

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 6 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

```

1  /* VRML 2.0 Reader
2  *
3  * ver1.1 2005/10/06 Masaaki IIYAMA (bug fix)
4  * ver1.0 2005/09/27 Masaaki IIYAMA
5  *
6  */
7
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <ctype.h>
11 #include "vrml.h"
12
13
14 /*
15 ///////////////////////////////////////////////////
16 */
17 #define MWS 256
18
19 static int strindex( char *s, char *t)
20 {
21     int          i, j, k;
22
23     for (i = 0; s[i] != '\0'; i++) {
24         for (j = i, k = 0; t[k] != '\0' && s[j] == t[k]; j++, k++) ;
25         if (k > 0 && t[k] == '\0')
26             return i;
27     }
28     return -1;
29 }
30
31 static int getword(
32     FILE *fp,
33     char word[],
34     int sl)
35 {
36     int i,c;
37
38     while ( (c = fgetc(fp)) != EOF && ( isspace(c) || c == '#' ) ) {
39         if ( c == '#' ) {
40             while ( (c = fgetc(fp)) != EOF && c != '\n' ) ;
41             if ( c == EOF ) return (0);
42         }

```

```

43     }
44     if ( c == EOF )
45         return (0);
46     ungetc(c,fp);
47
48     for ( i = 0 ; i < sl - 1 ; i++) {
49         word[i] = fgetc(fp);
50         if ( isspace(word[i]) )
51             break;
52     }
53     word[i] = '\0';
54
55     return i;
56 }
57
58 static int read_material(
59     FILE *fp,
60     Surface *surface,
61     char *b)
62 {
63     while (getword(fp,b,MWS)>0) {
64         if (strindex(b,"}")>=0) break;
65         else if (strindex(b,"diffuseColor") >= 0) {
66             getword(fp,b,MWS);
67             surface->diff[0] = atof(b);
68             getword(fp,b,MWS);
69             surface->diff[1] = atof(b);
70             getword(fp,b,MWS);
71             surface->diff[2] = atof(b);
72         }
73         else if (strindex(b,"ambientIntensity") >= 0) {
74             getword(fp,b,MWS);
75             surface->ambi = atof(b);
76         }
77         else if (strindex(b,"specularColor") >= 0) {
78             getword(fp,b,MWS);
79             surface->spec[0] = atof(b);
80             getword(fp,b,MWS);
81             surface->spec[1] = atof(b);
82             getword(fp,b,MWS);
83             surface->spec[2] = atof(b);
84         }
85         else if (strindex(b,"shininess") >= 0) {
86             getword(fp,b,MWS);
87             surface->shine = atof(b);
88         }
89     }
90     return 1;
91 }
92
93 static int count_point(
94     FILE *fp,
95     char *b)
96 {
97     int num=0;
98     while (getword(fp,b,MWS)>0) {
99         if (strindex(b,"[">=0) break;
100     }
101     while (getword(fp,b,MWS)>0) {
102         if (strindex(b,"]">=0) break;
103         else {
104             num++;
105         }
106     }
107     if ( num %3 != 0 ) {
108         fprintf(stderr,"invalid file type[number of points mismatch]\n");
109     }
110     return num/3;
111 }
112
113 static int read_point(
114     FILE *fp,
115     Polygon *polygon,
116     char *b)
117 {
118     int num=0;
119     while (getword(fp,b,MWS)>0) {
120         if (strindex(b,"[">=0) break;
121     }

