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 12 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 3 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=y 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=y 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 を求める計算である．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 での座標

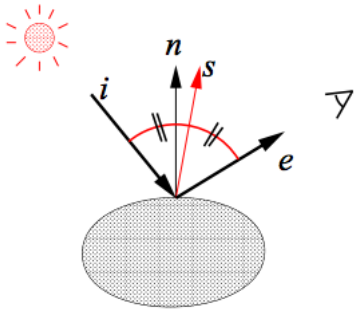

図 1　鏡面反射（テキストより）

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.

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      //printf("check x = %f\n", x);
62      //printf("check p[0] = %f\n", p[0]);
63      return x;
64  }
65
66  //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
67  //1直線上に無ければreturn 0;
68  //1直線上にあればreturn 1;
69  int lineOrNot(double *a, double *b, double *c){
70      if(a[0] == b[0]){
71          if(a[0] == c[0]){
72              return 1;
73          }
74          else{
75              return 0;
76          }
77      }
78      else{
79          if(c[1] == a[1] + ((b[1] - a[1]) / (b[0] - a[0])) * (c[0] - a[0])){
80              return 1;
81          }
82          else{
83              return 0;
84          }
85      }
86  }
87
88  //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
89  //引数a, b, cは投影平面上の3点
90  //eg)
91  //double a = {1.0, 2.0};
92  //nは法線ベクトル
93  //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.
94  //(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.)
95  void shading(double *a, double *b, double *c, double *n, double *A){
96      //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
97      if(lineOrNot(a, b, c) == 1){
98          //塗りつぶす点が無いので何もしない.
99      }
100     else{
101         //y座標の値が真ん中点をp、その他の点をq、rとする
102         //y座標の大きさはr <= p <= qの順
103         double p[2], q[2], r[2];
```

```
104         if(b[1] <= a[1] && a[1] <= c[1]){
105             memcpy(p, a, sizeof(double) * 2);
106             memcpy(q, c, sizeof(double) * 2);
107             memcpy(r, b, sizeof(double) * 2);
108         }
109         else{
110             if(c[1] <= a[1] && a[1] <= b[1]){
111                 memcpy(p, a, sizeof(double) * 2);
112                 memcpy(q, b, sizeof(double) * 2);
113                 memcpy(r, c, sizeof(double) * 2);
114             }
115             else{
116                 if(a[1] <= b[1] && b[1] <= c[1]){
117                     memcpy(p, b, sizeof(double) * 2);
118                     memcpy(q, c, sizeof(double) * 2);
119                     memcpy(r, a, sizeof(double) * 2);
120                 }
121                 else{
122                     if(c[1] <= b[1] && b[1] <= a[1]){
123                         memcpy(p, b, sizeof(double) * 2);
124                         memcpy(q, a, sizeof(double) * 2);
125                         memcpy(r, c, sizeof(double) * 2);
126                     }
127                     else{
128                         if(b[1] <= c[1] && c[1] <= a[1]){
129                             memcpy(p, c, sizeof(double) * 2);
130                             memcpy(q, a, sizeof(double) * 2);
131                             memcpy(r, b, sizeof(double) * 2);
132                         }
133                         else{
134                             if(a[1] <= c[1] && c[1] <= b[1]){
135                                 memcpy(p, c, sizeof(double) * 2);
136                                 memcpy(q, b, sizeof(double) * 2);
137                                 memcpy(r, a, sizeof(double) * 2);
138                             }
139                             else{
140                                 printf("エラーat2055\n");
141                                 printf("\na[1]=%f\tb[1]=%f\tc[1]=%f\n", a[1], b[1], c[1]);
142                                 perror(NULL);
143                             }
144                         }
145                     }
146                 }
147             }
148         }
149         //分割可能な三角形かを判定
150         if(p[1] == r[1] || p[1] == q[1]){
151             //分割できない
152 
153             //長さが1の光源方向ベクトルを作成する
154             //光源方向ベクトルの長さ
155             double length_1 =
156                 sqrt(pow(light_dir[0], 2.0) +
157                      pow(light_dir[1], 2.0) +
158                      pow(light_dir[2], 2.0));
159 
160             double light_dir_vec[3];
161             light_dir_vec[0] = light_dir[0] / length_1;
162             light_dir_vec[1] = light_dir[1] / length_1;
163             light_dir_vec[2] = light_dir[2] / length_1;
164 
165             //2パターンの三角形を特定
166             if(p[1] == r[1]){
167                 //debug
168                 //printf("\np[1] == r[1]\n");
169                 //x座標が p <= r となるように調整
170                 if(r[0] <  p[0]){
171                     double temp[2];
172                     memcpy(temp, r, sizeof(double) * 2);
173                     memcpy(r, p, sizeof(double) * 2);
174                     memcpy(p, temp, sizeof(double) * 2);
175                 }
176 
177                 //debug
178                 if(r[0] == p[0]){
179                   perror("エラーat958");
180                 }
181 
182                 //シェーディング処理
```

