〔非公開〕

ＴＲ－Ｍ－０００６

イメージベース仮想環境生成の研究

画像系列からのパノラマ画像の生成

西羅　光　Hikaru NISHIRA
間瀬　健二　Kenji MASE

１９９６．４．４

ＡＴＲ知能映像通信研究所

イメージベース仮想環境生成の研究

画像系列からのパノラマ画像の作成

(株)ATR 知能映像通信研究所 第 2 研究室

西羅 光　　間瀬 健二

目次

図目次

1 はじめに

仮想的に3次元空間を構成し、その中の空間を自由に行き来できるようなシステムを構成することを考えるとき、その構成には二つの方法が考えられる。

一つはコンピューターグラフィックスを用いるもので、仮想的な環境のデータ（形、色、大きさなど）を用意しておき、視点を変えたときの画像をコンピューターによる計算で求め表示していくものであり、HMD（ヘッドマウントディスプレイ）を使ったヴァーチャルリアリティーのシステムなどはテレビなどでも紹介され一般的にも広く知られている。この方式はどのような視点の画像でも表示でき、物の形などのデータも、単純なものであれば、それほど大きなものにはならないという利点がある一方で、計算機による高速な処理が要求されるので、実画像に匹敵するほどリアルな画像は現在のところ扱うことはできない。また画像処理専用のプロセッサーなどハードウエアの面で大掛かりで高額なものが必要になるなどの欠点がある。

もうひとつの方法にカメラなどで撮影した実画像をもとに仮想空間を構成するという方法がある。この方法の場合、視点の方向が限られることや、大きな画像データを何フレームも用意しておかなければならないなどの欠点はあるものの、よりリアルな画像を手軽に表示できるという利点がある。この方式を利用したものとして、すでにQuicktimeVR（アップル社）などの商品も発売されている。この方式の場合、パノラマ画像を作ることが共通の課題になってくる。

今回の実習では実画像系列からパノラマ画像を作るアルゴリズムの実現を目標に、いくつかの手法を用いて画像の合成を試みた。

2 手法

2.1 概要

画像系列の合成にあたっては、各フレーム間の関係を調べ, その上で適切な合成手法を用いて各フレームを貼り合わせていく必要がある。

今回の実習では静止画像をカメラの位置を変えながら撮った画像の合成を考えているので、各フレームはある一定の変換式で結ばれていると仮定することができる。このときフレーム間の関係は、変換パラメタを用いて記述することができる。そこで変換パラメタを求めることが問題になってくるが、今回Levenberg-Marquardt methodという多変数関数の最小化の手法を用いてパラメタの推定を行ってみた。

変換パラメタが求まると、次に変換された画像をどのように貼り合わせるかが問題になってくる。カメラの自動絞りなどの機能により、写っている物体が違うと、同じ場所でも輝度が違ってくる。そのため単純に重ね合わせただけでは画像間の輝度の不一致が目立ってしまい、自然な合成を行うことができない。人間の視覚は局所的な変化には敏感であるが、そのため局所的な変化さえ小さければ少しぐらいの大域的な変化には気が付きにくい。従って局所的な内挿による合成を施せば輝度の変化が小さくなり、人間の目には自然な合成画像が得られるものと考えられる。今回ボロノイ線図を利用した領域分割による手法を用いて、合成を試みた。

2.2 変換パラメタの推定

変換式および変換パラメタの評価の基準としては次の残差二乗和 (Sum of Squared Differences,SSD) を用いることにする。

$$
\begin{aligned}
E &= \sum_i [I_1(x_i', y_i') - I_0(x_i, y_i)]^2 \qquad (1)\\
\boldsymbol{x}' &= M\boldsymbol{x}'
\end{aligned}
$$

ここで $\boldsymbol{x} = (x, y)$ および $\boldsymbol{x}' = (x', y')$ は変換 M の前後の座標を表すものとする。フレームの間に変換式を仮定して、この評価関数Eを最小にするようなパラメタを推定値として採用することにする。以下特定の変換式についてこの最小化の手法を示す。

2.2.1 平行移動パラメタの推定

最初に、もっとも単純な変換式として平行移動を考えてみる。この場合SSDはつぎのようになる。

$$E = \sum_i [I_1(x_i + u, y_i + v) - I_0(x_i, y_i)]^2 \tag{2}$$

この式を最小化する (u, v) を求める方法としては、Eを (u, v) の関数と考え (u, v) を適当な範囲内で動かしてEが最小になる点を探すという方法が考えられる。この方法を画像に対して適用するとSSDは次のようになる。

$$E = \sum_i (b[i + u][j + v] - a[i][j])^2. \tag{3}$$

ここでa[i][j],b[i][j]は点 (i, j) における画素の値である。従って次のようなアルゴリズムが考えられる。(図1参照)

1. 画像上に代表点の列 (i_k, j_k) をとる。
2. $(b[i_k + u][j_k + v] - a[i_k][j_k])^2$ を計算する。
3. (i_k, j_k) のまわりに適当な小領域（たとえば5 x 5画素）をとり、その中で $(b[i + u][j + v] - a[i][j])^2$ を計算し和をとる。
4. 2,3を (u, v) を適当な探索領域内で動かして繰り返し、和が最小になる (u, v) を探す。最小になる (u, v) を (i_k, j_k) における平行移動の推定値とする。
5. 代表点の列 (i_k, j_k) のすべてについて推定値 $(u(i_k, j_k), v(i_k, j_k))$ を求め u, v の度数分布を調べる。
6. u, v の最頻値を画像全体の推定値として採用する。

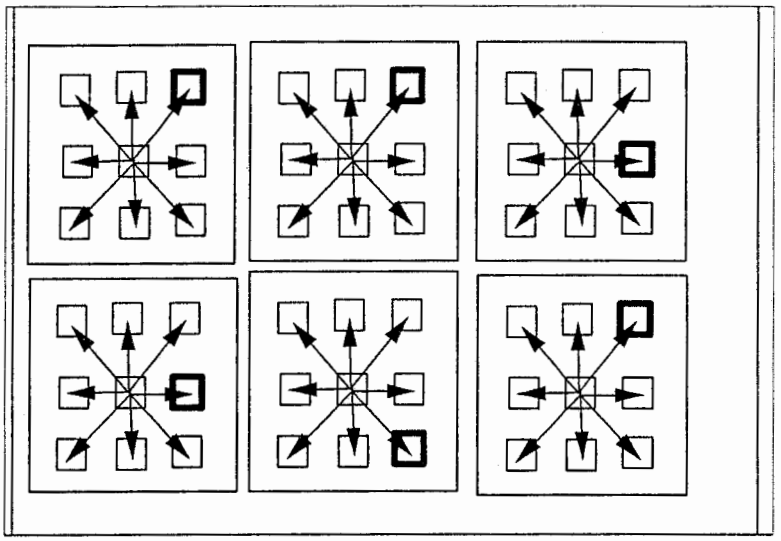


図 1: 平行移動パラメタの推定

この方法は直接的でわかりやすいが、以下のような問題点がある。

1. (u, v) も離散化せざるを得ないため正確な値は求められない。
2. 動きの大きい画像について (u, v) を求めようとすると、探索領域を大きくとる必要があり、計算量が大きくなる。
3. 変化の少ない領域や対応点のない領域の推定値は無意味である。
4. 画像は一般に平行移動だけでは一致しない。

このためこの手法は、大まかな推定値を求めることはできるが、正確な推定値を求めるには、特に動きの大きな画像については、適していないと考えられる。

2.2.2 Levenberg-Marquardt 法

多くのパラメタを含む一般の変換については、2.2.1 節のような直接的な方法は現実的ではない。そのため SSD を最小化するために何らかの最小化の手法を用いなければならない。今回用いた Levenberg-Marquardt 法 (以下 LM 法と呼ぶ) はそのような多変数関数の最小化の手法の一つである。以下にその概略を示す。

一般の多変数関数 $y = f(\boldsymbol{x})$ が $\boldsymbol{x} = \boldsymbol{x}_{min}$ （未知）で最小値をとるものとする。このとき $f(\boldsymbol{x})$ を $\boldsymbol{x} = \boldsymbol{x}_{min}$ のまわりでテーラー展開し 2 次の項までをとると、次のようになる。

$$
\begin{aligned}
f(\boldsymbol{x}) &= f(\boldsymbol{x}_{min}) + \sum_i \frac{\partial f}{\partial x_i} x_i + \frac{1}{2} \sum_{i,j} \frac{\partial^2 f}{\partial x_i \partial x_j} x_i x_j + \cdots \\
&\simeq c - \boldsymbol{b} \cdot \boldsymbol{x} + \frac{1}{2} \boldsymbol{x}^T \cdot A \cdot \boldsymbol{x} \qquad (4) \\
\boldsymbol{b} &= -\nabla f|_{x_{min}} \qquad (5) \\
A_{ij} &= \left. \frac{\partial^2 f}{\partial x_i \partial x_j} \right|_{\boldsymbol{x}_{min}} \qquad (6)
\end{aligned}
$$

式 (4) の勾配をとると

$$\nabla f = -\boldsymbol{b} + A\boldsymbol{x}, \qquad (7)$$

が得られる。最小値をとる $\boldsymbol{x}$ では $\nabla f = 0$ だから $A\boldsymbol{x}_{min} = \boldsymbol{b}$ が成り立つ。よって式 (7) との差をとると

$$
\begin{aligned}
A(\boldsymbol{x}_{min} - \boldsymbol{x}_i) &= -\nabla f(\boldsymbol{x}_i) \qquad (8) \\
\boldsymbol{x}_{min} - \boldsymbol{x}_i &= A^{-1}[-\nabla f(\boldsymbol{x}_i)] \qquad (9)
\end{aligned}
$$

となるので、ヘッセ行列 A と f の勾配 $[-\nabla f(\boldsymbol{x}_i)]$ より $\boldsymbol{x}_{min}$ を求めることができる。

以上の議論は式 (4) が f のよい近似となっていること、すなわち $\boldsymbol{x}_i$ が $\boldsymbol{x}_{min}$ の近くにあることが前提となっているので、$\boldsymbol{x}_i$ と $\boldsymbol{x}_{min}$ が離れている場合には適用できない。LM 法は $\boldsymbol{x}_i$ と $\boldsymbol{x}_{min}$ が離れている場合には $\boldsymbol{x}$ を f の勾配方向に一定量だけ動かして、$\boldsymbol{x}$ を $\boldsymbol{x}_{min}$ に近づけるという類似の最小化の手法である最急降下法を併用して、反復によってこの問題を解決しようとした手法である。具体的には λ という安定化因子を導入して、ヘッセ行列 A を次のような行列 A' で置き換える。

$$
\begin{aligned}
A'_{i,i} &= (1 + \lambda) A_{i,i} \qquad (10) \\
A'_{i,j} &= A_{i,j} (i \neq j) \qquad (11)
\end{aligned}
$$

$\lambda = 0$ のときは A' は A と一致する。一方 λ が大きいときには行列 A' は対角成分のみが大きくなるので、最急降下法に近い計算結果を与えることになる。従って次のようなアルゴリズムが構成できる。

1. λ および $\boldsymbol{x}_i$ を適当な値に定める。
2. 行列 A' および勾配 $\boldsymbol{b} = -\nabla f(\boldsymbol{x})$ の現在位置 $\boldsymbol{x}_i$ における値を求める。
3. 連立一次方程式 $A'\delta\boldsymbol{x} = \boldsymbol{b}$ を解く。
4. $f(\boldsymbol{x}_i + \delta\boldsymbol{x}) \geq f(\boldsymbol{x}_i)$ ならば λ を大きく（たとえば 10 倍）して 2. に戻る。
5. $f(\boldsymbol{x}_i + \delta\boldsymbol{x}) < f(\boldsymbol{x}_i)$ ならば $\boldsymbol{x}_i$ を $\boldsymbol{x}_i + \delta\boldsymbol{x}$ に更新する。
6. $f(\boldsymbol{x}_i)$ が十分小さくなったら終了。そうでなければ λ を小さく（たとえば 0.1 倍）して 2. に戻る。

この方法は、f の未知パラメタに関する微分が計算できないと適用できないということと、最小値ではなく極小値におちる可能性、それと関連して初期値をどこに設定するかという問題があるものの、柔軟ですぐれた方法であるといえる。変換式を仮定すれば、SSD(2) 式に対して、この手法を適用することができる。

2.2.3 変換式

平行移動以外の変換式として考えられるものにアフィン変換

$$x' = m_0 x + m_1 y + m_2 \tag{12}$$

$$y' = m_3 x + m_4 y + m_5 \tag{13}$$

が考えられる。画像を得るための光学系において測定系の2軸が正確に直交し、軸に沿った距離が正確に計測されていることを保証するのは難しい。式(13)のアフィン変換では、拡大縮小、平行移動、回転、傾斜歪み、縦横比変化を記述できるので[3]、より正確な位置あわせが実現できることになる。

合成に使う画像は多くの場合カメラで撮影したものである。従ってさらに正確に動きを記述するためには、次のような非線型の投影変換を用いる必要がある。

$$x' = \frac{m_0 x + m_1 y + m_2}{m_6 x + m_7 y + 1} \tag{14}$$

$$y' = \frac{m_3 x + m_4 y + m_5}{m_6 x + m_7 y + 1} \tag{15}$$

これらの変換式は6ないし8つの未知パラメータを含んでいる。しかし変換式は未知パラメータで微分可能なので、2.2.2節のLM法を適用することができる。さらにこれらの変換式は単純な平行移動も含んでいるので初期値に2.2.1節で求めた平行移動の推定値を使うことができる。またSSDの計算のときに選ぶ代表点も画像が重なり合う領域だけから選ぶことで、対応点が存在しない領域を計算することによる雑音を取り除くことができる。(図2参照)

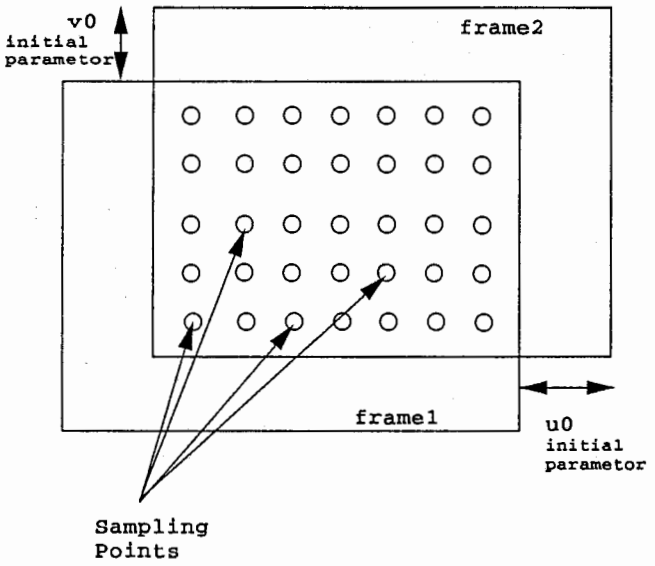


図 2: 初期値と代表点の設定

2.3 画像の貼り合わせ

変換式と変換パラメタをもとめて画像を変換すれば、本来2枚の画像はきれいに重なるはずであるが、実際には変換パラメタの推定誤差や画像間の輝度の不一致により、単純な重ねあわせではつなぎ目のところが目立ってしまうことが多い。そこで貼り合わせの際にもなんらかの合成の手法を用いて画像と画像が滑らかにつながるようにする必要がある。画像の合成を考えるときには、合成画像のある画素について、どの画像の画素を使ってどのような比率で合成するかを決めなければならない。その方法もいくつか考えられるが、今回はボロノイ線図を利用した領域分割と線形な内挿式を用いて合成を試みた。

2.3.1 ボロノイ線図

ボロノイ線図は次のように定義される平面の分割図形である。

平面上に指定された有限個の点の集合 $P = p_1, p_2, \cdots, p_n$ に対して領域 $R(p_i)$ を

$$R(p_i) = \{p | d(p, p_i) < d(p, p_j)\}, j \neq i \tag{16}$$

で定義したとき、$R(p_1), R(p_2), \cdots, R(p_n)$ で与えられる平面の分割図形をボロノイ線図という。ただし $d(p, q)$ は 2 点 p, q の Euclid 距離である。

すなわち 2 点間の垂直二等分線で区切られた図形と考えることもできる。(図 3 参照)

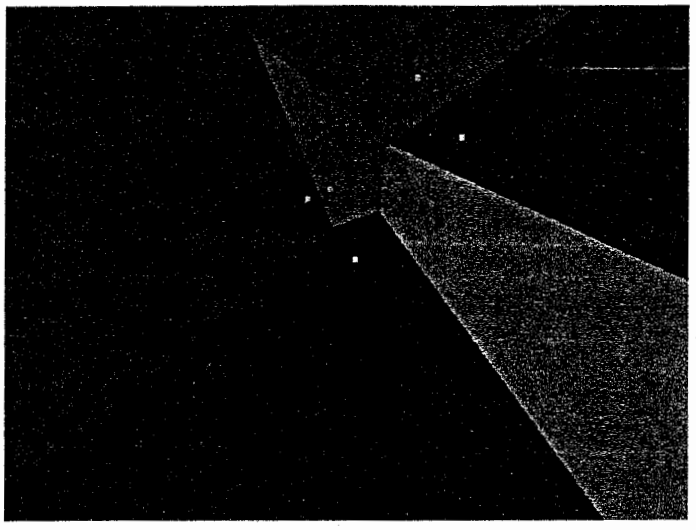


図 3: ボロノイ線図

2.3.2 領域分割

ボロノイ線図を利用して合成に使う画像の領域を決めるわけだが、ここでは点の集合 P として各フレームの重心を用いることにする。このとき各重心のまわりにとなりの重心との垂直二等分線がひかれることになる。そこで隣り合う 2 つの重心について、その垂直二等分線に平行で重心を通る 2 本の直線を考え、その 2 本の直線で挟まれた領域を遷移領域として、その 2 つの画像の画素を用いて合成画像の画素を決めることにする。2 枚のフレームの場合は図 4 のようになる。3 枚のフレームのときは図 5 のようになる。

一般の場合についても注目点から近い重心を順に 3 点選びだし、その注目点については選びだされた重心に対応する画像を使って合成を行うことにすれば 3 枚の合成の場合に帰着して考えることができる。実際の領域判定のアルゴリズムは以下のようなものを用いた。

1. 注目点と各画像の重心との距離を計算し、距離の小さいものを順に三つ選びだす。
2. 注目点が一番近い画像と二番目に近い画像の合成領域 (垂直二等分線に平行な二本の直線に挟まれた領域) に含まれるかどうかを判定する。もしも合成領域でない場合には、その点は一番近い画像の画素のみをそのまま用いる。
3. 合成領域に入る場合には、注目点が選びだされた 3 点で作る三角形の中にあるかどうかを判定する。もしも含まれる場合には、選びだされた 3 画像で合成を行う (三重合成領域)。含まれない場合には一番近い画像と二番目に近い画像で合成を行う (二重合成領域)。

