                                     (-1 * ip * diffuse_color[0] * light_rgb[0] * MAX)
296                                     + (pow(sn, shininess) * specular_color[0] * light_rgb[0] * MAX)
297                                     ;
298
299                                 image[i][j][1] =
300                                     (-1 * ip * diffuse_color[1] * light_rgb[1] * MAX)
301                                     + (pow(sn, shininess) * specular_color[1] * light_rgb[1] * MAX)
302                                     ;
303
304                                 image[i][j][2] =
305                                     (-1 * ip * diffuse_color[2] * light_rgb[2] * MAX)
306                                     + (pow(sn, shininess) * specular_color[2] * light_rgb[2] * MAX)
307                                     ;
308
309                                 //zバッファの更新
310                                 z_buf[i][j] = p_or[2];
311                             }
312                         }
313                         /* 点(j, i)のシェーディングここま
                               で========================================== */
314                     }
315                     //はみ出ている場合は描画しない
316                     else{}
317                 }
318
319             }
320
321             if(p[1] == q[1]){
322                 //x座標が p < q となるように調整
323                 if(q[0] <  p[0]){
324                     double temp[2];
325                     memcpy(temp, q, sizeof(double) * 2);
326                     memcpy(q, p, sizeof(double) * 2);
327                     memcpy(p, temp, sizeof(double) * 2);
328                 }
329
330                 //debug
331                 if(q[0] == p[0]){
332                     perror("エラーat1011");
333                 }
334
335                 //シェーディング処理
336                 //三角形pqrをシェーディング
337                 //y座標はp <= q
338                 //debug
339                 if(q[1] < p[1]){
```

```
340                     perror("エラーat1856");
341                 }
342
343                 int i;
344                 i = ceil(r[1]);
345
346                 for(i;
347                     r[1] <= i && i <= p[1];
348                     i++){
349
350                     //撮像部分からはみ出ていないかのチェック
351                     if( 0 <= i &&
352                         i <= (HEIGHT - 1)){
353                         double x1 = func1(p, r, i);
354                         double x2 = func1(q, r, i);
355
356                         int j;
357                         j = ceil(x1);
358
359                         for(j;
360                             x1 <= j && j <= x2 && 0 <= j && j <= (WIDTH - 1);
361                             j++){
362
363                             /* 鏡面反射を適用 */
364                             //描画する点の投影前の空間内の座標.
365                             double p_or[3];
366
367                             p_or[0] =
368                                 (j-(WIDTH/2))
369                                 *
370                                 ((n[0]*A[0]) + (n[1]*A[1]) + (n[2]*A[2]))
371                                 /
372                                 ((n[0]*(j-(WIDTH/2))) + (n[1]*(i-(HEIGHT/2))) + n[2]*FOCUS);
373
374                             p_or[1] =
375                                 (i-(MAX/2))
376                                 *
377                                 ((n[0]*A[0]) + (n[1]*A[1]) + (n[2]*A[2]))
378                                 /
379                                 ((n[0]*(j-(WIDTH/2))) + (n[1]*(i-(HEIGHT/2))) + n[2]*FOCUS);
380
381                             p_or[2] =
382                                 FOCUS
383                                 *
384                                 ((n[0]*A[0]) + (n[1]*A[1]) + (n[2]*A[2]))
385                                 /
386                                 ((n[0]*(j-(WIDTH/2))) + (n[1]*(i-(HEIGHT/2))) + n[2]*FOCUS);
387
388                             /* 描画中の点pから視点位置へ向かう単位方向ベクトルe計算 */
389                             //視点方向は原点に固定
390                             double e[3];
391                             e[0] = -1 * p_or[0];
392                             e[1] = -1 * p_or[1];
393                             e[2] = -1 * p_or[2];
394                             double length_e =
395                                 sqrt(pow(e[0], 2.0) +
396                                      pow(e[1], 2.0) +
397                                      pow(e[2], 2.0));
398                             e[0] = (e[0] / length_e);
399                             e[1] = (e[1] / length_e);
400                             e[2] = (e[2] / length_e);
401
402                             /* 光源から描画中の点pへの入射光ベクトルiを計算 */
403                             //平行光源のため光源方向は
404                             //const double light_dir[3] = {-1.0, -1.0, 2.0};
405                             //を用いる
406                             double i_vec[3];
407                             i_vec[0] = light_dir[0];
408                             i_vec[1] = light_dir[1];
409                             i_vec[2] = light_dir[2];
410                             double length_i =
411                                 sqrt(pow(i_vec[0], 2.0) +
412                                      pow(i_vec[1], 2.0) +
413                                      pow(i_vec[2], 2.0));
414                             i_vec[0] = (i_vec[0] / length_i);
415                             i_vec[1] = (i_vec[1] / length_i);
416                             i_vec[2] = (i_vec[2] / length_i);
417
418                             /* sベクトルを計算 */
```

