〔非公開〕

ＴＲ－Ｍ－００１６

多視点画像と奥行きマップを用いた任意視点映像生成についての検討

松　井　　明
Akira　MATSUI

朴　　鐘　一　　井　上　　誠　喜
Jong-il　Park　　Seiki　INOUE

１９９７．３．１０

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
任意視点映像生成についての検討

Arbitrary view generation from multiple-stereo image and its depth map.

1997年3月10日

概要

本報告書では，任意視点映像生成法についての提案し，本手法についての検討を行う．十字に配置されたカメラから得られる５つの画像と，それらからステレオマッチングによって得られる奥行きマップを用いて，任意視点映像の生成を行った．特に奥行きマップの変換における２つの手法について比較検討を行い、２つの手法の利点、欠点を明かにした。

ATR知能映像通信研究所 第3研究室

学外実習生

松井 明[1]

[1]奈良先端科学技術大学院大学,映像情報処理学講座

1　はじめに

本研究の目的は、数枚の実画像より任意視点映像を生成することである．実画像からの任意視点映像生成の手法として，画像のみを用いる方法と計測によって得られた奥行き情報を用いる方法が考えられる．本研究では後者の方法を選択している．撮影は十字に配置された５台のカメラで行い５枚の画像を得る．そしてその５枚の画像からステレオマッチングによって得られる奥行きマップを利用して任意の視点からの画像を生成する．

2　奥行き情報の取得

５台のカメラを十文字に配置してステレオマッチングによって，各ピクセルごとの奥行き情報を取得し，奥行きマップを生成する。詳細は文献 [1] を参照のこと．

3　任意視点映像の生成

3.1　提案手法の概念

図 1に本手法の概念をしめす．実際のカメラ以外の位置に仮想カメラを想定し，そのカメラから得られる画像を生成する．

奥行きマップを利用して新しい視点からの映像を生成するためには、大きく分けて２段階の処理が必要である。

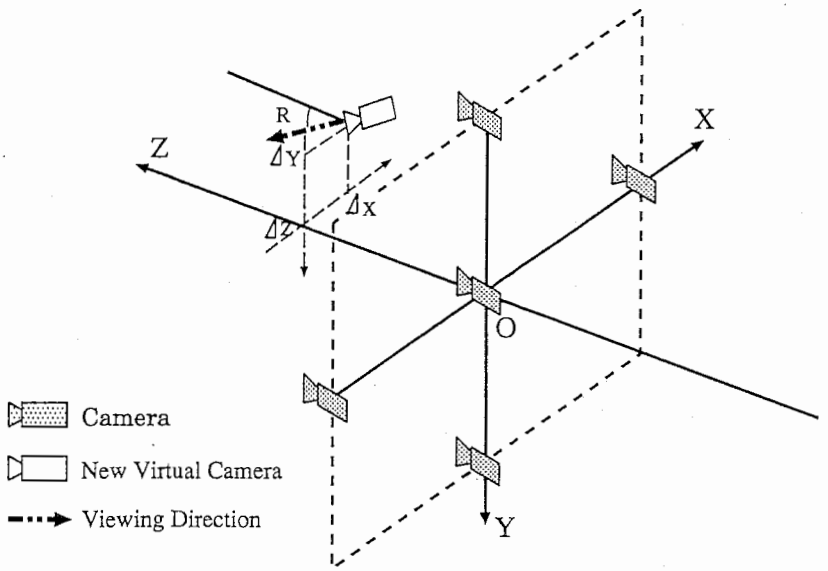


図 1: 任意視点映像生成の概念.

第1段階は視点（カメラ）の移動への対応である。視線方向（画角，仰角等）は変化しない．つまり仮想カメラを3次元空間中で平行移動させることに相当する．この処理は，奥行きが重要なパラメータになるため3次元処理に属する．まず，投影面上の座標変換によって奥行きマップを新しい視点のものに変換する．つづいて，5台のカメラの中から適当な映像を選択してテクスチャマッピングを行なう．この際、近くの物体の陰に隠れて見えなくなる領域と逆に新たに生ずる領域が存在する。前者はカメラから最も近くにある物体のテクスチャを用いればよいため特に問題はないが、後者は未知のデータを推測する必要がある。

第2段階は、目的に合わせて、視線方向、画角などを自由に調整するものである。奥行きによらないため、2次元処理だけで行うことができる．ここからは、第1段階を視点移動、第2段階を視線方向の変更と称する。

3.2 視点の移動

図2に示すように3次元カメラ座標系を定義する．十字に配置されたカメラのの中心のカメラの位置を3次元カメラ座標系の原点 O、水平方向を x 軸，垂直方向を y 軸，そして中心カメラの光軸を Z 軸とする．さらにカメラの焦点距離を F とする。

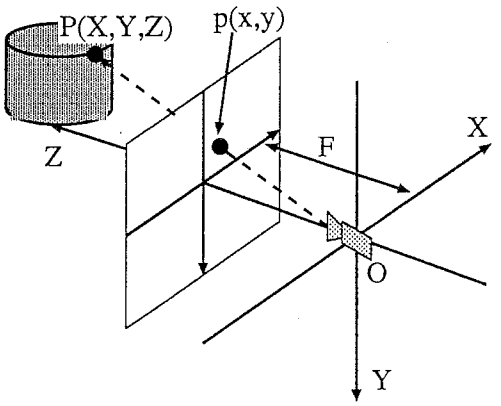


図 2: 3次元空間中の座標の画像平面への投影.

このように定義すると3次元空間中の点 $P = (X, Y, Z)$ は画像平面 $p = (x, y)$ に式 (1) のように投影される。

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

図3のように視点を任意の位置 $(\Delta X, \Delta Y, \Delta Z)$ に移動することを考える．3次元空間中の点 $P(X, Y, Z)$ を中心カメラにおける画像面に投影した点を $p = (x, y)$，仮想カメラにおいて画像

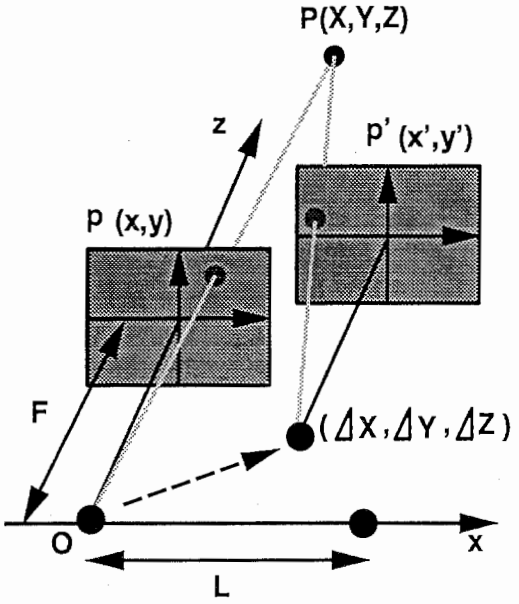


図 3: 画像面の座標の変換.

面上に投影した点を $p' = (x', y')$ と置く. 式 (1) と同様に仮想カメラにおける画像面への投影は

$$x' = F\frac{X - \Delta X}{Z - \Delta Z}, \quad y' = F\frac{Y - \Delta Y}{Z - \Delta Z} \tag{2}$$

に対応する。次に, 中心カメラと周りのカメラとの間の距離を L , 3 次元空間中のある点 ($P(X, Y, Z$ を中心のカメラと仮想カメラに投影したときの視差を d とすると $Z = FL/d$ の関係が成り立つため、それを式 (2) に代入すると。p と p' との関係は式 (3)(4) のように表される.

$$x' = \frac{FL}{FL - d\Delta Z}\left(x - \frac{d\Delta X}{L}\right) \tag{3}$$

$$y' = \frac{FL}{FL - d\Delta Z}\left(y - \frac{d\Delta Y}{L}\right) \tag{4}$$

となる。この式 (3)(4) により、中心のカメラで得られる画像面の座標 p を、任意の仮想カメラの画像面の座標 p' に変換することができる。視点を移動すると、隠される領域、新たに現れる領域 (図 6) が生じるため、その領域の処理に注意しなければならない。

本手法では、まず式 (3)(4) を用いて奥行きマップを変換し、それを利用してテクスチャーをマッピングする手法を用いる。

3.2.1 仮想視点における奥行きマップの生成

式(3)(4)おいて、dがx,yの関数であるため[2]、逆変換が解析的には求められない。つまり、中心カメラの画像面の座標pが任意視点における画像平面上の座標p'に対応することは正確にわかるが，任意視点における座標p'から座標pを求めることはできないということである．これは、反復的手法を用いて求めることも考えられるが、本稿では計算量を考慮し、pからp'に変換し，そのp'にpが示す奥行き値を適当なピクセルに割り当てることで奥行きマップの変換を行なう．このとき同時にテクスチャーのマッピングも行う．この割当の手法としてはいろいろ考えられるが，最近傍点に割り当てる方法と複数の変換点からの線形補間により割り当てる方法の2通りの方法を考える．[3]

最近傍点を割り当てる方法

図4のように式(3)(4)を用いて画像面上の座標変換を行ない，最も近いピクセルに元の奥行き値を割り当てる.

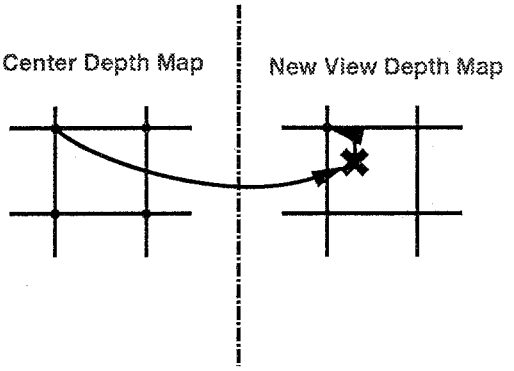


図 4: 最近傍点を割り当てによる奥行きマップの変換.

[2] $p'(x',y')$ における d(奥行き) の値未知である.

[3] 結果のところで示すが，最近傍割り当てを用いた手法では，問題点が生じたため線形補間による方法を利用することを考えた.

線形補間による方法

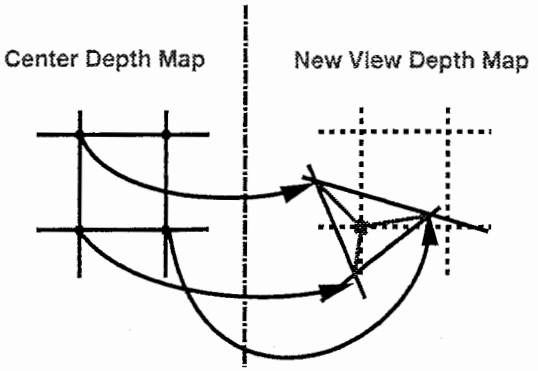


図 5: 線形補間法による奥行きマップの変換.

3画素を用いた線形補間の場合について示す. 図5のように, 中心カメラの画像面中の隣接する3画素を任意視点における画像面に変換した後, その三角形の内部に含まれる画素を3点の奥行き値から線形補間することで新しい奥行きマップを得る. このときに、変換後の三角形の面積の大きさにしきい値を設けて、しきい値より大きい場合は、線形補間を行わない。なぜならば、奥行きが急激に変化する奥行きマップのエッジ部分では、三角形が大きく歪み、ノイズ等の原因となるからである。また、このしきい値を設けることで、テクスチャーマッピングの際に重要な情報を持つ新たに生じる領域を正確に抽出する狙いもある。

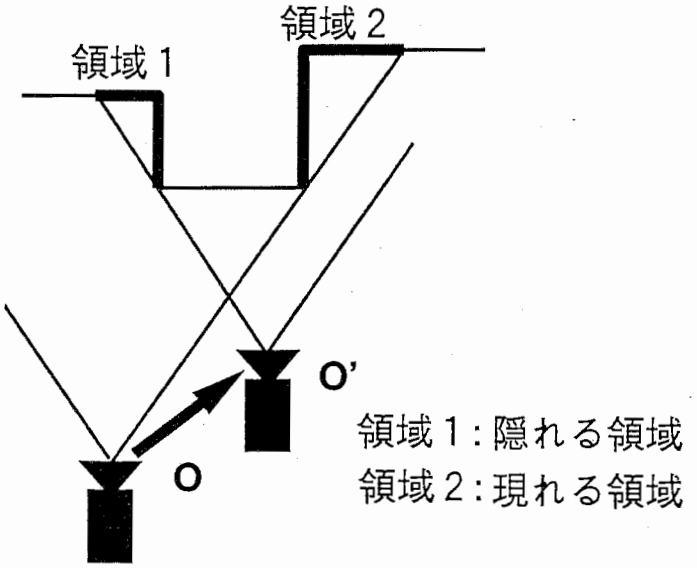


図 6: 視点移動により隠れる領域と新たに現れる領域.

また, 図6に示すように視点の移動にともなって新たに見えてくる領域（図6, 領域2）と

隠されて見えなくなる領域（図6，領域1）が存在する．まず一つ目に，隠される領域の処理は非常に簡単な処理によって解決できる。式(3)(4)の関係に基づいて、奥行きマップを仮想視点からのものに変換する際、隠される領域の変換後の位置には手前からのものが重なる。つまり，同じ画素に複数の点が割り当てられる．常識的に、奥行きの手前のものを選ぶことにより解決できる。

もう一つは新たに生じる領域の処理問題である。視点移動方向に向いた奥行きの不連続部では、視点移動に伴って移動前には見えていなかった領域が現れてくる（図6，領域1）。よって，初期データからこの領域の奥行き値を求めることは不可能である．奥行きマップの観点からすると、奥行き値の分からない領域が生じることになる。従って、補間をしなければならない。

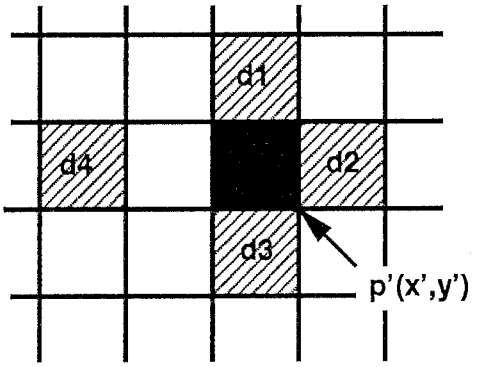


図 7: 視点移動により新たに現れる領域の補間.

いろいろな補間手法が考えられるが、本稿では、線形的に補間する。例えば，図7のように，$p'(x', y')$ が未知領域であり奥行きを補間することを考える．まず，上下左右の方向を探索し奥行き値が求められている画素を見つける（図の d_1, d_2, d_3, d_4）．その奥行き値とその画素までの距離（画素数：$l_1, \cdots, l_4$ とする）により，式 (5) のように未知の奥行き値 $d_{p'}$ を補間する．

$$d_{p'} = \sum_{i=1}^{4} \left(\frac{\frac{1}{l_i}}{\sum_{i=1}^{4} \frac{1}{l_i}} \times d_i \right) \tag{5}$$

これまでに説明をした手法によって求めた奥行きマップは新しい視点での本当の奥行きマップではない。従って、すべての奥行き値に $-\Delta Z$ の補正を加える。

3.2.2 テクスチャーマッピング

中心カメラから見える領域に対しては、前述したように奥行きマップの変換の際、同時にテキスチャーマッピングすることができる。しかし、新たに生じる領域に対しては，補間処理をしなければならない。本稿では、視点の移動方向に合わせてテキスチャーマッピングの対象映

像を選ぶことにする。例えば、視点を右に移動すると物体の右の部分が見えてくるため、右のカメラから撮った映像にはその領域の情報が含まれている可能性が高い。他の方向にも同じことがいえる従って，視点の移動方向によって場合分けし、移動方向に合わせて上下左右の映像からテクスチャーを持ってくるようにする。

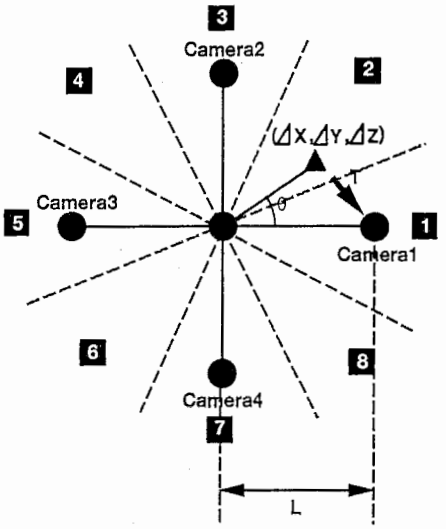


図 8: テクスチャーマッピングのためのカメラ平面の分割.

具体的には，図 8に示すようにカメラ平面を 8 つの領域に分割する．つまり，Z 軸方向への移動は考慮せずX軸，Y軸方向の移動分に対して領域分割を行なう．領域 1，3，5，7 は 6 0 度の幅，領域 2，4，6，8 は 3 0 度の幅を持つように分割している．そして，それぞれの領域に対して表 1 のようにテクスチャーを持ってくるための画像を選択する．例えば，領域 1 の方向に視点を移動した場合は，右側のカメラである camera1 から，領域 2 に移動した場合は camera1 と camera2 の両方から未知領域のテクスチャーを得る．

この分割を利用してテクスチャーマッピングを行なう方法を以下に示す.

1. 図 8の三角の点 $((\Delta X, \Delta Y, \Delta Z))$ に視点を移動したとする.

2. 図 8の分割と表 1 の対応により，テクスチャーの取得に利用する画像を選択する．（この場合は camera1 と camera2)

3. 移動した視点における奥行きマップを用いて，図 9 のように式 (3)(4) の変換によって，画像面（まずは camera1）への変換を行なう．つまり，$(\Delta X, \Delta Y, \Delta Z)$ における画像面の未知ピクセル $p'(x', y')$ から $T(L-\Delta X, L-\Delta Y, -\Delta Z)$ への変換を行なう $((p_1(x_1, y_1))$とする.

4. $(p_1(x_1, y_1))$ の画素値を c_{p_1} とする.

表 1: 視点の移動方向と利用するカメラの対応.

領域	テクスチャーを利用する画像
領域 1	camera1
領域 3	camera2
領域 5	camera3
領域 7	camera4
領域 2	camera1 & camera2
領域 4	camera2 & camera3
領域 6	camera3 & camera4
領域 7	camera4 & camera1

5. 同様に camera2 への変換を求め，その画素値を c_{p_2} とする.