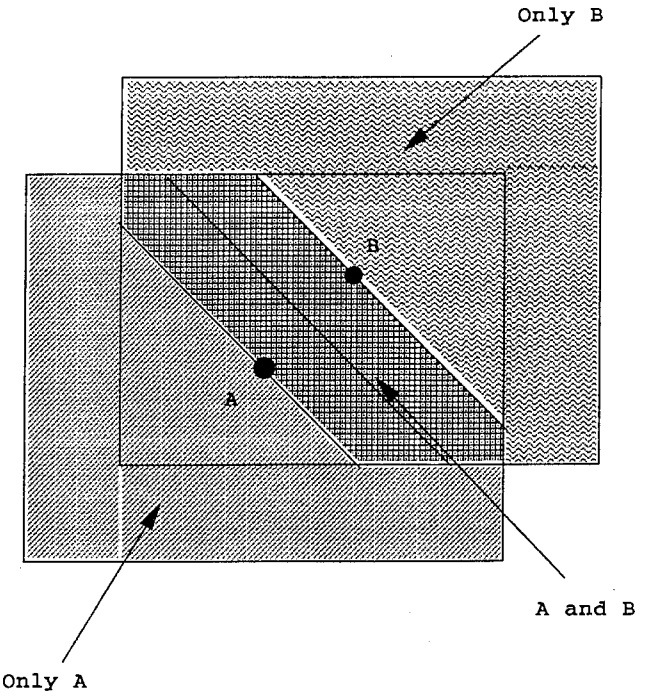


図 4: 領域分割 –2 枚の場合 –

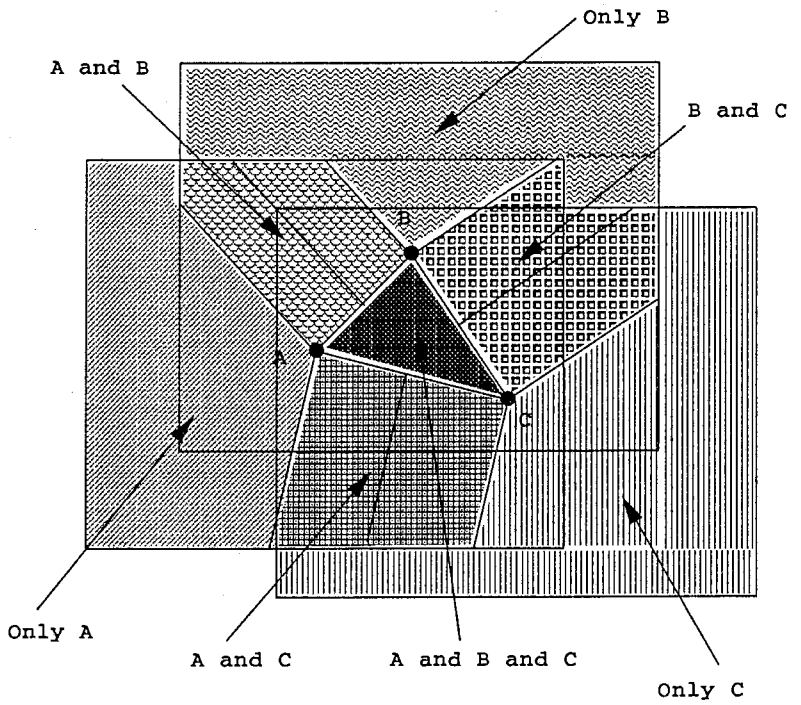


図 5: 領域分割 –3 枚の場合 –

二重合成領域に入るかどうかは、判定の対象である 2 枚の画像の重心の垂直二等分線と注目点との距離と、重心間の距離との大小によって判定することができる。三角形の中に入るかどうかは、次のように判定した。(図 6 参照)

1. 三角形の辺を構成する 3 本の直線の方程式を求める。
   (直線の式を $f_{ab}(x,y)=0, f_{bc}(x,y)=0, f_{ca}(x,y)=0,$ とする。)
2. 三角形の重心を計算する。
   ($x_g=\frac{x_a+x_b+x_c}{3}, y_g=\frac{y_a+y_b+y_c}{3}$)
3. 注目点の座標を (x_p,y_p) とするとき
   $f_{ab}(x_p,y_p)f_{bc}(x_p,y_p)f_{ca}(x_p,y_p)$ と $f_{ab}(x_g,y_g)f_{bc}(x_g,y_g)f_{ca}(x_g,y_g)$ が同符号であるかどうかを調べる。同符号でないなら三角形の内部にはない。
4. 同符号のときは、三頂点の x、y 座標それぞれの最大値と最小値を求める。もし注目点の座標が x、y 座標ともに最大値と最小値の間にあれば三角形の内部にある。

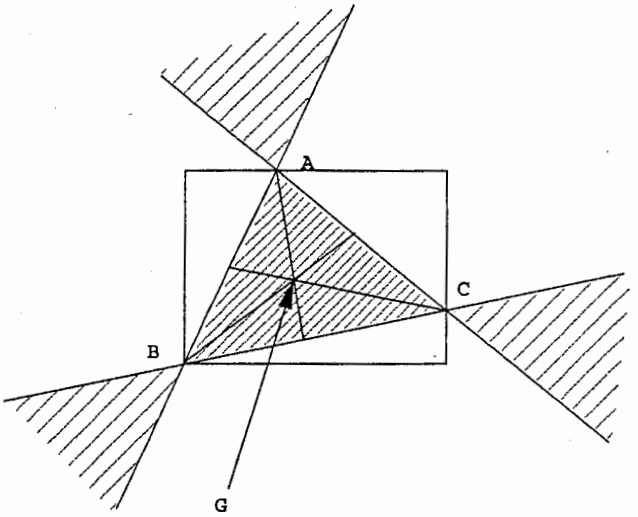


図 6: 三角形の内部にあるかどうかの判定

2.3.3 内挿式

内挿式としてもっともシンプルなものとしては、2 つないし 3 つの画素の単純平均をとる方式が考えられる。しかし合成領域が画像と画像の遷移領域であることを考えると、境界線からの距離に応じた加重平均をとった方がより滑らかに画像がつながると考えられる。そこで今回は重みとしてもっともシンプルな、境界線との距離に関して線形な重みをつけることにした。
二重合成領域のとき、重心間の距離を D、注目点 (x,y) と画像 A、B の境界線との距離をそれぞれ a,b、注目点に対応する A、B の画素をそれぞれ $A(x,y), B(x,y)$ とするとき、

$$P(x,y)=\frac{b}{D}A(x,y)+\frac{a}{D}B(x,y)$$

で内挿するものとする。
三重合成領域のときは、線形な重みもいくつかのとり方があるが、二重合成領域との連続性、プログラム上での表現の簡潔性を考慮した結果、図 7 のような内挿式を用いることにした。

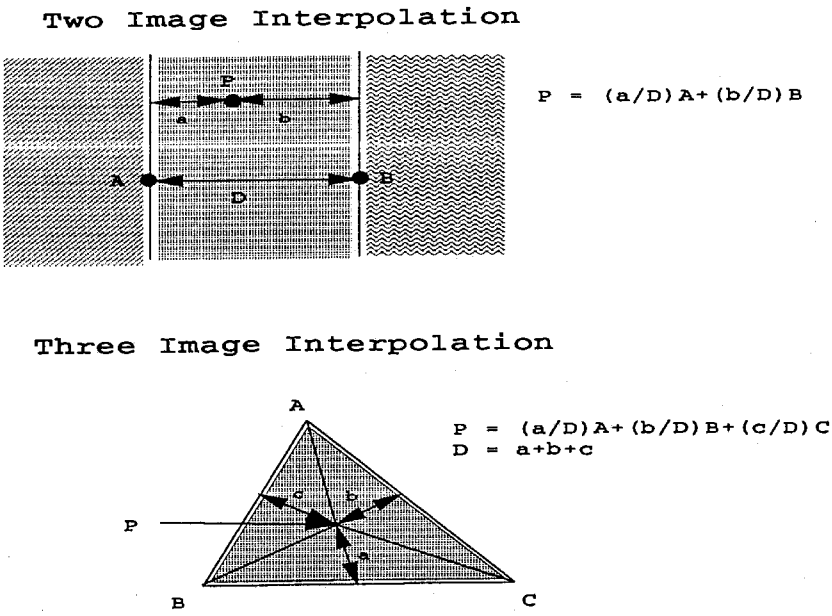


図 7: 内挿式

3 実験結果

3.1 実験に用いた画像

実験には位置を変えながら撮った静止画像 8 枚を使った。元の画像は IRIS RGB 形式のカラー画像で、1280x960 のサイズであった。実験には計算時間の圧縮とプログラムの単純化のために、サイズを半分の 640x480、カラーをモノクロにした。また解像度の高い画像は画素の変化が大きく、計算の際に誤差が大きくなるので、画像の解像度も小さくしてある。実際に使った画像の例を図 8 , 図 9 に示す。画像相互の位置関係は図 10 のようになっている。

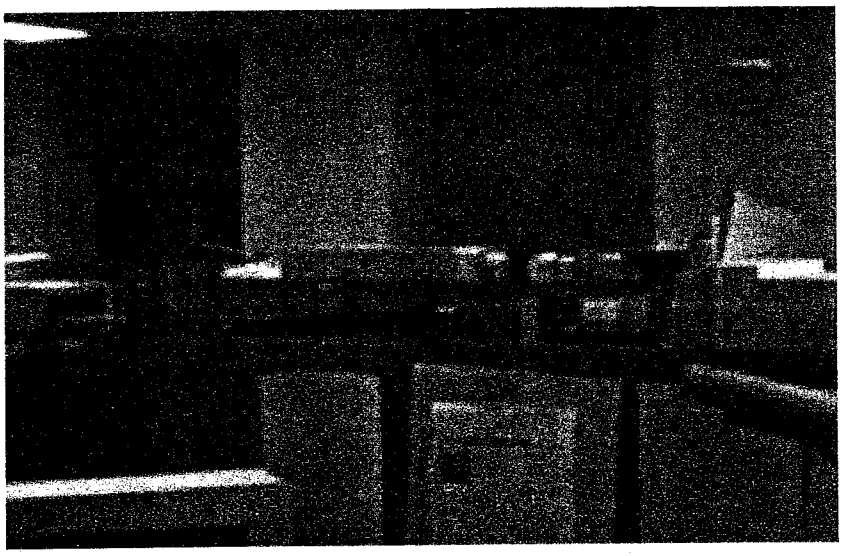


図 8: 画像例 1.room4

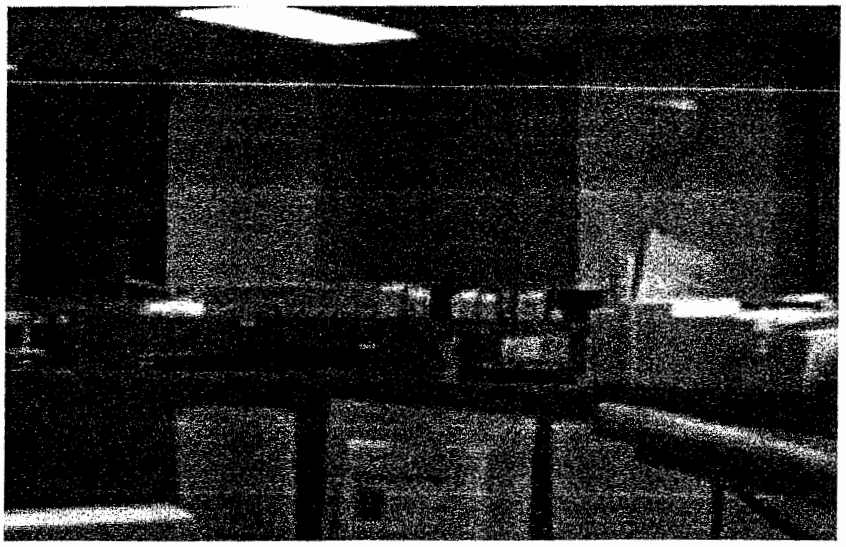
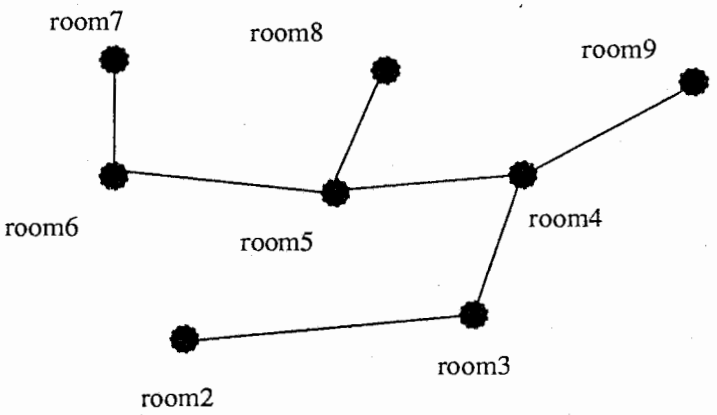


図 9: 画像例 2.room9



● は各画像の重心を表す。

——— は変換パラメタを求めた画像と画像をつないでいる。

図 10: 画像間の位置関係

3.2 平行移動

比較的近いと思われる 2 枚の画像について推定値を求めたところ次のような結果が得られた。

基本画像	対象画像	x 方向の変位推定値	y 方向の変位推定値
room5	room8	-18	-48
room2	room3	-96 以下	-10
room3	room4	-22	-58
room5	room6	82	-6
room4	room5	74	6
room6	room7	-2	-48
room4	room9	-62	-38

探索領域は x,y ともに -96 から 96 までで推定値は 4 づつとびとびに調べている。
変換して貼り合わせた例を図 11 に示す。画像の端の部分でずれが生じている事が分かる。

3.3 アフィン変換

平行移動のときと同じ組の画像について推定値を求めると以下のようになった。

基本画像	対象画像	m_0	m_1	m_2	m_3	m_4	m_5	SSD	反復回数
room5	room8	0.999	0.000	-15.68	0.001	1.000	-45.45	23	12
room2	room3	1.000	0.001	-105.0	0.001	1.003	-8.718	152	15
room3	room4	1.000	0.000	-20.43	-0.010	1.003	-59.88	309	14
room5	room6	1.005	-0.006	86.13	0.002	0.998	-4.442	22	11
room4	room5	0.998	0.004	75.44	0.002	1.004	8.884	63	22
room6	room7	0.999	0.000	-1.434	-0.000	1.000	-45.18	24	9
room4	room9	0.995	0.016	-64.87	0.002	0.9950	-35.11	21	20

平行移動のときの例と同じ画像を合成した場合、図 12のような画像が得られた。画像の端の部分 (特に右下) のずれが改善されていることがわかる。

6 枚の画像を貼り合わせたところ図 13 のような画像が得られた。端の部分にはまだずれがみられる部分もあるが、全体的には滑らかに合成されている。

3.4 非線型投影変換

平行移動のときと同じ組の画像について推定値を求めると以下のようになった。

基本画像	対象画像	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7	SSD	反復回数
room5	room8	1.000	-0.001	-17.81	0.002	1.000	-47.26	0.000	0.000	30	10
room2	room3	1.003	-0.001	-105.5	0.002	1.004	-8.386	0.000	-0.000	156	28
room3	room4	1.000	0.001	-19.59	-0.022	1.005	-61.01	0.000	0.000	197	24
room5	room6	1.001	-0.005	84.67	0.001	0.995	-4.142	-0.000	-0.000	12	20
room4	room5	1.003	-0.001	75.65	0.001	0.996	9.328	-0.000	-0.000	10	28
room6	room7	0.999	0.000	-2.139	-0.004	0.999	-46.92	0.000	0.000	24	16
room4	room9	0.997	0.009	-64.82	0.002	0.9952	-35.15	0.000	0.000	24	4

非線型成分である m_6, m_7 がほぼ 0 になっており、事実上アフィン変換と変りがなくなってしまっている。この場合カメラの動きが小さいのでアフィン変換で十分動きを捉えられていると考えられる。3.3 節と同じ画像を使った例を示すと図 15,14 のようになる。

なお 6 フレームの合成画像にみられる引っ掻き傷のような跡は、変換の際の丸めによって画素が切れてしまっているために生じたものである。これについてはプログラムの修正による補間を施せば解消することができる。

図 11: 平行移動による合成の例

図 12: アフィン変換による合成の例（2 フレームの場合）

図 13: アフィン変換による合成の例 (6 フレームの場合)

図 14: 非線型投影変換による合成の例 (6 フレームの場合)

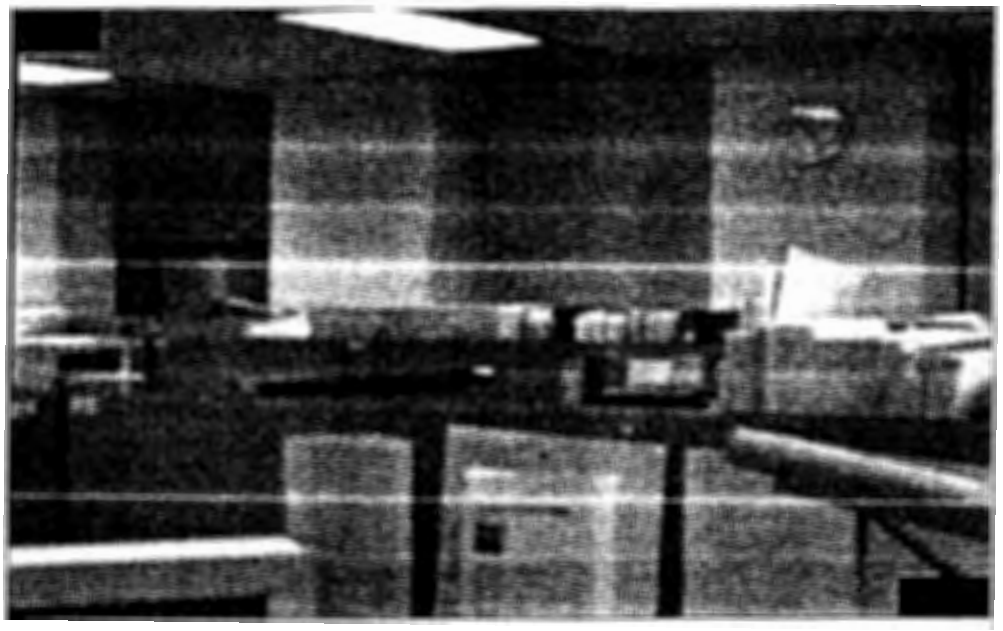


図 15: 非線型投影変換による合成の例 (2 フレームの場合)

4 おわりに

4.1 まとめ

今回できたことは、次のようにまとめられる。

- 平行移動パラメタの直接的推定。
  精度が悪い、計算に時間がかかる等の問題はあるものの、おおざっぱな位置あわせは可能である。反復法の初期値を決めるという意味では十分な方法であると考えられる。
- LM 法による変換パラメタの推定。
  未知パラメタで微分できる変換式を仮定し、さらに未知パラメタの概数値が分かっている場合には、LM 法を適用することで、さらによい推定値を求めることができる。
- アフィン変換、非線型投影変換による画像の変換。あまり大きく動かない今回のような画像の場合には、アフィン変換でも十分な位置あわせができることが確認された。
- ボロノイ線図を利用した画像の貼り合わせ。
  変換パラメタが精度良く求まっている場合には、合成を施す中心部分を滑らかにつなぐことができた。ただし合成を施さない画像の端の部分のずれまでは修正することはできない。

4.2 今後の課題

今回の実習では期間が短かったため、考案したアルゴリズム、試作したプログラムの十分な検証がなされていない。これまで述べてきた手法を様々な静止画像系列について適用し、アルゴリズム、手法の妥当性を検証していく必要がある。

プログラム（コーディング）レベルの課題としては、次のようなものが挙げられる。

- 変換計算の際の丸めによる、画像の切れの消去。
  切目の部分に内挿を施す必要がある。
- 領域分割アルゴリズムの見直し。
  合成するフレームが多くなると、時々明らかにおかしい合成が施される部分がある。

アルゴリズム構成レベルの課題としては

- 貼り合わせ方式の検討。
  現在のプログラムの方式では、変換画像と元の画像の画素の対応を、基準になる画像(最初にはり付ける画像)からの合成変換を計算することでとっているので、基準画像から離れた画像の場合、誤差の蓄積によりずれが大きくなってしまう。このような画像の並べ方に依存しないはりあわせの方式を検討する必要がある。
- 画像の環状接続。
  仮想空間の構成のためには全方向に画像が映し出されなければならない。現在のプログラムの方式では、鎖状に画像を接続するため広角画像は作れても、観察者を取り巻くような全方向に映し出される真のパノラマ画像にすることはできない。真のパノラマ画像を作成するためには画像を環状に接続するような方法も考える必要がある。

また今回は対象を静止画だけに限定しているが、より一般的な仮想環境を構成する場合には動画も扱う必要がある。動画の場合これまでのべてきたアフィン変換や投影変換だけでは背景部分の位置あわせしかできないため、画像から動物体を抽出する必要がでてくる。すなわちグローバルなオプティカルフローとローカルなオプティカルフローの分離という問題が出てくる。このような場合今までのように単純に格子上に代表点をとるわけにはいかなくなるなど、新たな問題が出てくる。

謝辞

本研究は西羅光[1] が学外実習生として ATR 滞在中（1996 年 2 月 26 日～3 月 15 日）に実施したものである。

本研究の機会を与えて下さった (株)ATR 知能映像通信研究所 葉原耕平会長、中津良平社長、ならびに第二研究室をはじめとする各研究室の皆様に感謝致します。

参考文献

[1] R.Szeliski,J.Coughlan : Hierarchial Spline-Based Image Registration, in Proceedings of IEEE Workshop on Representation of Visual Scenes, Cambridge, MA, June 1995.

