図形的特徴に基いて断片を再構成するアルゴリズムの再現

島根大学　総合理工学部　数理・情報システム学科

計算機科学講座　田中研究室

S103033　河原寛之

目次

第１章　はじめに

1989 年ベルリンの壁崩壊後、旧ドイツのシュタージ職員（東ドイツの秘密警察・諜報機関）は、自身らのスパイ活動の証拠となる記録を抹消・隠滅するために、大量の文書をシュレッダーにかける、もしくは手で破るという作業を数週間にわたり行った[1]。

これらの、細かく刻まれたおびただしい数の紙片は袋にいれられたまま、現在の連邦政府により保管され、修復・復元作業が続けられてきた。

しかし、これらの文書を人力によってつなぎ合わせる復旧作業方法では完成までに何世紀もかかると予想された。そこで「E-puzzler」と呼ばれる一連のシステムが導入された。

本研究では図形的特徴に基づき、再構成する手法を実装していく。

第 2 章　E-puzzler とは

E-puzzler とは Fraunhofer IPK によって開発された、引き裂かれた紙片をスキャンし、複雑な画像処理とパターン認識アルゴリズムを用いて完全な 1 枚の紙に復元するためのシステムである[1]。

E-puzzler は次の 3 つの大きな要素から成り立っている。

1. 特徴抽出

紙片の形状、用紙の色、フォント、罫線などさまざまな紙片の特徴を計算する。

2. 検索領域の削減

大量のデータを扱う場合、パズルの可能な組み合わせ数を減少させるために、同じような属性を持つ紙片のグループに分けられる。

3. マッチング

断片の類似性を縁に沿って比較し、断面を繋ぎ合わせていく。

第 3 章　実装方法

本研究では図形的特徴に基いて行うので、「断面の簡略化」、「断面の図形的特徴の抽出」、「各断面の類似性の評価」、「マッチング」の4つのセクションに分けられる[2]。今回与えられる断片については手動の入力としている。

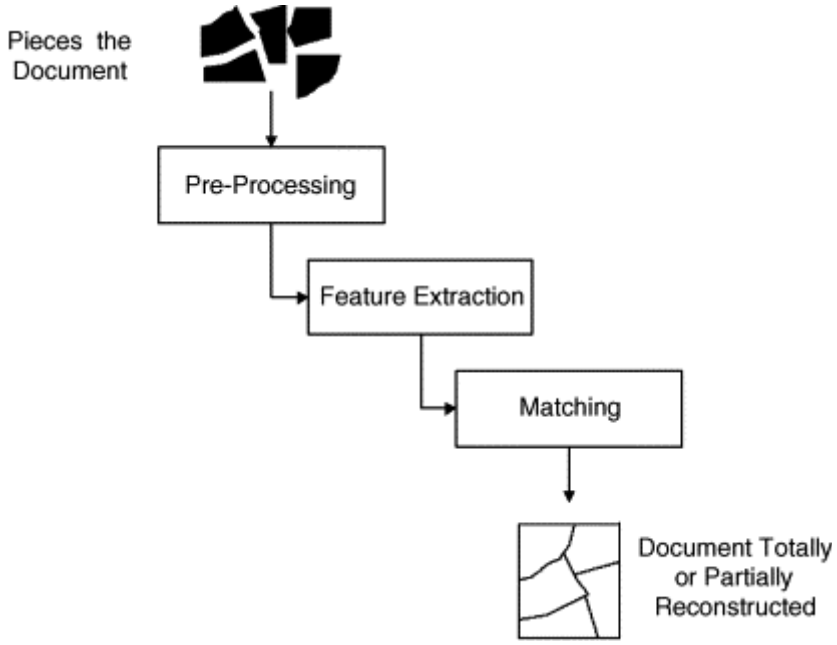

図 1　アルゴリズムの流れ[2]出典

3.1 断面の簡略化

実際に紙を手で破いた場合、その断面は不規則な境界を生成する。これらを扱うのは非常に困難であるため、断面の簡略化を行う必要がある。断面の簡略化には Douglas–Peucker のアルゴリズムを用いる[3]。

