CHAPTER 4

Simple Image Mosaics

Back in the days when photographs were printed on paper by special stores, one way to make a photograph of a large object was to take several different, overlapping pictures; print them; place one printed image down on a corkboard; then slide the other printed pictures around on a corkboard until they are registered to the first and one another; and then pin them down. This is a *mosaic* – a collection of pictures that have been registered (Figure 4.3). There are a number of reasons to build mosaics. You might simply not have the right camera, and so have to assemble a big picture out of small ones. In the overlapping portions, you have more than one picture, and can use this to make improved estimates of pixel values or to identify moving objects. You can show various interesting changes in the scene.

Methods for building mosaics out of digital images are now highly developed, and appear in a number of consumer applications. The key trick is registering the images. Procedures differ slightly depending on what class of geometric transformation is used to register the images. We will mostly discuss mosaics built by translating images; Chapter 33.2 describes methods relevant for more general transformations.

4.1 SIMPLE MOSAICS

For the moment, assume we have two images $\mathcal{A}$ and $\mathcal{B}$. These two images are overlapping views of a scene that can be aligned by a translation. These images are continuous functions of position in the image plane – they haven't been sampled yet. The fact that they can be aligned means that there is some t_x, t_y so that $\mathcal{A}(x, y) = \mathcal{B}(x - t_x, y - t_y)$ when both $x, y \in [0, 1] \times [0, 1]$ and $x + t_x, y + t_y \in [0, 1] \times [0, 1]$. Visualize this as placing $\mathcal{B}$ on top of $\mathcal{A}$, then sliding $\mathcal{B}$ by t_x, t_y; then the parts of $\mathcal{A}$ and $\mathcal{B}$ that overlap look the same. We are given $\mathcal{A}$ and $\mathcal{B}$ and must find t_x, t_y.

Least squares should spring to mind. We are dealing with sampled images, and so we will search for m, n that are integers, and minimize

$$C_{\text{reg}}(m, n) = \frac{1}{N_o} \sum_{\text{overlap}} (\mathcal{A}_{ij} - \mathcal{B}_{i-m,j-n})^2$$

where overlap is the rectangle of pixel locations with meaningful values for both $\mathcal{A}$ and $\mathcal{B}$ and N_o is the number of pixels in that rectangle. It is important that C_{reg} is an average, because we need to compare overlaps of different sizes (Figure 33.2)

Once we have a good estimate of the translation, we could refine it using bilinear interpolation (Section 33.2) to reconstruct values we don't have. But minimizing this cost function isn't just a piece of linear algebra. The obvious strategy for finding m, n is to apply each translation; compute the objective function; then

