〔非公開〕

ＴＲ－Ｍ－００２９

多視点画像と奥行きマップを用いた三次元シーンの再構築

青　木　　利　道
Toshimichi　AOKI

ジョンイル　パーク　　井　上　　誠　喜
Jong-Il　Park　　Seiki　INOUE

１９９７．９．３０

ＡＴＲ知能映像通信研究所

多視点画像と奥行きマップを用いた三次元シーンの再構築

3-D Scene Reconstruction from Multiple-stereo Image and Its Depth Map

概要

本報告書では，多視点画像とその奥行きマップから三次元シーンを復元する手法を述べる．視点の移動にともない現れる未知領域を，多視点画像と奥行きマップより推測し補間する手法についても述べる．

ATR 知能映像通信研究所 第 3 研究室

学外実習生 青木利道

1 はじめに

十字に配置されたカメラを用い5枚の画像を取得すると，その5枚の画像からステレオマッチングにより奥行きマップを得ことができる[1]．本研究では，カメラを等間隔に十字に配置し[Fig1]，5枚の画像と中心カメラの奥行きマップが得られているのもとする．

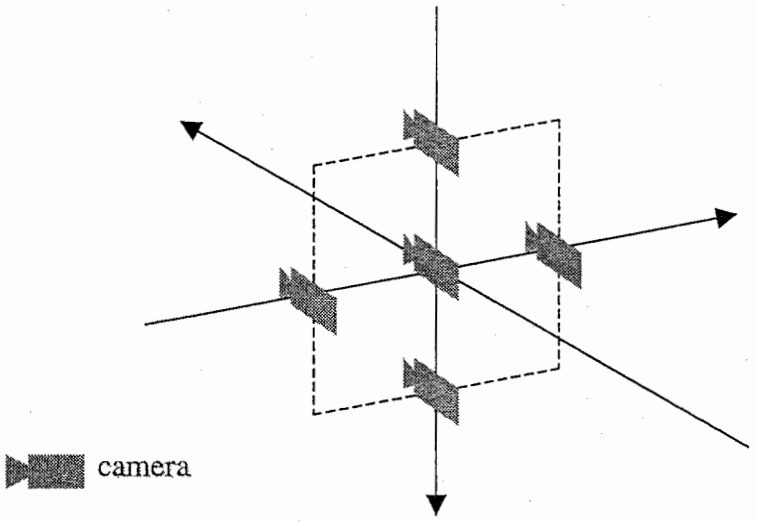


Fig1: カメラ配置

また，カメラ間の距離，カメラの焦点距離は既知で，奥行きマップはカメラ間の視差を輝度値に変換した視差画像[fig2]として得られているものとする．

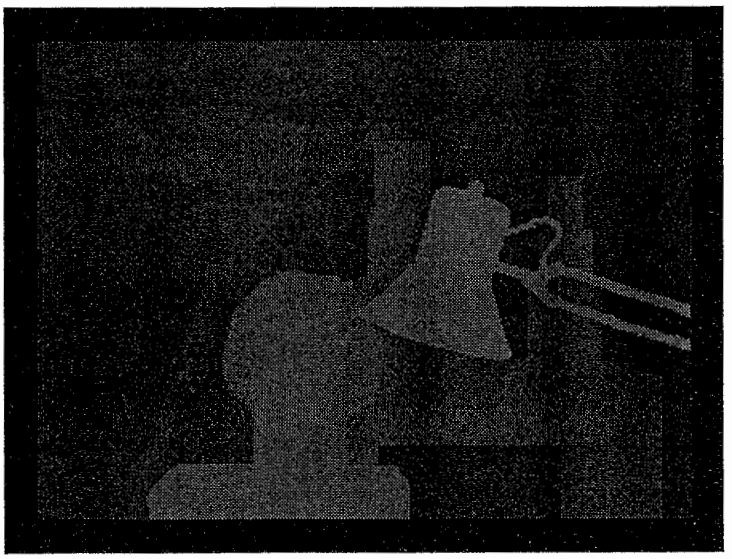


Fig2: 視差画像の例

2 三次元シーンの復元

Fig3 に示すように座標系を定義する．十字に配置されたカメラの中心カメラの位置を座標系の原点 0，水平方向を x 軸，垂直方向を y 軸，中心カメラの光軸方向を z 軸とする．またカメラの焦点距離を F とする．

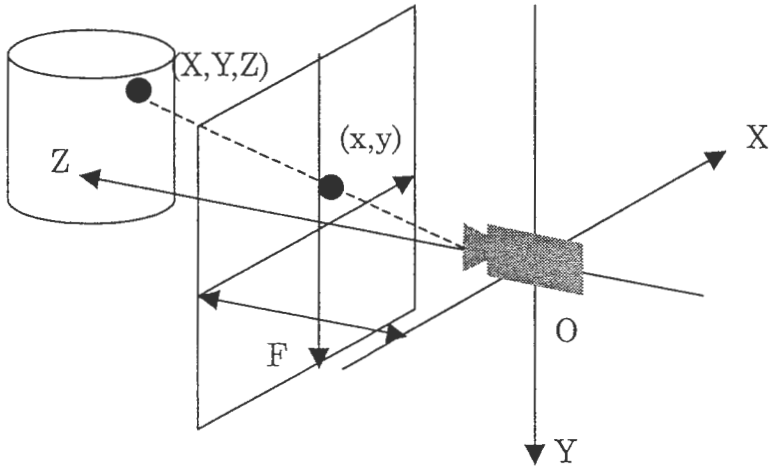


Fig3: 座標系の定義

このように定義すると式 1 のように，三次元空間中の点(X,Y,Z)は画像平面(x,y)に投影される．

$$x = F\frac{X}{Z}, \quad y = F\frac{Y}{Z} \tag{1}$$

また，中心カメラと他のカメラの距離を L，奥行きマップから得られる視差を d とすれば

$$Z = \frac{FL}{d} \tag{2}$$

である．式(1)と式(2)より三次元空間中の点(X,Y,Z)は

$$X = \frac{xL}{d}, \quad Y = \frac{yL}{d}, \quad Z = \frac{FL}{d} \tag{3}$$

のように三次元位置を復元することができる．

中心カメラ画像をテクスチャーマッピングすることで，三次元シーンの復元が可能である．

3 視点の移動により出現する未知領域の補間

復元した三次元シーンは中心カメラからの情報しか持っていないために，視点を中心カメラ位置以外に移動をすると，奥行き情報未知の領域が新たに生じてくる．三次元シーンを

復元したにも関わらず，視点の移動ができなくては一枚の実画像を見ているのと同じである．視点の移動を可能とするために，奥行き未知の領域を補間する．

3.1 視点移動により出現する未知領域

視点を中心カメラ位置以外に移動すると，Fig4 のように新たに見えてくる領域と隠されて見えなくなる領域が現れる．新たに現れる領域は，奥行き値が分からない．

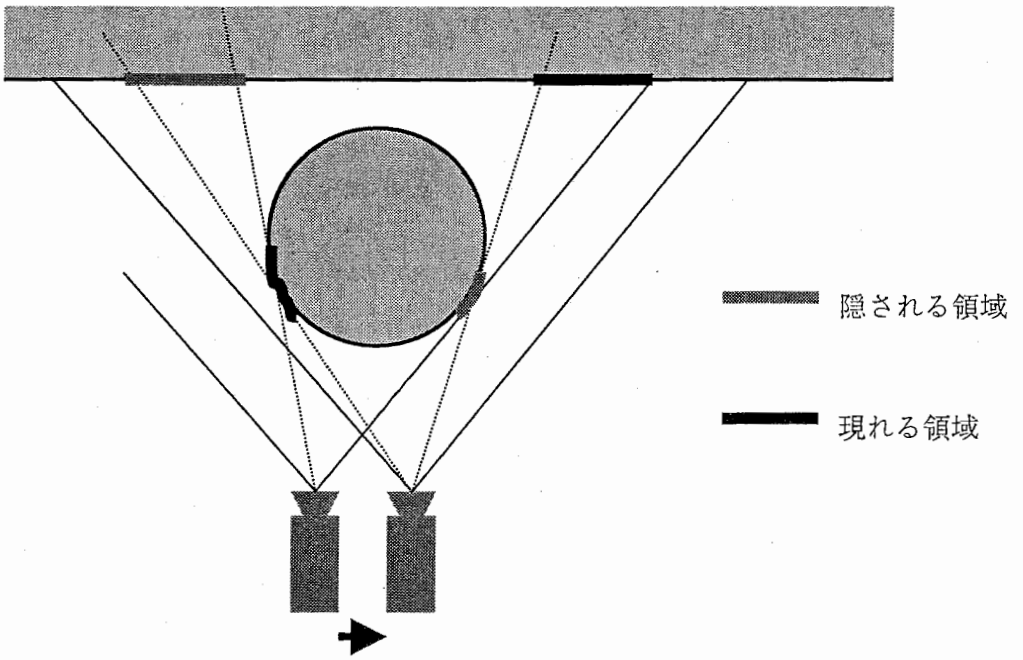


Fig4: 視点移動により現れる領域と隠される領域

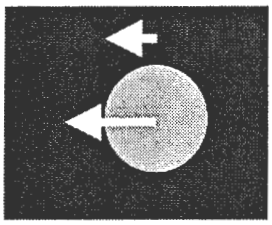
3.2 未知領域の補間

カメラを十字に配置して得られる画像は，5 枚ある．にもかかわらず，三次元シーンの復元には中心に画像と奥行きマップしか用いていない．ここで，上下左右のカメラの画像を有効に利用することを考える．カメラの画像はテクスチャデータとして用いることができる．加えて奥行き情報があれば三次元シーンは復元することができる．中心カメラの奥行きマップから，上下左右の奥行きマップを推定して，新たに現れる領域を補間する．