Douglas–Peucker のアルゴリズムとは、与えられた曲線を一連の点で近似し、近似した曲線の点の数を減少させるアルゴリズムである。このアルゴリズムはコンピュータグラフィックや地図上に

おける境界線データに対する描画点数低減などにしばしば用いられる。

Douglas–Peucker のアルゴリズムは以下の 4 つのセクションから成る。

1. 折れ線の始点と終点を結ぶ線分と各点の距離を求める。
2. すべての点との距離が許容誤差 ε 以内に入っていれば始点と終点だけを返して終了。
3. そうでなければ距離が最大の点 P を 1 つ選択する。
4. 始点から点 P までの折れ線と、点 P から終点までの折れ線のそれぞれについてまた 1 から処理を繰り返す。

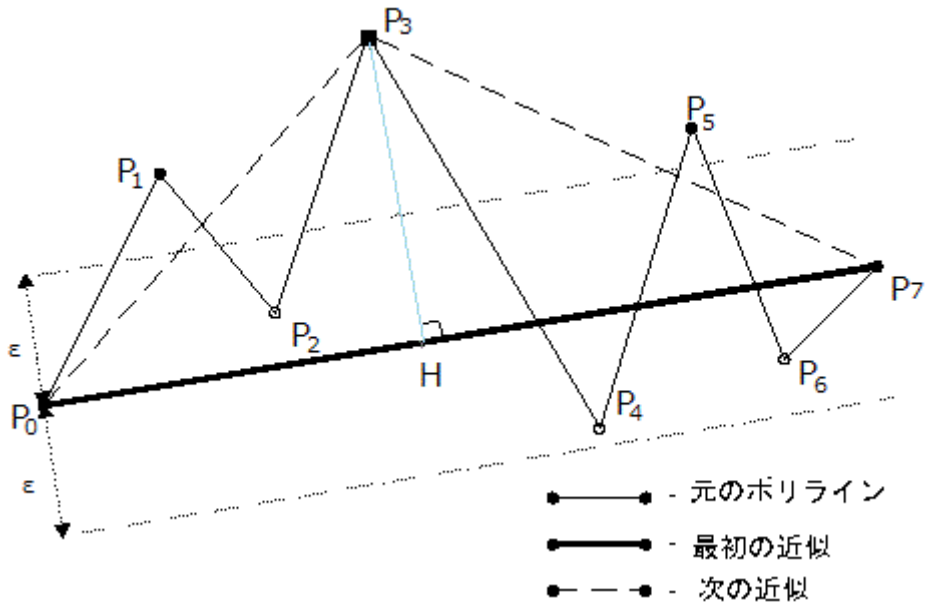

図2 Douglas–Peucker のアルゴリズム

図2のようにまず P0 と P7 を取り、線分 P0P7 に対して各点から垂線を引き、最も遠い点を選ぶ。この場合 P3 が最も遠いので P3 を選ぶ。P3 は許容誤差 ε 以内に入っていないのでこの点を取る。