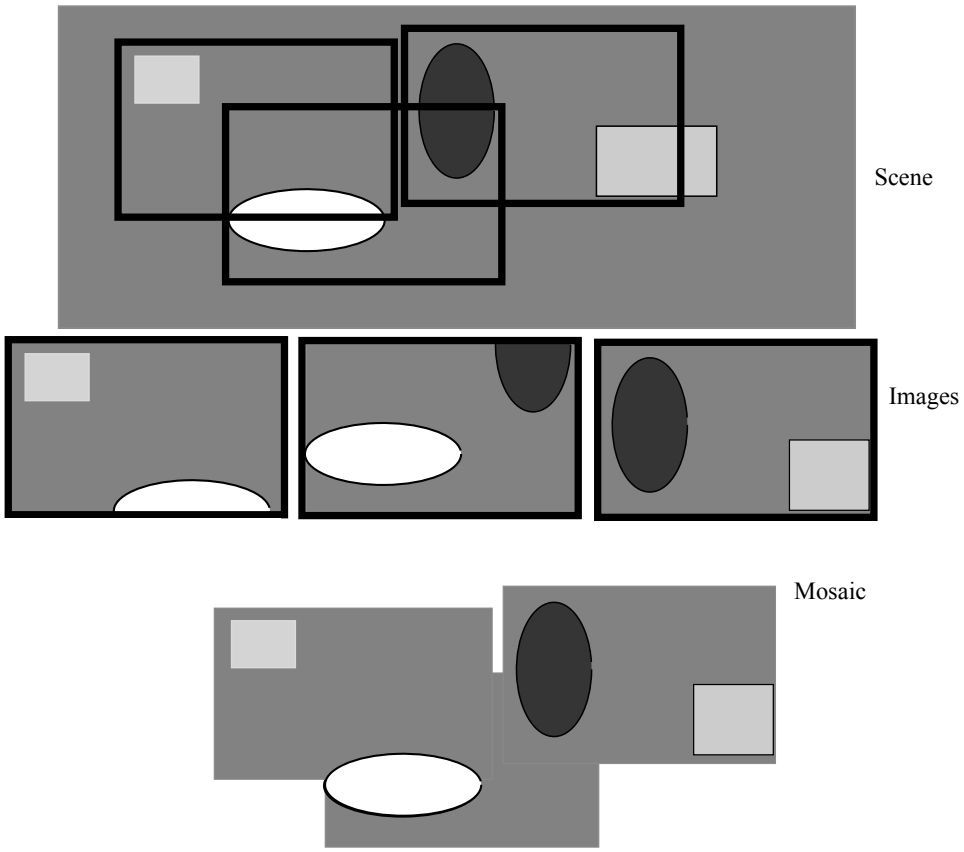

FIGURE 4.1: **Top** *shows a set of overhead images of a simple scene. The dark boundaries show each of three image frames.* **Center** *shows the actual images obtained in these frames.* **Bottom** *shows the mosaic that can be recovered by sliding images with respect to one another. This mosaic can't show features that haven't been imaged, but does show the relative configuration of scene components.*

take the translation with smallest value. This isn't a good strategy, because we must evaluate the objective function too often. The obvious modification – assume that m, n are in a small range, and search only those – doesn't help because we may not find the actual minimum. But notice that if we smoothed and subsampled $\mathcal{A}$ and $\mathcal{B}$, we could compute a coarse estimate of m, n from those, and then perhaps refine the estimate.

4.1.1 Registration, RGB, and Prokudin-Gorskii

Color photography is usually dated to the 1930's when it first became available to the public. In fact, James Clerk Maxwell described a method to capture a color photograph in an 1855 paper. The procedure likely looks straightforward to you: obtain three color filters, and take a picture of the scene through each of

these filters. Capturing these *color separations* presented a number of technical challenges, and the first color photograph was taken by Thomas Sutton in 1861. Actually displaying pictures obtained like this was tricky. One had to pass red light through the red separation, green through the green, and blue through the blue, then ensure all three resulting images lay on top of one another on screen. Turning them into the image files we are familiar with is also tricky, because each layer of the separation is typically a bit offset from the others (the camera moved slightly between photographs), and each layer has aged and been damaged slightly differently.

A class assignment, now hallowed by tradition in computer vision, but likely to have originated with A. Efros in 2010, studies this problem. It uses the pictures of Sergei Mikhailovich Prokudin-Gorskii (1863-1944) traveled the Russian empire and took color photographs of many scenes. He left Russia in 1918. His negatives survived and ended up in the Library of Congress. A digitized version of the collection is available online. The assignment asks students to register the color separations for some of these images.

This is very like forming a mosaic (the separations overlap; they can be aligned by translation). It presents two important challenges. First, the separations do not agree exactly when they overlap – if they did, the image would be a monochrome image – and so the cost function needs to be adjusted. Second, the high resolution version of the scans are quite big, and there can be moderately large offsets. Looking at each offset in turn is hideously expensive. We will deal with that problem in the next section.

Notation for generalizing the cost function is easy. Write $\mathcal{A}_{O(m,n)}$ (etc.) for the image window of $\mathcal{A}$ that overlaps the other image when that image is translated by m, n. We can write

$$C_{\text{reg}}(m,n) = d(\mathcal{A}_{O(m,n)}, \mathcal{B}_{O(m,n)}).$$

The original cost function had

$$d(\mathcal{A}_{O(m,n)}, \mathcal{B}_{O(m,n)}) = \frac{1}{N_o} \sum_{\text{overlap}} \left(\mathcal{A}_{ij} - \mathcal{B}_{i-m,j-n}\right)^2.$$

Useful alternatives include:

- The *cosine distance*, given by:

$$\sum_{\text{overlap}} \frac{(\mathcal{A}_{ij} * \mathcal{B}_{i-m,j-n})}{\sqrt{\sum_{\text{overlap}} \mathcal{A}_{ij}^2} \sqrt{\sum_{\text{overlap}} \mathcal{B}_{i-m,j-n}^2}}.$$

  Annoyingly, this cost function is largest when best, even though it's called a distance. Some authors subtract this distance from one (its largest value) to fix this.