```

```

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) {

```

```

201         fprintf(stderr,"Counting...[point]\n");
202         poly->vtx_num = count_point(fp, b);
203         flag_point = 1;
204     }
205     else if (strindex(b,"coordIndex")>=0) {
206         fprintf(stderr,"Counting...[coordIndex]\n");
207         poly->idx_num = count_index(fp, b);
208         flag_index = 1;
209     }
210     else if (getword(fp,b,MWS) <= 0) return 0;
211 }
212
213 flag_material = 0;
214 flag_point = 0;
215 flag_index = 0;
216
217 fseek(fp, 0, SEEK_SET);
218 poly->vtx = (double *)malloc(sizeof(double)*3*poly->vtx_num);
219 poly->idx = (int *)malloc(sizeof(int)*3*poly->idx_num);
220 while (flag_material==0 || flag_point==0 || flag_index==0) {
221     if (strindex(b,"Material")>=0) {
222         fprintf(stderr,"Reading...[Material]\n");
223         read_material(fp,surface,b);
224         flag_material = 1;
225     }
226     else if (strindex(b,"point")>=0) {
227         fprintf(stderr,"Reading...[point]\n");
228         read_point(fp,poly,b);
229         flag_point = 1;
230     }
231     else if (strindex(b,"coordIndex")>=0) {
232         fprintf(stderr,"Reading...[coordIndex]\n");
233         read_index(fp,poly,b);
234         flag_index = 1;
235     }
236     else if (getword(fp,b,MWS) <= 0) return 0;
237 }
238
239 return 1;
240 }
241
242 //ifdef DEBUG_SAMPLE
243 int main (int argc, char *argv[])
244 {
245     int i;
246     FILE *fp;
247     Polygon poly;
248     Surface surface;
249
250     fp = fopen(argv[1], "r");
251     read_one_obj(fp, &poly, &surface);
252
253     fprintf(stderr,"%d vertices found.\n",poly.vtx_num);
254     fprintf(stderr,"%d triangles found.\n",poly.idx_num);
255
256     /* i th vertex */
257     for ( i = 0 ; i < poly.vtx_num ; i++ ) {
258         fprintf(stdout,"%f%f%f#%dth vertex\n",
259             poly.vtx[i*3+0], poly.vtx[i*3+1], poly.vtx[i*3+2],
260             i);
261     }
262
263     /* i th triangle */
264     for ( i = 0 ; i < poly.idx_num ; i++ ) {
265         fprintf(stdout,"%d%d%d#%dth triangle\n",
266             poly.idx[i*3+0], poly.idx[i*3+1], poly.idx[i*3+2],
267             i);
268     }
269
270     /* material info */
271     fprintf(stderr, "diffuseColor%f%f%f\n", surface.diff[0], surface.diff[1], surface.diff[2]);
272     fprintf(stderr, "specularColor%f%f%f\n", surface.spec[0], surface.spec[1], surface.spec[2]);
273     fprintf(stderr, "ambientIntensity%f\n", surface.ambi);
274     fprintf(stderr, "shininess%f\n", surface.shine);
275     return 1;
276 }
277 //endif

```

```

1  /* VRML 2.0 Reader
2  *
3  * ver1.0 2005/09/27 Masaaki IIYAMA
4  *
5  */
6
7  typedef unsigned char uchar;
8
9  typedef struct {
10     int w; /* image width */
11     int h; /* image height */
12     uchar *pix; /* pointer to the PPM image buffer */
13     double *z; /* pointer to the z buffer */
14 } Image;
15
16 typedef struct {
17     uchar i[3]; /* Light Intensity in RGB */
18     double d[3]; /* Light Direction */
19     double p[3]; /* Light Source Location */
20     int light_type; /* 0:parallel, 1:point */
21     int color_type; /* 0:diffuse, 1:specular, 2:ambient */
22 } Light;
23
24 typedef struct {
25     double p[3]; /* Camera Position */
26     double d[3]; /* Camera Direction */
27     double f; /* Focus */
28 } Camera;
29
30 typedef struct {
31     double diff[3]; /* Diffuse in RGB */
32     double spec[3]; /* Specular in RGB */
33     double ambi; /* Ambient */
34     double shine; /* Shininess */
35 } Surface;
36
37 typedef struct {
38     double *vtx; /* Vertex List */
39     int *idx; /* Index List */
40     int vtx_num; /* number of vertice */
41     int idx_num; /* number of indices */
42 } Polygon;
43
44 int read_one_obj(
45     FILE *fp,
46     Polygon *poly,
47     Surface *surface);

```

3.2 各種定数

プログラム内部で使っ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 ファイ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. `double` 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 `projected_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 double 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、

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

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 $x_p - \frac{\text{WIDTH}}{2}$ 、 $y_p - \frac{\text{HEIGHT}}{2}$ 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（外積から求める）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

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <math.h>
5 #include <float.h>
6 #include <ctype.h>
7 #include "vrml.h"
8
9
10
11 //=====
12 //必要なデータ
13 #define FILENAME "image.ppm"
14 #define MAGICNUM "P3"
15 #define WIDTH 256
16 #define WIDTH_STRING "256"
17 #define HEIGHT 256
18 #define HEIGHT_STRING "256"
19 #define MAX 255
20 #define MAX_STRING "255"
21 #define FOCUS 256.0
22 #define Z_BUF_MAX
23
24 //diffuseColorを格納する配列
25 double diffuse_color[3];
26
27 //光源モデルは平行光源
28 //光源方向
29 const double light_dir[3] = {-1.0, -1.0, 2.0};
30 //光源明るさ
31 const double light_rgb[3] = {1.0, 1.0, 1.0};
32
33 //カメラ位置は原点であるものとして投影を行う。
34 //=====
35 //メモリ内に画像の描画領域を確保
36 double image[HEIGHT][WIDTH][3];
37 //zバッファ用の領域を確保
38 double z_buf[HEIGHT][WIDTH];
39 //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
40 double projected_ver_buf[3][2];
41
42
43 //2点 p、q を結ぶ直線上の y 座標が y であるような点の x 座標を返す関数
44 //eg)
45 //double p[2] = (1.0, 2.0);
46 double func1(double *p, double *q, double y){
47     double x;
48     if(p[1] > q[1]){
49         x = ((p[0] * (y - q[1])) + (q[0] * (p[1] - y))) / (p[1] - q[1]);
50     }
51     if(p[1] < q[1]){
52         x = ((q[0] * (y - p[1])) + (p[0] * (q[1] - y))) / (q[1] - p[1]);
53     }
54     if(p[1] == q[1]){
55         //解なし
```

