Automatic image mosaic assembly

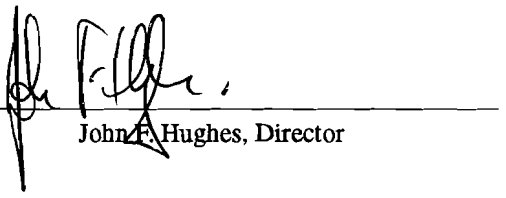
by
Dongbai Guo
Sc. B., Shanghai JiaoTong University., 1994
Sc. M., Brown University, Engineering, 1998

A thesis submitted in partial fulfillment of the
requirements for the Degree of Master of Science
in the Department of Computer Science at Brown University

Providence, Rhode Island
May 1999

This thesis by Dongbai Guo is accepted in its present form by
the Department of Computer Science as satisfying the thesis requirement
for the degree of Master of Science.

Date 4/27/99



John F. Hughes, Director

Approved by the Graduate Council

Date _____

_____

Peder J. Estrup
Dean of the Graduate School and Research

Abstract

We describe a method for assembling sub-images of a single large image (too large to be photographed or scanned as a whole) into a coherent mosaic with no prior information about their placement or adjacency. The technique is based on two assumptions: (1) that we have a fairly reliable image-registration subroutine, and (2) that the similarity measurement reported by the image-registration algorithm is larger on average between two properly registered images than between improperly registered images.

We also show how a modified version can be used to stitch together random images into a kind of artistic collage.

Contents

List of Figures

Chapter 1

Introduction

Building image mosaics arises in multiple applications [3, 10, 11], from scanning (in which large-format items are scanned in multiple small pieces and then reassembled in software), to environment mosaics (in which multiple photographs from a single center of projection are assembled into a single coherent whole), to microscope mosaics (in which multiple images of a sample, too large to see all at once, are recorded and then assembled into a coherent whole). There are two approaches to forming image mosaics. One is to gather the sample images in a regular fashion [8] and record the overlap information so that they can later be assembled into a whole [7]. There are microscope systems with stepper-motor controls for moving the sample table under the lens in two dimensions, a modest processor, and software to assemble the gathered images; these can cost $30,000 USD. In a traditional mosaic system, a user places the images in their approximate locations and the software then refines the positions to compose the whole [4, 10]. In the other approach, taken in our system, the software takes the input images in random order and determines and refines an overlap structure until a coherent whole evolves. We have applied this system to histological slices gathered under a high-power microscope (Figure 8.1), portions of a nautical chart gathered with a scanner (Figure 8.3), and (as a test of its robustness) to a set of photographs taken with a hand-held digital camera (Figure 8.4).

Since our approach is tolerant of error in image registration, it can be used to improve the robustness of a traditional mosaic system.

There are two essential components to our system. The first is an algorithm to determine optimal registrations for pairs of images; the second is the heuristic for assembling these into a coherent whole. We assume that the image registration procedure can take any pair of images and return a transformation from one to the other (for example, "shift up by 10 pixels and right by 8") that describes an optimal matching between them *for most image pairs*, and also return the resulting "image distance" between the images (see Appendix 1). In the case of the microscope and scanner images, the transformation is always a translation[1]; for the digital-camera images, the transformation is in general a projective transformation. Nonetheless, we used our translation-only matching algorithm and it managed to determine quite good matches. Our image-matching is done by a multi-scale algorithm (see Appendix 1), but any image-matching algorithm could replace it; as

[1] We scanned the nautical chart being careful to rotate as little as possible.

there is an extensive research literature on this subject [5, 15] and previous image mosaic papers cover this topic extensively [13, 8], we will not cover it in detail in this paper.

The second essential component is the heuristic for determining a layout from an error-contaminated set of image matches. The heuristic must take the $n(n-1)/2$ potential image overlaps and determine which are *spurious* (i.e., represent the best match between two pieces of the whole which do not in fact overlap), and which are *salient* (represent overlaps in the whole), and among the salient ones, for which pairs the image-matcher has determined the correct transformation. We assume that for most salient image-pairs, the image-matcher computes the correct transformation. We further assume that the image distance (see Appendix 1) for such a correctly-matched salient pair is less than the average image distance for spurious image-pairs. With these two assumptions, we take a collection of input images and build a mosaic as described below.

