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 2

工学部情報学科 3 回生 1029255242
勝見久央

作成日: 2015 年 11 月 3 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 1 のプログラム kadai01.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 ReportForKadai01.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んだ.
- 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

本課題以降の課題では 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、使用の方法としては vrml.c の main 関数は、自分で作成した main 関数の内部に取り込み、その他の関数についてはそのま

ま使用した. また、ヘッダファイルも作成したファイルごとにそのまま参照している. その他の主な変更点については次に示す.

-ADDED! z_buf[HEIGHT][WIDTH]
-DEPRECATED FILENAME
-ADDED! projected_ver_buf[3][2]
-DEPRICATED! void perspective_pro()
-MODIFIED! void shading(double *a, double *b, double *c, double *n, double *A)
-ADDED! static int strindex(char *s, char *t)
-ADDED! static int getword()
-ADDED! static int read_material(FILE *fp, Surface *surface, char *b)
-ADDED! static int count_point(FILE *fp, char *b)
-ADDED! static int read_point(FILE *fp, Polygon *polygon, char *b)
-ADDED! static int count_index(FILE *fp, char *b)
-ADDED! static int read_index(FILE *fp, Polygon *polygon, char *b)
-ADDED! int read_one_obj(FILE *fp, Polygon *poly, Surface *surface)
-MODIFIED! main(int argc, char *argv[])

リスト 1　vrml.c

```
/* VRML 2.0 Reader
 *
 * ver1.1 2005/10/06 Masaaki IIYAMA (bug fix)
 * ver1.0 2005/09/27 Masaaki IIYAMA
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include "vrml.h"


/*
//////////////////////////////////////////////////////////////////
*/
#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;
  }
```

```
122   while (getword(fp,b,MWS)>0) {
123     if (strindex(b,"]")>=0) break;
124     else {
125       polygon->vtx[num++] = atof(b);
126     }
127   }
128   return num/3;
129 }
130 
131 static int count_index(
132            FILE *fp,
133            char *b)
134 {
135   int num=0;
136   while (getword(fp,b,MWS)>0) {
137     if (strindex(b,"[")>=0) break;
138   }
139   while (getword(fp,b,MWS)>0) {
140     if (strindex(b,"]")>=0) break;
141     else {
142       num++;
143     }
144   }
145   if ( num %4 != 0 ) {
146     fprintf(stderr,"invalid␣file␣type[number␣of␣indices␣mismatch]\n");
147   }
148   return num/4;
149 }
150 
151 static int read_index(
152           FILE *fp,
153           Polygon *polygon,
154           char *b)
155 {
156   int num=0;
157   while (getword(fp,b,MWS)>0) {
158     if (strindex(b,"[")>=0) break;
159   }
160   while (getword(fp,b,MWS)>0) {
161     if (strindex(b,"]")>=0) break;
162     else {
163       polygon->idx[num++] = atoi(b);
164       if (num%3 == 0) getword(fp,b,MWS);
165     }
166   }
167   return num/3;
168 }
169 
170 int read_one_obj(
171      FILE *fp,
172      Polygon *poly,
173      Surface *surface)
174 {
175   char b[MWS];
176   int flag_material = 0;
177   int flag_point = 0;
178   int flag_index = 0;
179 
180   /* initialize surface */
181   surface->diff[0] = 1.0;
182   surface->diff[1] = 1.0;
183   surface->diff[2] = 1.0;
184   surface->spec[0] = 0.0;
185   surface->spec[1] = 0.0;
186   surface->spec[2] = 0.0;
187   surface->ambi = 0.0;
188   surface->shine = 0.2;
189 
190   if ( getword(fp,b,MWS) <= 0) return 0;
191 
192   poly->vtx_num = 0;
193   poly->idx_num = 0;
194 
195   while (flag_material==0 || flag_point==0 || flag_index==0) {
196     if      (strindex(b,"Material")>=0) {
197       getword(fp,b,MWS);
198       flag_material = 1;
199     }
200     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;
}

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

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

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

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

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

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

リスト 2　vrml.h

```
/* VRML 2.0 Reader
 *
 * ver1.0 2005/09/27 Masaaki IIYAMA
 *
 */

typedef unsigned char uchar;

typedef struct {
  int w; /* image width */
  int h; /* image height */
  uchar *pix; /* pointer to the PPM image buffer */
  double *z; /* pointer to the z buffer */
} Image;

typedef struct {
  uchar  i[3]; /* Light Intensity in RGB */
  double d[3]; /* Light Direction */
  double p[3]; /* Light Source Location */
  int    light_type; /* 0:parallel, 1:point */
  int    color_type; /* 0:diffuse, 1:specular, 2:ambient */
} Light;

typedef struct {
  double p[3]; /* Camera Position */
  double d[3]; /* Camera Direction */
  double f;    /* Focus */
} Camera;

typedef struct {
  double diff[3]; /* Diffuse in RGB */
  double spec[3]; /* Specular in RGB */
  double ambi; /* Ambient */
  double shine;   /* Shiness */
} Surface;

typedef struct {
  double *vtx; /* Vertex List */
  int    *idx; /* Index List */
  int    vtx_num; /* number of vertice */
  int    idx_num; /* number of indices */
} Polygon;