次に線分 P0P3、P3P7 を取り、以上の操作をすべての点が許容誤差以内になるまで繰り返す。

```
・・・・・略・・・・・
for (i = 0; i < mi; i++) {
      if (ax < px[f][i] && px[f][i] < bx) {
      /* 2点ABの間に点pが存在するかどうかの判定 */
            k = 1;
      /* 存在する場合 */
            mi = DP(ax, bx, ay, by, px, py, q, mi);
```

```
                break;
        }
}
if (k == 0) {
        /* 存在しない場合は次の点を取る */
        ax = bx;
        ay = by;
        ni = mi;
        for (i = mi + 1; i < id[f]; i++) {
                if (q[i] == 1) {
                        bx = px[f][i];
                        by = py[f][i];
                        mi = i;
                        break;
                }
        }
}

/* 点が存在するならばその中で最大のものを求め、次点を決める */
private static int DP(double ax, double bx, double ay, double by, double[][]
px2, double[][] py2, int[] q2, int mi2) {
        int i, mi3 = 0;
        double d, dmax = 0;
        for (i = 0; i < mi2; i++) {
        /* まだ距離を求めていない点のみを取る */
                if (q2[i] == 0) {
                        /* 距離を求める */
                        d = line(ax, ay, bx, by, px2[f][i], py2[f][i]);
                        /* 誤差判定 */
                        if (dmax < d && d > 5.0 ) {
                        /* 最も遠い点を取る */
                                dmax = d;
                                mi3 = i;
                        }
                        if (bx == px2[f][i]) {
                                break;
                        }
                }
        }
        q2[mi3] = 1;
        bx = px2[f][mi3];
        by = py2[f][mi3];
        return mi3;

}

public static double line(double ax, double ay, double bx, double by,
double px, double py) {
        double dx, dy, r;
        double t, cx, cy;
        // 線分ABのx座標とy座標の差
        dx = bx - ax;
        dy = by - ay;
        // 線分と垂線のベクトルの内積 (dx,dy)・(ax+dx*t-px,ay+dy*t-py) より
        r = dx * dx + dy * dy;
```

```
      t = (dx * (px - ax) + dy * (py - ay)) / (double) r;
      if (dx == 0 && dy == 0) { // A=Bの時
            return Math.sqrt((px - ax) * (px - ax) + (py - ay) * (py -
ay));
      } else {
            // 垂線の足(cx,xy)
            cx = (1 - t) * ax + t * bx;
            cy = (1 - t) * ay + t * by;
            return Math.sqrt((px - cx) * (px - cx) + (py - cy) * (py -
cy));
      }
}
```

リスト1　**Douglas–Peucker** のアルゴリズム

3.2 特徴抽出

3.1 の Douglas–Peucker のアルゴリズムによって簡略化された断面における頂点の角度と辺の長さをそれぞれ求める。各頂点は座標として与えられているので α はベクトルの内積より u･v=|u||v|cosα で与えられる。

```
private static void vector(double[][] px2, double[][] py2, int[] id int f) {
	double pro, sita, rad;
	double[] vda = new double[100], vdb = new double[100];
	double[] cos_sita = new double[100];
	int ax1, ay1, bx1, by1, i, j = f;
	while (j != 0){
		for (i = 0; i + 1 < = id[f]; i++) {
			if(i==0){
				ax1 = px2[f][i] - px2[f][id];
				ay1 = py2[f][i] - py2[f][id];
			}else{
				ax1 = px2[f][i] - px2[f][i - 1];
				ay1 = py2[f][i] - py2[f][i - 1];
			}
			bx1 = px2[f][i + 1] - px2[f][i];
			by1 = py2[f][i + 1] - py2[f][i];

			/* 辺の距離を求める */
			if(i==0){
				vda[i] = vdb[i - 1];
			}else{
				vda[i] = Math.sqrt((ax1 * ax1) + (ay1 * ay1));
			}

			vdb[i] = Math.sqrt((bx1 * bx1) + (by1 * by1));

			/* 内積 */
			pro = ax1 * bx1 + ay1 * by1;

			/* 角度 */
			cos_sita = pro / (vda[i] * vdb[i]);
			rad = Math.acos(cos_sita);
			sita[i] = 180 - (rad * 180.0 / Math.PI);

		}
	}
		eva(vda,vdb,sita,f,id);
}
```