```
419                             double s[3];
420                             s[0] = e[0] - i_vec[0];
421                             s[1] = e[1] - i_vec[1];
422                             s[2] = e[2] - i_vec[2];
423                             double s_length =
424                                 sqrt(pow(s[0], 2.0) + pow(s[1], 2.0) + pow(s[2], 2.0));
425                             s[0] = (s[0] / s_length);
426                             s[1] = (s[1] / s_length);
427                             s[2] = (s[2] / s_length);
428
429                             //内積sn
430                             double sn = ((s[0] * n[0]) +
431                                          (s[1] * n[1]) +
432                                          (s[2] * n[2]));
433                             if(sn <= 0){sn = 0;}
434
435                             //拡散反射
436                             /* 法線ベクトルnと光源方向ベクトルの内積を計算 */
437                             double ip =
438                                 (n[0] * i_vec[0]) +
439                                 (n[1] * i_vec[1]) +
440                                 (n[2] * i_vec[2]);
441
442                             if(0 <= ip){ip = 0;}
443
444
445                             //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）
446                             if(z_buf[i][j] < p_or[2]){}
447
448                             else{
449
450                                 image[i][j][0] =
451                                     (-1 * ip * diffuse_color[0] * light_rgb[0] * MAX)
452                                     + (pow(sn, shininess) * specular_color[0] * light_rgb[0] * MAX)
453                                     ;
454
455                                 image[i][j][1] =
456                                     (-1 * ip * diffuse_color[1] * light_rgb[1] * MAX)
457                                     + (pow(sn, shininess) * specular_color[1] * light_rgb[1] * MAX)
458                                     ;
459
460                                 image[i][j][2] =
461                                     (-1 * ip * diffuse_color[2] * light_rgb[2] * MAX)
462                                     + (pow(sn, shininess) * specular_color[2] * light_rgb[2] * MAX)
463                                     ;
464
465                                 z_buf[i][j] = p_or[2];
466                             }
467                         }
468                     }
469                     //撮像平面からはみ出る部分は描画しない
470                     else{}
471                 }
472             }
473
474         }
475         //分割できる
476         //分割してそれぞれ再帰的に処理
477         //分割後の三角形はpp2qとpp2r
478         else{
479             double p2[2];
480             p2[0] = func1(q, r, p[1]);
481             p2[1] = p[1];
482             //p2のほうがpのx座標より大きくなるようにする
483             if(p2[0] < p[0]){
484                 double temp[2];
485                 memcpy(temp, p2, sizeof(double) * 2);
486                 memcpy(p2, p, sizeof(double) * 2);
487                 memcpy(p, temp, sizeof(double) * 2);
488             }
489             //分割しても同一平面上なので法線ベクトルと
490             //平面上の任意の点は同じものを使える.
491             shading(p, p2, q, n, A);
492             shading(p, p2, r, n, A);
493         }
494     }
495 }
496
497 /* VRMLの読み込み */
```

```
/* ========================================================================== */
#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)
{
    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;
    }
    while (getword(fp,b,MWS)>0) {
        if (strindex(b,"]")>=0) break;
        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[]){
    /* VRML読み込み ============================================================ */
    int i;
    FILE *fp;
    Polygon poly;
    Surface surface;

```

```
735     fp = fopen(argv[1], "r");
736     read_one_obj(fp, &poly, &surface);
737 
738     fprintf(stderr,"%d␣vertice␣are␣found.\n",poly.vtx_num);
739     fprintf(stderr,"%d␣triangles␣are␣found.\n",poly.idx_num);
740 
741     //i th vertex
742     for ( i = 0 ; i < poly.vtx_num ; i++ ) {
743         fprintf(stdout,"%f␣%f␣%f␣#␣%d␣th␣vertex\n",
744                 poly.vtx[i*3+0], poly.vtx[i*3+1], poly.vtx[i*3+2],
745                 i);
746     }
747 
748     //i th triangle
749     for ( i = 0 ; i < poly.idx_num ; i++ ) {
750         fprintf(stdout,"%d␣%d␣%d␣#␣%d␣th␣triangle\n",
751                 poly.idx[i*3+0], poly.idx[i*3+1], poly.idx[i*3+2],
752                 i);
753     }
754 
755     /* material info */
756     fprintf(stderr, "diffuseColor␣%f␣%f␣%f\n", surface.diff[0], surface.diff[1], surface.diff[2]);
757     fprintf(stderr, "specularColor␣%f␣%f␣%f\n", surface.spec[0], surface.spec[1], surface.spec[2]);
758     fprintf(stderr, "ambientIntensity␣%f\n", surface.ambi);
759     fprintf(stderr, "shininess␣%f\n", surface.shine);
760     /* VRML読み込みここまで ========================================================== */
761 
762     FILE *fp_ppm;
763     char *fname = argv[2];
764     fp_ppm = fopen(argv[2], "w");
765 
766     //ファイルが開けなかったとき
767     if( fp_ppm == NULL ){
768         printf("%sファイルが開けません.\n", fname);
769         return -1;
770     }
771 
772     //ファイルが開けたとき
773     else{
774         //描画領域を初期化
775         for(int i = 0; i < 256; i++){
776             for(int j = 0; j < 256; j++){
777                 image[i][j][0] = 0.0 * MAX;
778                 image[i][j][1] = 0.0 * MAX;
779                 image[i][j][2] = 0.0 * MAX;
780             }
781         }
782 
783          //zバッファを初期化
784         for(int i = 0; i < 256; i++){
785             for(int j = 0; j < 256; j++){
786                 z_buf[i][j] = DBL_MAX;
787             }
788         }
789 
790         //diffuse_colorの格納
791         diffuse_color[0] = surface.diff[0];
792         diffuse_color[1] = surface.diff[1];
793         diffuse_color[2] = surface.diff[2];
794 
795         //shininessの格納
796         //!!!!!!!!!!!!!!!注意!!!!!!!!!!!!!!!!!
797         //(実験ページの追加情報を参照)
798         //各ファイルのshininessの値は
799         //av4 0.5
800         //av5 0.5
801         //iiyama1997 1.0
802         //aa053 1.0
803         //av007 0.34
804         shininess = surface.shine * 128;
805 
806         //speculorColorの格納
807         specular_color[0] = surface.spec[0];
808         specular_color[1] = surface.spec[1];
809         specular_color[2] = surface.spec[2];
810 
811         //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
812         double projected_ver_buf[3][2];
813 
```