Chapter 2

Related work

Some previous works concentrate on creating mosaics for 1D and 2D manifolds from known image adjacency information [10, 4]. Other works, although they do not require explicit input, derive such adjacency information from the time coherence of the input images [8]. Most of these works are based on minimizing the errors of the misalignment of neighboring image pairs. Shawhney *et al.* [9] recently proposed a global minimization stratergy, introducing the following target function for global alighnment :

$$\min_{\{P_i\}} \sum_{\mathbf{x}} var_i\{I_i(P_i(x))\} + \sigma^2(Area) \tag{2.1}$$

Where I_i is ith source image, P_i the maps a point in the mosaic space to a point in the image i's space. They suggest a two steps approach for minimizing this function, step 1, local registration, a squence of neighboring video frames laid out using local optimal registration; step 2, global adjacency information is derived from the coarse layout and further optimization is performed to minimized the above equation. The problem with this heuristic approach is their method only works with video panoramas, where the time coherence of the input images help deriving initial layout the mosaic.

Some work (e.g., [1]) has been done on a superficially similar problem that arises in nuclear magnet resonance imaging: reconstruction of a three-dimensional set of points from noisy distances between pairs of these points. This problem differs in several respects from our problem. As the authors point out, their alogrithm assume noise free input with no more than 50 percent of the data randomly corrupted. The input to our alogrithm are guesses of the image placement containing registration error. Our algorithm works even when the input consists of more than 90 percent corrupted data. The restriction of more than 50 percent correct input makes their algorithm impractical for image mosaic applications. [1]

[1]For mosaic on 2D manifolds, the total number of the input from pairwise image registration is $n(n-1)/2$, where n is the total number of input images, while the number of non-corrupted input pairs is proportional to kn where k is the average connectivity of the layout graph (typically less than 8). The input associated with pairs that are not adjacent in the layout (the oracle) is corrupted.

Chapter 3

Notation and Definitions

Let $\{I_1, \ldots, I_n\}$ be a set of n images, and I_m be a mosaic of all the images. The function $x_m = L_i(x_i)$ maps a point x_i in the image coordinates to the mosaic coordinates x_m. A layout $\mathcal{L}$ is a set of functions $\{L_1, \ldots, L_n\}$ that describe the positioning the images in the mosaic. $\mathcal{L}$ determines a graph we call the *layout graph* G_L. It has a node i for each image I_i; and there is an edge between node i and node j $(i < j)$ if images I_i and I_j overlap in the layout, and this edge is labeled by the transformation T_{ij} that maps x_i to x_j. [1] We denote this map from layout $\mathcal{L}$ to graph G_L by the suggested U.

Definition 1 *We say that a cycle $v_1 v_2 \ldots v_k$ in the graph G_L is* consistent *if the composition of the translations $T_{v_1 v_2}, T_{v_2 v_3}, \ldots T_{v_{k-1} v_k}, T_{v_k v_1}$ is the identity. A graph is* consistent *if all its cycles are consistent.* [2]

A graph is consistent if all the cycles of a cycle basis are consistent. And the reconstruction of the mosaic from a consistent graph is unique up to congruent.

The map U has a partial "inverse": if G is a connected graph whose edges are labeled by translations, and if G is consistent, then by placing one image I_i at some arbitrary location, say x_0, we can determine a placement of each other image I_j by finding a path $\{i, v_0, v_1, \ldots, v_k, j\}$ that goes from i to j in G. The image I_j is then placed at $T_{v_k j} \circ \ldots \circ T_{v_0, v_1} \circ T_{i, v_0} x_0$. This location is independent of the choice of path when the graph is consistent. This layout $\mathcal{L}(G)$ of images has the property that the labeled, connected and consistent graph G is a subgraph of $U(\mathcal{L}(G))$. The other edges of $U(\mathcal{L}(G))$ (those not in G) are called *spontaneous* edges.

We formalize the image mosaic problem as a minimization problem in the next few sections.

[1] Formally, $T_{ij} = L_j^{-1}(L_i)$

[2] In the language of algebraic topology, we would say that the assignment of transformations satisfies the *cocycle condition.*

Chapter 4

The Heuristic

It follows from definition 1 that any *spanning tree* is automatically consistent. We use this fact in our heuristic for mosaic assembly. The idea of the heuristic is as follows. Use image registration to guess, for every pair I_i and I_j of images, what transformation T_{ij} correctly describes the relative positions of these images (assuming these images overlap). Each guess gives us an edge. At the same time, assign a *cost* c_{ij} to each edge that depends on the image distance between the intersecting parts of I_i and I_j (see Appendix 1). This cost reflects our confidence in the correctness of T_{ij}: low confidence means high cost. Next, select a minimum-cost spanning tree T, construct the corresponding layout $\mathcal{L}(T)$, and then check the consistency of the corresponding layout graph $U(\mathcal{L}(T))$ by examining the spontaneous edges. Inconsistency in this graph indicates that some edge of the spanning tree has an incorrect transformation; we adjust the costs of the edges of the spanning tree, find the new minimum-cost spanning tree, and repeat.