6. c_{p_1} と c_{p_2} を $(\Delta X, \Delta Y, \Delta Z)$ と camera1 及び camera2 までの距離をキーとして線形的に加えることにより，$(\Delta X, \Delta Y, \Delta Z)$ における画像面の未知ピクセル $p'(x', y')$ の画素値を得る.

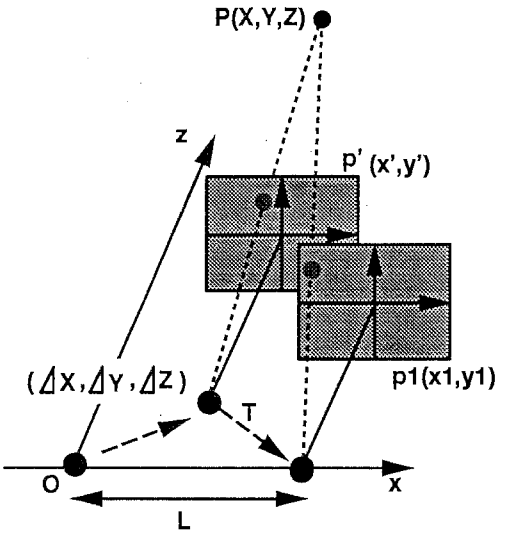


図 9: 周辺画像からのテクスチャーの取得.

また，1つの画像からテクスチャを得る場合は，上記のステップ 4 まで行ないその値を未知ピクセルの値として用いれば良い.

以上により，仮想視点における奥行きマップとカラー画像を取得することができる.

3.3 視線方向の変更と拡大縮小

前節で説明した視点の移動に加えて，視線方向の変更（仮想カメラの回転），画像の拡大縮小の処理（焦点距離の調節）を考える．視線方向の変更と映像の拡大縮小は2次元幾何変換によって実現できる [3]。

3.3.1 視線方向の変更

視線方向の変更は図10のような3次元空間座標のX軸，Y軸，Z軸のそれぞれを中心とする回転と考えられる．

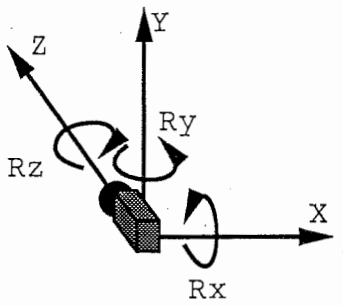


図 10: 視線方向の変更.

この各軸の回転後の座標 (X', Y', Z') は回転行列 R を用いて

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{6}$$

のように表される．ここで，X軸回りの回転角を α,Y軸回りを β,Z軸回りを γ とすると，回転行列 R は

$$\begin{aligned} R &= R_\alpha R_\beta R_\gamma \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \end{aligned} \tag{7}$$

となる．ここで，座標変換前の画像面を (x, y)，変換後の画像投影面を (x', y') とする．この変換後の座標 (X', Y', Z') においてカメラへの投影を考えると，

$$x' = F\frac{X'}{Z'} \quad , \quad y' = F\frac{Y'}{Z'} \tag{8}$$

が成り立つ．ゆえに，式(6),(7),(8) より，

$$x' = F\frac{r_{11}X + r12Y + r13Z}{r_{31}X + r32Y + r33Z} \tag{9}$$

$$y' = F\frac{r_{21}X + r22Y + r23Z}{r_{31}X + r32Y + r33Z} \tag{10}$$

の関係が成り立つ．そして，式(9),(10) の X, Y, Z をそれぞれ，x, y, F に置き換えることによって[4]を，ある画像に対して，X 軸 α,Y 軸 β,Z 軸 γ のようにカメラを回転させた画像を得ることができる．

3.3.2 画像の拡大縮小（焦点距離の変更）

画像を単に拡大縮小することは，焦点距離を短くする，長くすることと同義である．これは式(9),(10) の係数の F（焦点距離）を F' に変更することで実現できる．実際に焦点距離を指定することは，感覚的にわかりにくいことなのでズーム量 $(f = F'/F)$ を定義し，式(9),(10) に代入すると

$$x' = fF\frac{r_{11}X + r12Y + r13Z}{r_{31}X + r32Y + r33Z} \tag{11}$$

$$y' = fF\frac{r_{21}X + r22Y + r23Z}{r_{31}X + r32Y + r33Z} \tag{12}$$

のようになり，同時に視線方向の変更と拡大縮小が実現できる．

視線方向の変化量とズーム量は、画面の境界部が現れないように、また、出来上がった映像の品質が目的とする画質に符合するようにするため、ある範囲に制限される。

[4]この置き換えによって，3 次元空間中のの物体の座標 (X, Y, Z) を画像面の各ピクセルの 3 次元空間座標として置き換えたことになる．(x, y) は (X, Y, Z) が投影されたピクセルのために (X,Y,Z) の物体色と同じ色情報を持つ．

4　実験結果

4.1　実験データ

実験に使用した画像データを図11に示す．それぞれの画像サイズは 288×384 である。この画像に対する奥行きマップを図12に示す．この奥行きマップは実際に計測したものではなく，計測によって得られた視差データを手でマッピングした合成データである．また，実際の視差の値は０～１５の範囲であるが，見やすくするために０～２５５の値に変換している．

この実験データに対する実行結果を，カメラ平面内の移動，カメラ平面内の移動と光軸方向への移動（近付く場合，遠ざかる場合），の３つの場合に分けて示す．線形補間による割り当てでは，３点による線形補間を用いている．

図 11: 実験用データ．

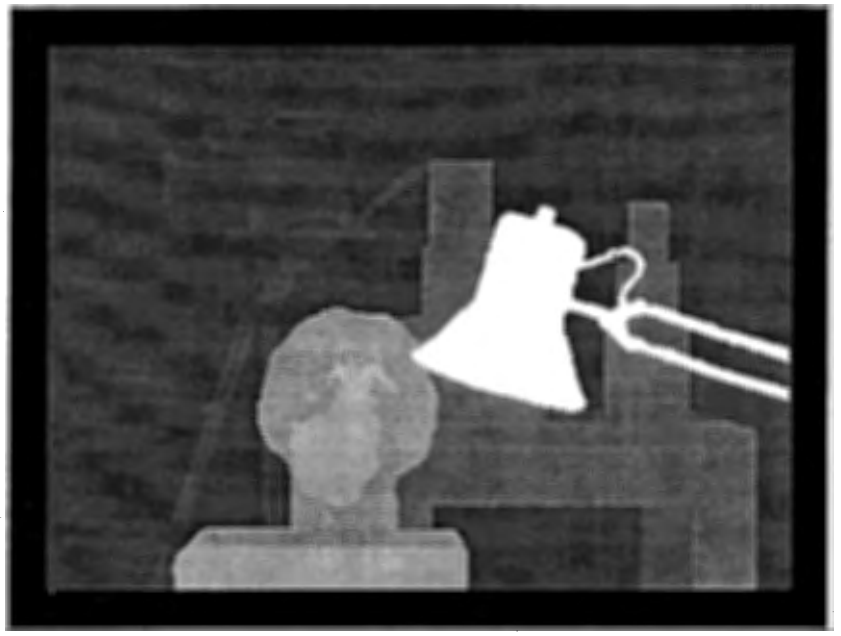
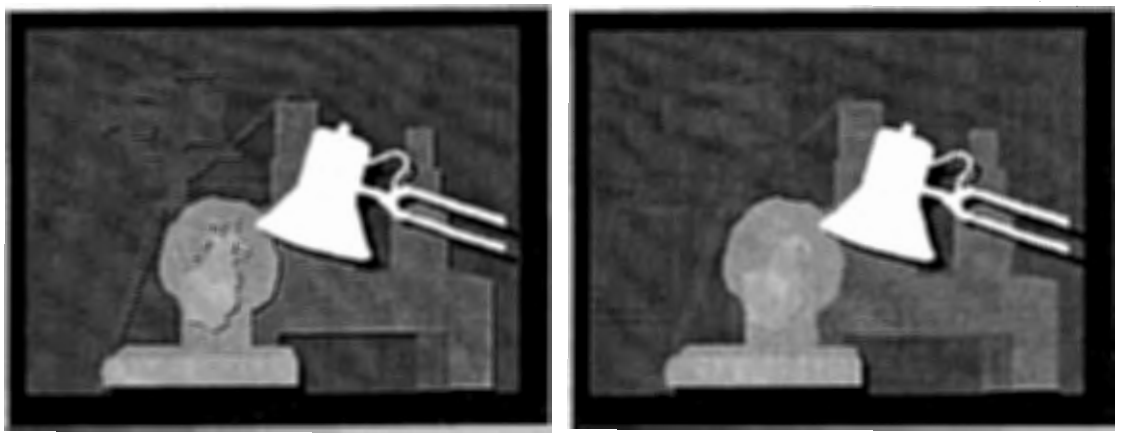


図 12: 実験用データ — 奥行きマップ —.

4.2 視点移動に関する結果

4.2.1 カメラ平面内の視点移動

カメラ平面内の移動（カメラの光軸方向には変化しない．）の結果を示す．まず，式(3)(4)を利用して奥行きマップを変換した結果を図13に示す．左の画像が最近傍点への割り当てによるもの，右の画像が線形補間による割り当てを用いたものである．また，図13の欠落した領域を補間した結果を図14に示す．欠落した領域というのは，視点の移動により新たに生ずる領域のことであり，物体の右側のエッジ付近に現れている黒い領域である．



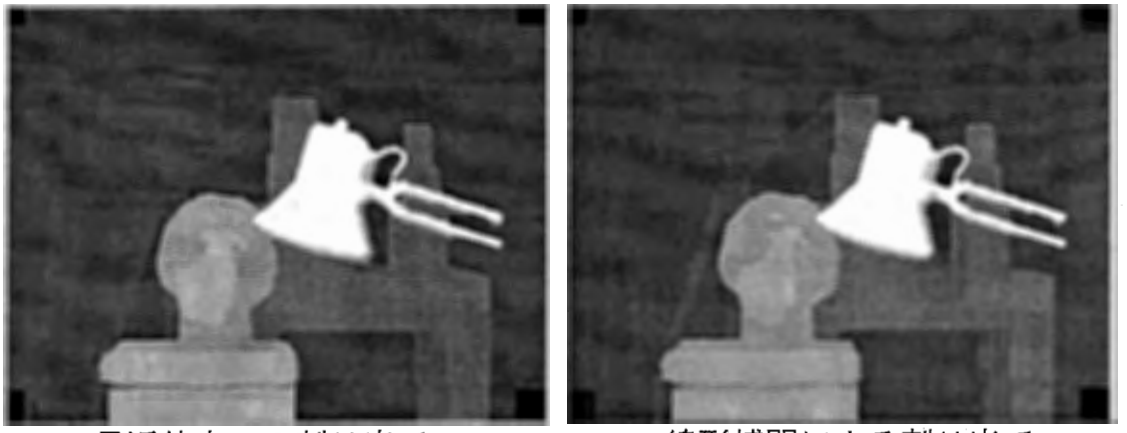
最近傍点への割り当て　　線形補間による割り当て

図 13: 実験結果 $(\Delta X, \Delta Y, \Delta Z) = (1L, 1L, 0L)$．— 欠落領域の補間前 —

図13を見ると，線形補間を用いたものと比べて最近傍点を割り当てたものの方が黒い領域（新たに生じた領域）が多いことが分かる．この新たに生じる領域は，後のテクスチャーマッピング処理で重要となってくるため正確に抽出できる必要がある．つまり，周りの画像からテクスチャを持ってくるために必要である．線形補間を用いた方法では、変換後の3つの点が張る三角形の面積にしきい値を設けてこのような領域を正確に抽出しようとしている。その結果として照明の右の領域が抽出されている。[5]しかし、この観点から言うと，最近傍の割り当てを用いた方が良いということになる．

この奥行きマップを利用して，テクスチャーマッピングを行うと，図15のような任意視点画像が得られる．奥行きマップの観点からいくと，最近傍割り当ての方が良い結果を示していたが，実際に再構成された任意視点映像（図15）を見ると，結果には大差は見られない。

[5]しきい値を設けない場合は、ほぼすべての領域が補間されて、新しく生じる領域は抽出できない。



最近傍点への割り当て　　線形補間による割り当て

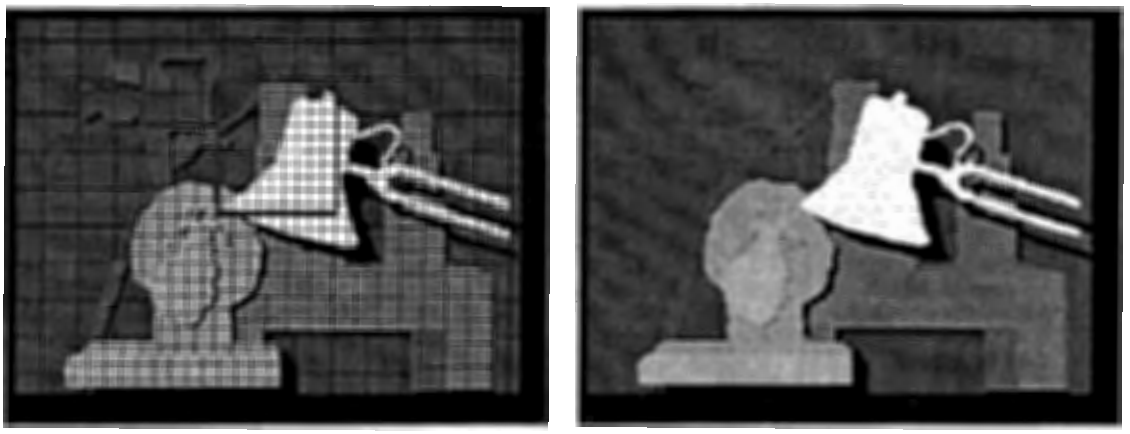
図 14: 実験結果 $(\Delta X, \Delta Y, \Delta Z) = (1L, 1L, 0L)$ ． — 欠落領域の補間後 —

最近傍点への割り当て　　線形補間による割り当て

図 15: 実験結果 $(\Delta X, \Delta Y, \Delta Z) = (1L, 1L, 0L)$ ．

4.2.2 対象に近づく方向に移動

4.2.1節の移動に加えて，z軸の正方向への移動（対象に近づく）を加えた場合の結果を示す．まず，式(3)(4)を利用して奥行きマップを変換した結果を図16に示す．左の画像が最近傍点への割り当てによるもの，右の画像が線形補間による割り当てを用いたものである．また，図16の欠落した領域を補間した結果を図18に示す．



最近傍点への割り当て　　線形補間による割り当て

図 16: 実験結果 $(\Delta X, \Delta Y, \Delta Z) = (1L, 1L, 2L)$．— 欠落領域の補間前 —

この，最近傍割り当てによる方法の場合は，網目状に欠落したノイズが見られる．これは，最近傍割り当てでは元の奥行きマップと新しい視点での奥行きマップのピクセルが1対1に対応するために生じる．なぜなら，この結果のように対象に近づく場合，実際には1対多の対応をしなければならないからである．もともと，複数点の線形補間による奥行きマップの変換を考えたのはこの問題を解決するためである．

図16で”0”の値となっている網目状のノイズは，補間によって図18のようになるので後の処理に問題はない．しかし，図17のように隠れた領域の奥行き値が網目に割り当てられている場合は欠落領域の補間の対象とはならないため図18の補間後の結果にノイズが残る．図18に横方向のノイズしか残っていないのは，図16の結果に対してメディアンフィルタをかけているためである．

複数の点の線形補間による方法はこの問題を解決している．図19を見れば明らかなように，右の線形補間を用いた方法が良い結果を示している．

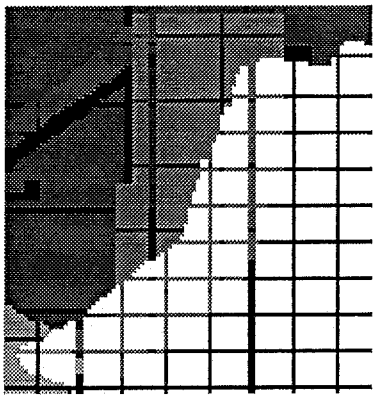
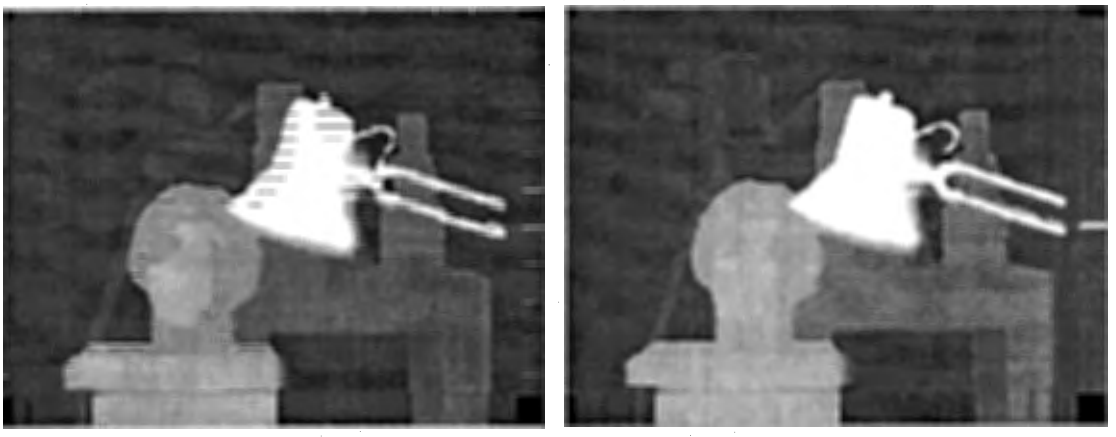
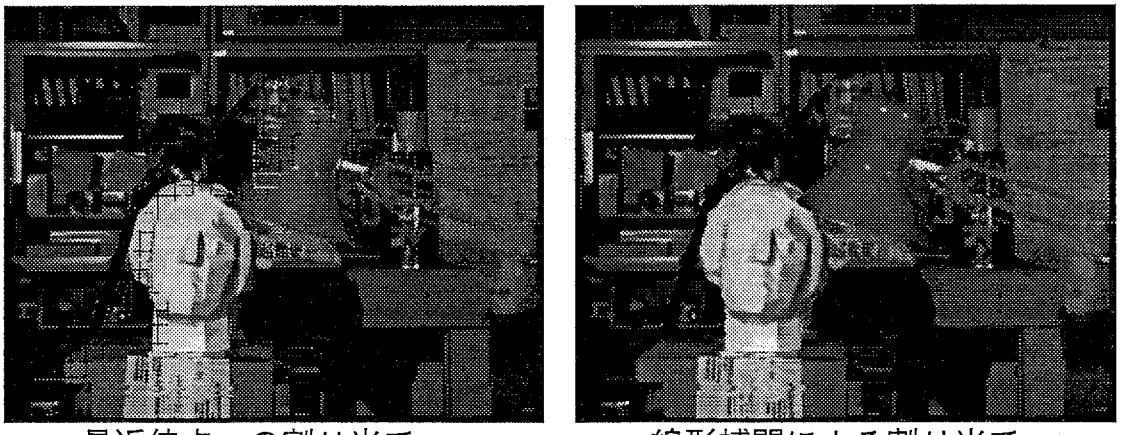