[2] W.H.Press, B.P.Flannery, S.A.Teukolsky, W.T.Vetterling : Numenical Recipes in C :Cambridge University Press.

[3] Horn：ロボットビジョン、NTT ヒューマンインタフェース研究所訳、朝倉書店、1993.

[4] K.Mase : Computing Field-of-View of Stitched Panorama to Create FoV Sensitive Virtual Enviornments : submitted to ICPR'96.

[5] 杉原厚吉：位相優先法 - 幾何学的アルゴリズムの数値的安定化のための一手法：情報処理学会グラフィクスと CAD シンポジウム講演集、1992.

[1]現所属 東京大学工学部計数工学科

付録

今回の実習で作成したプログラムのうち主要なものについて、解説する。自作した関数についても後でまとめて解説する。

A　プログラム

プログラム一覧

1. calcvec.c (Calculation Vector)
   直接走査により、SSD が最小になる平行移動成分を各代表点ごとに求め、指定されたファイルに出力するプログラム。

2. dattohist.c (Data to Histogram)
   プログラム calcvec から出力されたデータを読み込んで、x 成分、y 成分毎に度数分布表を作り標準出力に表示するプログラム。

3. sqimgsyn.c (Sequential Image Synthesize)
   平行移動成分を入力して画像を連続的にはりあわせていき、指定されたファイルに出力するプログラム。なお三重合成のアルゴリズムを作る前のプログラムなので二重合成のみで画像をはりあわせている。

4. sqimgsyna.c (Sequential Image Synthesize Affine Transform) 入力された平行移動成分の推定値を初期値として LM 法によりアフィン変換パラメタを計算し、それを基に画像をアフィン変換して、はりつけていくプログラム。三重合成も用いて画像をはりつけている。

5. sqimgsynp.c (Sequential Image Synthesize Projective Transform) sqimgsyna.c の非線型投影変換版。変換式と未知パラメタの数が違う他はまったく同じプログラム。

A.1　calcvec.c

54 行目からが実際の計算のループになる。54、55 行目は代表点のループである。代表点は x 方向に DX ずつ、y 方向に DY ずつの間隔で並べている。
56 行目は最小値を求めるために、実際にはとり得ない大きい値に初期化している。
58 行目からのループが変位ベクトルを動かして最小値を求めるループである。
62 行目からのループは小領域の和をとるループになる。64 行目から 68 行目までは走査範囲が画像の外に出てしまった場合の処理である。
配列 c の index を変換しているのは、配列に空きが生じないようにするためである。
74 行目で不等式に等号も含めたため、たとえ値が同じでも後に出現したものが優先されるようになっている。そのため無意味な領域では値が上に片寄ることになる。ファイルへの出力は 85 行目のフォーマットに従って出力している。

A.2　dattohist.c

37 行目からのループで入力ファイルからフォーマットに従ってデータを読み必要な部分を配列 xmov,ymov に格納している。
43 行目からが集計のループになる。44、46 行目で読んだデータを 0 から階級数の値に正規化して配列 hisx,hisy の各要素に積み上げている。
50 行目以降は出力処理である。最頻値も求めているが実際には無意味な領域が一番上にピークを作ってしまうので意味がない。

A.3 sqimgsyn.c

計算に必要な画像データを格納するために構造体 imgpos を定義している。ループのために左角の座標を格納するようにしている。
82 行目からの maxx,etc は最後に不要部分を切り取るために追跡するようにしたものである。
87 行目からのループではまず b（移動する画像）を c にはりつけ、その後に a を上書きしている。
99 行目から垂直二等分線のパラメタを計算し、vy の正負で場合分けして以後領域判定と内挿を行っている。本文では触れなかったが重み d は 112 行目または 129 行目のように計算することもできる。
139 行目以降が一般のはりつけループである。二重合成のみを扱っているのでやることは基本的に同じである。必要なデータを入力後、99 行目から 135 行目までの処理を関数化した関数 paste(169 行目) に値を渡して画像をはりつけている。
175 行目から不要部分を削ぎ落として、削ぎ落とした配列を関数 arraytoimg に渡して出力画像を RGB 形式に戻している。

A.4 sqimgsyna.c

sqimgsyn.c で定義した構造体 imgpos を拡張し、画像に関するすべてのデータを収納する構造体 imgdata を定義している。このうち gx,gy は前の 4 つのメンバーから計算することができるが、頻繁に使うものなのでメンバーに組み込むことにした。なお合成変換を求めるために、画像相互の変換関係を記録しておく必要があるので、自分自身の型を持つポインタをメンバーに含む自己参照構造体の構成をとっている。これだけ大きい構造体を関数に渡す場合には、オーバヘッドによる時間のロスをさけるために関数にはポインタを渡すようにしないと処理に膨大な時間がかかることになるため注意が必要である。
105 行目から 114 行目までは必要なメモリーを確保する処理である。
119 行目から 187 行目までが最初の 2 枚をはりつける処理である。148 行目の関数 mrqmina が LM 法で未知パラメタを求める関数である。157 行目の関数 getmaxmin は画像の変換先の範囲を求める関数である。158 行目の関数 intpl は変換した画像が丸めによって切れないように内挿してはりつける関数である。
183 行目の関数 doublesyn は二重合成領域の判定と内挿を同時に行う関数であり、合成領域にない場合には 0 をそうでない場合には内挿値を返すようになっている。
189 行目からの連続はりつけループではまずメニューを表示してどの画像との初期値を入力するかを聞いている。207 行目でパラメタを推定すると 209 行目からのループで 1 枚目の画像からの合成変換を求めている。合成変換パラメタで画像を変換し、それをいったん作業用の配列に入れている。234 行目からのループで白地にのみ書き込みを許している。
241 行目からが領域判別ループでまず最初に三角形の抽出を行っている。新しく加わった画像が三角形の頂点に選ばれないときは、合成領域ではないので残りの処理をスキップする。新しい重心が何番目に近いかで判定法が若干違ってくる。関数 doublesyn, 三角形の内部にあるかどうかを判定する関数 intriangle, 三重合成の内挿値を返す関数 triplesyn を用いて領域分割と内挿を行っている。

B 関数

B.1 画像のデータ形式を変換する関数

1. imgtoarray (Image to Array)
   RGB 形式のイメージファイルを読み込んで指定された配列に格納する関数。関数 getrow で 1 行ずつデータを読み込んで配列に格納する。別の引数に画像のサイズの値も返す。

2. arraytoimg (Array to Image)
   配列をサイズの値とともに受け取って RGB 形式のイメージファイルに戻す関数。関数 putrow で 1 行ずつ配列を読み込んでファイルに書き出している。

B.2 変換式に関する関数

1. aftrsf,aftrsf2 (Affine Transform)
   座標と変換パラメタを受け取って変換した座標を返す関数。aftrsf は int 型の値を返し aftrsf2 は float 型の値を返しているところが異なる。ところだけが異なる。

2. invaftrs (Inverse Affine transform)
   アフィン逆変換した座標を int 型で返す関数。

3. aftrsfsyn (Affine Transform Synthesize)
   第二引数 → 第一引数の順に合成変換したときの合成変換パラメタを第三引数に返す関数。

4. prtrsf,prtrsf2,invprtrsf,prtrsfsyn (Projective Transform,etc)
   非線型投影変換についての演算。アフィン変換に関するものに準ずる。

B.3 LM 法の実行に必要な関数

1. diff (Differential)
   配列と座標の値を受け取って第一引数で指定された方向の差分を返す関数。計算に必要な画素がない場合には NULL を返す。

2. sol (Solution)
   行列 a とベクトル b を受け取ってガウスの消去法で連立一次方程式の解を求め配列 x に格納する関数。

3. inv (Inverse)
   行列 a を受け取って逆行列 b を返す関数。行列 a と各座標軸方向の単位ベクトルに対して関数 sol を適用し、得られた解を列毎に並べて逆行列を求めている。逆行列は LM 法の実行に必ずしも必要なものではないが、誤差共分散を求める際に用いている。

4. err (Error)
   画像配列、代表点に関する情報、変換パラメタを受けとって SSD（用いた画素数で割った値）を計算して返す関数。代表点は図 2 に示したようにとっている。すなわち x0,y0 が重なり領域の左下角の座標、xe,ye が右上角の座標である。変数 nx,ny は代表点の間隔である。変換した座標が画像の範囲外に出てしまう時にはその画素は使わないようにしている。

5. grad (Gradient)
   $\frac{\partial I(x',y')}{\partial m_i}$ を計算して返す関数。実際には

$$\frac{\partial E}{\partial m_i} = \frac{\partial E}{\partial x'}\frac{\partial x'}{\partial m_i} + \frac{\partial E}{\partial y'}\frac{\partial y'}{\partial m_i}$$

   を計算することになる。配列 t が $\frac{\partial x'}{\partial m_i}, \frac{\partial y'}{\partial m_i}$ を表している。

6. setbeta setalpha (Set Beta,Set Alpha)
   LM 法で必要なヘッセ行列、勾配ベクトルを以下のように計算する関数。

$$\begin{aligned}
\beta_k &\equiv -\frac{1}{2}\frac{\partial E}{\partial m_k} \\
&= -\sum_i [I_1(x',y') - I_0(x,y)]\frac{\partial I(x',y')}{\partial m_i} \\
\alpha_{kl} &\equiv \frac{1}{2}\frac{\partial^2 E}{\partial m_k \partial m_l} \\
&\simeq \sum_i \frac{\partial I_1}{\partial m_k}\frac{\partial I_1}{\partial m_l}
\end{aligned}$$

ここでalphaの計算は二階微分を一階微分の積で近似している [2]。実際にはこの後対角成分に $(1+\lambda)$ をかけたものがalphaとなる。

B.4 LM法を実行する関数

1. mrqmina (Marquardt Minimization Affine Transform)
   LM法でアフィン変換パラメータを計算、表示する関数。最初に平行移動成分の推定値の入力を要求し、入力された値をもとに初期値のセット、代表点の決定を行ない、B.3節で定義した関数を用いて反復計算を行なっている。反復停止条件としては

$$\frac{E(m+\delta m)-E(m)}{E(m)} \leq \varepsilon$$

   を用いている。推定値が求まったらその値を反復回数、最終的なSSDの値、誤差共分散行列とともに表示する。なお誤差共分散行列は反復を停止した時のalphaの $\lambda = 0$ としたときの逆行列である [2]。

2. mrqminp (Marquardt Minimization Projective Transform)
   mrqminaの非線形投影変換版。変換に関係する関数が変わっただけで基本的に同じ関数。

B.5 幾何学的諸量を計算する関数

1. dpp (Distance between Point and Point)
   点と点の距離を計算する関数。

2. dpl (Distance between Point and Line)
   点と直線の距離を計算し符号つきで値を返す関数。

3. el,vl (Edge/Vertical Line)
   2点を結ぶ直線（2点の垂直二等分線）のパラメタを返す関数。

4. cp(Cross Point)
   2直線の交点の座標を計算しint型で返す関数。

B.6 領域分割と画素合成に用いる関数

1. intriangle (In Triangle)
   三角形の内部にあるかどうかを判定する関数。判定のアルゴリズムは2.3.2節(10ページ 図6)参照。

2. doublesyn (Double Synthesize)
   二重合成領域にあるかどうかを判定し、あれば内挿値を、なければ0を返す関数。判定アルゴリズム、内挿式は2.3.2、2.3.3節、11ページ 図7参照。

3. triplesyn (Triple Synthesize)
   三重合成領域の内挿値を返す関数。(11ページ 図7参照。)

B.7 画像の貼り付けのための関数

1. getmaxmin (Get Maximum and Minimum)
   画像のエッジに沿って走査、変換し変換画像の範囲を返す関数。

2. intpl (Interpolation)
   変換の際に丸めよる切れを埋めるように画素の値をを内挿しながら貼り付ける関数。

   内挿の手順は以下の通り。(図16参照)

(a) 注目点 $(x0, y0)$ を逆変換し (int 型にキャストして) 対応点 $(i0, j0)$ を求める。

(b) $(i0, j0)$ およびそのまわりの 9 点をアフィン変換してどこに変換されるかを調べる。

(c) 変換された点のうち、$(x0, y0)$ との距離が 1 未満の点を選び出す。

(d) 選び出された点と $(x0, y0)$ との距離に応じた重みをつけて加重平均し得られた値を内挿値とする。

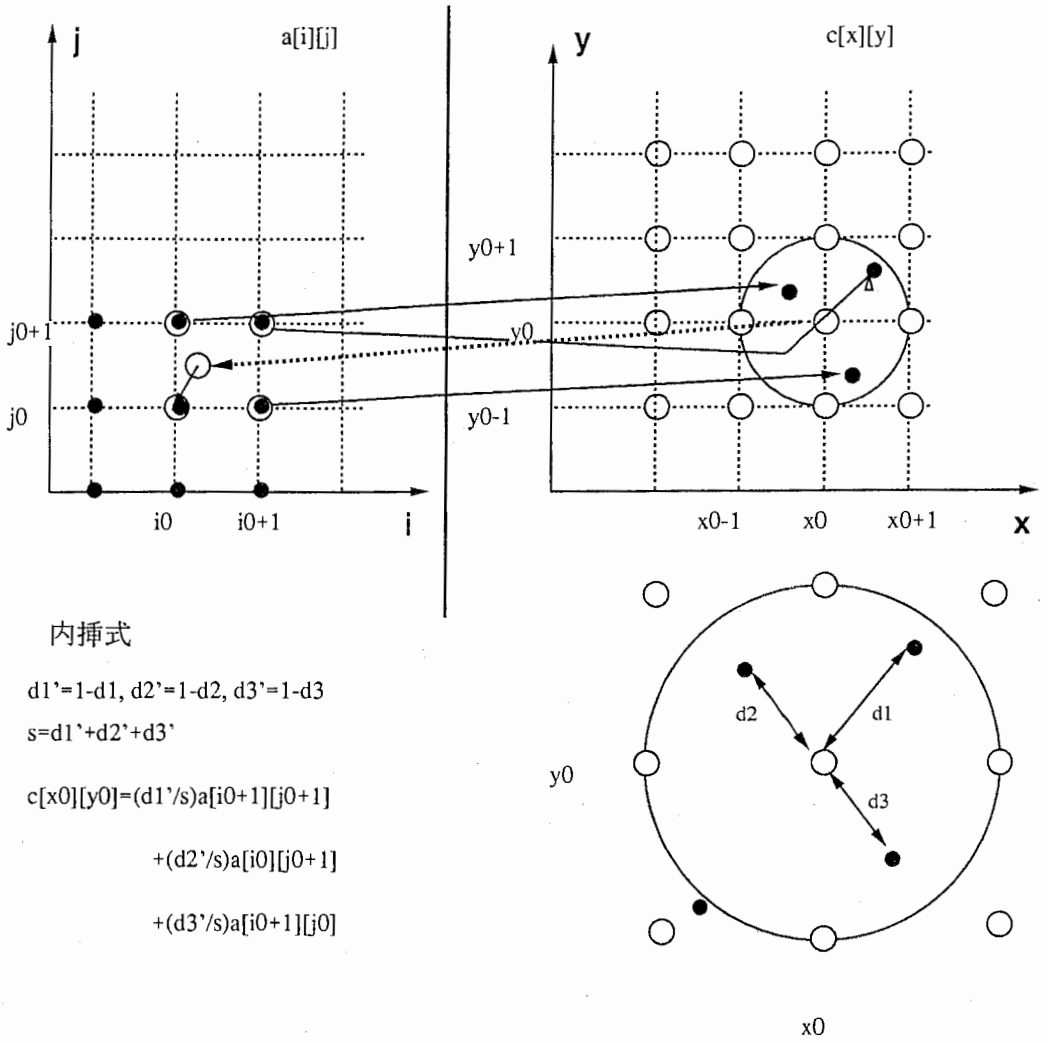


図 16: 画像の切れを防止するための内挿

B.8 その他の関数

1. max,maxf (Maximum/Maximum Float)
   渡された 2 つの値のうち大きい方を返す関数。max は int 型、maxf は float 型に対応。

2. min,minf (Minimum/Minimum Float)
   渡された 2 つの値のうち小さい方を返す関数。min は int 型、minf は float 型に対応。