For the sake of efficiency of the iterative part of the heuristic, we assume the existence of a table that provides a rough estimate of the image distance between each pair of images when they are overlapped in any given way. In our program, this table was produced during the image registration part of the heuristic: for scaled-down versions[1] of B_i and B_j, we tried all possible overlaps (discretized to 16-pixel steps in the original image), and recorded the image distance between the overlaps. By using this table during the iterative part of the heuristic, we avoid calls to the image-distance routine, although at the cost of some error.

4.1 Testing Cycle Consistency.

Before outlining the heuristic in greater detail, we describe a technique it uses to test cycle consistency.

A cycle $C = v_1, v_2, \ldots, v_k$ induces a transformation $Q = T_{v_k,v_1} \circ T_{v_{k-1},v_k} \circ T_{v_{k-2},v_{k-1}} \ldots T_{v_2,v_3} \circ T_{v_1,v_2}$. Since there are errors in the transformations produced by the image matcher, even for a cycle consisting entirely of correctly-matched images, the transformation Q will differ somewhat from the identity—more for longer cycles. To test this consistency, we pick four points $P_1, \ldots, P_4$ at the corners of a unit square and compute

[1] We used scalings of either 16 or 32 in each dimension

the consistency measure of the cycle as

$$cm(C) = \exp\left(-\frac{\sum_i ||P_i - Q(P_i)||}{(\text{length}(C) - 1)d}\right),$$

where d is a tuning constant and length(C) denotes the number of edges in the cycle C.[2]

4.2 Forming the graph

Image registration gives us a complete graph on n nodes. We prune this graph, retaining relatively few edges, before beginning the iterative part of the heuristic, in order to improve the efficiency of the latter.

We begin by examining all triangles in the input graph for consistency; for each triple of images (i,j,k) we test the cycle (i,j,k) and if it's consistent enough (see below), we keep it.

For triangles in the input graph, we chose the tuning constant $d = 16$ and used a threshold of $t_1 = 0.5$, so that a composite translation Q with a consistency measure of more than 0.5 has a cumulative error of no more than about 5.5 pixels.

We initialize our graph G to consist of all nodes, together with all the edges of all the sufficiently consistent triangles. Thus G has in it many triples of images that overlap nicely. On the other hand, it's unlikely to be connected, and there are many edges in the input that are promising candidates for inclusion in the final result; we add several more of these to get a good starting point.

In particular, for each node in G, we include the incident edge that has minimum cost. Furthermore, if the node has degree less than 4, we add incident edges until it has degree 4, adding in order of cost. The rationale for this is that in the output mosaic, we expect each image to be strongly matched with about four neighbors (above, below, left, and right), so we include some candidates for these matches. Finally, we assign a weight w_{ij} to each edge (i,j) of G; the weight w_{ij} is initialized to c_{ij}.

We assume that the graph G thus obtained is connected. In practice, this has not been a problem; the graph G has always been connected in our examples. To be disconnected G would have to have, say, two components with no really "good" edges between them. This represents a failure of the image matcher in a particular and consistent way – it's provided two regions of a jigsaw puzzle and no hint about how they fit together. If, however, G were disconnected, we'd need to add edges until the problem was resolved.

4.3 Iterative refinement

We now describe the iterative part of the heuristic in greater detail. The goal of this part is a spanning tree T, formed from the edges obtained in the image-registration part, for which $U(\mathcal{L}(T))$ is as consistent as possible, in a sense to be defined shortly. A final step, described in Section 4.5 takes this locally-consistent graph and creates a layout from it.

Our basic approach is as follows: find the minimum-cost spanning tree T, and obtain the corresponding layout graph G_L. Examine the spontaneous edges of G_L; depending on whether the cycle formed by such an

[2] In the case described in this paper, the vectors $P_i - Q(P_i)$ are all the same; in more general situations, Q may be a projective transformation, in which case they may differ.