int read_one_obj(
     FILE *fp,
     Polygon *poly,
     Surface *surface);
```

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. kadai01.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 ポリゴンデータ

課題 1 のプログラム kadai01.c ではポリゴンデータを関数内部で発生させていたが、課題 2 以降では VRML としてファイルから取り込むため、ポリゴンデータを指定する定数は記述していない.

3.2.3 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.

3.2.4 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]
  double 型 2 次元配列. kadai01.c での projected_ver と格納される頂点の意味が異なるため、名称を変更をした. kadai01.c では先に全ての頂点をまず透視投影し、その結果を配列 projected_ver に格納し、その後の操作ではそこから参照を行って使用していたが、kadai02.c 以降では、空間内の三角形 i についてシェーディングを行うという処理をループし、処理を行う 3 頂点ごとに毎回透視投影を施し、その結果を prjected_ver_buf に格納し、シェーディング時に参照する処理とした. これは、ループ処理の構造を設計しやすくするためであり、また、透視投影処理の計算の計算量は少ない処理で済むということ、頂点座標の数が膨大になると、膨大なメモリ量を確保したにも関わらず、一度のループで使用する頂点座標が 3 点のみであるため、直後の処理で使用しない大量の頂点座標のデータのためのメモリ領域を終始確保しなければならず、ハード面での無駄が多い、といった点を考慮した結果である.

3.3 関数外部仕様

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

kadai01.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)

kadai01.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、外部しようとしては kadai01.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)

kadai01.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)

kadai01.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)

kadai01.c のものを拡張、変更. 主な、変更点としては、shading 関数の引数に、シェーディングを行う三角形の法線ベクトルと、そのうちの一点の座標（ここでは、点 A の座標を用いることとしているが、本来は三角形平面上の一点であればどんなものでも良い.）を加えた. これは、シェーディング時に描画中の三角形内部の点の座標の、投影平面上の xy 座標から元の空間座標を算出し、z バッファと照らしあわせて描画するかどうかを判定するために使用する. 具体的には、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 での座標を (x_A, y_A, z_A)、投影平面の z 座標を z_p とすると、

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

として、求めることができる. なお、実際のプログラムでは投影後の点の座標の位置がカメラの中心に周辺に来るように、平行移動による修正を加えているため、計算時は $x_p - fracWIDTH2$、$y_p - fracHEIGHT2$ を使用した. これによって算出された値と、z バッファの値を比較し、z バッファよりも大きければ描画せずに次の点の描画に移り、そうでなければ z バッファの値を算出した z 座標の値に書き換えてシェーディングを行う. なお、引数として用いる法線ベクトルと三角形上の点 A の座標は main 関数内で計算し、shading 関数内の再帰呼び出し時は、三角形を分割しても、元の三角形を含む平面とその上の一点の座標は不変であるから、同じものを使って計算することができる.

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

kadai01.c のものを大幅変更、拡張. 作成する PPM ファイルの名前をコマンドライン引数として受け取って指定するよう変更を加えた. 前半部分ではコマンドライン引数として受け取った VRML ファイルを読み込む. 後半部分では三角形ごとにループを行い、その三点のシェーディングを行うため必要な投影平面上での座標の計算、法線ベクトルの算出（外積から求まる）を行い、その結果を shading 関数に引き渡す. そして、最後に画像の出力を行う.

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 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 FILENAME "image.ppm"
#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){
    double x;
    if(p[1] > q[1]){
        x = ((p[0] * (y - q[1])) + (q[0] * (p[1] - y))) / (p[1] - q[1]);
    }
    if(p[1] < q[1]){
        x = ((q[0] * (y - p[1])) + (p[0] * (q[1] - y))) / (q[1] - p[1]);
    }
    if(p[1] == q[1]){
```

```
56          //解なし
57          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"
58                  , p[0], p[1], q[0], q[1]);
59          perror(NULL);
60          return -1;
61      }
62      return x;
63  }
64
65  //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
66  //1直線上に無ければreturn 0;
67  //1直線上にあればreturn 1;
68  int lineOrNot(double *a, double *b, double *c){
69      if(a[0] == b[0]){
70          if(a[0] == c[0]){
71              return 1;
72          }
73          else{
74              return 0;
75          }
76      }
77      else{
78          if(c[1] == a[1] + ((b[1] - a[1]) / (b[0] - a[0])) * (c[0] - a[0])){
79              return 1;
80          }
81          else{
82              return 0;
83          }
84      }
85  }
86
87  //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
88  //引数a, b, cは投影平面上の3点
89  //eg)
90  //double a = {1.0, 2.0};
91  //nは法線ベクトル
92  //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.
93  //(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.)
94  void shading(double *a, double *b, double *c, double *n, double *A){
95      //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
96      if(lineOrNot(a, b, c) == 1){
97          //塗りつぶす点が無いので何もしない.
98      }
99      else{
100         //y座標の値が真ん中点をp、その他の点をq、rとする
101         //y座標の大きさはr <= p <= qの順
102         double p[2], q[2], r[2];
103         if(b[1] <= a[1] && a[1] <= c[1]){
104             memcpy(p, a, sizeof(double) * 2);
105             memcpy(q, c, sizeof(double) * 2);
106             memcpy(r, b, sizeof(double) * 2);
107         }
108         else{
109             if(c[1] <= a[1] && a[1] <= b[1]){
110                 memcpy(p, a, sizeof(double) * 2);
111                 memcpy(q, b, sizeof(double) * 2);
112                 memcpy(r, c, sizeof(double) * 2);
113             }
114             else{
115                 if(a[1] <= b[1] && b[1] <= c[1]){
116                     memcpy(p, b, sizeof(double) * 2);
117                     memcpy(q, c, sizeof(double) * 2);
118                     memcpy(r, a, sizeof(double) * 2);
119                 }
120                 else{
121                     if(c[1] <= b[1] && b[1] <= a[1]){
122                         memcpy(p, b, sizeof(double) * 2);
123                         memcpy(q, a, sizeof(double) * 2);
124                         memcpy(r, c, sizeof(double) * 2);
125                     }
126                     else{
127                         if(b[1] <= c[1] && c[1] <= a[1]){
128                             memcpy(p, c, sizeof(double) * 2);
129                             memcpy(q, a, sizeof(double) * 2);
130                             memcpy(r, b, sizeof(double) * 2);
131                         }
132                         else{
133                             if(a[1] <= c[1] && c[1] <= b[1]){
134                                 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++){

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