図 17: 実験結果 $(\Delta X, \Delta Y, \Delta Z) = (1L, 1L, 2L)$ ． における最近傍割り当ての問題点



最近傍点への割り当て　　線形補間による割り当て

図 18: 実験結果 $(\Delta X, \Delta Y, \Delta Z) = (1L, 1L, 2L)$ ． — 欠落領域の補間後 —

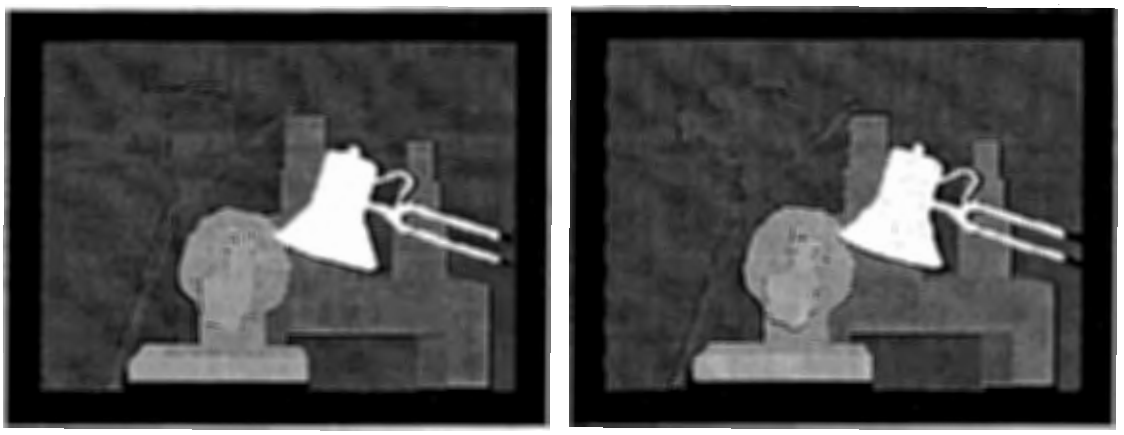


最近傍点への割り当て　　線形補間による割り当て

図 19: 実験結果 $(\Delta X, \Delta Y, \Delta Z) = (1L, 1L, 2L)$ ．

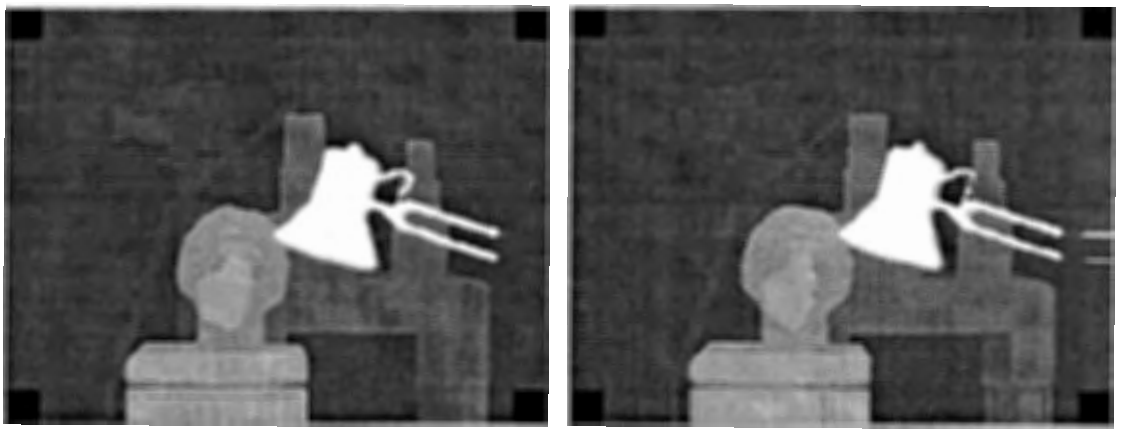
4.2.3 対象から遠ざかる方向に移動

4.2.1節の移動に加えて，z軸の負方向への移動（対象から遠ざかる）を加えた場合の結果を示す．まず，式(3)(4)を利用して奥行きマップを変換した結果を図20に示す．左の画像が最近傍点への割り当てによるもの，右の画像が線形補間による割り当てを用いたものである．また，図20の欠落した領域を補間した結果を図21に示す．



最近傍点への割り当て　　線形補間による割り当て

図 20: 実験結果 $(\Delta X, \Delta Y, \Delta Z) = (1L, 1L, -2L)$ ． — 欠落領域の補間前 —



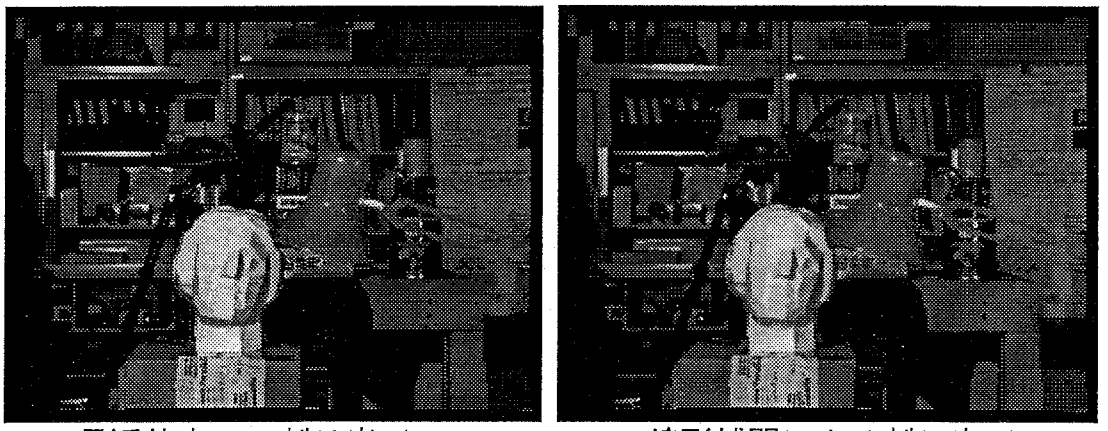
最近傍点への割り当て　　線形補間による割り当て

図 21: 実験結果 $(\Delta X, \Delta Y, \Delta Z) = (1L, 1L, -2L)$ ． — 欠落領域の補間後 —

この結果は両方の方法ともに同様な結果が得られている．実際の処理では線形補間による方法で，3点の座標変換を行った後に，その面積がしきい値よりも大きい場合はその3点は処理しないようにしている．これは，すべてのものを対象とした場合，奥行きマップのエッジ付近で三角形に大きな歪みが生じる．この歪みは，新たに生じる領域の抽出の妨げとなる．よって，

新たに生じる領域を明確にするために面積の大きさによる棄却を行っている．この面積による棄却を行うときに，遠ざかる方向では小さい値で棄却しているため，最近傍割り当てと同じような割り当てが行われていると考えられる．[6]

しかし，図22を見ると，最近傍割り当てによる方法では石膏像の左側でエッジが2重になった感じを受けるのに対し，線形補間の方法は，ぼやけた感じがする．この部分は，欠落領域の補間方法および欠落領域のテクスチャ取得の方法の改善により，より実環境に近い画像を生成できるのではないかと考える．



最近傍点への割り当て　　線形補間による割り当て

図 22: 実験結果 $(\Delta X, \Delta Y, \Delta Z) = (1L, 1L, -2L)$.

[6]近づく方向の時は，近づくにしたがってしきい値を大きくするようにしている．

4.3 視線方向の変更

視線方向の変更の結果を図23に示す．これは，視点の移動およびx軸，y軸回りの回転はなし（上下左右へのカメラの振れがない．）で，z軸に関して3°回転したものである．この結果を見ると，元の画像で直線だったものが回転によって，階段状にギザギザになっていることが分かる．

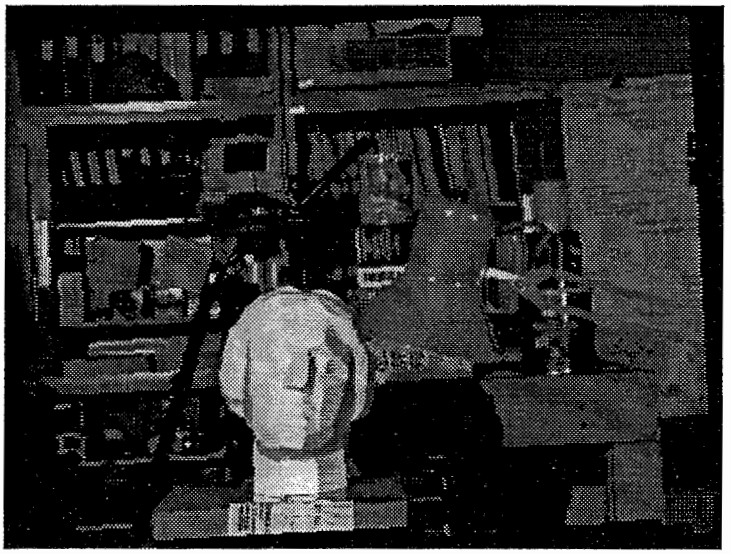


図 23: 実験結果（Z軸に関して3°回転）．

5 考察

5.1 奥行きマップの変換について．

図13のカメラ平面内移動の結果では，新たに生じる領域が正確に得られている最近傍割り当ての方が良い結果を示している．図16の対象に近づく動きを加えた場合は，最近傍割り当てにおいて欠落部分を補間しきれない領域が生じるため正確な奥行きマップが得られない．図20の対象から遠ざかる動きを加えた場合は，どちらも同じような結果を示している．これらを総合すると，線形補間による奥行きマップ変換方法が良い結果を示していることが分かる．

ここで，線形補間による方法ではZ軸方向の移動に対して，棄却面積を変化させているが，現状は経験的に決定している値であるため，より深い検討が必要だと考える．

また，線形補間による方法はわずかにボケを生じる．カメラに近く新たに生じる領域が存在するようなエッジでは仕方がないことであるが，背景部分のエッジのぼけは少し気になる．そこで，遠くの物体に対しては最近傍割り当てを用い，近くの物体に対しては線形補間による方法をもちいるのが良いのではないかと考える．しかし，あらゆる入力に対して2つの方法を切り替えるための閾値を自動決定するのは困難である．

また，本手法では欠落部分を補間する際に簡単な線形補間を用いている．これは，図24のパターン1のように，カメラから見えている物体が背景とくっついているように補間されていることに相当する．この実験データでは，図24のパターン2のように背景とはくっついていないため得られる奥行きマップは不自然なものになってしまう．しかし，入力されるデータのみからこのパターンの違いを判別するのは不可能であるため，解決策は存在しない．

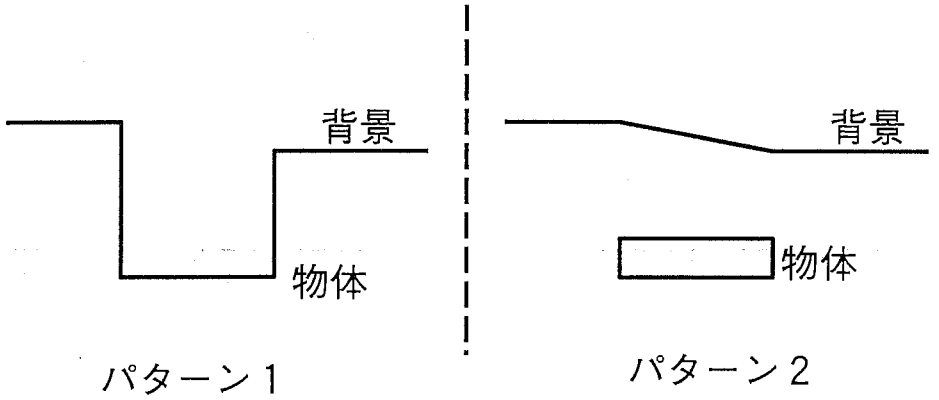


図 24: 未知領域.

また，3点による線形補間を用いる方法に対して，4点による方法も実装し比較したが，ほぼ同様な結果であったので本レポートでは割愛した．

5.2 テクスチャマッピングについて．

本手法では，Ｘ軸方向とＹ軸方向の移動量によりテクスチャマッピングするための画像を選択しているが，実際には，図20のように，右下のほうへ移動したにも関わらず，石膏像の左に新たに領域が生じている．これにより，結果として得られる画像では石膏像の右側は不自然になってしまう．また他の場合においても二つの画像を組み合わせて[7]ピクセルの値を決めているため，ぼやけた感じのエッジが構成されている．

これを解決するためには，１ピクセルごとに最適な１枚の画像からマッピングを行う必要がある．

5.3 視線方向の変更について．

結果を見れば明らかなように，直線部分が視線方向の変更により階段状にギザギザになっている．これは画素を１対１に対応させるのではなく，補間することで対応づけると解決できる。

6 まとめ

十字に配置されたカメラから得られる５枚のカラー画像と，それらからステレオマッチングによって得られる奥行きマップを用いて，任意視点映像を生成する手法について検討した．特に奥行きマップの変換における２つの手法について比較検討を行った．その結果次のような２つの方法の利点および欠点が明かになった。

- 最近傍割り当てによる方法
  - 利点：視点移動により新たに生じる領域を正確に抽出することが可能。
  - 欠点：対象に近づく方向に移動したときに、網目状のノイズが現れる。
- ３点（４点）による線形補間による方法
  - 利点：あらゆる視点移動に対応できる。（最近傍割り当てによる方法の欠点を補う。）
  - 欠点： 視点移動により新たに生じる領域を正確に抽出できない。しかし、変換後の３点が張る三角形の面積に対して棄却を行った場合、１～３画素程度の領域が抽出できないだけであるので結果にあまり影響は与えない。また、少々のぼけを生じる。

この奥行きマップの変更以外の処理（テクスチャマッピング，視線方向の変更）についても検討し今後の課題を明らかにすることができた．

[7]Ｘ軸方向に $1L$，Ｙ軸方向に $1L$ の移動なので２枚の画像からテクスチャを得ることになる．

謝辞

本実習において多大な御指導をいただい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誠喜室長をはじめとする知能映像通信研究所第三研究室の方々に，深く感謝致します．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かりて厚く御礼申し上げます．また，実験のためのデータを提供して下さいました筑波大学，大田友一教授に厚く御礼申し上げます．

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.

[2] Jong-Il park,Seiki Inoue、” Image Expression Based on Disparity Estimation from Multiple Cameras”,JW-MMC'96.

[3] Jong-Il park、Nobuyuki Yagi,Kazumasa Enami,Kiyoharu Aizawa” Estimation of Camera Parameters from Image Sequence for Model-Based Video Coding”,IEEE Transactions on Circuits and Systems for Video Technology 1994 vol4.No3 .

[4] 北原格、佐藤清秀，大田友一,”多眼ステレオ法を用いた運動視差の再現可能な３次元画像表示”,テレビジョン学会誌 Vol.50,No9.

[5] Kiyohide SATOH,Itaru KITAHARA,Yuoichi OHTA, ” 3D Image Display with Motion Parallax by Camera Matrix Stereo”,IEEE Proceedings of MULTIMEDIA'96.