視点を右カメラ位置まで移動し，現れる領域を補完する手法を述べる．奥行きマップの輝度値は視差量を示すので，各々の画素を視差量分左へ移動する．隠される領域については，複数の画素値が割り当てられるが，これは常識的に手前の画素値の画素値を選ぶとする．値を持たない画素が生じるが，これが新たに現れる領域である．この値を持たない画素を推測し補間することで，視点の移動を助ける．現実の世界で考えると，近くのものが大きく動き，遠くのものはあまり動かないために，近くのものの後ろにあった遠くの領域が現

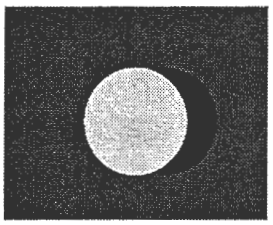
れる．この事から，新たに現れた領域は，隣接する領域の中でもっとも遠くの領域の画素値で補完しても多くの場合問題無い．そこで，新たに現れた領域に隣接する中でもっとも遠くの領域の画素値で奥行き情報の補間を行う．奥行き情報の補間を行った領域を「補間領域」と呼ぶことにする．補間領域を，右カメラ画像を用いてテクスチャーマッピングする．中心カメラの情報から復元した三次元シーンに，このテクスチャーマッピングした右カメラ補間領域を追加することで，カメラの右移動を助けることができる．追加手順をFig5 に示す．

Step1



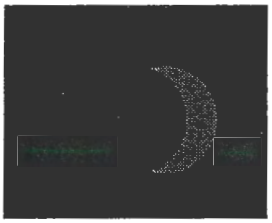
視差量に応じて，書く画素を左へ移動する．一つの画素に複数重なった場合にはもっとも近くの画素値とする．

Step2



ラスタスキャンし，未知領域を見つける．同時に隣接する領域の画素値（奥行き）を調べる．

Step3



未知領域を，遠くの領域の画素値で補完する．この補間領域を，中心カメラの座標系に戻し追加する．

Fig5: 右カメラ補間領域の生成法

同様に上下，左カメラ画像を用いた補間領域を生成し，中心カメラ座標で再構成したシーンに追加することである程度のカメラ移動を可能にする．

4 実験

提案した補間手法の有効性を確認するために，与えられた画像と奥行きマップを用いて実験を行った．

4．1 実験データ

実験に使用したデータを Fig6 に示す．配置は，実際のカメラ配置に対応している．それぞれの画像サイズは 288*384 である．これらの画像より得られた奥行きマップを Fig7 に示す．

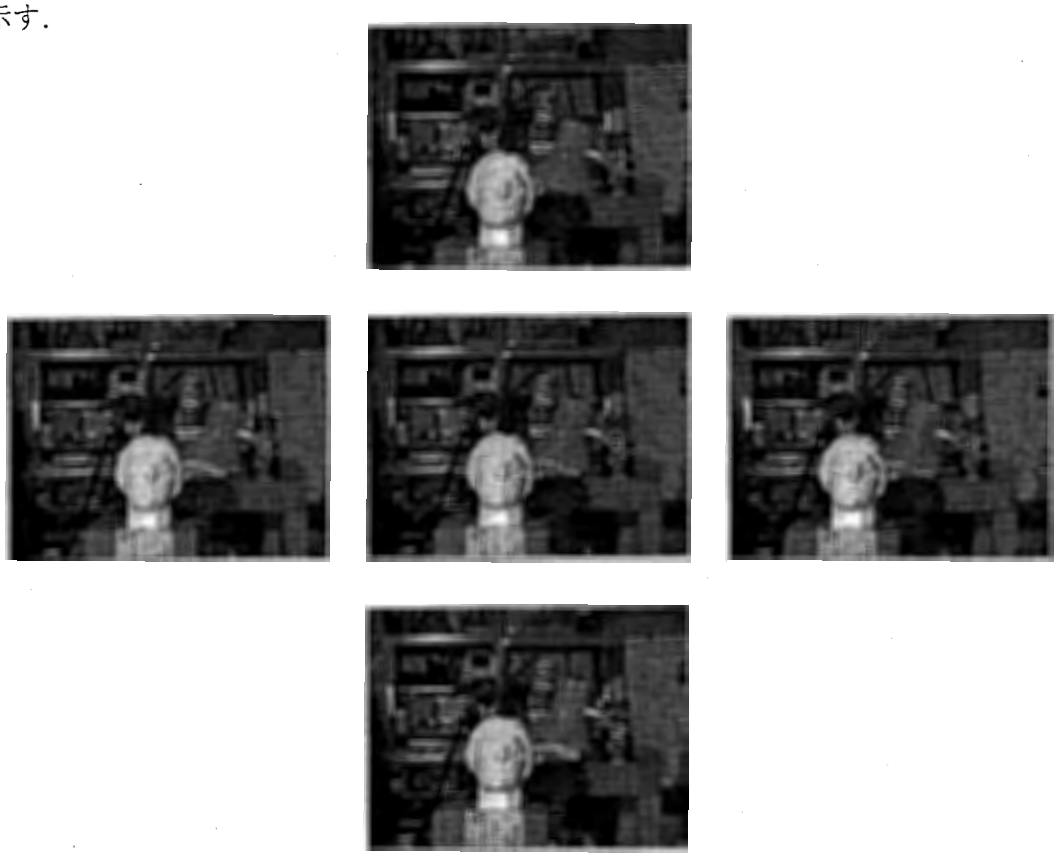


Fig6: 実験データ

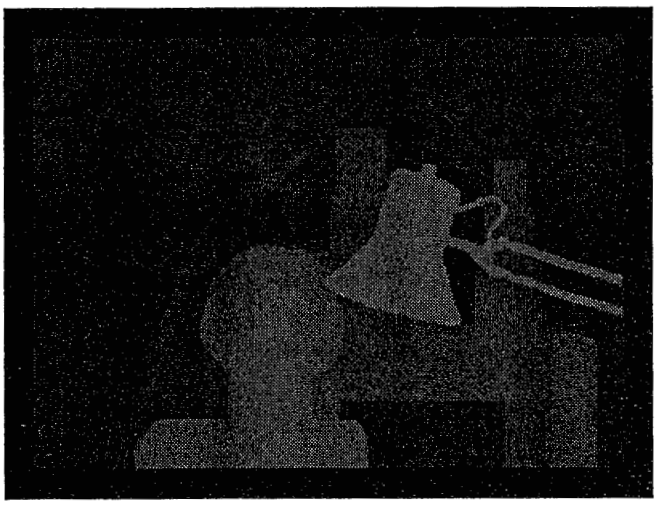


Fig7: 実験データ　ー奥行きマップー

4．2 復元した三次元シーンのビューア

復元した三次元シーンは，点の三次元位置と色の情報しか持っていない．点を三次元空間に配置しても，実世界のように裏にものが隠れるといった様子は表現しにくい．そこで，隣接するてんでポリゴンをはる処理を行い面を構成し，印面消去をすることで，実世界と同様の見えかたをするビューアを作成した．初期位置（中心カメラ位置）で，ポリゴンの方向ベクトルと中心カメラの光軸方向ベクトルがほぼ垂直になる場合，そのようなポリゴンが存在する確率は低いので除去してある．以下の実験結果の表示にはすべてこのビューアを用いている．

4．3 補間を行わない三次元シーンの復元

中心カメラ画像を，５つの画像から得られた奥行きマップのみから三次元シーンを復元する実験した．復元したシーンを中心カメラ位置から見た結果を Fig8 に示す．また，視線方向はそのままで視点位置を左に移動した結果を Fig9 に示す．