```
183                 //三角形pqrをシェーディング
184                 //y座標はp <= r
185                 //debug
186                 if(r[1] < p[1]){
187                     perror("エラーat1855");
188                 }
189
190                 /* 点(j, i)のシェーディング============================================== */
191                 int i;
192                 i = ceil(p[1]);
193                 for(i;
194                     p[1] <= i && i <= q[1];
195                     i++){
196
197                     //撮像平面からはみ出ていないかのチェック
198                     if(0 <= i
199                        &&
200                        i <= (HEIGHT - 1)){
201                             double x1 = func1(p, q, i);
202                             double x2 = func1(r, q, i);
203                             int j;
204                             j = ceil(x1);
205
206                             for(j;
207                                 x1 <= j && j <= x2 && 0 <= j && j <= (WIDTH - 1);
208                                 j++){
209
210
211                                 /* 鏡面反射を計算 */
212
213                                 //描画する点の投影前の空間内の座標.
214                                 double p_or[3];
215
216                                 p_or[0] =
217                                     (j-(WIDTH/2))
218                                     *
219                                     ((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                                 p_or[1] =
224                                     (i-(HEIGHT/2))
225                                     *
226                                     ((n[0]*A[0]) + (n[1]*A[1]) + (n[2]*A[2]))
227                                     /
228                                     ((n[0]*(j-(WIDTH/2))) + (n[1]*(i-(HEIGHT/2))) + n[2]*FOCUS);
229
230                                p_or[2] =
231                                     FOCUS
232                                     *
233                                     ((n[0]*A[0]) + (n[1]*A[1]) + (n[2]*A[2]))
234                                     /
235                                     ((n[0]*(j-(WIDTH/2))) + (n[1]*(i-(HEIGHT/2))) + n[2]*FOCUS);
236
237                                 /* 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 */
238                                 //視点方向は原点に固定
239                                 double e[3];
240                                 e[0] = -1 * p_or[0];
241                                 e[1] = -1 * p_or[1];
242                                 e[2] = -1 * p_or[2];
243                                 double length_e =
244                                     sqrt(pow(e[0], 2.0) +
245                                          pow(e[1], 2.0) +
246                                          pow(e[2], 2.0));
247                                 e[0] = (e[0] / length_e);
248                                 e[1] = (e[1] / length_e);
249                                 e[2] = (e[2] / length_e);
250
251                                 /* iは光源から描画中の点pへの入射光ベクトルを計算 */
252                                 //平行光源のため光源方向は
253                                 //const double light_dir[3] = {-1.0, -1.0, 2.0};
254                                 //を用いる
255                                 double i_vec[3];
256                                 i_vec[0] = light_dir[0];
257                                 i_vec[1] = light_dir[1];
258                                 i_vec[2] = light_dir[2];
259                                 double length_i =
260                                     sqrt(pow(i_vec[0], 2.0) +
261                                          pow(i_vec[1], 2.0) +
```

```
                                pow(i_vec[2], 2.0));
                        i_vec[0] = (i_vec[0] / length_i);
                        i_vec[1] = (i_vec[1] / length_i);
                        i_vec[2] = (i_vec[2] / length_i);

                        /* sベクトルを計算 */
                        double s[3];
                        s[0] = e[0] - i_vec[0];
                        s[1] = e[1] - i_vec[1];
                        s[2] = e[2] - i_vec[2];
                        double s_length =
                            sqrt(pow(s[0], 2.0) +
                                 pow(s[1], 2.0) +
                                 pow(s[2], 2.0));
                        s[0] = (s[0] / s_length);
                        s[1] = (s[1] / s_length);
                        s[2] = (s[2] / s_length);