```

56         printf("\n引数が不正です.\n2点\n(%f,%f)\n(%f,%f)\nはy座標が同じです.\n"
57                , p[0], p[1], q[0], q[1]);
58         perror(NULL);
59         return -1;
60     }
61     return x;
62 }
63
64 //3点 a[2] = {x, y},,が1直線上にあるかどうかを判定する関数
65 //1直線上に無ければ return 0;
66 //1直線上にあれば return 1;
67 int lineOrNot(double *a, double *b, double *c){
68     if(a[0] == b[0]){
69         if(a[0] == c[0]){
70             return 1;
71         }
72         else{
73             return 0;
74         }
75     }
76     else{
77         if(c[1] == a[1] + ((b[1] - a[1]) / (b[0] - a[0])) * (c[0] - a[0])){
78             return 1;
79         }
80         else{
81             return 0;
82         }
83     }
84 }
85
86 //投影された三角形 abc にラスタライズ、クリッピングでシェーディングを行う関数
87 //引数 a, b, c は投影平面上の3点
88 //eg)
89 //double a = {1.0, 2.0};
90 //n は法線ベクトル
91 //A は投影前の3点からなる三角形平面上の任意の点の座標.
92 //(3点 A, B, C のうちいずれでも良いが main 関数内の A を使うものとする.)
93 void shading(double *a, double *b, double *c, double *n, double *A){
94     //3点 が1直線上に並んでいるときはシェーディングができない
95     if(lineOrNot(a, b, c) == 1){
96         //塗りつぶす点が無いので何もしない.
97     }
98     else{
99         //y座標の値が真ん中点を p、その他の点を q、r とする
100        //y座標の大きさは r <= p <= q の順
101        double p[2], q[2], r[2];
102        if(b[1] <= a[1] && a[1] <= c[1]){
103            memcpy(p, a, sizeof(double) * 2);
104            memcpy(q, c, sizeof(double) * 2);
105            memcpy(r, b, sizeof(double) * 2);
106        }
107        else{
108            if(c[1] <= a[1] && a[1] <= b[1]){
109                memcpy(p, a, sizeof(double) * 2);
110                memcpy(q, b, sizeof(double) * 2);
111                memcpy(r, c, sizeof(double) * 2);
112            }
113            else{
114                if(a[1] <= b[1] && b[1] <= c[1]){
115                    memcpy(p, b, sizeof(double) * 2);
116                    memcpy(q, c, sizeof(double) * 2);
117                    memcpy(r, a, sizeof(double) * 2);
118                }
119                else{
120                    if(c[1] <= b[1] && b[1] <= a[1]){
121                        memcpy(p, b, sizeof(double) * 2);
122                        memcpy(q, a, sizeof(double) * 2);
123                        memcpy(r, c, sizeof(double) * 2);
124                    }
125                    else{
126                        if(b[1] <= c[1] && c[1] <= a[1]){
127                            memcpy(p, c, sizeof(double) * 2);
128                            memcpy(q, a, sizeof(double) * 2);
129                            memcpy(r, b, sizeof(double) * 2);
130                        }
131                        else{
132                            if(a[1] <= c[1] && c[1] <= b[1]){
133                                memcpy(p, c, sizeof(double) * 2);
134                                memcpy(q, b, sizeof(double) * 2);

```

```

135         memcpy(r, a, sizeof(double) * 2);
136     }
137     else{
138         printf("エラーat2055\n");
139         printf("\na[1]=%f\tb[1]=%f\tc[1]=%f\n", a[1], b[1], c[1]);
140         perror(NULL);
141     }
142 }
143 }
144 }
145 }
146 }
147
148 //分割可能な三角形かを判定
149 if(p[1] == r[1] || p[1] == q[1]){
150     //分割できない
151
152     //長さが1の光源方向ベクトルを作成する
153     //光源方向ベクトルの長さ
154     double length_1 =
155         sqrt(pow(light_dir[0], 2.0) +
156             pow(light_dir[1], 2.0) +
157             pow(light_dir[2], 2.0));
158
159     double light_dir_vec[3];
160     light_dir_vec[0] = light_dir[0] / length_1;
161     light_dir_vec[1] = light_dir[1] / length_1;
162     light_dir_vec[2] = light_dir[2] / length_1;
163
164     // 法線ベクトル n と光源方向ベクトルの内積
165     double ip =
166         (n[0] * light_dir_vec[0]) +
167         (n[1] * light_dir_vec[1]) +
168         (n[2] * light_dir_vec[2]);
169
170     if(0 <= ip){
171         ip = 0;
172     }
173
174     //2パターンの三角形を特定
175     if(p[1] == r[1]){
176         //debug
177         //printf("\np[1] == r[1]\n");
178         //x座標が p <= r となるように調整
179         if(r[0] < p[0]){
180             double temp[2];
181             memcpy(temp, r, sizeof(double) * 2);
182             memcpy(r, p, sizeof(double) * 2);
183             memcpy(p, temp, sizeof(double) * 2);
184         }
185
186         //debug
187         if(r[0] == p[0]){
188             perror("エラーat958");
189         }
190
191         //シェーディング処理
192         //三角形 pqr をシェーディング
193         //y座標は p <= r
194         //debug
195         if(r[1] < p[1]){
196             perror("エラーat1855");
197         }
198
199         int i;
200         i = ceil(p[1]);
201         for(i;
202             p[1] <= i && i <= q[1];
203             i++){
204
205             //撮像平面からはみ出していないかのチェック
206             if(0 <= i
207                 &&
208                 i <= (HEIGHT - 1)){
209                 double x1 = func1(p, q, i);
210                 double x2 = func1(r, q, i);
211                 int j;
212                 j = ceil(x1);
213

```