```
814         //シェーディング
815         //三角形ごとのループ
816         for(int i = 0; i < poly.idx_num; i++){
817             //各点の透視投影処理
818             for(int j = 0; j < 3; j++){
819                 double xp = poly.vtx[(poly.idx[i*3+j])*3 + 0];
820                 double yp = poly.vtx[(poly.idx[i*3+j])*3 + 1];
821                 double zp = poly.vtx[(poly.idx[i*3+j])*3 + 2];
822                 double zi = FOCUS;
823 
824                 //debug
825                 if(zp == 0){
826                     printf("\n(%f\t%f\t%f) i=%d, j=%d\n", xp, yp, zp, i, j);
827                     perror("\nエラー0934\n");
828                     //break;
829                 }
830 
831                 double xp2 = xp * (zi / zp);
832                 double yp2 = yp * (zi / zp);
833                 double zp2 = zi;
834 
835                 //座標軸を平行移動
836                 projected_ver_buf[j][0] = (MAX / 2) + xp2;
837                 projected_ver_buf[j][1] = (MAX / 2) + yp2;
838             }
839 
840             double a[2], b[2], c[2];
841             a[0] = projected_ver_buf[0][0];
842             a[1] = projected_ver_buf[0][1];
843             b[0] = projected_ver_buf[1][0];
844             b[1] = projected_ver_buf[1][1];
845             c[0] = projected_ver_buf[2][0];
846             c[1] = projected_ver_buf[2][1];
847 
848             double A[3], B[3], C[3];
849             A[0] = poly.vtx[(poly.idx[i*3+0])*3 + 0];
850             A[1] = poly.vtx[(poly.idx[i*3+0])*3 + 1];
851             A[2] = poly.vtx[(poly.idx[i*3+0])*3 + 2];
852 
853             B[0] = poly.vtx[(poly.idx[i*3+1])*3 + 0];
854             B[1] = poly.vtx[(poly.idx[i*3+1])*3 + 1];
855             B[2] = poly.vtx[(poly.idx[i*3+1])*3 + 2];
856 
857             C[0] = poly.vtx[(poly.idx[i*3+2])*3 + 0];
858             C[1] = poly.vtx[(poly.idx[i*3+2])*3 + 1];
859             C[2] = poly.vtx[(poly.idx[i*3+2])*3 + 2];
860 
861             //ベクトルAB, ACから外積を計算して
862             //法線ベクトルnを求める
863             double AB[3], AC[3], n[3];
864             AB[0] = B[0] - A[0];
865             AB[1] = B[1] - A[1];
866             AB[2] = B[2] - A[2];
867 
868             AC[0] = C[0] - A[0];
869             AC[1] = C[1] - A[1];
870             AC[2] = C[2] - A[2];
871 
872             n[0] = (AB[1] * AC[2]) - (AB[2] * AC[1]);
873             n[1] = (AB[2] * AC[0]) - (AB[0] * AC[2]);
874             n[2] = (AB[0] * AC[1]) - (AB[1] * AC[0]);
875 
876             //長さを1に調整
877             double length_n =
878                 sqrt(pow(n[0], 2.0) +
879                      pow(n[1], 2.0) +
880                      pow(n[2], 2.0));
881 
882             n[0] = n[0] / length_n;
883             n[1] = n[1] / length_n;
884             n[2] = n[2] / length_n;
885 
886             //平面iの投影先の三角形をシェーディング
887             shading(a, b, c, n, A);
888         }
889 
890         //ヘッダー出力
891         fputs(MAGICNUM, fp_ppm);
892         fputs("\n", fp_ppm);
```