リスト2　特徴抽出のアルゴリズム

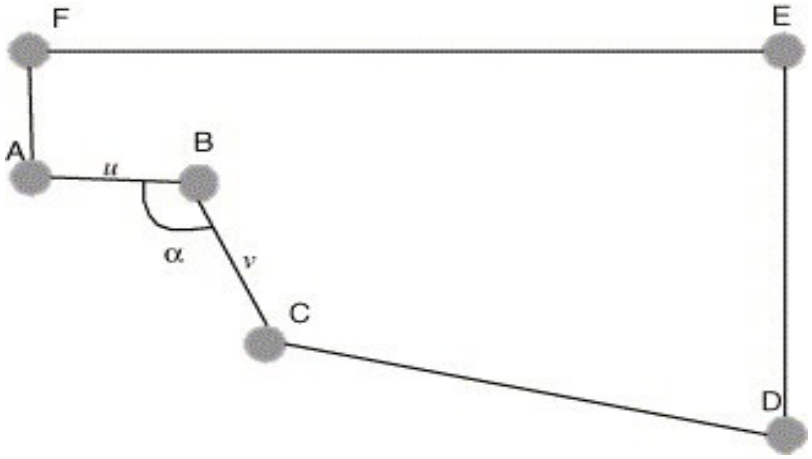

図3　特徴抽出

表1　図3における各頂点の特徴

頂点	角度(°　)	距離		座標	
		次	前	X	Y
A	270	40	45	10	70
B	120	45	43.6	55	67
C	200	43.6	155.7	67	25
D	245	115.7	11	180	0
E	270	110	170	180	110
F	270	170	40	10	110

3.3 各断面の類似性の評価

ある2つの断面において図4のように頂点Aで重ねた場合、その両端の辺の一致具合を評価する。

- 両辺の長さが一致していれば+5
- どちらかの長さが一致していれば+1
- 一致した部分の長さが断片の外周の長さの1/5以上ならば+2
- 一致した部分の長さが断片の外周の長さの1/10以上ならば+1
- それ以外の場合、評価の値は増加しない

```
・・・・・略・・・・・

/* 両辺の長さが一致しているか */
if (dp1 - dp2 == 0 && dn1 - dn2 == 0)
      Wm[k] += 5;

/* どちらかが一致 */

if (dp1 - dp2 == 0 || dn1 - dn2 == 0)
      Wm[k] += 1;

/* 一致部分の長さが断片の周囲の長さに対して1/5以上の時 */
if ((dp2 + dn2) / pr[pa] > 0.2)
      Wm[k] += 2;

/* 一致部分の長さが断片の周囲の長さに対して1/10以上の時 */
if ((dp2 + dn2) / pr[pa] > 0.1)
      Wm[k] += 1;


      /* 最も評価の高い点を取る */
if (max > Wm[k]) {
      max = Wm[k];
      ip = i;
      jp = j;
}
k++;
            }
      }
```

リスト3　各断面の評価のアルゴリズム

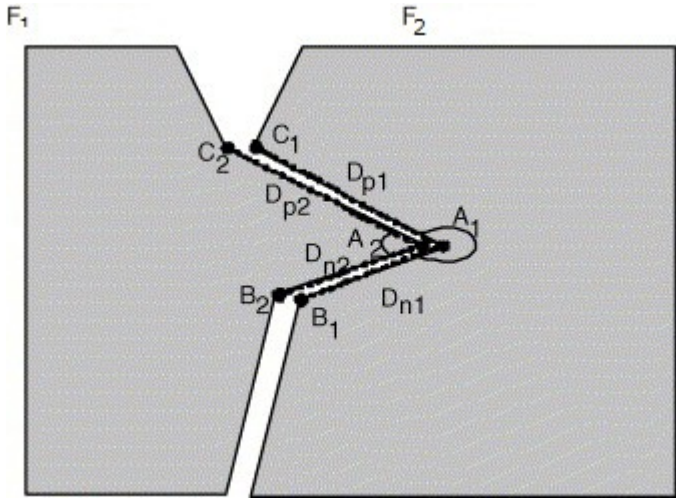

図4　頂点 A における評価

図4において、頂点 A1 と A2 を重ねた時 Dn1 と Dn2、Dp1 と Dp2 の長さの一致具合を求める。さらに、一致した長さが断片の外周に占める割合を求め、評価していく。

この操作を各断面、各頂点に総当りで行う。

3.4 マッチング

3.3 で求めた評価の値に基づいて、評価の高い組み合わせからマッチングしていく。
マッチングは以下の流れで行われる。

- 評価の最も高い点（A とする）を選ぶ
- 選んだ頂点Aを重ねる
- 動かした断片のそれぞれの頂点を平行移動し、頂点A周りに回転させる
- 他の頂点同士で重なる点があればその頂点を消滅させる

