

# 画像処理実験 最終レポート

09430509

今田将也

2020 年 10 月 27 日

本実験では、複数の画像を合成して広い範囲の映ったパノラマ画像を作成し、より正確な合成を行う方法を学ぶ。各時間ごとの課題の考察について述べたあと、最終的なパノラマ画像の合成結果について述べる。

## 1 画像フォーマット / 画像の表示

計算機で画像を扱うには、次のようなことを理解する必要がある。プログラム内部で画像を表現する際のデータ構造。画像ファイル入出力の方法と画像圧縮。画像処理の方法。

### 1.1 課題 配列の数値を書き換え、適当に編集して、画像をレポートに挿入しなさい。

画素が白か黒等の 2 種類の画像を 2 値画像と呼ぶ。画像の大きさのうち、横の画素数を幅 (width)，縦の画素数を高さ (height) と呼ぶ。各画素の色は 1 つの数値で表される。これらの数値を画素値あるいは輝度値 (intensity value) 等と呼ぶ。通常、画素値は 8 ビット符号なし整数 (unsigned char) である。画素値の最小値が黒、最大値が白、中間の値が様々な濃さの灰色を表すような画像を濃淡画像と呼ぶ。

カラー画像の各画素は光の三原色（赤 R, 緑 G, 青 B）の各成分からなり、画素が極めて小さいとき加法混色によって様々な色に見える。カラー画像の配列表現法は複数の種類がある。一つは、各画素に対して R, G, B の輝度を並べる方法である。

画像の編集をした結果を図 1, 図 2, 図 3 に示した。



図 1: 1-1 の画像

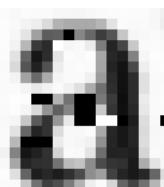


図 2: 1-2 の画像

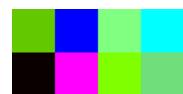


図 3: 1-3 の画像

## 1.2 カラー画像から R, G, B への分解

以下のように、配列 dst の p+0 要素の赤成分のみを代入することで実装した。また、p+1 要素、p+2 要素に src[p+0] を代入することで緑成分のみ、青成分のみを取り出した。

(プログラム一部省略)

```
dst[p+0]=src[p+0];  
dst[p+1]=0;  
dst[p+2]=0;
```



図 4: 赤色の画像



図 5: 緑色の画像



図 6: 青色の画像

## 1.3 画像の拡大

画像の拡大をするために、画像の色を抽出する手順の前に拡大を行う手順を入れた。以下はそのソースコードを一部抜粋した。

```
//拡大  
var xx=Math.round(x/4)+200,yy=Math.round(y/4)+100;  
q=(wdt*yy+xx)*4;  
dst[p+0]=src[q+0];  
dst[p+1]=src[q+1];  
dst[p+2]=src[q+2];  
  
//色の抽出  
p=(y*wdt+x)*4;  
dst[p+0]=r/49;  
dst[p+1]=g/49;  
dst[p+2]=b/49;  
dst[p+3]=255;
```



図 7: 4 倍に拡大した画像

#### 1.4 C 言語で画像処理

講義のページにある image.h,image.c,image2.c,image2.hxx を利用し, easyProcess.c を作成後以下のコマンドにて実行をおこなった.

```
$ gcc --no-warnings -O easyProcess.c image.c image2.c  
$ ./a.out 0.jpg a.jpg
```

生成された画像を図 8 に示した.

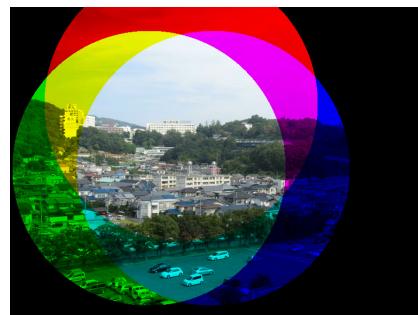


図 8: easyProcess にて生成された画像

##### 1.4.1 赤色のみ抽出

IElem(dst,x,y,0) に IElem(src,x,y,0) で赤要素のみを入れた. 緑と青に関しては, 例のように 0 を入れるつまり無視すれば良いので, コメントアウトすることで実装した. 図 9 に示す

##### 1.4.2 緑色のみ抽出

赤色のときと同様の方法で実装した. 図 10 に示す



図 9: 赤色のみ抽出した画像



図 10: 緑色のみ抽出した画像



図 11: 青色のみ抽出した画像

### 1.4.3 青色のみ抽出

赤色のときと同様の方法で実装した。図 11 に示す

### 1.4.4 画像の拡大

画像の拡大は、IElem の src で指定されている箇所の x と y をそれぞれ変更することで拡大の実装ができる。以下そのコードを一部抜粋したものである

```
int qx = round((x)/2);
int qy = round((y)/2);
IElem(dst,x,y,0) = IElem(src,qx,qy,0)*disk(x,y,320,180)/256; // red at (x,y)
```

また、その結果の画像を図 12 に示す。



図 12: 拡大した画像

### 1.4.5 スマホで撮影した画像での実験

実験元の画像を図 13 に示す。

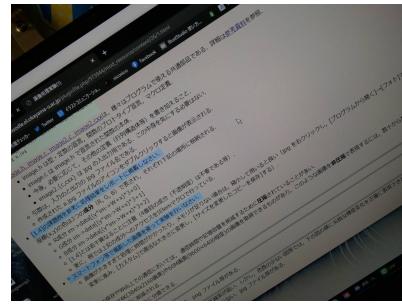


図 13: スマホで撮影した画像

easyProcess で拡大について実験を行った。その結果は図 14 に示した。

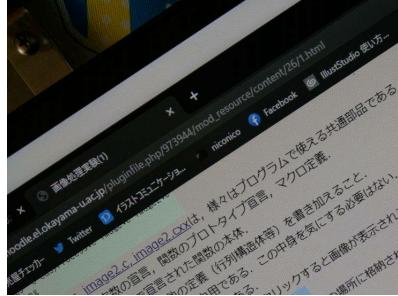


図 14: スマホで撮影した画像を拡大した結果

### 1.5 様々な型式でのファイルサイズ（および画質）を比較

元形式が jpg 形式の画像を用いて, png, gif, bmp, webp についてそれぞれ比較を行った. 画像の縦幅, 横幅に変化はなく, 純粹にファイルの書き出し形式のみ変更を行った.

形式	ファイルサイズ (B)
.jpg(元画像)	7,692,612
.png	29,699,714
.gif	13,502,072
.bmp	36,578,442
.webp	7,192,510

無圧縮である Windows 独自規格の bmp ファイルについて見てみると, やはりファイルサイズは大きくなっている. また, 次いで可逆圧縮形式の png のファイルサイズも大きくなっていた. Web などではかなり使われている形式のため, 小さくなるかと思っていたがそうではなかった. そして非可逆圧縮の gif は元画像の約 2 倍程度にまでファイルサイズが大きくなっている. 最後に, 非可逆圧縮, 可逆圧縮の両方を扱える webp についてだが, ここでは可逆圧縮で実験を行った. 結果, jpg よりも 500KB, 少なくなった.

## 2 画像の幾何学変換

遠景を同一視点から撮影した画像を重ね合わせるには, 射影変換を用いる. 射影変換は  $3 \times 3$  行列で表されるため積によって変換の合成を行うことができる. なお, 基本的な幾何学変換（回転・平行移動・拡大・縮小）は 3 行目が  $(0 \ 0 \ 1)$  の射影変換で表すことができる.

### 2.1 $(x,y)$ と $(u,v)$ の位置に, ほぼ同じ物体が観測されていることを確認

図 15 からほぼ同じ物体が同じ位置に観測されている

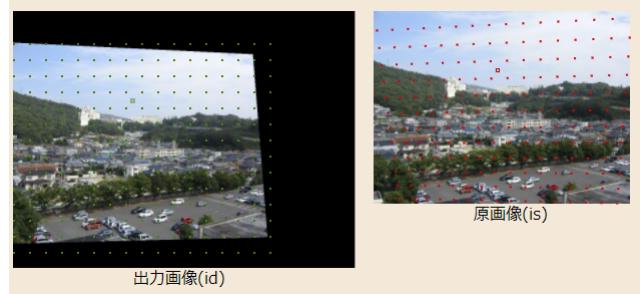


図 15: 出力画像と元画像

## 2.2 射影変換 a の要素を変化させ画像の変化を確認

変更前の配列 a

```
a=[[ .866 , -.5 , 160] ,
[ .5 , .866 , -300] ,
[-.001 , 0 , 1]];
```

変更後の配列 a

```
a=[[ .433 , -.5 , 80] ,
[ .5 , .433 , -150] ,
[-.0001 , 0 , 1]];
```



図 16: 配列変更後の出力画像

## 2.3 C 言語による実装例をもとに 上記と同様の処理を行う

実装にあたり、ImageClear 関数が定義されていなかったため、image.c に ImageClear 関数をまず定義した。参考資料から、画像のすべての要素を黒で塗りつぶせば良いため、画像の画素数分ループ処理を行

い、全ての画素に黒の値である 0 を代入することで実装した。以下にそのソースを示す。また結果は図 17 に示した。

```
void ImageClear(Image *im){  
    for(int y=0;y<im->H;y++){  
        for(int x=0;x<im->W;x++){  
            IElem(im,x,y,0) = 0;  
            IElem(im,x,y,1) = 0;  
            IElem(im,x,y,2) = 0;  
        }  
    }  
}
```



図 17: C 言語による処理結果

## 2.4 m0d を適当な行列に変更しても、img1 と img0 の位置関係が保たれることを確認

まず、図 18 に m0d 行列の変更前の画像を示す。そして、次のように画像の全体の座標位置を右にずらすよう変更を加えたところ、図 19 のような結果が得られた。この結果から、img1 と img0 の位置関係は m0d を適当な行列に変更しても保たれていることが確認できた。

```
var m0d=[[ 2.667,0,-500 ],  
          [ 0,2.667,-100 ],  
          [ 0,0,1 ]];
```

## 2.5 行列 m10 を修正し、ずれのない合成画像を作成することを試みる

いくつかの行列を修正し、試みたがうまくずれをなくすことができなかった。



図 18: m0d 行列変更前



図 19: m0d 行列変更後

## 2.6 m10 を算出し、合成に用いなさい

実験のページに記載されているページの特徴点を動かすことで正しい m10 を算出し、合成に用いた。合成をしたい元画像の両方で特徴点が正確に同じ物体を指すようにするということを行った。以下が算出した配列である。なお、図 20 が合成後の画像である。左下の部分をみると多少ブレが見られるものの多くの部分でブレがなくなっているためこの配列を回答とした。

0.930421	0.018946	118.490952
-0.043542	0.976597	22.193656
-0.000097	0.000002	1.000000

## 2.7 3x3 行列の積を計算する関数 mult33 を実装し、 $m1d = m10 \cdot m0d$ を計算し、画像を合成

3x3 行列は、左の行列 1 行目の各要素を右側の行列 1 列目の各要素にかけて、それら 3 つの項を足しあわせ、1 行 1 列要素とする。左の行列 1 行目の各要素を右側の行列 2 列目の各要素にかけて … という手順を繰り返していく、3 行 3 列要素目までを計算すればよい。

その実装としては、2 重ループを用いて行った。左側は右側の 3 つ分が終わると次の行に進み、また、掛け合わず場所は定義より決まっているため、以下のソースコードのようにした。

```
void mult33(double d[3][3], double a[3][3], double b[3][3]){
    for(int i=0; i<3; i++){
        for(int j=0; j<3; j++){
            d[i][j] = a[i][0]*b[0][j] + a[i][1]*b[1][j] + a[i][2]*b[2][j];
        }
    }
}
```



図 20: m10 を修正した後の合成画像

そして、これを pano0.c の main 関数にて使用したのが以下である。ImageImageProjectionAlpha は、行列 m0d を使って im (0.jpg) を id に書き込む。m0d は結果を出力画像の中心寄りに表示するための平行移動（右に 100, 下に 100）を行う射影変換である。行列 m1d を使った ImageImageProjectionAlpha は、id に im (1.jpg) を書き込む。m10 は img1 と img0 の関係を表し、m0d は img0 と出力画像の関係を表す。img1 と出力画像の関係は m10 と m0d の積で表される。このときにパノラマ合成を行う必要があり、m10 に、先程求めた算出後の配列を使用。また、m1d は m10 と m0d の積で表されるため、mult33 を使い、積を計算した。パノラマ合成後の結果画像は図 21 に示した。

```

double m0d[] [3]={
    1,0,-100,
    0,1,-100,
    0,0,1
};

im=ImageRead("0.jpg");
ImageImageProjectionAlpha(id,im,m0d,.5);
double m10[] [3]={
    0.930421,    0.018946,    118.490952,
    -0.043542,   0.976597,    22.193656,
    -0.000097,   0.000002,    1.000000
}, m1d[3][3];
mult33(m1d,m10,m0d);

```



図 21: C 言語でのパノラマ合成の結果画像

2.8 「原画像の座標  $(u,v)$  を出力画像の座標  $(x,y)$  に変換する行列」を  $A$  とすると, 上記の実装で用いている  $a$  は  $A$  の逆行列である.