                        //内積sn
                        double sn =
                            ((s[0] * n[0]) + (s[1] * n[1]) + (s[2] * n[2]));

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

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

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

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

                        else{
                            image[i][j][0] =
                                (-1 * ip * diffuse_color[0] * light_rgb[0] * MAX)
                                + (pow(sn, shininess) * specular_color[0] * light_rgb[0] * MAX)
                                ;

                            image[i][j][1] =
                                (-1 * ip * diffuse_color[1] * light_rgb[1] * MAX)
                                + (pow(sn, shininess) * specular_color[1] * light_rgb[1] * MAX)
                                ;

                            image[i][j][2] =
                                (-1 * ip * diffuse_color[2] * light_rgb[2] * MAX)
                                + (pow(sn, shininess) * specular_color[2] * light_rgb[2] * MAX)
                                ;

                            //zバッファの更新
                            z_buf[i][j] = p_or[2];
                        }
                    }
                    /* 点(j, i)のシェーディングここまで========================================== */
                }
                //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];
                memcpy(temp, q, sizeof(double) * 2);
                memcpy(q, p, sizeof(double) * 2);
                memcpy(p, temp, sizeof(double) * 2);
            }

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

            //シェーディング処理
```

```
340                 //三角形pqrをシェーディング
341                 //y座標はp <= q
342                 //debug
343                 if(q[1] < p[1]){
344                     perror("エラーat1856");
345                 }
346
347                 int i;
348                 i = ceil(r[1]);
349
350                 for(i;
351                     r[1] <= i && i <= p[1];
352                     i++){
353
354                     //撮像部分からはみ出ていないかのチェック
355                     if( 0 <= i &&
356                         i <= (HEIGHT - 1)){
357                         double x1 = func1(p, r, i);
358                         double x2 = func1(q, r, i);
359
360                         int j;
361                         j = ceil(x1);
362
363                         for(j;
364                             x1 <= j && j <= x2 && 0 <= j && j <= (WIDTH - 1);
365                             j++){
366
367                             /* 鏡面反射を適用 */
368                             //描画する点の投影前の空間内の座標.
369                             double p_or[3];
370
371                             p_or[0] =
372                                 (j-(WIDTH/2))
373                                 *
374                                 ((n[0]*A[0]) + (n[1]*A[1]) + (n[2]*A[2]))
375                                 /
376                                 ((n[0]*(j-(WIDTH/2))) + (n[1]*(i-(HEIGHT/2))) + n[2]*FOCUS);
377
378                             p_or[1] =
379                                 (i-(MAX/2))
380                                 *
381                                 ((n[0]*A[0]) + (n[1]*A[1]) + (n[2]*A[2]))
382                                 /
383                                 ((n[0]*(j-(WIDTH/2))) + (n[1]*(i-(HEIGHT/2))) + n[2]*FOCUS);
384
385                             p_or[2] =
386                                 FOCUS
387                                 *
388                                 ((n[0]*A[0]) + (n[1]*A[1]) + (n[2]*A[2]))
389                                 /
390                                 ((n[0]*(j-(WIDTH/2))) + (n[1]*(i-(HEIGHT/2))) + n[2]*FOCUS);
391
392                             /* 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 */
393                             //視点方向は原点に固定
394                             double e[3];
395                             e[0] = -1 * p_or[0];
396                             e[1] = -1 * p_or[1];
397                             e[2] = -1 * p_or[2];
398                             double length_e =
399                                 sqrt(pow(e[0], 2.0) +
400                                      pow(e[1], 2.0) +
401                                      pow(e[2], 2.0));
402                             e[0] = (e[0] / length_e);
403                             e[1] = (e[1] / length_e);
404                             e[2] = (e[2] / length_e);
405
406                             /* 光源から描画中の点pへの入射光ベクトルiを計算 */
407                             //平行光源のため光源方向は
408                             //const double light_dir[3] = {-1.0, -1.0, 2.0};
409                             //を用いる
410                             double i_vec[3];
411                             i_vec[0] = light_dir[0];
412                             i_vec[1] = light_dir[1];
413                             i_vec[2] = light_dir[2];
414                             double length_i =
415                                 sqrt(pow(i_vec[0], 2.0) +
416                                      pow(i_vec[1], 2.0) +
417                                      pow(i_vec[2], 2.0));
418                             i_vec[0] = (i_vec[0] / length_i);
```

