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 1

工学部情報学科 3 回生 1029255242
勝見久央

作成日: 2015 年 10 月 7 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 C 言語で作成した.

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.

- ポリゴンデータはプログラム内部で与え、頂点座標をランダムに生成する.
- PPM 画像の大きさは
$$256 \times 256$$
- カメラ位置は (x,y,z) = (0.0, 0.0, 0.0)
- カメラ方向あ h (x,y,z) = (0.0, 0.0, 1.0)
- カメラ焦点距離は 256.0
- ポリゴンには拡散反射を施す

3 プログラムの仕様

3.1 留意点

座標、RGB 値等はデータ型を double として計算し、RGB 値については出力時には一旦 round 関数を用いて丸めて int 型に変換した後 char 型に変換して出力した. 各点の座標については double 型のサイズ 3 の配列に xyz 座標を格納して処理した. プログラム内には頂点座標の例として 2 パターン分もコメントで記述している. 大文字アルファベットの定数はマクロである.

3.2 各種定数

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.

- FILENAME
  ファイル名を指定. ここでは image.ppma としている.
- 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

次の定数はポリゴンデータとして定める定数である.

- VER_NUM
  ポリゴンの頂点数
- SUR_NUM
  ポリゴンを生成する三角形平面の数
- ver[VER_NUM][3]
  VRML の Coordinate ノードの point フィールドの値. ポリゴンを形成する各点の座標を格納する. 点 i の座標は (sur[i][0], sur[i][1], sur[i][2]) である. 初期化は main 関数内で行う. double 型配列.
- sur[SUR_NUM][3]
  VRML の IndexedFaceSet ノード内の coordIndex フィールドの値. ポリゴンを形成する三角形を指定する. 三角形は点 sur[i][0]、sur[i][1]、sru[i][2] の 3 点からなる.int 型配列.
- diffuse_color[3]
  VRML の Material ノードの diffuseColor フィールドの値. 配列の先頭から順に RGB 値を表す. double 型配列.

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 を使用
- 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 大域変数