これを確認するために,  $\begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$  の, 逆行列  $\begin{pmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{pmatrix}$  をかけたときに 2 倍になるかどうかを見る.

homography.c の配列  $a$  を先程の逆行列のように変え, 実行すると図 22 のようになった.



図 22: 結果

きちんと拡大されていたことが確認できた。

### 3 変換行列の算出

#### 3.1 1.jpg を 0.jpg に重ねる 3x3 行列 m10 の計算法の詳細の手順の確認

初期状態が以下の表を確認する。この動作は行列  $A^T(Q^T)$ , R, ベクトル  $b^T$ ,  $tmp^T$  という操作で、45 回実行することで、4 組の対応点の座標から射影変換行列が計算される。

```
initialized
256.000000 0.000000 347.000000 0.000000 263.000000 0.000000 413.000000 0.000000
218.000000 0.000000 220.000000 0.000000 367.000000 0.000000 315.000000 0.000000
1.000000 0.000000 1.000000 0.000000 1.000000 0.000000 1.000000 0.000000
0.000000 256.000000 0.000000 347.000000 0.000000 263.000000 0.000000 413.000000
0.000000 218.000000 0.000000 220.000000 0.000000 367.000000 0.000000 315.000000
0.000000 1.000000 0.000000 1.000000 0.000000 1.000000 0.000000 1.000000
-94976.000000 -58880.000000 -160661.000000 -79810.000000 -100729.000000 -99677.000000 -218890.000000 -135051.000000
-80878.000000 -50140.000000 -101860.000000 -50600.000000 -140561.000000 -139093.000000 -166950.000000 -103005.000000

0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

371.000000 230.000000 463.000000 230.000000 383.000000 379.000000 530.000000 327.000000

0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
```

45 回実行後の行列が以下のようであった。

```
done
0.392371 0.000000 0.531847 0.000000 0.403100 0.000000 0.633005 0.000000
0.013540 0.000000 -0.437106 0.000000 0.876890 0.000000 -0.199546 0.000000
0.746709 0.000000 0.224407 0.000000 -0.041838 0.000000 -0.624754 0.000000
0.000000 0.392371 0.000000 0.531847 0.000000 0.403100 0.000000 0.633005
0.000000 0.013540 0.000000 -0.437106 0.000000 0.876890 0.000000 -0.199546
0.000000 0.746709 0.000000 0.224407 0.000000 -0.041838 0.000000 -0.624754
-0.295032 -0.448602 0.378993 0.576267 0.142043 0.215980 -0.226004 -0.343644
-0.448602 0.295032 0.576267 -0.378993 0.215980 -0.142043 -0.343644 0.226004

652.443867 549.877189 1.960322 0.000000 0.000000 0.000000 -301875.042231 -248248.300143
0.000000 165.750042 0.253778 0.000000 0.000000 0.000000 24290.303105 -46513.794686
0.000000 0.000000 0.304524 0.000000 0.000000 0.000000 33993.758735 26933.037130
0.000000 0.000000 0.000000 652.443867 549.877189 1.960322 -191217.160969 -167855.917552
0.000000 0.000000 0.000000 0.000000 165.750042 0.253778 -26368.674762 -79976.364619
0.000000 0.000000 0.000000 0.000000 0.000000 0.304524 26667.777043 21377.198091
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 7596.900540 1712.683064
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 5458.391125

371.000000 230.000000 463.000000 230.000000 383.000000 379.000000 530.000000 327.000000

0.980063 0.155844 98.500361 -0.055756 1.153389 0.503900 -0.000139 0.000316
```

#### 3.2 C 言語による実装例

講義ページ内の C 言語による実装のファイルを編集した。具体的には、MatrixQRDecompColMajor 関数と MatrixSimeqLr 関数を書き換えた。この 2 つの関数には、8 行 8 列分の操作はなかったため、すでに記載してある手続きと、資料の PDF の 4 組の場合の射影変換行列を参考に実装をした。

まず、MatrixQRDecompColMajor は、資料の PDF 内の式 (12) からの処理を行う。なお、実装例の配列 aT が 3 行分しか用意されていないが、実際には 8 行分必要であることに注意して、以下のように 8 行分宣言を行った。

```
double *aT[] = { Row(mt,0), Row(mt,1), Row(mt,2), Row(mt,3), Row(mt,4),
    Row(mt,5), Row(mt,6), Row(mt,7) } ;
```

次のような A の各列を  $a_i$  と表した行列

$$A := [a_0, a_1 \dots], R := (r_{ij}) \quad (1)$$

行列は、資料の PDF の式 (12) から式 (21) により Q に書き換えられ、積 QR はもとの A と等しくなる。具体的な動作手順としては、 $a_0$  の場合、単位ベクトルへの変換。 $a_1$  の場合、1 回の直交化と単位ベクトルの変換。 $a_2$  の場合、2 回の直交化のための動作と単位ベクトルへの変換というように規則に沿ったものになっている。なお、この過程で、R の要素も得られており、式 (12), 式 (14), 式 (17) 等の分母が対角項  $r_{ii}$ 、式 (13), 式 (15), 式 (16) 等の内積 ( $a_i^T a_j$ ) が非対角項  $r_{ij}$  である。その実装のソースを一部抜粋したのを以下で示す。

```
Elem(mtR,0,0) = t = sqrt(VP(aT[0],aT[0],W));
VSS(aT[0], 1/t, W);
```

//////////

```
Elem(mtR,0,1) = t = VP(aT[0], aT[1], W);
VSA(aT[1], aT[0], -t, W);
```

```
Elem(mtR,1,1) = t = sqrt(VP(aT[1],aT[1],W));
VSS(aT[1], 1/t, W);
```

//////////

```
Elem(mtR,0,2) = t = VP(aT[0], aT[2], W);
VSA(aT[2], aT[0], -t, W);
```

```
Elem(mtR,1,2) = t = VP(aT[1], aT[2], W);
VSA(aT[2], aT[1], -t, W);
```

```
Elem(mtR,2,2) = t = sqrt(VP(aT[2],aT[2],W));
VSS(aT[2], 1/t, W);
```

次に、MatrixSimeqLr は PDF 内の式 (22) の後ろに書いてある後退代入を行う。上三角行列で有ることに注意する。

$$x_2 = t_2/r_{22} \quad (2)$$

$$x_1 = (t_1 - x_2 r_{12})/r_{11} \quad (3)$$

$$x_0 = (t_0 - \sum_{i=1}^2 x_i r_{0i})/r_{00} \quad (4)$$

目的の解が  $i$  個なら,  $x_i$  の  $i$  が小さくなるにつれ, 引いていく要素が多くなる.  $3 \times 3$  の例を参考に,  $8 \times 8$  の実装したもののでソースコードの一部を抜粋したものを示す.

```
B[7] = B[7] / Elem(mtR,7,7);
B[6] = (B[6]-B[7]*Elem(mtR,6,7)) / Elem(mtR,6,6);
B[5] = (B[5]-B[6]*Elem(mtR,5,6)-B[7]*Elem(mtR,5,7)) / Elem(mtR,5,5);
B[4] = (B[4]-B[5]*Elem(mtR,4,5)-B[6]*Elem(mtR,4,6)-B[7]*Elem(mtR,4,7)) / Elem(mtR,4,4);
```

これを実装し, 実際に出力された画像を図 23 に示す.



図 23: 実際に出力された画像

### 3.3 どのような大きさの行列でも扱えるように, ループを用いて実装

まず, MatrixQRDecompColMajor の配列の個数を動的に確保することから始めた.

```
double **aT = (double **)malloc((sizeof (double *)) * mt->H);
```

配列の確保は malloc でポインタのポインタ aT と宣言することで, 後の処理で配列のように扱える. double \*\*aT という宣言は, ポインタ aT が示す先もポインタ\*aT であり, そのポインタ\*aT という宣言は, ポインタ aT が示す先もポインタ\*\*aT であるということである. その状態で, malloc により, double \*ポインタを目的の要素分宣言することで, 配列要素 aT[mt->H] 分求めることができる. この配列 aT[0],aT[1]... はポインタであり, その示す先の\*aT[0],\*aT[1]... は double 型であり, ポインタの配列を宣言することになる. その後, aT[0] から aT[mt->H] までの各要素に Row を格納することで mt->H がいくらであっても確保できる.

```
double **aT = (double **)malloc((sizeof (double *)) * mt->H);
//縦に伸びていく
for(int get=0; get<mt->H;get++){
    aT[get] = (double *)malloc((sizeof (double *) * mt->W));
    for(int i=0; i<mt->W; i++)
        aT[get][i] = 0.0;
}
```

```

    aT[get] = Row(mt, get);
}

```

そして、その後のシュミットの直交化手順だが、二重ループを用いた。確保した要素分(W個分)のaT[i]について手続きを施す。iが増えるに連れ、行うVSA(ベクトルのスカラ倍加算)の回数も増えることに注意すると、その回数はiを超えない回数行う事となる。そして、jとiが同じ時にベクトルを単位ベクトルにする動作をする。

以下がそのソースコードである。

```

for(int i=0; i < W; i++){
    for(int j=0; j <= i; j++){
        // printf("%d,%d\n", i, j);
        if(j == i){
            Elec(mtR, j, i) = t = sqrt(VP(aT[j], aT[i], W));
            VSS(aT[i], 1/t, W);
        }else{
            Elec(mtR, j, i) = t = VP(aT[j], aT[i], W);
            VSA(aT[i], aT[j], -t, W);
        }
    }
}

```

もう一つ、MatrixSimeqLrについて。要素数は、mtR->Wにより可変的に行うことができる。ここで、 $8 \times 8$ の場合を考えてみる。配列の要素数は0~7の8個である。B[7]の場合、自身をElec(mtR,7,7)で割る動作のみ、B[6]の場合、自身からB[7]にElec(mtR,6,7)をかけたものを引き、Elec(mtR,6,6)で割る。B[5]の場合、自身からB[6]にElec(mtR,5,6)をかけたものを引き、B[7]にElec(mtR,5,7)をかけたものを引き、Elec(mtR,6,6)で割るという手続きが続く。これを2重ループに落とし込むと、大きいループの回数は、要素数個分繰り返す。今回は、iの大きい順から手続きを施した。一度初回を飛ばし、2回目のループであるi=6のときについて考えると、 $i+1 = 7$ で一つ大きい添字の要素を参照でき、 $\sum$ と同等の処理を内部のループで行える。 $\sum$ のくりかえし最大回数は、要素数個分であるから、 $mtR->W = 8$ つまり、配列の添字7までである。その後、Elecで割るという動作を行えば良い。では初回のi=7の時を見てみる。一つ添字の大きい要素は8だが、内部の $\sum$ にあたるループの上限7を超えてるのでループは実行されず、自身をElecで割るという動作のみ行われることになる。