```

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

```

```

293         /
294         ((n[0]*(j-(WIDTH/2))) + (n[1]*(i-(HEIGHT/2))) + n[2]*FOCUS);
295
296     // 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)
297     if(z_buf[i][j] < z){}
298
299     else{
300         image[i][j][0] =
301             -1 * ip * diffuse_color[0] *
302             light_rgb[0] * MAX;
303         image[i][j][1] =
304             -1 * ip * diffuse_color[1] *
305             light_rgb[1] * MAX;
306         image[i][j][2] =
307             -1 * ip * diffuse_color[2] *
308             light_rgb[2] * MAX;
309
310         // zバッファの更新
311         z_buf[i][j] = z;
312     }
313 }
314 }
315 //撮像平面からはみ出る部分は描画しない
316 else{}
317 }
318 }
319
320 }
321 //分割できる
322 //分割してそれぞれ再帰的に処理
323 //分割後の三角形はpp2qとpp2r
324 else{
325     double p2[2];
326     p2[0] = func1(q, r, p[1]);
327     p2[1] = p[1];
328     //p2のほうがpのx座標より大きくなるようにする
329     if(p2[0] < p[0]){
330         double temp[2];
331         memcpy(temp, p2, sizeof(double) * 2);
332         memcpy(p2, p, sizeof(double) * 2);
333         memcpy(p, temp, sizeof(double) * 2);
334     }
335     //分割して処理
336     //分割しても同一平面上なので法線ベクトルと
337     //平面上の任意の点は同じものを使える。
338     shading(p, p2, q, n, A);
339     shading(p, p2, r, n, A);
340 }
341 }
342 }
343
344 /* VRMLの読み込み */
345 /* ===== */
346 #define MWS 256
347
348 static int strindex( char *s, char *t)
349 {
350     int i, j, k;
351
352     for (i = 0; s[i] != '\0'; i++) {
353         for (j = i, k = 0; t[k] != '\0' && s[j] == t[k]; j++, k++) ;
354         if (k > 0 && t[k] == '\0')
355             return i;
356     }
357     return -1;
358 }
359
360 static int getword(
361     FILE *fp,
362     char word[],
363     int sl)
364 {
365     int i,c;
366
367     while ( (c = fgetc(fp)) != EOF && ( isspace(c) || c == '#' ) ) {
368         if ( c == '#' ) {
369             while ( (c = fgetc(fp)) != EOF && c != '\n' ) ;
370             if ( c == EOF ) return (0);
371         }

```

```

372     }
373     if ( c == EOF )
374         return (0);
375     ungetc(c,fp);
376
377     for ( i = 0 ; i < sl - 1 ; i++) {
378         word[i] = fgetc(fp);
379         if ( isspace(word[i]) )
380             break;
381     }
382     word[i] = '\0';
383
384     return i;
385 }
386
387 static int read_material(
388     FILE *fp,
389     Surface *surface,
390     char *b)
391 {
392     while (getword(fp,b,MWS)>0) {
393         if (strindex(b,"}")>=0) break;
394         else if (strindex(b,"diffuseColor") >= 0) {
395             getword(fp,b,MWS);
396             surface->diff[0] = atof(b);
397             getword(fp,b,MWS);
398             surface->diff[1] = atof(b);
399             getword(fp,b,MWS);
400             surface->diff[2] = atof(b);
401         }
402         else if (strindex(b,"ambientIntensity") >= 0) {
403             getword(fp,b,MWS);
404             surface->ambi = atof(b);
405         }
406         else if (strindex(b,"specularColor") >= 0) {
407             getword(fp,b,MWS);
408             surface->spec[0] = atof(b);
409             getword(fp,b,MWS);
410             surface->spec[1] = atof(b);
411             getword(fp,b,MWS);
412             surface->spec[2] = atof(b);
413         }
414         else if (strindex(b,"shininess") >= 0) {
415             getword(fp,b,MWS);
416             surface->shine = atof(b);
417         }
418     }
419     return 1;
420 }
421
422 static int count_point(
423     FILE *fp,
424     char *b)
425 {
426     int num=0;
427     while (getword(fp,b,MWS)>0) {
428         if (strindex(b,"[">=0) break;
429     }
430     while (getword(fp,b,MWS)>0) {
431         if (strindex(b,"]">=0) break;
432         else {
433             num++;
434         }
435     }
436     if ( num %3 != 0 ) {
437         fprintf(stderr,"invalid file type[number_of_points_mismatch]\n");
438     }
439     return num/3;
440 }
441
442 static int read_point(
443     FILE *fp,
444     Polygon *polygon,
445     char *b)
446 {
447     int num=0;
448     while (getword(fp,b,MWS)>0) {
449         if (strindex(b,"[">=0) break;
450     }

```