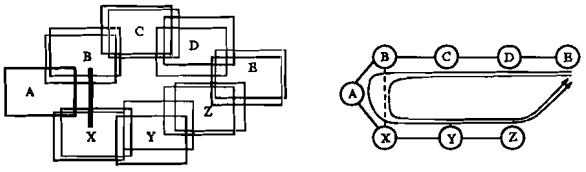


Figure 4.1: The graph shown determines the layout in blue; the true layout is in red; the discrepancy is due to image-registration error. The spontaneous edge *EZ* is inconsistent because of accumulated error in the cycle *EDCBAXYZ*. But once edge *BX* is added, the shorter cycle *EDCBXYZ* can be used, and the error is reduced to an acceptable level. Because edge *BX* is not perfectly consistent with *AB* and *AX*, there's no consistent way to draw the layout once *BX* is added. But *BX* serves as a sort of "strut" to make the structure more rigid and to mask the consistency of *AB* and *AX* from spontaneous edges far from them (like *EZ*).

edge with the tree T is consistent or not, decrease or increase the weight of the spanning-tree edges on the cycle. This will reward edges whose inclusion in the spanning tree leads to consistency, and punish edges whose inclusion leads to inconsistency.

However, we must cope with the fact that even if the edges along a path of T are salient edges (they correspond to pairs of images that overlap in the large image), the transformations associated with these edges may not be precisely correct. Small errors in these transformations accumulate to produce large errors in the relative position of images whose nodes are distant in the spanning tree. Thus there are inconsistent cycles that would be consistent if the local transformations were precisely correct.

We use a procedure TRICONSIST to help us overcome and correct local errors. This procedure iteratively selects edges to add to the spanning tree, obtaining a graph G^*. Edges are added only if the cycles they form in G^* are highly consistent. We derive the relative position of a pair of images B_i and B_j from the transformations on the shortest path in G^* that connects nodes i and j. Since G^* has more edges than T, shortest paths in G^* tend to be much shorter than those in T, so there is less opportunity for small errors to accumulate. The procedure is given at the end of this section.

The core loop. We now give pseudocode for the iterative part.

0. UPDATEWEIGHTS(G_L, G^*, T)
1. **repeat**
2. $T \leftarrow$ MINWEIGHTSPANNINGTREE(G,W)
3. $G_L = U(\mathcal{L}(T))$
4. $G^* \leftarrow$ TRICONSIST(G_L, T)
5. UPDATEWEIGHTS(G_L, G^*, T)
6. $G \leftarrow G_L$
7. **until** CHECKSTOP(G_L)

Note that in Step 6, we update G to be the graph G_L obtained in Step 3. Thus the edges from which we select the minimum-weight spanning tree in the next iteration correspond to images that overlap in the current layout.

Updating the weights. As we were examining spontaneous edges as candidates for inclusion in G^*, some were consistent with the data in G^* and some were not. An edge of G^* may have been involved in multiple consistency tests; if most of those tests were successful, we'd like to believe that the edge is salient and that

its transform is correct. If most were failures, we conjecture that the edge is not salient or that its transform is incorrect. based on these conjectures, we decrease or increase the weight of the edge to increase or decrease its chance of inclusion in the spanning tree found in the next iteration.

To compute the weight update, the procedure uses the table described at the beginning of this section. It accesses the table via the procedure lookup($\mathcal{C}, e$). This procedure takes a cycle $\mathcal{C}$ in the graph containing an edge $e = (i,j)$ and computes the composition ρ of the transformations along the path $\mathcal{C} - e$. It then obtains from the table the cost of overlapping B_i and B_j according to the transformation nearest to ρ. In essence it asks "if the transformation on e matched perfectly with the rest of the transforms in E, what would the image-matching cost on e have been?"[3]

0. UPDATEWEIGHTS(G_L, G^*, T):
1. **foreach** edge $ij \in G_L - G^*$
2. find the shortest path p in G^* join vertices i,j, $\mathcal{C} \leftarrow p + (i,j)$
3. **if** $cm(\mathcal{C}) > t_1$, **then** $cost \leftarrow c_{ij}$, $\mathcal{C}' \leftarrow \mathcal{C}$
4. **else** $cost \leftarrow$ lookup($\mathcal{C}, ij$), $\mathcal{C}' \leftarrow \mathcal{C} - \{ij\}$
5. **if** $cost < c_1$
6. **foreach** edge (g,h) in $\mathcal{C}'$, $c_{gh} \leftarrow (1-\epsilon)c_{gh}$
7. **else**
8. **foreach** edge (g,h) in $\mathcal{C}'$, $c_{gh} \leftarrow (1+\epsilon)c_{gh}$

Note that if the transform for e was found from a lookup, we don't change the cost on edge e, since that's the cost of *e with the transform that labels it.*

The procedure TRICONSIST(G_L, T) The procedure starts by setting G^* to consist of the edges in T. If there is an edge e in G_L forming a triangle with edges already in G^*, and if the triangle is highly consistent and its total edge-weight is small, the edge e is added to G^*. Step 7 lowers the weights of the edges in the triangle. (The weights of these edges would have been lowered in UPDATEWEIGHTS but for their inclusion in G^*.)