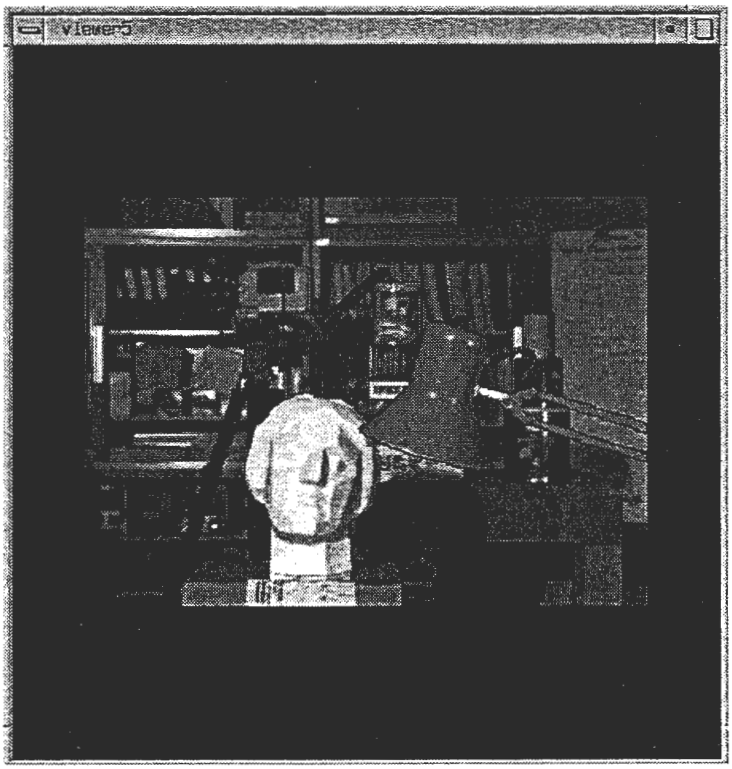


Fig8: 三次元シーンの復元結果 －補間なし－

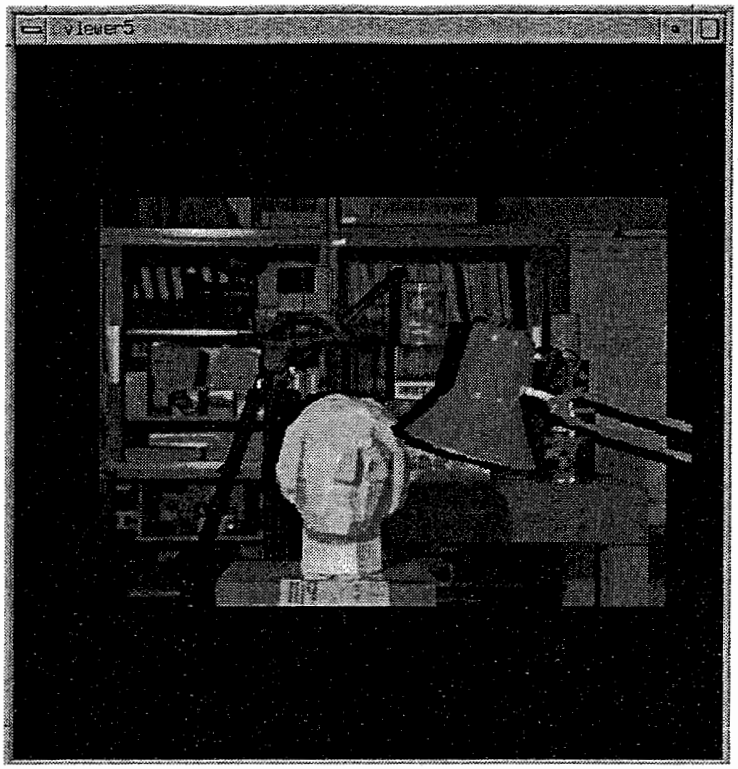


Fig9: 三次元シーンの復元結果 ー補間なし，視点左に移動ー

視点が中心カメラ位置にあるときは，中心カメラ画像を見ているのと変わりがない．視点の移動を行うと，未知領域が現れてくる．視点が移動できるので，三次元空間的な間隔はつかむことができるが，未知領域の存在か気になる．

4．4 補間を行う三次元シーンの復元

つぎに，復元した三次元シーンに，補間領域を追加する実験を行う．生成した補間領域をFig10 に示す．配置は，補間領域のカメラ位置に対応する．補間領域の奥行き情報と対応するカメラの画像からシーンの一部を復元し，先に復元したシーンに追加する．復元結果をFig11 に示す．また，先と同様に視線方向そのままで視点位置を少し左に移動した結果をFig12 に示す．