```
893         fputs(WIDTH_STRING, fp_ppm);
894         fputs("␣", fp_ppm);
895         fputs(HEIGHT_STRING, fp_ppm);
896         fputs("\n", fp_ppm);
897         fputs(MAX_STRING, fp_ppm);
898         fputs("\n" ,fp_ppm);
899
900         //image の出力
901         for(int i = 0; i < 256; i++){
902             for(int j = 0; j < 256; j++){
903                 char r[256];
904                 char g[256];
905                 char b[256];
906                 char str[1024];
907
908                 sprintf(r, "%d", (int)round(image[i][j][0]));
909                 sprintf(g, "%d", (int)round(image[i][j][1]));
910                 sprintf(b, "%d", (int)round(image[i][j][2]));
911                 sprintf(str, "%s\t%s\t%s\n", r, g, b);
912                 fputs(str, fp_ppm);
913             }
914         }
915     }
916     fclose(fp_ppm);
917     fclose(fp);
918
919     printf("\nppm ファイル␣%s␣に画像を出力しました.\n", fname );
920     return 1;
921 }
```

5 実行例

kadai03.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 EvalKadai03.sh

```
1  #!/bin/sh
2  SRC=kadai03.c
3
4  WRL4=sample/av4.wrl
5  PPM4=Kadai03ForAv4.ppm
6
7  WRLhead=sample/head.wrl
8  PPMhead=Kadai03ForHead.ppm
9
10 WRL1997=sample/iiyama1997.wrl
11 PPM1997=Kadai03ForIiyama1997.ppm
12
13
14 echo start!!
15 gcc -Wall $SRC
16
17 ./a.out $WRL4 $PPM4
18 open $PPM4
19
20 ./a.out $WRLhead $PPMhead
21 open $PPMhead
22
23 ./a.out $WRL1997 $PPM1997
24 open $PPM1997
25
26 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 EvalKadai03.sh
```

を実行した. なお、出力画像は図 2、図 3、図 4 のようになった.

6　工夫点、問題点、感想

今回のプログラムでも本来は避けるべきグローバル領域に諸々の変数の格納領域を確保してしまっている．そもそも main 関数内にグローバル変数を置くことが懸念される最大の理由は、関数名のダブり及び予期しないグローバル変数の書き換えによるバグの発見の困難さである．そのため、最初から初期化されている変数などについては、const を型宣言に加えてこれを防止するなどの処置がとられる．しかしながら今回のプログラムではグローバル変数を多用している．これは、このプログラムが完全に個人で作るものであるため、自分だけが注意すれば変数名のダブりや意図しない書き換えについては防止できること、グローバル変数として使用している変数を全て引数として関数呼び出しの際に与えると、引数が多過ぎてプログラムの可読性が落ちる等のことを考慮に入れたためである．ただ、最初の kadai01.c の段階でモジュールの分け方を失敗したことに起因する部分もあるので、そのあたりを工夫していれば綺麗にコードが書けたはずだと後悔している．

7　APPENDIX

ベースとした kadai02.c のプログラムを付加しておく．

リスト 3　kadai02.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <float.h>
#include <ctype.h>
#include "vrml.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 Z_BUF_MAX

//diffuseColorを格納する配列
double diffuse_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};

//カメラ位置は原点であるものとして投影を行う.
//=========================================================================
//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_buf[3][2];


//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){
```

```
46      double x;
47      if(p[1] > q[1]){
48          x = ((p[0] * (y - q[1])) + (q[0] * (p[1] - y))) / (p[1] - q[1]);
49      }
50      if(p[1] < q[1]){
51          x = ((q[0] * (y - p[1])) + (p[0] * (q[1] - y))) / (q[1] - p[1]);
52      }
53      if(p[1] == q[1]){
54          //解なし
55          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"
56                  , p[0], p[1], q[0], q[1]);
57          perror(NULL);
58          return -1;
59      }
60      return x;
61  }
62
63  //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
64  //1直線上に無ければreturn 0;
65  //1直線上にあればreturn 1;
66  int lineOrNot(double *a, double *b, double *c){
67      if(a[0] == b[0]){
68          if(a[0] == c[0]){
69              return 1;
70          }
71          else{
72              return 0;
73          }
74      }
75      else{
76          if(c[1] == a[1] + ((b[1] - a[1]) / (b[0] - a[0])) * (c[0] - a[0])){
77              return 1;
78          }
79          else{
80              return 0;
81          }
82      }
83  }
84
85  //投影された三角形abcにラスタライズ、クリッピングでシェーディングを行う関数
86  //引数a, b, cは投影平面上の3点
87  //eg)
88  //double a = {1.0, 2.0};
89  //nは法線ベクトル
90  //Aは投影前の3点からなる三角形平面上の任意の点の座標.
91  //(3点A、B、Cのうちいずれでも良いがmain関数内のAを使うものとする.)
92  void shading(double *a, double *b, double *c, double *n, double *A){
93      //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
94      if(lineOrNot(a, b, c) == 1){
95          //塗りつぶす点が無いので何もしない.
96      }
97      else{
98          //y座標の値が真ん中点をp、その他の点をq、rとする
99          //y座標の大きさはr <= p <= qの順
100         double p[2], q[2], r[2];
101         if(b[1] <= a[1] && a[1] <= c[1]){
102             memcpy(p, a, sizeof(double) * 2);
103             memcpy(q, c, sizeof(double) * 2);
104             memcpy(r, b, sizeof(double) * 2);
105         }
106         else{
107             if(c[1] <= a[1] && a[1] <= b[1]){
108                 memcpy(p, a, sizeof(double) * 2);
109                 memcpy(q, b, sizeof(double) * 2);
110                 memcpy(r, c, sizeof(double) * 2);
111             }
112             else{
113                 if(a[1] <= b[1] && b[1] <= c[1]){
114                     memcpy(p, b, sizeof(double) * 2);
115                     memcpy(q, c, sizeof(double) * 2);
116                     memcpy(r, a, sizeof(double) * 2);
117                 }
118                 else{
119                     if(c[1] <= b[1] && b[1] <= a[1]){
120                         memcpy(p, b, sizeof(double) * 2);
121                         memcpy(q, a, sizeof(double) * 2);
122                         memcpy(r, c, sizeof(double) * 2);
123                     }
124                     else{
```