以下のソースコードである。

```

for(int i=mtR->W-1; 0<=i; i--){
    for(int j=i+1; j<mtR->W; j++){
        B[i] -= B[j]*Elec(mtR, i, j);
    }
    B[i] = B[i] / Elec(mtR, i, i);
}

```

実行したところ、図 23 と同じ結果が得られた。

### 3.4 合成画像の品質を改善

元々示されていた特徴点は、

256,218, 371,230,  
347,220, 463,230,  
263,367, 383,379,  
413,315, 530,327,

だが、出力された画像は図 23 であり、あまり精度がよくない。それを以下のように変更し、なるべく特徴点同士が離れるようにした。

147,535,268,544,  
116,209,235,224,  
509,205,629,210,  
432,517,550,530

それにより実行した結果が図 24 である。少し粗はあるものの、図 24 と比べると大きく改善しただろう。



図 24: 特徴点の改善後

### 3.5 自分で撮影した画像を合成

撮影した画像がとても大きいサイズだったため、実験同様 768x576 までサイズを落として実行した。合成に用いた画像は、図 25、図 26 である。正確に同じ視点を撮ることができなかつたため、真横に 100px 分だけ移動した画像になっている特徴点は、以下のように設定した。

```
215,320, 315,319,  
220,507, 320,506,  
651,480, 751,479,  
600,135, 700,134
```



図 25: 合成 1 枚目



図 26: 合成 2 枚目

プログラムを実行すると、図 27 のようなパノラマ画像が合成された。

## 4 特徴点の自動検出

画像合成の自動化には、特徴点の自動検出と自動対応付けが必要である。これは、次のように 2 段階で行う。各画素が特徴点としてふさわしいかどうかを測る指標を計算する（第 4 回の課題）。それが極大となる点を探し出す（第 5 回の課題）。これらの計算の主要部分は、畳み込みフィルタである（ある点の出力値はその周辺の入力値の線形和で計算される）。畳み込みフィルタは計算量が多いが、並列性が高いため GPU やマルチコア CPU によって効率的に計算できる。

### 4.1 cpu で「特徴点らしさ」画像を出力

資料の例を元に実装を完成させた。DElem, Elem が image.h に定義されていなかったので、第 3 回および第 2 回を元に移植させた。そして、Matrix 構造体 Matrix \*MatrixAlloc についても移植をおこなった。

最後に、今回の要である ImageFeature 関数についてだが、iy に関する計算。そして、 $\sum_{ixx}$  に相当する計算を行う処理を加えた。以下がその部分のソースコードである。

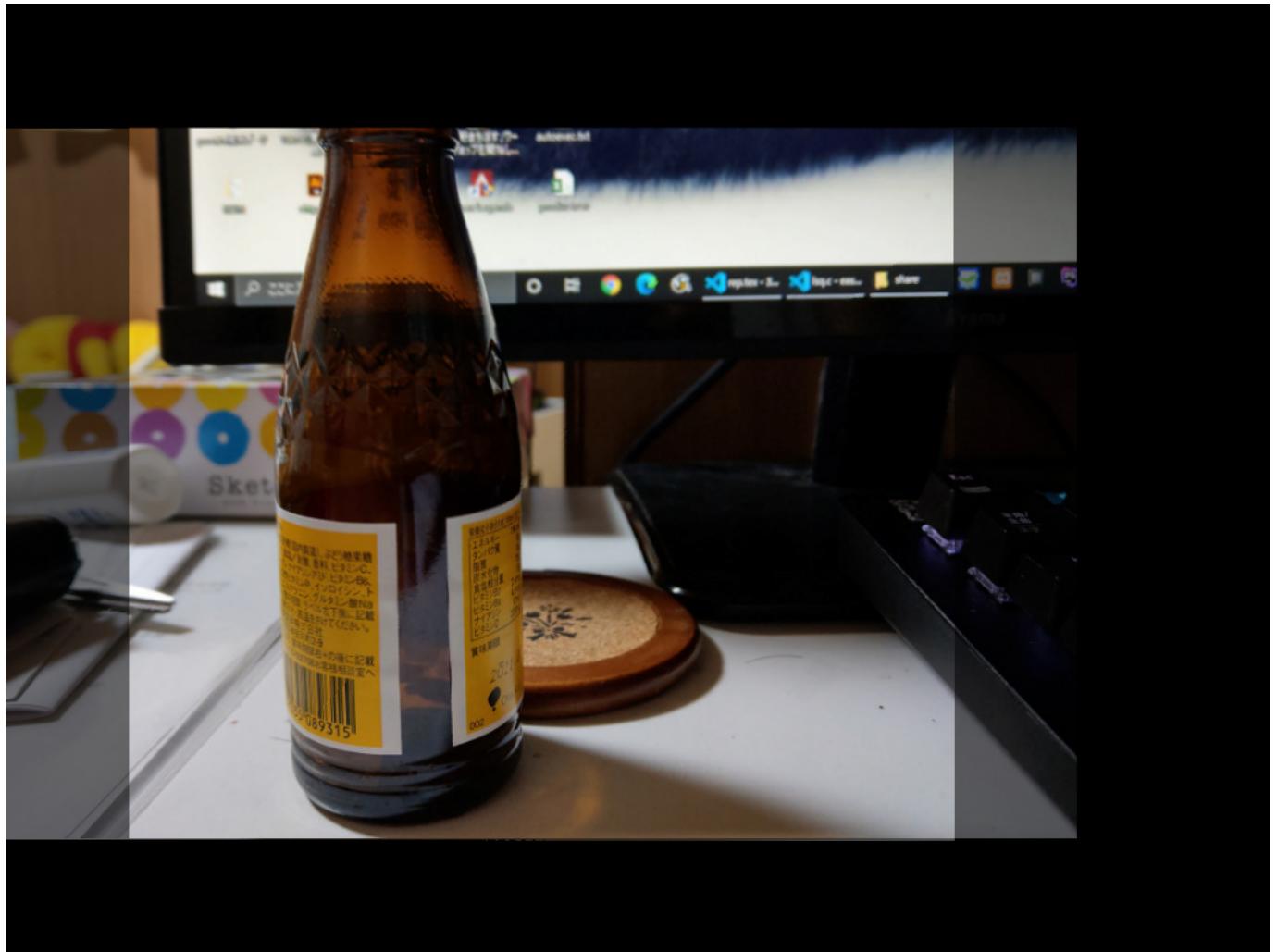


図 27: 自分で撮影した画像のパノラマ

```
ix = IElem(im, x+u+1, y+v, 1) - IElem(im, x+u-1, y+v, 1);
iy = IElem(im, x+u, y+v+1, 1) - IElem(im, x+u, y+v-1, 1);
ixx += ix*ix; // ixx だけでなく ixy,iyy も計算する.
iyy += iy*iy;
ixy += ix*iy;
```

さらに、DElem には行列

$$\begin{pmatrix} i_{xx} & i_{xy} \\ i_{xy} & i_{yy} \end{pmatrix}$$

の小さい方の固有値を入れることで求める事ができるが、2行2列の行列の固有値は2次方程式の解の公式で求めることと同じであるため解の公式を使った。

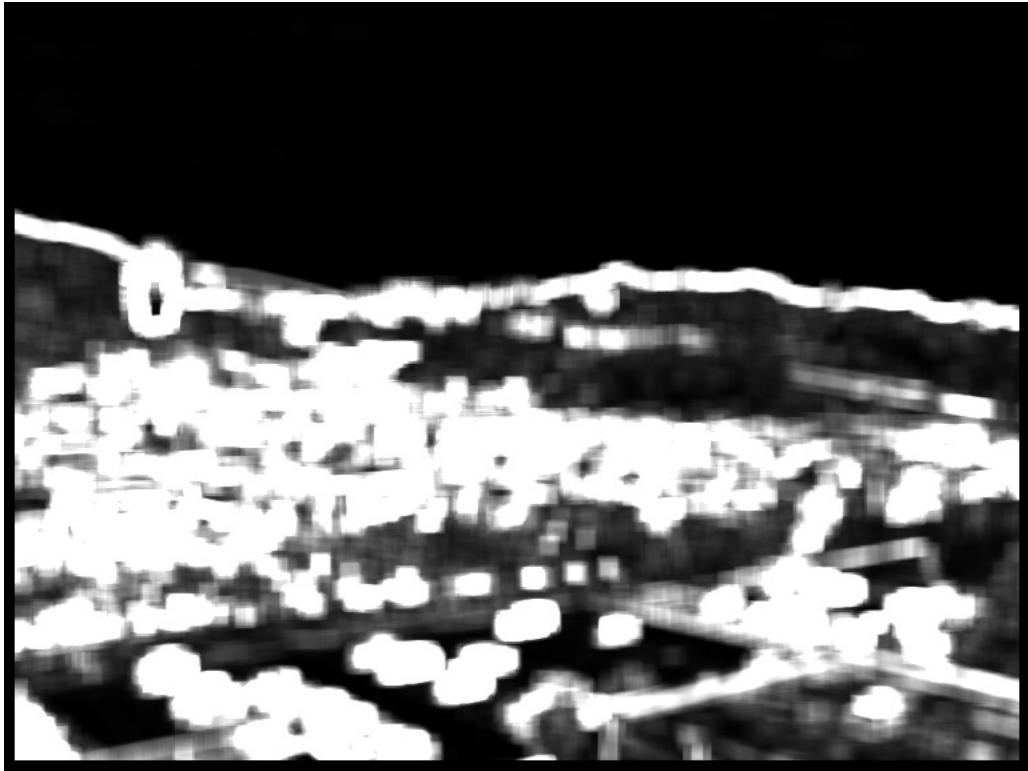


図 28: 出力結果

```
DElem(im2,x,y) =
((ixx + iyy)-sqrt(pow(ixx+iyy,2.0)-4.0*(ixx*iyy-ixy*ixy)))/2.0;
```

$W$ で、自分の周りの上下左右の7マスを見て特徴点を取っている。この実装で出力された結果は図28に示した。このときの  $W$  は 7 である。処理時間は 3.9Ghz の CPU で 144msec だった。

## 4.2 CPU での分解

資料を元に、3重ループを2回行うことで実装した。資料には、特徴点検出の  $ixx, ixy, iyy$  は  $15 \times 15$  の平滑化であり 225 個の和である。これを直接計算するより、横に隣接する 15 画素の和を中間配列に格納し、この配列の縦 15 要素の和によって最終結果を計算するとあるので、それを再現するようにした。出力結果は図29に示す。計算時間は 36msec であった。

```
int x,y,u,v,ix,iy;
for(y=1;y<im->H-1;y++) for(x=W+1;x<im->W-W-1;x++){
    double ixx,ixy,iyy;
    ixx=iyy=ixy=0;
    for(u=-W;u<=W;u++){
        if(u>0) u-=W;
        if(u<-W) u+=W;
        ixx+=im->px[y*x+u];
        iyy+=im->py[y*x+u];
        ixy+=im->px[y*x+u]*im->py[y*x+u];
    }
    DElem(im2,x,y) = ((ixx + iyy)-sqrt(pow(ixx+iyy,2.0)-4.0*(ixx*iyy-ixy*ixy)))/2.0;
}
```

```

    ix = IElem(im, x+u+1, y, 1) - IElem(im, x+u-1, y, 1);
    iy = IElem(im, x+u, y+1, 1) - IElem(im, x+u, y-1, 1);
    ixx += ix*ix;
    iyy += iy*iy;
    ixy += ix*iy;
}
DElem(im3,x,y) = ixx;
DElem(im4,x,y) = ixy;
DElem(im5,x,y) = iyy;
}

for(y=W+1;y<im->H-W-1;y++) for(x=W+1;x<im->W-W-1;x++){
    double ixx,ixy,iyy;
    ixx=iyy=ixy=0;
    for(v=-W;v<=W;v++){
        ixx += DElem(im3, x, y+v);
        ixy += DElem(im4, x, y+v);
        iyy += DElem(im5, x, y+v);
    }
    DElem(im2,x,y) =
        ((ixx + iyy)+sqrt(pow(ixx+iyy,2.0)-4.0*(ixx*iyy-ixy*ixy)))/2.0;
}

```

### 4.3 実行時間の比較

CPU と CPU 分解の 2 つの時間を示す.