```
214                         int j;
215                         j = ceil(x1);
216
217                         for(j;
218                             x1 <= j && j <= x2 && 0 <= j && j <= (WIDTH - 1);
219                             j++){
220
221                             //描画する点の空間内のz座標.
222                             double z =
223                                 FOCUS * ((n[0]*A[0]) + (n[1]*A[1]) + (n[2]*A[2]))
224                                 /
225                                 ((n[0]*(j-(WIDTH/2))) + (n[1]*(i-(HEIGHT/2))) + n[2]*FOCUS);
226
227                             //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）
228                             if(z_buf[i][j] < z){
229                                 //描画しない
230
231                             }
232
233                             else{
234                                 image[i][j][0] =
235                                     -1 * ip * diffuse_color[0] *
236                                     light_rgb[0] * MAX;
237
238                                 image[i][j][1] =
239                                     -1 * ip * diffuse_color[1] *
240                                     light_rgb[1] * MAX;
241                                 image[i][j][2] =
242                                     -1 * ip * diffuse_color[2] *
243                                     light_rgb[2] * MAX;
244
245                                 //zバッファの更新
246                                 z_buf[i][j] = z;
247                             }
248                         }
249                     }
250                     //はみ出ている場合は描画しない
251                     else{}
252                 }
253
254             }
255
256             if(p[1] == q[1]){
257                 //x座標が p < q となるように調整
258                 if(q[0] <  p[0]){
259                     double temp[2];
260                     memcpy(temp, q, sizeof(double) * 2);
261                     memcpy(q, p, sizeof(double) * 2);
262                     memcpy(p, temp, sizeof(double) * 2);
263                 }
264
265                 //debug
266                 if(q[0] == p[0]){
267                     perror("エラーat1011");
268                 }
269
270                 //シェーディング処理
271                 //三角形pqrをシェーディング
272                 //y座標はp <= q
273
274                 //debug
275                 if(q[1] < p[1]){
276                     perror("エラーat1856");
277                 }
278
279                 int i;
280                 i = ceil(r[1]);
281                 for(i;
282                     r[1] <= i && i <= p[1];
283                     i++){
284
285                     //撮像部分からはみ出ていないかのチェック
286                     if( 0 <= i &&
287                         i <= (HEIGHT - 1)){
288                         double x1 = func1(p, r, i);
289                         double x2 = func1(q, r, i);
290
291                         int j;
292                         j = ceil(x1);
```

```
293
294                             for(j;
295                                 x1 <= j && j <= x2 && 0 <= j && j <= (WIDTH - 1);
296                                 j++){
297
298                                 //描画する点の空間内のz座標.
299                                 double z =
300                                     FOCUS * ((n[0]*A[0]) + (n[1]*A[1]) + (n[2]*A[2]))
301                                     /
302                                     ((n[0]*(j-(WIDTH/2))) + (n[1]*(i-(HEIGHT/2))) + n[2]*FOCUS);
303
304                                 //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）
305                                 if(z_buf[i][j] < z){
306                                     //描画しない
307                                 }
308
309                                 else{
310                                     image[i][j][0] =
311                                         -1 * ip * diffuse_color[0] *
312                                         light_rgb[0] * MAX;
313                                     image[i][j][1] =
314                                         -1 * ip * diffuse_color[1] *
315                                         light_rgb[1] * MAX;
316                                     image[i][j][2] =
317                                         -1 * ip * diffuse_color[2] *
318                                         light_rgb[2] * MAX;
319
320                                     //zバッファの更新
321                                     z_buf[i][j] = z;
322                                 }
323                             }
324                         }
325                         //撮像平面からはみ出る部分は描画しない
326                         else{}
327                     }
328                 }
329
330             }
331             //分割できる
332             //分割してそれぞれ再帰的に処理
333             //分割後の三角形はpp2qとpp2r
334             else{
335
336                 double p2[2];
337
338                 p2[0] = func1(q, r, p[1]);
339                 p2[1] = p[1];
340                 //p2のほうがpのx座標より大きくなるようにする
341                 if(p2[0] < p[0]){
342                     double temp[2];
343                     memcpy(temp, p2, sizeof(double) * 2);
344                     memcpy(p2, p, sizeof(double) * 2);
345                     memcpy(p, temp, sizeof(double) * 2);
346                 }
347                 //分割して処理
348                 //分割しても同一平面上なので法線ベクトルと
349                 //平面上の任意の点は同じものを使える.
350                 shading(p, p2, q, n, A);
351                 shading(p, p2, r, n, A);
352             }
353         }
354     }
355
356
357
358
359
360 /* VRML 2.0 Reader
361  *
362  * ver1.1 2005/10/06 Masaaki IIYAMA (bug fix)
363  * ver1.0 2005/09/27 Masaaki IIYAMA
364  *
365  */
366 /*
367 ////////////////////////////////////////////////////////////////
368 */
369 #define MWS 256
370
371 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) {
```

```
451         if (strindex(b,"[")>=0) break;
452     }
453     while (getword(fp,b,MWS)>0) {
454         if (strindex(b,"]")>=0) break;
455         else {
456             num++;
457         }
458     }
459     if ( num %3 != 0 ) {
460         fprintf(stderr,"invalid␣file␣type[number␣of␣points␣mismatch]\n");
461     }
462     return num/3;
463 }
464 
465 static int read_point(
466                       FILE *fp,
467                       Polygon *polygon,
468                       char *b)
469 {
470     int num=0;
471     while (getword(fp,b,MWS)>0) {
472         if (strindex(b,"[")>=0) break;
473     }
474     while (getword(fp,b,MWS)>0) {
475         if (strindex(b,"]")>=0) break;
476         else {
477             polygon->vtx[num++] = atof(b);
478         }
479     }
480     return num/3;
481 }
482 
483 static int count_index(
484                        FILE *fp,
485                        char *b)
486 {
487     int num=0;
488     while (getword(fp,b,MWS)>0) {
489         if (strindex(b,"[")>=0) break;
490     }
491     while (getword(fp,b,MWS)>0) {
492         if (strindex(b,"]")>=0) break;
493         else {
494             num++;
495         }
496     }
497     if ( num %4 != 0 ) {
498         fprintf(stderr,"invalid␣file␣type[number␣of␣indices␣mismatch]\n");
499     }
500     return num/4;
501 }
502 
503 static int read_index(
504                       FILE *fp,
505                       Polygon *polygon,
506                       char *b)
507 {
508     int num=0;
509     while (getword(fp,b,MWS)>0) {
510         if (strindex(b,"[")>=0) break;
511     }
512     while (getword(fp,b,MWS)>0) {
513         if (strindex(b,"]")>=0) break;
514         else {
515             polygon->idx[num++] = atoi(b);
516             if (num%3 == 0) getword(fp,b,MWS);
517         }
518     }
519     return num/3;
520 }
521 
522 int read_one_obj(
523                  FILE *fp,
524                  Polygon *poly,
525                  Surface *surface)
526 {
527     char b[MWS];
528     int flag_material = 0;
529     int flag_point = 0;
```