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

        // 法線ベクトルnと光源方向ベクトルの内積
        double ip =
            (n[0] * light_dir_vec[0]) +
            (n[1] * light_dir_vec[1]) +
            (n[2] * light_dir_vec[2]);

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

        //2パターンの三角形を特定
        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];
                memcpy(temp, r, sizeof(double) * 2);
                memcpy(r, p, sizeof(double) * 2);
                memcpy(p, temp, sizeof(double) * 2);
            }

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

            //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 <= r
            //debug
            if(r[1] < p[1]){
                perror("エラーat1855");
            }

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

```
204                     //撮像平面からはみ出ていないかのチェック
205                     if(0 <= i
206                        &&
207                        i <= (HEIGHT - 1)){
208                             double x1 = func1(p, q, i);
209                             double x2 = func1(r, q, i);
210                             int j;
211                             j = ceil(x1);
212
213                             for(j;
214                                 x1 <= j && j <= x2 && 0 <= j && j <= (WIDTH - 1);
215                                 j++){
216
217                                 //描画する点の空間内のz座標.
218                                 double z =
219                                     FOCUS * ((n[0]*A[0]) + (n[1]*A[1]) + (n[2]*A[2]))
220                                     /
221                                     ((n[0]*(j-(WIDTH/2))) + (n[1]*(i-(HEIGHT/2))) + n[2]*FOCUS);
222
223                                 //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）
224                                 if(z_buf[i][j] < z){}
225
226                                 else{
227                                     image[i][j][0] =
228                                         -1 * ip * diffuse_color[0] *
229                                         light_rgb[0] * MAX;
230                                     image[i][j][1] =
231                                         -1 * ip * diffuse_color[1] *
232                                         light_rgb[1] * MAX;
233                                     image[i][j][2] =
234                                         -1 * ip * diffuse_color[2] *
235                                         light_rgb[2] * MAX;
236
237                                     //zバッファの更新
238                                     z_buf[i][j] = z;
239                                 }
240                             }
241                     }
242                     //はみ出ている場合は描画しない
243                     else{}
244                 }
245             }
246
247             if(p[1] == q[1]){
248                 //x座標が p < q となるように調整
249                 if(q[0] <  p[0]){
250                     double temp[2];
251                     memcpy(temp, q, sizeof(double) * 2);
252                     memcpy(q, p, sizeof(double) * 2);
253                     memcpy(p, temp, sizeof(double) * 2);
254                 }
255
256                 //debug
257                 if(q[0] == p[0]){
258                     perror("エラーat1011");
259                 }
260
261                 //シェーディング処理
262                 //三角形pqrをシェーディング
263                 //y座標はp <= q
264
265                 //debug
266                 if(q[1] < p[1]){
267                     perror("エラーat1856");
268                 }
269
270                 int i;
271                 i = ceil(r[1]);
272                 for(i;
273                     r[1] <= i && i <= p[1];
274                     i++){
275
276                     //撮像部分からはみ出ていないかのチェック
277                     if( 0 <= i &&
278                         i <= (HEIGHT - 1)){
279                         double x1 = func1(p, r, i);
280                         double x2 = func1(q, r, i);
281
282                         int j;
```

```
                        j = ceil(x1);

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

                            //描画する点の空間内のz座標.
                            double z =
                                FOCUS * ((n[0]*A[0]) + (n[1]*A[1]) + (n[2]*A[2]))
                                /
                                ((n[0]*(j-(WIDTH/2))) + (n[1]*(i-(HEIGHT/2))) + n[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] < z){}

                            else{
                                image[i][j][0] =
                                    -1 * ip * diffuse_color[0] *
                                    light_rgb[0] * MAX;
                                image[i][j][1] =
                                    -1 * ip * diffuse_color[1] *
                                    light_rgb[1] * MAX;
                                image[i][j][2] =
                                    -1 * ip * diffuse_color[2] *
                                    light_rgb[2] * MAX;

                                //zバッファの更新
                                z_buf[i][j] = 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];
            //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];
                memcpy(temp, p2, sizeof(double) * 2);
                memcpy(p2, p, sizeof(double) * 2);
                memcpy(p, temp, sizeof(double) * 2);
            }
            //分割して処理