付録

A　プログラム

付録としてプログラムのリストを添付する．添付するプログラムは次のものである．

- vpmv_linear3/vpmove.c: 任意視点映像生成のプログラム（奥行きマップの変換は 3 点による線形補間）
- vpmv_near/vpmove.c：任意視点映像生成のプログラム（奥行きマップの変換は最近傍割り当て）
- obj/nr_utl.c：配列の生成，クリア，消去等を実現するマクロ集.
- inc/nr_utl.h： obj/nr_utl.c に関するヘッダ.
- obj/img_utln.c：画像ファイルの入出力を扱うマクロ集.
- inc/img_utl.h： obj/img_utln.c に関するヘッダ.
- obj/osflt.c：その他のマクロ.
- inc/osflt.h： obj/osflt.c に関するヘッダ.
- vpmparas.h：画像のサイズ，解像度等に関するヘッダ.

```
/*********************************************************************
* 視点移動プログラム                                                 *
* new_depth_map は3点による線形補間。                                *
* 仮想カメラの3次元移動 と 回転 が可能。                             *
* マッピングの補間は角度による線形補間。                             *
* 実行：vpmove filename centerx centery depthfile vx vy vz rx ry rz *
*********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "../inc/nr_utl.h"
#include "../inc/img_utl.h"
#include "../inc/img_macro.h"
#include "../inc/osflt.h"
#include "vpmparas.h"

#define SR 40
/*#define SCALE ((float)SR/255.)*/
#define SCALE 1.0
#define TINY 1.0e-20

#define TINY_A ((float)1.0/1.7320508)
#define BIG ((float)1.7320508)

/*#define TINY_A 1.0e-20
#define BIG 1.0e20*/

#define FG 0
#define DP 1
#define ON 1
#define OFF 0
#define NL 5
#define WS 7
#define EM 7
#define DEPTH 3
#define PI 3.141592

#define LOW 0
#define HIGHER 1
#define RIGHT 2
#define LEFT 3

#define FI 800          /*フォーカス*/
#ifdef _SINCOS_
void sincos();
#define SINCOS(x,s,c)   sincos(x,&s,&c)
#else
double sin(), cos();
#define SINCOS(x,s,c)   s=sin(x);c=cos(x)
#endif

unsigned char **img_frnt;       /*中心画像*/
unsigned char **img_bak1;
unsigned char **img_bak2;
unsigned char **img_outp;
unsigned char **dep_cntr;       /*奥行きマップ*/
unsigned char **dep_outp;
unsigned char **mark_intpl;
float CPX,CPY,CPZ;

typedef struct{
  int x;
  int y;
} XYCOOD;
```



```
/*回転行列用構造体*/
typedef struct{
  float r1;
  float r2;
  float r3;
  float r4;
  float r5;
  float r6;
  float r7;
  float r8;
  float r9;
}Rotate;

/*--------------------------------------------------------------------------*/
void show_usage(name)
     char *name;
{
  fprintf(stderr,"Usage: %s filename centerx centery depthfile vx vy vz rx ry rz\n",n
ame);
  fprintf(stderr,"       centerx: (int) x coord. of center image in camera array\n");
  fprintf(stderr,"       centery: (int) y coord. of          // \n");
  fprintf(stderr,"       vx: x coord. of virtual camera (-1.0~1.0)\n");
  fprintf(stderr,"       vy: y coord. of virtual camera (-1.0~1.0)\n");
  fprintf(stderr,"       vz: z coord. of virtual camera (-1.0~1.0)\n");
  fprintf(stderr,"       rx: x rotate. of virtual camera (deg)\n");
  fprintf(stderr,"       ry: y rotate. of virtual camera (deg)\n");
  fprintf(stderr,"       rz: z rotate. of virtual camera (deg)\n");
  exit(1);
}
/*--------------------------------------------------------------------------*/
int select_direction(vx,vy)
     float vx,vy;
/*mappingの際に上下左右どの画像からマッピングを行なうか決定するために
視点移動方向により分類を行なう。*/
{
  if(vx==0.0)
    if(vy>0.0) return(4);
    else return(2);
  if(vx>0.0)
    if(vy/vx >= BIG) return(4);
    else if(vy/vx < BIG && vy/vx >= TINY_A) return(8);
    else if(vy/vx < TINY_A && vy/vx >= -TINY_A) return(1);
    else if(vy/vx < -TINY_A && vy/vx >= -BIG) return(5);
    else return(2);
  else
    if(vy/vx >= BIG) return(2);
    else if(vy/vx < BIG && vy/vx >= TINY_A) return(6);
    else if(vy/vx < TINY_A && vy/vx >= -TINY_A) return(3);
    else if(vy/vx < -TINY_A && vy/vx >= -BIG) return(7);
    else return(4);
}
/*--------------------------------------------------------------------------*/
unsigned char lintpl(img,sx,sy,x,y)
     unsigned char **img;  /*new Depth Image*/
     int sx,sy,x,y;
/*座標変換後決定されなかった画素の補間を行なう。*/
{
  int tx,ty,k;
  int iv[4];
  float w[4],ws,sum;

  /*(x,y)補間ポイント*/
  tx=x;
  ty=y;

  /*左方向でDepth != 0 の画素を探す。*/
```

```
  while( ty > 1 && img[--ty][tx] == 0 )
    ;
  iv[0]=img[ty][tx];
  if(ty == 1) w[0]=TINY;
  else w[0]=1./(y-ty+TINY);

  ty=y;
  /*右方向でDepth != 0 の画素を探す。*/
  while( ty < sy && img[++ty][tx] == 0 )
    ;
  iv[1]=img[ty][tx];
  if(ty == sy) w[1]=TINY;
  else w[1]=1./(ty-y+TINY);

  tx=x;
  ty=y;
  /*上方向でDepth != 0 の画素を探す。*/
  while( tx > 1 && img[ty][--tx] == 0 )
    ;
  iv[2]=img[ty][tx];
  if(tx == 1) w[2]=TINY;
  else  w[2]=1./(x-tx+TINY);

  tx=x;
  /*下方向でDepth != 0 の画素を探す。*/
  while( tx < sx && img[ty][++tx] == 0 )
    ;
  iv[3]=img[ty][tx];
  if(ty == sx) w[3]=TINY;
  else  w[3]=1./(tx-x+TINY);

  /*wの効果->その画素が遠いほど効果は薄くなる。*/
  sum=0.0;
  ws=0.0;
  for(k=0;k<4;k++)
    if(iv[k] > 0)
      {
        sum += w[k]*iv[k];
        ws += w[k];
      }
  if(ws < TINY) return(0);
  return((unsigned char)(sum/ws+0.5));

}
/*-------------------------------------------------------------------------*/
three_point_get(dep_in,img_in,vx,vy,vz,i,j,x,y,dep,r,g,b,mode)
     unsigned char **dep_in,**img_in;
     float vx,vy,vz;
     int i,j;
     float *x,*y,*dep,*r,*g,*b;
     int mode;
{
  int k,l;
  float dz,d;

  /******* 座標格納順序  *********************
   ** mode = 0:  0       *   mode = 1 :  0    1
   **                    *
   **            1    2  *                    2
   ******************************************/
  /*
   * 0
   */

  l=0;k=0;
  d = (float)dep_in[i+k][j+l]*SCALE;
```



```
  dz = 1.0-d*vz/255.0;

  /*SCALE = ((float)SR/255.) SR = 40,座標変換*/
  y[0]=((float)i+(float)k-SY/2-d*vy)/dz+SY/2;
  x[0]=((float)j+(float)l-SX/2-d*vx)/dz+SX/2;
  dep[0] = dep_in[i+k][j+l];
  r[0] = img_in[i+k][3*(j+l)-2];
  g[0] = img_in[i+k][3*(j+l)-1];
  b[0] = img_in[i+k][3*(j+l)  ];
  /*
   * 1
   */
  if(mode == 0) {k=1;l=0;}
  else{k=0;l=1;}
  d = (float)dep_in[i+k][j+l]*SCALE;
  dz = 1.0-d*vz/255.0;

  /*SCALE = ((float)SR/255.) SR = 40,座標変換*/
  y[1]=((float)i+(float)k-SY/2-d*vy)/dz+SY/2;
  x[1]=((float)j+(float)l-SX/2-d*vx)/dz+SX/2;
  dep[1] = dep_in[i+k][j+l];
  r[1] = img_in[i+k][3*(j+l)-2];
  g[1] = img_in[i+k][3*(j+l)-1];
  b[1] = img_in[i+k][3*(j+l)  ];

  /*
   * 2
   */
  k=1;l=1;
  d = (float)dep_in[i+k][j+l]*SCALE;
  dz = 1.0-d*vz/255.0;

  /*SCALE = ((float)SR/255.) SR = 40,座標変換*/
  y[2]=((float)i+(float)k-SY/2-d*vy)/dz+SY/2;
  x[2]=((float)j+(float)l-SX/2-d*vx)/dz+SX/2;
  dep[2] = dep_in[i+k][j+l];
  r[2] = img_in[i+k][3*(j+l)-2];
  g[2] = img_in[i+k][3*(j+l)-1];
  b[2] = img_in[i+k][3*(j+l)  ];
}

void push(int x,int y,XYCOOD* buf)
     /*スタックへの格納,
       ただし、格納する型はXYCOOD型。
       値は (x,y) 座標*/
{
  buf[buf[0].x+1].x = x;
  buf[buf[0].x+1].y = y;
  buf[0].x++;
}

int pop(XYCOOD* buf,XYCOOD* ret)
/************************
スタック(buf)からの取り出し、
成功:  0 を返す。
空の時:-1を返す。
*************************/
{
  if(buf[0].x > 0){
    ret->x = buf[buf[0].x].x;
    ret->y = buf[buf[0].x].y;
    buf[0].x--;
    return(0);
  }
  else return(-1);
}
```

```
int check_pt(float *x,float *y,int d1,int d2,XYCOOD *xy,int p)

/*****************************************************************/
/* float *x:                                                     */
/* float *y:                                                     */
/* int d1:                                                       */
/* int d2: 入力座標                                              */
/* XYCOOD *xy: グリッド座標群。                                  */
/* int p: LOW,HIGHER,RIGHT,LEFTの値をとる                        */
/* x[d1],y[d1]-x[d2],y[d2]から得られる、式を求め、pの領域を制限 */
/* することでpx,pyを除去する。                                   */
/*****************************************************************/
{
  int i;
  XYCOOD  buf[1024],check;
  float a,b;

  buf[0].x = 0; /*スタック buf の初期化、
                  buf[0].xが格納されているデータの個数を表す、*/


  if((p == HIGHER || p == LOW) && y[d2] - y[d1] == 0.0)
    /* 直線 y = ?  の場合 */
    switch(p){
    case HIGHER:
      while(pop(xy,&check) == 0){    /*一つずつ座標を取り出しチェック*/
        if(check.y >= y[d1])
          push(check.x,check.y,buf);
      }
      while(pop(buf,&check) == 0){
        push(check.x,check.y,xy);
      }
      break;
    case LOW:
      while(pop(xy,&check) == 0){
        if(check.y <= y[d1])
          push(check.x,check.y,buf);
      }
      while(pop(buf,&check) == 0){
        push(check.x,check.y,xy);
      }
    }
  else if((p == LEFT || p == RIGHT) && x[d2] - x[d1] == 0.0)
    /* 直線 x = ?  の場合 */
    switch(p){

    case LEFT:
      while(pop(xy,&check) == 0){
        if(check.x <= x[d1])
          push(check.x,check.y,buf);
      }
      while(pop(buf,&check) == 0){
        push(check.x,check.y,xy);
      }
      break;
    case RIGHT:
      while(pop(xy,&check) == 0){
        if(check.x >= x[d1])
          push(check.x,check.y,buf);
      }
      while(pop(buf,&check) == 0){
        push(check.x,check.y,xy);
      }
```



```
    }
  else if(y[d2] - y[d1] == 0.0 || x[d2] - x[d1] == 0.0);
  else {                          /* 直線 y = ax+b の場合*/
    a = (y[d2] - y[d1])/(x[d2] - x[d1]);
    b = y[d1] - a * x[d1];
    /*printf("a = %f b = %f \n ",a,b);*/
    switch(p){
    case LEFT:
      while(pop(xy,&check) == 0){
        if(check.x <= (check.y-b)/a)
          push(check.x,check.y,buf);
      }
      while(pop(buf,&check) == 0){
        push(check.x,check.y,xy);
      }
      break;
    case RIGHT:
      while(pop(xy,&check) == 0){
        if(check.x >= (check.y-b)/a)
          push(check.x,check.y,buf);
      }
      while(pop(buf,&check) == 0){
        push(check.x,check.y,xy);
      }
      break;
    case HIGHER:
      while(pop(xy,&check) == 0){
        if(check.y >= a * check.x +b)
          push(check.x,check.y,buf);
      }
      while(pop(buf,&check) == 0){
        push(check.x,check.y,xy);
      }
      break;
    case LOW:
      while(pop(xy,&check) == 0){
        if(check.y <= a * check.x +b)
          push(check.x,check.y,buf);
      }
      while(pop(buf,&check) == 0){
        push(check.x,check.y,xy);
      }
    }

  }
}
int dir_check(float *x,float *y,int p1,int p2){
  int p;
  float a,b;

  if(p1 == 0){
    if(p2 == 1) p = 2;
    else  p = 1;
  }else if(p1 == 1){
    if(p2 == 0) p = 2;
    else p = 0;
  }else if(p1 == 2){
    if(p2 == 0) p = 1;
    else p = 0;
  }

  if(x[p1] <= x[p] && x[p2] <= x[p]) return RIGHT;
  else if(x[p1] > x[p] && x[p2] > x[p]) return LEFT;
  else if(y[p1] <= y[p] && y[p2] <= y[p]) return HIGHER;
  else if(y[p1] > y[p] && y[p2] > y[p]) return LOW;
```

```
  a = (y[p2] - y[p1])/(x[p2] - x[p1]);
  b = y[p1] - a * x[p1];
  if(a < 1.0){
    if(y[p] >= a * x[p] +b) return HIGHER;
    else return LOW;
  }else{
    if(x[p] <= (y[p]-b)/a) return LEFT;
    else return RIGHT;
  }
}

int in_cood(float *x,float *y,XYCOOD *p_buf,float vz)
/*********************************************************
3点で張られる三角形中のグリッド（座標）を求める。
入力 float *x
     float *y
出力 XYCOOD *p_buf

返値 要素数 または 0 （ないとき、またはエラー時）
*********************************************************/
{
  int i,j,k;
  float xmax=-1000.0,xmin=1000.0,ymax=-1000.0,ymin=1000.0;
  float men,siki,m=0.0;

  p_buf[0].x = 0; /*要素数を示す。*/

  if(vz < 0.0) siki = 5;
  siki = m * vz/4.0 * 150 +1;
  if(m * vz > 4.0) siki=150;



  /*
   *外接矩形中の点を求める。
   */
  for(i=0;i<3;i++){
    if(xmax < x[i]) {xmax = x[i];}
    if(xmin > x[i]) {xmin = x[i];}
    if(ymax < y[i]) {ymax = y[i];}
    if(ymin > y[i]) {ymin = y[i];}
  }

  /*
   * 外接矩形の面積が大きい場合は無視する。
   */
  men = (xmax-xmin)*(ymax-ymin);
  if(men > siki) {return 0; }

  for(i=(int)xmin;i<=(int)xmax;i++){
    for(j=(int)ymin;j<=(int)ymax;j++){
      if(xmin > 0.0 && ymin >  0.0 && xmax <  640 &&  ymax < 480){
        push(i,j,p_buf);
        /*fprintf(stdout,"n=%d\n",p_buf[0].x);
        fprintf(stdout,"i=%d,j=%d \n",i,j);*/
      }
    }
  }
  /*fprintf(stdout,"\n\n");*/
  /*
   *領域を限定して、三角形の内部のグリッドのみを求める。
   */

  /*2点 P0-P1*/
  /*fprintf(stdout,"P1 = %d,P2 = %d \n",y_sort[0],y_sort[1]);*/
  check_pt(x,y,0,1,p_buf,dir_check(x,y,0,1));
```



```
  /*for(j=1;j<=p_buf[0].x;j++)
    fprintf(stdout,"i=%d,j=%d \n",p_buf[j].x,p_buf[j].y);
  fprintf(stdout,"\n\n");*/

  /*2点 P1-P2*/
   check_pt(x,y,1,2,p_buf,dir_check(x,y,1,2));

  /*2点 P2-P0*/
   check_pt(x,y,2,0,p_buf,dir_check(x,y,2,0));


  if(p_buf[0].x > 0) return p_buf[0].x;
  else return 0;
}

/*-------------------------------------------------------------------------*/
int sort(kk,n)
/*バブルソート*/
     int *kk,n;
[
   /* sort an integer array of size n in the ascending order
    */

   int i,j,ti,*odr,val;

   odr=ivector(1,n);
   for(i=1;i<=n;i++)
      odr[i]=i;

   for(i=1;i<n;i++)
      for(j=i+1;j<=n;j++)
         if(kk[j]<kk[i])
         [
            ti=kk[i];
            kk[i]=kk[j];
            kk[j]=ti;
            ti=odr[j];
            odr[j]=odr[i];
            odr[i]=ti;
         ]
   val=odr[(n+1)/2];
   free_ivector(odr,1,n);
   return(val);
]

/*-------------------------------------------------------------------------*/
void fill_in(img,dimg,sx,sy)
   unsigned char **img,**dimg;
   int sx,sy;
/*メディアンフィルタ*/
[
   int i,j,k,m,dmax,itmp[10],idx;
   unsigned char **td,**ti;

   td=cmatrix(1,sy,1,sx);
   ti=cmatrix(1,sy,1,3*sx);
   if((td==NULL)||(ti==NULL))
   [  printf("Memory not enough. Fail in Fill_in().\n");
      exit(1);
   ]
   for(i=1;i<=sy;i++)
     for(j=1;j<=sx;j++)
       td[i][j]=dimg[i][j];
   for(i=1;i<=sy;i++)
     for(j=1;j<=3*sx;j++)
       ti[i][j]=img[i][j];
```

```
    for(i=3;i<=sy-2;i++)
      for(j=3;j<=sx-2;j++)
        {
          dmax=0;
          idx=4;
          for(k=(-1);k<=1;k++)
            for(m=(-1);m<=1;m++)
              itmp[(k+1)*3+m+2]=td[i+k][j+m];

          idx = sort(itmp,9);
          idx--;
          dimg[i][j]=itmp[5];
          k=idx/3-1;
          m=(idx%3)-1;
          img[i][3*j-2]=ti[i+k][3*(j+m)-2];
          img[i][3*j-1]=ti[i+k][3*(j+m)-1];
          img[i][3*j]  =ti[i+k][3*(j+m)];
        }
    free_cmatrix(td,1,sy,1,sx);
    free_cmatrix(ti,1,sy,1,3*sx);

  }


/*-------------------------------------------------------------------------*/
void new_depth_map(img_in,dep_in,sx,sy,vx,vy,vz,img_out,dep_out,mark)
     unsigned char **img_in,**dep_in,**img_out,**dep_out,**mark;