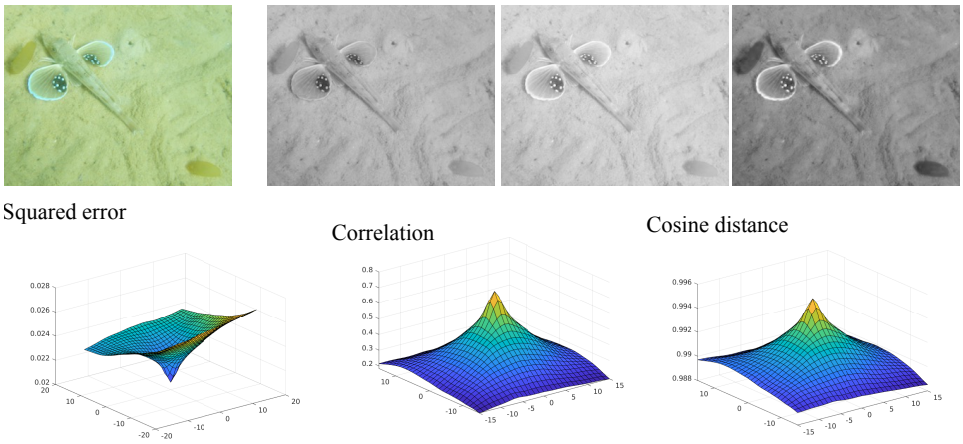

FIGURE 4.2: **Top left** *shows a Gurnard, flashing its pectoral fins in alarm, at Long Beach in Cape Town.* **Top rest** *shows the color separations of this image (in red, green, blue order). The image is slightly blue-green (taken at about 5 meters depth, where water absorbs red light), and this shows as a darker red separation.* **Bottom** *shows how various cost functions react to registering red to blue. The correct alignment is at 0, 0 and the images are 257 by 323. Notice that: all the extrema are in the right place, but the correlation and cosine distance must be maximized, and the squared error minimized; the squared error changes relatively little from the best to the worst, because the blue image is rather unlike the red; both cosine distance and correlation are much more sensitive.*

- The *correlation coefficient*, given by:

$$\sum_{\text{overlap}} \frac{(\mathcal{A}_{ij} - \mu_A) * (\mathcal{B}_{i-m,j-n} - \mu_B)}{\sqrt{\sum_{\text{overlap}} \mathcal{A}_{ij}^2} \sqrt{\sum_{\text{overlap}} \mathcal{B}_{i-m,j-n}^2}}$$

$$\text{where } \mu_A = \frac{1}{N_O} \sum_{overlap} \mathcal{A}_{ij} \text{ and}$$

$$\text{where } \mu_B = \frac{1}{N_O} \sum_{overlap} \mathcal{B}_{ij}.$$

  This is big for the best alignment. Notice how this corrects for the mean of the overlap in each window.

Each is in the range -1 to 1, and neither scales with the size of the overlap neighborhood. Terminology in this area is severely confused. The cosine distance isn't a distance; it is sometimes referred to as *normalized correlation*; and sometimes as *correlation*. Several functions similar to correlation are referred to as correlation.

4.2 TECHNIQUE: SCALE AND IMAGE PYRAMIDS

TODO: two zebra images?