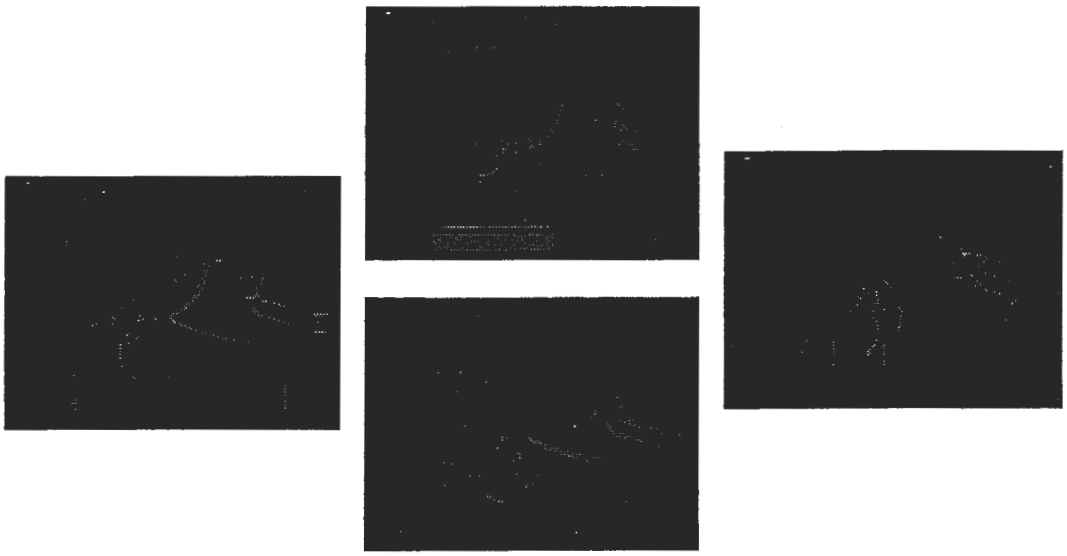


Fig10: 補間領域の奥行き情報

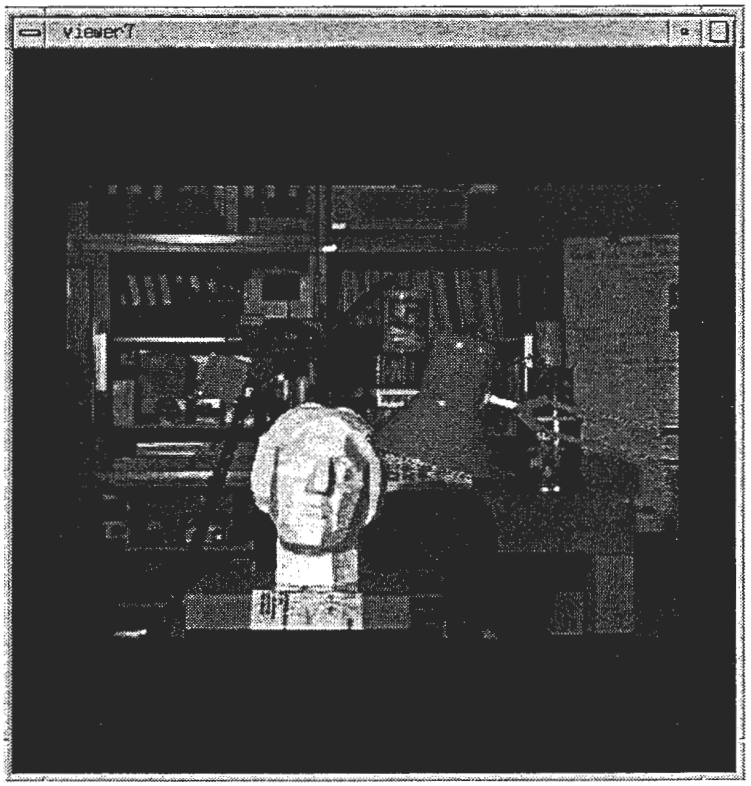


Fig11: 三次元シーンの復元結果 －補間あり－

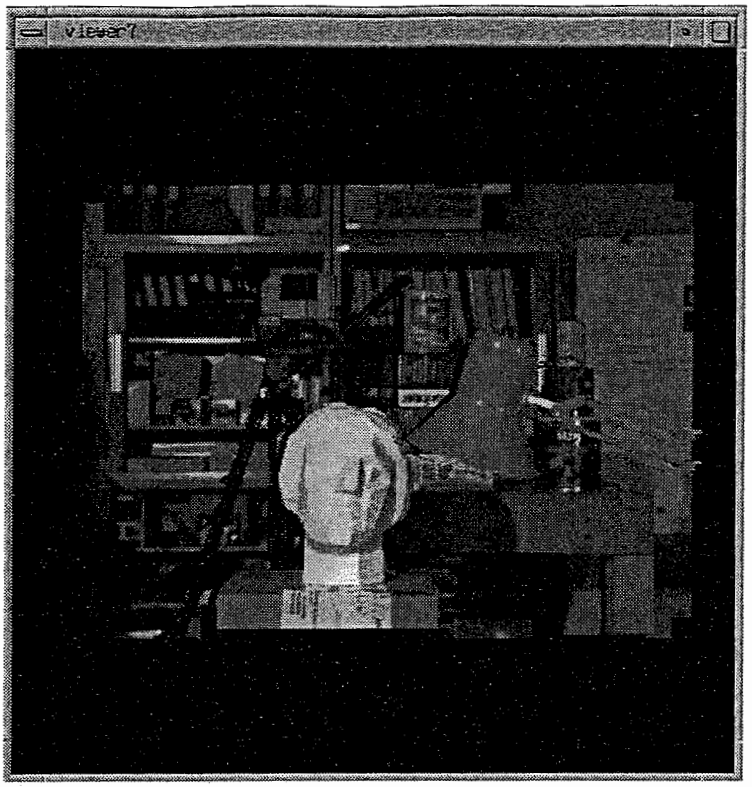


Fig12: 三次元シーンの復元結果 ー補間あり，視点左に移動ー

視点を左に移動したが，未知領域はあまり目立たない．補間による効果か確認できる．右カメラ位置より視点を右に移動すると，未知領域が現れる．上下，左についても同様である．つまり，カメラが存在する領域内では視点を自由に動かすことが可能になった．

5 考察

三次元シーンに補間処理を行うことにより，ある程度の視点の移動を可能にした．視点が上下左右にのカメラ位置の範囲内にあれば，未知領域が出現することはない．実世界と同様な見えがたが実現できる．視点をさらに大きくしたときの補間処理は，テクスチャの情報すらないために難しい．

奥行きの補間方法には，線形補間を用いるなどの方法が考えられる．奥行き情報の補間がうまくいっていないために，不自然にみえる部分もあったが補間の手法を改善することで解決できるかもしれない．

まとめ

今回の自習期間中に行ったことをまとめる．

1. 三次元シーンの復元
2. 復元した三次元シーン内を移動可能なビューア作成
3. 未知領域の補間による，視点移動への対応

未知領域の補間処理方法を提案し，実装を行うことで補間処理は有効であることが確認できた．

謝辞

本実習において多大なご指導をいただきました朴鐘一研究員に心から感謝の意を表します．また，数々の助言やご支援をいただきました井上研究室長をはじめとする知能映像通信研究所第３研究室の方々に深く感謝いたします．また，私に貴重な体験の場を提供して下さいましたＡＴＲ知能映像通信研究所ならびに奈良先端科学技術大学院大学の関係諸氏にこの場を借りてお礼申し上げます．

参考文献

[1]朴鐘一，福田浩士，井上誠喜，“シーン記述のための奥行き情報抽出とその利用法”，ITE'96

ソースプログラム

本研究のために作成したプログラムについて説明する.