            //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.
            shading(p, p2, q, n, A);
            shading(p, p2, r, n, A);
        }
    }
}

/* VRMLの読み込み */
/* ========================================================================== */
#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)
{
    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;
}
```

```
441 static int read_point(
442                       FILE *fp,
443                       Polygon *polygon,
444                       char *b)
445 {
446     int num=0;
447     while (getword(fp,b,MWS)>0) {
448         if (strindex(b,"[")>=0) break;
449     }
450     while (getword(fp,b,MWS)>0) {
451         if (strindex(b,"]")>=0) break;
452         else {
453             polygon->vtx[num++] = atof(b);
454         }
455     }
456     return num/3;
457 }
458
459 static int count_index(
460                        FILE *fp,
461                        char *b)
462 {
463     int num=0;
464     while (getword(fp,b,MWS)>0) {
465         if (strindex(b,"[")>=0) break;
466     }
467     while (getword(fp,b,MWS)>0) {
468         if (strindex(b,"]")>=0) break;
469         else {
470             num++;
471         }
472     }
473     if ( num %4 != 0 ) {
474         fprintf(stderr,"invalid␣file␣type[number␣of␣indices␣mismatch]\n");
475     }
476     return num/4;
477 }
478
479 static int read_index(
480                       FILE *fp,
481                       Polygon *polygon,
482                       char *b)
483 {
484     int num=0;
485     while (getword(fp,b,MWS)>0) {
486         if (strindex(b,"[")>=0) break;
487     }
488     while (getword(fp,b,MWS)>0) {
489         if (strindex(b,"]")>=0) break;
490         else {
491             polygon->idx[num++] = atoi(b);
492             if (num%3 == 0) getword(fp,b,MWS);
493         }
494     }
495     return num/3;
496 }
497
498 int read_one_obj(
499                  FILE *fp,
500                  Polygon *poly,
501                  Surface *surface)
502 {
503     char b[MWS];
504     int flag_material = 0;
505     int flag_point = 0;
506     int flag_index = 0;
507
508     /* initialize surface */
509     surface->diff[0] = 1.0;
510     surface->diff[1] = 1.0;
511     surface->diff[2] = 1.0;
512     surface->spec[0] = 0.0;
513     surface->spec[1] = 0.0;
514     surface->spec[2] = 0.0;
515     surface->ambi = 0.0;
516     surface->shine = 0.2;
517
518     if ( getword(fp,b,MWS) <= 0) return 0;
519
```

```
520     poly->vtx_num = 0;
521     poly->idx_num = 0;
522
523     while (flag_material==0 || flag_point==0 || flag_index==0) {
524         if      (strindex(b,"Material")>=0) {
525             getword(fp,b,MWS);
526             flag_material = 1;
527         }
528         else if (strindex(b,"point")>=0) {
529             fprintf(stderr,"Counting... [point]\n");
530             poly->vtx_num = count_point(fp, b);
531             flag_point = 1;
532         }
533         else if (strindex(b,"coordIndex")>=0) {
534             fprintf(stderr,"Counting... [coordIndex]\n");
535             poly->idx_num = count_index(fp, b);
536             flag_index = 1;
537         }
538         else if (getword(fp,b,MWS) <= 0) return 0;
539     }
540
541     flag_material = 0;
542     flag_point = 0;
543     flag_index = 0;
544
545     fseek(fp, 0, SEEK_SET);
546     poly->vtx = (double *)malloc(sizeof(double)*3*poly->vtx_num);
547     poly->idx = (int *)malloc(sizeof(int)*3*poly->idx_num);
548     while (flag_material==0 || flag_point==0 || flag_index==0) {
549         if      (strindex(b,"Material")>=0) {
550             fprintf(stderr,"Reading... [Material]\n");
551             read_material(fp,surface,b);
552             flag_material = 1;
553         }
554         else if (strindex(b,"point")>=0) {
555             fprintf(stderr,"Reading... [point]\n");
556             read_point(fp,poly,b);
557             flag_point = 1;
558         }
559         else if (strindex(b,"coordIndex")>=0) {
560             fprintf(stderr,"Reading... [coordIndex]\n");
561             read_index(fp,poly,b);
562             flag_index = 1;
563         }
564         else if (getword(fp,b,MWS) <= 0) return 0;
565     }
566
567     return 1;
568 }
569 /* ====================================================================== */
570
571 int main (int argc, char *argv[]){
572     /* VRML読み込み ============================================================ */
573     int i;
574     FILE *fp;
575     Polygon poly;
576     Surface surface;
577
578     fp = fopen(argv[1], "r");
579     read_one_obj(fp, &poly, &surface);
580
581     printf("\npoly.vtx_num\n");
582     fprintf(stderr,"%d vertice are found.\n",poly.vtx_num);
583     printf("\npoly.idx_num\n");
584     fprintf(stderr,"%d triangles are found.\n",poly.idx_num);
585
586     /* i th vertex */
587     printf("\npoly.vtx[i*3+0, 2, 3]\n");
588     for ( i = 0 ; i < poly.vtx_num ; i++ ) {
589         fprintf(stdout,"%f %f %f # %d th vertex\n",
590                 poly.vtx[i*3+0], poly.vtx[i*3+1], poly.vtx[i*3+2],
591                 i);
592     }
593
594     /* i th triangle */
595     printf("\npoly.idx[i*3+0, 2, 3]\n");
596     for ( i = 0 ; i < poly.idx_num ; i++ ) {
597         fprintf(stdout,"%d %d %d # %d th triangle\n",
598                 poly.idx[i*3+0], poly.idx[i*3+1], poly.idx[i*3+2],
```

```
599                 i);
600     }
601
602     /* material info */
603     fprintf(stderr, "diffuseColor␣%f␣%f␣%f\n", surface.diff[0], surface.diff[1], surface.diff[2]);