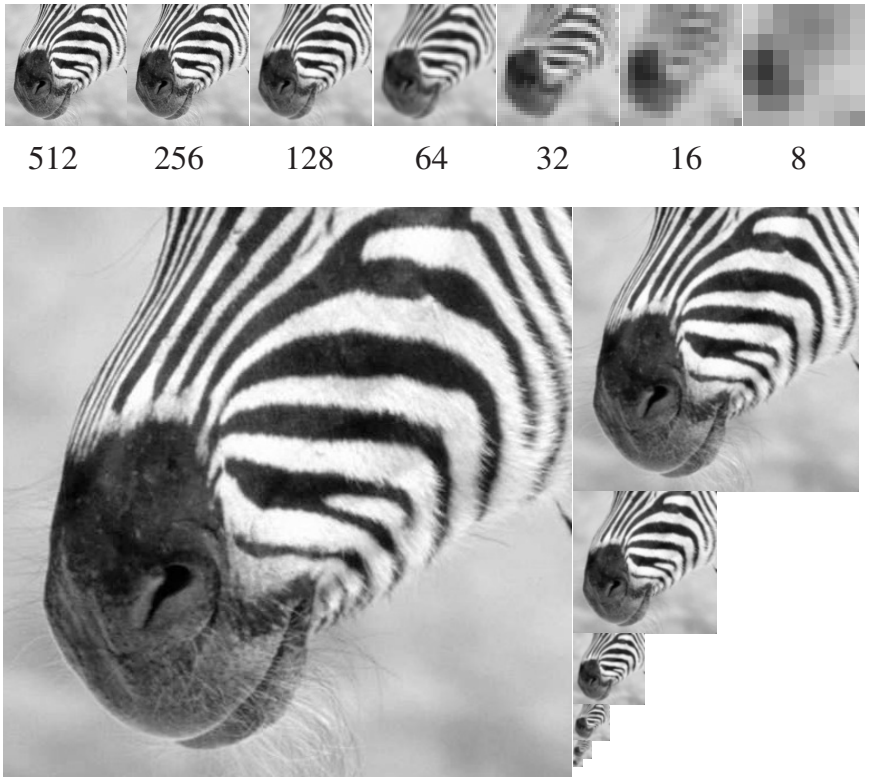

FIGURE 4.3: *A Gaussian pyramid of images running from 512x512 to 8x8. On the top row, we have shown each image at the same size (so that some have bigger pixels than others), and the lower part of the figure shows the images to scale. Notice that if we convolve each image with a fixed-size filter, it responds to quite different phenomena. An 8x8 pixel block at the finest scale might contain a few hairs; at a coarser scale, it might contain an entire stripe; and at the coarsest scale, it contains the animal's muzzle.*

Images look quite different at different scales. An *image pyramid* is a collection of smoothed and resampled representations of an image. The name comes from a visual analogy. Typically, each layer of the pyramid is half the width and half the height of the previous layer; if we were to stack the layers on top of each other, a pyramid would result. In a *Gaussian pyramid*, each layer is smoothed by a symmetric Gaussian kernel and resampled to get the next layer (Figure 4.2). These pyramids are most convenient if the image dimensions are a power of two or a multiple of a power of two. The smallest image is the most heavily smoothed; the layers are often referred to as *coarse scale* versions of the image.

Now look at the zebra's muzzle in Figure 4.2, and think about registering this image to itself. The 8×8 version has very few pixels, and looks like a medium dark bar, darker at the muzzle end. Finding a translation to register this image to itself should be fairly straightforward, and unambiguous. Assume we find m_8, n_8.

In the 16×16 version, some stripes are visible. Registering this image to itself might be more difficult, because the stripes will create local minima of the cost function (check you follow this remark; think about what happens if you have the images registered, and then shift the muzzle perpendicular to the stripes). But if we have an estimate of the translation from the 8×8 version, we do not need to search a large range of translations to register the 16×16 version. We need to look only at four translations: $2 * m_8, 2 * n_8$; $2 * m_8 + 1, 2 * n_8$; $2 * m_8, 2 * n_8 + 1$; and $2 * m_8 + 1, 2 * n_8 + 1$. The same reasoning applies when going from the 16×16 version to the 32×32 version, and so on. This strategy is known as *coarse-to-fine search*.