     int sx,sy;
     float vx,vy,vz;
{
  int i,j,k,nx,ny,p3,tmp=0,np,colx,coly;
  unsigned char **tmpimg,new_dep;
  float x[3],y[3],dep[3],red[3],green[3],blue[3],np_val;
  double distance[5],new_d,new_r,new_g,new_b,aaa;
  int p_num,mode;
  XYCOOD pxy[1024];

  /*領域確保*/
  tmpimg = cmatrix(1,sy,1,sx);

  if(tmpimg==NULL){
    fprintf(stderr,"Insufficient memory in allocating tmpimg.\n");
    exit(1.0);
  }

  /*初期化*/
  for(i=1;i<=sy;i++)
    for(j=1;j<=sx;j++)
      tmpimg[i][j]=0;

  for(i=1;i<sy;i++){
    for(j=1;j<sx;j++){
      for(mode = 0;mode<2;mode++){
        /*３点の座標変換を行う．*/
        three_point_get(dep_in,img_in,vx,vy,vz,i,j,x,y,dep,red,green,blue,mode);
        /*３点で張られる三角形に含まれるグリッドを求める．*/
        p_num =  in_cood(x,y,pxy,vz);
        if(p_num == 0) tmp++;

        for(k=1;k<=p_num;k++){
          /*EM = 7,画像の外枠７画素は無視する。*/
          if( (pxy[k].x < 1+EM) || (pxy[k].x > sx-EM)
             || (pxy[k].y < 1+EM) || (pxy[k].y > sy-EM))
            continue;
          else{
            distance[0] = 0.0;np_val = 100000.0;
```



```
          /*距離を求める。*/
          for(p3=0;p3<3;p3++){
            distance[p3+1] =
              sqrt( (double)(x[p3] - pxy[k].x)*(x[p3] - pxy[k].x)+
                   (double)(y[p3] -  pxy[k].y)*(y[p3] -  pxy[k].y));
            distance[0] += 1.0/distance[p3+1];
            if(distance[p3+1] < np_val)
              {np_val = (float)distance[p3+1];np=p3;}
          }

          new_d = 0.0;
          /*線形補間*/
          for(p3=0;p3<3;p3++){
            if(distance[p3+1] == 0.0) /*グリッドにちょうど乗る時*/
            {new_d = (double)dep[p3];break;}
            new_d += ((aaa = ((1.0/distance[p3+1])/distance[0]) )
                      * (double)dep[p3]);
          }

          if(new_d >= 255.0) {new_d = 255.0;}
          if(new_d < 0.0) {new_d = 0.0;}
          new_dep = (unsigned char)new_d;

          /*一番近くの値を用いる。*/
          if(new_dep > tmpimg[pxy[k].y][pxy[k].x]) {
            tmpimg[pxy[k].y][pxy[k].x]      =  new_dep;  /*  コピー*/

            /*画像の線形補間*/
            new_r = 0.0;
            new_g = 0.0;
            new_b = 0.0;
            for(p3=0;p3<3;p3++){
              if(distance[p3+1] == 0.0) /*グリッドにちょうど乗る時*/
              {
                new_r = (double)red[p3];
                new_g = (double)green[p3];
                new_b = (double)blue[p3];
                break;
              }
              new_r += ((1.0/distance[p3+1])/distance[0]) * (double)red[p3];
              new_g += ((1.0/distance[p3+1])/distance[0])*(double)green[p3];
              new_b += ((1.0/distance[p3+1])/distance[0])* (double)blue[p3];
            }

            if(new_r >= 255.0) {new_r = 255.0;}
            if(new_g >= 255.0) {new_g = 255.0;}
            if(new_b >= 255.0) {new_b = 255.0;}
            if(new_r < 0.0) {new_r = 0.0;}
            if(new_g < 0.0) {new_g = 0.0;}
            if(new_b < 0.0) {new_b = 0.0;}

            img_out[pxy[k].y][3*pxy[k].x-2] = (unsigned char)new_r;
            img_out[pxy[k].y][3*pxy[k].x-1] = (unsigned char)new_g;
            img_out[pxy[k].y][3*pxy[k].x  ] = (unsigned char)new_b;

            /*３点中の最も距離が近い点のピクセル値を用いる。
            coly = 0; colx = 0;
            if(np == 1)
              if(mode == 0)
                coly = 1;
              else
                colx = 1;
            else if(np = 2) {coly = 1; colx = 1;}

            img_out[pxy[k].y][3*pxy[k].x-2] = img_in[i+coly][3*(j+colx)-2];
```

```
              img_out[pxy[k].y][3*pxy[k].x-1] = img_in[i+coly][3*(j+colx)-1];
              img_out[pxy[k].y][3*pxy[k].x  ] = img_in[i+coly][3*(j+colx)];*/
            }
          }
        }
      }
    }
  }
  /*中間結果画像の出力*/
  image_write(tmpimg,sx,sy,"test.dpt",PGM);
  image_write(img_out,sx,sy,"test.rgb",PPM);

  /*メディアンフィルタ*/
  fill_in(img_out,tmpimg,sx,sy);

  fprintf(stderr,"!! midle result image save !!\n");

  fprintf(stderr,"!! hokan !! \n");
  for(i=1;i<=sy;i++)
    for(j=1;j<=sx;j++){
      if(tmpimg[i][j] == 0){
        dep_out[i][j]=lintpl(tmpimg,sx,sy,j,i);
        mark[i][j]=ON;
      }
      else
        dep_out[i][j]=tmpimg[i][j];
    }

  free_cmatrix(tmpimg,1,sy,1,sx);
}

/*--------------------------------------------------------------------------*/
void mapping(img_cntr,img_1,img_2,
             dep_cur,mark,sx,sy,vx,vy,vz,intpl_dir,img_out)
     unsigned char **img_cntr,**img_1,**img_2,**dep_cur,**mark,**img_out;
     int sx,sy,intpl_dir;
     float vx,vy,vz;
{
  int i,j,nx,ny;
  float vvx1,vvy1,vvx2,vvy2,d,dz,pr1,pr2;
  double big_deg,tiny_deg,v_deg,range;
  unsigned char **tmpimg;

  tmpimg=cmatrix(1,sy,1,3*sx);

  /*初期化*/
  for(i=1;i<=sy;i++)
    for(j=1;j<=3*sx;j++)
      tmpimg[i][j]=0;

  switch(intpl_dir){
  case 1:
    vvx1=vx-1.0;
    vvy1=vy;
    break;
  case 2:
    vvx1=vx;
    vvy1=vy+1.0;
    break;
  case 3:
    vvx1=vx+1.0;
    vvy1=vy;
    break;
  case 4:
    vvx1=vx;
    vvy1=vy-1.0;
```



```
    break;
  case 5:
    vvx1=vx-1.0;
    vvy1=vy;
    vvx2=vx;
    vvy2=vy+1.0;
    break;
  case 6:
    vvx1=vx;
    vvy1=vy+1.0;
    vvx2=vx+1.0;
    vvy2=vy;
    break;
  case 7:
    vvx1=vx+1.0;
    vvy1=vy;
    vvx2=vx;
    vvy2=vy-1.0;
    break;
  case 8:
    vvx1=vx;
    vvy1=vy-1.0;
    vvx2=vx-1.0;
    vvy2=vy;
    break;
  default:
    fprintf(stderr,"Error! Improper direction of interpolation.\n");
    exit(1.0);
  }
  switch(intpl_dir){
  case 1:
  case 2:
  case 3:
  case 4:
    for(i=1;i<=sy;i++)
      for(j=1;j<=sx;j++)
        {
          if(mark[i][j] == ON)
            {
              d = (float)dep_cur[i][j]*SCALE;
              d /= (1.0-d*vz/255.0);      /*z方向の移動分に対するdepthの補正*/
              dz = 1.0+d*vz/255.0;
              ny=(int)(((float)i-SY/2+d*vvy1)/dz+SY/2+0.5);
              nx=(int)(((float)j-SX/2+d*vvx1)/dz+SX/2+0.5);
              if( nx < 1+EM || nx > sx-EM || ny < 1+EM || ny > sy-EM)
                mark[i][j]=2;
              else
                {  img_out[i][3*j-2]=img_1[ny][3*nx-2];
                   img_out[i][3*j-1]=img_1[ny][3*nx-1];
                   img_out[i][3*j  ]=img_1[ny][3*nx];
                 }
            }
        }
    for(i=1;i<=sy;i++)
      for(j=1;j<=sx;j++)
        if(mark[i][j] == 2)
          {  tmpimg[i][3*j-2]=0;
             tmpimg[i][3*j-1]=0;
             tmpimg[i][3*j]=0;
           }

    /*              intplrgb(img_out,sx,sy,j,i,tmpimg);
     */
    for(i=1;i<=sy;i++)
      for(j=1;j<=sx;j++)
        if(mark[i][j] == 2)
```

```
          {  img_out[i][3*j-2]=tmpimg[i][3*j-2];
             img_out[i][3*j-1]=tmpimg[i][3*j-1];
             img_out[i][3*j  ]=tmpimg[i][3*j];
          }
      break;
    case 5:
    case 6:
    case 7:
    case 8:
      for(i=1;i<=sy;i++)
        for(j=1;j<=sx;j++){

          if(mark[i][j] == ON){
            d = (float)dep_cur[i][j]*SCALE;
            d /= (1.0-d*vz/255.0);
            dz = 1.0+d*vz/255.0;
            ny=(int)(((float)i-SY/2+d*vvy1)/dz+SY/2+0.5);
            nx=(int)(((float)j-SX/2+d*vvx1)/dz+SX/2+0.5);

            if( nx < 1+EM || nx > sx-EM || ny < 1+EM || ny > sy-EM)
              mark[i][j]=2;
            else {
              tmpimg[i][3*j-2]=img_1[ny][3*nx-2];
              tmpimg[i][3*j-1]=img_1[ny][3*nx-1];
              tmpimg[i][3*j]  =img_1[ny][3*nx];
            }

            d = (float)dep_cur[i][j]*SCALE;
            d /= (1.0-d*vz/255.0);
            dz = 1.0+d*vz/255.0;
            ny=(int)(((float)i-SY/2+d*vvy2)/dz+SY/2+0.5);
            nx=(int)(((float)j-SX/2+d*vvx2)/dz+SX/2+0.5);

            if( nx < 1+EM || nx > sx-EM || ny < 1+EM || ny > sy-EM){
              if(mark[i][j]==2)
                mark[i][j]=4;
              else
                mark[i][j]=3;
            }
            else{
              img_out[i][3*j-2]=img_2[ny][3*nx-2];
              img_out[i][3*j-1]=img_2[ny][3*nx-1];
              img_out[i][3*j]  =img_2[ny][3*nx];
            }
          }
        }

    big_deg = atan((double)BIG);
    tiny_deg = atan((double)TINY_A);
    range = big_deg - tiny_deg;
    v_deg = atan2(vy,vx)+PI;

    while(v_deg > PI/2)
    v_deg = v_deg - PI/2;

    pr1 = (big_deg- v_deg)/range;
    pr2 = (v_deg - tiny_deg)/range;

    fprintf(stderr,"big_deg = %f \n  tiny_deg = %f \n  range  = %f \n  v_deg  = %f \n
  pr1 = %f\n   pr2 = %f \n",big_deg*180/PI,tiny_deg*180/PI,range*180/PI,v_deg*180/PI,
pr1,pr2);
    for(i=1;i<=sy;i++)
      for(j=1;j<=sx;j++){
        switch(mark[i][j]){

        case 1:
```



```
          img_out[i][3*j-2]=(unsigned char)
            ((int)tmpimg[i][3*j-2]*pr1+(int)img_out[i][3*j-2]*pr2);
          img_out[i][3*j-1]=(unsigned char)
            ((int)tmpimg[i][3*j-1]*pr1+(int)img_out[i][3*j-1]*pr2);
          img_out[i][3*j]  =(unsigned char)
            ((int)tmpimg[i][3*j  ]*pr1+(int)img_out[i][3*j]*pr2);
          break;

        case 2:
          break;

        case 3:
          img_out[i][3*j-2]=tmpimg[i][3*j-2];
          img_out[i][3*j-1]=tmpimg[i][3*j-1];
          img_out[i][3*j]  =tmpimg[i][3*j];
          break;

        case 4:
          img_out[i][3*j-2]=0;
          img_out[i][3*j-1]=0;
          img_out[i][3*j]  =0;
          /*      intplrgb(img_out,sx,sy,j,i,img_out);
           */                 break;
        default:
          ;
        }
      }
    }
  free_cmatrix(tmpimg,1,sy,1,3*sx);
}
/*--------------------------------------------------------------------------------*/

void camera_rotate(in_img,out_img,rx,ry,rz,zoom,color)
unsigned char **in_img,**out_img;
float rx,ry,rz,zoom;
int color;
{
  Rotate cc;
  double sa,ca,sb,cb,sg,cg;
  double P,Q,R;

  double xo,yo;
  int i,j;

  SINCOS (rx, sa, ca);
  SINCOS (ry, sb, cb);
  SINCOS (rz, sg, cg);

  cc.r1 = cb * cg;
  cc.r2 = cg * sa * sb - ca * sg;
  cc.r3 = sa * sg + ca * cg * sb;
  cc.r4 = cb * sg;
  cc.r5 = sa * sb * sg + ca * cg;
  cc.r6 = ca * sb * sg - cg * sa;
  cc.r7 = -sb;
  cc.r8 = cb * sa;
  cc.r9 = ca * cb;

  if(color == 0)
    for(i=1;i<=SX;i++)
      for(j=1;j<=SY;j++){
        out_img[j][i] = 0;
      }

  else if(color == 1){
    for(i=1;i<=SX;i++)
```

```
      for(j=1;j<=SY;j++){
        out_img[j][3*i-2] = 0 ;
        out_img[j][3*i-1] = 0 ;
        out_img[j][3*i  ] = 0 ;
      }
  }

  for(i=1;i<=SX;i++)
    for(j=1;j<=SY;j++){
      i -= SX/2;
      j -= SY/2;

      P=cc.r1 * i  +  cc.r2 * j  +  cc.r3 * FI;
      Q=cc.r4 * i  +  cc.r5 * j  +  cc.r6 * FI;
      R=cc.r7 * i  +  cc.r8 * j  +  cc.r9 * FI;
      xo = 1.0/zoom * FI * P / R;
      yo = 1.0/zoom * FI * Q / R;

      i += SX/2;
      j += SY/2;
      xo += SX/2;
      yo += SY/2;

      if((int)xo > 0 && (int)xo <= SX && (int)yo > 0 && (int)yo <= SY){
        if(color == 0)
          out_img[j][i] = in_img[(int)yo][(int)xo];
        else if(color == 1){
          out_img[j][3*i-2] = in_img[(int)yo][3*(int)xo-2];
          out_img[j][3*i-1] = in_img[(int)yo][3*(int)xo-1];
          out_img[j][3*i  ] = in_img[(int)yo][3*(int)xo  ];
        }
      }
    }
}

main(argc,argv)
     int argc;
     char *argv[];
{

  FILE *fp;

  char c,str[50];
  int i,j,new_key,sx=SX,sy=SY,sxf=SX,syf=SY,*img_type,cx,cy,intpl_dir;
  /*SX = 640,SY = 480 in vpmparas.h*/
  float vx,vy,vz,rx,ry,rz;
  float zoom=1.0;

  if(argc<11 || argc>11) show_usage(argv[0]);
  /*(cx,cy) center camera coodinate(camera matrix 8x8)*/
  cx=atoi(argv[2]);
  cy=atoi(argv[3]);

  /*(vx,vy,vz) view point move.each -1.0~1.0 */
  vx = (float)atof(argv[5]);
  vy = (float)atof(argv[6]);
  vz = (float)atof(argv[7]);

  /*(rx,ry,rz) view point rotate.*/
  rx = (float)atof(argv[8]) * PI/180.0;
  ry = (float)atof(argv[9]) * PI/180.0;
  rz = (float)atof(argv[10])* PI/180.0;

  img_type   = ivector(0,NL);   /*NL=5 in header*/
  img_frnt   = cmatrix(1,SY,1,3*SX);  /*Center Image*/
  img_bak1   = cmatrix(1,SY,1,3*SX);  /*この画像よりマッピングを行なう1*/
```



```
  img_bak2   = cmatrix(1,SY,1,3*SX);  /*この画像よりマッピングを行なう2*/
  img_outp   = cmatrix(1,SY,1,3*SX);
  dep_cntr   = cmatrix(1,SY,1,SX);    /*Depth Image*/
  dep_outp   = cmatrix(1,SY,1,SX);
  mark_intpl = cmatrix(1,SY,1,SX);  /*座標変換によって得られなかった画素*/

  sprintf(str,"%s_%d_%d",argv[1],cx,cy);
  image_read(img_frnt,&sxf,&syf,str,&img_type[FG]);  /*FG = 0*/

  /*Image size*/
  sx=SX;
  sy=SY;

  image_read(dep_cntr,&sx,&sy,argv[4],&img_type[DP]); /*DP = 1*/

  do
    {
      CPX = vx;
      CPY = vy;
      CPZ = vz;

      new_depth_map(img_frnt,dep_cntr,
                    sx,sy,vx,vy,vz,img_outp,dep_outp,mark_intpl);

      ykh(dep_outp,sx,sy,WS,1,dep_cntr);
      ykh(dep_cntr,sx,sy,WS,1,dep_outp);
      ykh(dep_outp,sx,sy,WS,1,dep_cntr);

      /*テクスチャーマッピングに用いる画像を開く*/
      switch(intpl_dir=select_direction(vx,vy))
        {
        case 1:
          sprintf(str,"%s_%d_%d",argv[1],cx+1,cy);
          image_read(img_bak1,&sxf,&syf,str,&img_type[FG]);
          break;
        case 2:
          sprintf(str,"%s_%d_%d",argv[1],cx,cy-1);
          image_read(img_bak1,&sxf,&syf,str,&img_type[FG]);
          break;
        case 3:
          sprintf(str,"%s_%d_%d",argv[1],cx-1,cy);
          image_read(img_bak1,&sxf,&syf,str,&img_type[FG]);
          break;
        case 4:
          sprintf(str,"%s_%d_%d",argv[1],cx,cy+1);
          image_read(img_bak1,&sxf,&syf,str,&img_type[FG]);
          break;
        case 5:
          sprintf(str,"%s_%d_%d",argv[1],cx+1,cy);
          image_read(img_bak1,&sxf,&syf,str,&img_type[FG]);
          sprintf(str,"%s_%d_%d",argv[1],cx,cy-1);
          image_read(img_bak2,&sxf,&syf,str,&img_type[FG]);
          break;
        case 6:
          sprintf(str,"%s_%d_%d",argv[1],cx,cy-1);
          image_read(img_bak1,&sxf,&syf,str,&img_type[FG]);
          sprintf(str,"%s_%d_%d",argv[1],cx-1,cy);
          image_read(img_bak2,&sxf,&syf,str,&img_type[FG]);
          break;
        case 7:
          sprintf(str,"%s_%d_%d",argv[1],cx-1,cy);
          image_read(img_bak1,&sxf,&syf,str,&img_type[FG]);
          sprintf(str,"%s_%d_%d",argv[1],cx,cy+1);
          image_read(img_bak2,&sxf,&syf,str,&img_type[FG]);
          break;