```
530     int flag_index = 0;
531
532     /* initialize surface */
533     surface->diff[0] = 1.0;
534     surface->diff[1] = 1.0;
535     surface->diff[2] = 1.0;
536     surface->spec[0] = 0.0;
537     surface->spec[1] = 0.0;
538     surface->spec[2] = 0.0;
539     surface->ambi = 0.0;
540     surface->shine = 0.2;
541
542     if ( getword(fp,b,MWS) <= 0) return 0;
543
544     poly->vtx_num = 0;
545     poly->idx_num = 0;
546
547     while (flag_material==0 || flag_point==0 || flag_index==0) {
548         if      (strindex(b,"Material")>=0) {
549             getword(fp,b,MWS);
550             flag_material = 1;
551         }
552         else if (strindex(b,"point")>=0) {
553             fprintf(stderr,"Counting...␣[point]\n");
554             poly->vtx_num = count_point(fp, b);
555             flag_point = 1;
556         }
557         else if (strindex(b,"coordIndex")>=0) {
558             fprintf(stderr,"Counting...␣[coordIndex]\n");
559             poly->idx_num = count_index(fp, b);
560             flag_index = 1;
561         }
562         else if (getword(fp,b,MWS) <= 0) return 0;
563     }
564
565     flag_material = 0;
566     flag_point = 0;
567     flag_index = 0;
568
569     fseek(fp, 0, SEEK_SET);
570     poly->vtx = (double *)malloc(sizeof(double)*3*poly->vtx_num);
571     poly->idx = (int *)malloc(sizeof(int)*3*poly->idx_num);
572     while (flag_material==0 || flag_point==0 || flag_index==0) {
573         if      (strindex(b,"Material")>=0) {
574             fprintf(stderr,"Reading...␣[Material]\n");
575             read_material(fp,surface,b);
576             flag_material = 1;
577         }
578         else if (strindex(b,"point")>=0) {
579             fprintf(stderr,"Reading...␣[point]\n");
580             read_point(fp,poly,b);
581             flag_point = 1;
582         }
583         else if (strindex(b,"coordIndex")>=0) {
584             fprintf(stderr,"Reading...␣[coordIndex]\n");
585             read_index(fp,poly,b);
586             flag_index = 1;
587         }
588         else if (getword(fp,b,MWS) <= 0) return 0;
589     }
590
591     return 1;
592 }
593
594
595 int main (int argc, char *argv[])
596 {
597     int i;
598     FILE *fp;
599     Polygon poly;
600     Surface surface;
601
602     fp = fopen(argv[1], "r");
603     read_one_obj(fp, &poly, &surface);
604
605     printf("\npoly.vtx_num\n");
606     fprintf(stderr,"%d␣vertice␣are␣found.\n",poly.vtx_num);
607     printf("\npoly.idx_num\n");
608     fprintf(stderr,"%d␣triangles␣are␣found.\n",poly.idx_num);
```

```
609
610     /* i th vertex */
611     printf("\npoly.vtx[i*3+0,␣2,␣3]\n");
612     for ( i = 0 ; i < poly.vtx_num ; i++ ) {
613         fprintf(stdout,"%f␣%f␣%f␣#␣%d␣th␣vertex\n",
614                 poly.vtx[i*3+0], poly.vtx[i*3+1], poly.vtx[i*3+2],
615                 i);
616     }
617
618     /* i th triangle */
619     printf("\npoly.idx[i*3+0,␣2,␣3]\n");
620     for ( i = 0 ; i < poly.idx_num ; i++ ) {
621         fprintf(stdout,"%d␣%d␣%d␣#␣%d␣th␣triangle\n",
622                 poly.idx[i*3+0], poly.idx[i*3+1], poly.idx[i*3+2],
623                 i);
624     }
625
626     /* material info */
627     fprintf(stderr, "diffuseColor␣%f␣%f␣%f\n", surface.diff[0], surface.diff[1], surface.diff[2]);
628     fprintf(stderr, "specularColor␣%f␣%f␣%f\n", surface.spec[0], surface.spec[1], surface.spec[2]);
629     fprintf(stderr, "ambientIntensity␣%f\n", surface.ambi);
630     fprintf(stderr, "shininess␣%f\n", surface.shine);
631
632     //=====================================================================
633     //=====================================================================
634     //=====================================================================
635     //=====================================================================
636
637     FILE *fp_ppm;
638     char *fname = FILENAME;
639
640
641     fp_ppm = fopen(argv[2], "w" );
642
643     //ファイルが開けなかったとき
644     if( fp_ppm == NULL ){
645         printf("%sファイルが開けません.\n", fname);
646         return -1;
647     }
648
649     //ファイルが開けたとき
650     else{
651         fprintf(stderr, "\n初期の頂点座標は以下\n");
652         for(int i = 0; i < poly.vtx_num; i++){
653             //fprintf(stderr, "%f\t%f\t%f\n", ver[i][0], ver[i][1], ver[i][2]);
654             fprintf(stderr, "%f\t%f\t%f\n", poly.vtx[i*3+0], poly.vtx[i*3+1], poly.vtx[i*3+2]);
655         }
656         fprintf(stderr, "\n");
657
658         //描画領域を初期化
659         for(int i = 0; i < 256; i++){
660             for(int j = 0; j < 256; j++){
661                 image[i][j][0] = 0.0 * MAX;
662                 image[i][j][1] = 0.0 * MAX;
663                 image[i][j][2] = 0.0 * MAX;
664             }
665         }
666
667          //zバッファを初期化
668         for(int i = 0; i < 256; i++){
669             for(int j = 0; j < 256; j++){
670                 z_buf[i][j] = DBL_MAX;
671             }
672         }
673
674         //diffuse_colorの格納
675         diffuse_color[0] = surface.diff[0];
676         diffuse_color[1] = surface.diff[1];
677         diffuse_color[2] = surface.diff[2];
678
679         //シェーディング
680         //三角形ごとのループ
681         for(int i = 0; i < poly.idx_num; i++){
682             //各点の透視投影処理
683             for(int j = 0; j < 3; j++){
684                 double xp = poly.vtx[(poly.idx[i*3+j])*3 + 0];
685                 double yp = poly.vtx[(poly.idx[i*3+j])*3 + 1];
686                 double zp = poly.vtx[(poly.idx[i*3+j])*3 + 2];
687                 double zi = FOCUS;
```