- makeAddImage.c
  未知領域の補間を行うための奥行きマップ，およびテクスチャデータを生成する.
  入力は，中心カメラの奥行きマップと実画像（テクスチャデータ）.
  オプションにより，補間を行うカメラ位置（上下左右）を選択.
  入出力画像タイプは，ppm 形式のみ.
- 5imageViewer.c
  復元した三次元シーンを見ることができるビューア.
  入力は，中心カメラの奥行きマップと実画像．上下左右の補間データ（奥行きマップとテクスチャデータ）.
  左ボタンドラッグで，xy 方向に平行移動可能．中心ボタンドラッグで z 方向に傾向移動可能.
  右ボタンのメニューにより，ポイント表示，ワイヤーフレーム表示，ポリゴン表示の選択が可能.
  入力画像タイプは，ppm 形式のみ.
  ソース中の定数 NUM を変更することにより，入力画像数を変更できる．1 にすれば 1 対のデータ（奥行きマップ，テクスチャ）から復元した三次元シーンを見ることも可能.

```
/*-------------------------------------------------------
|   makeAddImage.c                                      |
|   1997/10/20                                          |
|   Toshimichi AOKI (NAIST)                             |
 -------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>

#define  TOP  0        /* 生成補間画像用スウィッチ */
#define  BUTTOM  1
#define  RIGHT  2
#define  LEFT  3
#define  COLOR  0     /* ppm画像タイプ */
#define  GREY  1

typedef struct img {
        char type;           /* 画像タイプ */
        unsigned int xsize; /* xサイズ */
        unsigned int ysize; /* yサイズ */
        unsigned int depth; /* depth */
        char *rdata;         /* r成分 */
        char *gdata;         /* g成分 */
        char *bdata;         /* b成分 */
} IMAGE;

/*-------------------------------------------------
   ppg/pgm(raw)形式のファイルをimg構造体に読み込む

     返り値  読み込み成功   0
                  失敗  -1
-------------------------------------------------*/
int
readppm(filename, data)
    char *filename;
    IMAGE *data;
{
    FILE *fp;
    char buf[80];
    unsigned long i = 0, size;

    fp = fopen(filename, "rb");
    if(fp == NULL) return -1;

    fgets(buf, 80, fp);
    if(strncmp("P6", buf, 2) == 0) data->type = COLOR;
    else if(strncmp("P5", buf, 2) == 0) data->type = GREY;
    else {
        puts("unknown file type");
        return -1;
    }
    do{
        fgets(buf, 80, fp);
    }while(buf[0] == '#');
    sscanf(buf, "%d %d", &(data->xsize), &(data->ysize));
    fgets(buf, 80, fp);
    sscanf(buf, "%d", &(data->depth));

    size = data->xsize*data->ysize;
    data->rdata = (char *)malloc(size);
    if(data->type == COLOR){
        data->gdata = (char *)malloc(size);
        data->bdata = (char *)malloc(size);
        while(i < size){
            data->rdata[i] = fgetc(fp);
            data->gdata[i] = fgetc(fp);
            data->bdata[i] = fgetc(fp);
```



```
            i++;
        }
    } else {
        fread(data->rdata, size, 1, fp);
    }
    fclose(fp);
    return 0;
}

/*-------------------------------------------------------
   img構造体の中身をppg/pgm(raw)形式のファイルに書き出す

     返り値  書き込み成功          0
             ファイルオープン失敗  -1
-------------------------------------------------------*/
int writeppm(filename, data)
    char *filename;
    IMAGE *data;
{
        FILE *fp;
        char buf[80];
        unsigned long i=0, size;
        if((fp=fopen(filename, "wb"))==NULL){
          fprintf(stderr, "cannot open file: %s\n", filename);
          return -1;
        }
        if(data->type==COLOR) fprintf(fp,"P6\n");
        else fprintf(fp,"P5\n");
        fprintf(fp,"# \n");
        fprintf(fp,"%d %d\n", data->xsize, data->ysize);
        fprintf(fp,"%d\n", data->depth);
        size = data->xsize*data->ysize;
        if(data->type==GREY)
                fwrite(data->rdata, size, 1, fp);
        else while(i<size){
                fputc(data->rdata[i], fp);
                fputc(data->gdata[i], fp);
                fputc(data->bdata[i], fp);
                i++;
        }
        fclose(fp);
        return 0;
}

/*-------------------------------------------------------
  補間画像生成
-------------------------------------------------------*/
void
makeAddImage(depth_image, color_image, add_depth_image, add_color_image, sw)
    IMAGE depth_image, color_image;          /* 入力画像 */
    IMAGE *add_depth_image, *add_color_image; /* 出力画像 */
    int sw;                                  /* 生成画像スウィッチ */
{
    int x, y, xx, yy;
    int a, a_max = 0, a_min = 255;

    char *buff_data1, *buff_data2;           /* テンポラリバッファ */

    add_depth_image->type = depth_image.type;
    add_depth_image->xsize = depth_image.xsize;
    add_depth_image->ysize = depth_image.ysize;
    add_depth_image->depth = depth_image.depth;
    add_depth_image->rdata = (char *)malloc(depth_image.xsize*depth_image.ysize);

    add_color_image->type = color_image.type;
    add_color_image->xsize = color_image.xsize;
```

```
    add_color_image->ysize = color_image.ysize;
    add_color_image->depth = color_image.depth;
    add_color_image->rdata = (char *)malloc(color_image.xsize*color_image.ysize);
    add_color_image->gdata = (char *)malloc(color_image.xsize*color_image.ysize);
    add_color_image->bdata = (char *)malloc(color_image.xsize*color_image.ysize);

/* 視差の最大値、最小値の取得 */
    for(y = 0; y < depth_image.ysize; y++){
        for(x = 0; x < depth_image.xsize; x++){
            if(a_max < depth_image.rdata[y * depth_image.xsize + x])
                a_max = depth_image.rdata[y * depth_image.xsize + x];
            if(a_min > depth_image.rdata[y * depth_image.xsize + x])
                a_min = depth_image.rdata[y * depth_image.xsize + x];
        }
    }

    switch(sw){
/*
  上カメラ補間画像生成
*/
    case TOP:
        buff_data1 = (char *)malloc(depth_image.xsize*depth_image.ysize);

/* 視差量だけ画像をスライドさせる */
        for(a = a_min; a <= a_max; a++){
            for(x = 0; x < depth_image.xsize; x++){
                for(y = 0; y < depth_image.ysize; y++){
                    if(y + depth_image.rdata[y * depth_image.xsize + x] < depth_image
.ysize)
                        buff_data1[(y + depth_image.rdata[y * depth_image.xsize + x])
 * depth_image.xsize + x] = depth_image.rdata[y * depth_image.xsize + x];
                }
            }
        }
        buff_data2 = (char *)malloc(depth_image.xsize*depth_image.ysize);

/* 補間する奥行きマップを生成 */
        for(x = 0; x < depth_image.xsize; x++){
            for(y = depth_image.ysize - 1; y >= 0; y--){
                if(buff_data1[y * depth_image.xsize + x] == 0){
                    for(yy = y - 1; yy >= 0; yy--){
                        if(buff_data1[yy * depth_image.xsize + x] != 0){
                            buff_data2[y * depth_image.xsize + x] = buff_data1[yy * d
epth_image.xsize + x];
                            break;
                        }
                    }
                }
            }
        }
        free(buff_data1);

/* 中心カメラの座標系に戻す */
        for(a = a_min; a <= a_max; a++){
            for(x = 0; x < depth_image.xsize; x++){
                for(y = 0; y < depth_image.ysize; y++){
                    if(buff_data2[y * depth_image.xsize + x] != 0 && y - buff_data2[y
 * depth_image.xsize + x] >= 0){
                        add_depth_image->rdata[(y - buff_data2[y * depth_image.xsize
+ x]) * depth_image.xsize + x] = buff_data2[y * depth_image.xsize + x];
                        add_color_image->rdata[(y - buff_data2[y * depth_image.xsize
+ x]) * depth_image.xsize + x] = color_image.rdata[y * depth_image.xsize + x];
                        add_color_image->gdata[(y - buff_data2[y * depth_image.xsize
+ x]) * depth_image.xsize + x] = color_image.gdata[y * depth_image.xsize + x];
                        add_color_image->bdata[(y - buff_data2[y * depth_image.xsize
+ x]) * depth_image.xsize + x] = color_image.bdata[y * depth_image.xsize + x];
```



```
                    }
                }
            }
        }
        free(buff_data2);

/* つなぎ目をごまかすため、1ラインデータを追加する */
        for(x = 0; x < depth_image.xsize; x++){
            for(y = 0; y < depth_image.ysize - 1; y++){
                if(add_depth_image->rdata[y * add_depth_image->xsize + x] == 0 && add
_depth_image->rdata[(y + 1) * add_depth_image->xsize + x] != 0){
                    add_depth_image->rdata[y * add_depth_image->xsize + x] = add_dept
h_image->rdata[(y + 1) * add_depth_image->xsize + x];
                    add_color_image->rdata[y * add_depth_image->xsize + x] = add_colo
r_image->rdata[(y + 1) * add_depth_image->xsize + x];
                    add_color_image->gdata[y * add_depth_image->xsize + x] = add_colo
r_image->gdata[(y + 1) * add_depth_image->xsize + x];
                    add_color_image->bdata[y * add_depth_image->xsize + x] = add_colo
r_image->bdata[(y + 1) * add_depth_image->xsize + x];
                }
            }
        }
        break;

/*
  下カメラ補間画像生成
*/
    case BUTTOM:
        buff_data1 = (char *)malloc(depth_image.xsize*depth_image.ysize);

/* 視差量だけ画像をスライドさせる */
        for(a = a_min; a <= a_max; a++){
            for(x = 0; x < depth_image.xsize; x++){
                for(y = depth_image.ysize - 1; y >= 0; y--){
                    if(y - depth_image.rdata[y * depth_image.xsize + x] >= 0)
                        buff_data1[(y - depth_image.rdata[y * depth_image.xsize + x])