3. absf (Absolute Float)
   渡された値 (float 型) の絶対値を返す関数。

```
  1 : /*
  2 :  *          calcvec.c -
  3 :  *
  4 :  *          2枚の画像をとりこんで、差分の二乗計算により
  5 :  *          格子点上の変位ベクトルを求めるプログラム（試作）
  6 :  *
  7 :  *          西羅 光  -1996
  8 :  */
  9 : #include <stdio.h>
 10 : #include "image.h"
 11 :
 12 : #define SQR(a) ((a)*(a))
 13 :
 14 : #define SDS 2    /*小領域のサイズ*/
 15 : #define WS  96   /*探索領域のサイズ*/
 16 : #define WDX 4    /*探索領域のx方向の格子間隔*/
 17 : #define WDY 4    /*探索領域のy方向の格子間隔*/
 18 : #define DX  10   /*x方向の格子間隔*/
 19 : #define DY  10   /*y方向の格子間隔*/
 20 :
 21 : void imgtoarray(char *iname , short a[][960],int *xs,int *ys);
 22 : int max(int a, int b);
 23 :
 24 : main(int argc,char **argv)
 25 : {
 26 :     FILE *fout;
 27 :   short a[1280][960],b[1280][960];  /*画像格納用配列*/
 28 :     int xsizea,ysizea,xsizeb,ysizeb; /*画像のサイズ*/
 29 :     int xsize,ysize;
 30 :     int i,j;
 31 :   unsigned int x0,y0;          /*格子点*/
 32 :   int w,h;
 33 :   int c[100][100];              /*画素の差分の二乗を格納*/
 34 :   int minc[3];                 /*cの最小値とそのときのwとhを格納*/
 35 :   int u[1280][960],v[1280][960];    /*変位ベクトル*/
 36 :
 37 :   if( argc < 4 ){
 38 :           fprintf(stderr,"usage : calcvec image1.rgb image2.rgb output\n");
 39 :           exit(1);
 40 :   }
 41 :
 42 :   imgtoarray(argv[1],a,&xsizea,&ysizea);  /*aに1枚目の画像を格納*/
 43 :   imgtoarray(argv[2],b,&xsizeb,&ysizeb);  /*bに2枚目の画像を格納*/
 44 :
 45 :   /*サイズの決定*/
 46 :   xsize = max(xsizea,xsizeb); ysize = max(ysizea,ysizeb);
 47 :
 48 :   if((fout = fopen(argv[3],"w")) == NULL){
 49 :           fprintf(stderr,"can't open %s\n",argv[3]);
 50 :           exit(1);
 51 :   }
 52 :
 53 :   /*全画面上を走査*/
 54 :   for(x0=SDS+DX; x0<=xsize; x0=x0+DX){
 55 :           for(y0=SDS+DY; y0<=ysize; y0=y0+DY){
 56 :                   minc[0] = 10000;
 57 :                   /*探索領域上を走査*/
 58 :                   for(w=-WS; w<=WS; w=w+WDX){
 59 :                           for(h=-WS; h<=WS; h=h+WDY){
 60 :                                   c[(w+WS)/WDX][(h+WS)/WDY] = 0;     /*初期化*/
 61 :                                   /*小領域上を走査*/
 62 :                                   for(i=-SDS; i<=SDS; i++){
 63 :                                           for(j=-SDS; j<=SDS; j++){
 64 :                                                   if((x0+w+i < 0) || (y0+h+j <
0)
 65 :                                                           ||(x0+i < 0) || (y0+j < 0)
```



```
 66 :                                                           ||(x0+w+i > xsize) || (y0+h+
j > ysize)
 67 :                                                           ||(x0+i > xsize) || (y0+j >
ysize))
 68 :                                                           continue;
 69 :                                                   c[(w+WS)/WDX][(h+WS)/WDY] =
/*和の計算*/
 70 :                                                           c[(w+WS)/WDX][(h+WS)/WDY] +
 71 :                                                           SQR( b[x0+w+i][y0+h+j] - a[x
0+i][y0+j] );
 72 :                                           }
 73 :                                   }
 74 :                                   if(minc[0] >= c[(w+WS)/WDX][(h+WS)/WDY]){  /*
最小値の追跡*/
 75 :                                     minc[0] = c[(w+WS)/WDX][(h+WS)/WDY]; /*値を
更新*/
 76 :                                     minc[1] = w;  /*x成分を更新*/
 77 :                                     minc[2] = h;  /*y成分を更新*/
 78 :                           }
 79 :                   }
 80 :           }
 81 :           /*変位ベクトルの決定*/
 82 :           u[x0][y0] = minc[1];
 83 :           v[x0][y0] = minc[2];
 84 :           /*ファイルへの出力*/
 85 :           fprintf(fout,"point %d,%d vector %d,%d\n",x0,y0,minc[1],minc[
2]);
 86 :   }
 87 :   }
 88 :   fclose(fout);
 89 : }
 90 :
 91 : void imgtoarray(char *iname , short a[][960] ,int *xs ,int *ys)
 92 : {
 93 :     IMAGE *iimage;
 94 :     unsigned int xsize,ysize;
 95 :     unsigned int x,y;
 96 :   short ibuf[960];
 97 :
 98 :     if( (iimage=iopen(iname,"r")) == NULL ) {
 99 :   fprintf(stderr,"imgtoarray: can't open input file %s\n","iname);
100 :   exit(1);
101 :     }
102 :     xsize = iimage->xsize;
103 :     ysize = iimage->ysize;
104 :
105 :   *xs = xsize; *ys = ysize;
106 :
107 :     for(y=0; y<ysize; y++) {
108 :       getrow(iimage,ibuf,y,0);
109 :           for(x=0; x<xsize; x++){
110 :                   a[x][y] = ibuf[x];
111 :           }
112 :   }
113 :     iclose(iimage);
114 :     return;
115 : }
116 :
117 : int max(int a, int b)
118 : {
119 :   if(a >= b){
120 :           return(a);
121 :   }
122 :   else{
123 :           return(b);
124 :   }
```

```
125 : }
126 :
127 :
128 :
129 :
130 :
131 :
132 :
133 :
134 :
135 :
136 :
137 :
138 :
139 :
140 :
141 :
142 :
143 :
144 :
145 :
146 :
147 :
```

```
 1 : /*
 2 :  *          dattohist.c -
 3 :  *
 4 :  *          配列データを読み込んでヒストグラムを作るプログラム
 5 :  *
 6 :  *          西羅 光    -1996
 7 :  */
 8 : #include <stdio.h>
 9 :
10 : #define WS  96 /*探索領域のサイズ*/
11 : #define RW  4  /*階級幅*/
12 : #define RN  WS / RW  /*階級数*/
13 :
14 : main(int argc,char **argv[])
15 : {
16 :   FILE *fin;
17 :   char dmyp[100],dmyv[100]; /*データ中の文字列を格納*/
18 :           short xmov[5000],ymov[5000];
19 :   char buf[40]; /*1行読んだデータを格納*/
20 :   int numdat = 0; /*データの数*/
21 :   int i;
22 :   int dmyx,dmyy,dmyvx,dmyvy;
23 :   int hisx[100] = {0},hisy[100] = {0};
24 :   int hdmyx,hdmyy;
25 :   int modex[2] = {0,0},modey[2] = {0,0};
26 :
27 :   if(argc < 2){
28 :           fprintf(stderr,"usage : dattohist data.dat \n");
29 :           exit(1);
30 :   }
31 :
32 :   if((fin = fopen(argv[1],"r")) == NULL){
33 :           fprintf(stderr,"can't open %s\n",argv[1]);
34 :           exit(1);
35 :   }
36 :
37 :   while(fgets(buf,40,fin) != NULL){
38 :           numdat++;
39 :           sscanf(buf,"%s%d,%d%s%d,%d",&dmyp,&dmyx,&dmyy,&dmyv,&dmyvx,&dmyvy);
40 :           xmov[numdat-1] = dmyvx; ymov[numdat-1] = dmyvy;
41 :   }
42 :
43 :   for(i=0;i<=numdat-2;i++){
44 :           hdmyx = xmov[i] / RW;
45 :           hisx[RN + hdmyx]++;
46 :           hdmyy = ymov[i] / RW;
47 :           hisy[RN + hdmyy]++;
48 :   }
49 :
50 :   for(i=0;i<=2*RN;i++){
51 :           printf("%3d < x <= %3d == %4d    ",(i-1-RN)*RW,(i-RN)*RW,hisx[i]);
52 :           if(hisx[i] >= modex[1]){
53 :                   modex[1] = hisx[i];
54 :                   modex[0] = i;
55 :           }
56 :           printf("%3d < y <= %3d == %4d\n",(i-1-RN)*RW,(i-RN)*RW,hisy[i]);
57 :           if(hisy[i] >= modey[1]){
58 :                   modey[1] = hisy[i];
59 :                   modey[0] = i;
60 :           }
61 :   }
62 :
63 :   printf("modex -> %3d < x <= %3d\n",(modex[0]-1-RN)*RW,(modex[0]-RN)*RW);
64 :   printf("modey -> %3d < y <= %3d\n",(modey[0]-1-RN)*RW,(modey[0]-RN)*RW);
65 : }
66 :
```



```
67 :
68 :
```

```
 1 : /*
 2 :  *          sqimgsyn.c -
 3 :  *
 4 :  *          複数の移動画像を重ね合わせて合成するプログラム
 5 :  *  (重なる領域に重心間の垂直二等分線を基準に線形な重みをつけて合成する場合)
 6 :  *
 7 :  *          西羅 光    -1996
 8 :  */
 9 : #include <stdio.h>
10 : #include <math.h>
11 : #include "image.h"
12 :
13 : #define MAXV 96
14 : #define XSIZE 1280
15 : #define YSIZE 960
16 :
17 : struct imgpos{        /*画像の位置、サイズに関する情報を格納*/
18 :   char name[20];    /*ファイル名*/
19 :   int xs;           /*画像のサイズ*/
20 :   int ys;
21 :   int swx;          /*画像の左下角の座標*/
22 :   int swy;
23 : };
24 :
25 : void imgtoarray(char *iname, short a[][960], int *xs, int *ys);
26 : void arraytoimg(short a[][960], char *oname, int xs, int ys);
27 :
28 : void paste(int vx, int vy, struct imgpos img1, short a[][960],
29 :              struct imgpos img2, short b[][960], short c[][YSIZE]);
30 :
31 : int max(int a, int b);
32 : int min(int a, int b);
33 :
34 : main(int argc, char **argv)
35 : {
36 :   IMAGE *iimage1,*iimage2,*oimage;
37 :   short a[1280][960],b[1280][960];  /*入力画像格納用*/
38 :   short c[XSIZE][YSIZE];        /*出力画像格納用*/
39 :   int vx,vy;  /*変位ベクトル*/
40 :   struct imgpos img[10];   /*画像の位置、サイズに関する情報を格納*/
41 :   struct imgpos *sp;
42 :   char name[20];
43 :   int xsizea,ysizea,xsizeb,ysizeb;  /*画像のサイズ*/
44 :   int i,j;
45 :   int n;
46 :   int numimg;
47 :   int g1x,g1y,g2x,g2y;  /*各画像の重心の座標*/
48 :   float m;  /*垂直二等分線の傾き*/
49 :   float d;  /*線分上の位置および合成のパラメタ*/
50 :   int maxx,maxy,minx,miny;
51 :
52 :   if(argc < 2){
53 :           fprintf(stderr,"usage : sqimgsyn oimage.rgb\n");
54 :           exit(1);
55 :   }
56 :
57 :   sp = img;  /*ポインタの初期化*/
58 :
59 :   printf("input filename of image1.\n");
60 :   gets(name);
61 :   imgtoarray(name,a,&xsizea,&ysizea);     /*1枚目の画像の格納*/
62 :   strcpy(sp->name,name);                  /*構造体にデータを入力*/
63 :   sp->xs = xsizea; sp->ys = ysizea;
64 :   sp->swx = (XSIZE - xsizea) / 2; sp->swx = (YSIZE - ysizea) / 2;
65 :   sp++;
66 :
```



```
67 :   printf("input filename of image2.\n");
68 :   gets(name);
69 :   imgtoarray(name,b,&xsizeb,&ysizeb);     /*2枚目の画像の格納*/
70 :   strcpy(sp->name,name);                  /*構造体にデータを入力*/
71 :   sp->xs = xsizeb; sp->ys = ysizeb;
72 :
73 :   /*変位ベクトルの入力*/
74 :   printf("input x-factor of displacement vector.\n");
75 :   scanf("%d",&vx);
76 :   printf("input y-factor of displacement vector.\n");
77 :   scanf("%d",&vy);
78 :
79 :   sp->swx = img[0].swx - vx; sp->swy = img[0].swy - vy;
80 :   sp++;
81 :
82 :   maxx = max(img[0].swx + img[0].xs,img[1].swx + img[1].xs);
83 :   maxy = max(img[0].swy + img[0].ys,img[1].swy + img[1].ys);
84 :   minx = min(img[0].swx,img[1].swx); miny = min(img[0].swy,img[1].swy);
85 :
86 :   /*画像の合成その1--重なり無視--*/
87 :   for(i=img[1].swx;i<img[1].swx+img[1].xs;i++){
88 :           for(j=img[1].swy;j<img[1].swy+img[1].ys;j++){
89 :                   c[i][j] = b[i-img[1].swx][j-img[1].swy];
90 :           }
91 :   }
92 :
93 :   for(i=img[0].swx;i<img[0].swx+img[0].xs;i++){
94 :           for(j=img[0].swy;j<img[0].swy+img[0].ys;j++){
95 :                   c[i][j] = a[i-img[0].swx][j-img[0].swy];
96 :           }
97 :   }
98 :
99 :   g1x = img[0].swx + img[0].xs / 2; g1y = img[0].swy + img[0].ys / 2;
100 :  g2x = img[1].swx + img[1].xs / 2; g2y = img[1].swy + img[1].ys / 2;
101 :  m = - ((float)vx) / ((float)vy);     /*垂直二等分線の傾き*/
102 :  if(vy <= 0){
103 :          for(i=img[1].swx;i<img[0].swx+img[0].xs;i++){
104 :                  for(j=img[1].swy;j<img[0].swy+img[0].ys;j++){
105 :                          if(j <= (int)(m * (i - g1x) + g1y)){
106 :                                  continue;
107 :                          }
108 :                          else if(j >= (int)(m * (i - g2x) + g2y)){
109 :                                  c[i][j] = b[i-img[1].swx][j-img[1].swy];
110 :                          }
111 :                          else{
112 :                                  d = (j-m*i+m*g2x-g2y) / (vy-m*vx);
113 :                                  c[i][j] = (short)(d*a[i-img[0].swx][j-img[0].
swy]
114 :                                                                  +(1-d)*b[i-
img[1].swx][j-img[1].swy]);
115 :                          }
116 :                  }
117 :          }
118 :  }
119 :  else{
120 :          for(i=img[1].swx;i<img[0].swx+img[0].xs;i++){
121 :                  for(j=img[1].swy;j<img[0].swy+img[0].ys;j++){
122 :                          if(j >= (int)(m * (i - g1x) + g1y)){
123 :                                  continue;
124 :                          }
125 :                          else if(j <= (int)(m * (i - g2x) + g2y)){
126 :                                  c[i][j] = b[i-img[1].swx][j-img[1].swy];
127 :                          }
128 :                          else{
129 :                                  d = (j-m*i+m*g2x-g2y) / (vy-m*vx);
130 :                                  c[i][j] = (short)(d*a[i-img[0].swx][j-img[0].
```

```
swy]
131 :                                                                    +(1-d)*b[i-
img[1].swx][j-img[1].swy]);
132 :                                }
133 :                        }
134 :                }
135 :    }
136 :    printf("calculating done.\n");
137 :    numimg = 1;
138 :
139 :    while(1){
140 :            printf("input image number\n");
141 :            for(i=0;i<=numimg;i++){
142 :                    printf("%d. %s\n",i,img[i].name);
143 :            }
144 :            printf("%d. quit\n",i);
145 :            scanf("%d",&n);
146 :            (void)getchar();
147 :            if(n > numimg)
148 :              break;
149 :            imgtoarray(img[n].name,a,&xsizea,&ysizea);
150 :            printf("input filename of image\n");
151 :            gets(name);
152 :            imgtoarray(name,b,&xsizeb,&ysizeb);
153 :            strcpy(sp->name,name);        /*構造体にデータを入力*/
154 :            sp->xs = xsizeb; sp->ys = ysizeb;
155 :
156 :            /*変位ベクトルの入力*/
157 :            printf("input x-factor of displacement vector.\n");
158 :            scanf("%d",&vx);
159 :            printf("input y-factor of displacement vector.\n");
160 :            scanf("%d",&vy);
161 :
162 :            sp->swx = img[n].swx - vx; sp->swy = img[n].swy - vy;
163 :            numimg++;
164 :            maxx = max(maxx,sp->swx + sp->xs); maxy = max(maxy,sp->swy + sp->ys);
165 :            minx = min(minx,sp->swx); miny = min(miny,sp->swy);
166 :            sp++;
167 :
168 :            printf("now calculating!\n");
169 :            paste(vx, vy , img[n], a, img[numimg], b, c);
170 :    }
171 :
172 :    printf("now calculating synthesized image.\n");
173 :
174 :    /*不要部分の削除*/
175 :    for(i=minx;i<maxx;i++){
176 :            for(j=miny;j<maxy;j++){
177 :                    c[i-minx][j-miny] = c[i][j];
178 :            }
179 :    }
180 :
181 :    arraytoimg(c, argv[1],maxx-minx+1,maxy-miny+1); /*出力用画像の生成*/
182 :    printf("calculating done.\n");
183 : }
184 :
185 :
186 : void imgtoarray(char *iname, short a[][960], int *xs, int *ys)
187 : {
188 :      IMAGE *iimage;
189 :      unsigned int xsize, ysize;
190 :      unsigned int x,y;
191 :    short ibuf[960];
192 :
193 :      if( (iimage=iopen(iname,"r")) == NULL ) {
194 :    fprintf(stderr,"imgtoarray: can't open input file %s\n",*iname);
```



```
195 :    exit(1);
196 :      }
197 :      xsize = iimage->xsize;
198 :      ysize = iimage->ysize;
199 :
200 :    *xs = xsize; *ys = ysize;
201 :
202 :      for(y=0; y<ysize; y++) {
203 :        getrow(iimage,ibuf,y,0);
204 :            for(x=0; x<xsize; x++){
205 :                    a[x][y] = ibuf[x];
206 :            }
207 :    }
208 :      iclose(iimage);
209 :      return;
210 : }
211 :
212 :
213 : void arraytoimg(short a[][960], char *oname, int xs, int ys)
214 : {
215 :    IMAGE *oimage;
216 :    unsigned int xsize, ysize;
217 :    unsigned int x,y;
218 :      short    obuf[1280];
219 :
220 :    xsize = xs; ysize = ys;
221 :    oimage = iopen(oname,"w",RLE(1),2,xsize,ysize);
222 :    for(y=0;y<ysize;y++){
223 :            for(x=0;x<xsize;x++){
224 :                    obuf[x] = a[x][y];
225 :            }
226 :            putrow(oimage,obuf,y,0);
227 :    }
228 :    iclose(oimage);
229 :    return;
230 : }
231 : void paste(int vx, int vy, struct imgpos img1, short a[][960],
232 :                struct imgpos img2, short b[][960], short c[][YSIZE])
233 : {
234 :    int i,j;
235 :    int g1x,g1y,g2x,g2y;
236 :    float m,d;
237 :
238 :    for(i=img2.swx;i<img2.swx+img2.xs;i++){
239 :                    for(j=img2.swy;j<img2.swy+img2.ys;j++){
240 :                            if(((i>=img2.swx)&&(i<img1.swx+img1.xs)) &&
241 :                               ((j>=img2.swy)&&(j<img1.swy+img1.ys))){
242 :                                    continue;
243 :                            }
244 :                            else{
245 :                                    c[i][j] = b[i-img2.swx][j-img2.swy];
246 :                            }
247 :                    }
248 :            }
249 :    g1x = img1.swx + img1.xs / 2; g1y = img1.swy + img1.ys / 2;
250 :    g2x = img2.swx + img2.xs / 2; g2y = img2.swy + img2.ys / 2;
251 :    m = - ((float)vx) / ((float)vy);      /*垂直二等分線の傾き*/
252 :    if(vy <= 0){
253 :            for(i=img2.swx;i<img1.swx+img1.xs;i++){
254 :                    for(j=img2.swy;j<img1.swy+img1.ys;j++){
255 :                            if(j <= (int)(m * (i - g1x) + g1y)){
256 :                                    continue;
257 :                            }
258 :                            else if(j >= (int)(m * (i - g2x) + g2y)){
259 :                                    c[i][j] = b[i-img2.swx][j-img2.swy];
260 :                            }
```

```
261 :                                   else{
262 :                                           d = (j-m*i+m*g2x-g2y) / (vy-m*vx);