```
688
689                 //debug
690                 if(zp == 0){
691                     printf("\n(%f\t%f\t%f) i=%d, j=%d\n", xp, yp, zp, i, j);
692                     perror("\nエラー0934\n");
693                     //break;
694                 }
695
696                 double xp2 = xp * (zi / zp);
697                 double yp2 = yp * (zi / zp);
698                 double zp2 = zi;
699
700                 //座標軸を平行移動
701                 projected_ver_buf[j][0] = (MAX / 2) + xp2;
702                 projected_ver_buf[j][1] = (MAX / 2) + yp2;
703             }
704
705             double a[2], b[2], c[2];
706             a[0] = projected_ver_buf[0][0];
707             a[1] = projected_ver_buf[0][1];
708             b[0] = projected_ver_buf[1][0];
709             b[1] = projected_ver_buf[1][1];
710             c[0] = projected_ver_buf[2][0];
711             c[1] = projected_ver_buf[2][1];
712
713             double A[3], B[3], C[3];
714             A[0] = poly.vtx[(poly.idx[i*3+0])*3 + 0];
715             A[1] = poly.vtx[(poly.idx[i*3+0])*3 + 1];
716             A[2] = poly.vtx[(poly.idx[i*3+0])*3 + 2];
717
718             B[0] = poly.vtx[(poly.idx[i*3+1])*3 + 0];
719             B[1] = poly.vtx[(poly.idx[i*3+1])*3 + 1];
720             B[2] = poly.vtx[(poly.idx[i*3+1])*3 + 2];
721
722             C[0] = poly.vtx[(poly.idx[i*3+2])*3 + 0];
723             C[1] = poly.vtx[(poly.idx[i*3+2])*3 + 1];
724             C[2] = poly.vtx[(poly.idx[i*3+2])*3 + 2];
725
726             //ベクトルAB, ACから外積を計算して
727             //法線ベクトルnを求める
728             double AB[3], AC[3], n[3];
729             AB[0] = B[0] - A[0];
730             AB[1] = B[1] - A[1];
731             AB[2] = B[2] - A[2];
732
733             AC[0] = C[0] - A[0];
734             AC[1] = C[1] - A[1];
735             AC[2] = C[2] - A[2];
736
737             n[0] = (AB[1] * AC[2]) - (AB[2] * AC[1]);
738             n[1] = (AB[2] * AC[0]) - (AB[0] * AC[2]);
739             n[2] = (AB[0] * AC[1]) - (AB[1] * AC[0]);
740
741             //長さを1に調整
742             double length_n =
743                 sqrt(pow(n[0], 2.0) +
744                      pow(n[1], 2.0) +
745                      pow(n[2], 2.0));
746
747             n[0] = n[0] / length_n;
748             n[1] = n[1] / length_n;
749             n[2] = n[2] / length_n;
750
751             //平面iの投影先の三角形をシェーディング
752             shading(a, b, c, n, A);
753         }
754
755
756
757
758         //ヘッダー出力
759         fputs(MAGICNUM, fp_ppm);
760         fputs("\n", fp_ppm);
761         fputs(WIDTH_STRING, fp_ppm);
762         fputs(" ", fp_ppm);
763         fputs(HEIGHT_STRING, fp_ppm);
764         fputs("\n", fp_ppm);
765         fputs(MAX_STRING, fp_ppm);
766         fputs("\n" ,fp_ppm);
```

```
767
768         //imageの出力
769         for(int i = 0; i < 256; i++){
770             for(int j = 0; j < 256; j++){
771                 char r[256];
772                 char g[256];
773                 char b[256];
774                 char str[1024];
775
776                 sprintf(r, "%d", (int)round(image[i][j][0]));
777                 sprintf(g, "%d", (int)round(image[i][j][1]));
778                 sprintf(b, "%d", (int)round(image[i][j][2]));
779                 sprintf(str, "%s\t%s\t%s\n", r, g, b);
780                 fputs(str, fp_ppm);
781             }
782         }
783
784     }
785     fclose(fp_ppm);
786     fclose(fp);
787
788     printf("\nppmファイル␣%s␣の作成が完了しました.\n", argv[2]);
789     return 1;
790 }
```

5 実行例

kadai02.c と同一のディレクトリに次のプログラムを置き、

リスト 4 EvalKadai02.sh