方法	時間 (msec)
CPU	144
CPU 分解	34

### 4.4 自分で撮影した画像で実験

作成した C プログラムに、図 30 を与えて出力された結果を図 31 に示した.

### 4.5 W の値をコマンドラインから指定

まず、image.h の ImageDrawBox の宣言に W を整数型で追加した。そして、使用する際にコマンドライン引数を atoi を使い、整数型に変えることで対応した。

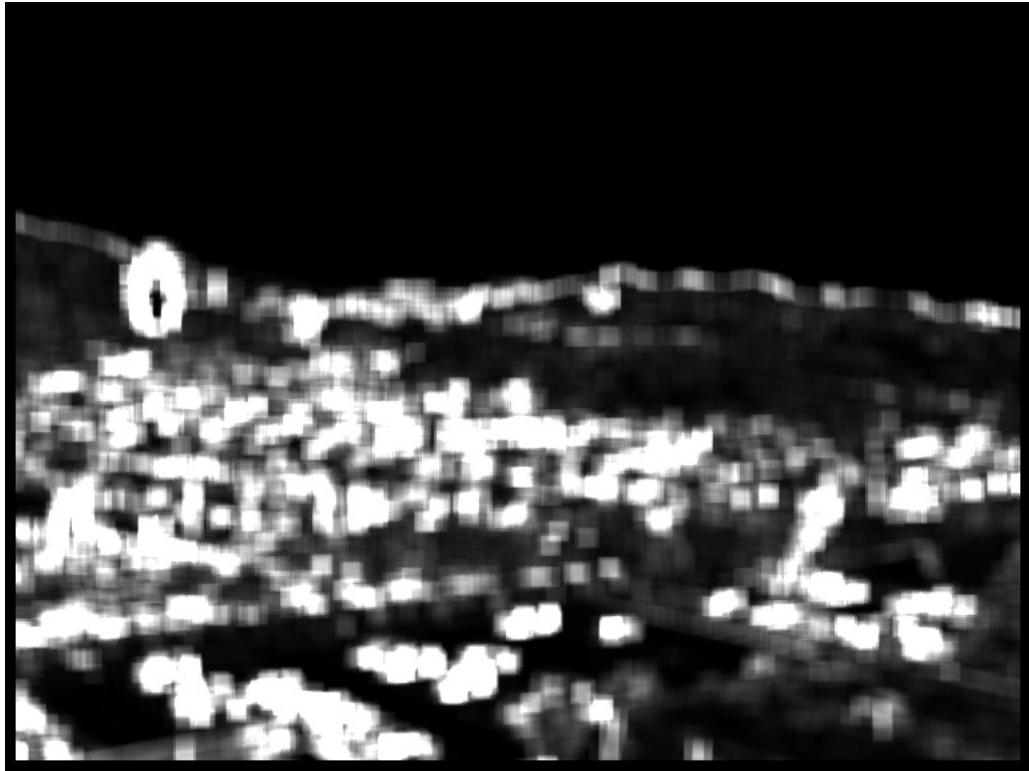


図 29: 出力結果

```
ImageDrawBox(im, kk[i][0], kk[i][1], atoi(av[2]));
```

#### 4.6 W の値を変えて複数回実行

w の値を 0 から 10 まで変えて画像の出力をした。数字が大きくなるにつれて処理時間も増えていった。その出力結果を数枚添付する。W=2 から W=4あたりが綺麗に輪郭が出ている。W=7 以上は輪郭が太くなりすぎてあまり綺麗でなかった。

## 5 特徴点の自動検出 2

特徴点指標画像の極大点を検出する。極大の判定は容易に並列実行できるが、極大の記録は並列的ではないため注意が必要である。前項で得られた極大点は、周辺よりは特徴的だが、画像全体で特徴的であるとは限らない。そこで、検出された点から画像全体で特徴的な点を選び出す。結果は図 38 と図 39 に示した。



図 30: 元画像

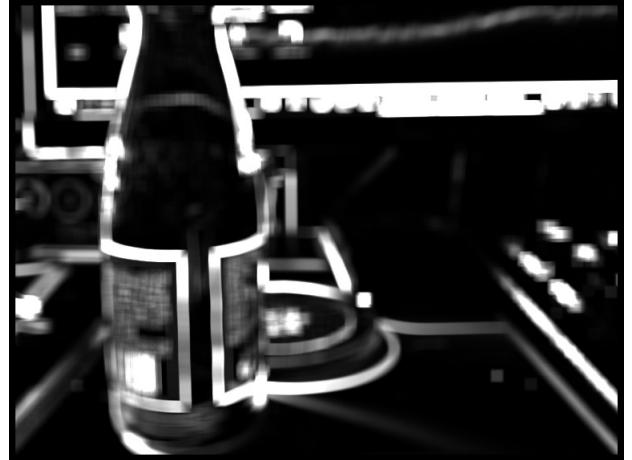


図 31: 出力結果

### 5.1 極大値を探し出し、その座標を配列に記録する部分を完成

外側の2重ループで、 $x$ 方向と $y$ 方向の走査をしている。中身で $W$ に対応する矩形領域内の最大値を探す。 $W$ はコマンドライン引数から与えることができる。このとき、 $W$ の値が小さすぎると、点が出てこずにセグメンテーションフォールトが起きてしまう。

そして、最大値が操作している点と同じならそこを特徴点として記録している。以下そのソースコードである。

```

for(y=W+1;y<im2->H-W-1;y++){
    for(x=W+1;x<im2->W-W-1;x++){
        double max=-1;
        for(v=-W;v<=W;v++){
            for(u=-W;u<=W;u++){
                // (x,y)を中心とする 15x15 の矩形領域内で DElem(im2,x+u,y+v) の最大値を探す。
                if(DElem(im2, x+u, y+v) > max){
                    max = DElem(im2,x+u,y+v);
                }
            }
        }
        // 最大値が DElem(im2,x,y) と等しいなら、(x,y) を特徴点として記録する。
        if(max == DElem(im2,x,y)){
            a = n++;
            w[a][0] = x;
            w[a][1] = y;
        }
    }
}

```



図 32: W=2



図 33: W=3

```
w[a][2] = max;  
}  
}  
}
```

## 5.2 得られた極大点リストから、「特徴点らしさ」の大きいものを N 個選び出す

ここでは、qsort 関数を用いてソートをした。

```
qsort(kk, kw, sizeof(kk[0]), desc);
```

kk は予め宣言されている配列で、kw は MatrixLocalMax の返り値として赤い点の個数が与えられている。要素の 1 つ分は x,y,max について 3 つ分があるので kk[0] で記載。desc という関数は降順に並び替えるものである。

```
int desc(int left[3], int right[3])  
{  
    return right[2]-left[2];  
}
```

出力された点を一部抜粋すると、max の値で降順に並んでいる。

```
383 465 790611  
614 440 755117  
354 500 745975  
403 462 714745
```



図 34: W=4



図 35: W=5

```
242 528 701629
295 498 656782
687 482 626184
```

### 5.3 速度とメモリを効率化

実装例で確保された領域 `int kk[9999][2]` の要素数は、検出され得る最大数より遙かに少なく問題が起きる可能性があると資料にあるのはうなづける。また、画素数と同数の作業領域を確保するのは無駄が多い。そこで資料を参考に、現在までに発見された点を、`MAX` 個まで保持できる領域を用意し、`MAX+1` 個目以降の追加時には、新しい候補か現在の最下位候補のどちらかを破棄することで、候補が `MAX` 個を越えない様にする。実装には挿入ソートを用いた。以下そのソースコードである。

```
for(v=-W;v<=W;v++) for(u=-W;u<=W;u++){
    if(DElem(im2, x+u, y+v) > max){
        max = DElem(im2,x+u,y+v);
    }
}
if( max == DElem(im2, x, y) ){
    a = n;
    if(n < MAX) { n++; }
    for(;a>0 && w[a-1][2] < DElem(im2, x, y);a--) {
        w[a][0]=w[a-1][0]; w[a][1]=w[a-1][1]; w[a][2]=w[a-1][2];
    }
    w[a][0]=x; w[a][1]=y; w[a][2]=max;
}
```

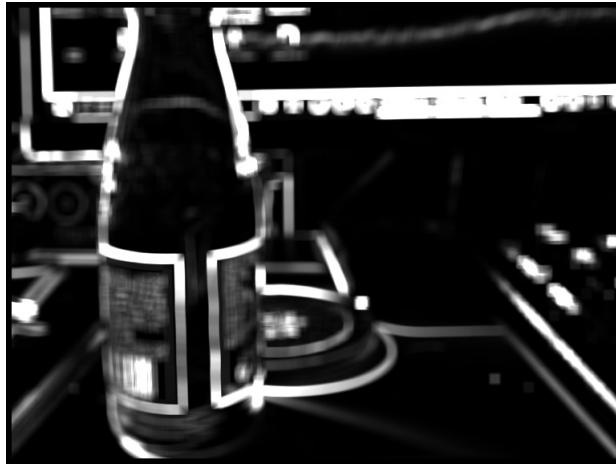


図 36: W=6



図 37: W=7

## 6 特徴点の自動対応付け 1

高精度のパノラマ画像を生成するには正しい射影変換行列が必要であり、このためには両方の画像で同じ物体を指している特徴点対が4組以上必要である。まず、各画像で独立に検出された特徴点群の全ての組み合わせに対して類似度を計算し、この中から同一物体を指す組を見つけ出す。最も簡単な類似度の指標は、特徴点周辺の小領域の画素値の差の2乗和（Sum of Squared Differences）である。特徴点対を選ぶ際、一つの点が複数の点と対応しないようにすることや、点の優先順位について考慮する。

### 6.1 greedy.c を完成させ、第1画像の第 i 特徴点と、第2画像の第 j 特徴点の類似度（下記の値）を全ての i,j の組について計算し、行列の (i,j) 要素に格納した表を作りなさい

結果

53	124	5	59	41	55	72	53	54	116	92	63	85	35	108	101	40	61	52	83	76	56	56	53	67	69	64	57	47	92
66	112	38	41	59	47	56	45	54	75	102	65	84	52	100	82	50	57	51	63	70	48	50	53	56	60	55	49	44	99
83	120	83	83	78	77	68	43	88	63	115	78	108	74	95	46	80	56	41	61	80	49	86	62	69	50	72	79	76	99
43	88	51	67	9	63	61	49	32	90	101	40	66	43	92	74	44	62	45	52	54	39	42	43	54	69	61	46	35	108
13	96	54	69	45	85	80	59	63	121	122	52	97	54	112	96	54	68	55	68	67	56	66	63	59	91	71	58	45	103
76	134	32	57	49	27	50	54	48	97	115	77	95	53	119	93	59	57	62	68	83	55	58	63	64	61	64	55	56	121
59	117	38	42	45	40	52	49	43	81	90	57	74	43	94	70	47	51	46	61	65	41	43	40	56	57	46	42	40	94
82	114	55	57	65	30	34	46	58	76	103	66	90	57	91	69	63	45	44	64	69	49	55	49	61	44	46	58	56	116
115	98	101	81	95	75	66	47	90	21	94	79	83	90	71	26	85	41	46	71	76	63	69	58	64	29	50	60	80	89
78	76	68	60	64	49	38	28	57	39	86	53	56	75	61	39	62	39	43	50	54	49	46	52	53	36	34	33	47	73
60	119	52	58	32	47	59	46	3	90	121	63	71	46	121	76	62	60	59	53	65	45	45	63	62	70	57	41	48	129
62	94	55	67	60	63	51	28	67	57	51	47	59	44	48	36	47	35	15	72	59	31	54	28	52	32	33	51	52	61
57	54	65	55	45	78	65	52	64	70	64	4	43	51	56	68	37	53	40	57	34	49	28	36	43	54	35	32	27	62
124	102	102	102	111	97	76	70	135	66	41	65	68	98	14	63	75	67	51	120	83	78	74	44	70	46	47	81	80	56
61	93	54	58	59	44	38	39	59	67	72	44	62	45	72	44	49	11	29	57	56	41	53	35	45	33	36	37	47	85
56	135	36	63	51	52	61	46	51	95	80	56	75	9	91	76	46	46	32	76	70	32	61	38	55	57	48	59	55	102
106	100	98	90	88	85	68	47	85	46	76	70	71	92	58	3	77	41	47	81	81	63	71	49	72	38	48	51	77	72
138	137	94	108	121	124	111	97	140	105	22	79	69	94	44	91	74	83	60	163	111	96	77	48	95	71	67	92	48	
55	88	37	61	46	61	62	47	60	96	67	36	56	39	74	77	6	54	35	76	53	50	40	32	52	62	49	45	37	65
65	109	56	66	49	48	50	35	53	68	83	57	82	45	67	50	62	43	27	70	67	12	59	42	44	47	44	51	53	104
58	63	83	69	54	68	63	46	56	72	160	61	87	78	129	79	74	58	66	10	42	62	57	80	55	64	69	50	43	129
103	74	89	75	75	94	80	66	81	64	66	49	6	77	61	68	61	65	59	83	58	81	48	56	71	49	39	46	55	58
60	50	80	52	51	78	71	56	66	73	114	36	58	61	91	81	49	64	49	38	1	56	41	61	53	62	58	41	28	89
62	86	57	42	62	58	53	44	76	62	74	44	69	39	52	71	45	49	28	65	41	32	47	37	33	41	38	55	40	76
39	116	46	62	34	55	56	43	44	102	92	42	76	31	88	75	45	50	27	65	55	30	55	38	59	62	50	50	43	113
54	57	52	29	48	54	63	55	48	68	101	39	55	54	91	86	43	56	56	37	29	50	26	56	31	60	50	28	16	85
92	88	73	69	78	59	49	35	78	31	68	58	57	65	49	35	64	28	30	64	59	51	59	43	54	5	32	52	65	65