-

4 プログラム本体

プログラム本体は次のようになった．

リスト 1　キャプション

```
#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[256][256][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("2点\n(%f,␣%f)\n(%f,␣%f)\nはy座標が同じ.", p[0], p[1], q[0], q[1]);
        perror(NULL);
        return -1;
    }
    //printf("check x = %f\n", x);
    //printf("check p[0] = %f\n", p[0]);
    return x;
}

//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
//1直線上に無ければreturn 0;
//1直線上にあればreturn 1;
int lineOrNot(double *a, double *b, double *c){
    if(a[0] == b[0]){
        if(a[0] == c[0]){
            return 1;
        }
        else{
            return 0;
        }
    }
    else{
        if(c[1] == a[1] + ((b[1] - a[1]) / (b[0] - a[0])) * (c[0] - a[0])){
            return 1;
        }
        else{
            return 0;
        }
    }
}

//頂点座標の配列verに透視投影を行う関数
```

```
void perspective_pro(){
    for(int i = 0; i < VER_NUM; i++){
        double xp = ver[i][0];
        double yp = ver[i][1];
        double zp = ver[i][2];
        double zi = FOCUS;

        double xp2 = xp * (zi / zp);
        double yp2 = yp * (zi / zp);
        double zp2 = zi;

        //座標軸を平行移動
        //projected_ver[i][0] = xp2;
        //projected_ver[i][1] = yp2;
        projected_ver[i][0] = (MAX / 2) + xp2;
        projected_ver[i][1] = (MAX / 2) + yp2;
    }
}

//投影された三角形 a b c 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
//引数a, b, cは投影平面上の3点
//eg)
//double a = {1.0, 2.0};
//nは法線ベクトル
void shading(double *a, double *b, double *c, double *n){
    //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
    if(lineOrNot(a, b, c) == 1){
        //塗りつぶす点が無いので何もしない.
        //debug
        printf("\n3点\naの座標(%f,\t%f)\nbの座標(%f,\t%f)\ncの座標(%f,\t%f)\nは一直線上の3点です\n"
               ,a[0], a[1], b[0], b[1], c[0], c[1]);
    }
    else{
        //y座標の値が真ん中点をp、その他の点をq、rとする
        //y座標の大きさはr <= p <= qの順
        double p[2], q[2], r[2];
        if(b[1] <= a[1] && a[1] <= c[1]){
            memcpy(p, a, sizeof(double) * 2);
            memcpy(q, c, sizeof(double) * 2);
            memcpy(r, b, sizeof(double) * 2);
        }
        else{
            if(c[1] <= a[1] && a[1] <= b[1]){
                memcpy(p, a, sizeof(double) * 2);
                memcpy(q, b, sizeof(double) * 2);
                memcpy(r, c, sizeof(double) * 2);
            }
            else{
                if(a[1] <= b[1] && b[1] <= c[1]){
                    memcpy(p, b, sizeof(double) * 2);
                    memcpy(q, c, sizeof(double) * 2);
                    memcpy(r, a, sizeof(double) * 2);
                }
                else{
                    if(c[1] <= b[1] && b[1] <= a[1]){
                        memcpy(p, b, sizeof(double) * 2);
                        memcpy(q, a, sizeof(double) * 2);
                        memcpy(r, c, sizeof(double) * 2);
                    }
                    else{
                        if(b[1] <= c[1] && c[1] <= a[1]){
                            memcpy(p, c, sizeof(double) * 2);
                            memcpy(q, a, sizeof(double) * 2);
                            memcpy(r, b, sizeof(double) * 2);
                        }
                        else{
                            if(a[1] <= c[1] && c[1] <= b[1]){
                                memcpy(p, c, sizeof(double) * 2);
                                memcpy(q, b, sizeof(double) * 2);
                                memcpy(r, a, sizeof(double) * 2);
                            }
                            else{
                                printf("エラー\n");
                                perror(NULL);
                            }
                        }
                    }
                }
            }
```

```
218         }
219         printf("\n3点の座標は\npの座標(%f,␣%f)\nqの座標(%f,␣%f)\nrの座標(%f,␣%f)\n"
220                ,p[0], p[1], q[0], q[1], r[0], r[1]);
221         //分割可能な三角形かを判定
222         if(p[1] == r[1] || p[1] == q[1]){
223             //分割できない
224             printf("\n三角形\npの座標(%f,␣%f)\nqの座標(%f,␣%f)\nrの座標(%f,␣%f)\
                  nは分割できないのでこのままシェーディング\n"
225                    ,p[0], p[1], q[0], q[1], r[0], r[1]);
226
227             //長さが1の光源方向ベクトルを作成する
228             //光源方向ベクトルの長さ
229             double length_l =
230                 sqrt(pow(light_dir[0], 2.0) +
231                      pow(light_dir[1], 2.0) +
232                      pow(light_dir[2], 2.0));
233
234             double light_dir_vec[3];
235             light_dir_vec[0] = light_dir[0] / length_l;
236             light_dir_vec[1] = light_dir[1] / length_l;
237             light_dir_vec[2] = light_dir[2] / length_l;
238
239             // 法線ベクトルnと光源方向ベクトルの内積
240             double ip =
241                 (n[0] * light_dir_vec[0]) +
242                 (n[1] * light_dir_vec[1]) +
243                 (n[2] * light_dir_vec[2]);
244
245             //2パターンの三角形を特定
246             if(p[1] == r[1]){