604     fprintf(stderr, "specularColor␣%f␣%f␣%f\n", surface.spec[0], surface.spec[1], surface.spec[2]);
605     fprintf(stderr, "ambientIntensity␣%f\n", surface.ambi);
606     fprintf(stderr, "shininess␣%f\n", surface.shine);
607     /* VRML読み込みここまで ========================================================== */
608
609     FILE *fp_ppm;
610     char *fname = argv[2];
611     fp_ppm = fopen(argv[2], "w");
612
613     //ファイルが開けなかったとき
614     if( fp_ppm == NULL ){
615         printf("%sファイルが開けません.\n", fname);
616         return -1;
617     }
618
619     //ファイルが開けたとき
620     else{
621         fprintf(stderr, "\n初期の頂点座標は以下\n");
622         for(int i = 0; i < poly.vtx_num; i++){
623             fprintf(stderr, "%f\t%f\t%f\n", poly.vtx[i*3+0], poly.vtx[i*3+1], poly.vtx[i*3+2]);
624         }
625         fprintf(stderr, "\n");
626
627         //描画領域を初期化
628         for(int i = 0; i < 256; i++){
629             for(int j = 0; j < 256; j++){
630                 image[i][j][0] = 0.0 * MAX;
631                 image[i][j][1] = 0.0 * MAX;
632                 image[i][j][2] = 0.0 * MAX;
633             }
634         }
635
636          //zバッファを初期化
637         for(int i = 0; i < 256; i++){
638             for(int j = 0; j < 256; j++){
639                 z_buf[i][j] = DBL_MAX;
640             }
641         }
642
643         //diffuse_colorの格納
644         diffuse_color[0] = surface.diff[0];
645         diffuse_color[1] = surface.diff[1];
646         diffuse_color[2] = surface.diff[2];
647
648         //シェーディング
649         //三角形ごとのループ
650         for(int i = 0; i < poly.idx_num; i++){
651             //各点の透視投影処理
652             for(int j = 0; j < 3; j++){
653                 double xp = poly.vtx[(poly.idx[i*3+j])*3 + 0];
654                 double yp = poly.vtx[(poly.idx[i*3+j])*3 + 1];
655                 double zp = poly.vtx[(poly.idx[i*3+j])*3 + 2];
656                 double zi = FOCUS;
657
658                 //debug
659                 if(zp == 0){
660                     printf("\n(%f\t%f\t%f)␣i=%d,␣j=%d\n", xp, yp, zp, i, j);
661                     perror("\nエラー0934\n");
662                     //break;
663                 }
664
665                 double xp2 = xp * (zi / zp);
666                 double yp2 = yp * (zi / zp);
667                 double zp2 = zi;
668
669                 //座標軸を平行移動
670                 projected_ver_buf[j][0] = (MAX / 2) + xp2;
671                 projected_ver_buf[j][1] = (MAX / 2) + yp2;
672             }
673
674             double a[2], b[2], c[2];
675             a[0] = projected_ver_buf[0][0];
676             a[1] = projected_ver_buf[0][1];
677             b[0] = projected_ver_buf[1][0];
```

```
678             b[1] = projected_ver_buf[1][1];
679             c[0] = projected_ver_buf[2][0];
680             c[1] = projected_ver_buf[2][1];
681
682             double A[3], B[3], C[3];
683             A[0] = poly.vtx[(poly.idx[i*3+0])*3 + 0];
684             A[1] = poly.vtx[(poly.idx[i*3+0])*3 + 1];
685             A[2] = poly.vtx[(poly.idx[i*3+0])*3 + 2];
686
687             B[0] = poly.vtx[(poly.idx[i*3+1])*3 + 0];
688             B[1] = poly.vtx[(poly.idx[i*3+1])*3 + 1];
689             B[2] = poly.vtx[(poly.idx[i*3+1])*3 + 2];
690
691             C[0] = poly.vtx[(poly.idx[i*3+2])*3 + 0];
692             C[1] = poly.vtx[(poly.idx[i*3+2])*3 + 1];
693             C[2] = poly.vtx[(poly.idx[i*3+2])*3 + 2];
694
695             //ベクトルAB, ACから外積を計算して
696             //法線ベクトルnを求める
697             double AB[3], AC[3], n[3];
698             AB[0] = B[0] - A[0];
699             AB[1] = B[1] - A[1];
700             AB[2] = B[2] - A[2];
701
702             AC[0] = C[0] - A[0];
703             AC[1] = C[1] - A[1];
704             AC[2] = C[2] - A[2];
705
706             n[0] = (AB[1] * AC[2]) - (AB[2] * AC[1]);
707             n[1] = (AB[2] * AC[0]) - (AB[0] * AC[2]);
708             n[2] = (AB[0] * AC[1]) - (AB[1] * AC[0]);
709
710             //長さを1に調整
711             double length_n =
712                 sqrt(pow(n[0], 2.0) +
713                      pow(n[1], 2.0) +
714                      pow(n[2], 2.0));
715
716             n[0] = n[0] / length_n;
717             n[1] = n[1] / length_n;
718             n[2] = n[2] / length_n;
719
720             //平面iの投影先の三角形をシェーディング
721             shading(a, b, c, n, A);
722         }
723
724
725         //ヘッダー出力
726         fputs(MAGICNUM, fp_ppm);
727         fputs("\n", fp_ppm);
728         fputs(WIDTH_STRING, fp_ppm);
729         fputs("␣", fp_ppm);
730         fputs(HEIGHT_STRING, fp_ppm);
731         fputs("\n", fp_ppm);
732         fputs(MAX_STRING, fp_ppm);
733         fputs("\n" ,fp_ppm);
734
735         //imageの出力
736         for(int i = 0; i < 256; i++){
737             for(int j = 0; j < 256; j++){
738                 char r[256];
739                 char g[256];
740                 char b[256];
741                 char str[1024];
742
743                 sprintf(r, "%d", (int)round(image[i][j][0]));
744                 sprintf(g, "%d", (int)round(image[i][j][1]));
745                 sprintf(b, "%d", (int)round(image[i][j][2]));
746                 sprintf(str, "%s\t%s\t%s\n", r, g, b);
747                 fputs(str, fp_ppm);
748             }
749         }
750
751     }
752     fclose(fp_ppm);
753     fclose(fp);
754
755     printf("\nppmファイル␣%s␣に画像を出力しました.\n", argv[2]);
756     return 1;
```

```
757 }
```

