地図システムのマルチポイントラバーシーティング

2R－9

鈴鹿 豊明
日立ソフトウェアエンジニアリング（株）

1　はじめに

計算機上の地図データにはベクトルデータのものと、ラスタデータのものがある。ラスタデータの場合、入力はイメージスキャナで行うが、一度に読み込める面積には限度があるため広域の地図を扱う際は分割してスキャンする。

ところが、スキャンした複数の地図を単に並べただけでは境界にずれが生じうまくつながらない。原因としてはスキャン時の地図原稿の傾き・スキャナのフィードの滑べり・湿度による紙の伸縮等が考えられる。この問題に対し、ラバーシーティングという技術で解決を図る[Lupn87] [Saal88]。ラバーシーティングとはラスタの画像をラバー（ゴム）のシートと考え、伸び縮みをさせて変形させ、境界付近のマッチングを図る技術である。従来はこの変形のために四つの点を引張るものだけが採用されていたが、充分な補正精度を得ることができなかった。

そこで我々は、ずれている対象図形を任意の数だけ指定し補正するアルゴリズム、すなわちゴムシート上の何点でも引っ張ることのできるアルゴリズム（マルチポイントラバーシーティング）を考案した。

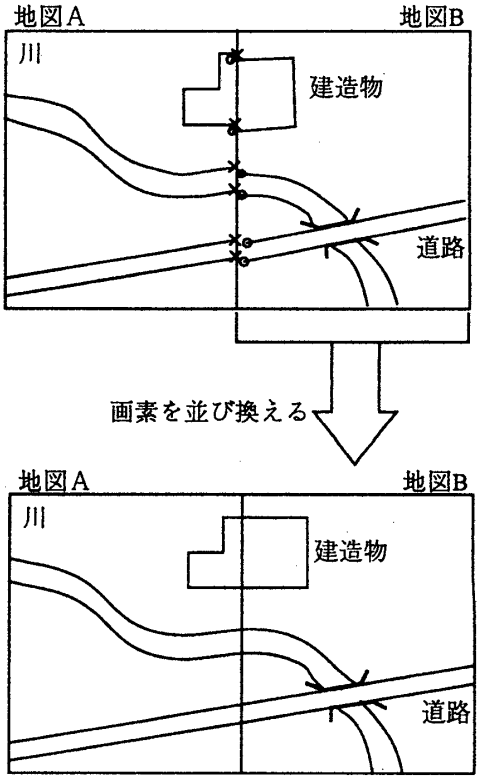

図 1: 補正の概要

2　実現手法

2.1　概要

図 1に補正の概要を示す。地図 A と地図 B は別々にスキャンした地図である。図の上側はスキャンしただけのものであり、「ずれ」が生じている。まずこの「ずれ」を補正するため本来一致すべき点を○印と×印を付けて指定する。その指定にしたがって地図 B を補正をしたものが図の下側の地図 B' である。この○印と×印を結ぶベクトルを以下補正ベクトルと呼ぶ。

2.2　アルゴリズム

マルチポイントラバーシーティングのアルゴリズムは、任意の n 画素を別の n 画素に移動させた時、他の画素をいかに移動させるかという問題に言い替えることができる。

図 2は画像データを現実のラバーシートになぞらえてモデル化したものである。○印は画素を表し、それらがバネで相互につながれている。この中でいくつかの画素を引っ張って移動させた時、他の画素はどう移動するかを考えてみる。

ここで、画像の大きさを $r \times s = N$、各画素の元の座標を $\vec{P}_{i,j}$、移動後の座標を $\vec{P}'_{i,j}$、とすると x 方向 y 方向それぞれの各バネの長さの変位は

$$\Delta x_{i,j} = |\vec{P}_{i+1,j} - \vec{P}_{i,j}| - |\vec{P}'_{i+1,j} - \vec{P}'_{i,j}|$$

$$\Delta y_{i,j} = |\vec{P}_{i,j+1} - \vec{P}_{i,j}| - |\vec{P}'_{i,j+1} - \vec{P}'_{i,j}|$$

であり、バネ定数を k とすると全体のエネルギーは

$$E = \sum_{i=1}^{r-1}\sum_{j=1}^{s}(\frac{1}{2}k\Delta x_{i,j}^2) + \sum_{i=1}^{r}\sum_{j=1}^{s-1}(\frac{1}{2}k\Delta y_{i,j}^2)$$

となり、この E が最小になるように $\vec{P}'_{i,j}$ を決めれば良

Multipoint rubber sheeting on GIS
Toyoaki SUZUKA
Hitachi Software Engineering Co., Ltd.