```

451     while (getword(fp,b,MWS)>0) {
452         if (strindex(b,""]>=0) break;
453         else {
454             polygon->vtx[num++] = atof(b);
455         }
456     }
457     return num/3;
458 }
459
460 static int count_index(
461     FILE *fp,
462     char *b)
463 {
464     int num=0;
465     while (getword(fp,b,MWS)>0) {
466         if (strindex(b,"[">=0) break;
467     }
468     while (getword(fp,b,MWS)>0) {
469         if (strindex(b,""]>=0) break;
470         else {
471             num++;
472         }
473     }
474     if ( num %4 != 0 ) {
475         fprintf(stderr,"invalid_file_type[number_of_indices_mismatch]\n");
476     }
477     return num/4;
478 }
479
480 static int read_index(
481     FILE *fp,
482     Polygon *polygon,
483     char *b)
484 {
485     int num=0;
486     while (getword(fp,b,MWS)>0) {
487         if (strindex(b,"[">=0) break;
488     }
489     while (getword(fp,b,MWS)>0) {
490         if (strindex(b,""]>=0) break;
491         else {
492             polygon->idx[num++] = atoi(b);
493             if (num%3 == 0) getword(fp,b,MWS);
494         }
495     }
496     return num/3;
497 }
498
499 int read_one_obj(
500     FILE *fp,
501     Polygon *poly,
502     Surface *surface)
503 {
504     char b[MWS];
505     int flag_material = 0;
506     int flag_point = 0;
507     int flag_index = 0;
508
509     /* initialize surface */
510     surface->diff[0] = 1.0;
511     surface->diff[1] = 1.0;
512     surface->diff[2] = 1.0;
513     surface->spec[0] = 0.0;
514     surface->spec[1] = 0.0;
515     surface->spec[2] = 0.0;
516     surface->ambi = 0.0;
517     surface->shine = 0.2;
518
519     if ( getword(fp,b,MWS) <= 0) return 0;
520
521     poly->vtx_num = 0;
522     poly->idx_num = 0;
523
524     while (flag_material==0 || flag_point==0 || flag_index==0) {
525         if (strindex(b,"Material">=0) {
526             getword(fp,b,MWS);
527             flag_material = 1;
528         }
529         else if (strindex(b,"point">=0) {

```