247                 //x座標が p <= r となるように調整
248                 if(r[0] <  p[0]){
249                     double temp[2];
250                     memcpy(temp, r, sizeof(double) * 2);
251                     memcpy(r, p, sizeof(double) * 2);
252                     memcpy(p, temp, sizeof(double) * 2);
253                     //debug
254                     printf("\n交換後の3点の座標は\npの座標(%f,␣%f)\nqの座標(%f,␣%f)\nrの座標(%f,␣%f)\
                          n"
255                            ,p[0], p[1], q[0], q[1], r[0], r[1]);
256                 }
257                 //debug
258                 if(r[0] == p[0]){
259                     perror("エラー958");
260                 }
261                 //シェーディング処理
262                 //シェーディングの際に画面からはみ出した部分をどう扱うか
263                 //以下の実装はxy座標の範囲を0 <= x, y <= 256として実装している
264                 //三角形pqrをシェーディング
265                 //y座標はp <= r
266                 //debug
267                 if(r[1] < p[1]){
268                     perror("エラーat1855");
269                 }
270                 int i;
271                 i = ceil(p[1]);
272                 for(i;
273                     p[1] <= i && i <= q[1] && 0 <= i && i <= (HEIGHT - 1);
274                     i++){
275                     double x1 = func1(p, q, i);
276                     double x2 = func1(r, q, i);
277                     int j;
278                     j = ceil(x1);
279
280                     for(j;
281                         x1 <= j && j <= x2 && 0 <= j && j <= (WIDTH - 1);
282                         j++){
283
284                         image[i][j][0] =
285                             -1 * ip * diffuse_color[0] *
286                             light_rgb[0] * MAX;
287                         image[i][j][1] =
288                             -1 * ip * diffuse_color[1] *
289                             light_rgb[1] * MAX;
290                         image[i][j][2] =
291                             -1 * ip * diffuse_color[2] *
292                             light_rgb[2] * MAX;
293                     }
294                 }
```

```
295
296             }
297
298             if(p[1] == q[1]){
299                 //x座標が p < q となるように調整
300                 if(q[0] <  p[0]){
301                     double temp[2];
302                     memcpy(temp, q, sizeof(double) * 2);
303                     memcpy(q, p, sizeof(double) * 2);
304                     memcpy(p, temp, sizeof(double) * 2);
305                     //debug
306                     printf("\n交換後の3点の座標は\npの座標(%f,␣%f)\nqの座標(%f,␣%f)\nrの座標(%f,␣%f)\
                            n"
307                            ,p[0], p[1], q[0], q[1], r[0], r[1]);
308                 }
309                 //debug
310                 if(q[0] == p[0]){
311                     perror("エラーat1011");
312                 }
313                 //シェーディング処理
314                 //三角形pqrをシェーディング
315                 //y座標はp <= q
316                 //debug
317                 if(q[1] < p[1]){
318                     perror("エラーat1856");
319                 }
320                 int i;
321                 i = ceil(r[1]);
322
323
324                 for(i;
325                     r[1] <= i && i <= p[1] && 0 <= i && i <= (HEIGHT - 1);
326                     i++){
327                     double x1 = func1(p, r, i);
328                     double x2 = func1(q, r, i);
329
330                     int j;
331                     j = ceil(x1);
332
333                     for(j;
334                         x1 <= j && j <= x2 && 0 <= j && j <= (WIDTH - 1);
335                         j++){
336
337                         image[i][j][0] =
338                             -1 * ip * diffuse_color[0] *
339                             light_rgb[0] * MAX;
340                         image[i][j][1] =
341                             -1 * ip * diffuse_color[1] *
342                             light_rgb[1] * MAX;
343                         image[i][j][2] =
344                             -1 * ip * diffuse_color[2] *
345                             light_rgb[2] * MAX;
346                     }
347                 }
348             }
349
350         }
351         //分割できる
352         //分割してそれぞれ再帰的に処理
353         //分割後の三角形はpp2qとpp2r
354         else{
355             double p2[2];
356
357             p2[0] = func1(q, r, p[1]);
358             p2[1] = p[1];
359             //p2のほうがpのx座標より大きくなるようにする
360             if(p2[0] < p[0]){
361                 double temp[2];
362                 memcpy(temp, p2, sizeof(double) * 2);
363                 memcpy(p2, p, sizeof(double) * 2);
364                 memcpy(p, temp, sizeof(double) * 2);
365             }
366             //分割しても法線ベクトルは同一
367             shading(p, p2, q, n);
368             shading(p, p2, r, n);
369         }
370     }
371 }
372
```

```
373 int main(void){
374     FILE *fp;
375     char *fname = FILENAME;
376 
377     fp = fopen( fname, "w" );