        case 8:
```

```
          sprintf(str,"%s_%d_%d",argv[1],cx,cy+1);
          image_read(img_bak1,&sxf,&syf,str,&img_type[FG]);
          sprintf(str,"%s_%d_%d",argv[1],cx+1,cy);
          image_read(img_bak2,&sxf,&syf,str,&img_type[FG]);
          break;
        }
      /*テクスチャーマッピング*/
      mapping(img_frnt,img_bak1,img_bak2,dep_cntr,mark_intpl,
              sx,sy,vx,vy,vz,intpl_dir,img_outp);
      fprintf(stderr,"!! mapping end !! \n");

      /*視線方向の変更*/
      camera_rotate(img_outp,img_frnt,rx,ry,rz,zoom,1);
      camera_rotate(dep_outp,dep_cntr,rx,ry,rz,zoom,0);

      image_write(img_frnt,sxf,syf,
                  "/home/is/akira-ma/WORK/ATR/tmp3/tmp.rgb",img_type[FG]);
      image_write(dep_cntr,sxf,syf,"/home/is/akira-ma/WORK/ATR/tmp3/tmp.dpt",PGM);

      sprintf(str,"imgview /home/is/akira-ma/WORK/ATR/tmp3/tmp.rgb &");
      system(str);
      sprintf(str,"imgview /home/is/akira-ma/WORK/ATR/tmp3/tmp.dpt &");
      system(str);

      fprintf(stderr,"Current Virtual Camera Position= (%5.2f,%5.2f,%5.2f)-(%5.2f,%5.
2f,%5.2f).\n",vx,vy,vz,rx,ry,rz);
      fprintf(stderr,"If you want to save the composed image, type \'s\'.\n");
      fprintf(stderr,"               terminate this program , type \'q\'.\n");
      fprintf(stderr,"               or press [return] to continue.\n");
      fflush(stdin);
      scanf("%c",&c);

      if(c == 's')
        {
          sprintf(str,"%s_%d_%d_%02d_%02d_%02d",argv[1],cx,cy,(int)(vx*100),(int)(vy*
100),(int)(vz*100));
          image_write(img_outp,sxf,syf,str,img_type[FG]);
          sprintf(str,"%s.depth",str);
          image_write(dep_outp,sxf,syf,str,PGM);
        }

      if(c == 'q')
        break;

      fprintf(stderr,"Insert a new virtual camera coordinate.\n");
      fflush(stdin);
      scanf("%f %f %f %f %f %f",&vx,&vy,&vz,&rx,&ry,&rz);

      image_read(dep_cntr,&sx,&sy,argv[4],&img_type[DP]);
      clear_cmatrix(img_bak1,1,SY,1,3*SX);
      clear_cmatrix(img_bak2,1,SY,1,3*SX);
      clear_cmatrix(img_outp,1,SY,1,3*SX);
      clear_cmatrix(dep_outp,1,SY,1,SX);
      clear_cmatrix(mark_intpl,1,SY,1,SX);
    } while(1);

  free_ivector(img_type,0,NL);
  free_cmatrix(img_frnt,1,SY,1,3*SX);
  free_cmatrix(img_bak1,1,SY,1,3*SX);
  free_cmatrix(img_bak2,1,SY,1,3*SX);
  free_cmatrix(img_outp,1,SY,1,3*SX);
  free_cmatrix(dep_cntr,1,SY,1,SX);
  free_cmatrix(dep_outp,1,SY,1,SX);
  free_cmatrix(mark_intpl,1,SY,1,SX);
}
```

```
/*視点移動プログラム
new_depth_map は再近傍点割り当て。
３次元移動 回転。
マッピングの補間は角度による線形補間。
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "../inc/nr_utl.h"
#include "../inc/img_utl.h"
#include "../inc/img_macro.h"
#include "../inc/osflt.h"
#include "vpmparas.h"

#define SR 40
/*#define SCALE ((float)SR/255.)*/
#define SCALE 1.0

#define TINY 1.0e-20

/*#define TINY_A 1.0e-20
#define BIG 1.0e20*/

#define TINY_A ((float)1.0/1.7320508)
#define BIG ((float)1.7320508)

#define FG 0
#define DP 1
#define ON 1
#define OFF 0
#define NL 5
#define WS 7
#define EM 7
#define DEPTH 3
#define PI 3.141592

#define RED   2
#define GREEN 1
#define BLUE  0

#define LOW 0
#define HIGHER 1
#define RIGHT 2
#define LEFT 3

#define FI 800
#define PAI 3.141592
#ifdef _SINCOS_
void sincos();
#define SINCOS(x,s,c)    sincos(x,&s,&c)
#else
double sin(), cos();
#define SINCOS(x,s,c)    s=sin(x);c=cos(x)
#endif

unsigned char **img_frnt;
unsigned char **img_bak1;
unsigned char **img_bak2;
unsigned char **img_outp;
unsigned char **dep_cntr;
unsigned char **dep_outp;
unsigned char **mark_intpl;
float CPX,CPY,CPZ;

typedef struct{
```



```
  int x;
  int y;
} XYCOOD;

typedef struct{
  float r1;
  float r2;
  float r3;
  float r4;
  float r5;
  float r6;
  float r7;
  float r8;
  float r9;
}Rotate;

/*--------------------------------------------------------------------------*/
void show_usage(name)
     char *name;
{
  fprintf(stderr,"Usage: %s filename centerx centery depthfile vx vy vz rx ry rz\n",n
ame);
  fprintf(stderr,"       centerx: (int) x coord. of center image in camera array\n");
  fprintf(stderr,"       centery: (int) y coord. of             // \n");
  fprintf(stderr,"       vx: x coord. of virtual camera (-1.0~1.0)\n");
  fprintf(stderr,"       vy: y coord. of virtual camera (-1.0~1.0)\n");
  fprintf(stderr,"       vz: z coord. of virtual camera (-1.0~1.0)\n");
  fprintf(stderr,"       rx: x rotate. of virtual camera (deg)\n");
  fprintf(stderr,"       ry: y rotate. of virtual camera (deg)\n");
  fprintf(stderr,"       rz: z rotate. of virtual camera (deg)\n");
  exit(1);
}
/*--------------------------------------------------------------------------*/
int select_direction(vx,vy)
     float vx,vy;
/*mappingの際に上下左右どの画像からマッピングを行なうか決定するために
視点移動方向により分類を行なう。*/
{
  if(vx==0.0)
    if(vy>0.0) return(4);
    else return(2);
  if(vx>0.0)
    if(vy/vx >= BIG) return(4);
    else if(vy/vx < BIG && vy/vx >= TINY_A) return(8);
    else if(vy/vx < TINY_A && vy/vx >= -TINY_A) return(1);
    else if(vy/vx < -TINY_A && vy/vx >= -BIG) return(5);
    else return(2);
  else
    if(vy/vx >= BIG) return(2);
    else if(vy/vx < BIG && vy/vx >= TINY_A) return(6);
    else if(vy/vx < TINY_A && vy/vx >= -TINY_A) return(3);
    else if(vy/vx < -TINY_A && vy/vx >= -BIG) return(7);
    else return(4);
}
/*--------------------------------------------------------------------------*/
unsigned char lintpl(img,sx,sy,x,y)
     unsigned char **img;  /*new Depth Image*/
     int sx,sy,x,y;
/*座標変換後決定されなかった画素の補間を行なう。*/
{
  int tx,ty,k;
  int iv[4];
  float w[4],ws,sum;

  /*(x,y)補間ポイント*/
  tx=x;
```

```
  ty=y;

  /*左方向でDepth != 0 の画素を探す。*/
  while( ty > 1 && img[--ty][tx] == 0 )
    ;
  iv[0]=img[ty][tx];
  if(ty == 1) w[0]=TINY;
  else w[0]=1./(y-ty+TINY);

  ty=y;
  /*右方向でDepth != 0 の画素を探す。*/
  while( ty < sy && img[++ty][tx] == 0 )
    ;
  iv[1]=img[ty][tx];
  if(ty == sy) w[1]=TINY;
  else w[1]=1./(ty-y+TINY);

  tx=x;
  ty=y;
  /*上方向でDepth != 0 の画素を探す。*/
  while( tx > 1 && img[ty][--tx] == 0 )
    ;
  iv[2]=img[ty][tx];
  if(tx == 1) w[2]=TINY;
  else  w[2]=1./(x-tx+TINY);

  tx=x;
  /*下方向でDepth != 0 の画素を探す。*/
  while( tx < sx && img[ty][++tx] == 0 )
    ;
  iv[3]=img[ty][tx];
  if(ty == sx) w[3]=TINY;
  else  w[3]=1./(tx-x+TINY);

  /*wの効果->その画素が遠いほど効果は薄くなる。*/
  sum=0.0;
  ws=0.0;
  for(k=0;k<4;k++)
    if(iv[k] > 0)
      {
        sum += w[k]*iv[k];
        ws += w[k];
      }
  if(ws < TINY) return(0);
  return((unsigned char)(sum/ws+0.5));

}

/*--------------------------------------------------------------------------*/
int sort(kk,n)
/*バブルソート*/
     int *kk,n;
{
   /* sort an integer array of size n in the ascending order
    */

   int i,j,ti,*odr,val;

   odr=ivector(1,n);
   for(i=1;i<=n;i++)
      odr[i]=i;

   for(i=1;i<n;i++)
      for(j=i+1;j<=n;j++)
         if(kk[j]<kk[i])
         {
```



```
            ti=kk[i];
            kk[i]=kk[j];
            kk[j]=ti;
            ti=odr[j];
            odr[j]=odr[i];
            odr[i]=ti;
         }
   val=odr[(n+1)/2];
   free_ivector(odr,1,n);
   return(val);
}

/*--------------------------------------------------------------------------*/
void fill_in(img,dimg,sx,sy)
   unsigned char **img,**dimg;
   int sx,sy;
/*メディアンフィルタ*/
{
   int i,j,k,m,dmax,itmp[10],idx;
   unsigned char **td,**ti;

   td=cmatrix(1,sy,1,sx);
   ti=cmatrix(1,sy,1,3*sx);
   if((td==NULL)||(ti==NULL))
   {  printf("Memory not enough. Fail in Fill_in().\n");
      exit(1);
   }
   for(i=1;i<=sy;i++)
     for(j=1;j<=sx;j++)
       td[i][j]=dimg[i][j];
   for(i=1;i<=sy;i++)
     for(j=1;j<=3*sx;j++)
       ti[i][j]=img[i][j];
   for(i=3;i<=sy-2;i++)
     for(j=3;j<=sx-2;j++)
       {
         dmax=0;
         idx=4;
         for(k=(-1);k<=1;k++)
           for(m=(-1);m<=1;m++)
             itmp[(k+1)*3+m+2]=td[i+k][j+m];

         idx = sort(itmp,9);
         idx--;
         dimg[i][j]=itmp[5];
         k=idx/3-1;
         m=(idx%3)-1;
         img[i][3*j-2]=ti[i+k][3*(j+m)-2];
         img[i][3*j-1]=ti[i+k][3*(j+m)-1];
         img[i][3*j]  =ti[i+k][3*(j+m)];
       }
   free_cmatrix(td,1,sy,1,sx);
   free_cmatrix(ti,1,sy,1,3*sx);

 }


/*--------------------------------------------------------------------------*/

void new_depth_map(img_in,dep_in,sx,sy,vx,vy,vz,img_out,dep_out,mark)
     unsigned char **img_in,**dep_in,**img_out,**dep_out,**mark;
     int sx,sy;
     float vx,vy,vz;
{
  int i,j,nx,ny;
  unsigned char **tmpimg;
```

```
  float d,dz,fx,fy;


  /*領域確保*/
  tmpimg=cmatrix(1,sy,1,sx);
  if(tmpimg==NULL)
    {  fprintf(stderr,"Insufficient memory in allocating tmpimg.\n");
       exit(1.0);
     }

  /*初期化*/
  for(i=1;i<=sy;i++)
    for(j=1;j<=sx;j++)
      tmpimg[i][j]=0;

  for(i=1;i<=sy;i++)
    for(j=1;j<=sx;j++)
      {
        d = (float)dep_in[i][j]*SCALE;
        dz = 1.0-d*vz/255.0;

        /*SCALE = ((float)SR/255.) SR = 40,座標変換*/
        fy=(((float)i-SY/2-d*vy)/dz+SY/2);
        fx=(((float)j-SX/2-d*vx)/dz+SX/2);

        /*四捨五入*/
        nx = (int)fx;
        if(fx-(float)nx >= 0.5) {nx++;}
        ny = (int)fy;
        if(fy-(float)ny >= 0.5) {ny++;}

        /*EM = 7,画像の外枠７画素は無視する。*/
        if( nx < 1+EM || nx > sx-EM || ny < 1+EM || ny > sy-EM)
          continue;
        else
          {
            if(dep_in[i][j] > tmpimg[ny][nx])
              {  tmpimg[ny][nx]=dep_in[i][j];    /*コピー*/
                 img_out[ny][3*nx-2]=img_in[i][3*j-2]; /*RGB*/
                 img_out[ny][3*nx-1]=img_in[i][3*j-1];
                 img_out[ny][3*nx  ]=img_in[i][3*j];
               }
          }
      }
  image_write(tmpimg,sx,sy,"test.dpt",PGM);
  image_write(img_out,sx,sy,"test.rgb",PPM);
  for(i=1;i<=sy;i++)
    for(j=1;j<=sx;j++)
      {
        if(tmpimg[i][j] == 0)
          {
            dep_out[i][j]=lintpl(tmpimg,sx,sy,j,i);
            mark[i][j]=ON;
          }
        else
          dep_out[i][j]=tmpimg[i][j];
      }
  free_cmatrix(tmpimg,1,sy,1,sx);
}
/*--------------------------------------------------------------------------*/
void mapping(img_cntr,img_1,img_2,
             dep_cur,mark,sx,sy,vx,vy,vz,intpl_dir,img_out)
     unsigned char **img_cntr,**img_1,**img_2,**dep_cur,**mark,**img_out;
     int sx,sy,intpl_dir;
     float vx,vy,vz;
{
```



```
  int i,j,nx,ny;
  float vvx1,vvy1,vvx2,vvy2,d,dz,pr1,pr2;
  double big_deg,tiny_deg,v_deg,range;
  unsigned char **tmpimg;

  tmpimg=cmatrix(1,sy,1,3*sx);

  /*初期化*/
  for(i=1;i<=sy;i++)
    for(j=1;j<=3*sx;j++)
      tmpimg[i][j]=0;

  switch(intpl_dir){
  case 1:
    vvx1=vx-1.0;
    vvy1=vy;
    break;
  case 2:
    vvx1=vx;
    vvy1=vy+1.0;
    break;
  case 3:
    vvx1=vx+1.0;
    vvy1=vy;
    break;
  case 4:
    vvx1=vx;
    vvy1=vy-1.0;
    break;
  case 5:
    vvx1=vx-1.0;
    vvy1=vy;
    vvx2=vx;
    vvy2=vy+1.0;
    break;
  case 6:
    vvx1=vx;
    vvy1=vy+1.0;
    vvx2=vx+1.0;
    vvy2=vy;
    break;
  case 7:
    vvx1=vx+1.0;
    vvy1=vy;
    vvx2=vx;
    vvy2=vy-1.0;
    break;
  case 8:
    vvx1=vx;
    vvy1=vy-1.0;
    vvx2=vx-1.0;
    vvy2=vy;
    break;
  default:
    fprintf(stderr,"Error! Improper direction of interpolation.\n");
    exit(1.0);
  }
  switch(intpl_dir){
  case 1:
  case 2:
  case 3:
  case 4:
    for(i=1;i<=sy;i++)
      for(j=1;j<=sx;j++)
        {
          if(mark[i][j] == ON)
```

```
          {
            d = (float)dep_cur[i][j]*SCALE;
            d /= (1.0-d*vz/255.0);
            dz = 1.0+d*vz/255.0;
            ny=(int)(((float)i-SY/2+d*vvy1)/dz+SY/2+0.5);
            nx=(int)(((float)j-SX/2+d*vvx1)/dz+SX/2+0.5);
            if( nx < 1+EM || nx > sx-EM || ny < 1+EM || ny > sy-EM)
              mark[i][j]=2;
            else
              {  img_out[i][3*j-2]=img_1[ny][3*nx-2];
                 img_out[i][3*j-1]=img_1[ny][3*nx-1];
                 img_out[i][3*j  ]=img_1[ny][3*nx];
               }
          }
      }
    for(i=1;i<=sy;i++)
      for(j=1;j<=sx;j++)
        if(mark[i][j] == 2)
          {  tmpimg[i][3*j-2]=0;
             tmpimg[i][3*j-1]=0;
             tmpimg[i][3*j]=0;
           }

    /*              intplrgb(img_out,sx,sy,j,i,tmpimg);
     */
    for(i=1;i<=sy;i++)
      for(j=1;j<=sx;j++)
        if(mark[i][j] == 2)
          {  img_out[i][3*j-2]=tmpimg[i][3*j-2];
             img_out[i][3*j-1]=tmpimg[i][3*j-1];
             img_out[i][3*j  ]=tmpimg[i][3*j];
           }
    break;
  case 5:
  case 6:
  case 7:
  case 8:
    for(i=1;i<=sy;i++)
      for(j=1;j<=sx;j++){

        if(mark[i][j] == ON){
          d = (float)dep_cur[i][j]*SCALE;
          d /= (1.0-d*vz/255.0);
          dz = 1.0+d*vz/255.0;
          ny=(int)(((float)i-SY/2+d*vvy1)/dz+SY/2+0.5);
          nx=(int)(((float)j-SX/2+d*vvx1)/dz+SX/2+0.5);

          if( nx < 1+EM || nx > sx-EM || ny < 1+EM || ny > sy-EM)
            mark[i][j]=2;
          else {
            tmpimg[i][3*j-2]=img_1[ny][3*nx-2];
            tmpimg[i][3*j-1]=img_1[ny][3*nx-1];
            tmpimg[i][3*j]  =img_1[ny][3*nx];
          }

          d = (float)dep_cur[i][j]*SCALE;
          d /= (1.0-d*vz/255.0);
          dz = 1.0+d*vz/255.0;
          ny=(int)(((float)i-SY/2+d*vvy2)/dz+SY/2+0.5);
          nx=(int)(((float)j-SX/2+d*vvx2)/dz+SX/2+0.5);

          if( nx < 1+EM || nx > sx-EM || ny < 1+EM || ny > sy-EM){
            if(mark[i][j]==2)
              mark[i][j]=4;
            else
              mark[i][j]=3;
```



```
          }
          else{
            img_out[i][3*j-2]=img_2[ny][3*nx-2];
            img_out[i][3*j-1]=img_2[ny][3*nx-1];
            img_out[i][3*j]  =img_2[ny][3*nx];
          }
        }
      }

    big_deg = atan((double)BIG);
    tiny_deg = atan((double)TINY_A);
    range = big_deg - tiny_deg;
    v_deg = atan2(vy,vx)+PI;

    while(v_deg > PI/2)
    v_deg = v_deg - PI/2;

    pr1 = (big_deg- v_deg)/range;
    pr2 = (v_deg - tiny_deg)/range;

    fprintf(stderr,"big_deg = %f \n  tiny_deg = %f \n  range  = %f \n  v_deg  = %f \n
  pr1 = %f\n   pr2 = %f \n",big_deg*180/PI,tiny_deg*180/PI,range*180/PI,v_deg*180/PI,
pr1,pr2);
    for(i=1;i<=sy;i++)
      for(j=1;j<=sx;j++){