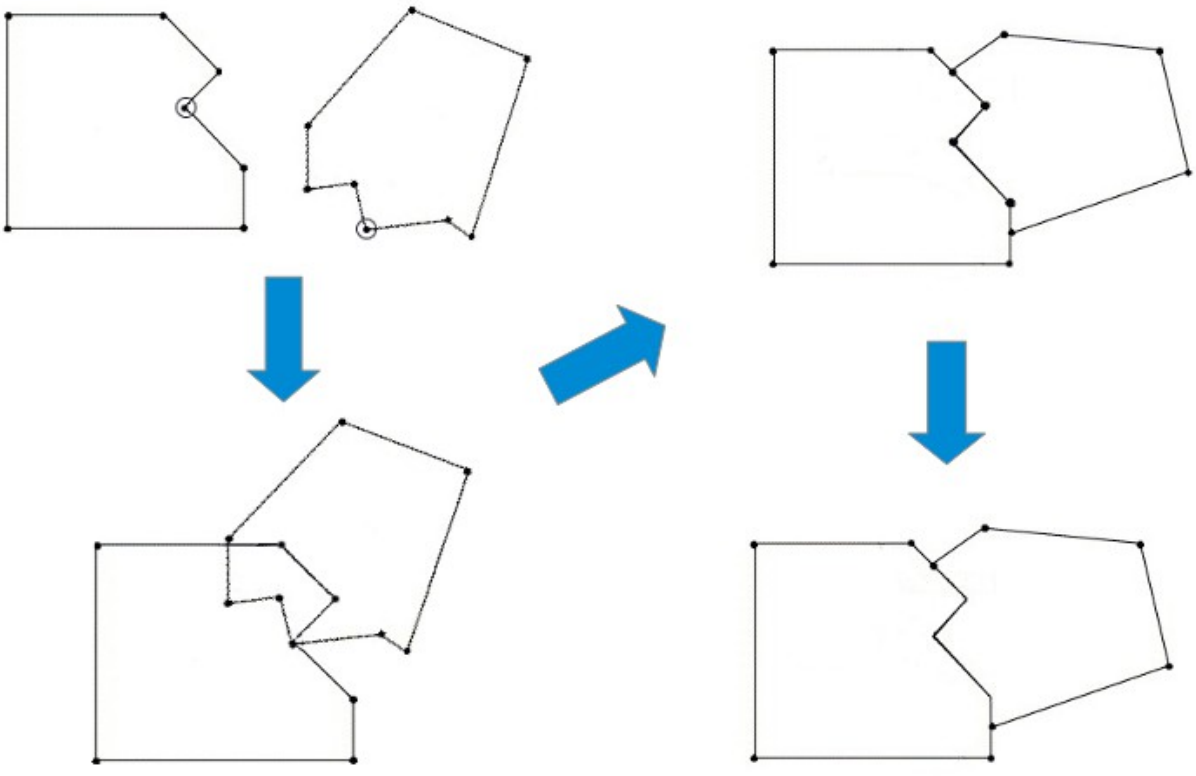

図5　マッチング

```
private static void matchi(double[][] px2, double[][] py2, int ip, int jp,
      int pa, int pb, int[] id, double dn1, double dp1) {
      double gx, gy;
      int i, j;

      /* 評価最高点の座標を重ねるように平行移動 */
      gx = px2[pb][jp] - px2[pa][ip];
      gy = py2[pb][jp] - py2[pa][ip];
      /* 他の点も移動 */
      for (j = 0; j < id[pb]; j++) {
            px2[pb][j] -= gx;
            py2[pb][j] -= gy;
      }

      /* 両辺が一致するように回転させる */
      double xx, yy;
      double ax1, ay1, bx1, by1;
      double pro, cos, sin;
      ax1 = px2[pa][ip] - px2[pb][jp];
      ay1 = py2[pa][ip] - py2[pb][jp];
      bx1 = px2[pa][ip + 1] - px2[pb][jp + 1];
      by1 = py2[pa][ip + 1] - py2[pb][jp + 1];

      pro = ax1 * bx1 + ay1 * by1;
      cos = pro / (dn1 * dp1);
      sin = Math.sqrt(1 - (cos * cos));
            /* 点の周りの回転 */
      for (j = 0; j < id[pb]; j++) {
            xx = px2[pb][j] - px2[pa][ip];
            yy = py2[pb][j] - py2[pa][ip];

            px2[pb][j] = xx * cos - yy * sin;
            py2[pb][j] = xx * sin + yy * cos;

      }


            /* 頂点の消滅 */
      for (i = 0; i < id[pa]; i++) {
            for (j = 0; j < id[pb]; j++) {
                  if (px2[pb][j] == px2[pa][i] && py2[pb][j] == py2[pa][i]) {
                        px2[pb][j] = -1;
                        px2[pa][i] = -1;
                        py2[pb][j] = -1;
                        py2[pa][i] = -1;
                  }
            }
      }
```