0. TRICONSIST(G_L, T):
1. $G^* \leftarrow T$
2. **until** stable
3. **foreach** triangle $s = (i,j,k)$ in G_L
4. **if** two edges, say (i,j) and (j,k) of s are in G^*
5. **if** $cm(s) > t_2$ and $W_{ij} + W_{jk} + W_{ki} < c_0$
6. $G^* \leftarrow G^* \cup (k,i)$
7. lower the weights of all edges in the triangle s.

Again, there are tuning constants t_2 and c_0 in the code. We choose t_2 to accept translations of less than three pixels. We choose c_0 by examining the costs of all $n(n+1)/2$ pairs of images in the input, and, knowing that the output images will have some average degree (typically about 4), we look at the lowest $5n$ costs and use the largest of these as our threshold, so that we are likely to include any salient edges (edges corresponding to overlap in the large image).

Stopping. *CheckStop* checks whether the average cost, over all edges of G_L, is below some threshold t_3.

[3] We could actually invoke the image matcher on the images B_i and B_j with the given transformation, but such invocations are costly and the lookup-table approach works well in practice.

4.4 Tuning

There are several constants in the process that affect its performance; they influence how consistent a triangle must be for admission to the graph (d, t_1), how consistent (t_2) it must be and how costly (c_0) it may be before being added to the consistency graph, how consistent a cycle must be to affect the weights (t_1 again), not to mention how fast the weights should change(ϵ). Our program is robust in the sense that ϵ can be chosen anywhere within a rather large interval without significantly affecting the results.

4.5 Creating Output

The graph G is now locally consistent – small cycles in it have very small pixel errors. But since there are multiple paths from one node to another, and they may not be consistent, we cannot simply apply $\mathcal{L}$ to G. We therefore treat G, with its edge labels, as a collection of constraints: if image B_i is to be placed at location X_i, then for any edge (i,j) in G, we need $X_i - T_{ij}X_j = 0$. We assign some X_i to the origin, and then this collection of constraints on the X_i has a unique least-squares solution, which we compute. We now have positions for all the images, and can lay them out.

To actually create an output image, we must do something at the places where images overlap. We simply blend between any overlapping images: the weight w_i of the pixel Z_i from image i is the shortest distance from Z_i to any border of image i, so that the image "fades out" in the blended result as we approach its border. Thus the output pixel is

$$Z = \frac{\sum_i w_i Z_i}{\sum_i w_i}. \tag{4.1}$$

Chapter 5

Art Image Blending

In figure 8.2 we have created a collage of art images. Since these do not come from a single coherent whole, the assumptions of the heuristic are invalid; applied directly, it tends to place all the images atop one another, since a great many of them have a bright center/top, and a dark border/bottom. We therefore did a little preselection as follows:

We started with 212 images, and computed the pairwise overlaps/costs. We selected a subset of these images, including each image that matched very well with some other image. In the UPDATEWEIGHTS portion of the heuristic, we added a penalty for high-degree vertices so as to encourage "spreading out" of the images.[1] Moreover, we stopped the program early. Figure 8.2 is one of the layouts arising during the run.

In this final collage, there are only about 20 images; as we created the final layout, we successively laid down the images. If laying down an image caused some pixel to be covered by more than four images, we skipped that image in the collage. This *ad hoc* method helped reduce the noise-and-clutter that comes from too many overlapping images.

[1] For each edge of G_L, we multiplied its weight by a linear function in the max degree of its endpoints. For a typical node of degree four, this represents about a 5% penalty.

Chapter 6

Discussion

Our heuristic is a kind of loose gradient search. Presumably with noisy enough input data, it can get locked in local extrema. Because for some image sets, there's no way to know where certain image-pieces fit—they falsely match too many other pieces – there's no way in general to prove that a process for this sort of mosaicking converges to the "right" answer.

The method works quickly and with surprisingly good results: we can assemble the chart images from the pairwise registration information in less than one second[1]; a close examination of the results shows that even with the relatively rough scans of the original (the chart was wrinkled in some places, easy to see along the top edge), the registration is quite good.

Two solid black images can be overlapped in many different ways, while two complicated and varied images typically admit at most one good registration. It has been suggested to us to use a weighting scheme in which the confidence of a potential image overlap depends on the variance of the two images.

[1]For 20 images, the entire registration process requires 90 mins. on a 200MHz UltraSPARC; however, we typically register several pairs in parallel to speed up this process.