        switch(mark[i][j]){

        case 1:
          img_out[i][3*j-2]=(unsigned char)
            ((int)tmpimg[i][3*j-2]*pr1+(int)img_out[i][3*j-2]*pr2);
          img_out[i][3*j-1]=(unsigned char)
            ((int)tmpimg[i][3*j-1]*pr1+(int)img_out[i][3*j-1]*pr2);
          img_out[i][3*j]  =(unsigned char)
            ((int)tmpimg[i][3*j  ]*pr1+(int)img_out[i][3*j]*pr2);
          break;

        case 2:
          break;

        case 3:
          img_out[i][3*j-2]=tmpimg[i][3*j-2];
          img_out[i][3*j-1]=tmpimg[i][3*j-1];
          img_out[i][3*j]  =tmpimg[i][3*j];
          break;

        case 4:
          img_out[i][3*j-2]=0;
          img_out[i][3*j-1]=0;
          img_out[i][3*j]  =0;
          /*      intplrgb(img_out,sx,sy,j,i,img_out);
           */                  break;
        default:
          ;
        }
      }
    }
  free_cmatrix(tmpimg,1,sy,1,3*sx);
}
/*-------------------------------------------------------------------------*/

void camera_rotate(in_img,out_img,rx,ry,rz,zoom,color)
unsigned char **in_img,**out_img;
float rx,ry,rz,zoom;
int color;
{
  Rotate cc;
```

```
  double sa,ca,sb,cb,sg,cg;
  double P,Q,R;

  double xo,yo;
  int i,j;

  SINCOS (rx, sa, ca);
  SINCOS (ry, sb, cb);
  SINCOS (rz, sg, cg);

  cc.r1 = cb * cg;
  cc.r2 = cg * sa * sb - ca * sg;
  cc.r3 = sa * sg + ca * cg * sb;
  cc.r4 = cb * sg;
  cc.r5 = sa * sb * sg + ca * cg;
  cc.r6 = ca * sb * sg - cg * sa;
  cc.r7 = -sb;
  cc.r8 = cb * sa;
  cc.r9 = ca * cb;

  if(color == 0)
    for(i=1;i<=SX;i++)
      for(j=1;j<=SY;j++){
        out_img[j][i] = 0;
      }

  else if(color == 1){
    for(i=1;i<=SX;i++)
      for(j=1;j<=SY;j++){
        out_img[j][3*i-2] = 0 ;
        out_img[j][3*i-1] = 0 ;
        out_img[j][3*i  ] = 0 ;
      }
  }

  for(i=1;i<=SX;i++)
    for(j=1;j<=SY;j++){
      i -= SX/2;
      j -= SY/2;

      P=cc.r1 * i  +  cc.r2 * j  +  cc.r3 * FI;
      Q=cc.r4 * i  +  cc.r5 * j  +  cc.r6 * FI;
      R=cc.r7 * i  +  cc.r8 * j  +  cc.r9 * FI;
      xo = 1.0/zoom * FI * P / R;
      yo = 1.0/zoom * FI * Q / R;

      i += SX/2;
      j += SY/2;
      xo += SX/2;
      yo += SY/2;

      if((int)xo > 0 && (int)xo <= SX && (int)yo > 0 && (int)yo <= SY){
        if(color == 0)
          out_img[j][i] = in_img[(int)yo][(int)xo];
        else if(color == 1){
          out_img[j][3*i-2] = in_img[(int)yo][3*(int)xo-2];
          out_img[j][3*i-1] = in_img[(int)yo][3*(int)xo-1];
          out_img[j][3*i  ] = in_img[(int)yo][3*(int)xo  ];
        }
      }
    }
}

main(argc,argv)
     int argc;
     char *argv[];
```



```
{

  FILE *fp;

  char c,str[50];
  int i,j,new_key,sx=SX,sy=SY,sxf=SX,syf=SY,*img_type,cx,cy,intpl_dir;
  /*SX = 640,SY = 480 in vpmparas.h*/
  float vx,vy,vz,rx,ry,rz;
  float zoom=1.0;

  if(argc<11 || argc>11) show_usage(argv[0]);
  /*(cx,cy) center camera coodinate(camera matrix 8x8)*/
  cx=atoi(argv[2]);
  cy=atoi(argv[3]);

  /*(vx,vy,vz) view point move.each -1.0~1.0 */
  vx = (float)atof(argv[5]);
  vy = (float)atof(argv[6]);
  vz = (float)atof(argv[7]);

  /*(rx,ry,rz) view point rotate.*/
  rx = (float)atof(argv[8]) * PAI/180.0;
  ry = (float)atof(argv[9]) * PAI/180.0;
  rz = (float)atof(argv[10])* PAI/180.0;

  img_type   = ivector(0,NL);   /*NL=5 in header*/
  img_frnt   = cmatrix(1,SY,1,3*SX);  /*Center Image*/
  img_bak1   = cmatrix(1,SY,1,3*SX);  /*この画像よりマッピングを行なう1*/
  img_bak2   = cmatrix(1,SY,1,3*SX);  /*この画像よりマッピングを行なう2*/
  img_outp   = cmatrix(1,SY,1,3*SX);
  dep_cntr   = cmatrix(1,SY,1,SX);    /*Depth Image*/
  dep_outp   = cmatrix(1,SY,1,SX);
  mark_intpl = cmatrix(1,SY,1,SX);  /*座標変換によって得られなかった画素*/

  sprintf(str,"%s_%d_%d",argv[1],cx,cy);
  image_read(img_frnt,&sxf,&syf,str,&img_type[FG]);  /*FG = 0*/

  /*Image size*/
  sx=SX;
  sy=SY;

  image_read(dep_cntr,&sx,&sy,argv[4],&img_type[DP]); /*DP = 1*/

  do
    {
      CPX = vx;
      CPY = vy;
      CPZ = vz;

      new_depth_map(img_frnt,dep_cntr,
                    sx,sy,vx,vy,vz,img_outp,dep_outp,mark_intpl);

      ykh(dep_outp,sx,sy,WS,1,dep_cntr);
      ykh(dep_cntr,sx,sy,WS,1,dep_outp);
      ykh(dep_outp,sx,sy,WS,1,dep_cntr);

      switch(intpl_dir=select_direction(vx,vy))
        {
        case 1:
          sprintf(str,"%s_%d_%d",argv[1],cx+1,cy);
          image_read(img_bak1,&sxf,&syf,str,&img_type[FG]);
          break;
        case 2:
          sprintf(str,"%s_%d_%d",argv[1],cx,cy-1);
          image_read(img_bak1,&sxf,&syf,str,&img_type[FG]);
          break;
```

```
        case 3:
          sprintf(str,"%s_%d_%d",argv[1],cx-1,cy);
          image_read(img_bak1,&sxf,&syf,str,&img_type[FG]);
          break;
        case 4:
          sprintf(str,"%s_%d_%d",argv[1],cx,cy+1);
          image_read(img_bak1,&sxf,&syf,str,&img_type[FG]);
          break;
        case 5:
          sprintf(str,"%s_%d_%d",argv[1],cx+1,cy);
          image_read(img_bak1,&sxf,&syf,str,&img_type[FG]);
          sprintf(str,"%s_%d_%d",argv[1],cx,cy-1);
          image_read(img_bak2,&sxf,&syf,str,&img_type[FG]);
          break;
        case 6:
          sprintf(str,"%s_%d_%d",argv[1],cx,cy-1);
          image_read(img_bak1,&sxf,&syf,str,&img_type[FG]);
          sprintf(str,"%s_%d_%d",argv[1],cx-1,cy);
          image_read(img_bak2,&sxf,&syf,str,&img_type[FG]);
          break;
        case 7:
          sprintf(str,"%s_%d_%d",argv[1],cx-1,cy);
          image_read(img_bak1,&sxf,&syf,str,&img_type[FG]);
          sprintf(str,"%s_%d_%d",argv[1],cx,cy+1);
          image_read(img_bak2,&sxf,&syf,str,&img_type[FG]);
          break;
        case 8:
          sprintf(str,"%s_%d_%d",argv[1],cx,cy+1);
          image_read(img_bak1,&sxf,&syf,str,&img_type[FG]);
          sprintf(str,"%s_%d_%d",argv[1],cx+1,cy);
          image_read(img_bak2,&sxf,&syf,str,&img_type[FG]);
          break;
        }
      mapping(img_frnt,img_bak1,img_bak2,dep_cntr,mark_intpl,
              sx,sy,vx,vy,vz,intpl_dir,img_outp);
      fprintf(stderr,"!! mapping end !! \n");

      camera_rotate(img_outp,img_frnt,rx,ry,rz,zoom,1);
      camera_rotate(dep_outp,dep_cntr,rx,ry,rz,zoom,0);

      image_write(img_frnt,sxf,syf,
                  "../tmp/tmp.rgb",img_type[FG]);
      image_write(dep_cntr,sxf,syf,"../tmp/tmp.dpt",PGM);

      sprintf(str,"imgview ../tmp/tmp.rgb &");
      system(str);
      sprintf(str,"imgview ../tmp/tmp.dpt &");
      system(str);

      fprintf(stderr,"Current Virtual Camera Position= (%5.2f,%5.2f,%5.2f)-(%5.2f,%5.
2f,%5.2f).\n",vx,vy,vz,rx,ry,rz);
      fprintf(stderr,"If you want to save the composed image, type \'s\'.\n");
      fprintf(stderr,"                terminate this program , type \'q\'.\n");
      fprintf(stderr,"                or press [return] to continue.\n");
      fflush(stdin);
      scanf("%c",&c);

      if(c == 's')
        {
          sprintf(str,"%s_%d_%d_%02d_%02d_%02d",argv[1],cx,cy,(int)(vx*100),(int)(vy*
100),(int)(vz*100));
          image_write(img_outp,sxf,syf,str,img_type[FG]);
          sprintf(str,"%s.depth",str);
          image_write(dep_outp,sxf,syf,str,PGM);
        }
```



```
      if(c == 'q')
        break;

      fprintf(stderr,"Insert a new virtual camera coordinate.\n");
      fflush(stdin);
      scanf("%f %f %f %f %f %f",&vx,&vy,&vz,&rx,&ry,&rz);

      image_read(dep_cntr,&sx,&sy,argv[4],&img_type[DP]);
      clear_cmatrix(img_bak1,1,SY,1,3*SX);
      clear_cmatrix(img_bak2,1,SY,1,3*SX);
      clear_cmatrix(img_outp,1,SY,1,3*SX);
      clear_cmatrix(dep_outp,1,SY,1,SX);
      clear_cmatrix(mark_intpl,1,SY,1,SX);
    } while(1);

  free_ivector(img_type,0,NL);
  free_cmatrix(img_frnt,1,SY,1,3*SX);
  free_cmatrix(img_bak1,1,SY,1,3*SX);
  free_cmatrix(img_bak2,1,SY,1,3*SX);
  free_cmatrix(img_outp,1,SY,1,3*SX);
  free_cmatrix(dep_cntr,1,SY,1,SX);
  free_cmatrix(dep_outp,1,SY,1,SX);
  free_cmatrix(mark_intpl,1,SY,1,SX);
}
```

```
#include <stdio.h>

void image_read(image, xsize, ysize, filename)
   unsigned char **image;
   int xsize;
   int ysize;
   char *filename;
{
   FILE *fp;

   int i;

   if ((fp = fopen(filename, "rb")) == NULL)
   {
      fprintf(stderr,"Error in reading %s. \n",filename);
      exit(-1);
   }
   for(i=1;i<=ysize;i++)
      fread(image[i]+1,sizeof(char),(xsize),fp);
   fclose(fp);
}

void image_write(image, xsize, ysize, filename)
   unsigned char **image;
   int xsize;
   int ysize;
   char *filename;
{
   FILE *fp;

   int i;

   if ((fp = fopen(filename, "wb")) == NULL)
   {
      fprintf(stderr,"Error in writing %s.\n",filename);
      exit(-1);
   }
   for(i=1;i<=ysize;i++)
      fwrite(image[i]+1,sizeof(char),(xsize),fp);
   fclose(fp);
}
```

```
/* This file contains image read/write functions.
 * PGM, PPM formats are detected automatically and processed properly.
 * For RAW, CRAW formats, xsize, ysize should be given.
 * Example of calling these routines
 *      image_read(image, &xsize, &ysize, filename, &filetype);
 *      image_write(image, xsize, ysize, filename, filetype);
 * Programmed by Jong-Il Park on June 11, 1996.
 * Revision Report: Support SGI format(Oct.30, 1996)
 */

#include <stdio.h>
#include "../inc/nr_utl.h"
#include "../inc/img_macro.h"

#define MAX_HEADER 10000

/*-------------------------------------------------------------------------*/
unsigned char rgb2y(r,g,b)
   unsigned char r,g,b;
{
   return((unsigned char)(0.3*r+0.59*g+0.11*b));
}
/*-------------------------------------------------------------------------*/
void image_read(image, xsize, ysize, filename, filetype)
   unsigned char **image;
   int *xsize;
   int *ysize;
   char *filename;
   int *filetype;
{
   FILE *fp;

   int i,j,c,max_value,sxt,syt,num_read;
   short magic;
   unsigned char **iap,**iop,dummy[MAX_HEADER];

   if ((fp = fopen(filename, "r")) == NULL)
   {
      fprintf(stderr,"Can't open %s. \n",filename);
      exit(-1);
   }
   if( (c=fgetc(fp)) == 'P')
   {  if( (c=fgetc(fp)) == '5')   /*---------- PGM ----------*/
      {
         *filetype=PGM;
         if( (c=fgetc(fp)) != '\n' && c != ' ')
         {  fprintf(stderr,"Error in Reading Image Header. Check Header of %s\n",file
name);
            exit(1);
         }
         while(1)
         {
            if( (c=fgetc(fp)) == '#')
               while( (c=fgetc(fp)) != '\n' )
                  ;
            else
            {  fseek (fp , -1L, SEEK_CUR);
               break;
            }
         }
         fscanf(fp,"%d %d",xsize,ysize);
         fscanf(fp,"%d",&max_value);
         fseek (fp , 1L, SEEK_CUR);
         for(i=1;i<=*ysize;i++)
            if(*xsize != (num_read=fread(image[i]+1,sizeof(char),(*xsize),fp)))
            {
```



```
               fprintf(stderr,"Error in reading %s %dth row(%d byte read).\n",filenam
e,i,num_read);
               exit(1);
            }
         if((c=fgetc(fp)) !=  EOF)
         {
            fprintf(stderr,"File size not match in reading %s.\n",filename);
            exit(1);
         }
      }
      else if(c == '6')           /*---------- PPM ----------*/
      {
         *filetype=PPM;
         if( (c=fgetc(fp)) != '\n' && c != ' ')
         {  fprintf(stderr,"Error in Reading Image Header. Check Header of %s\n",file
name);
            exit(1);
         }
         while(1)
         {
            if( (c=fgetc(fp)) == '#')
               while( (c=fgetc(fp)) != '\n' )
                  ;
            else
            {  fseek (fp , -1L, SEEK_CUR);
               break;
            }
         }
         fscanf(fp,"%d %d",xsize,ysize);
         fscanf(fp,"%d",&max_value);
         fseek (fp , 1L, SEEK_CUR);
         for(i=1;i<=*ysize;i++)
            if((*xsize*3) != (num_read=fread(image[i]+1,sizeof(char),(*xsize*3),fp)))
            {
               fprintf(stderr,"Error in reading %s %dth row(%d byte read).\n",filenam
e,i,num_read);
               exit(1);
            }
         if((c=fgetc(fp)) !=  EOF)
         {
            fprintf(stderr,"File size not match in reading %s.\n",filename);
            exit(1);
         }
      }
/* Need modification below for CRAW format */

      else                        /*---------- RAW ----------*/
      {
         *filetype=RAW;
         fseek (fp , 0L, SEEK_SET);
         for(i=1;i<=*ysize;i++)
            if(*xsize != (num_read=fread(image[i]+1,sizeof(char),(*xsize),fp)))
            {
               fprintf(stderr,"Error in reading %s %dth row(%d byte read).\n",filenam
e,i,num_read);
               exit(1);
            }
         if((c=fgetc(fp)) !=  EOF)
         {
            fprintf(stderr,"File size not match in reading %s.\n",filename);
            exit(1);
         }
      }
   }
   else
   {
```

```
      fseek (fp , 0L, SEEK_SET);
      fread(&magic,sizeof(short),1,fp);
      if(magic == 474)         /*---------- SGI ----need to modify---*/
      {

         *filetype=SGI;

{         char Storage, Bpc;
         unsigned short Dimension,XSize,YSize,ZSize;
         long PixMin,PixMax;
         char Dummy1[4],ImageName[80];
         long ColorMap;
         char Dummy2[404];

         fread(&Storage,sizeof(char),1,fp);
printf("Storage=%d\n",Storage);
         if(Storage != 0x00)
         {
            fprintf(stderr,"Can't cope with compressed SGI format in reading %s.\n",f
ilename,i,num_read);
            exit(1);
         }

         fread(&Bpc,sizeof(char),1,fp);
         if(Bpc != 0x01)
         {
            fprintf(stderr,"Can't cope with 2B/pel SGI format in reading %s.\n",filen
ame,i,num_read);
            exit(1);
         }
)
         fread(dummy,sizeof(char),508,fp);
         iap=cmatrix(1,*ysize*4,1,*xsize);
         for(i=0;i<*ysize;i++)
            if(*xsize != (num_read=fread(iap[*ysize*2-i]+1,sizeof(char),(*xsize),fp))
)
               {
                  fprintf(stderr,"Error in reading %s %dth row(%d byte read).\n",filenam
e,i,num_read);
                  exit(1);
               }
         for(i=0;i<*ysize;i++)
            if(*xsize != (num_read=fread(iap[*ysize*3-i]+1,sizeof(char),(*xsize),fp))