263 :                                           c[i][j] = (short)(d*a[i-img1.swx][j-img1.swy]
264 :                                                                   +(1-d)*b[i-
img2.swx][j-img2.swy]);
265 :                                   }
266 :                           }
267 :                   }
268 :           }
269 :           else{
270 :                   for(i=img2.swx;i<img1.swx+img1.xs;i++){
271 :                           for(j=img2.swy;j<img1.swy+img1.ys;j++){
272 :                                   if(j <= (int)(m * (i - g1x) + g1y)){
273 :                                           continue;
274 :                                   }
275 :                                   else if(j >= (int)(m * (i - g2x) + g2y)){
276 :                                           c[i][j] = b[i-img2.swx][j-img2.swy];
277 :                                   }
278 :                                   else{
279 :                                           d = (j-m*i+m*g2x-g2y) / (vy-m*vx);
280 :                                           c[i][j] = (short)(d*a[i-img1.swx][j-img1.swy]
281 :                                                                   +(1-d)*b[i-
img2.swx][j-img2.swy]);
282 :                                   }
283 :                           }
284 :                   }
285 :           }
286 :           printf("caluculating done.\n");
287 :           return;
288 : }
289 :
290 : int max(int a, int b)
291 : {
292 :           if(a >= b){
293 :                   return(a);
294 :           }
295 :           else{
296 :                   return(b);
297 :           }
298 : }
299 :
300 : int min(int a, int b)
301 : {
302 :           if(a >= b){
303 :                   return(b);
304 :           }
305 :           else{
306 :                   return(a);
307 :           }
308 : }
```

```
 1 : /*
 2 :  *       sqimgsyna.c -
 3 :  *
 4 :  *       2枚の画像の間のアフィン変換パラメタをLevenberg-Marquardt Method
 5 :  *       によって計算し、一方の画像を計算されたパラメタで変換して、連続的に
 6 :  *       合成していくプログラム
 7 :  *
 8 :  *       西羅 光  -1996
 9 :  */
10 : #include <stdio.h>
11 : #include <math.h>
12 : #include "image.h"
13 :
14 : #define SQR(a) ((a)*(a))
15 :
16 : #define GX 640    /*画面の重心*/
17 : #define GY 480
18 :
19 : #define EPSIRON 0.001
20 : #define SR 10
21 :
22 : struct imgdata{
23 :   char name[20];  /*ファイル名*/
24 :   short a[1280][960];  /*画素を格納*/
25 :   int xs;       /*画像のサイズ*/
26 :   int ys;       /*画像のサイズ*/
27 :   int swx;      /*画像の変換後の左下角のx座標*/
28 :   int swy;      /*画像の変換後の左下角のy座標*/
29 :   int gx;       /*画像の重心のx座標*/
30 :   int gy;       /*画像の重心のx座標*/
31 :   float m[6];   /*アフィン変換パラメタ*/
32 :   struct imgdata *p;  /*変換時に基準にとった画像*/
33 : };
34 :
35 : void imgtoarray(char *iname, short a[][960], int *xs ,int *ys);
36 : void arraytoimg(short a[][960], char *oname, int xs, int ys);
37 :
38 : float diff(int z, short a[][960], int x, int y);  /*微分演算*/
39 : void sol(float a[6][6], float b[6], float x[6]);  /*連立一次方程式の解*/
40 : void inv(float a[6][6], float b[6][6]);  /*逆行列を求める*/
41 :
42 : void aftrsf(int x0, int y0, int *x, int *y, float m[6]); /*アフィン変換*/
43 : void aftrsf2(int x0, int y0, float *x, float *y, float m[6]); /*アフィン変換*/
44 : void invaftrsf(int x, int y, int *x0, int *y0, float m[6]); /*アフィン逆変換*/
45 : void aftrsfsyn(float l[6], float m[6], float n[6]);  /*合成変換のパラメタ*/
46 :
47 : float err(short a[][960], short b[][960], int x0, int xe, int xs,
48 :             int y0, int ye, int ys, float m[6]); /*二乗誤差関数*/
49 : int grad(short b[][960], float m[6], int x0, int y0, float g[6]);
50 : void setbeta(float beta[6], short a[][960], short b[][960],
51 :                    int x0, int xe, int xs, int y0, int ye, int ys, float m[6]);
52 : void setalpha(float alpha[6][6], short a[][960],short b[][960],int x0, int xe,
53 :                     int xs, int y0, int ye, int ys, float m[6], float lamda);
54 :
55 : void mrqmina(char *name1, char *name2, float m[6]);  /*パラメタを推定*/
56 :
57 : float dpp(float x1, float y1, float x2, float y2);  /*点と点の距離*/
58 : void el(int x1, int y1, int x2, int y2, float l[3]);
59 :                                                    /*2点を結ぶ直線*/
60 : void vl(int x1, int y1, int x2, int y2, float l[3]);
61 :                                                    /*垂直二等分線*/
62 : float dpl(int x, int y, float l[3]); /*点と直線の距離*/
63 : int cp(float l1[3], float l2[3], int *x, int *y);   /*2直線の交点*/
64 :
65 : int intriangle(struct imgdata *a, struct imgdata *b, struct imgdata *c,
66 :                      int i, int j);        /*三角形の内部にあるかどうかの判定*/
```



```
67 :
68 : void getmaxmin(int gx, int gy, int xs, int ys,
69 :                       int *mx, int *may, int *mix, int *miy, float m[6]);
70 :                                                    /*走査範囲の決定*/
71 : void intpl(short a[][960], int swx, int swy, int maxx, int maxy,
72 :               int minx, int miny, short c[][960], float m[6]);
73 :                                                    /*変換画像の内挿*/
74 : short doublesyn(struct imgdata *a, struct imgdata *b, int i, int j);
75 :                                                    /*画素の二重合成*/
76 : short triplesyn(struct imgdata *a, struct imgdata *b, struct imgdata *c,
77 :                            int i, int j);                  /*画素の三重合成*
/
78 :
79 : int max(int a, int b);
80 : float maxf(float a, float b);
81 : int min(int a, int b);
82 : float minf(float a, float b);
83 : float absf(float a);
84 :
85 : main()
86 : {
87 :     short a[1280][960];  /*入力画像の格納*/
88 :   short c[1280][960];  /*出力画像の格納*/
89 :   short d[1280][960];  /*作業用配列*/
90 :   short s,sl;
91 :   float m[6],ml[6],m2[6],ms[6]; /*アフィン変換パラメタ*/
92 :   int xsize,ysize;  /*画像のサイズ*/
93 :   int i,j,k,l,x,y,i0,j0,x0,y0,x1,y1;
94 :   int maxx,maxy,minx,miny;  /*全体の大きさ*/
95 :   float da,db,ab,h;
96 :   float dv,ds,dt;
97 :   struct imgdata *img;
98 :   struct imgdata *sp,*wp;
99 :   struct imgdata *sa,*sb,*sc;
100 :   struct imgdata *gv,*gs,*gt;
101 :   int n,num;
102 :   char name[20];
103 :   int bufa,bufb,bufc,bufd;   /*作業用変数*/
104 :
105 :   printf("Sequential Image Synthesize.\n");
106 :   while(1){
107 :           printf("How many images will you synthesize?\n");
108 :           scanf("%d",&i);
109 :           (void)getchar();
110 :           img = (struct imgdata *)malloc(sizeof(struct imgdata) * i);
111 :           if(img == NULL)
112 :             printf("can't get memory!\n");
113 :           else
114 :             break;
115 :   }
116 :   n = 0;
117 :   sp = img;
118 :   /*1枚目の画像の入力*/
119 :   printf("input filename of first image.\n");
120 :   gets(name);
121 :   imgtoarray(name,sp->a,&xsize,&ysize);
122 :   /*構造体にデータを入力*/
123 :   strcpy(sp->name,name);
124 :   sp->xs = xsize; sp->ys = ysize;
125 :   sp->swx = GX - xsize / 2; sp->swy = GY - ysize / 2;
126 :   sp->gx = GX; sp->gy = GY;
127 :   sp->m[0] = 1; sp->m[1] = 0; sp->m[2] = 0;
128 :   sp->m[3] = 0; sp->m[4] = 1; sp->m[5] = 0;
129 :   sp->p = sp;
130 :   /*全体のサイズの初期化*/
131 :   minx = sp->swx; miny = sp->swy;
```

```
132 :   maxx = sp->swx + sp->xs; maxy = sp->swy + sp->ys;
133 :   /*出力用配列への格納*/
134 :   for(i=minx;i<maxx;i++){
135 :           for(j=miny;j<maxy;j++){
136 :                   c[i][j] = sp->a[i-sp->swx][j-sp->swy];
137 :           }
138 :   }
139 :   n++;
140 :   sp++;
141 :   /*2枚目の画像の入力*/
142 :   printf("input filename of second image.\n");
143 :   gets(name);
144 :   imgtoarray(name,sp->a,&xsize,&ysize);
145 :   /*構造体にデータを入力*/
146 :   strcpy(sp->name,name);
147 :   sp->xs = xsize; sp->ys = ysize;
148 :   mrqmina(sp->name,img->name,m);  /*パラメタの推定*/
149 :   aftrsf(img->swx,img->swy,&bufa,&bufb,m);
150 :   sp->swx = bufa; sp->swy = bufb;
151 :   aftrsf(img->gx,img->gy,&bufa,&bufb,m);
152 :   sp->gx = bufa; sp->gy = bufb;
153 :   for(i=0;i<=5;i++)
154 :     sp->m[i] = m[i];
155 :   sp->p = sp - 1;
156 :   /*2枚目の画像の貼付け*/
157 :   getmaxmin(GX,GY,sp->xs,sp->ys,&bufa,&bufb,&bufc,&bufd,m);
158 :   intpl(sp->a,img->swx,img->swy,bufa,bufb,bufc,bufd,c,m);
159 :   /*全体のサイズの更新*/
160 :   maxx = max(maxx,bufa); maxy = max(maxy,bufb);
161 :   minx = min(minx,bufc); miny = min(miny,bufd);
162 :   /*領域判別ループ*/
163 :   for(i=minx;i<maxx;i++){
164 :           for(j=miny;j<maxy;j++){
165 :                   invaftrsf(i,j,&x,&y,m);
166 :                   /*上書きされた領域の修復*/
167 :                   db = dpp(i,j,sp->gx,sp->gy); da = dpp(i,j,img->gx,img->gy);
168 :                   /*indexの変換式*/
169 :                   i0 = i - img->swx; j0 = j - img->swy;
170 :                   x0 = x - img->swx; y0 = y - img->swy;
171 :                   if((i0>=0)&&(i0<xsize)&&(j0>=0)&&(j0<ysize)){
172 :                           if(da <= db)
173 :                             c[i][j] = img->a[i0][j0];
174 :                           else if((x0<0)||(x0>=xsize)||(y0<0)||(y0>=ysize))
175 :                             c[i][j] = img->a[i0][j0];
176 :                   }
177 :                   /*範囲外の領域の処理*/
178 :                   if((i0<0)||(i0>=xsize)||(j0<0)||(j0>=ysize)
179 :                      ||(x0<0)||(x0>=xsize)||(y0<0)||(y0>=ysize))
180 :                     continue;
181 :                   /*合成領域の判別と内挿*/
182 :                   sa = img; sb = sp;
183 :                   s = doublesyn(sa,sb,i,j);
184 :                   if(s != NULL)
185 :                     c[i][j] = s;
186 :           }
187 :   }
188 :   /*連続貼付けループ*/
189 :   while(1){
190 :           n++;
191 :           sp++;
192 :           printf("input base image number.\n");
193 :           for(i=0;i<n;i++)
194 :             printf("%d. %s\n",i,img[i].name);
195 :           printf("%d. quit\n",i);
196 :           scanf("%d",&num);
197 :           (void)getchar();
```



```
198 :           if(num >= n)
199 :             break;
200 :           printf("input filename of the new image.\n");
201 :           gets(name);
202 :           imgtoarray(name,sp->a,&xsize,&ysize);
203 :           /*構造体にデータを入力*/
204 :           strcpy(sp->name,name);
205 :           sp->xs = xsize; sp->ys = ysize;
206 :           sp->p = img + num;
207 :           mrqmina(sp->name,(img+num)->name,m);     /*パラメタの推定*/
208 :           /*合成変換パラメタの決定*/
209 :           wp = sp;
210 :           for(i=0;i<=5;i++)
211 :             ms[i] = m[i];
212 :           do{
213 :                   for(i=0;i<=5;i++)
214 :                     m1[i] = ms[i];
215 :                   wp = wp->p;
216 :                   for(i=0;i<=5;i++)
217 :                     m2[i] = wp->m[i];
218 :                   aftrsfsyn(m1,m2,ms);
219 :           }while(wp != img);
220 :           aftrsf(img->swx,img->swy,&bufa,&bufb,ms);
221 :           sp->swx = bufa; sp->swy = bufb;
222 :           aftrsf(img->gx,img->gy,&bufa,&bufb,ms);
223 :           sp->gx = bufa; sp->gy = bufb;
224 :           for(i=0;i<=5;i++)
225 :             sp->m[i] = m[i];
226 :           /*新しい画像の（作業用配列への）貼付け*/
227 :           getmaxmin(img->gx,img->gy,sp->xs,sp->ys,
228 :                                     &bufa,&bufb,&bufc,&bufd,ms);
229 :           intpl(sp->a,img->swx,img->swy,bufa,bufb,bufc,bufd,d,ms);
230 :           /*全体のサイズの更新*/
231 :           maxx = max(maxx,bufa); maxy = max(maxy,bufb);
232 :           minx = min(minx,bufc); miny = min(miny,bufd);
233 :           /*白地への貼付け*/
234 :           for(i=bufc;i<bufa;i++){
235 :                   for(j=bufd;j<bufb;j++){
236 :                           if((c[i][j] == 0) && (d[i][j] != 0))
237 :                             c[i][j] = d[i][j];
238 :                   }
239 :           }
240 :           /*領域判別ループ*/
241 :           for(i=img->gx-sp->xs/2;i<img->gx-sp->xs/2+sp->xs;i++){
242 :                   for(j=img->gy-sp->ys/2;j<img->gy-sp->ys/2+sp->ys;j++){
243 :                           aftrsf(i,j,&x,&y,ms);
244 :
245 :                           dv = 5000; ds = 5000; dt = 5000;
246 :                           gs = img; gt = img;
247 :                           for(k=0;k<=n;k++){
248 :                                   l = 1;
249 :                                   if(dpp(x,y,img[k].gx,img[k].gy) <= dv){
250 :                                           gt = gs; gs = gv; gv = img + k;
251 :                                           dt = ds; ds = dv;
252 :                                           dv = dpp(x,y,img[k].gx,img[k].gy);
253 :                                           l = 3;
254 :                                   }
255 :                                   else if(dpp(x,y,img[k].gx,img[k].gy) <= ds){
256 :                                           gt = gs; gs = img + k;
257 :                                           dt = ds;
258 :                                           ds = dpp(x,y,img[k].gx,img[k].gy);
259 :                                           l = 2;
260 :                                   }
261 :                                   else if(dpp(x,y,img[k].gx,img[k].gy) <= dt){
262 :                                           gt = img + k;
263 :                                           dt = dpp(x,y,img[k].gx,img[k].gy);
```

```
264 :                                   }
265 :                                   else
266 :                                     l = 0;
267 :                           }
268 :                           if(l == 0)
269 :                             continue;
270 :                           else if(l == 3){
271 :                                   s = doublesyn(gv,gs,x,y);
272 :                                   if(!s){
273 :                                           if(d[x][y] != 0)
274 :                                             c[x][y] = d[x][y];
275 :                                           continue;
276 :                                   }
277 :                                   if(intriangle(gv,gs,gt,x,y))
278 :                                     c[x][y] = triplesyn(gv,gs,gt,x,y);
279 :                                   else
280 :                                     c[x][y] = s;
281 :                           }
282 :                           else if(l == 2){
283 :                                   s = doublesyn(gv,gs,x,y);
284 :                                   if(!s)
285 :                                     continue;
286 :                                   if(intriangle(gv,gs,gt,x,y))
287 :                                     c[x][y] = triplesyn(gv,gs,gt,x,y);
288 :                                   else
289 :                                     c[x][y] = s;
290 :                           }
291 :                           else{
292 :                                   s = doublesyn(gv,gt,x,y);
293 :                                   s1 = doublesyn(gs,gt,x,y);
294 :                                   if(s*s1){
295 :                                           if(intriangle(gv,gs,gt,x,y))
296 :                                             c[x][y] = triplesyn(gv,gs,gt,x,y);
297 :                                   }
298 :                           }
299 :                   }
300 :           }
301 :   }
302 :   /*不要部分の削除*/
303 :   for(i=minx;i<maxx;i++){
304 :           for(j=miny;j<maxy;j++){
305 :                   c[i-minx][j-miny] = c[i][j];
306 :           }
307 :   }
308 :   printf("input filename of output.\n");
309 :   gets(name);
310 :   arraytoimg(c,name,maxx-minx+1,maxy-miny+1);
311 : }
312 :
313 : void imgtoarray(char *iname , short a[][960] ,int *xs ,int *ys)
314 : {
315 :     IMAGE *iimage;
316 :     unsigned int xsize, ysize;
317 :     unsigned int x,y;
318 :   short   ibuf[960];
319 :
320 :     if( (iimage=iopen(iname,"r")) == NULL ) {
321 :   fprintf(stderr,"imgtoarray: can't open input file %s\n",*iname);
322 :   exit(1);
323 :     }
324 :     xsize = iimage->xsize;
325 :     ysize = iimage->ysize;
326 :
327 :   *xs = xsize; *ys = ysize;
328 :
329 :     for(y=0; y<ysize; y++) {
```



```
330 :       getrow(iimage,ibuf,y,0);
331 :           for(x=0; x<xsize; x++){
332 :                   a[x][y] = ibuf[x];
333 :           }
334 :   }
335 :     iclose(iimage);
336 :     return;
337 : }
338 :
339 : void arraytoimg(short a[][960], char *oname, int xsize, int ysize)
340 : {
341 :   IMAGE *oimage;
342 :   unsigned int x,y;
343 :   short    obuf[1280];
344 :
345 :   oimage = iopen(oname,"w",RLE(1),2,xsize,ysize);
346 :   for(y=0;y<ysize;y++){
347 :           for(x=0;x<xsize;x++){
348 :                   obuf[x] = a[x][y];
349 :           }
350 :           putrow(oimage,obuf,y,0);
351 :   }
352 :   iclose(oimage);
353 :   return;
354 : }
355 :
356 : float diff(int z, short a[][960], int x, int y)
357 : {
358 :   float d;
359 :
360 :   if(z == 1){
361 :           if((x+1)>=640)
362 :             d = (float)(a[x][y] - a[x-1][y]);