```

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

```



```

609
610 FILE *fp_ppm;
611 char *fname = FILENAME;
612 fp_ppm = fopen(argv[2], "w" );
613
614 //ファイルが開けなかったとき
615 if( fp_ppm == NULL ){
616     printf("%s ファイルが開けません.\n", fname);
617     return -1;
618 }
619
620 //ファイルが開けたとき
621 else{
622     fprintf(stderr, "\n初期の頂点座標は以下\n");
623     for(int i = 0; i < poly.vtx_num; i++){
624         fprintf(stderr, "%f\t%f\t%f\n", poly.vtx[i*3+0], poly.vtx[i*3+1], poly.vtx[i*3+2]);
625     }
626     fprintf(stderr, "\n");
627
628     //描画領域を初期化
629     for(int i = 0; i < 256; i++){
630         for(int j = 0; j < 256; j++){
631             image[i][j][0] = 0.0 * MAX;
632             image[i][j][1] = 0.0 * MAX;
633             image[i][j][2] = 0.0 * MAX;
634         }
635     }
636
637     //zバッファを初期化
638     for(int i = 0; i < 256; i++){
639         for(int j = 0; j < 256; j++){
640             z_buf[i][j] = DBL_MAX;
641         }
642     }
643
644     //diffuse_colorの格納
645     diffuse_color[0] = surface.diff[0];
646     diffuse_color[1] = surface.diff[1];
647     diffuse_color[2] = surface.diff[2];
648
649     //シェーディング
650     //三角形ごとのループ
651     for(int i = 0; i < poly.idx_num; i++){
652         //各点の透視投影処理
653         for(int j = 0; j < 3; j++){
654             double xp = poly.vtx[(poly.idx[i*3+j])*3 + 0];
655             double yp = poly.vtx[(poly.idx[i*3+j])*3 + 1];
656             double zp = poly.vtx[(poly.idx[i*3+j])*3 + 2];
657             double zi = FOCUS;
658
659             //debug
660             if(zp == 0){
661                 printf("\n(%f\t%f\t%f) 1/i=%d, 1/j=%d\n", xp, yp, zp, i, j);
662                 perror("\nエラー0934\n");
663                 //break;
664             }
665
666             double xp2 = xp * (zi / zp);
667             double yp2 = yp * (zi / zp);
668             double zp2 = zi;
669
670             //座標軸を平行移動
671             projected_ver_buf[j][0] = (MAX / 2) + xp2;
672             projected_ver_buf[j][1] = (MAX / 2) + yp2;
673         }
674
675         double a[2], b[2], c[2];
676         a[0] = projected_ver_buf[0][0];
677         a[1] = projected_ver_buf[0][1];
678         b[0] = projected_ver_buf[1][0];
679         b[1] = projected_ver_buf[1][1];
680         c[0] = projected_ver_buf[2][0];
681         c[1] = projected_ver_buf[2][1];
682
683         double A[3], B[3], C[3];
684         A[0] = poly.vtx[(poly.idx[i*3+0])*3 + 0];
685         A[1] = poly.vtx[(poly.idx[i*3+0])*3 + 1];
686         A[2] = poly.vtx[(poly.idx[i*3+0])*3 + 2];
687

```

```

688     B[0] = poly.vtx[(poly.idx[i*3+1])*3 + 0];
689     B[1] = poly.vtx[(poly.idx[i*3+1])*3 + 1];
690     B[2] = poly.vtx[(poly.idx[i*3+1])*3 + 2];
691
692     C[0] = poly.vtx[(poly.idx[i*3+2])*3 + 0];
693     C[1] = poly.vtx[(poly.idx[i*3+2])*3 + 1];
694     C[2] = poly.vtx[(poly.idx[i*3+2])*3 + 2];
695
696     //ベクトルAB, ACから外積を計算して
697     //法線ベクトルnを求める
698     double AB[3], AC[3], n[3];
699     AB[0] = B[0] - A[0];
700     AB[1] = B[1] - A[1];
701     AB[2] = B[2] - A[2];
702
703     AC[0] = C[0] - A[0];
704     AC[1] = C[1] - A[1];
705     AC[2] = C[2] - A[2];
706
707     n[0] = (AB[1] * AC[2]) - (AB[2] * AC[1]);
708     n[1] = (AB[2] * AC[0]) - (AB[0] * AC[2]);
709     n[2] = (AB[0] * AC[1]) - (AB[1] * AC[0]);
710
711     //長さを1に調整
712     double length_n =
713         sqrt(pow(n[0], 2.0) +
714             pow(n[1], 2.0) +
715             pow(n[2], 2.0));
716
717     n[0] = n[0] / length_n;
718     n[1] = n[1] / length_n;
719     n[2] = n[2] / length_n;
720
721     //平面iの投影先の三角形をシェーディング
722     shading(a, b, c, n, A);
723 }
724
725
726 //ヘッダー出力
727 fputs(MAGICNUM, fp_ppm);
728 fputs("\n", fp_ppm);
729 fputs(WIDTH_STRING, fp_ppm);
730 fputs("\n", fp_ppm);
731 fputs(HEIGHT_STRING, fp_ppm);
732 fputs("\n", fp_ppm);
733 fputs(MAX_STRING, fp_ppm);
734 fputs("\n", fp_ppm);
735
736 //imageの出力
737 for(int i = 0; i < 256; i++){
738     for(int j = 0; j < 256; j++){
739         char r[256];
740         char g[256];
741         char b[256];
742         char str[1024];
743
744         sprintf(r, "%d", (int)round(image[i][j][0]));
745         sprintf(g, "%d", (int)round(image[i][j][1]));
746         sprintf(b, "%d", (int)round(image[i][j][2]));
747         sprintf(str, "%s\t%s\t%s\n", r, g, b);
748         fputs(str, fp_ppm);
749     }
750 }
751
752 }
753 fclose(fp_ppm);
754 fclose(fp);
755
756 printf("\nppmファイル%sに画像を出力しました.\n", argv[2]);
757 return 1;
758 }

```

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、