)
               {
                  fprintf(stderr,"Error in reading %s %dth row(%d byte read).\n",filenam
e,i,num_read);
                  exit(1);
               }
         for(i=0;i<*ysize;i++)
            if(*xsize != (num_read=fread(iap[*ysize*4-i]+1,sizeof(char),(*xsize),fp))
)
               {
                  fprintf(stderr,"Error in reading %s %dth row(%d byte read).\n",filenam
e,i,num_read);
                  exit(1);
               }
         if((c=fgetc(fp)) !=  EOF)
         {
            fprintf(stderr,"File size not match in reading %s.\n",filename);
            exit(1);
         }
         for(i=1;i<=*ysize;i++)
         for(j=1;j<=*xsize;j++)
            image[i][j]=rgb2y(iap[*ysize+i][j],iap[*ysize*2+i][j],iap[*ysize*3+i][j])
;
```



```
         free_cmatrix(iap,1,*ysize*4,1,*xsize);

      }
      else                        /*---------- RAW ----------*/
      {
         *filetype=RAW;
         fseek (fp , 0L, SEEK_SET);
         for(i=1;i<=*ysize;i++)
            if(*xsize != (num_read=fread(image[i]+1,sizeof(char),(*xsize),fp)))
            {
               fprintf(stderr,"Error in reading %s %dth row(%d byte read).\n",filenam
e,i,num_read);
               exit(1);
            }
         if((c=fgetc(fp)) !=  EOF)
         {
            fprintf(stderr,"File size not match in reading %s.\n",filename);
            exit(1);
         }
      }
   }
   fclose(fp);
}
/*-------------------------------------------------------------------------------*/
void image_write(image, xsize, ysize, filename, filetype)
   unsigned char **image;
   int xsize;
   int ysize;
   char *filename;
   int filetype;
{
   FILE *fp;

   int i,max_value=MAXVALUE;

   if ((fp = fopen(filename, "w")) == NULL)
   {
      fprintf(stderr,"Error in writing %s.\n",filename);
      exit(-1);
   }
   switch(filetype)
   {
      case RAW:
         for(i=1;i<=ysize;i++)
            fwrite(image[i]+1,sizeof(char),(xsize),fp);
         break;
      case PGM:
         fprintf(fp,"P5\n");
         fprintf(fp,"%d %d\n",xsize,ysize);
         fprintf(fp,"%d\n",max_value);
         for(i=1;i<=ysize;i++)
            fwrite(image[i]+1,sizeof(char),(xsize),fp);
         break;
      case PPM:
         fprintf(fp,"P6\n");
         fprintf(fp,"%d %d\n",xsize,ysize);
         fprintf(fp,"%d\n",max_value);
         for(i=1;i<=ysize;i++)
            fwrite(image[i]+1,sizeof(char),(xsize*3),fp);
         break;
      case CRAW:
         for(i=1;i<=ysize;i++)
            fwrite(image[i]+1,sizeof(char),(xsize*3),fp);
         break;
   }
   fclose(fp);
```

```
}
```

```
#include <math.h>
#include <stdio.h>
#include <malloc.h>

#define NR_END          1
#define FREE_ARG        char*

void    nrerror(errtxt)
        char    errtxt[];
{
        fprintf(stderr,"Numerical Recipe Run-time Error...\n");
        fprintf(stderr,"%s\n",errtxt);
        fprintf(stderr,"...now exiting to system...\n");
        exit(1);
}
float   *vector(nl,nh)
        int     nl,nh;
{
        float   *v;

        v = (float *)malloc((unsigned)(nh-nl+1+NR_END)*sizeof(float));
        if(!v) nrerror("Allocation failure in VECTOR()");
        return(v-nl+NR_END);
}
void    free_vector(v,nl,nh)
        float   *v;
        int     nl,nh;
{
        free((char *)(v+nl-NR_END));
}
unsigned char   *cvector(nl,nh)
        int     nl,nh;
{
        unsigned char   *v;

        v = (unsigned char *)malloc((unsigned)(nh-nl+1+NR_END)*sizeof(unsigned char))
;
        if(!v) nrerror("Allocation failure in CVECTOR()");
        return(v-nl+NR_END);
}
void    free_cvector(v,nl,nh)
        char    *v;
        int     nl,nh;
{
        free((char *)(v+nl-NR_END));
}
int     *ivector(nl,nh)
        int     nl,nh;
{
        int     *v;

        v = (int *)malloc((unsigned)(nh-nl+1+NR_END)*sizeof(int));
        if(!v) nrerror("Allocation failure in IVECTOR()");
        return(v-nl+NR_END);
}
void    free_ivector(v,nl,nh)
        int     *v;
        int     nl,nh;
{
        free((char *)(v+nl-NR_END));
}
unsigned long   *lvector(nl,nh)
        long    nl,nh;
{
        unsigned long   *v;
```



```
        v = (unsigned long *)
                malloc((unsigned)(nh-nl+1+NR_END)*sizeof(unsigned long));
        if(!v) nrerror("Allocation failure in LVECTOR()");
        return(v-nl+NR_END);
}
void    free_lvector(v,nl,nh)
        unsigned long   *v;
        long    nl,nh;
{
        free((char *)(v+nl-NR_END));
}
double  *dvector(nl,nh)
        int     nl,nh;
{
        double  *v;

        v = (double *)malloc((unsigned)(nh-nl+1+NR_END)*sizeof(double));
        if(!v) nrerror("Allocation failure in DVECTOR()");
        return(v-nl+NR_END);
}
void    free_dvector(v,nl,nh)
        double  *v;
        int     nl,nh;
{
        free((char *)(v+nl-NR_END));
}
float   **matrix(nrl,nrh,ncl,nch)
        int     nrl,nrh,ncl,nch;
{
        int     i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1;
        float   **m;

        m = (float **)malloc((unsigned)(nrh-nrl+1+NR_END)*sizeof(float*));
        if(!m) nrerror("Allocation failure in MATRIX()");
        m += NR_END;
        m -= nrl;

        i=nrl;
        m[i] = (float *)malloc
                ((unsigned)(nrow*ncol+NR_END)*sizeof(float));
        if(!m[i]) nrerror("Allocation failure 2 in MATRIX()");
        m[i] += NR_END;
        m[i] -= ncl;

        for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+nch-ncl+1;

        return(m);
}
double  **dmatrix(nrl,nrh,ncl,nch)
        int     nrl,nrh,ncl,nch;
{
        int     i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1;
        double  **m;

        m = (double **)malloc((unsigned)(nrh-nrl+1+NR_END)*sizeof(double*));
        if(!m) nrerror("Allocation failure in DMATRIX()");
        m += NR_END;
        m -= nrl;

        i=nrl;
        m[i] = (double *)malloc
                ((unsigned)(nrow*ncol+NR_END)*sizeof(double));
        if(!m[i]) nrerror("Allocation failure 2 in DMATRIX()");
        m[i] += NR_END;
        m[i] -= ncl;
```

```
	for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+nch-ncl+1;

	return(m);
}
unsigned char   **cmatrix(nrl,nrh,ncl,nch)
	int     nrl,nrh,ncl,nch;
{
	int     i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1;
	unsigned char   **m;

	m = (unsigned char **)malloc
			((unsigned)(nrh-nrl+1+NR_END)*sizeof(char*));
	if(!m) nrerror("Allocation failure in CMATRIX()");
	m += NR_END;
	m -= nrl;

	i=nrl;
	m[i] = (unsigned char *)malloc
		((unsigned)(nrow*ncol+NR_END)*sizeof(char));
	if(!m[i]) nrerror("Allocation failure 2 in CMATRIX()");
	m[i] += NR_END;
	m[i] -= ncl;

	for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+nch-ncl+1;

	return(m);
}
int     **imatrix(nrl,nrh,ncl,nch)
	int     nrl,nrh,ncl,nch;
{
	int     i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1;
	int     **m;

	m = (int **)malloc((unsigned)(nrh-nrl+1+NR_END)*sizeof(int*));
	if(!m) nrerror("Allocation failure in IMATRIX()");
	m += NR_END;
	m -= nrl;

	i=nrl;
	m[i] = (int *)malloc
		((unsigned)(nrow*ncol+NR_END)*sizeof(int));
	if(!m[i]) nrerror("Allocation failure 2 in IMATRIX()");
	m[i] += NR_END;
	m[i] -= ncl;

	for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+nch-ncl+1;

	return(m);
}
double  ***dtensor(nrl,nrh,ncl,nch,ndl,ndh)
	int     nrl,nrh,ncl,nch,ndl,ndh;
{
	int     i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1,ndep=ndh-ndl+1;
	double  ***t;

	t = (double ***)malloc((unsigned)(nrow+NR_END)*sizeof(double**));
	if(!t) nrerror("allocation failure 1 in dtensor().");
	t += NR_END;
	t -= nrl;

	t[nrl] = (double **)malloc
			((unsigned)(nrow*ncol+NR_END)*sizeof(double*));
	if(!t[nrl]) nrerror("allocation failure 2 in dtensor().");
	t[nrl] += NR_END;
	t[nrl] -= ncl;
```



```
	t[nrl][ncl] =
		(double *)malloc
			((unsigned)(nrow*ncol*ndep+NR_END)*sizeof(double));
	if(!t[nrl][ncl]) nrerror("allocation failure 3 in dtensor().");
	t[nrl][ncl] += NR_END;
	t[nrl][ncl] -= ndl;

	for(j=ncl+1;j<=nch;j++) t[nrl][j]=t[nrl][j-1]+ndep;
	for(i=nrl+1;i<=nrh;i++)
	{
		t[i]=t[i-1]+ncol;
		t[i][ncl]=t[i-1][ncl]+ncol*ndep;
		for(j=ncl+1;j<=nch;j++) t[i][j]=t[i][j-1]+ndep;
	}
	return(t);
}
unsigned char   ***ctensor(nrl,nrh,ncl,nch,ndl,ndh)
	int     nrl,nrh,ncl,nch,ndl,ndh;
{
	int     i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1,ndep=ndh-ndl+1;
	unsigned char   ***t;

	t = (unsigned char ***)malloc((unsigned)(nrow+NR_END)*sizeof(char**));
	if(!t) nrerror("allocation failure 1 in ctensor().");
	t += NR_END;
	t -= nrl;

	t[nrl] = (unsigned char **)malloc
		((unsigned)(nrow*ncol+NR_END)*sizeof(char*));
	if(!t[nrl]) nrerror("allocation failure 2 in ctensor().");
	t[nrl] += NR_END;
	t[nrl] -= ncl;

	t[nrl][ncl] =
		(unsigned char *)malloc
		((unsigned)(nrow*ncol*ndep+NR_END)*sizeof(char));
	if(!t[nrl][ncl]) nrerror("allocation failure 3 in ctensor().");
	t[nrl][ncl] += NR_END;
	t[nrl][ncl] -= ndl;

	for(j=ncl+1;j<=nch;j++) t[nrl][j]=t[nrl][j-1]+ndep;
	for(i=nrl+1;i<=nrh;i++)
	{
		t[i]=t[i-1]+ncol;
		t[i][ncl]=t[i-1][ncl]+ncol*ndep;
		for(j=ncl+1;j<=nch;j++) t[i][j]=t[i][j-1]+ndep;
	}
	return(t);
}
void    free_dtensor(t,nrl,nrh,ncl,nch,ndl,ndh)
	double  ***t;
	int     nrl,nrh,ncl,nch,ndl,ndh;
{
	free((FREE_ARG) (t[nrl][ncl]+ndl-NR_END));
	free((FREE_ARG) (t[nrl]+ncl-NR_END));
	free((FREE_ARG) (t+nrl-NR_END));
}
void    free_ctensor(t,nrl,nrh,ncl,nch,ndl,ndh)
	unsigned char   ***t;
	int     nrl,nrh,ncl,nch,ndl,ndh;
{
	free((FREE_ARG) (t[nrl][ncl]+ndl-NR_END));
	free((FREE_ARG) (t[nrl]+ncl-NR_END));
	free((FREE_ARG) (t+nrl-NR_END));
}
void    free_matrix(m,nrl,nrh,ncl,nch)
```

```
        float   **m;
        int     nrl,nrh,ncl,nch;
{
        free((char *)(m[nrl]+ncl-NR_END));
        free((char *)(m+nrl-NR_END));
}
void    free_dmatrix(m,nrl,nrh,ncl,nch)
        double  **m;
        int     nrl,nrh,ncl,nch;
{
        free((char *)(m[nrl]+ncl-NR_END));
        free((char *)(m+nrl-NR_END));
}
void    free_cmatrix(m,nrl,nrh,ncl,nch)
        unsigned char   **m;
        int     nrl,nrh,ncl,nch;
{
        free((char *)(m[nrl]+ncl-NR_END));
        free((char *)(m+nrl-NR_END));
}
void    free_imatrix(m,nrl,nrh,ncl,nch)
        int     **m;
        int     nrl,nrh,ncl,nch;
{
        free((char *)(m[nrl]+ncl-NR_END));
        free((char *)(m+nrl-NR_END));
}
double  atod(s)
        char    s[];
{
        double  val,power;
        int     i,sign;

        for(i=0; s[i]==' ' || s[i]=='\n' || s[i]=='\t'; i++)
                ;
        sign  = 1;
        if( s[i]=='+' || s[i]=='-' )
                sign = (s[i++]=='+') ?  1 : -1;
        for(val=0.0; s[i] >='0' &&  s[i] <= '9'; i++)
                val = 10.0*val  + (double)(s[i] - '0');
        if(s[i]=='.')
                i++;
        for(power=1.0;  s[i]>='0'  &&   s[i]<='9'; i++)
        {       val =  10.0*val + (double)(s[i] - '0');
                power *=  10.0;
        }
        return((double)sign*val/power);
}
void clear_cmatrix(img,rl,ru,cl,cu)
   unsigned char **img;
   int rl,ru,cl,cu;
{
   int i,j;

   for(i=rl;i<=ru;i++)
   for(j=cl;j<=cu;j++)
      img[i][j]=0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include "../inc/nr_utl.h"


/*-------------------------------------------------------------------------*/
void sorta(kk,nl,nu)
   int *kk,nl,nu;
{
   /* sort an integer array of size n in the ascending order
    */

   int i,j,ti;

   for(i=nl;i<nu;i++)
      for(j=i+1;j<=nu;j++)
         if(kk[j]<kk[i])
         {
            ti=kk[i];
            kk[i]=kk[j];
            kk[j]=ti;
         }
}
/*-------------------------------------------------------------------------*/
int get_minimum(kk,nl,nu)
   int *kk,nl,nu;
{
   int i,min;

   min=nl;
   for(i=nl+1;i<=nu;i++)
      if(kk[min]>kk[i])
         min=i;
   return(min);
}
/*-------------------------------------------------------------------------*/
int os_avrg(kk,nl,nu)
   int *kk,nl,nu;
{
   int i,avg;

   avg=0;;
   for(i=nl+1;i<nu;i++)
      avg += kk[i];
   avg = (int)((float)avg/(nu-nl-1)+0.5);
   return(avg);
}
/*-------------------------------------------------------------------------*/
void ykh(img,sx,sy,ws,method,img_o)
   unsigned char **img,**img_o;
   int sx,sy,ws,method;
{
   int i,j,k,n,hws=ws/2;
   int **wk,*msr,min_n;

   wk=imatrix(1,4,1,ws);
   msr=ivector(1,4);

   for(i=1;i<=sy;i++)
   for(j=1;j<=sx;j++)
   {

      if(j <= hws || j > sx-hws || i <= hws || i > sy-hws)
      {  img_o[i][j]=img[i][j];
         continue;
      }
```



```
      for(k=1;k<=ws;k++)
      {
         wk[1][k]=img[i][j-hws-1+k];
         wk[2][k]=img[i-hws-1+k][j];
         wk[3][k]=img[i-hws-1+k][j-hws-1+k];
         wk[4][k]=img[i-hws-1+k][j+hws+1-k];
      }
      for(n=1;n<=4;n++)
      {
         sorta(wk[n],1,ws);
         msr[n]=wk[n][ws-1]-wk[n][2];
      }
      min_n=get_minimum(msr,1,4);

      switch(method)
      {
        case 1: img_o[i][j]=(unsigned char)wk[min_n][hws+1];
                break;
        case 2: img_o[i][j]=(unsigned char)os_avrg(wk[min_n],1,ws);
                break;
        default:img_o[i][j]=(unsigned char)wk[min_n][hws+1];
                break;
      }
   }
   free_ivector(msr,1,4);
   free_imatrix(wk,1,4,1,ws);
}
```

```
#define RAW 0
#define CRAW 1
#define PGM 2
#define PPM 3
#define SGI 474
#define MAXVALUE 255
#define MAX_SX 720
#define MAX_SY 540
```

```
#define RAW 0
#define CRAW 1
#define PGM 2
#define PPM 3
#define MAXVALUE 255
#define MAX_SX 720
#define MAX_SY 540

void image_read(),image_write();
```

```
void    nrerror(),free_vector(),free_cvector(),free_ivector(),free_lvector(),
        free_dvector(),free_matrix(),free_cmatrix(),
        free_dmatrix(),free_imatrix(),free_dtensor(),free_ctensor(),
        clear_cmatrix();
int     *ivector(),**imatrix();
float   *vector(),**matrix();
double  *dvector(),**dmatrix(),***dtensor();
unsigned char   ***ctensor(),**cmatrix(),*cvector();
unsigned long   *lvector();
double  atod();
```

```
void sorta(),ykh();
int get_minimum(),os_avrg();
```

```
float ran1(),gasdev();
```