リスト4　マッチングのアルゴリズム

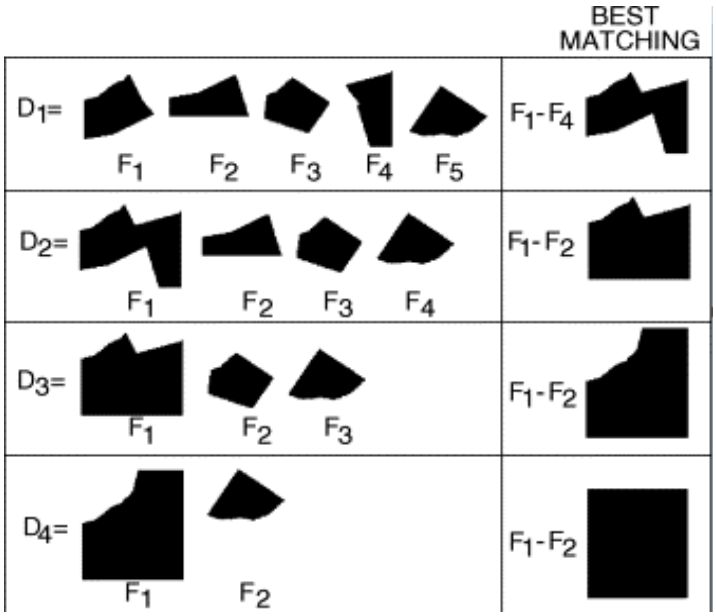

図6　断片の再構成の流れ[2]出典

複数の断片を扱う場合、新たに生成された断片に対しても同様の操作を繰り返しマッチングを行っていく。断片のマッチングが不可能になったとき終了する。

第 4 章　再現結果

本研究では断片の形状は手動での入力ではあるが断片の再構成を行うことが出来た。
以下がその結果である。

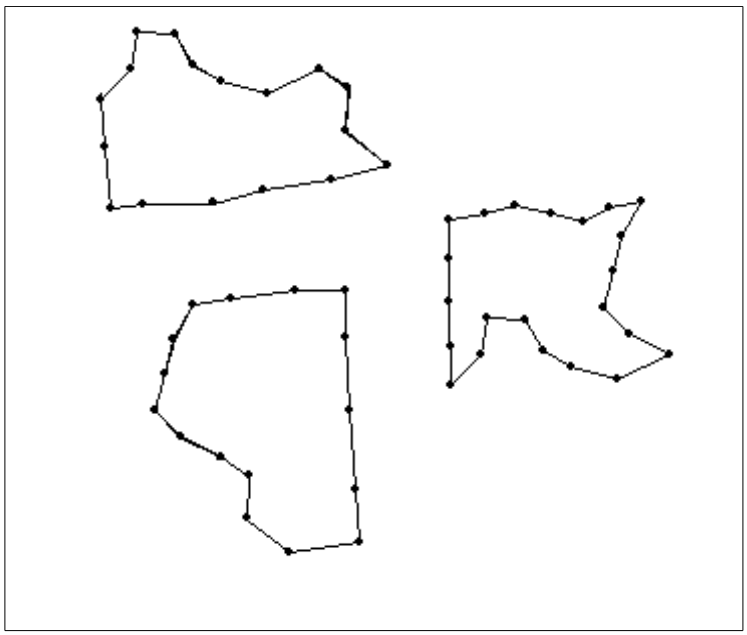

図 7　入力された断片

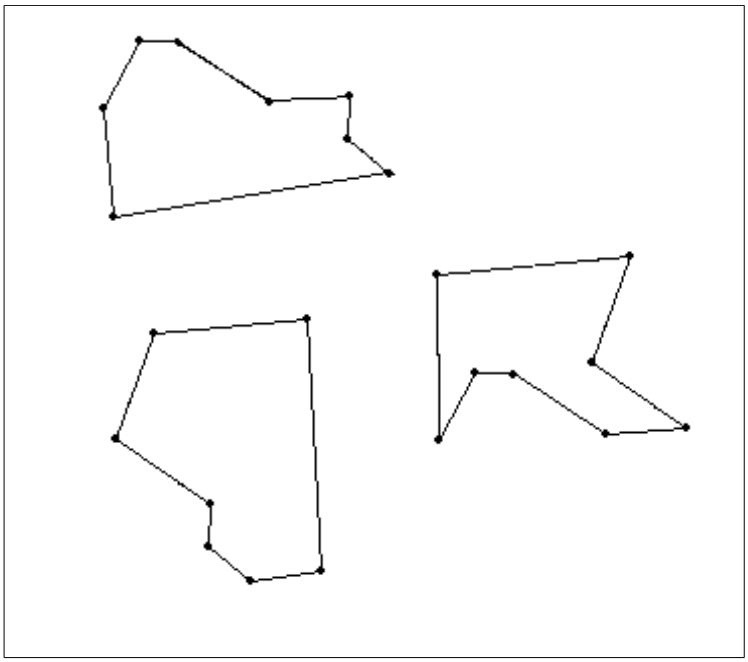

図 8　簡略化された断片

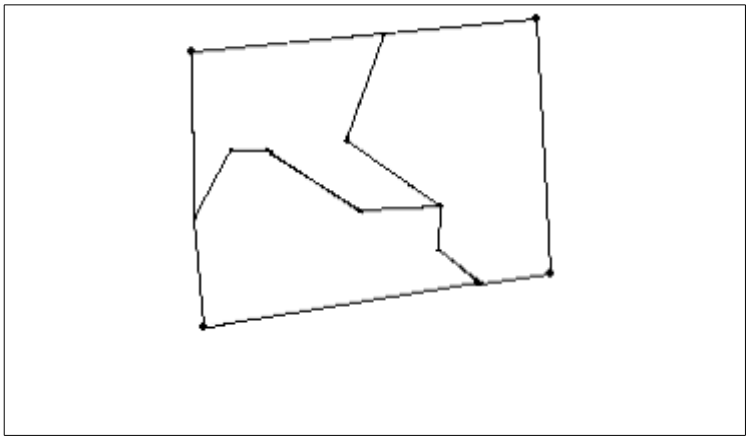

図 9　再構成された断片

第５章　今後の課題

本研究で作成した再構成をアルゴリズムは断片に図形的特徴のみに基いて作成してあるため、色、フォント、罫線などの特徴には対応していない。

また、断片の形状を手動で与えているので、実際の紙片にも対応した再構成アルゴリズムの作成が今後の課題として挙げられる。

謝辞

本研究を進めるにあたり、最後まで熱心に御指導して頂きました田中章司郎教授には心より御礼申し上げます。

同研究室の皆様にも、数々の御協力と御助言を頂きましたこと、厚く御礼申し上げます。

なお、本論文、本研究で作成したプログラム及び、データ、並びに関連する発表資料等の全ての知的財産権を本研究の指導教官の田中章司郎教授に譲渡致します。

引用･参考文献

[1]Fraunhofer IPK “Automatisierte virtuelle Rekonstruktion der zerrissenen Stasi-Akten,”<http://www.ipk.fraunhofer.de/> ,2007

[2]Edson Justino, Luiz S. Oliveira “Reconstructing shredded documents through feature matching,” Forensic Science International . ,pp140-147 ,2006

[3]D. Douglas, T. Peucker “Algorithms for the reduction of the number of points required to repres ent a digitized line or its caricature,” Can.Cartogr. , pp. 112-122 ,1973

[4]H.C.G. Leitao, J. Stolfi “A multiscale method for the reassembly of two-dimensional fragmented objects,” IEEE Trans.Pattern Anal.Mach.Intel. , pp. 1239-1251 ,2002