```
1  #!/bin/sh
2  SRC=kadai02.c
3
4  WRL1=sample/av1.wrl
5  PPM1=Kadai02ForAv1.ppm
6
7  WRL2=sample/av2.wrl
8  PPM2=Kadai02ForAv2.ppm
9
10 WRL3=sample/av3.wrl
11 PPM3=Kadai02ForAv3.ppm
12
13 WRL4=sample/av4.wrl
14 PPM4=Kadai02ForAv4.ppm
15
16 WRLhead=sample/head.wrl
17 PPMhead=Kadai02ForHead.ppm
18
19 WRL1997=sample/iiyama1997.wrl
20 PPM1997=Kadai02ForIiyama1997.ppm
21
22
23 echo start!!
24 gcc -Wall $SRC
25 ./a.out $WRL1 $PPM1
26 open $PPM1
27
28 ./a.out $WRL2 $PPM2
29 open $PPM2
30
31 ./a.out $WRL3 $PPM3
32 open $PPM3
33
34 ./a.out $WRL4 $PPM4
35 open $PPM4
36
37 ./a.out $WRLhead $PPMhead
38 open $PPMhead
39
40 ./a.out $WRL1997 $PPM1997
41 open $PPM1997
42
43 echo completed!! "\xF0\x9f\x8d\xbb"
```

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

```
$ sh EvalKadai02.sh
```

を実行した. なお、それぞれ**??**が av1.wrl、**??**が av2.wrl、**??**が av3.wrl、**??**が av4.wrl、**??**が head.wrl、**??**が iiyama1997.wrl を出力した結果である.

6 問題点

問題点としては、初期段階でモジュール化をうまく考えなかったため、グローバル変数を多用する事になってしまった点、場合分けを多用しすぎてしまい、自分でもこれで必要十分なのかが把握できなくなってしまった点、などが挙げられる.

7 工夫点

工夫した点としては、コメントを随所に入れて見やすいコードを心がけた点、マクロを多用して後でプログラムに変更を加えやすいようにした点、ポリゴンデータなどの入力データは、勝手に書き換わらないよう const で宣言した点、後々の課題で使えそうな処理をモジュール化して書いた点、などが挙げられる.

8 APPENDIX

ベースとした kadai01.c のプログラムを付加しておく.

リスト 5 kadai01.c

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



//========================================================================
//必要なデータ
#define FILENAME "image.ppm"
#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

//パターン1=======================
/* #define VER_NUM 5 */
/* #define SUR_NUM 4 */
/* const double ver[VER_NUM][3] = { */
/*      {0, 0, 400}, */
/*      {-200, 0, 500}, */
/*      {0, 150, 500}, */
/*      {200, 0, 500}, */
/*      {0, -150, 500} */
/* }; */
/* const int sur[SUR_NUM][3] = { */
/*      {0, 1, 2}, */
/*      {0, 2, 3}, */
/*      {0, 3, 4}, */
```

```
/*     {0, 4, 1} */
/* }; */
//================================


//パターン2=======================
/* #define VER_NUM 6 */
/* #define SUR_NUM 2 */
/* const double ver[VER_NUM][3] = { */
/*     {-200, 0, 500}, */
/*     {200, -100, 500}, */
/*     {100, -200, 400}, */
/*     {-100, -100, 500}, */
/*     {50, 200,  400}, */
/*     {100, 100, 500} */
/* }; */
/* const int sur[SUR_NUM][3] = { */
/*     {0, 1, 2}, */
/*     {3, 4, 5}, */
/* }; */
//================================



//パターン3（ランダム座標）=======
#define VER_NUM 5
#define SUR_NUM 4

//ランダムな座標を格納するための領域を確保
//頂点座標はmain関数内で格納
double ver[VER_NUM][3];

const int sur[SUR_NUM][3] = {
    {0, 1, 2},
    {0, 2, 3},
    {0, 3, 4},
    {0, 4, 1}
};
//================================


//diffuseColor
const double diffuse_color[3] = {0.0, 1.0, 0.0};

//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};
//========================================================================


//メモリ内に画像の描画領域を確保
double image[HEIGHT][WIDTH][3];

//投影された後の2次元平面上の各点の座標を格納する領域
double projected_ver[VER_NUM][2];



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

```
113 }
114
115 int lineOrNot(double *a, double *b, double *c){
116     if(a[0] == b[0]){
117         if(a[0] == c[0]){
118             return 1;
119         }
120         else{
121             return 0;
122         }
123     }
124     else{
125         if(c[1] == a[1] + ((b[1] - a[1]) / (b[0] - a[0])) * (c[0] - a[0])){
126             return 1;
127         }
128         else{
129             return 0;
130         }
131     }
132 }
133
134 void perspective_pro(){
135     for(int i = 0; i < VER_NUM; i++){
136         double xp = ver[i][0];
137         double yp = ver[i][1];
138         double zp = ver[i][2];
139         double zi = FOCUS;
140
141         double xp2 = xp * (zi / zp);
142         double yp2 = yp * (zi / zp);
143         double zp2 = zi;
144
145         //座標軸を平行移動
146         //projected_ver[i][0] = xp2;
147         //projected_ver[i][1] = yp2;
148         projected_ver[i][0] = (MAX / 2) + xp2;
149         projected_ver[i][1] = (MAX / 2) + yp2;
150     }
151 }
152
153
154 void shading(double *a, double *b, double *c, double *n){
155     //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
156     if(lineOrNot(a, b, c) == 1){
157         //塗りつぶす点が無いので何もしない.
158     }
159     else{
160         //y座標の値が真ん中点をp、その他の点をq、rとする
161         //y座標の大きさはr <= p <= qの順
162         double p[2], q[2], r[2];
163         if(b[1] <= a[1] && a[1] <= c[1]){
164             memcpy(p, a, sizeof(double) * 2);
165             memcpy(q, c, sizeof(double) * 2);
166             memcpy(r, b, sizeof(double) * 2);
167         }
168         else{
169             if(c[1] <= a[1] && a[1] <= b[1]){
170                 memcpy(p, a, sizeof(double) * 2);
171                 memcpy(q, b, sizeof(double) * 2);
172                 memcpy(r, c, sizeof(double) * 2);
173             }
174             else{
175                 if(a[1] <= b[1] && b[1] <= c[1]){
176                     memcpy(p, b, sizeof(double) * 2);
177                     memcpy(q, c, sizeof(double) * 2);
178                     memcpy(r, a, sizeof(double) * 2);
179                 }
180                 else{
181                     if(c[1] <= b[1] && b[1] <= a[1]){
182                         memcpy(p, b, sizeof(double) * 2);
183                         memcpy(q, a, sizeof(double) * 2);
184                         memcpy(r, c, sizeof(double) * 2);
185                     }
186                     else{
187                         if(b[1] <= c[1] && c[1] <= a[1]){
188                             memcpy(p, c, sizeof(double) * 2);
189                             memcpy(q, a, sizeof(double) * 2);
190                             memcpy(r, b, sizeof(double) * 2);
191                         }
```