4.2.1 The Gaussian Pyramid

With a little notation, we can write simple expressions for the layers of a Gaussian pyramid. The operator $S^{\downarrow}$ downsamples an image; in particular, the j, kth element of $S^{\downarrow}(\mathcal{I})$ is the $2j$, $2k$th element of $\mathcal{I}$. The nth level of a pyramid $P(\mathcal{I})$ is denoted $P(\mathcal{I})_n$. With this notation, we have

$$\begin{aligned} P_{\text{Gaussian}}(\mathcal{I})_{n+1} &= S^{\downarrow}(G_\sigma * * P_{\text{Gaussian}}(\mathcal{I})_n) \\ &= (S^{\downarrow} G_\sigma) P_{\text{Gaussian}}(\mathcal{I})_n) \end{aligned}$$

(where we have written G_σ for the linear operator that takes an image to the convolution of that image with a Gaussian). The finest scale layer is the original image:

$$P_{\text{Gaussian}}(\mathcal{I})_1 = \mathcal{I}.$$

TODO: make this a procedure box

Set the finest scale layer to the image
For each layer, going from next to finest to coarsest
  Obtain this layer by smoothing the next finest
  layer with a Gaussian, and then subsampling it
end

Algorithm 4.1: *Forming a Gaussian Pyramid.*

TODO: procedure box for translation registering an image to another

4.3 BUILDING MOSAICS

Here is a simple procedure to build a mosaic from a set of images $\{\mathcal{I}_1, \ldots, \mathcal{I}_N\}$, sketched in Figure 4.3. Construct a large array of pixels value "unknown" to serve as the mosaic. Choose an image – say $\mathcal{I}_1$ – to place at the center, and transfer $\mathcal{I}_1$'s pixels to the corresponding locations in the mosaic. In Figure 4.3, step I shows this. Now find an image $\mathcal{I}_k$ which overlaps with an image in the mosaic and register it to that image, then update the mosaic using its pixels. For step II in Figure 4.3 this is $\mathcal{I}_2$ which is registered to $\mathcal{I}_1$. Proceed until there aren't any images that overlap

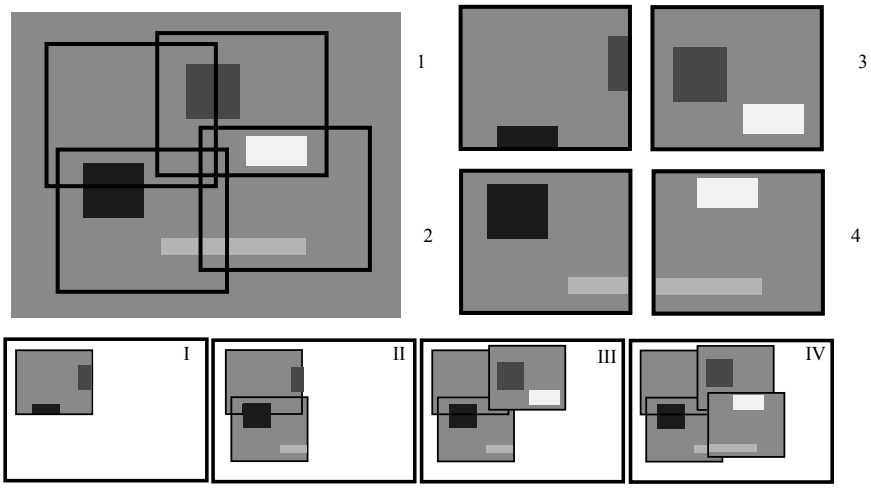

FIGURE 4.4: **Top left** shows a simple scene with the location of four images; these are shown on the **top right**. The steps of creating a mosaic are sketched in the Roman numeral panes on the **bottom**. In this case, the translation of the fourth image with respect to the first is poorly estimated; more detail in the text.

(step III, $\mathcal{I}_3$ to $\mathcal{I}_2$; step IV, $\mathcal{I}_4$ to $\mathcal{I}_3$). Now compute the pixel values in the mosaic using all the overlap information.

In some applications, the image with a good overlap will be obvious. For example, if you build a mosaic out of overhead aerial images, the images are going to be timestamped, and the next image will overlap rather well with the current image. In other cases, you can try to register coarse scale versions of the images with one another. Not all pairs will register, but those that do with a small enough translation will likely have a good overlap, and you can use this to obtain an overlapping image.