 * depth_image.xsize + x] = depth_image.rdata[y * depth_image.xsize + x];
                }
            }
        }
        buff_data2 = (char *)malloc(depth_image.xsize*depth_image.ysize);

/* 補間する奥行きマップを生成 */
        for(x = 0; x < depth_image.xsize; x++){
            for(y = 0; y < depth_image.ysize; y++){
                if(buff_data1[y * depth_image.xsize + x] == 0){
                    for(yy = y + 1; yy < depth_image.ysize; yy++){
                        if(buff_data1[yy * depth_image.xsize + x] != 0){
                            buff_data2[y * depth_image.xsize + x] = buff_data1[yy * d
epth_image.xsize + x];
                            break;
                        }
                    }
                }
            }
        }
        free(buff_data1);

/* 中心カメラの座標系に戻す */
        for(a = a_min; a <= a_max; a++){
            for(x = 0; x < depth_image.xsize; x++){
                for(y = 0; y < depth_image.ysize; y++){
                    if(buff_data2[y * depth_image.xsize + x] != 0 && y + buff_data2[y
 * depth_image.xsize + x] < depth_image.ysize){
                        add_depth_image->rdata[(y + buff_data2[y * depth_image.xsize
+ x]) * depth_image.xsize + x] = buff_data2[y * depth_image.xsize + x];
```

```
                        add_color_image->rdata[(y + buff_data2[y * depth_image.xsize
+ x]) * depth_image.xsize + x] = color_image.rdata[y * depth_image.xsize + x];
                        add_color_image->gdata[(y + buff_data2[y * depth_image.xsize
+ x]) * depth_image.xsize + x] = color_image.gdata[y * depth_image.xsize + x];
                        add_color_image->bdata[(y + buff_data2[y * depth_image.xsize
+ x]) * depth_image.xsize + x] = color_image.bdata[y * depth_image.xsize + x];
                    }
                }
            }
        }
        free(buff_data2);

/* つなぎ目をごまかすため、1ラインデータを追加する */
        for(x = 0; x < depth_image.xsize; x++){
            for(y = depth_image.ysize; y >= 0; y--){
                if(add_depth_image->rdata[y * add_depth_image->xsize + x] == 0 && add
_depth_image->rdata[(y - 1) * add_depth_image->xsize + x] != 0){
                    add_depth_image->rdata[y * add_depth_image->xsize + x] = add_dept
h_image->rdata[(y - 1) * add_depth_image->xsize + x];
                    add_color_image->rdata[y * add_depth_image->xsize + x] = add_colo
r_image->rdata[(y - 1) * add_depth_image->xsize + x];
                    add_color_image->gdata[y * add_depth_image->xsize + x] = add_colo
r_image->gdata[(y - 1) * add_depth_image->xsize + x];
                    add_color_image->bdata[y * add_depth_image->xsize + x] = add_colo
r_image->bdata[(y - 1) * add_depth_image->xsize + x];
                }
            }
        }
        break;

/*
  右カメラ補間画像生成
*/
    case RIGHT:
        buff_data1 = (char *)malloc(depth_image.xsize*depth_image.ysize);

/* 視差量だけ画像をスライドさせる */
        for(a = a_min; a <= a_max; a++){
            for(y = 0; y < depth_image.ysize; y++){
                for(x = 0; x < depth_image.xsize; x++){
                    if(x - depth_image.rdata[y * depth_image.xsize + x] >= 0)
                        buff_data1[y * depth_image.xsize + x - depth_image.rdata[y *
depth_image.xsize + x]] = depth_image.rdata[y * depth_image.xsize + x];
                }
            }
        }
        buff_data2 = (char *)malloc(depth_image.xsize*depth_image.ysize);

/* 補間する奥行きマップを生成 */
        for(y = 0; y < depth_image.ysize; y++){
            for(x = 0; x < depth_image.xsize; x++){
                if(buff_data1[y * depth_image.xsize + x] == 0){
                    for(xx = x + 1; xx < depth_image.xsize; xx++){
                        if(buff_data1[y * depth_image.xsize + xx] != 0){
                            buff_data2[y * depth_image.xsize + x] = buff_data1[y * de
pth_image.xsize + xx];
                            break;
                        }
                    }
                }
            }
        }
        free(buff_data1);

/* 中心カメラの座標系に戻す */
        for(a = a_min; a <= a_max; a++){
```



```
            for(y = 0; y < depth_image.ysize; y++){
                for(x = depth_image.xsize - 1; x >= 0; x--){
                    if(buff_data2[y * depth_image.xsize + x] != 0 && x + buff_data2[y
 * depth_image.xsize + x] < depth_image.xsize){
                        add_depth_image->rdata[y * depth_image.xsize + x + buff_data2
[y * depth_image.xsize + x]] = buff_data2[y * depth_image.xsize + x];
                        add_color_image->rdata[y * depth_image.xsize + x + buff_data2
[y * depth_image.xsize + x]] = color_image.rdata[y * depth_image.xsize + x];
                        add_color_image->gdata[y * depth_image.xsize + x + buff_data2
[y * depth_image.xsize + x]] = color_image.gdata[y * depth_image.xsize + x];
                        add_color_image->bdata[y * depth_image.xsize + x + buff_data2
[y * depth_image.xsize + x]] = color_image.bdata[y * depth_image.xsize + x];
                    }
                }
            }
        }
        free(buff_data2);

/* つなぎ目をごまかすため、1ラインデータを追加する */
        for(y = 0; y < add_depth_image->ysize; y++){
            for(x = add_depth_image->xsize; x > 0; x--){
                if(add_depth_image->rdata[y * add_depth_image->xsize + x] == 0 && add
_depth_image->rdata[y * add_depth_image->xsize + x - 1] != 0){
                    add_depth_image->rdata[y * add_depth_image->xsize + x] = add_dept
h_image->rdata[y * add_depth_image->xsize + x - 1];
                    add_color_image->rdata[y * add_depth_image->xsize + x] = add_colo
r_image->rdata[y * add_depth_image->xsize + x - 1];
                    add_color_image->gdata[y * add_depth_image->xsize + x] = add_colo
r_image->gdata[y * add_depth_image->xsize + x - 1];
                    add_color_image->bdata[y * add_depth_image->xsize + x] = add_colo
r_image->bdata[y * add_depth_image->xsize + x - 1];
                }
            }
        }
        break;

/*
  左カメラ補間画像生成
*/
    case LEFT:
        buff_data1 = (char *)malloc(depth_image.xsize*depth_image.ysize);

/* 視差量だけ画像をスライドさせる */
        for(a = a_min; a <= a_max; a++){
            for(y = 0; y < depth_image.ysize; y++){
                for(x = 0; x < depth_image.xsize; x++){
                    if(x + depth_image.rdata[y * depth_image.xsize + x] < depth_image
.xsize)
                        buff_data1[y * depth_image.xsize + x + depth_image.rdata[y *
depth_image.xsize + x]] = depth_image.rdata[y * depth_image.xsize + x];
                }
            }
        }
        buff_data2 = (char *)malloc(depth_image.xsize*depth_image.ysize);

/* 補間する奥行きマップを生成 */
        for(y = 0; y < depth_image.ysize; y++){
            for(x = depth_image.xsize - 1; x >= 0; x--){
                if(buff_data1[y * depth_image.xsize + x] == 0){
                    for(xx = x - 1; xx >= 0; xx--){
                        if(buff_data1[y * depth_image.xsize + xx] != 0){
                            buff_data2[y * depth_image.xsize + x] = buff_data1[y * de
pth_image.xsize + xx];
                            break;
                        }
                    }
```

```
                }
            }
        }
        free(buff_data1);

/* 中心カメラの座標系に戻す */
        for(a = a_min; a <= a_max; a++){
            for(y = 0; y < depth_image.ysize; y++){
                for(x = 0; x < depth_image.xsize; x++){
                    if(buff_data2[y * depth_image.xsize + x] != 0 && x - buff_data2[y
 * depth_image.xsize + x] >= 0){
                        add_depth_image->rdata[y * depth_image.xsize + x - buff_data2
[y * depth_image.xsize + x]] = buff_data2[y * depth_image.xsize + x];
                        add_color_image->rdata[y * depth_image.xsize + x - buff_data2
[y * depth_image.xsize + x]] = color_image.rdata[y * depth_image.xsize + x];
                        add_color_image->gdata[y * depth_image.xsize + x - buff_data2
[y * depth_image.xsize + x]] = color_image.gdata[y * depth_image.xsize + x];
                        add_color_image->bdata[y * depth_image.xsize + x - buff_data2
[y * depth_image.xsize + x]] = color_image.bdata[y * depth_image.xsize + x];
                    }
                }
            }
        }
        free(buff_data2);

/* つなぎ目をごまかすため、1ラインデータを追加する */
        for(y = 0; y < add_depth_image->ysize; y++){
            for(x = 0; x < depth_image.xsize - 1; x++){
                if(add_depth_image->rdata[y * add_depth_image->xsize + x] == 0 && add
_depth_image->rdata[y * add_depth_image->xsize + x + 1] != 0){
                    add_depth_image->rdata[y * add_depth_image->xsize + x] = add_dept
h_image->rdata[y * add_depth_image->xsize + x + 1];
                    add_color_image->rdata[y * add_depth_image->xsize + x] = add_colo