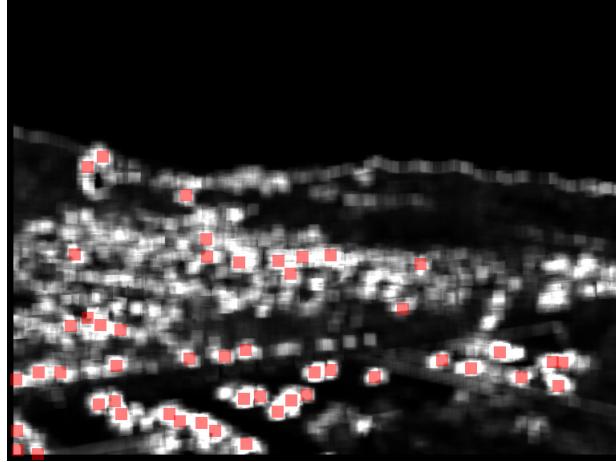


図 38: 50 個のとき

59	74	64	46	48	56	44	39	53	62	78	30	49	39	58	56	44	50	32	47	29	33	38	34	45	42	26	34	31	84
95	63	90	63	80	87	77	54	81	33	62	55	43	84	53	34	67	52	49	65	57	64	43	48	49	37	45	44	52	42
70	70	58	60	48	78	71	48	55	55	63	33	39	53	70	45	40	36	40	55	52	54	32	30	52	38	36	26	39	56

表中の色の異なる要素は、次の条件を満たす様に選ばれている: (i) 各行に 1 つ, (ii) 各列に 1 つ, (iii) 色の異なる要素の合計が小さい。第 i 行, 第 j 列に色がついているとき, 第 1 画像の第 i 特徴点と第 2 画像の第 j 特徴点が同一の物体であると期待される。

## 6.2 上記の条件 (i)–(iii) の意味を考察しなさい。

(i)(ii) 画像 1 に対して画像 2 の特徴点が複数選ばれないことを意味している。 (iii) 画像 1 と画像 2 の特徴点の類似度が高いということを意味している

## 6.3 方法 1 の問題点

- SSD の和が方法 2 と比べて大きいので精度がよくない。
- もし対応する特徴点が二枚目の画像にないときはすごく異なる対応点が選ばれてしまう。
- 同じ列に選ばれた点よりもとても強い特徴点がいても選ばれないことがある。

## 6.4 方法 2 の問題点

- 方法 1 同様, もし対応する特徴点が二枚目の画像にないときはすごく異なる対応点が選ばれてしまう。
- INFINITY で上書きしたものは本当にいらないものなのかどうかの判断ができていない。

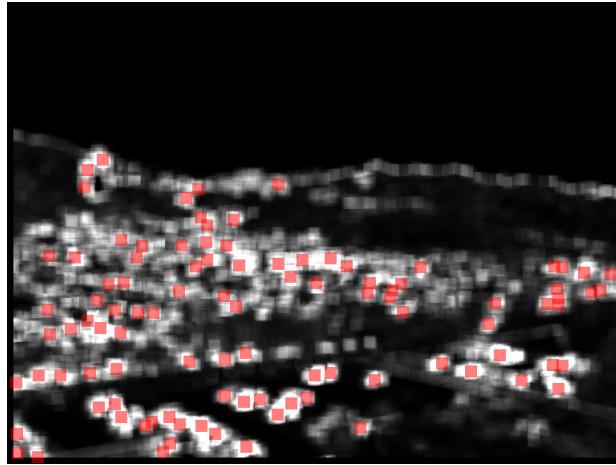


図 39: 100 個のとき

## 6.5 実装例 1 を greedy.c に追加し, 動作させなさい. また, これをもとに方法 2 を完成させなさい

以下が方法 2 のソースコードである. 点の数だけ対応を見つける際に, 各行, 各列の最小値を見つけて, かぶらないようにその点の横と縦を見ないようにする動作をしている.

```

int matchMethod2(double w[][][4],Matrix*mt,Image*im ,
    int x1[][][2],int N1,Image*im2,int x2[][][2],int N2){
    int i,j,k,l,n = 0;
    int minElem[3] = {0,0,INFINITY_INT};
    // SSD の表中の最小値
    for(i=0;i<MAX;i++){//点の数
        for(j=0;j<mt->W;j++){//横幅
            for(k=0;k<mt->H;k++){//縦幅
                if(Elem(mt,j,k) < minElem[2]){
                    minElem[0] = j;
                    minElem[1] = k;
                    minElem[2] = (int)Elem(mt,j,k);
                    //printf("%d,%d,%d\n",minElem[0],minElem[1],minElem[2]);
                }
            }
        }
        for(k=0;k<MAX;k++) Elem(mt,minElem[0],k)=INFINITY_INT;//横固定
        for(k=0;k<MAX;k++) Elem(mt,k,minElem[1])=INFINITY_INT;//縦固定
        //MatrixPrint(mt);
    }
}

```

```

printf("%d,%d,%d,%d,\n",x1[minElem[0]][0],x1[minElem[0]][1],
x2[minElem[1]][0],x2[minElem[1]][1]);
w[n][0] = x1[minElem[0]][0];
w[n][1] = x1[minElem[0]][1];
w[n][2] = x2[minElem[1]][0];
w[n][3] = x2[minElem[1]][1];
n++;
minElem[2]=INFINITY_INT;
}
return n;
}

```

## 6.6 4組の特徴点対を選び、射影変換行列を計算し、合成画像を作成しなさい。

算出した特徴点

315,495,564,507,  
 259,538,507,548,  
 119,196,367,211,  
 142,517,393,522,

出力した画像は図 40 に記載した。



図 40: 結果

## 6.7 下記の観点で考察または実装しなさい.

SSD(i,j) の表の作成は、 N1 や N2 が大きい時にはかなりの演算を要する。

### 6.7.1 calcSSDtable の i のループを openMP で並列化することは容易である。

実際に、これはループの前に`#pragma omp parallel for private (j)` の記載をつけ、`fopenmp` オプションをつけてコンパイルし、実行できた

## 7 特徴点の自動対応付け 2

厳密解法や全解探索が難しい問題でも、乱数によって近似的に扱える場合がある。ここでは RANSAC と呼ばれるアルゴリズムを用いて特徴点の自動対応付けを行う。

### 7.1 RANSAC (あるいは独自の方法) で自動対応付けを実現し、画像を合成しなさい

パノラマ画像での各画像から抽出してきた特徴量は、もう一方の画像に対応点が存在していない「外れ値」が含まれており、最小二乗法によりどの点とどの点が対応するかを計算すると、外れ値ではうまく対応関係が作れない。そこで、パノラマ画像作成での対応点計算には RANSAC により外れ値は見ないよう上手い対応点を見つける。

RANSAC では以下の 3 つの手続きを最適な評価値(スコア)が(3)で出てくるまで繰り返す

- (1):あらかじめ決めておいたサンプル数 n だけランダムにデータをサンプリング
- (2):サンプルした n 個のデータを用い、モデルのパラメータを算出(最小二乗法で誤差を最小化)
- (3):処理 2 で求めたモデルのパラメータに対して、あらかじめ範囲を決めておいた閾値内のデータのみを用いて、どれくらい正しいかのスコアを評価

この(1)～(3)の処理による各繰り返しにおいて、外れ値ではない正しい特徴点ができているかを判断する。

#### 7.1.1 問題点

試行回数が少ないと上手く合成できない。閾値が適正でないと上手く合成できない。

#### 7.1.2 改善策

試行回数を増やし、閾値を小さくすることで改善できた。試行回数は対応点集合の中に含まれる誤りの割合を p とすることで見積もることができるが今回は余力がないためやっていない。

## 7.2 合成結果が芳しくない場合は、 w[] に用意する候補数、「重なり」， 試行回数等を変える。

実験結果の図 41 は閾値を 5, 試行回数を 100 万回にして行った。

閾値が大きいと上手く出ず, 試行回数が少なくとも上手く出なかった。

## 7.3 下記の観点で考察または実装しなさい

### 7.3.1 4つの乱数 (i0,i1,i2,i3) を発生させるとき, 同じものが 2 度以上選ばれないようにする方法

以下実装したソースコードである。

```
void initRndAry(int rndAry[MAX]){
    for(int i = 0; i<MAX; i++){
        rndAry[i] = i;
    }
    srand(__rdtsc()); // __rdtsc については第4回を参照
}

void chooseFourNumbers(int rndAry[MAX]){
    for(int i=0;i<4;i++){
        int j, t;
        j = (int)((long long)rand()*(MAX-i)/(RAND_MAX+1LL))+i; // 亂数関数は stdlib.h で
        // 宣言されている。
        t = rndAry[i];
        rndAry[i] = rndAry[j];
        rndAry[j] = t;
    }
}
```

rndAry は, 0 から MAX-1 までの数を 1 つずつ保持している。この配列の内容を入れ替える操作によつて, 数値の順序は変わるが, 全ての数値が一回づつ現れる性質は変わらないため, 相異なる数個の乱数が得られる。