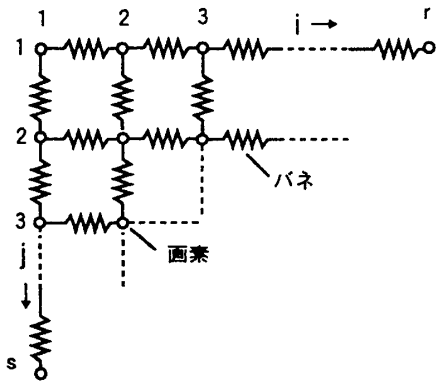

図 2: ラバーシートモデル

い。ところが、この計算量は膨大なものであり、実用的な時間内に処理することができない。

そこでより簡易な方法として次の式を考案した。これにしたがって計算すれば実用的な時間内に、（ラバーシートのモデルに対して）近似的に画素の移動先を計算することができる。ここで、補正ベクトルの元の点を $\vec{C}_i$、移動先を $\vec{C}'_i (1 \leq i \leq n)$、元の画像の座標を $\vec{X}$、その移動先の座標を $\vec{X}'$ とする。

$$\vec{X}' = \vec{X} + \sum_{i=1}^{n} \frac{1}{S|\vec{C}_i - \vec{X}|}(\vec{C}'_i - \vec{C}_i)$$

但し、

$$S = \sum_{i=1}^{n} \frac{1}{|\vec{C}_i - \vec{X}|}$$

である。

この式の意味は次のようなものである。

1. $\vec{X}' - \vec{X}$ は、各補正ベクトルに対しある係数を乗じたものの和となる。
   例） $\vec{X}' - \vec{X} = \alpha(\vec{C}'_1 - \vec{C}_1) + \beta(\vec{C}'_2 - \vec{C}_2)$

2. ある補正ベクトルの影響がそのベクトル付近では大きく、遠くでは小さくなるように、係数を $\vec{X}$ の $\vec{C}$ からの距離に反比例したものとする。
   例）距離を 3 及び 5 とすると、$\alpha = \frac{1}{3}$、$\beta = \frac{1}{5}$ となる。

3. 但し、この係数をそのまま使用すると、補正ベクトル付近で非連続な補正がかかってしまうので、係数の和が 1 になるように正規化する。すなわち各係数を全ての係数の和で割る。
   例）上の係数を計算し直すと。

$$\alpha = \frac{\frac{1}{3}}{\frac{1}{3}+\frac{1}{5}}, \beta = \frac{\frac{1}{5}}{\frac{1}{3}+\frac{1}{5}}$$

この分母が S である。

ところで、この式は入力画像の画素 $\vec{X}$ から、出力画像の画素 $\vec{X}'$ を求めるものであるが、これでは出力画像の全ての画素を対応させることができず、隙間が生じてしまう。そこで、逆に出力画像の画素 $\vec{X}'$ から入力画像の画素 $\vec{X}$ を対応させる逆関数の式が必要となる。これは、元の式の記号に「′」がついている項からは「′」を削除し、逆についていないものには付加するということで得られる。

3 高速化

上述の式を用いた時の計算量は全画素数と補正ベクトルの数の積となる（$O(nN)$）。これはバネによるモデルと比べれば小さいが、絶対的に小さいとはいえない。このため四分木の手法を用いて高速化を図った。四分木の原理は、画像をある大きさの正方形のブロックに分割し、注目したブロックがその中で、ある意味で“一様”ならば、そのブロックの単位で処理し、ブロックが“複雑”ならば、そのブロックをより小さな 4 個のブロックに分割し、以後、再帰的に処理するというものである。ラバーシーティングの性質上、画像の狭い範囲を見た時に各画素の移動ベクトルはほとんど同じと考えられ、四分木のアルゴリズムを適用した。

4 まとめ

4.1 成果

1. 小さい単位の地図を張り合わせて大きな地図として扱うための技術を確立した。

2. 四分木アルゴリズムにより元の式と同等の精度を保ちながら 8 倍程度の高速化を図ることができた。

4.2 今後の課題

1. このアルゴリズムのベクトルデータへの適用

2. 補正量が大きくかつ画像自体が大きい時に、補正ベクトルの影響範囲が狭くなってしまいうまくいかないことがある。

参考文献

[Saal88] Alan Saalfeld: Conflation Automated map compilation, Int. J. Geographical Information Systems, 1988, Vol.2, No.3.

[Lupn87] Anthony E. Lupien and William H Moreland: A General Approach to Map Conflation, Proceedings of AUTOCRATO 8 held in Baltimore in March 1987.