Chapter 7

Generalization

The heuristic described is for images that can be matched under translation. The same heuristic can be generalized to images that are extracted from a single composite entity but are related by transformations in a large group – all rigid motions, for example, or all projective transformations. The cocyle condition is still a necessary condition for consistency. Of course, if the images are all from a common center of projection, then the composite image is spherical rather than planar, and it's important to treat it as such rather than trying to do a projection to a plane [14]. Fortunately, our process separates out the finding and stabilizing of the inter-image transformations from the process of making an output image. As Szeliski points out [13], the general transformation between images taken with the same camera from the same point is a three-dimensional subgroup of the projective group, so the possibility of searching that group for matches, just as we search the 2D translation group, is at least plausible.

Chapter 8

Future work

As most of the computational time is spent on image registration, one of future direction of this work is to cut down the image registration computation. Typically only kn (k typically less than 4) pairs of image registration is required to recontruction the mosaic, while we are registering n^2 pairs. The following modification to the original algrithm should help reduce the registration cost.

0. FeatureVector$I$
1. seperate an image I into M by N blocks
2. **foreach** block I_ij of I
3. V_I = *concatenate*(V_I,*featureVec*(I_ij))

where FEATUREVEC can be a series of statistics of the image block, such as the average value for each color channel, the density of the edge, and texture of the block, etc. The purpose of dividing an image into blocks is to increase the reliability of the feature vector representation of an image. Feature vectors are frequently used in image retrival and matching applications. One of the basic assumption of feature vectors is that they are a coarse image similarity measurement, which fit our purpose here for initialization of the graph. A second choice is to compute the lookup table from the start. However this computation, though more reliable, is also very expensive, because it gives the coarse global registration between all image pairs while the feature vector method only provides a single value that relates to the coarse registration.

Now, instead of the registering image pairs, we compute the feature vector distance (the inner product) between image pairs, and assign this distance to be the weight of the MINWEIGHTSPANNINGTREE computation. The legitimacy of this approach is that neighboring images of a mosaic in general will have similar image content and therefore closer feature vectors distance. Once we have a minimal spanning tree T, we then compute the global multiscale image registration between every pair of images represented of its edges and label each edge of the T with the optimal transformation associated with the lowest cost. From the minimal spanning tree T, we may compute the layout graph G_L, which is consistent by default. For each edge of G_L that is not in T, we compute the cost of the image pair under the derived transformation. The step is not a registration computation because the image pair transformation is known, therefore can be performed quickly.

We then check whether we may stop at this time. If not, we probably started with a wrong T and populated G_L with wrong edges. One stratergy is to pick more edges of low feature vector distance and perform global registration. Once we have more than $n-1$ edges labelled via global registration, we may compute a spanning tree with respect to pairwise image cost instead of image feature-vector distance, which therefore leads to better accuracy. A second stratergy is to compute the local registration for edges of G_L with high image cost. The rationale behind this approach is that the transformation labeling this edge may contain too much cumulative error, which therefore leads to higher image cost. Unfortunately the local registration does not save a significant amount of time over global registration because of our multiscale approach. Nontheless, if the initial tree T is likely to be the right spanning tree of the embedded solution and the registration error is high, the second strategy is preferable to the first one.

Such a process can keep iterating until we populate the graph with sufficient number of edges. We can then construct the lookup table and use our previous algorithm to continue for futher iteration.

Therefore, following algorithm would like to be faster for computing a mosaic:

0. fastMosaic
1. **foreach** image B_i
2. V_i = FEATUREVECTOR(B_i)
4. **foreach** image pair i, j
5. approximate weight, $w = dot(V_i, V_j)$
6. $T \leftarrow$ MINWEIGHTSPANNINGTREE(G,W)
7. **foreach** each ij of T
8. compute the global registration between B_i, B_j
9. label edge ij with the optimal transformation and its cost
10. $G_L = U(\mathcal{L}(T))$
11. **foreach** edge ij of G_L
12. **if** cost of edge ij is not labelled,
13. lookup edge ij cost for the labelled transformation
14. CHECKSTOP(G_L)
15. **until** perform global registration for more than kn edges
16. compute lookup table
17. **foreach** edge ij that does not have a cost label yet
18. set it cost and transformation based on the lookup table
19. computeMosaic

Figure 8.1: A mosaic of histological microscope data. Boxed region has blending turned off to show seams. The left side is images without illumination adjustment, the right side is images adjusted by the method in Appendix 2. (20 640 x 480 images)