上記の rndAry[0] から rndAry[3] を使う部分では, 配列 w[][4] から 4 行を選んで変換行列を算出する以下にそのソースコードを示した。

```
void calcHomography(double H[3][3], double w[][4], int rndAry[MAX],
Matrix *cmA, Matrix *vt, Matrix *mtR, Matrix *tmp){
    int a = rndAry[0], b = rndAry[1], c = rndAry[2], d = rndAry[3];
    double ww[] [4] = {
        w[a][0], w[a][1], w[a][2], w[a][3],
        w[b][0], w[b][1], w[b][2], w[b][3],
```

```

w[c][0], w[c][1], w[c][2], w[c][3],
w[d][0], w[d][1], w[d][2], w[d][3],
};

// create A (col-major)
for(int i=0;i<4;i++){
    Elem(cmA,0,i*2) = ww[i][0];
    Elem(cmA,1,i*2) = ww[i][1];
    Elem(cmA,2,i*2) = 1;
    Elem(cmA,3,i*2) = 0;
    Elem(cmA,4,i*2) = 0;
    Elem(cmA,5,i*2) = 0;
    Elem(cmA,6,i*2) = -ww[i][0]*ww[i][2];
    Elem(cmA,7,i*2) = -ww[i][1]*ww[i][2];
    Elem(cmA,0,i*2+1) = 0;
    Elem(cmA,1,i*2+1) = 0;
    Elem(cmA,2,i*2+1) = 0;
    Elem(cmA,3,i*2+1) = ww[i][0];
    Elem(cmA,4,i*2+1) = ww[i][1];
    Elem(cmA,5,i*2+1) = 1;
    Elem(cmA,6,i*2+1) = -ww[i][0]*ww[i][3];
    Elem(cmA,7,i*2+1) = -ww[i][1]*ww[i][3];
    Elem(vt,0,i*2) = ww[i][2];
    Elem(vt,0,i*2+1) = ww[i][3];
}

MatrixQRDecompColMajor(mtR, cmA);
MatrixMultT(tmp, vt, cmA);
MatrixSimeqLr(tmp, mtR);

H[0][0] = Elem(tmp, 0, 0);
H[0][1] = Elem(tmp, 0, 1);
H[0][2] = Elem(tmp, 0, 2);
H[1][0] = Elem(tmp, 0, 3);
H[1][1] = Elem(tmp, 0, 4);
H[1][2] = Elem(tmp, 0, 5);
H[2][0] = Elem(tmp, 0, 6);
H[2][1] = Elem(tmp, 0, 7);
H[2][2] = 1;
}

```

そして、変換行列の信頼度を評価するため、上記の「検出された点と変換された点の距離が十分に小さい点」の数を計算する。以下そのソースコード部分である。

```

int calcScore(double H[3][3], double w[][4]){
    int score=0;
    for(int i=0;i<MAX;i++){
        double x=w[i][0], y=w[i][1], u=w[i][2], v=w[i][3],
               x_prime = H[0][0] * x + H[0][1] * y + H[0][2],
               y_prime = H[1][0] * x + H[1][1] * y + H[1][2],
               z_prime = H[2][0] * x + H[2][1] * y + H[2][2], // 変換行列と (x,y,1)^T の積

        du = (x_prime/z_prime) - u,
        dv = (y_prime/z_prime) - v; // (x,y) を変換した座標(第2回の概要を参照)と (u,v) の
差
        if(du*du+dv*dv < 10) score++; // w[i] は正しい;
        else score += 0; // w[i] は正しくない;
    }
    return score;
}

```

この実装により得られた画像は図 41 に示した。

## 8 まとめ

これまでの内容をまとめて、入力された 2 枚以上の画像を自動で合成するプログラムを作成した。ここではその問題点と工夫した点について述べる。

実装したソースコードは第 10 章にまとめて記述する。内容は出力画像領域を確保し、第 1 画像を読み込む。画像から特徴点を検出し、有効な点を選び出す。第 2 画像を読み込む。画像から特徴点を検出し、有効な点を選び出す。貪欲法・RANSAC 等を用い適切な 4 組の対応点を選び、変換行列を算出する。前項で得られた変換行列をほぼ満たす n 組の対応点を使って、より良い変換行列を算出する。第 1 画像と第 2 画像を合成。その後、第 3 画像を読み込み、同様の処理を行い、第 2 画像と合成した後、すべての 3 枚を張り合わせたものを画像ファイルに出力するという動作である。

### 8.1 問題点

今回作成したパノラマ画像のプログラムは、3 枚の入力しか対応しておらず、入力された画像の枚数に応じて動的に画像を描画してくれるものではない。

また、一部並列動作や分解などを行っていないため、高速なプログラムにはなっていないこと。

合成する他方の画像に対応する特徴点が見つけられない場合に上手く画像を合成できること。

異なる視点からの画像だと合成がうまくできないこと。



図 41: 結果

## 8.2 工夫した点

2枚でなく、3枚で合成ができるようにした。

画像によって上手く特徴点を見つけられる閾値が違うので、コマンドラインで実行をするときに閾値と、走査する範囲を指定できるようにした。第2から第4引数までが画像のデータで、第4引数が閾値、第5引数が $15 \times 15$ などの走査範囲の指定である。

実行例

```
./a.out IMAG1537.jpg IMAG1538.jpg IMAG1539.jpg 4 5
```

MatrixLocalMax という特徴点がいいものか見る関数の処理量が大きいため、縦の走査と横の走査で処理を分解し、なるべく軽い動作にしたこと。

自分の画像で合成結果を試したこと。撮る画像によっては上手く特徴点を見つけられないことがあり、そもそも画像の撮り方に苦戦をした。

image.c という 1 つのファイルにまとめることで、image.h をヘッダーに読み込むようにして、コンパイル時の指定を少なくし、構造的にも 1 つのファイルに複数の関数が並ばないように役割を決めた作成にした。

### 8.3 自分で撮影した写真の合成を試みなさい

作成したプログラムを用いて撮影した画像を合成することを試みた。当初、黒いキーボードの上に、白い色のマスコットを置いた画像で行っていた。しかし、いくら閾値などを変えてても特徴点を 4 点も見つけられなかったり、best\_score の値が 4 とか 6 あまり大きな値を見つけられず失敗した。図 42 がその結果である

その後、どうにか特徴点らしきものが多く出そうな画像を撮って行き閾値や走査する範囲を変えてみると先程よりは上手く合成できた。図 43 がその結果である。

## 9 感想

本講義では、複数枚のパノラマ画像の合成についての基本的な内容についての実際の実装方法を実装することで、その内部処理および画像処理の仕組みの一部について知ることができたようだ。GPU による実装や、妥当な行列が得られるまでの試行回数の見積もり、遙かにより効率的な特徴点数が多い時の類似度の表から最小値を k 回探し出す calcSSDtable 関数の実装などの米印 2 つの発展課題などに取り組む余力がなかったが、今回の実験で画像処理について興味が湧いたので取り組んでみたい。

## 10 実装したコード

### 10.1 image.h

```
#pragma once

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>

typedef struct _Image
{
    unsigned char *data;
    int W, H;
} Image;

typedef struct {
    double *data;
    int W,H;
} Matrix;

#define IElem(_im, _x, _y, _c) (_im)->data[_y * (_im)->W * 3 + (_x)*3 + (_c)]
#define ELEM(_a,_b,_c)  (_a)->data[_a->W*(_b)+(_c)]
#define DELEM(_a,_b,_c)  (_a)->data[_a->W*(_c)+(_b)]
#define Row(_a,_b)  ((_a)->data+(_a)->W*_b)
#define isInsideImage(is, u, v) (0 <= u && u < is->W && 0 <= v && v < is->H)
#define MAX 30
```

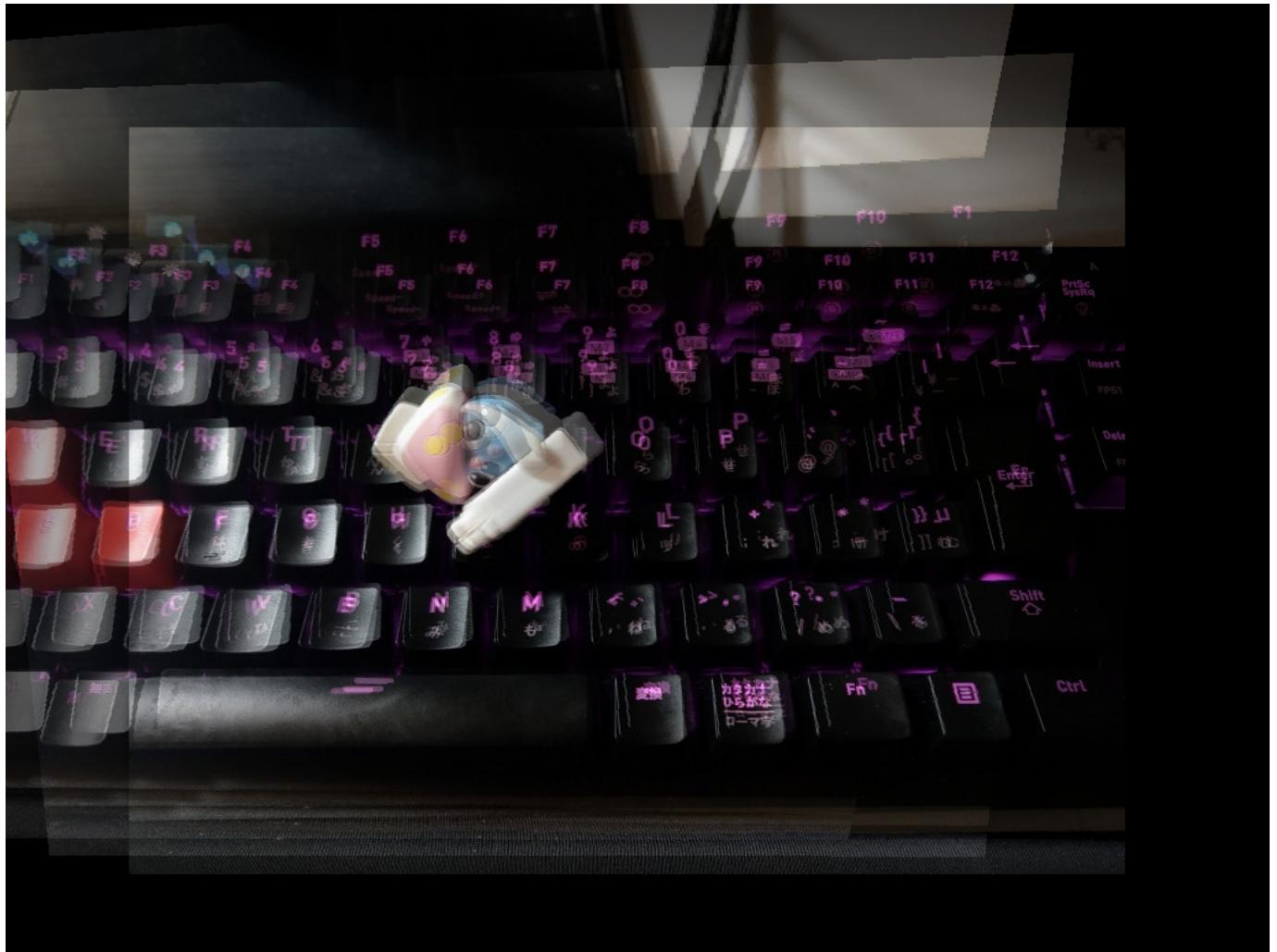


図 42: 自分での合成結果 1

```
#ifdef _WIN32
#include<intrin.h>
#else
#define __rdtsc() ({ long long a,d; __asm__ volatile ("rdtsc":=a"(a"),=d"(d)); d<<32|a; })
#endif

Image *ImageAlloc(int W, int H);
Image *ImageRead(const char *name);
void ImageFree(Image *im);
void ImageWrite(const char *name, Image *im);
void ImageClear(Image *im);
void ImageProjectionAlpha(Image *id, Image *is, double a[3][3], double alpha);
void ImageDrawBox(Image *im, int x, int y, int W);
void mult33(double d[3][3],double a[3][3],double b[3][3]);
Matrix *MatrixAlloc(int H,int _W);
double ImageSSD(Image*im,int x1,int y1, Image*im2,int x2,int y2,int W);
void calcSSDtable(Matrix*mt,Image*im ,int x1[] [3],int N1,Image*im2,int x2[] [3],int N2, int W);
int matchMethod1(double w[] [4],Matrix*mt,Image*im ,int x1[] [3],int N1,Image*im2,int x2[] [3],int N2);
int matchMethod2(double w[] [4],Matrix*mt,Image*im ,int x1[] [3],int N1,Image*im2,int x2[] [3],int N2);
void ImageFeature(Matrix*im2,Image*im,int W);
int MatrixLocalMax(int w[] [3], Matrix*im2, int W);
void MatrixSimeqLr(Matrix*mtB,Matrix*mtR);
void MatrixQRDecompColMajor(Matrix*mtR,Matrix*mt);
```