378     //ファイルが開けなかったとき
379     if( fp == NULL ){
380         printf("%sファイルが開けません.\n", fname);
381         return -1;
382     }
383 
384     //ファイルが開けたとき
385     else{
386         //頂点座標をランダムに設定
387         srand(10);
388         for(int i = 0; i < VER_NUM; i++){
389             ver[i][0] = rand() % 80;
390             ver[i][1] = rand() % 80;
391             ver[i][2] = rand() % 50 + 30;
392         }
393 
394         printf("\n初期の頂点座標は以下\n");
395         for(int i = 0; i < VER_NUM; i++){
396             printf("%f\t%f\t%f\n", ver[i][0], ver[i][1], ver[i][2]);
397         }
398         printf("\n");
399 
400         //描画領域を初期化=======================================
401         for(int i = 0; i < 256; i++){
402             for(int j = 0; j < 256; j++){
403                 image[i][j][0] = 0.0 * MAX;
404                 image[i][j][1] = 0.0 * MAX;
405                 image[i][j][2] = 0.0 * MAX;
406             }
407         }
408         //=====================================================
409 
410         //ヘッダー出力
411         fputs(MAGICNUM, fp);
412         fputs("\n", fp);
413         fputs(WIDTH_STRING, fp);
414         fputs("␣", fp);
415         fputs(HEIGHT_STRING, fp);
416         fputs("\n", fp);
417         fputs(MAX_STRING, fp);
418         fputs("\n" ,fp);
419 
420         //各点の透視投影処理
421         perspective_pro();
422 
423         printf("\n撮像領域上の各点の座標のprojected_verの値\n");
424         for(int i = 0; i < VER_NUM; i++){
425             printf("%f\t%f\n", projected_ver[i][0], projected_ver[i][1]);
426         }
427         printf("\n");
428 
429         //シェーディング
430         for(int i = 0; i < SUR_NUM; i++){
431             double a[2], b[2], c[2];
432 
433             a[0] = projected_ver[(sur[i][0])][0];
434             a[1] = projected_ver[(sur[i][0])][1];
435             b[0] = projected_ver[(sur[i][1])][0];
436             b[1] = projected_ver[(sur[i][1])][1];
437             c[0] = projected_ver[(sur[i][2])][0];
438             c[1] = projected_ver[(sur[i][2])][1];
439             //debug
440             printf("\n3点\naの座標(%f,\t%f)\nbの座標(%f,\t%f)\ncの座標(%f,\t%f)\
                    nのシェーディングを行います.\n"
441                    ,a[0], a[1], b[0], b[1], c[0], c[1]);
442 
443 
444             //冗長な処理
445             //透視投影処理の際に法線ベクトル、
446             //光源からの距離を同時に求めておけばよかったが
447             //今更変更できないのでここで再び求める
448 
449             //法線ベクトルを計算
450             //投影前の3点の座標を取得
```

```
451             double A[3], B[3], C[3];
452             A[0] = ver[(sur[i][0])][0];
453             A[1] = ver[(sur[i][0])][1];
454             A[2] = ver[(sur[i][0])][2];
455
456             B[0] = ver[(sur[i][1])][0];
457             B[1] = ver[(sur[i][1])][1];
458             B[2] = ver[(sur[i][1])][2];
459
460             C[0] = ver[(sur[i][2])][0];
461             C[1] = ver[(sur[i][2])][1];
462             C[2] = ver[(sur[i][2])][2];
463
464             //debug
465             printf("\n3次元空間内の3点の座標は\n(%f,\t%f,\t%f)\n(%f,\t%f,\t%f)\n(%f,\t%f,\t%f)\nです\
                    n",
466                    A[0], A[1], A[2],
467                    B[0], B[1], B[2],
468                    C[0], C[1], C[2]);
469
470             //ベクトルAB, ACから外積を計算して
471             //法線ベクトルnを求める
472             double AB[3], AC[3], n[3];
473             AB[0] = B[0] - A[0];
474             AB[1] = B[1] - A[1];
475             AB[2] = B[2] - A[2];
476
477             AC[0] = C[0] - A[0];
478             AC[1] = C[1] - A[1];
479             AC[2] = C[2] - A[2];
480
481             n[0] = (AB[1] * AC[2]) - (AB[2] * AC[1]);
482             n[1] = (AB[2] * AC[0]) - (AB[0] * AC[2]);
483             n[2] = (AB[0] * AC[1]) - (AB[1] * AC[0]);
484
485             //長さを1に調整
486             double length_n =
487                 sqrt(pow(n[0], 2.0) +
488                      pow(n[1], 2.0) +
489                      pow(n[2], 2.0));
490
491             n[0] = n[0] / length_n;
492             n[1] = n[1] / length_n;
493             n[2] = n[2] / length_n;
494
495             printf("\n法線ベクトルは\n(%f,\t%f,\t%f)\nです\n",
496                    n[0], n[1], n[2]);
497
498             //平面iの投影先の三角形をシェーディング
499             shading(a, b, c, n);
500         }
501
502         //imageの出力
503         for(int i = 0; i < 256; i++){
504             for(int j = 0; j < 256; j++){
505                 char r[256];
506                 char g[256];
507                 char b[256];
508                 char str[1024];
509
510                 sprintf(r, "%d", (int)round(image[i][j][0]));
511                 sprintf(g, "%d", (int)round(image[i][j][1]));
512                 sprintf(b, "%d", (int)round(image[i][j][2]));
513                 sprintf(str, "%s\t%s\t%s\n", r, g, b);
514                 fputs(str, fp);
515             }
516         }
517
518
519
520     }
521     fclose(fp);
522
523     printf("\nppmファイル␣%s␣の作成が完了しました.\n", fname );
524     return 0;
525 }
```

5 実行例