363 :           else
364 :             d = (float)(a[x+1][y] - a[x][y]);
365 :   }
366 :   else if(z == 2){
367 :           if((y+1)>=480)
368 :             d = (float)(a[x][y] - a[x][y-1]);
369 :           else
370 :             d = (float)(a[x][y+1] - a[x][y]);
371 :   }
372 :   else{
373 :           d = NULL;
374 :   }
375 :   return(d);
376 : }
377 :
378 : void sol(float a[6][6], float b[6], float x[6])
379 : {
380 :   int i,j,k;
381 :   float c;
382 :
383 :   /*前進消去*/
384 :   for(i=0;i<=5;i++){
385 :           x[i] = 0.0;  /*後退代入のための初期化*/
386 :           for(j=i;j<=5;j++){
387 :                   if(j==i)
388 :                           b[j] = b[j] / a[i][i];
389 :                   else
390 :                           b[j] = b[j] - c * b[i];
391 :                   for(k=j+1;k<=5;k++){
392 :                           if(j==i)
393 :                                   a[j][k] = a[j][k] / a[i][i];
394 :                           else{
395 :                                   c = a[j][i] / a[i][i];
```

```
396 :                                   a[j][k] = a[j][k] - c * a[i][k];
397 :                           }
398 :                   }
399 :           }
400 :   }
401 :   /*後退代入*/
402 :   for(i=5;i>=0;i--){
403 :           c = 0.0;
404 :           for(j=5;j>i;j--){
405 :                   c += a[i][j] * x[j];
406 :           }
407 :           x[i] = b[i] - c;
408 :   }
409 : }
410 :
411 : void inv(float a[6][6], float b[6][6])  /*逆行列を求める*/
412 : {
413 :   float x[6],c[6];
414 :   int i,j,k;
415 :
416 :   for(i=0;i<=5;i++){
417 :           for(j=0;j<=5;j++){
418 :                   if(j==i)
419 :                     c[j] = 1.0;
420 :                   else
421 :                     c[j] = 0.0;
422 :           }
423 :           sol(a,c,x);
424 :           for(k=0;k<=5;k++){
425 :                   b[k][i] = x[k];
426 :           }
427 :   }
428 : }
429 :
430 : void aftrsf(int x0, int y0, int *x, int *y, float m[6])  /*アフィン変換*/
431 : {
432 :   float x1,y1;
433 :
434 :   x1 = m[0] * x0 + m[1] * y0 + m[2];
435 :   y1 = m[3] * x0 + m[4] * y0 + m[5];
436 :   *x = (int)x1;
437 :   *y = (int)y1;
438 : }
439 :
440 : void aftrsf2(int x0, int y0, float *x, float *y, float m[6])  /*アフィン変換*/
441 : {
442 :   *x = m[0] * x0 + m[1] * y0 + m[2];
443 :   *y = m[3] * x0 + m[4] * y0 + m[5];
444 : }
445 :
446 : void invaftrsf(int x, int y, int *x0, int *y0, float m[6]) /*アフィン逆変換*/
447 : {
448 :   float x1,y1;
449 :   float det;
450 :
451 :   det = m[0] * m[4] - m[1] * m[3];
452 :   x1 = ( m[4] * x - m[1] * y + m[1] * m[5] - m[2] * m[4]) / det;
453 :   y1 = (-m[3] * x + m[0] * y + m[3] * m[2] - m[0] * m[5]) / det;
454 :   *x0 = (int)x1;
455 :   *y0 = (int)y1;
456 : }
457 :
458 : void aftrsfsyn(float l[6], float m[6], float n[6])
459 : {
460 :   n[0] = l[0] * m[0] + l[1] * m[3];
461 :   n[1] = l[0] * m[1] + l[1] * m[4];
```



```
462 :   n[2] = l[0] * m[2] + l[1] * m[5] + l[2];
463 :   n[3] = l[3] * m[0] + l[4] * m[3];
464 :   n[4] = l[3] * m[1] + l[4] * m[4];
465 :   n[5] = l[3] * m[2] + l[4] * m[5] + l[5];
466 : }
467 :
468 : float err(short a[][960], short b[][960], int x0, int xe, int xs,
469 :             int y0, int ye, int ys, float m[6]) /*二乗誤差関数*/
470 : {
471 :   int i,j;
472 :   int x,y;
473 :   int nx,ny;
474 :   int s; /*画素数*/
475 :   float e;
476 :
477 :   e = 0.0;
478 :   nx = (xe - x0) / SR; ny = (ye - y0) / SR;
479 :   s = 0;
480 :
481 :   for(i=x0+nx;i<xe;i+=nx){
482 :           for(j=y0+ny;j<ye;j+=ny){
483 :                   aftrsf(i,j,&x,&y,m);
484 :                   if((x<0) || (y<0) || (x>=xs) || (y>=ys))
485 :                     continue;
486 :                   else{
487 :                           e += SQR(b[x][y] - a[i][j]);
488 :                           s++;
489 :                   }
490 :           }
491 :   }
492 :   return(e/s);
493 : }
494 :
495 : int grad(short b[][960], float m[6], int x0, int y0, float g[6])
496 : {
497 :   int t[6];
498 :   int x,y;
499 :   int i;
500 :
501 :   t[0] = x0; t[3] = x0;
502 :   t[1] = y0; t[4] = y0;
503 :   t[2] =  1; t[5] =  1;
504 :
505 :   aftrsf(x0,y0,&x,&y,m);
506 :   if((x<0) || (y<0) || (x>=640) || (y>=480))
507 :     return(1);
508 :
509 :   for(i=0;i<=2;i++)
510 :     g[i] = diff(1,b,x,y) * t[i];
511 :   for(i=3;i<=5;i++)
512 :     g[i] = diff(2,b,x,y) * t[i];
513 :
514 :   return(0);
515 : }
516 :
517 : void setbeta(float beta[6], short a[][960], short b[][960],
518 :                     int x0, int xe,int xs, int y0, int ys, int ye, float m[6])
519 : {
520 :   int i,j,k;
521 :   int x,y;
522 :   int nx,ny;
523 :   float g[6];
524 :
525 :   nx = (xe - x0) / SR; ny = (ye - y0) / SR;
526 :   for(k=0;k<=5;k++){
527 :           beta[k] = 0.0;
```

```
528 :           for(i=x0+nx;i<xe;i+=nx){
529 :                   for(j=y0+ny;j<ye;j+=ny){
530 :                           aftrsf(i,j,&x,&y,m);
531 :                           if((x<0) || (y<0) || (x>=xs) || (y>=ys))
532 :                             continue;
533 :                           else if(grad(b,m,i,j,g))
534 :                             continue;
535 :                           else
536 :                             beta[k] += (b[x][y] - a[i][j]) * g[k];
537 :                   }
538 :           }
539 :           beta[k] = -1 * beta[k];
540 :   }
541 :   return;
542 : }
543 :
544 : void setalpha(float alpha[6][6], short a[][960], short b[][960], int x0,
545 :                     int xe, int xs, int y0, int ye, int ys,  float m[6], float
lamda)
546 : {
547 :   int i,j,k,l;
548 :   int x,y;
549 :   int nx,ny;
550 :   float g[6];
551 :
552 :   nx = (xe - x0) / SR; ny = (ye - y0) / SR;
553 :   for(k=0;k<=5;k++){
554 :           for(l=k;l<=5;l++){
555 :                   alpha[k][l] = 0.0;
556 :                   for(i=x0+nx;i<xe;i+=nx){
557 :                           for(j=y0+ny;j<ye;j+=ny){
558 :                                   aftrsf(i,j,&x,&y,m);
559 :                                   if((x<0) || (y<0) || (x>=xs) || (y>=ys))
560 :                                     continue;
561 :                                   else if(grad(b,m,i,j,g))
562 :                                     continue;
563 :                                   else
564 :                                     alpha[k][l] += g[k] * g[l];
565 :                           }
566 :                   }
567 :                   if(k==l)
568 :                     alpha[k][l] *= 1 + lamda;
569 :                   else
570 :                     alpha[l][k] = alpha[k][l];
571 :           }
572 :   }
573 :   return;
574 : }
575 :
576 : void mrqmina(char *name1, char *name2, float m[6])
577 : {
578 :   short a[1280][960],b[1280][960];
579 :   float beta[6];      /*誤差関数の勾配ベクトル*/
580 :   float alpha[6][6]; /*ヘッセ行列*/
581 :   float cov[6][6];    /*推定値の誤差共分散行列*/
582 :   float lamda;        /*安定化因子*/
583 :   float dm[6];        /*mの修正値*/
584 :   int x0,y0,xe,ye;  /*代表点に関する情報*/
585 :   float e,eo,en;    /*二乗誤差関数の値*/
586 :   int xsize,ysize;
587 :   int i,j;
588 :   int r;    /*反復回数をカウント*/
589 :   int xdmy,ydmy;
590 :
591 :   imgtoarray(name1,a,&xsize,&ysize);
592 :   imgtoarray(name2,b,&xsize,&ysize);
```



```
593 :
594 :   /*初期値（平行移動成分）の入力*/
595 :   printf("input estimate of x-factor of vector %s to %s\n",name1,name2);
596 :   scanf("%d",&xdmy);
597 :   printf("input estimate of y-factor of vector %s to %s\n",name1,name2);
598 :   scanf("%d",&ydmy);
599 :   m[2] = (float)xdmy; m[5] = (float)ydmy;
600 :   m[0] = 1.0; m[1] = 0.0; m[3] = 0.0; m[4] = 1.0;
601 :
602 :   /*代表点の決定*/
603 :   x0 = max(0,-xdmy); xe = min(xsize,xsize-xdmy);
604 :   y0 = max(0,-ydmy); ye = min(ysize,ysize-ydmy);
605 :
606 :   /*初期化*/
607 :   e = err(a,b,x0,xe,xsize,y0,ye,ysize,m);
608 :   lamda = 0.001;
609 :   printf("e = %f\n",e);
610 :   r = 0;
611 :
612 :   /*ループ*/
613 :   do{
614 :           r++;
615 :           setbeta(beta,a,b,x0,xe,xsize,y0,ye,ysize,m);          /*ベータの設定*/
616 :           setalpha(alpha,a,b,x0,xe,xsize,y0,ye,ysize,m,lamda); /*アルファの設定
*/
617 :
618 :           sol(alpha,beta,dm);  /*連立方程式の解*/
619 :
620 :           for(i=0;i<=5;i++)
621 :                   m[i] += dm[i];
622 :           en = err(a,b,x0,xe,xsize,y0,ye,ysize,m);
623 :           if(en > e){     /*誤差関数の評価*/
624 :                   lamda *= 10.0;
625 :                   for(i=0;i<=5;i++)
626 :                     m[i] -= dm[i];
627 :                   continue;
628 :           }
629 :           else{
630 :                   lamda *= 0.1;
631 :                   eo = e;
632 :                   e = en;
633 :                   printf("e = %f\n",e);
634 :           }
635 :   }while((en > eo) || (eo - en) / eo > EPSIRON);      /*継続条件*/
636 :
637 :   printf("%d times  iteration.\n",r);
638 :   /*推定値の表示*/
639 :   printf("calculated estimates are below.\n");
640 :   for(i=0;i<=5;i++)
641 :           printf("m[%d] = %f\n",i,m[i]);
642 :   /*誤差共分散行列の表示*/
643 :   lamda = 0.0;
644 :   setalpha(alpha,a,b,x0,xe,xsize,y0,ye,ysize,m,lamda);
645 :   inv(alpha,cov);
646 :   printf("covariance matrix of the standard errors\n");
647 :   printf("in the fitted parameters m are below.\n");
648 :   for(i=0;i<=5;i++){
649 :           for(j=0;j<=5;j++){
650 :                   printf("%.4f ",cov[i][j]);
651 :           }
652 :           printf("\n");
653 :   }
654 :   return;
655 : }
656 :
657 : float dpp(float x1, float y1, float x2, float y2)  /*点と点の距離*/
```

```
658 : {
659 :   float d;
660 :
661 :   d = SQR(x1 - x2) + SQR(y1 - y2);
662 :   return(sqrt(d));
663 : }
664 :
665 : void el(int x1, int y1, int x2, int y2, float l[3])
666 : {
667 :   l[0] = (float)(y1 - y2);
668 :   l[1] = (float)(x2 - x1);
669 :   l[2] = (float)((x1 - x2) * y1 - x1 * (y1 - y2));
670 : }
671 :
672 : void vl(int x1, int y1, int x2, int y2, float l[3])
673 : {
674 :   l[0] = (float)(x2 - x1);
675 :   l[1] = (float)(y2 - y1);
676 :   l[2] = -(float)((SQR(y2) - SQR(y1) + SQR(x2) - SQR(x1))) / 2;
677 : }
678 :
679 : float dpl(int x, int y, float l[3])
680 : {
681 :   float d;
682 :
683 :   d = (l[0] * x + l[1] * y + l[2]) / sqrt(SQR(l[0]) + SQR(l[1]));
684 :   return(d);
685 : }
686 :
687 : int cp(float l1[3], float l2[3], int *x, int *y)
688 : {
689 :   float det;
690 :
691 :   det = l1[0] * l2[1] - l2[0] * l1[1];
692 :   if(det == 0.0)
693 :     return(0);
694 :   *x = (int)(- (l2[1] * l1[2] - l1[1] * l2[2]) / det);
695 :   *y = (int)(- (l1[0] * l2[2] - l2[0] * l1[2]) / det);
696 :   return(1);
697 : }
698 :
699 : int intriangle(struct imgdata *a, struct imgdata *b, struct imgdata *c,
700 :                int i, int j)
701 : {
702 :   float tgx,tgy;  /*三角形の重心の座標*/
703 :   float lab[3],lbc[3],lca[3];
704 :   float e,f;
705 :   float maxx,maxy,minx,miny;
706 :   float ir,jr;
707 :
708 :   tgx = (a->gx + b->gx + c->gx) / 3.0;
709 :   tgy = (a->gy + b->gy + c->gy) / 3.0;
710 :   el(a->gx,a->gy,b->gx,b->gy,lab);
711 :   el(b->gx,b->gy,c->gx,c->gy,lbc);
712 :   el(c->gx,c->gy,a->gx,a->gy,lca);
713 :
714 :   e = dpl(tgx,tgy,lab) * dpl(tgx,tgy,lbc) * dpl(tgx,tgy,lca);
715 :   f = dpl(i,j,lab) * dpl(i,j,lbc) * dpl(i,j,lca);
716 :   if((e * f) <= 0)
717 :     return(0);   /*三角形の内部にない*/
718 :   maxx = maxf(a->gx,maxf(b->gx,c->gx));
719 :   maxy = maxf(a->gy,maxf(b->gy,c->gy));
720 :   minx = minf(a->gx,minf(b->gx,c->gx));
721 :   miny = minf(a->gy,minf(b->gy,c->gy));
722 :   ir = (float)i; jr = (float)j;
723 :   if((ir>=maxx)||(ir<=miny)||(jr>=maxy)||(jr<=miny))
```



```
724 :     return(0);  /*三角形の内部にない*/
725 :   else
726 :     return(1);  /*三角形の内部にある*/
727 : }
728 :
729 : void getmaxmin(int gx, int gy, int xs, int ys,
730 :                  int *mx, int *may, int *mix, int *miy, float m[6])
731 : {
732 :   int i,j;
733 :   int x0,y0;
734 :   int maxx,maxy,minx,miny;
735 :
736 :   maxx = 0; maxy = 0; minx = 1280; miny = 960;
737 :   /*エッジに沿って走査*/
738 :   for(i=gx-xs/2;i<gx-xs/2+xs;i++){
739 :           aftrsf(i,gy-ys/2,&x0,&y0,m);
740 :           maxx = max(x0,maxx); maxy = max(y0,maxy);
741 :           minx = min(x0,minx); miny = min(y0,miny);
742 :   }
743 :   for(j=gy-ys/2;j<gy-ys/2+ys;j++){
744 :           aftrsf(gx-xs/2+xs-1,j,&x0,&y0,m);
745 :           maxx = max(x0,maxx); maxy = max(y0,maxy);
746 :           minx = min(x0,minx); miny = min(y0,miny);
747 :   }
748 :   for(i=gx-xs/2+xs-1;i>=gx-xs/2;i--){
749 :           aftrsf(i,gy-ys/2+ys-1,&x0,&y0,m);
750 :           maxx = max(x0,maxx); maxy = max(y0,maxy);
751 :           minx = min(x0,minx); miny = min(y0,miny);
752 :   }
753 :   for(j=gy-ys/2+ys-1;j>=gy-ys/2;j--){
754 :           aftrsf(gx-xs/2,j,&x0,&y0,m);
755 :           maxx = max(x0,maxx); maxy = max(y0,maxy);
756 :           minx = min(x0,minx); miny = min(y0,miny);
757 :   }
758 :   maxx = min(1280,maxx); maxy = min(960,maxy);
759 :   minx = max(0,minx); miny = max(0,miny);
760 :
761 :   *mx = maxx; *may = maxy; *mix = minx; *miy = miny;
762 : }
763 :
764 :
765 : void intpl(short a[][960], int swx, int swy, int maxx, int maxy,
766 :            int minx, int miny, short c[][960], float m[6])
767 : {
768 :   int x,y;
769 :   int x0,y0;
770 :   int i,j,k,l;
771 :   float xf,yf;
772 :   float xbuf[10],ybuf[10],d[10];
773 :   int ibuf[10],jbuf[10];
774 :   float s;
775 :   float d0;
776 :
777 :   for(x=minx;x<maxx;x++){
778 :           for(y=miny;y<maxy;y++){
779 :                   k = 0;
780 :                   invaftrsf(x,y,&x0,&y0,m);
781 :                   for(i=(x0-1);i<=(x0+1);i++){
782 :                           for(j=(y0-1);j<=(y0+1);j++){
783 :                                   aftrsf2(i,j,&xf,&yf,m);
784 :                                   if(dpp(x,y,xf,yf) < 1.0){
785 :                                           xbuf[k] = xf; ybuf[k] = yf;
786 :                                           ibuf[k] = i;  jbuf[k] = j;
787 :                                           k++;
788 :                                   }
789 :                           }
```

```
790 :                   }
791 :                   s = 0.0;
792 :                   for(l=0;l<k;l++){
793 :                           d[l] = 1 - dpp(x,y,xbuf[l],ybuf[l]);
794 :                           s += d[l];
795 :                   }
796 :                   c[x][y] = 0;
797 :                   for(l=0;l<k;l++)
798 :                     c[x][y] += (short)(d[l] / s * a[ibuf[l]-swx][jbuf[l]-swy]);
799 :           }
800 :   }
801 : }
802 :
803 :
804 : short doublesyn(struct imgdata *a, struct imgdata *b, int i, int j)
805 : {
806 :   int ax1,ay1,bx1,by1,ax0,ay0,bx0,by0,ax,ay,bx,by;
807 :   float l[3];
808 :   float h,ab,d;
809 :   float da,db;
810 :   int xs,ys;
811 :   short s;
812 :
813 :   invaftrsf(i,j,&ax1,&ay1,a->m);
814 :   invaftrsf(a->swx,a->swy,&ax0,&ay0,a->m);
815 :   invaftrsf(i,j,&bx1,&by1,b->m);
816 :   invaftrsf(b->swx,b->swy,&bx0,&by0,b->m);
817 :   ax = ax1 - ax0; ay = ay1 - ay0;
818 :   bx = bx1 - bx0; by = by1 - by0;
819 :   da = dpp(i,j,a->gx,a->gy); db = dpp(i,j,b->gx,b->gy);
820 :   vl(a->gx,a->gy,b->gx,b->gy,1);
821 :   h = dpl(i,j,l);
822 :   ab = dpp(a->gx,a->gy,b->gx,b->gy);
823 :   d = 2 * h / ab;
824 :
825 :   if((d > -1) && (d < 1)){
826 :           d = (1 + d) / 2;
827 :           d = maxf(d,1-d);
828 :           if(da >= db)
829 :             s = (short)(d * b->a[bx][by] + (1 - d) * a->a[ax][ay]);
830 :           else
831 :             s = (short)(d * a->a[ax][ay] + (1 - d) * b->a[bx][by]);
832 :           return(s);
833 :   }
834 :   else
835 :     return(0);
836 : }
837 :
838 : short triplesyn(struct imgdata *a, struct imgdata *b, struct imgdata *c,
839 :                               int i, int j)
840 : {
841 :   float lpa[3],lpb[3],lpc[3],lab[3],lbc[3],lca[3];
842 :   int  ax0,ay0,ax1,ay1,bx0,by0,bx1,by1,cx0,cy0,cx1,cy1;
843 :   int ax,ay,bx,by,cx,cy;
844 :   int dx,dy,ex,ey,fx,fy;
845 :   float d,e,f;
846 :   float h;
847 :   short s;
848 :   int xs,ys;
849 :
850 :   invaftrsf(i,j,&ax1,&ay1,a->m);
851 :   invaftrsf(a->swx,a->swy,&ax0,&ay0,a->m);
852 :   invaftrsf(i,j,&bx1,&by1,b->m);
853 :   invaftrsf(b->swx,b->swy,&bx0,&by0,b->m);
854 :   invaftrsf(i,j,&cx1,&cy1,c->m);
855 :   invaftrsf(c->swx,c->swy,&cx0,&cy0,c->m);
```



```
856 :   ax = ax1 - ax0; ay = ay1 - ay0;
857 :   bx = bx1 - bx0; by = by1 - by0;
858 :   cx = cx1 - cx0; cy = cy1 - cy0;
859 :
860 :   el(a->gx,a->gy,i,j,lpa);
861 :   el(b->gx,b->gy,i,j,lpb);
862 :   el(c->gx,c->gy,i,j,lpc);
863 :   el(a->gx,a->gy,b->gx,b->gy,lab);
864 :   el(b->gx,b->gy,c->gx,c->gy,lbc);
865 :   el(c->gx,c->gy,a->gx,a->gy,lca);
866 :
867 :   cp(lpa,lbc,&dx,&dy);
868 :   cp(lpb,lca,&ex,&ey);
869 :   cp(lpc,lab,&fx,&fy);
870 :
871 :   d = dpp(i,j,dx,dy);