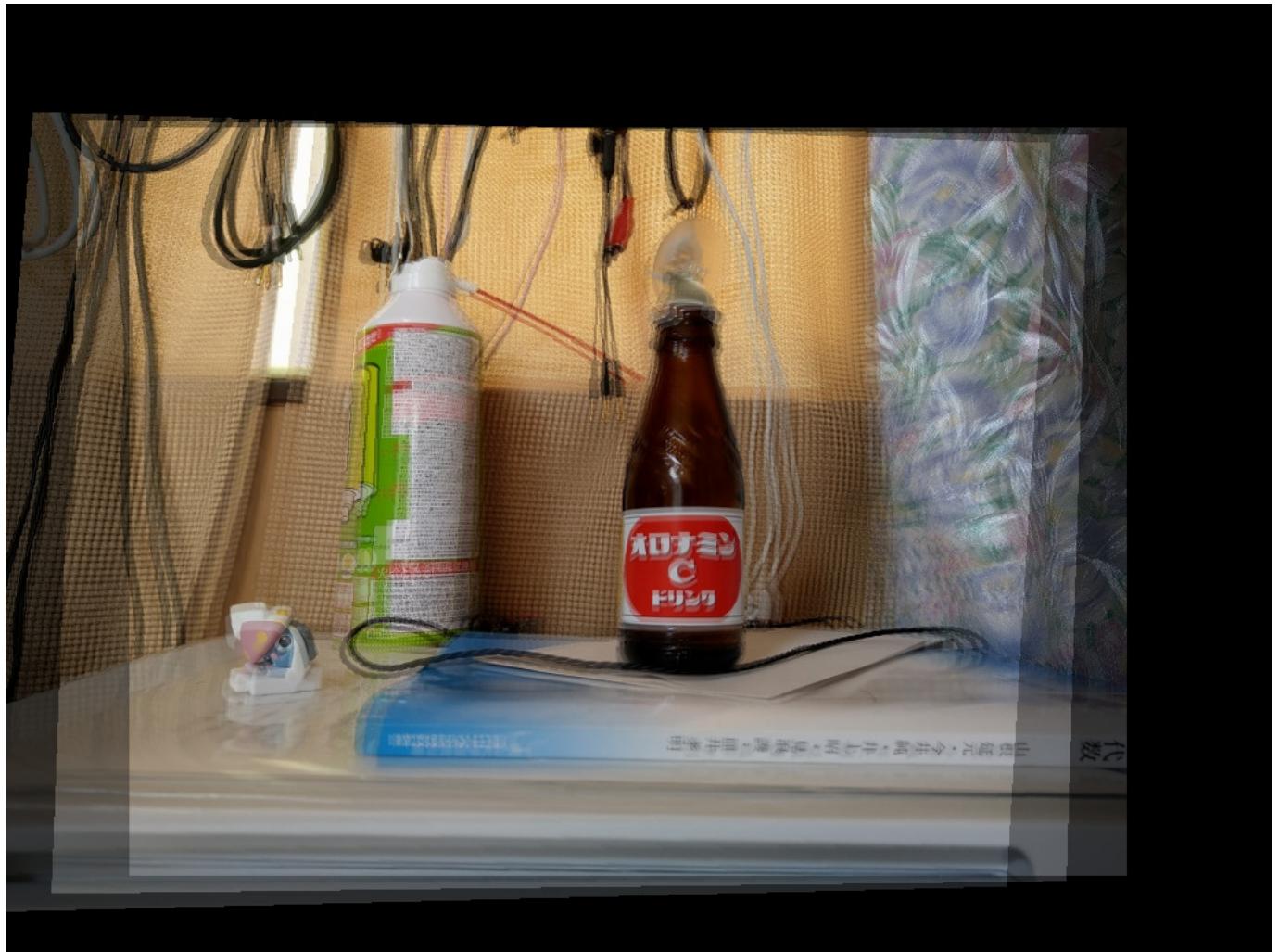


図 43: 自分での合成結果 2

```
void MatrixCopyT(Matrix*mtD,Matrix*mt);
void MatrixClear(Matrix*mt);
void MatrixCopy(Matrix*mtD,Matrix*mt);
void VSA(double*d,double*a,double s,int N);
void VSS(double*d,double s,int N);
double VP(double*a,double*b,int N);
```

## 10.2 image.c

```
#include "image.h"

union {int i; float f;} _inf={0x7f800000};
#define INFINITY _inf.f // 2重定義エラーが出るなら、この定義を削除
#define INFINITY_INT _inf.i

Image *ImageAlloc(int w, int h)
{
    Image *im = (Image *)malloc(sizeof(Image));
    im->W = w;
```

```

im->H = h;
im->data = (unsigned char *)malloc(3 * w * h);
return im;
}

void ImageClear(Image *im){
    for(int y=0;y<im->H;y++){
        for(int x=0;x<im->W;x++){
            IElem(im,x,y,0) = 0;
            IElem(im,x,y,1) = 0;
            IElem(im,x,y,2) = 0;
        }
    }
}

void mult33(double d[3][3],double a[3][3],double b[3][3]){
    for(int i=0;i<3;i++){
        for(int j=0;j<3;j++){
            d[i][j] = a[i][0]*b[0][j] + a[i][1]*b[1][j] + a[i][2]*b[2][j];
        }
    }
}

void ImageImageProjectionAlpha(Image*id,Image*is,double a[3][3],double alpha){
    int x,y,u,v;
    double r;
    for(y=0;y<id->H;y++) for(x=0;x<id->W;x++){
        r=1/(a[2][0]*x+a[2][1]*y+a[2][2]);
        u=r*(a[0][0]*x+a[0][1]*y+a[0][2]);
        v=r*(a[1][0]*x+a[1][1]*y+a[1][2]);
        if( isInsideImage(is,u,v ) ){
            IElem(id,x,y,0)+=IElem(is,u,v,0)*alpha,
            IElem(id,x,y,1)+=IElem(is,u,v,1)*alpha,
            IElem(id,x,y,2)+=IElem(is,u,v,2)*alpha;
        }
    }
}

Matrix *MatrixAlloc(int _H,int _W){
    Matrix*mt=(Matrix*)malloc(sizeof(Matrix));
    mt->W = _W;
    mt->H = _H;
    mt->data=(double*)malloc(mt->W*mt->H*sizeof(double));
    return mt;
}

Image *ImageRead(const char *name)
{
    int W, H;
    Image *im;
    char tmp[128];
    sprintf(tmp, "convert %s ppm:-", name);
    FILE *fp = popen(tmp, "r");
    if (fp == NULL)
    {
        fprintf(stderr, "could not read %s\n", name);
        return 0;
    }
    fscanf(fp, "%*s%d%d%s%c", &W, &H);
    //fprintf(stderr,"w x h=%d %d %d\n",W,H);
    im = ImageAlloc(W, H);
    fread(im->data, 1, W * H * 3, fp);
    pclose(fp);
    return im;
}

void ImageWrite(const char *name, Image *im)
{
    char tmp[128];
    sprintf(tmp, "convert ppm:- %s", name);
    FILE *fp = popen(tmp, "w");
    if (fp == NULL)
    {
        fprintf(stderr, "could not write %s\n", name);
        return;
    }
    fprintf(fp, "P6 %d %d 255\n", im->W, im->H);
    fwrite(im->data, 1, im->W * im->H * 3, fp);
    pclose(fp);
}

double ImageSSD(Image*im,int x1,int y1, Image*im2,int x2,int y2,int W){
    int i,j;
    double sr=0,sb=0,dr,dg,db;
    for(i=-W;i<=W;i++) for(j=-W;j<=W;j++){
        dr = IElem(im, x1+j, y1+i, 0) - IElem(im2, x2+j , y2+i, 0);

```

```

dg = IElem(im, x1+j, y1+i, 1) - IElem(im2, x2+j, y2+i, 1);
db = IElem(im, x1+j, y1+i, 2) - IElem(im2, x2+j, y2+i, 2);
sr += dr*dr;
sg += dg*dg;
sb += db*db;
}
return sr+sg+sb;
}

void calcSSDtable(Matrix*mt,Image*im,int x1[][3],int N1,Image*im2,int x2[][3],int N2,int W){
    int i,j;
    #pragma omp parallel for private (j)
    for(i=0;i<N1;i++){
        for(j=0;j<N2;j++){
            Elemt(mt,i,j) = ImageSSD(im,x1[i][0],x1[i][1],im2,x2[j][0],x2[j][1],W);
        }
    }
}

int matchMethod1(double w[][4],Matrix*mt,Image*im,int x1[][3],int N1,Image*im2,int x2[][3],int N2){
    int i,j,k,ji,n=0;

    for(i=0;i<N1;i++){
        double sm=INFINITY,t;
        for(j=0;j<N2;j++){
            t = Elemt(mt,i,j);
            if(sm>t) sm=t, ji=j;
        }
        //上の printf で表示されるものを w[n][] に格納。
        w[n][0] = x1[i][0];
        w[n][1] = x1[i][1];
        w[n][2] = x2[ji][0];
        w[n][3] = x2[ji][1];
        n++;
        for(k=0;k<N1;k++) { Elemt(mt,k,ji)=INFINITY; }
    }
    printf("%d\n",n);
    return n;
}

int matchMethod2(double w[][4],Matrix*mt,Image*im,int x1[][3],int N1,Image*im2,int x2[][3],int N2){
    int i,j,k,l,n = 0;
    int minElem[3] = {0,0,INFINITY_INT};
    // SSD の表中の最小値
    for(i=0;i<MAX;i++){//点の数だけ
        for(j=0;j<n->W;j++){//横幅
            for(k=0;k<mt->H;k++){//縦幅
                if(Elemt(mt,j,k) < minElem[2]){
                    minElem[0] = j;
                    minElem[1] = k;
                    minElem[2] = (int)Elemt(mt,j,k);
                    //printf("%d,%d,%d\n",minElem[0],minElem[1],minElem[2]);
                }
            }
        }
        for(k=0;k<MAX;k++) Elemt(mt,minElem[0],k)=INFINITY_INT;//横固定して爆破
        for(k=0;k<MAX;k++) Elemt(mt,k,minElem[1])=INFINITY_INT;//縦固定して爆破
        //MatrixPrint(mt);
        //printf("%d,%d,%d,%d,\n",x1[minElem[0]][0],x1[minElem[0]][1],x2[minElem[1]][0],x2[minElem[1]][1]);
        w[n][0] = x1[minElem[0]][0];
        w[n][1] = x1[minElem[0]][1];
        w[n][2] = x2[minElem[1]][0];
        w[n][3] = x2[minElem[1]][1];
        n++;
        minElem[2]=INFINITY_INT;
    }
    return n;
}

void ImageFeature(Matrix*im2,Image*im,int W){//特徴点らしさ
    int x,y,u,v,ix,iy;
    for(y=W+1;y<im->H-W-1;y++) for(x=W+1;x<im->W-W-1;x++){
        double ixx,ixy,iyy;
        ixx=iy=ixy=0;
        for(v=-W;v<=W;v++)for(u=-W;u<=W;u++){
            ix = IElem(im, x+u+1, y+v, 1) - IElem(im, x+u-1, y+v, 1);
            iy = IElem(im, x+u, y+v+1, 1) - IElem(im, x+u, y+v-1, 1);
            ixx += ix*ix; // ixx だけでなく ixy, iyy も計算する。
            iyy += iy*iy;
            ixy += ixy*ixy;
        }
        DElem(im2,x,y) = ((ixx + iyy)-sqrt(pow(ixx+iyy,2.0)-4.0*(ixx*iyy-ixy*ixy)))/2.0;
    }
}