```
                    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\n");
                            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]){
                //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);
                }
                //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

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

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

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

                            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] =
```

```
271                                 -1 * ip * diffuse_color[2] *
272                             light_rgb[2] * MAX;
273                         }
274                     }
275                     //はみ出ている場合は描画しない
276                     else{}
277                 }
278             }
279
280             if(p[1] == q[1]){
281                 //x座標が p < q となるように調整
282                 if(q[0] <  p[0]){
283                     double temp[2];
284                     memcpy(temp, q, sizeof(double) * 2);
285                     memcpy(q, p, sizeof(double) * 2);
286                     memcpy(p, temp, sizeof(double) * 2);
287                 }
288
289                 //シェーディング処理
290                 //三角形pqrをシェーディング
291                 //y座標はp <= q
292
293                 int i;
294                 i = ceil(r[1]);
295
296                 for(i;
297                     r[1] <= i && i <= p[1];
298                     i++){
299                     //撮像平面からはみ出ていないかのチェック
300                     if(0 <= i && i <= (HEIGHT - 1)){
301                         double x1 = func1(p, r, i);
302                         double x2 = func1(q, r, i);
303
304                         int j;
305                         j = ceil(x1);
306
307                         for(j;
308                             x1 <= j && j <= x2 && 0 <= j && j <= (WIDTH - 1);
309                             j++){
310
311                             image[i][j][0] =
312                                 -1 * ip * diffuse_color[0] *
313                                 light_rgb[0] * MAX;
314                             image[i][j][1] =
315                                 -1 * ip * diffuse_color[1] *
316                                 light_rgb[1] * MAX;
317                             image[i][j][2] =
318                                 -1 * ip * diffuse_color[2] *
319                                 light_rgb[2] * MAX;
320                         }
321                     }
322                     //はみ出ている場合は描画しない
323                     else{}
324                 }
325             }
326
327         }
328         //分割できる
329         //分割してそれぞれ再帰的に処理
330         //分割後の三角形はpp2qとpp2r
331         else{
332             double p2[2];
333
334             p2[0] = func1(q, r, p[1]);
335             p2[1] = p[1];
336             //p2のほうがpのx座標より大きくなるようにする
337             if(p2[0] < p[0]){
338                 double temp[2];
339                 memcpy(temp, p2, sizeof(double) * 2);
340                 memcpy(p2, p, sizeof(double) * 2);
341                 memcpy(p, temp, sizeof(double) * 2);
342             }
343             //分割しても法線ベクトルは同一
344             shading(p, p2, q, n);
345             shading(p, p2, r, n);
346         }
347     }
348 }
349
```

```
350 int main(void){
351     FILE *fp;
352     char *fname = FILENAME;
353 
354 
355     fp = fopen( fname, "w" );
356     //ファイルが開けなかったとき
357     if( fp == NULL ){
358         printf("%sファイルが開けません.\n", fname);
359         return -1;
360     }
361 
362     //ファイルが開けたとき
363     else{
364         //頂点座標をランダムに設定==================================
365         srand(10);
366 
367         ver[0][0] = 0 + (rand()%30) - (rand()%30);
368         ver[0][1] = 0 + (rand()%50) - (rand()%50);
369         ver[0][2] = 400 + (rand()%50) - (rand()%50);
370 
371         ver[1][0] = -200 + (rand()%50) - (rand()%50);
372         ver[1][1] = 0 + (rand()%50) - (rand()%50);
373         ver[1][2] = 500 + (rand()%50) - (rand()%50);
374 
375         ver[2][0] = 0 + (rand()%50) - (rand()%50);
376         ver[2][1] = 150 + (rand()%50) - (rand()%50);
377         ver[2][2] = 500 + (rand()%50) - (rand()%50);
378 
379         ver[3][0] = 200 + (rand()%50) - (rand()%50);
380         ver[3][1] = 0 + (rand()%50) - (rand()%50);
381         ver[3][2] = 500 + (rand()%50) - (rand()%50);
382 
383         ver[4][0] = 0 + (rand()%50) - (rand()%50);
384         ver[4][1] = -150 + (rand()%50) - (rand()%50);
385         ver[4][2] = 500 + (rand()%50) - (rand()%50);
386 
387         //=========================================================
388 
389         //描画領域を初期化=======================================
390         for(int i = 0; i < 256; i++){
391             for(int j = 0; j < 256; j++){
392                 image[i][j][0] = 0.0 * MAX;
393                 image[i][j][1] = 0.0 * MAX;
394                 image[i][j][2] = 0.0 * MAX;
395             }
396         }
397         //=========================================================
398 
399         //ヘッダー出力
400         fputs(MAGICNUM, fp);
401         fputs("\n", fp);
402         fputs(WIDTH_STRING, fp);
403         fputs("␣", fp);
404         fputs(HEIGHT_STRING, fp);
405         fputs("\n", fp);
406         fputs(MAX_STRING, fp);
407         fputs("\n" ,fp);
408 
409         //各点の透視投影処理
410         perspective_pro();
411 
412         //シェーディング
413         for(int i = 0; i < SUR_NUM; i++){
414             double a[2], b[2], c[2];
415 
416             a[0] = projected_ver[(sur[i][0])][0];
417             a[1] = projected_ver[(sur[i][0])][1];
418             b[0] = projected_ver[(sur[i][1])][0];
419             b[1] = projected_ver[(sur[i][1])][1];
420             c[0] = projected_ver[(sur[i][2])][0];
421             c[1] = projected_ver[(sur[i][2])][1];
422 
423             //法線ベクトルを計算
424             //投影前の3点の座標を取得
425             double A[3], B[3], C[3];
426             A[0] = ver[(sur[i][0])][0];
427             A[1] = ver[(sur[i][0])][1];
428             A[2] = ver[(sur[i][0])][2];
```