Figure 8.2: Art image

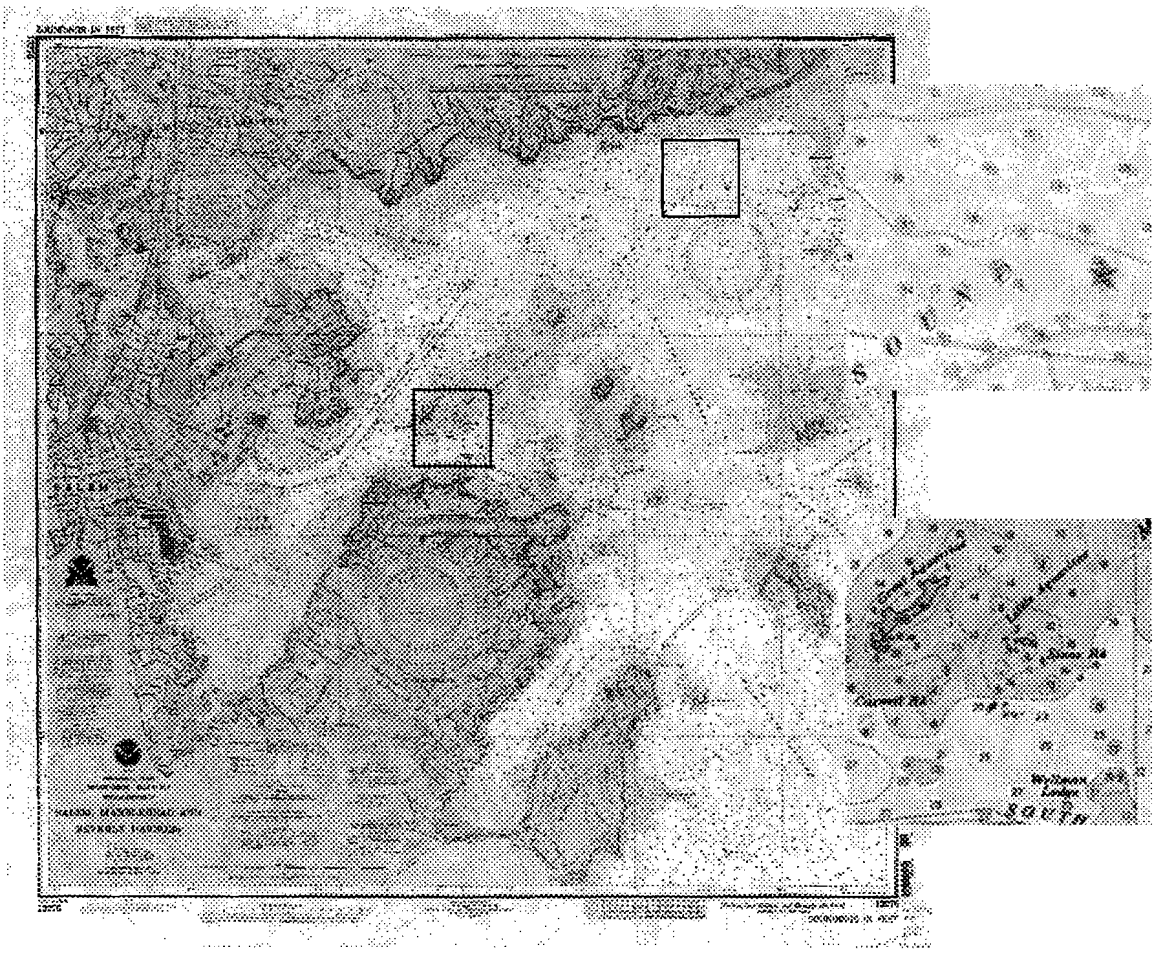


Figure 8.3: A nautical chart reassembled from scanned pieces. (36 612 × 1008 images)

Figure 8.4: Handheld photographs of a building. These are not all taken from the same point of view, and no attempt was made to correct for rotation or keystoning. (12 1024 × 1536 images)

Bibliography

[1] B. Berger, J. Kleinberg, and T. Leighton. Reconstructing a three-dimensional model with arbitrary errors. In *In Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pp. 449–458, 1996.

[2] T.J. Cham and R. Cipolla. A statistical framework for long-range feature matching in uncalibrated image mosaicing. *CVPR98*, pp. 441–447, 1998.

[3] S. E. Chen. Quicktime VR - an image-based approach to virtual environment navigation. In *SIGGRAPH 95 Conference Proceedings*, pp. 29–38, 1995.

[4] P. Dani and S. Chaudhuri. Automated assembling of images: Image montage preparation. *PR*, 28(3):431–445, March 1995.

[5] E. DeCastro and C. Morandi. Registration of translated and rotated images using finite fourier transforms. *PAMI*, 9(5):700–703, September 1987.

[6] S. Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Pattern Anal. and Machine Intell.*, 11(7):674–693, 1989.