```

```

int MatrixLocalMax(int w[][3], Matrix*im2, int W){//特徴点がほんとにそうかどうか
    int x,y,u,v,n=0,a;
    for(y=W+1;y<im2->H-W-1;y++) for(x=W+1;x<im2->W-W-1;x++){
        double max=-1;
        for(v=-W,v<=W,v++) for(u=-W;u<=W;u++){
            if(DElem(im2, x+u, y+v) > max){
                max = DElem(im2,x+u,y+v);
            }
        }
        if( max == DElem(im2, x, y ) ){
            a = n;
            if(a < MAX) { n++; }
            for(;a>0 && w[a-1][2] < DElem(im2, x, y);a--){
                w[a][0]=w[a-1][0]; w[a][1]=w[a-1][1]; w[a][2]=w[a-1][2];
            }
            w[a][0]=x; w[a][1]=y; w[a][2]=max;
        }
    }
    return n; // 記録した点の数
}
double VP(double*a,double*b,int N){
    double s=0;
    int i;
    for(i=0;i<N;i++) s += a[i] * b[i] ;
    return s;
}
void VSS(double*d,double s,int N){
    int i;
    for(i=0;i<N;i++) d[i] *= s;
}
void VSA(double*d,double*a,double s,int N){
    int i;
    for(i=0;i<N;i++) d[i] += a[i] * s;
}
void MatrixClear(Matrix*mt){
    memset(mt->data,0,mt->W*mt->H*sizeof(double));
}

void MatrixCopy(Matrix*mtD,Matrix*mt){
    memmove(mtD->data,mt->data,mt->W*mt->H*sizeof(double));
}

void MatrixCopyT(Matrix*mtD,Matrix*mt){
    int i,j;
    for(i=0;i<mtD->H;i++)
        for(j=0;j<mtD->W;j++)
            ElelmtD,i,j) = Elelmt, j,i);
}

void MatrixMultT(Matrix*mtD,Matrix*mtA,Matrix*mtB){
    // D = A B^T
    int i,j;
    for(i=0;i<mtA->H;i++){
        for(j=0;j<mtB->H;j++){
            ElelmtD,i,j) = VP( Row(mtA,i), Row(mtB,j), mtA->W);
        }
    }
}

void MatrixQRDecompColMajor(Matrix*mtR,Matrix*mt){
    double t;
    int W = mt->W;
    double **aT = (double **)malloc((sizeof (double *)) * mt->H);
    for(int get=0; get<mt->H;get++){
        aT[get] = Row(mt, get);
    }
    MatrixClear(mtR);
    for(int i=0; i < W; i++){
        for(int j=0; j <= i; j++){
            if(j == i){
                ElelmtR,j,i) = t = sqrt(VP(aT[j],aT[i],W));
                VSS(aT[i], 1/t, W);
            }else{
                ElelmtR,j,i) = t = VP(aT[j], aT[i], W);
                VSA(aT[i], aT[j], -t, W);
            }
        }
    }
}

void MatrixSimeqLr(Matrix*mtB,Matrix*mtR){
    // B = B L^{-1}
    double * B = Row(mtB,0);
    for(int i=mtR->W-1; 0<=i; i--){

```

```

        for(int j=i+1; j<mtR->W; j++){
            B[i] -= B[j]*Elem(mtR,i,j);
        }
        B[i] = B[i] / Elem(mtR,i,i);
    }
}

10.3 all.c

#include "image.h"

#define TRIAL 1000000

void initRndAry(int rndAry[MAX]){
    for(int i = 0; i<MAX; i++){
        rndAry[i] = i;
    }
    srand(_-_rdtsc()); // _-_rdtsc については第4回を参照.
}

void chooseFourNumbers(int rndAry[MAX]){
    for(int i=0;i<4;i++){
        int j, t;
        j = (int)((long long)rand()*(MAX-i)/(RAND_MAX+1LL))+i; // 亂数関数は stdlib.h で宣言されている.
        t = rndAry[i];
        rndAry[i] = rndAry[j];
        rndAry[j] = t;
    }
}

void calcHomography(double H[3][3],double w[][4],int rndAry[MAX], Matrix *cmA, Matrix *vt, Matrix *mtR, Matrix *tmp){
    int a = rndAry[0], b = rndAry[1], c = rndAry[2], d = rndAry[3];
    double ww[] [4] = {
        w[a][0], w[a][1], w[a][2], w[a][3],
        w[b][0], w[b][1], w[b][2], w[b][3],
        w[c][0], w[c][1], w[c][2], w[c][3],
        w[d][0], w[d][1], w[d][2], w[d][3],
    };
    // create A (col-major)
    for(int i=0;i<4;i++){
        Elem(cmA,0,i*2 )=ww[i][0];
        Elem(cmA,1,i*2 )=ww[i][1];
        Elem(cmA,2,i*2 )=1;
        Elem(cmA,3,i*2 )=0;
        Elem(cmA,4,i*2 )=0;
        Elem(cmA,5,i*2 )=0;
        Elem(cmA,6,i*2 )=-ww[i][0]*ww[i][2];
        Elem(cmA,7,i*2 )=-ww[i][1]*ww[i][2];
        Elem(cmA,0,i*2+1)=0;
        Elem(cmA,1,i*2+1)=0;
        Elem(cmA,2,i*2+1)=0;
        Elem(cmA,3,i*2+1)=ww[i][0];
        Elem(cmA,4,i*2+1)=ww[i][1];
        Elem(cmA,5,i*2+1)=1;
        Elem(cmA,6,i*2+1)=-ww[i][0]*ww[i][3];
        Elem(cmA,7,i*2+1)=-ww[i][1]*ww[i][3];
        Elem(vt ,0,i*2 )=ww[i][2];
        Elem(vt ,0,i*2+1)=ww[i][3];
    }
    MatrixQRDecompColMajor(mtR,cmA);
    MatrixMultT(tmp,vt,cmA);
    MatrixSimeqlr(tmp,mtR);

    H[0][0] = Elem(tmp,0,0);
    H[0][1] = Elem(tmp,0,1);
    H[0][2] = Elem(tmp,0,2);
    H[1][0] = Elem(tmp,0,3);
    H[1][1] = Elem(tmp,0,4);
    H[1][2] = Elem(tmp,0,5);
    H[2][0] = Elem(tmp,0,6);
    H[2][1] = Elem(tmp,0,7);
    H[2][2] = 1;
}

int calcScore(double H[3][3], double w[][4], int limit){
    int score=0;
    for(int i=0;i<MAX;i++){
        double x=w[i][0], y=w[i][1], u=w[i][2], v=w[i][3],
        x_prime = H[0][0] * x + H[0][1] * y + H[0][2],
        y_prime = H[1][0] * x + H[1][1] * y + H[1][2],
        z_prime = H[2][0] * x + H[2][1] * y + H[2][2], // 変換行列と (x,y,1)^T の積
    }
}

```

```

        du = (x_prime/z_prime) - u,
        dv = (y_prime/z_prime) - v; // (x,y) を変換した座標(第2回の概要を参照)と(u,v)の差
        if(du*du+dv*dv <= limit) score++; // w[i] は正しい;
        else score += 0; // w[i] は正しくない;
    }
    return score;
}

// void createPanorama(Matrix *id, Image *im, Image *im2, int bestH[3][3]){
//     double m0d[3][3]={
//         1,0,-100,
//         0,1,-100,
//         0,0,1
//     },m1d[3][3];
//     ImageImageProjectionAlpha(id,im,m0d,.5);
//     mult33(m1d,bestH,m0d);
//     ImageImageProjectionAlpha(id,im2,m1d,.5);
//     ImageWrite("panorama.jpg",id);
// }

int main(int ac,char**av){
    Image *im, *im2, *im3;
    Matrix *imf, *mp, *id;
    int N1, N2, N3, nm;
    double w[999][4];
    int x1[MAX+1][3], x2[MAX+1][3], x3[MAX+1][3];

    double m0d[3][3]={
        1,0,-100,
        0,1,-100,
        0,0,1
    },m1d[3][3],m2d[3][3];

    // 多数の変換行列から最も信頼度の高い行列を選ぶ(RANSAC)
    double bestH[3][3];
    int best_score = 0; // 既に発見された最良の変換行列と得点
    int rndAry[MAX];
    Matrix *cmA, *vt, *mtR, *tmp; // 作業領域をここで確保して calcHomography で使う

    /*av[1] = W の値、それ以降は画像ファイル名*/
    im = ImageRead(av[1]);
    printf("read1\n");

    im2 = ImageRead(av[2]);
    printf("read2\n");

    imf = MatrixAlloc(im->H, im->W);

    ImageFeature(imf, im, atoi(av[5]));
    N1 = MatrixLocalMax(x1, imf, atoi(av[5]));
    printf("N1:%d\n",N1);

    ImageFeature(imf, im2, atoi(av[5]));
    N2 = MatrixLocalMax(x2, imf, atoi(av[5]));
    printf("N2:%d\n",N2);

    mt = MatrixAlloc(N1, N2);
    printf("mt\n");

    calcSSDtable(mt, im, x1, N1, im2, x2, N2, atoi(av[5]));
    printf("calcSSDtable\n");

    nm = matchMethod2(w, mt, im, x1, N1, im2, x2, N2);
    printf("nm:%d\n",nm);

    // 多数の変換行列から最も信頼度の高い行列を選ぶ(RANSAC)
    cmA = MatrixAlloc(8,8);
    vt = MatrixAlloc(1,8);
    mtR = MatrixAlloc(8,8);
    tmp = MatrixAlloc(1,8);
    printf("Alloc:cmA,vt,mtR,tmp\n");

    initRndAry(rndAry);
    printf("init andary\n");

    for(int trial = 0; trial<TRIAL; trial++){
        double H[3][3]; // 選んだ4点から計算される変換行列の置き場
        chooseFourNumbers(rndAry);
        calcHomography(H,w,rndAry,cmA,vt,mtR,tmp);
        int score = calcScore(H,w,atoi(av[4]));

        if(best_score < score){
            bestH[0][0] = H[0][0];
            bestH[0][1] = H[0][1];
        }
    }
}

```

```

        bestH[0][2] = H[0][2];
        bestH[1][0] = H[1][0];
        bestH[1][1] = H[1][1];
        bestH[1][2] = H[1][2];
        bestH[2][0] = H[2][0];
        bestH[2][1] = H[2][1];
        bestH[2][2] = H[2][2];
        best_score = score;
    }
}
printf("best_score:%d\n",best_score);

id = ImageAlloc(1024,768);
ImageClear(id);

ImageImageProjectionAlpha(id,im,m0d,.333);
mult33(mid,bestH,m0d);
ImageImageProjectionAlpha(id,im2,m1d,.333);

// 2 と 3
im3 = ImageRead(av[3]);
printf("read3\n");

ImageFeature(imf, im3, atoi(av[5]));
N3 = MatrixLocalMax(x3, imf, atoi(av[5]));
printf("N3:%d\n",N3);

mt = MatrixAlloc(N2, N3);
printf("mt\n");

calcSSDtable(mt, im2, x2, N2, im3, x3, N3, atoi(av[5]));
printf("calcSSDtable\n");

nm = matchMethod2(w, mt, im2, x2, N2, im3, x3, N3);
printf("nm:%d\n",nm);

cmA = MatrixAlloc(8,8);
vt = MatrixAlloc(1,8);
mtR = MatrixAlloc(8,8);
tmp = MatrixAlloc(1,8);
printf("Alloc:cmA,vt,mtR,tmp\n");

initRndAry(rndAry);
printf("init andary\n");

for(int trial = 0; trial<TRIAL; trial++){
    double H[3][3]; // 選んだ 4 点から計算される変換行列の置き場
    chooseFourNumbers(rndAry);
    calcHomography(H,w,rndAry,cmA,vt,mtR,tmp);
    int score = calcScore(H,w,atoi(av[4]));

    if(best_score < score){
        bestH[0][0] = H[0][0];
        bestH[0][1] = H[0][1];
        bestH[0][2] = H[0][2];
        bestH[1][0] = H[1][0];
        bestH[1][1] = H[1][1];
        bestH[1][2] = H[1][2];
        bestH[2][0] = H[2][0];
        bestH[2][1] = H[2][1];
        bestH[2][2] = H[2][2];
        best_score = score;
    }
}
printf("best_score:%d\n",best_score);

mult33(m2d,bestH,mid);
ImageImageProjectionAlpha(id,im3,m2d,.333);

ImageWrite("panorama.jpg",id);
printf("created\n");

return 0;
}

```