```
                        i_vec[1] = (i_vec[1] / length_i);
                        i_vec[2] = (i_vec[2] / length_i);

                        /* sベクトルを計算 */
                        double s[3];
                        s[0] = e[0] - i_vec[0];
                        s[1] = e[1] - i_vec[1];
                        s[2] = e[2] - i_vec[2];
                        double s_length =
                            sqrt(pow(s[0], 2.0) + pow(s[1], 2.0) + pow(s[2], 2.0));
                        s[0] = (s[0] / s_length);
                        s[1] = (s[1] / s_length);
                        s[2] = (s[2] / s_length);

                        //内積sn
                        double sn = ((s[0] * n[0]) +
                                     (s[1] * n[1]) +
                                     (s[2] * n[2]));
                        if(sn <= 0){sn = 0;}

                        //拡散反射
                        /* 法線ベクトルnと光源方向ベクトルの内積を計算 */
                        double ip =
                            (n[0] * i_vec[0]) +
                            (n[1] * i_vec[1]) +
                            (n[2] * i_vec[2]);

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


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

                        else{

                            image[i][j][0] =
                                (-1 * ip * diffuse_color[0] * light_rgb[0] * MAX)
                                + (pow(sn, shininess) * specular_color[0] * light_rgb[0] * MAX)
                                ;

                            image[i][j][1] =
                                (-1 * ip * diffuse_color[1] * light_rgb[1] * MAX)
                                + (pow(sn, shininess) * specular_color[1] * light_rgb[1] * MAX)
                                ;

                            image[i][j][2] =
                                (-1 * ip * diffuse_color[2] * light_rgb[2] * MAX)
                                + (pow(sn, shininess) * specular_color[2] * light_rgb[2] * MAX)
                                ;

                            z_buf[i][j] = p_or[2];
                        }
                    }
                }
                //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);
    }
```

```
498     }
499 }
500
501 /* VRMLの読み込み */
502 /* ================================================================== */
503 #define MWS 256
504
505 static int strindex( char *s, char *t)
506 {
507     int         i, j, k;
508
509     for (i = 0; s[i] != '\0'; i++) {
510         for (j = i, k = 0;  t[k] != '\0' && s[j] == t[k]; j++, k++) ;
511         if (k > 0 && t[k] == '\0')
512             return i;
513     }
514     return -1;
515 }
516
517 static int getword(
518                     FILE *fp,
519                     char word[],
520                     int sl)
521 {
522     int i,c;
523
524     while ( (c = fgetc(fp)) != EOF && ( isspace(c) || c == '#')) {
525         if ( c == '#' ) {
526             while ( (c = fgetc(fp)) != EOF && c != '\n') ;
527             if ( c == EOF ) return (0);
528         }
529     }
530     if ( c == EOF )
531         return (0);
532     ungetc(c,fp);
533
534     for ( i = 0 ; i < sl - 1 ; i++) {
535         word[i] = fgetc(fp);
536         if ( isspace(word[i]) )
537             break;
538     }
539     word[i] = '\0';
540
541     return i;
542 }
543
544 static int read_material(
545                         FILE *fp,
546                         Surface *surface,
547                         char *b)
548 {
549     while (getword(fp,b,MWS)>0) {
550         if      (strindex(b,"}")>=0) break;
551         else if (strindex(b,"diffuseColor") >= 0) {
552             getword(fp,b,MWS);
553             surface->diff[0] = atof(b);
554             getword(fp,b,MWS);
555             surface->diff[1] = atof(b);
556             getword(fp,b,MWS);
557             surface->diff[2] = atof(b);
558         }
559         else if (strindex(b,"ambientIntensity") >= 0) {
560             getword(fp,b,MWS);
561             surface->ambi = atof(b);
562         }
563         else if (strindex(b,"specularColor") >= 0) {
564             getword(fp,b,MWS);
565             surface->spec[0] = atof(b);
566             getword(fp,b,MWS);
567             surface->spec[1] = atof(b);
568             getword(fp,b,MWS);
569             surface->spec[2] = atof(b);
570         }
571         else if (strindex(b,"shininess") >= 0) {
572             getword(fp,b,MWS);
573             surface->shine = atof(b);
574         }
575     }
576     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;
}
```