[7] D.L. Milgram. Adaptive techniques for photomosaicking. *TC*, 26:1175–1180, 1977.

[8] S. Peleg and J. Herman. Panoramic mosaics by manifold projection. In *CVPR97*, pp. 338–343, 1997.

[9] H. Sawhney and R. Kumar. Robust video mosaicing through topology inference and local to global alignment. In *ECCV98,European Conf. Computer Vision*, volume 2, pp. 103–119, 1998.

[10] K. Schutte and A. Vossepoel. Accurate mosaicking of scanned maps, or how to generate a virtual a0 scanner. In *ACSI95*, volume 1, pp. 353–359, 1995.

[11] H.Y. Shum, M. Han, and R. Szeliski. Interactive construction of 3d models from panoramic mosaics. In *CVPR98*, pp. 427–433, 1998.

[12] E.J. Stollnitz, T.D. DeRose, and D. H. Salesin. Wavelets for computer graphics: A primer. *CGA*, 15(3):75–85, March 1995.

[13] Richard Szeliski and Heung-Yeung Shum. Creating full view panoramic mosaics and environment maps. In *SIGGRAPH 97 Conference Proceedings*, pp. 251–258, 1997.

[14] Y.P. Tan, S.R. Kulkarni, and P.J. Ramadge. The instability of planar mosaicking. In *Proceedings of the Tenth Yale Workshop on Adaptive and Learning Systems*, pp. 147–152, 1998.

[15] I. Zoghiami, O.P. Faugeras, and R. Deriche. Using geometric corners to build a 2d mosaic from a set of images. In *CVPR97*, pp. 420–425, 1997.

Appendix 1: Image Registration and Image Distance

For pairwise image matching, we use a multiscale search scheme [2, 4]. Given an image pair B_a and B_b, we first transform them into multiscale representations $\{B_a^s\}, \{B_b^s\}$, where superscript-s is the image scale, using a wavelet transform [6, 12]. At the coarsest representation c, which is typically the original image scaled down 32 times, we compute the image distance $d_{ab}(T_{ab}^c)$ for every integer displacement between the two images. We then find the five best matches (local minima of d) and refine them at the next resolution $c+1$. To refine a matching position T_{ab}^c, we first convert it to the new scale and then search locally (typically in a 5×5 pixel region) for the transformation T_{ij}^{c+1} that has the smallest image distance in this region. The refinement process stops at the highest resolution.

To measure the image distance between two images, we use following equation:

$$d_{ab}(T_{ab}^s) = \sum_i^{w^s} \sum_j^{h^s} \frac{\left(B_a^s(i,j) - B_b^s(i+x^s, j+y^s)\right)^2}{(w^s - x^s + 1)(h^s - y^s + 1)},$$

where $T_{ij}^s = (x^s, y^s)$ is the relative image translation of B_b^s to B_a^s, and w^s, h^s is the image size at the scale s. Image distance is the average squared pixel difference for the overlap region. In practice, to increase the robustness of the registration, we require two images have an overlap area of at least 20% of the image area. If a transformation T_{ab} leads to a smaller overlap area, we reset its associated distance $d_{ab}(T_{ij})$ to a very large value to prevent it being a candidate for good matches. The image matcher finally returns the smallest image distance we encounter at the highest resolution and its associated transformation.

Appendix 2: Microscopic image illumination adjument

Microscopic images suffers from non-uniform illumination artifact, which affects the correctness of the image registration. We have devoloped an automated method to adjust for the non-uniform illumination of the microscopic images. It has the advantage over earlier methods that the same region affected by different non-uniform illumination patterns has the same adjusted image.

We first divide an image into multiple blocks. The block size is chosen so that it is much larger than the variation of the local image details, yet small with repect to the illumination variation. This is not a pair of contradictory requirements for high magnification microscopic images, since illumination varies at a much larger scale than the image details. Typical, for 640×480 images, block sizes anywhere between 32×32 and 64X64 works well.

For each image block I, we adjust its centroid of histogram to a fixed intensity h_λ by following equation.

$$I_\lambda^{he} = I_\lambda \times \frac{h_\lambda}{i_\lambda^c}$$

where i_λ^c is the histogram centroid of I:

$$i_\lambda^c = \frac{\int_0^\infty i_\lambda h(i) di}{\int_0^\infty h(i) di}$$

The constant h_c is the same for all images. We randomly pick one image from the input image set and choose its histogram centriod to be h_c. This may create a problem for images with large changes in image foreground (at a scale close to that of the non-uniform illumination). In such cases, this method removes the foreground difference along with the non-uniform illumination.