```
1 $ sh EvalKadai02.sh
```

を実行した. 出力画像は最終ページに付与した.

6 工夫点、問題点、感想

前半部分でも述べたように、自分としてはできるだけ計算の処理や、使用するメモリ量を節約し、無駄なデータを長時間大量に保持する事のないようにプログラムを書いたつもりである. C 言語は Java 等と比べて、プログラマがハード面の挙動にも気を使ってプログラムを書く必要があるが、レンダリングなどの大量の計算処理を行う必要のあるプログラムでは、普段の実験で書くようなプログラムよりも一層、無駄のないプログラムを書く必要があるのだと感じた. 実際、最初の段階でうまく設計をしなかったために途中で膨大な場合分けが発生してしまい、気をつけて書いたつもりでも、emacs 上で動かすシェルでは動作が重すぎて、一つの

VRML を表示させるのに数分かってしまったこともあった. また、リアルタイムでレンダリング処理を行うゲーム機などではそういった処理を一瞬で行っているという凄さに改めて気付かされ、そういった部分にも興味湧いた.

7 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

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <math.h>
5
6
7
8 //=====
9 //必要なデータ
10 #define FILENAME "image.ppm"
11 #define MAGICNUM "P3"
12 #define WIDTH 256
13 #define WIDTH_STRING "256"
14 #define HEIGHT 256
15 #define HEIGHT_STRING "256"
16 #define MAX 255
17 #define MAX_STRING "255"
18 #define FOCUS 256.0
19
20 //パターン1=====
21 /* #define VER_NUM 5 */
22 /* #define SUR_NUM 4 */
23 /* const double ver[VER_NUM][3] = { */
24 /*     {0, 0, 400}, */
25 /*     {-200, 0, 500}, */
26 /*     {0, 150, 500}, */
27 /*     {200, 0, 500}, */
28 /*     {0, -150, 500} */
29 /* }; */
30 /* const int sur[SUR_NUM][3] = { */
31 /*     {0, 1, 2}, */
32 /*     {0, 2, 3}, */
33 /*     {0, 3, 4}, */
34 /*     {0, 4, 1} */
35 /* }; */
36 //=====
37
38 //パターン2=====
39 /* #define VER_NUM 6 */
40 /* #define SUR_NUM 2 */
41 /* const double ver[VER_NUM][3] = { */
42 /*     {-200, 0, 500}, */
43 /*     {200, -100, 500}, */
44 /*     {100, -200, 400}, */
45 /*     {-100, -100, 500}, */
46 /*     {50, 200, 400}, */
47 /*     {100, 100, 500} */
48 /* }; */
49 /* const int sur[SUR_NUM][3] = { */
50 /*     {0, 1, 2}, */
51 /*     {3, 4, 5}, */
52 /* }; */
53 //=====
54
55
56
57 //パターン3 (ランダム座標) =====
58 #define VER_NUM 5
59 #define SUR_NUM 4
60
61 //ランダムな座標を格納するための領域を確保
62 //頂点座標は main 関数内で格納
63
```

```

64 double ver[VER_NUM][3];
65
66 const int sur[SUR_NUM][3] = {
67     {0, 1, 2},
68     {0, 2, 3},
69     {0, 3, 4},
70     {0, 4, 1}
71 };
72 //=====
73
74
75 //diffuseColor
76 const double diffuse_color[3] = {0.0, 1.0, 0.0};
77
78 //光源モデルは平行光源
79 //光源方向
80 const double light_dir[3] = {-1.0, -1.0, 2.0};
81 //光源明るさ
82 const double light_rgb[3] = {1.0, 1.0, 1.0};
83 //=====
84
85
86 //メモリ内に画像の描画領域を確保
87 double image[HEIGHT][WIDTH][3];
88
89 //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
90 double projected_ver[VER_NUM][2];
91
92
93
94 //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
95 //eg)
96 //double p[2] = (1.0, 2.0);
97 double func1(double *p, double *q, double y){
98     double x;
99     if(p[1] > q[1]){
100         x = ((p[0] * (y - q[1])) + (q[0] * (p[1] - y))) / (p[1] - q[1]);
101     }
102     if(p[1] < q[1]){
103         x = ((q[0] * (y - p[1])) + (p[0] * (q[1] - y))) / (q[1] - p[1]);
104     }
105     if(p[1] == q[1]){
106         //解なし
107         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",
108             p[0], p[1], q[0], q[1]);
109         perror(NULL);
110         return -1;
111     }
112     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                         }
192                         else{
193                             if(a[1] <= c[1] && c[1] <= b[1]){
194                                 memcpy(p, c, sizeof(double) * 2);
195                                 memcpy(q, b, sizeof(double) * 2);
196                                 memcpy(r, a, sizeof(double) * 2);
197                             }
198                             else{
199                                 printf("エラー\n");
200                                 perror(NULL);
201                             }
202                         }
203                     }
204                 }
205             }
206         }
207
208         //分割可能な三角形かを判定
209         if(p[1] == r[1] || p[1] == q[1]){
210             //分割できない
211
212             //長さがlの光源方向ベクトルを作成する
213             //光源方向ベクトルの長さ
214             double length_l =
215                 sqrt(pow(light_dir[0], 2.0) +
216                     pow(light_dir[1], 2.0) +
217                     pow(light_dir[2], 2.0));
218
219             double light_dir_vec[3];
220             light_dir_vec[0] = light_dir[0] / length_l;
221             light_dir_vec[1] = light_dir[1] / length_l;