872 :   e = dpp(i,j,ex,ey);
873 :   f = dpp(i,j,fx,fy);
874 :   h = d + e + f;
875 :
876 :   s = (short)((d * a->a[ax][ay] + e * b->a[bx][by] + f * c->a[cx][cy]) / h);
877 :   return(s);
878 : }
879 :
880 : int max(int a, int b)
881 : {
882 :   if(a>=b)
883 :     return(a);
884 :   else
885 :     return(b);
886 : }
887 :
888 : float maxf(float a, float b)
889 : {
890 :   if(a>=b)
891 :     return(a);
892 :   else
893 :     return(b);
894 : }
895 :
896 : int min(int a, int b)
897 : {
898 :   if(a<=b)
899 :     return(a);
900 :   else
901 :     return(b);
902 : }
903 :
904 : float minf(float a, float b)
905 : {
906 :   if(a<=b)
907 :     return(a);
908 :   else
909 :     return(b);
910 : }
911 :
912 : float absf(float a)
913 : {
914 :   if(a >= 0)
915 :     return(a);
916 :   else
917 :     return(-a);
918 : }
919 :
920 :
921 :
```

```
/*
 *       sqimgsynp.c -
 *
 *       2枚の画像の間の非線型投影変換パラメタをLevenberg-Marquardt Method
 *       によって計算し、一方の画像を計算されたパラメタで変換して、連続的に
 *       合成していくプログラム
 *
 *       西羅 光  -1996
 */
#include <stdio.h>
#include <math.h>
#include "image.h"

#define SQR(a) ((a)*(a))

#define GX 640    /*画面の重心*/
#define GY 480

#define EPSIRON 0.0001
#define SR 10

struct imgdata{
  char name[20];  /*ファイル名*/
  short a[1280][960];  /*画素を格納*/
  int xs;       /*画像のサイズ*/
  int ys;       /*画像のサイズ*/
  int swx;      /*画像の変換後の左下角のx座標*/
  int swy;      /*画像の変換後の左下角のy座標*/
  int gx;       /*画像の重心のx座標*/
  int gy;       /*画像の重心のx座標*/
  float m[8];   /*投影変換パラメタ*/
  struct imgdata *p;  /*変換時に基準にとった画像*/
};

void imgtoarray(char *iname, short a[][960], int *xs ,int *ys);
void arraytoimg(short a[][960], char *oname, int xs, int ys);

float diff(int z, short a[][960], int x, int y);  /*微分演算*/
void sol(float a[8][8], float b[8], float x[8]);  /*連立一次方程式の解*/
void inv(float a[8][8], float b[8][8]);  /*逆行列を求める*/

void prtrsf(int x0, int y0, int *x, int *y, float m[8]); /*投影変換*/
void prtrsf2(int x0, int y0, float *x, float *y, float m[8]); /*投影変換*/
void invprtrsf(int x, int y, int *x0, int *y0, float m[8]); /*投影逆変換*/
void prtrsfsyn(float l[8], float m[8], float n[8]);  /*合成変換のパラメタ*/

float err(short a[][960], short b[][960], int x0, int xe, int xs,
          int y0, int ye, int ys, float m[8]); /*二乗誤差関数*/
int grad(short b[][960], float m[8], int x0, int y0, float g[8]);
void setbeta(float beta[8], short a[][960], short b[][960],
             int x0, int xe, int xs, int y0, int ye, int ys, float m[8]);
void setalpha(float alpha[8][8], short a[][960],short b[][960],int x0, int xe,
              int xs, int y0, int ye, int ys, float m[8], float lamda);

void mrqminp(char *name1, char *name2, float m[8]);  /*パラメタを推定*/

float dpp(float x1, float y1, float x2, float y2);  /*点と点の距離*/
void el(int x1, int y1, int x2, int y2, float l[3]);
                                                    /*2点を結ぶ直線*/
void vl(int x1, int y1, int x2, int y2, float l[3]);
                                                    /*垂直二等分線*/
float dpl(int x, int y, float l[3]); /*点と直線の距離*/
int cp(float l1[3], float l2[3], int *x, int *y);   /*2直線の交点*/

int intriangle(struct imgdata *a, struct imgdata *b, struct imgdata *c,
               int i, int j);        /*三角形の内部にあるかどうかの判定*/
```



```

void getmaxmin(int gx, int gy, int xs, int ys,
               int *mx, int *may, int *mix, int *miy, float m[8]);
                                                    /*走査範囲の決定*/
void intpl(short a[][960], int swx, int swy, int maxx, int maxy,
           int minx, int miny, short c[][960], float m[8]);
                                                    /*変換画像の内挿*/
short doublesyn(struct imgdata *a, struct imgdata *b, int i, int j);
                                                    /*画素の二重合成*/
short triplesyn(struct imgdata *a, struct imgdata *b, struct imgdata *c,
                int i, int j);                      /*画素の三重合成*
/

int max(int a, int b);
float maxf(float a, float b);
int min(int a, int b);
float minf(float a, float b);
float absf(float a);

main()
{
  short c[1280][960];  /*出力画像の格納*/
  short d[1280][960];  /*作業用配列*/
  short s,s1;
  float m[8],m1[8],m2[8],ms[8]; /*投影変換パラメタ*/
  int xsize,ysize;  /*画像のサイズ*/
  int i,j,k,l,x,y,i0,j0,x0,y0,x1,y1;
  int maxx,maxy,minx,miny;  /*全体の大きさ*/
  float da,db,ab,h;
  float dv,ds,dt;
  struct imgdata *img;
  struct imgdata *sp,*wp;
  struct imgdata *sa,*sb,*sc;
  struct imgdata *gv,*gs,*gt;
  int n,num;
  char name[20];
  int bufa,bufb,bufc,bufd;   /*作業用変数*/

  printf("Sequential Image Synthesize.\n");
  while(1){
          printf("How many images will you synthesize?\n");
          scanf("%d",&i);
          (void)getchar();
          img = (struct imgdata *)malloc(sizeof(struct imgdata) * i);
          if(img == NULL)
            printf("can't get memory!\n");
          else
            break;
  }
  n = 0;
  sp = img;
  /*1枚目の画像の入力*/
  printf("input filename of first image.\n");
  gets(name);
  imgtoarray(name,sp->a,&xsize,&ysize);
  /*構造体にデータを入力*/
  strcpy(sp->name,name);
  sp->xs = xsize; sp->ys = ysize;
  sp->swx = GX - xsize / 2; sp->swy = GY - ysize / 2;
  sp->gx = GX; sp->gy = GY;
  sp->m[0] = 1; sp->m[1] = 0; sp->m[2] = 0;
  sp->m[3] = 0; sp->m[4] = 1; sp->m[5] = 0;
  sp->m[6] = 0; sp->m[7] = 0;
  sp->p = sp;
  /*全体のサイズの初期化*/
  minx = sp->swx; miny = sp->swy;
```

```
132:    maxx = sp->swx + sp->xs; maxy = sp->swy + sp->ys;
133:    /*出力用配列への格納*/
134:    for(i=minx;i<maxx;i++){
135:            for(j=miny;j<maxy;j++){
136:                    c[i][j] = sp->a[i-sp->swx][j-sp->swy];
137:            }
138:    }
139:    n++;
140:    sp++;
141:    /*2枚目の画像の入力*/
142:    printf("input filename of second image.\n");
143:    gets(name);
144:    imgtoarray(name,sp->a,&xsize,&ysize);
145:    /*構造体にデータを入力*/
146:    strcpy(sp->name,name);
147:    sp->xs = xsize; sp->ys = ysize;
148:    mrqminp(sp->name,img->name,m);  /*パラメタの推定*/
149:    prtrsf(img->swx,img->swy,&bufa,&bufb,m);
150:    sp->swx = bufa; sp->swy = bufb;
151:    prtrsf(img->gx,img->gy,&bufa,&bufb,m);
152:    sp->gx = bufa; sp->gy = bufb;
153:    for(i=0;i<=7;i++)
154:      sp->m[i] = m[i];
155:    sp->p = sp - 1;
156:    /*2枚目の画像の貼付け*/
157:    getmaxmin(GX,GY,sp->xs,sp->ys,&bufa,&bufb,&bufc,&bufd,m);
158:    intp1(sp->a,img->swx,img->swy,bufa,bufb,bufc,bufd,c,m);
159:    /*全体のサイズの更新*/
160:    maxx = max(maxx,bufa); maxy = max(maxy,bufb);
161:    minx = min(minx,bufc); miny = min(miny,bufd);
162:    /*領域判別ループ*/
163:    for(i=minx;i<maxx;i++){
164:            for(j=miny;j<maxy;j++){
165:                    invprtrsf(i,j,&x,&y,m);
166:                    /*上書きされた領域の修復*/
167:                    db = dpp(i,j,sp->gx,sp->gy); da = dpp(i,j,img->gx,img->gy);
168:                    /*indexの変換式*/
169:                    i0 = i - img->swx; j0 = j - img->swy;
170:                    x0 = x - img->swx; y0 = y - img->swy;
171:                    if((i0>=0)&&(i0<xsize)&&(j0>=0)&&(j0<ysize)){
172:                            if(da <= db)
173:                              c[i][j] = img->a[i0][j0];
174:                            else if((x0<0)||(x0>=xsize)||(y0<0)||(y0>=ysize))
175:                              c[i][j] = img->a[i0][j0];
176:                    }
177:                    /*範囲外の領域の処理*/
178:                    if((i0<0)||(i0>=xsize)||(j0<0)||(j0>=ysize)
179:                       ||(x0<0)||(x0>=xsize)||(y0<0)||(y0>=ysize))
180:                      continue;
181:                    /*合成領域の判別と内挿*/
182:                    sa = img; sb = sp;
183:                    s = doublesyn(sa,sb,i,j);
184:                    if(s != NULL)
185:                      c[i][j] = s;
186:            }
187:    }
188:    /*連続貼付けループ*/
189:    while(1){
190:            n++;
191:            sp++;
192:            printf("input base image number.\n");
193:            for(i=0;i<n;i++)
194:              printf("%d. %s\n",i,img[i].name);
195:            printf("%d. quit\n",i);
196:            scanf("%d",&num);
197:            (void)getchar();
```



```
198:            if(num >= n)
199:              break;
200:            printf("input filename of the new image.\n");
201:            gets(name);
202:            imgtoarray(name,sp->a,&xsize,&ysize);
203:            /*構造体にデータを入力*/
204:            strcpy(sp->name,name);
205:            sp->xs = xsize; sp->ys = ysize;
206:            sp->p = img + num;
207:            mrqminp(sp->name,(img+num)->name,m);     /*パラメタの推定*/
208:            /*合成変換パラメタの決定*/
209:            wp = sp;
210:            for(i=0;i<=7;i++)
211:              ms[i] = m[i];
212:            do{
213:                    for(i=0;i<=7;i++)
214:                      m1[i] = ms[i];
215:                    wp = wp->p;
216:                    for(i=0;i<=7;i++)
217:                      m2[i] = wp->m[i];
218:                    prtrsfsyn(m1,m2,ms);
219:            }while(wp != img);
220:            prtrsf(img->swx,img->swy,&bufa,&bufb,ms);
221:            sp->swx = bufa; sp->swy = bufb;
222:            prtrsf(img->gx,img->gy,&bufa,&bufb,ms);
223:            sp->gx = bufa; sp->gy = bufb;
224:            for(i=0;i<=7;i++)
225:              sp->m[i] = m[i];
226:            /*新しい画像の（作業用配列への）貼付け*/
227:            getmaxmin(img->gx,img->gy,sp->xs,sp->ys,
228:                                      &bufa,&bufb,&bufc,&bufd,ms);
229:            intp1(sp->a,img->swx,img->swy,bufa,bufb,bufc,bufd,d,ms);
230:            /*全体のサイズの更新*/
231:            maxx = max(maxx,bufa); maxy = max(maxy,bufb);
232:            minx = min(minx,bufc); miny = min(miny,bufd);
233:            /*自地への貼付け*/
234:            for(i=bufc;i<bufa;i++){
235:                    for(j=bufd;j<bufb;j++){
236:                            if((c[i][j] == 0) && (d[i][j] != 0))
237:                              c[i][j] = d[i][j];
238:                    }
239:            }
240:            /*領域判別ループ*/
241:            for(i=img->gx-sp->xs/2;i<img->gx-sp->xs/2+sp->xs;i++){
242:                    for(j=img->gy-sp->ys/2;j<img->gy-sp->ys/2+sp->ys;j++){
243:                            prtrsf(i,j,&x,&y,ms);
244:
245:                            dv = 5000; ds = 5000; dt = 5000;
246:                            gs = img; gt = img;
247:                            for(k=0;k<=n;k++){
248:                                    l = 1;
249:                                    if(dpp(x,y,img[k].gx,img[k].gy) <= dv){
250:                                            gt = gs; gs = gv; gv = img + k;
251:                                            dt = ds; ds = dv;
252:                                            dv = dpp(x,y,img[k].gx,img[k].gy);
253:                                            l = 3;
254:                                    }
255:                                    else if(dpp(x,y,img[k].gx,img[k].gy) <= ds){
256:                                            gt = gs; gs = img + k;
257:                                            dt = ds;
258:                                            ds = dpp(x,y,img[k].gx,img[k].gy);
259:                                            l = 2;
260:                                    }
261:                                    else if(dpp(x,y,img[k].gx,img[k].gy) <= dt){
262:                                            gt = img + k;
263:                                            dt = dpp(x,y,img[k].gx,img[k].gy);
```

```
264:                                    }
265:                                    else
266:                                      l = 0;
267:                            }
268:                            if(l == 0)
269:                              continue;
270:                            else if(l == 3){
271:                                    s = doublesyn(gv,gs,x,y);
272:                                    if(!s){
273:                                            if(d[x][y] != 0)
274:                                              c[x][y] = d[x][y];
275:                                            continue;
276:                                    }
277:                                    if(intriangle(gv,gs,gt,x,y))
278:                                      c[x][y] = triplesyn(gv,gs,gt,x,y);
279:                                    else
280:                                      c[x][y] = s;
281:                            }
282:                            else if(l == 2){
283:                                    s = doublesyn(gv,gs,x,y);
284:                                    if(!s)
285:                                      continue;
286:                                    if(intriangle(gv,gs,gt,x,y))
287:                                      c[x][y] = triplesyn(gv,gs,gt,x,y);
288:                                    else
289:                                      c[x][y] = s;
290:                            }
291:                            else{
292:                                    s = doublesyn(gv,gt,x,y);
293:                                    s1 = doublesyn(gs,gt,x,y);
294:                                    if(s*s1){
295:                                            if(intriangle(gv,gs,gt,x,y))
296:                                              c[x][y] = triplesyn(gv,gs,gt,x,y);
297:                                    }
298:                            }
299:                    }
300:            }
301:    }
302:    /*不要部分の削除*/
303:    for(i=minx;i<maxx;i++){
304:            for(j=miny;j<maxy;j++){
305:                    c[i-minx][j-miny] = c[i][j];
306:            }
307:    }
308:    printf("input filename of output.\n");
309:    gets(name);
310:    arraytoimg(c,name,maxx-minx+1,maxy-miny+1);
311: }
312:
313: void imgtoarray(char *iname , short a[][960] ,int *xs ,int *ys)
314: {
315:      IMAGE *iimage;
316:      unsigned int xsize, ysize;
317:      unsigned int x,y;
318:    short   ibuf[960];
319:
320:      if( (iimage=iopen(iname,"r")) == NULL ) {
321:    fprintf(stderr,"imgtoarray: can't open input file %s\n",*iname);
322:    exit(1);
323:      }
324:      xsize = iimage->xsize;
325:      ysize = iimage->ysize;
326:
327:    *xs = xsize; *ys = ysize;
328:
329:      for(y=0; y<ysize; y++) {
```



```
330:        getrow(iimage,ibuf,y,0);
331:            for(x=0; x<xsize; x++){
332:                    a[x][y] = ibuf[x];
333:            }
334:    }
335:      iclose(iimage);
336:      return;
337: }
338:
339: void arraytoimg(short a[][960], char *oname, int xsize, int ysize)
340: {
341:    IMAGE *oimage;
342:    unsigned int x,y;
343:    short    obuf[1280];
344:
345:    oimage = iopen(oname,"w",RLE(1),2,xsize,ysize);
346:    for(y=0;y<ysize;y++){
347:            for(x=0;x<xsize;x++){
348:                    obuf[x] = a[x][y];
349:            }
350:            putrow(oimage,obuf,y,0);
351:    }
352:    iclose(oimage);
353:    return;
354: }
355:
356: float diff(int z, short a[][960], int x, int y)
357: {
358:    float d;
359:
360:    if(z == 1){
361:            if((x+1)>=640)
362:              d = (float)(a[x][y] - a[x-1][y]);
363:            else
364:              d = (float)(a[x+1][y] - a[x][y]);
365:    }
366:    else if(z == 2){
367:            if((y+1)>=480)
368:              d = (float)(a[x][y] - a[x][y-1]);
369:            else
370:              d = (float)(a[x][y+1] - a[x][y]);
371:    }
372:    else{
373:            d = NULL;
374:    }
375:    return(d);
376: }
377:
378: void sol(float a[8][8], float b[8], float x[8])
379: {
380:    int i,j,k;
381:    float c;
382:
383:    /*前進消去*/
384:    for(i=0;i<=7;i++){
385:            x[i] = 0.0;  /*後退代入のための初期化*/
386:            for(j=i;j<=7;j++){
387:                    if(j==i)
388:                            b[j] = b[j] / a[i][i];
389:                    else
390:                            b[j] = b[j] - c * b[i];
391:                    for(k=j+1;k<=7;k++){
392:                            if(j==i)
393:                                    a[j][k] = a[j][k] / a[i][i];
394:                            else{
395:                                    c = a[j][i] / a[i][i];
```

```
396:                                    a[j][k] = a[j][k] - c * a[i][k];
397:                            }
398:                    }
399:            }
400:    }
401:    /*後退代入*/
402:    for(i=7;i>=0;i--){
403:            c = 0.0;
404:            for(j=7;j>i;j--){
405:                    c += a[i][j] * x[j];
406:            }
407:            x[i] = b[i] - c;
408:    }
409: }
410:
411: void inv(float a[8][8], float b[8][8])  /*逆行列を求める*/
412: {
413:    float x[8],c[8];
414:    int i,j,k;
415:
416:    for(i=0;i<=7;i++){
417:            for(j=0;j<=7;j++){
418:                    if(j==i)
419:                      c[j] = 1.0;
420:                    else
421:                      c[j] = 0.0;