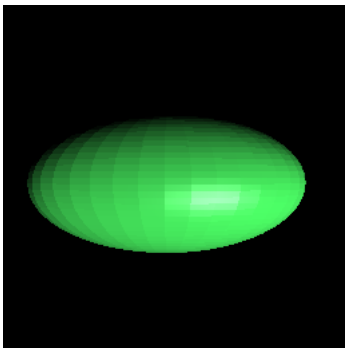

図 2　av4.wrl の出力結果

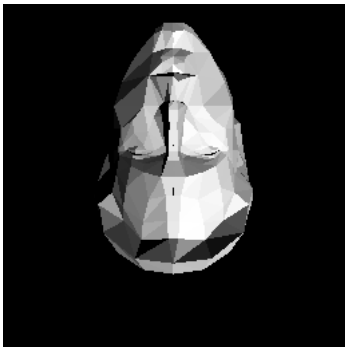

図 3　head.wrl の出力結果

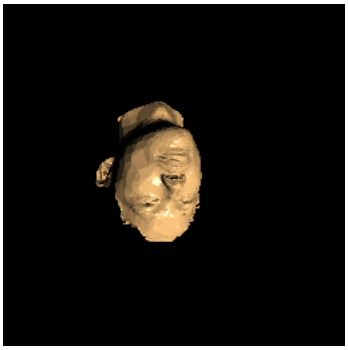

図 4　iiyama1997.wrl の出力結果