```
656 int read_one_obj(
657                  FILE *fp,
658                  Polygon *poly,
659                  Surface *surface)
660 {
661     char b[MWS];
662     int flag_material = 0;
663     int flag_point = 0;
664     int flag_index = 0;
665 
666     /* initialize surface */
667     surface->diff[0] = 1.0;
668     surface->diff[1] = 1.0;
669     surface->diff[2] = 1.0;
670     surface->spec[0] = 0.0;
671     surface->spec[1] = 0.0;
672     surface->spec[2] = 0.0;
673     surface->ambi = 0.0;
674     surface->shine = 0.2;
675 
676     if ( getword(fp,b,MWS) <= 0) return 0;
677 
678     poly->vtx_num = 0;
679     poly->idx_num = 0;
680 
681     while (flag_material==0 || flag_point==0 || flag_index==0) {
682         if      (strindex(b,"Material")>=0) {
683             getword(fp,b,MWS);
684             flag_material = 1;
685         }
686         else if (strindex(b,"point")>=0) {
687             fprintf(stderr,"Counting...␣[point]\n");
688             poly->vtx_num = count_point(fp, b);
689             flag_point = 1;
690         }
691         else if (strindex(b,"coordIndex")>=0) {
692             fprintf(stderr,"Counting...␣[coordIndex]\n");
693             poly->idx_num = count_index(fp, b);
694             flag_index = 1;
695         }
696         else if (getword(fp,b,MWS) <= 0) return 0;
697     }
698 
699     flag_material = 0;
700     flag_point = 0;
701     flag_index = 0;
702 
703     fseek(fp, 0, SEEK_SET);
704     poly->vtx = (double *)malloc(sizeof(double)*3*poly->vtx_num);
705     poly->idx = (int *)malloc(sizeof(int)*3*poly->idx_num);
706     while (flag_material==0 || flag_point==0 || flag_index==0) {
707         if      (strindex(b,"Material")>=0) {
708             fprintf(stderr,"Reading...␣[Material]\n");
709             read_material(fp,surface,b);
710             flag_material = 1;
711         }
712         else if (strindex(b,"point")>=0) {
713             fprintf(stderr,"Reading...␣[point]\n");
714             read_point(fp,poly,b);
715             flag_point = 1;
716         }
717         else if (strindex(b,"coordIndex")>=0) {
718             fprintf(stderr,"Reading...␣[coordIndex]\n");
719             read_index(fp,poly,b);
720             flag_index = 1;
721         }
722         else if (getword(fp,b,MWS) <= 0) return 0;
723     }
724 
725     return 1;
726 }
727 /* ========================================================================= */
728 
729 
730 
731 
732 int main (int argc, char *argv[]){
733     /* VRML読み込み ============================================================ */
734     int i;
```

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

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