422:            }
423:            sol(a,c,x);
424:            for(k=0;k<=7;k++){
425:                    b[k][i] = x[k];
426:            }
427:    }
428: }
429:
430: void prtrsf(int x0, int y0, int *x, int *y, float m[8])  /*投影変換*/
431: {
432:    float x1,y1;
433:
434:    x1 = (m[0] * x0 + m[1] * y0 + m[2]) / (m[6] * x0 + m[7] * y0 + 1);
435:    y1 = (m[3] * x0 + m[4] * y0 + m[5]) / (m[6] * x0 + m[7] * y0 + 1);
436:    *x = (int)x1;
437:    *y = (int)y1;
438: }
439:
440: void prtrsf2(int x0, int y0, float *x, float *y, float m[8])  /*投影変換*/
441: {
442:    *x = (m[0] * x0 + m[1] * y0 + m[2]) / (m[6] * x0 + m[7] * y0 + 1);
443:    *y = (m[3] * x0 + m[4] * y0 + m[5]) / (m[6] * x0 + m[7] * y0 + 1);
444: }
445:
446: void invprtrsf(int x, int y, int *x0, int *y0, float m[8]) /*投影逆変換*/
447: {
448:    float x1,y1;
449:    float det;
450:    float n[9];
451:
452:    n[0] = m[4] - m[5] * m[7];
453:    n[1] = m[2] * m[7] - m[1];
454:    n[2] = m[1] * m[5] - m[2] * m[4];
455:    n[3] = m[5] * m[6] - m[3];
456:    n[4] = m[0] - m[2] * m[6];
457:    n[5] = m[2] * m[3] - m[0] * m[5];
458:    n[6] = m[3] * m[7] - m[4] * m[6];
459:    n[7] = m[1] * m[6] - m[0] * m[7];
460:    n[8] = m[0] * m[4] - m[1] * m[3];
461:
```



```
462:    x1 = (n[0] * x + n[1] * y + n[2]) / (n[6] * x + n[7] * y + n[8]);
463:    y1 = (n[3] * x + n[4] * y + n[5]) / (n[6] * x + n[7] * y + n[8]);
464:    *x0 = (int)x1;
465:    *y0 = (int)y1;
466: }
467:
468: void prtrsfsyn(float l[8], float m[8], float n[8])
469: {
470:    float det;
471:
472:    det = l[6] * m[2] + l[7] * m[5] + 1;
473:    n[0] = (l[0] * m[0] + l[1] * m[3] + l[2] * m[6]) / det;
474:    n[1] = (l[0] * m[1] + l[1] * m[4] + l[2] * m[7]) / det;
475:    n[2] = (l[0] * m[2] + l[1] * m[5] + l[2] ) / det;
476:    n[3] = (l[3] * m[0] + l[4] * m[3] + l[5] * m[6]) / det;
477:    n[4] = (l[3] * m[1] + l[4] * m[4] + l[5] * m[7]) / det;
478:    n[5] = (l[3] * m[2] + l[4] * m[5] + l[5]) / det;
479:    n[6] = (l[6] * m[0] + l[7] * m[3] + m[6]) / det;
480:    n[7] = (l[6] * m[1] + l[7] * m[4] + m[7]) / det;
481: }
482:
483: float err(short a[][960], short b[][960], int x0, int xe, int xs,
484:              int y0, int ye, int ys, float m[8]) /*二乗誤差関数*/
485: {
486:    int i,j;
487:    int x,y;
488:    int nx,ny;
489:    int s; /*画素数*/
490:    float e;
491:
492:    e = 0.0;
493:    nx = (xe - x0) / SR; ny = (ye - y0) / SR;
494:    s = 0;
495:
496:    for(i=x0+nx;i<xe;i+=nx){
497:            for(j=y0+ny;j<ye;j+=ny){
498:                    prtrsf(i,j,&x,&y,m);
499:                    if((x<0) || (y<0) || (x>=xs) || (y>=ys))
500:                      continue;
501:                    else{
502:                            e += SQR(b[x][y] - a[i][j]);
503:                            s++;
504:                    }
505:            }
506:    }
507:    return(e/s);
508: }
509:
510: int grad(short b[][960], float m[8], int x0, int y0, float g[8])
511: {
512:    int t[8];
513:    int x,y;
514:    int i;
515:    float d;
516:
517:    d = m[6] * x0 + m[7] * y0 + 1;
518:
519:    t[0] = x0; t[3] = x0;
520:    t[1] = y0; t[4] = y0;
521:    t[2] =  1; t[5] =  1;
522:    t[6] = -(m[0] * x0 + m[1] * y0 + m[2])  / SQR(d);
523:    t[7] = -(m[3] * x0 + m[4] * y0 + m[5])  / SQR(d);
524:
525:    prtrsf(x0,y0,&x,&y,m);
526:    if((x<0) || (y<0) || (x>=640) || (y>=480))
527:      return(1);
```

```
528:
529:    for(i=0;i<=2;i++)
530:      g[i] = diff(1,b,x,y) * t[i];
531:    for(i=3;i<=5;i++)
532:      g[i] = diff(2,b,x,y) * t[i];
533:    g[6] = (diff(1,b,x,y) * t[6] + diff(2,b,x,y) * t[7]) * x0;
534:      g[7] = (diff(1,b,x,y) * t[6] + diff(2,b,x,y) * t[7]) * y0;
535:
536:    return(0);
537: }
538:
539: void setbeta(float beta[8], short a[][960], short b[][960],
540:                      int x0, int xe,int xs, int y0, int ys, int ye, float m[8])
541: {
542:    int i,j,k;
543:    int x,y;
544:    int nx,ny;
545:    float g[8];
546:
547:    nx = (xe - x0) / SR; ny = (ye - y0) / SR;
548:    for(k=0;k<=8;k++){
549:            beta[k] = 0.0;
550:            for(i=x0+nx;i<xe;i+=nx){
551:                    for(j=y0+ny;j<ye;j+=ny){
552:                            prtrsf(i,j,&x,&y,m);
553:                            if((x<0) || (y<0) || (x>=xs) || (y>=ys))
554:                              continue;
555:                            else if(grad(b,m,i,j,g))
556:                              continue;
557:                            else
558:                              beta[k] += (b[x][y] - a[i][j]) * g[k];
559:                    }
560:            }
561:            beta[k] = -1 * beta[k];
562:    }
563:    return;
564: }
565:
566: void setalpha(float alpha[8][8], short a[][960], short b[][960], int x0,
567:                      int xe, int xs, int y0, int ye, int ys,  float m[8], float
lamda)
568: {
569:    int i,j,k,l;
570:    int x,y;
571:    int nx,ny;
572:    float g[8];
573:
574:    nx = (xe - x0) / SR; ny = (ye - y0) / SR;
575:    for(k=0;k<=7;k++){
576:            for(l=k;l<=7;l++){
577:                    alpha[k][l] = 0.0;
578:                    for(i=x0+nx;i<xe;i+=nx){
579:                            for(j=y0+ny;j<ye;j+=ny){
580:                                    prtrsf(i,j,&x,&y,m);
581:                                    if((x<0) || (y<0) || (x>=xs) || (y>=ys))
582:                                      continue;
583:                                    else if(grad(b,m,i,j,g))
584:                                      continue;
585:                                    else
586:                                      alpha[k][l] += g[k] * g[l];
587:                            }
588:                    }
589:                    if(k==l)
590:                      alpha[k][l] *= 1 + lamda;
591:                    else
592:                      alpha[l][k] = alpha[k][l];
```



```
593:            }
594:    }
595:    return;
596: }
597:
598: void mrqminp(char *name1, char *name2, float m[8])
599: {
600:    short a[1280][960],b[1280][960];
601:    float beta[8];      /*誤差関数の勾配ベクトル*/
602:    float alpha[8][8]; /*ヘッセ行列*/
603:    float cov[8][8];    /*推定値の誤差共分散行列*/
604:    float lamda;        /*安定化因子*/
605:    float dm[8];        /*mの修正値*/
606:    int x0,y0,xe,ye;  /*代表点に関する情報*/
607:    float e,eo,en;    /*二乗誤差関数の値*/
608:    int xsize,ysize;
609:    int i,j;
610:    int r;    /*反復回数をカウント*/
611:    int xdmy,ydmy;
612:
613:    imgtoarray(name1,a,&xsize,&ysize);
614:    imgtoarray(name2,b,&xsize,&ysize);
615:
616:    /*初期値（平行移動成分）の入力*/
617:    printf("input estimate of x-factor of vector %s to %s\n",name1,name2);
618:    scanf("%d",&xdmy);
619:    printf("input estimate of y-factor of vector %s to %s\n",name1,name2);
620:    scanf("%d",&ydmy);
621:    m[2] = (float)xdmy; m[5] = (float)ydmy;
622:    m[0] = 1.0; m[1] = 0.0; m[3] = 0.0; m[4] = 1.0;
623:    m[6] = 0.0; m[7] = 0.0;
624:
625:    /*代表点の決定*/
626:    x0 = max(0,-xdmy); xe = min(xsize,xsize-xdmy);
627:    y0 = max(0,-ydmy); ye = min(ysize,ysize-ydmy);
628:
629:    /*初期化*/
630:    e = err(a,b,x0,xe,xsize,y0,ye,ysize,m);
631:    lamda = 0.001;
632:    printf("e = %f\n",e);
633:    r = 0;
634:
635:    /*ループ*/
636:    do{
637:            r++;
638:            setbeta(beta,a,b,x0,xe,xsize,y0,ye,ysize,m);          /*ベータの設定*/
639:            setalpha(alpha,a,b,x0,xe,xsize,y0,ye,ysize,m,lamda); /*アルファの設定
*/
640:
641:            sol(alpha,beta,dm);  /*連立方程式の解*/
642:
643:            for(i=0;i<=7;i++)
644:                    m[i] += dm[i];
645:            en = err(a,b,x0,xe,xsize,y0,ye,ysize,m);
646:            eo = e;
647:            if(en > e){    /*誤差関数の評価*/
648:                    lamda *= 10.0;
649:                    for(i=0;i<=7;i++)
650:                      m[i] -= dm[i];
651:            }
652:            else{
653:                    lamda *= 0.1;
654:                    e = en;
655:                    printf("e = %f\n",e);
656:            }
657:    }while((en > eo)||((eo - en) / eo > EPSIRON));      /*継続条件*/
```

```
658:
659:    printf("%d times  iteration.\n",r);
660:    /*推定値の表示*/
661:    printf("calculated estimates are below.\n");
662:    for(i=0;i<=7;i++)
663:            printf("m[%d] = %f\n",i,m[i]);
664:    /*誤差共分散行列の表示*/
665:    lamda = 0.0;
666:    setalpha(alpha,a,b,x0,xe,xsize,y0,ye,ysize,m,lamda);
667:    inv(alpha,cov);
668:    printf("covariance matrix of the standard errors\n");
669:    printf("in the fitted parameters m are below.\n");
670:    for(i=0;i<=7;i++){
671:            for(j=0;j<=7;j++){
672:                    printf("%.4f ",cov[i][j]);
673:            }
674:            printf("\n");
675:    }
676:    return;
677: }
678:
679: float dpp(float x1, float y1, float x2, float y2)  /*点と点の距離*/
680: {
681:    float d;
682:
683:    d = SQR(x1 - x2) + SQR(y1 - y2);
684:    return(sqrt(d));
685: }
686:
687: void el(int x1, int y1, int x2, int y2, float l[3])
688: {
689:    l[0] = (float)(y1 - y2);
690:    l[1] = (float)(x2 - x1);
691:    l[2] = (float)((x1 - x2) * y1 - x1 * (y1 - y2));
692: }
693:
694: void vl(int x1, int y1, int x2, int y2, float l[3])
695: {
696:    l[0] = (float)(x2 - x1);
697:    l[1] = (float)(y2 - y1);
698:    l[2] = -(float)((SQR(y2) - SQR(y1) + SQR(x2) - SQR(x1))) / 2;
699: }
700:
701: float dpl(int x, int y, float l[3])
702: {
703:    float d;
704:
705:    d = (l[0] * x + l[1] * y + l[2]) / sqrt(SQR(l[0]) + SQR(l[1]));
706:    return(d);
707: }
708:
709: int cp(float l1[3], float l2[3], int *x, int *y)
710: {
711:    float det;
712:
713:    det = l1[0] * l2[1] - l2[0] * l1[1];
714:    if(det == 0.0)
715:      return(0);
716:    *x = (int)(- (l2[1] * l1[2] - l1[1] * l2[2]) / det);
717:    *y = (int)(- (l1[0] * l2[2] - l2[0] * l1[2]) / det);
718:    return(1);
719: }
720:
721: int intriangle(struct imgdata *a, struct imgdata *b, struct imgdata *c,
722:                        int i, int j)
723: {
```



```
724:    float tgx,tgy;  /*三角形の重心の座標*/
725:    float lab[3],lbc[3],lca[3];
726:    float e,f;
727:    float maxx,maxy,minx,miny;
728:    float ir,jr;
729:
730:    tgx = (a->gx + b->gx + c->gx) / 3.0;
731:    tgy = (a->gy + b->gy + c->gy) / 3.0;
732:    el(a->gx,a->gy,b->gx,b->gy,lab);
733:    el(b->gx,b->gy,c->gx,c->gy,lbc);
734:    el(c->gx,c->gy,a->gx,a->gy,lca);
735:
736:    e = dpl(tgx,tgy,lab) * dpl(tgx,tgy,lbc) * dpl(tgx,tgy,lca);
737:    f = dpl(i,j,lab) * dpl(i,j,lbc) * dpl(i,j,lca);
738:    if((e * f) <= 0)
739:      return(0);   /*三角形の内部にない*/
740:    maxx = maxf(a->gx,maxf(b->gx,c->gx));
741:    maxy = maxf(a->gy,maxf(b->gy,c->gy));
742:    minx = minf(a->gx,minf(b->gx,c->gx));
743:    miny = minf(a->gy,minf(b->gy,c->gy));
744:    ir = (float)i; jr = (float)j;
745:    if((ir>=maxx)||(ir<=miny)||(jr>=maxy)||(jr<=miny))
746:      return(0);  /*三角形の内部にない*/
747:    else
748:      return(1);  /*三角形の内部にある*/
749: }
750:
751: void getmaxmin(int gx, int gy, int xs, int ys,
752:                        int *mx, int *may, int *mix, int *miy, float m[8])
753: {
754:    int i,j;
755:    int x0,y0;
756:    int maxx,maxy,minx,miny;
757:
758:    maxx = 0; maxy = 0; minx = 1280; miny = 960;
759:    /*エッジに沿って走査*/
760:    for(i=gx-xs/2;i<gx-xs/2+xs;i++){
761:            prtrsf(i,gy-ys/2,&x0,&y0,m);
762:            maxx = max(x0,maxx); maxy = max(y0,maxy);
763:            minx = min(x0,minx); miny = min(y0,miny);
764:    }
765:    for(j=gy-ys/2;j<gy-ys/2+ys;j++){
766:            prtrsf(gx-xs/2+xs-1,j,&x0,&y0,m);
767:            maxx = max(x0,maxx); maxy = max(y0,maxy);
768:            minx = min(x0,minx); miny = min(y0,miny);
769:    }
770:    for(i=gx-xs/2+xs-1;i>=gx-xs/2;i--){
771:            prtrsf(i,gy-ys/2+ys-1,&x0,&y0,m);
772:            maxx = max(x0,maxx); maxy = max(y0,maxy);
773:            minx = min(x0,minx); miny = min(y0,miny);
774:    }
775:    for(j=gy-ys/2+ys-1;j>=gy-ys/2;j--){
776:            prtrsf(gx-xs/2,j,&x0,&y0,m);
777:            maxx = max(x0,maxx); maxy = max(y0,maxy);
778:            minx = min(x0,minx); miny = min(y0,miny);
779:    }
780:    maxx = min(1280,maxx); maxy = min(960,maxy);
781:    minx = max(0,minx); miny = max(0,miny);
782:
783:    *mx = maxx; *may = maxy; *mix = minx; *miy = miny;
784: }
785:
786:
787: void intpl(short a[][960], int swx, int swy, int maxx, int maxy,
788:                int minx, int miny, short c[][960], float m[8])
789: {
```

```
790:   int x,y;
791:   int x0,y0;
792:   int i,j,k,l;
793:   float xf,yf;
794:   float xbuf[10],ybuf[10],d[10];
795:   int ibuf[10],jbuf[10];
796:   float s;
797:   float d0;
798:
799:   for(x=minx;x<maxx;x++){
800:           for(y=miny;y<maxy;y++){
801:                   k = 0;
802:                   invprtrsf(x,y,&x0,&y0,m);
803:                   for(i=(x0-1);i<=(x0+1);i++){
804:                           for(j=(y0-1);j<=(y0+1);j++){
805:                                   prtrsf2(i,j,&xf,&yf,m);
806:                                   if(dpp(x,y,xf,yf) < 1.0){
807:                                           xbuf[k] = xf; ybuf[k] = yf;
808:                                           ibuf[k] = i;  jbuf[k] = j;
809:                                           k++;
810:                                   }
811:                           }
812:                   }
813:                   s = 0.0;
814:                   for(l=0;l<k;l++){
815:                           d[l] = 1 - dpp(x,y,xbuf[l],ybuf[l]);
816:                           s += d[l];
817:                   }
818:                   c[x][y] = 0;
819:                   for(l=0;l<k;l++)
820:                     c[x][y] += (short)(d[l] / s * a[ibuf[l]-swx][jbuf[l]-swy]);
821:           }
822:   }
823: }
824:
825:
826: short doublesyn(struct imgdata *a, struct imgdata *b, int i, int j)
827: {
828:   int ax1,ay1,bx1,by1,ax0,ay0,bx0,by0,ax,ay,bx,by;
829:   float l[3];
830:   float h,ab,d;
831:   float da,db;
832:   int xs,ys;
833:   short s;
834:
835:   invprtrsf(i,j,&ax1,&ay1,a->m);
836:   invprtrsf(a->swx,a->swy,&ax0,&ay0,a->m);
837:   invprtrsf(i,j,&bx1,&by1,b->m);
838:   invprtrsf(b->swx,b->swy,&bx0,&by0,b->m);
839:   ax = ax1 - ax0; ay = ay1 - ay0;
840:   bx = bx1 - bx0; by = by1 - by0;
841:   da = dpp(i,j,a->gx,a->gy); db = dpp(i,j,b->gx,b->gy);
842:   vl(a->gx,a->gy,b->gx,b->gy,l);
843:   h = dpl(i,j,l);
844:   ab = dpp(a->gx,a->gy,b->gx,b->gy);
845:   d = 2 * h / ab;
846:
847:   if((d > -1) && (d < 1)){
848:           d = (1 + d) / 2;
849:           d = maxf(d,1-d);
850:           if(da >= db)
851:             s = (short)(d * b->a[bx][by] + (1 - d) * a->a[ax][ay]);
852:           else
853:             s = (short)(d * a->a[ax][ay] + (1 - d) * b->a[bx][by]);
854:           return(s);
855:   }
```



```
856:   else
857:     return(0);
858: }
859:
860: short triplesyn(struct imgdata *a, struct imgdata *b, struct imgdata *c,
861:                                 int i, int j)
862: {
863:   float lpa[3],lpb[3],lpc[3],lab[3],lbc[3],lca[3];
864:   int  ax0,ay0,ax1,ay1,bx0,by0,bx1,by1,cx0,cy0,cx1,cy1;
865:   int ax,ay,bx,by,cx,cy;
866:   int dx,dy,ex,ey,fx,fy;
867:   float d,e,f;
868:   float h;
869:   short s;
870:   int xs,ys;
871:
872:   invprtrsf(i,j,&ax1,&ay1,a->m);
873:   invprtrsf(a->swx,a->swy,&ax0,&ay0,a->m);
874:   invprtrsf(i,j,&bx1,&by1,b->m);
875:   invprtrsf(b->swx,b->swy,&bx0,&by0,b->m);
876:   invprtrsf(i,j,&cx1,&cy1,c->m);
877:   invprtrsf(c->swx,c->swy,&cx0,&cy0,c->m);
878:   ax = ax1 - ax0; ay = ay1 - ay0;
879:   bx = bx1 - bx0; by = by1 - by0;
880:   cx = cx1 - cx0; cy = cy1 - cy0;
881:
882:   el(a->gx,a->gy,i,j,lpa);
883:   el(b->gx,b->gy,i,j,lpb);
884:   el(c->gx,c->gy,i,j,lpc);
885:   el(a->gx,a->gy,b->gx,b->gy,lab);
886:   el(b->gx,b->gy,c->gx,c->gy,lbc);
887:   el(c->gx,c->gy,a->gx,a->gy,lca);
888:
889:   cp(lpa,lbc,&dx,&dy);
890:   cp(lpb,lca,&ex,&ey);
891:   cp(lpc,lab,&fx,&fy);
892:
893:   d = dpp(i,j,dx,dy);
894:   e = dpp(i,j,ex,ey);
895:   f = dpp(i,j,fx,fy);
896:   h = d + e + f;
897:
898:   s = (short)((d * a->a[ax][ay] + e * b->a[bx][by] + f * c->a[cx][cy]) / h);
899:   return(s);
900: }
901:
902: int max(int a, int b)
903: {
904:   if(a>=b)
905:     return(a);
906:   else
907:     return(b);
908: }
909:
910: float maxf(float a, float b)
911: {
912:   if(a>=b)
913:     return(a);
914:   else
915:     return(b);
916: }
917:
918: int min(int a, int b)
919: {
920:   if(a<=b)
921:     return(a);
```

```
922:    else
923:      return(b);
924: }
925:
926: float minf(float a, float b)
927: {
928:   if(a<=b)
929:     return(a);
930:   else
931:     return(b);
932: }
933:
934: float absf(float a)
935: {
936:   if(a >= 0)
937:     return(a);
938:   else
939:     return(-a);
940: }
941:
942:
943:
944:
945:
946:
947:
948:
949:
950:
951:
952:
```