r_image->rdata[y * add_depth_image->xsize + x + 1];
                    add_color_image->gdata[y * add_depth_image->xsize + x] = add_colo
r_image->gdata[y * add_depth_image->xsize + x + 1];
                    add_color_image->bdata[y * add_depth_image->xsize + x] = add_colo
r_image->bdata[y * add_depth_image->xsize + x + 1];
                }
            }
        }
        break;

    default:
        fprintf(stderr, "Unknown option!!\n");
        exit(0);
    }
}

/*------------------------------------------------
  メイン関数
------------------------------------------------*/
int
main(argc, argv)
    int argc;
    char **argv;
{
    int sw;
    IMAGE depth_image, add_depth_image, color_image, add_color_image;

    if(argc != 6){
        fprintf(stderr, "Usage : %s <option> <input_depth_image> <output_depth_image>
 <input_color_image> <output_color_image>\n\toption : [-t top] or [-b buttom] or [-r
right] or [-l left]\n\tinput_file : ppm_file\n", argv[0]);
        exit(0);
    }
```



```
    if(argv[1][0] != '-'){
        fprintf(stderr, "Usage : %s <option> <input_depth_image> <output_depth_image>
 <input_color_image> <output_color_image>\n\toption : [-t top] or [-b buttom] or [-r
right] or [-l left]\n\tinput_file : ppm_file\n", argv[0]);
        exit(0);
    }
    if(argv[1][1] == 't')
        sw = TOP;
    else if(argv[1][1] == 'b')
        sw = BUTTOM;
    else if(argv[1][1] == 'r')
        sw = RIGHT;
    else if(argv[1][1] == 'l')
        sw = LEFT;
    else{
        fprintf(stderr, "Usage : %s <option> <input_depth_image> <input_color_image>
<output_depth_image> <output_color_image>\n\toption : [-t top] or [-b buttom] or [-r
right] or [-l left]\n\tinput_file : ppm_file\n", argv[0]);
        exit(0);
    }

    if(readppm(argv[2], &depth_image) < 0){
        fprintf(stderr, "cannot open file - %s\n", argv[2]);
        exit(0);
    }
    if(readppm(argv[3], &color_image) < 0){
        fprintf(stderr, "cannot open file - %s\n", argv[4]);
        exit(0);
    }

    makeAddImage(depth_image, color_image, &add_depth_image, &add_color_image, sw);

    if(writeppm(argv[4], &add_depth_image) < 0){
        fprintf(stderr, "cannot open file - %s\n", argv[3]);
        exit(0);
    }
    if(writeppm(argv[5], &add_color_image) < 0){
        fprintf(stderr, "cannot open file - %s\n", argv[5]);
        exit(0);
    }

    return 0;
}
```

```
/*-------------------------------------------------------
|   5imageViewer.c                                      |
|   1997/10/20                                          |
|   Toshimichi AOKI (NAIST)                             |
 -------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>

#define  ABS(x) ((x > 0) ? x : 0 - x)
#define  NUM 5          /* 入力画像枚数 */
#define  WIDTH  640     /* 入力画像の上限サイズ */
#define  HEIGHT  480
#define  INC 1          /* 解像度変更  (1でそのまま、2で1／2) */
#define  GAP 3.0        /* 低信頼度ポリゴン除去用閾値 */
#define  COLOR 0        /* ppm画像タイプ */
#define  GREY 1

typedef struct depth_data{   /* 奥行きマップ構造体 */
    float x;
    float y;
    float z;
} DEPTH_DATA;
typedef struct color_data{   /* テクスチャマップ構造体 */
    int r;
    int g;
    int b;
} COLOR_DATA;
typedef struct img {
        char type;          /* 画像タイプ */
        unsigned int xsize; /* xサイズ */
        unsigned int ysize; /* yサイズ */
        unsigned int depth; /* depth */
        char *rdata;        /* r成分 */
        char *gdata;        /* g成分 */
        char *bdata;        /* b成分 */
} IMAGE;

DEPTH_DATA depth[NUM][HEIGHT][WIDTH];
COLOR_DATA color[NUM][HEIGHT][WIDTH];

static GLfloat slide_x  = 0.0;   /* 視点移動量 */
static GLfloat slide_y  = 0.0;
static GLfloat slide_z  = 0.0;
static GLint   xy_on  = 0;       /* xy方向移動フラグ */
static GLint   z_on = 0;         /* z方向移動フラグ*/
static GLint   mouse_on = 0;     /* マウス入力フラグ */
static GLint   width    = 500;   /* ウィンドウサイズ */
static GLint   height   = 500;
static GLint   last_x   = 0;     /* マウスのドラッグ開始位置 */
static GLint   last_y   = 0;
static GLint   motionMenu;
static GLint   Point    = 0;     /* 出力形式フラグ */
static GLint   Wire     = 0;
static GLint   Poly     = 1;

/*
   ppg/pgm(raw)形式のファイルをimg構造体に読み込む

   返り値  読み込み成功   0
           失敗          -1
*/
int
readppm(filename, data)
```



```
    char *filename;
    IMAGE *data;
{
        FILE *fp;
        char buf[80];
        unsigned long i=0, size;

        fp = fopen(filename, "rb");
        if(fp==NULL) return -1;

        fgets(buf, 80, fp);
        if(strncmp("P6", buf, 2)==0) data->type = COLOR;
        else if(strncmp("P5", buf, 2)==0) data->type = GREY;
        else {
                puts("unknown file type");
                return -1;
        }
        do{
                fgets(buf, 80, fp);
        }while(buf[0]=='#');
        sscanf(buf, "%d %d", &(data->xsize), &(data->ysize));
        fgets(buf, 80, fp);
        sscanf(buf, "%d", &(data->depth));

        size = data->xsize*data->ysize;
        data->rdata = (char *)malloc( size );
        if(data->type==COLOR){
                data->gdata = (char *)malloc( size );
                data->bdata = (char *)malloc( size );
                while(i<size){
                        data->rdata[i]=fgetc(fp);
                        data->gdata[i]=fgetc(fp);
                        data->bdata[i]=fgetc(fp);
                        i++;
                }
        } else {
                fread(data->rdata, size, 1, fp);
        }
        fclose(fp);
        return 0;
}

/*
  モデルの表示
*/
void
modelDraw(void)
{
    int x, y, a;
    float cal1, cal2, cal3;

/*
  ポイント表示
*/
    if(Point){
        for(a = 0; a < NUM; a++){
            for(y = 0; y < HEIGHT - INC; y += INC){
                for(x = 0; x < WIDTH - INC; x += INC){
                    if(depth[a][y][x].z != 0.0){
                        glBegin(GL_POINTS);
                        glColor3ub(color[a][y][x].r, color[a][y][x].g, color[a][y][x]
.b);
                        glVertex3f(depth[a][y][x].x, depth[a][y][x].y, depth[a][y][x]
.z);
                        glEnd();
                    }
```

```
                }
            }
        }
    }

/*
  ワイヤーフレーム表示
*/
    if(Wire){
        for(a = 0; a < NUM; a++){
            for(y = 0; y < HEIGHT - INC; y += INC){
                for(x = 0; x < WIDTH - INC; x += INC){
                    if(depth[a][y][x].z != 0.0 && depth[a][y+INC][x].z != 0.0 && dept
h[a][y][x+INC].z != 0.0){
                        cal1 = depth[a][y][x].z - depth[a][y+INC][x].z;
                        cal2 = depth[a][y+INC][x].z - depth[a][y][x+INC].z;
                        cal3 = depth[a][y][x+INC].z - depth[a][y][x].z;
                        if(ABS(cal1) < GAP && ABS(cal2) < GAP && ABS(cal3) < GAP){
                            glBegin(GL_LINE_LOOP);
                            glColor3ub(color[a][y][x].r, color[a][y][x].g, color[a][y
][x].b);
                            glVertex3f(depth[a][y][x].x, depth[a][y][x].y, depth[a][y
][x].z);
                            glColor3ub(color[a][y+INC][x].r, color[a][y+INC][x].g, co
lor[a][y+INC][x].b);
                            glVertex3f(depth[a][y+INC][x].x, depth[a][y+INC][x].y, de
pth[a][y+INC][x].z);
                            glColor3ub(color[a][y][x+INC].r, color[a][y][x+INC].g, co
lor[a][y][x+INC].b);
                            glVertex3f(depth[a][y][x+INC].x, depth[a][y][x+INC].y, de
pth[a][y][x+INC].z);
                            glEnd();
                        }
                        cal1 = depth[a][y+INC][x+INC].z - depth[a][y+INC][x].z;
                        cal2 = depth[a][y+INC][x].z - depth[a][y][x+INC].z;
                        cal3 = depth[a][y][x+INC].z - depth[a][y+INC][x+INC].z;
                        if(ABS(cal1) < GAP && ABS(cal2) < GAP && ABS(cal3) < GAP){
                            glBegin(GL_LINE_LOOP);
                            glColor3ub(color[a][y+INC][x+INC].r, color[a][y+INC][x+IN
C].g, color[a][y+INC][x+INC].b);
                            glVertex3f(depth[a][y+INC][x+INC].x, depth[a][y+INC][x+IN
C].y, depth[a][y+INC][x+INC].z);
                            glColor3ub(color[a][y][x+INC].r, color[a][y][x+INC].g, co
lor[a][y][x+INC].b);
                            glVertex3f(depth[a][y][x+INC].x, depth[a][y][x+INC].y, de
pth[a][y][x+INC].z);
                            glColor3ub(color[a][y+INC][x].r, color[a][y+INC][x].g, co
lor[a][y+INC][x].b);
                            glVertex3f(depth[a][y+INC][x].x, depth[a][y+INC][x].y, de
pth[a][y+INC][x].z);
                            glEnd();
                        }
                    }
                }
            }
        }
    }

/*
  ポリゴン表示
*/
    if(Poly){
        for(a = 0; a < NUM; a++){
            for(y = 0; y < HEIGHT - INC; y += INC){
                for(x = 0; x < WIDTH - INC; x += INC){