```

```

222     light_dir_vec[2] = light_dir[2] / length_1;
223
224 // 法線ベクトル  $n$  と光源方向ベクトルの内積
225 double ip =
226     (n[0] * light_dir_vec[0]) +
227     (n[1] * light_dir_vec[1]) +
228     (n[2] * light_dir_vec[2]);
229
230 if(0 <= ip){
231     ip = 0;
232 }
233
234 //2パターンの三角形を特定
235 if(p[1] == r[1]){
236     //x座標が  $p \leq r$  となるように調整
237     if(r[0] < p[0]){
238         double temp[2];
239         memcpy(temp, r, sizeof(double) * 2);
240         memcpy(r, p, sizeof(double) * 2);
241         memcpy(p, temp, sizeof(double) * 2);
242     }
243     //シェーディング処理
244     //三角形  $pqr$  をシェーディング
245     //y座標は  $p \leq r$ 
246
247     int i;
248     i = ceil(p[1]);
249     for(i;
250         p[1] <= i && i <= q[1];
251         i++){
252
253         //撮像平面からはみ出していないかのチェック
254         if(0 <= i && i <= (HEIGHT - 1)){
255             double x1 = func1(p, q, i);
256             double x2 = func1(r, q, i);
257             int j;
258             j = ceil(x1);
259
260             for(j;
261                 x1 <= j && j <= x2 && 0 <= j && j <= (WIDTH - 1);
262                 j++){
263
264                 image[i][j][0] =
265                     -1 * ip * diffuse_color[0] *
266                     light_rgb[0] * MAX;
267                 image[i][j][1] =
268                     -1 * ip * diffuse_color[1] *
269                     light_rgb[1] * MAX;
270                 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  $p \leq q$ 
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し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("\n", 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 }

```

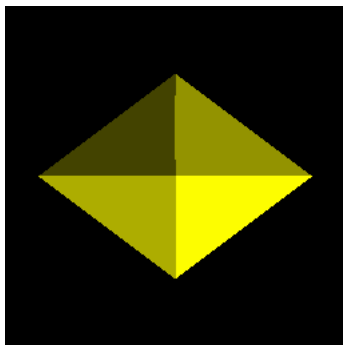


図 1 av1.wrl の出力結果

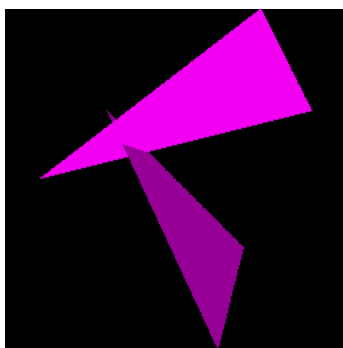


図 2 av2.wrl の出力結果

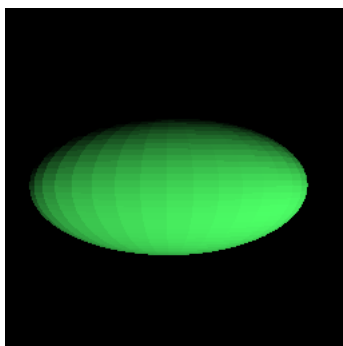


図 3 av3.wrl の出力結果

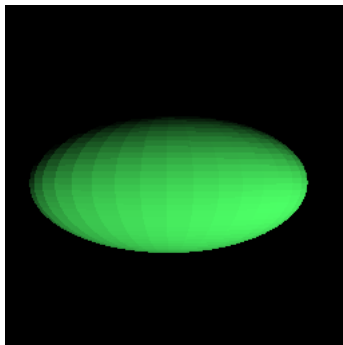


図 4 av4.wrl の出力結果

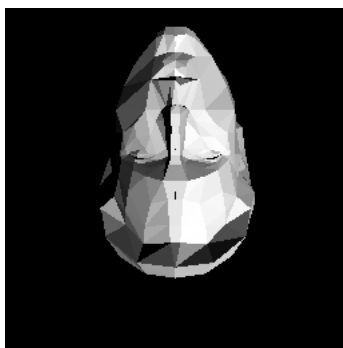


図 5 head.wrl の出力結果

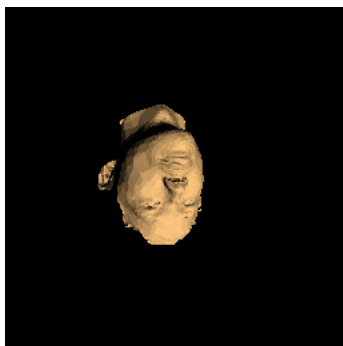


図 6 iiyama1997.wrl の出力結果