```
429
430             B[0] = ver[(sur[i][1])][0];
431             B[1] = ver[(sur[i][1])][1];
432             B[2] = ver[(sur[i][1])][2];
433
434             C[0] = ver[(sur[i][2])][0];
435             C[1] = ver[(sur[i][2])][1];
436             C[2] = ver[(sur[i][2])][2];
437
438             //ベクトルAB, ACから外積を計算して
439             //法線ベクトルnを求める
440             double AB[3], AC[3], n[3];
441             AB[0] = B[0] - A[0];
442             AB[1] = B[1] - A[1];
443             AB[2] = B[2] - A[2];
444
445             AC[0] = C[0] - A[0];
446             AC[1] = C[1] - A[1];
447             AC[2] = C[2] - A[2];
448
449             n[0] = (AB[1] * AC[2]) - (AB[2] * AC[1]);
450             n[1] = (AB[2] * AC[0]) - (AB[0] * AC[2]);
451             n[2] = (AB[0] * AC[1]) - (AB[1] * AC[0]);
452
453             //長さを1に調整
454             double length_n =
455                 sqrt(pow(n[0], 2.0) +
456                      pow(n[1], 2.0) +
457                      pow(n[2], 2.0));
458
459             n[0] = n[0] / length_n;
460             n[1] = n[1] / length_n;
461             n[2] = n[2] / length_n;
462
463             //平面iの投影先の三角形をシェーディング
464             shading(a, b, c, n);
465         }
466
467         //imageの出力
468         for(int i = 0; i < 256; i++){
469             for(int j = 0; j < 256; j++){
470                 char r[256];
471                 char g[256];
472                 char b[256];
473                 char str[1024];
474
475                 sprintf(r, "%d", (int)round(image[i][j][0]));
476                 sprintf(g, "%d", (int)round(image[i][j][1]));
477                 sprintf(b, "%d", (int)round(image[i][j][2]));
478                 sprintf(str, "%s\t%s\t%s\n", r, g, b);
479                 fputs(str, fp);
480             }
481         }
482     }
483     fclose(fp);
484
485     printf("\nppmファイル␣%s␣の作成が完了しました.\n", fname );
486     return 0;
487 }
```

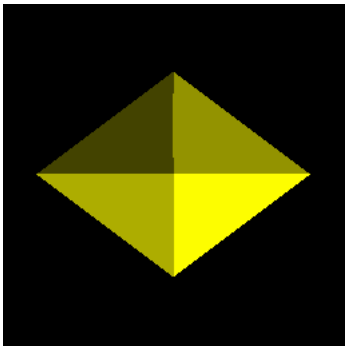

図 1　av1.wrl の出力結果

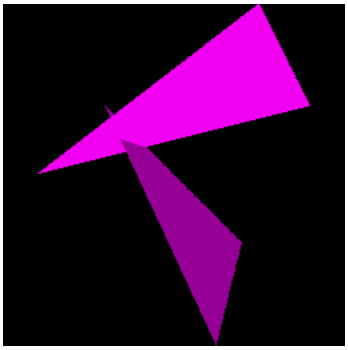

図 2　av2.wrl の出力結果

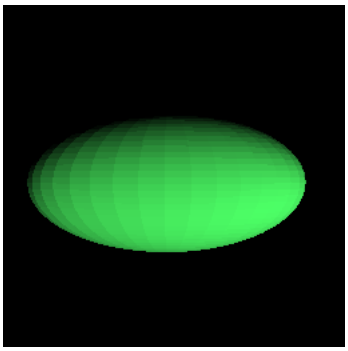

図 3　av3.wrl の出力結果

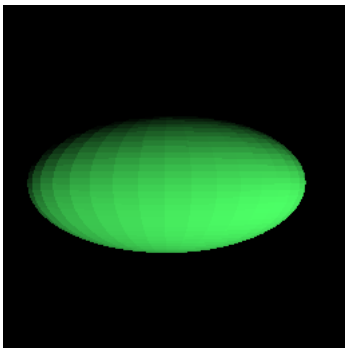

図 4　av4.wrl の出力結果

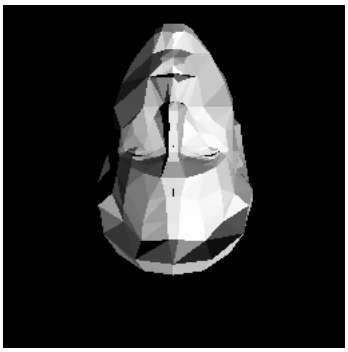

図 5　head.wrl の出力結果

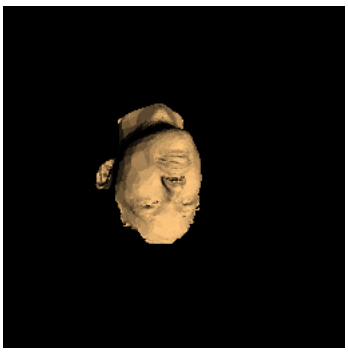

図 6　iiyama1997.wrl の出力結果