                    if(depth[a][y][x].z != 0.0 && depth[a][y+INC][x].z != 0.0 && dept
```



```
h[a][y][x+INC].z != 0.0){
                            cal1 = depth[a][y][x].z - depth[a][y+INC][x].z;
                            cal2 = depth[a][y+INC][x].z - depth[a][y][x+INC].z;
                            cal3 = depth[a][y][x+INC].z - depth[a][y][x].z;
                            if(ABS(cal1) < GAP && ABS(cal2) < GAP && ABS(cal3) < GAP){
                                glBegin(GL_POLYGON);
                                glColor3ub(color[a][y][x].r, color[a][y][x].g, color[a][y
][x].b);
                                glVertex3f(depth[a][y][x].x, depth[a][y][x].y, depth[a][y
][x].z);
                                glColor3ub(color[a][y+INC][x].r, color[a][y+INC][x].g, co
lor[a][y+INC][x].b);
                                glVertex3f(depth[a][y+INC][x].x, depth[a][y+INC][x].y, de
pth[a][y+INC][x].z);
                                glColor3ub(color[a][y][x+INC].r, color[a][y][x+INC].g, co
lor[a][y][x+INC].b);
                                glVertex3f(depth[a][y][x+INC].x, depth[a][y][x+INC].y, de
pth[a][y][x+INC].z);
                                glEnd();
                            }
                            cal1 = depth[a][y+INC][x+INC].z - depth[a][y+INC][x].z;
                            cal2 = depth[a][y+INC][x].z - depth[a][y][x+INC].z;
                            cal3 = depth[a][y][x+INC].z - depth[a][y+INC][x+INC].z;
                            if(ABS(cal1) < GAP && ABS(cal2) < GAP && ABS(cal3) < GAP){
                                glBegin(GL_POLYGON);
                                glColor3ub(color[a][y+INC][x+INC].r, color[a][y+INC][x+IN
C].g, color[a][y+INC][x+INC].b);
                                glVertex3f(depth[a][y+INC][x+INC].x, depth[a][y+INC][x+IN
C].y, depth[a][y+INC][x+INC].z);
                                glColor3ub(color[a][y][x+INC].r, color[a][y][x+INC].g, co
lor[a][y][x+INC].b);
                                glVertex3f(depth[a][y][x+INC].x, depth[a][y][x+INC].y, de
pth[a][y][x+INC].z);
                                glColor3ub(color[a][y+INC][x].r, color[a][y+INC][x].g, co
lor[a][y+INC][x].b);
                                glVertex3f(depth[a][y+INC][x].x, depth[a][y+INC][x].y, de
pth[a][y+INC][x].z);
                                glEnd();
                            }
                        }
                    }
                }
            }
        }
    }
}

/*
  画面の表示
*/
void
modelDisplay(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);     /* Clear Window */
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(slide_x * 0.01, slide_y * 0.01, 0.0 + slide_z * 0.01, slide_x * 0.01, s
lide_y * 0.01, 1.0 + slide_z * 0.01, 0.0, -1.0, 0.0);
    glPushMatrix();
    modelDraw();
    glPopMatrix();
    glFlush();
    glutSwapBuffers();      /* Swap drawbuffer and viewbuffer */
}

/*
  視点の移動
```

```
*/
void
modelMotion(int x, int y)
{
    if(mouse_on){
        if(xy_on){
            if((last_x != x || last_y != y)){

                slide_x = slide_x + ((float)x - (float)last_x);
                slide_y = slide_y + ((float)y - (float)last_y);
                glutPostRedisplay();
                last_x = x;
                last_y = y;
            }
        }

        if(z_on){
            if(last_y != y){
                slide_z = slide_z + ((float)y - (float)last_y);
                glutPostRedisplay();
                last_x = x;
                last_y = y;
            }
        }
    }
}

/*
  マウスのイベント
*/
void
mouse(int button, int state, int x, int y)
{
    if(button == GLUT_LEFT_BUTTON){
        if(state == GLUT_DOWN){
            xy_on = 1;
            mouse_on = 1;
            last_x = x;
            last_y = y;
        }
        else if(state == GLUT_UP){
            xy_on = 0;
            mouse_on = 0;
        }
    }
    else if(button == GLUT_MIDDLE_BUTTON){
        if(state == GLUT_DOWN){
            z_on = 1;
            mouse_on = 1;
            last_x = x;
            last_y = y;
        }
        else if(state == GLUT_UP){
            z_on = 0;
            mouse_on = 0;
        }
    }
}

/*
  ウィンドウのサイズ変更時の処理
*/
void
reshapeWindow(int x, int y)
{
    glViewport(0, 0, x, y);
```



```
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(40.0, (GLfloat)x / (GLfloat)y, 0.1, 80.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, -1.0, 0.0);

    width = x;
    height = y;
}

/*
  三次元情報の復元
*/
void
makeData(depth_image, color_image, a)
    IMAGE depth_image;
    IMAGE color_image;
    int a;
{
  int x, y;

  for(y = 0; y < depth_image.ysize; y++){
      for(x = 0; x < depth_image.xsize; x++){
          if(depth_image.rdata[y * depth_image.xsize + x] != 0){
              depth[a][y][x].z = 50.0 * 6.0 / (float)depth_image.rdata[y * depth_imag
e.xsize + x];
          depth[a][y][x].x = ((float)x - (float)depth_image.xsize / 2.0) * 0.01 / 6.0
 * depth[a][y][x].z;
          depth[a][y][x].y = ((float)y - (float)depth_image.ysize / 2.0) * 0.01 / 6.0
 * depth[a][y][x].z;
          }
          else
              depth[a][y][x].z = 0.0;
      }
  }

  for(y = 0; y < color_image.ysize; y++){
      for(x = 0; x < color_image.xsize; x++){
          color[a][y][x].r = color_image.rdata[y * color_image.xsize + x];
          color[a][y][x].g = color_image.gdata[y * color_image.xsize + x];
          color[a][y][x].b = color_image.bdata[y * color_image.xsize + x];
      }
  }
}

/*
  メニューの選択処理
*/
void
menu(int value)
{
    if(value < 5){
        switch(value){
        case 1:
            Point = 1;
            Wire = 0;
            Poly = 0;
            glutPostRedisplay();
            break;
        case 2:
            Point = 0;
            Wire = 1;
            Poly = 0;
```

```
            glutPostRedisplay();
            break;
        case 3:
            Point = 0;
            Wire = 0;
            Poly = 1;
            glutPostRedisplay();
            break;
        case 4:
            exit(0);
            break;
        }
    }
}

/*
  右ボタンメニューの作成
*/
void
makeMenu(void)
{
    motionMenu = glutCreateMenu(menu);
    glutAddMenuEntry("Point", 1);
    glutAddMenuEntry("Wire", 2);
    glutAddMenuEntry("Poly", 3);

    glutCreateMenu(menu);
    glutAddSubMenu("View...", motionMenu);
    glutAddMenuEntry("Quit", 4);

    glutAttachMenu(GLUT_RIGHT_BUTTON);
}

/*
  メイン関数
*/
int
main(int argc, char **argv)
{
    int i;
    IMAGE depth_image1, color_image1, depth_image2, color_image2;
    IMAGE depth_image3, color_image3, depth_image4, color_image4;
    IMAGE depth_image5, color_image5;

    if(argc != 11){
        fprintf(stderr, "Usage : %s", argv[0]);
        for(i = 0; i < NUM; i++){
            fprintf(stderr, " <depth_image> <color_image>");
        }
        fprintf(stderr, "\n");
        exit(0);
    }
    if(readppm(argv[1], &depth_image1) < 0){
        fprintf(stderr, "cannot open file - %s\n", argv[1]);
        exit(0);
    }
    if(readppm(argv[2], &color_image1) < 0){
        fprintf(stderr, "cannot open file - %s\n", argv[2]);
        exit(0);
    }
    if(readppm(argv[3], &depth_image2) < 0){
        fprintf(stderr, "cannot open file - %s\n", argv[3]);
        exit(0);
    }
    if(readppm(argv[4], &color_image2) < 0){
        fprintf(stderr, "cannot open file - %s\n", argv[4]);
```



```
        exit(0);
    }
    if(readppm(argv[5], &depth_image3) < 0){
        fprintf(stderr, "cannot open file - %s\n", argv[5]);
        exit(0);
    }
    if(readppm(argv[6], &color_image3) < 0){
        fprintf(stderr, "cannot open file - %s\n", argv[6]);
        exit(0);
    }
    if(readppm(argv[7], &depth_image4) < 0){
        fprintf(stderr, "cannot open file - %s\n", argv[7]);
        exit(0);
    }
    if(readppm(argv[8], &color_image4) < 0){
        fprintf(stderr, "cannot open file - %s\n", argv[8]);
        exit(0);
    }
    if(readppm(argv[9], &depth_image5) < 0){
        fprintf(stderr, "cannot open file - %s\n", argv[9]);
        exit(0);
    }
    if(readppm(argv[10], &color_image5) < 0){
        fprintf(stderr, "cannot open file - %s\n", argv[10]);
        exit(0);
    }

    makeData(depth_image1, color_image1, 0);
    makeData(depth_image2, color_image2, 1);
    makeData(depth_image3, color_image3, 2);
    makeData(depth_image4, color_image4, 3);
    makeData(depth_image5, color_image5, 4);

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(width, height);
    glutCreateWindow(argv[0]);

    glClearColor(0.0, 0.0, 0.0, 0.0);

    glEnable(GL_DEPTH_TEST);
    glCullFace(GL_BACK);
    glEnable(GL_CULL_FACE);

    makeMenu();

    glutReshapeFunc(reshapeWindow);
    glutDisplayFunc(modelDisplay);
    glutMouseFunc(mouse);
    glutMotionFunc(modelMotion);
    glutMainLoop();

    return 0;
}
```