```
893
894         //ヘッダー出力
895         fputs(MAGICNUM, fp_ppm);
896         fputs("\n", fp_ppm);
897         fputs(WIDTH_STRING, fp_ppm);
898         fputs("␣", fp_ppm);
899         fputs(HEIGHT_STRING, fp_ppm);
900         fputs("\n", fp_ppm);
901         fputs(MAX_STRING, fp_ppm);
902         fputs("\n" ,fp_ppm);
903
904         //image の出力
905         for(int i = 0; i < 256; i++){
906             for(int j = 0; j < 256; j++){
907                 char r[256];
908                 char g[256];
909                 char b[256];
910                 char str[1024];
911
912                 sprintf(r, "%d", (int)round(image[i][j][0]));
913                 sprintf(g, "%d", (int)round(image[i][j][1]));
914                 sprintf(b, "%d", (int)round(image[i][j][2]));
915                 sprintf(str, "%s\t%s\t%s\n", r, g, b);
916                 fputs(str, fp_ppm);
917             }
918         }
919     }
920     fclose(fp_ppm);
921     fclose(fp);
922
923     printf("\nppm ファイル␣%s␣に画像を出力しました.\n", fname );
924     return 1;
925 }
```

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. シェルスクリプト実行時に要求される引数で読み込む VRML ファイルを選択することができる. 出力画像は図 2、図 3、図 4 のようになった.

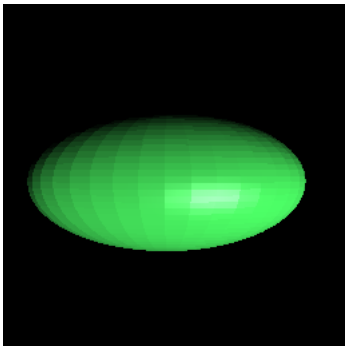

図 2　av4.wrl の出力結果

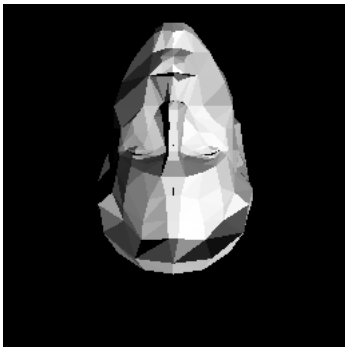

図 3　head.wrl の出力結果

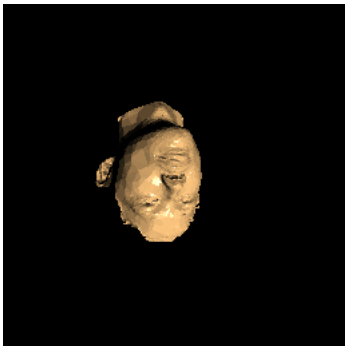

図 4　iiyama1997.wrl の出力結果

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．しかしこの点について自分として少し考えた部分がある．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直接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ということ、さらには多数の引数分のメモリを（z_buf などは殆どの場合要素数が 256×256 個以上になる）関数呼び出し、再帰呼出しの度にスタック領域に確保しなければならず、無駄が多いと考えたためである．

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)
```

```
44  //double p[2] = (1.0, 2.0);
45  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{
125                         if(b[1] <= c[1] && c[1] <= a[1]){
126                             memcpy(p, c, sizeof(double) * 2);
127                             memcpy(q, a, sizeof(double) * 2);
128                             memcpy(r, b, sizeof(double) * 2);
129                         }
130                         else{
131                             if(a[1] <= c[1] && c[1] <= b[1]){
132                                 memcpy(p, c, sizeof(double) * 2);
133                                 memcpy(q, b, sizeof(double) * 2);
134                                 memcpy(r, a, sizeof(double) * 2);
135                             }
136                             else{
137                                 printf("エラーat2055\n");
138                                 printf("\na[1]=%f\tb[1]=%f\tc[1]=%f\n", a[1], b[1], c[1]);
139                                 perror(NULL);
140                             }
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             // 法線ベクトルnと光源方向ベクトルの内積
164             double ip =
165                 (n[0] * light_dir_vec[0]) +
166                 (n[1] * light_dir_vec[1]) +
167                 (n[2] * light_dir_vec[2]);
168
169             if(0 <= ip){
170                 ip = 0;
171             }
172
173             //2パターンの三角形を特定
174             if(p[1] == r[1]){
175                 //debug
176                 //printf("\np[1] == r[1]\n");
177                 //x座標が p <= r となるように調整
178                 if(r[0] <  p[0]){
179                     double temp[2];
180                     memcpy(temp, r, sizeof(double) * 2);
181                     memcpy(r, p, sizeof(double) * 2);
182                     memcpy(p, temp, sizeof(double) * 2);
183                 }
184
185                 //debug
186                 if(r[0] == p[0]){
187                   perror("エラーat958");
188                 }
189
190                 //シェーディング処理
191                 //三角形pqrをシェーディング
192                 //y座標はp <= r
193                 //debug
194                 if(r[1] < p[1]){
195                     perror("エラーat1855");
196                 }
197
198                 int i;
199                 i = ceil(p[1]);
200                 for(i;
201                     p[1] <= i && i <= q[1];
```

```
202                     i++){
203
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);
```

```
                        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;
```

```
}

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;

```

```
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 }
```