There are a variety of ways to compute a mosaic from registered images, depending on what one is trying to achieve. Many locations in the mosaic are overlapped by multiple images. At these locations, you have more than one estimate of the pixel value (one from each image that overlaps that location). One strategy is to simply average these values. There are more interesting possibilities than taking a mean. For example, imagine you want to build a mosaic that suppresses the movement of a moving object (Figure 33.2). Averaging will produce a mosaic with a blurred version of the object. Instead, collect all the values for each location, and take the median. As the figure illustrates, this will suppress the moving object. Alternatively, imagine you want to emphasize the movement; then collect all the values, and the value most unlike the median.

4.3.1 Bundle Adjustment

Our simple procedure does not produce the best possible mosaic. To see this, assume you have images $\mathcal{I}_1, \ldots, \mathcal{I}_4$, as in Figure 4.5, and you introduce them into the mosaic in that order. Write $T_{2\to 1}$ for the translation to align $\mathcal{I}_2$ with $\mathcal{I}_1$ by

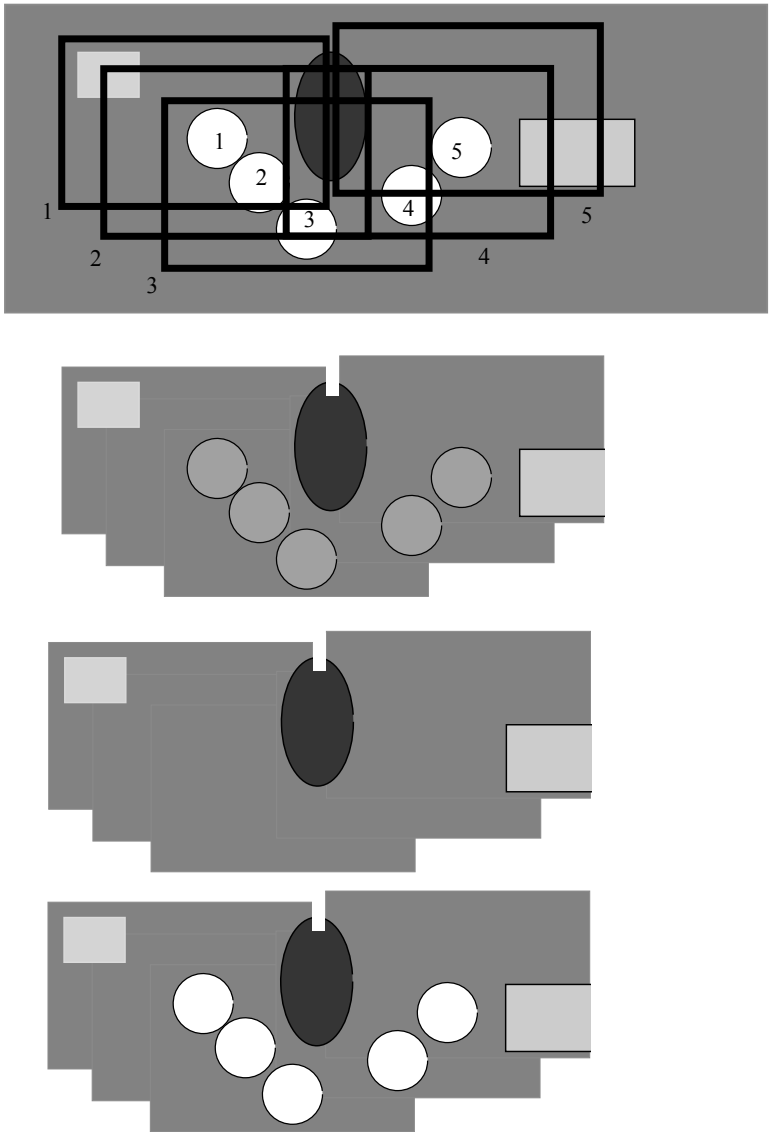

FIGURE 4.5: **Top** *shows a set of images of a simple scene with a moving object (the circle). This is at location 1 in frame 1, and so on. The background does not move. The frames are registered to one another to produce a mosaic. The value at a location in the mosaic is a summary of possibly many pixel values (one for each image that overlaps that location). Different procedures for summarization lead to quite different mosaics.* **Row 2** *shows what happens if one averages. The circles affect the average, and appear. But using a median will produce* **row 3** *– here the moving object has been suppressed, and we can see the background. Finally, using the pixel value most different from the median yields* **row 4**, *where the motion of the circle is now visible.*

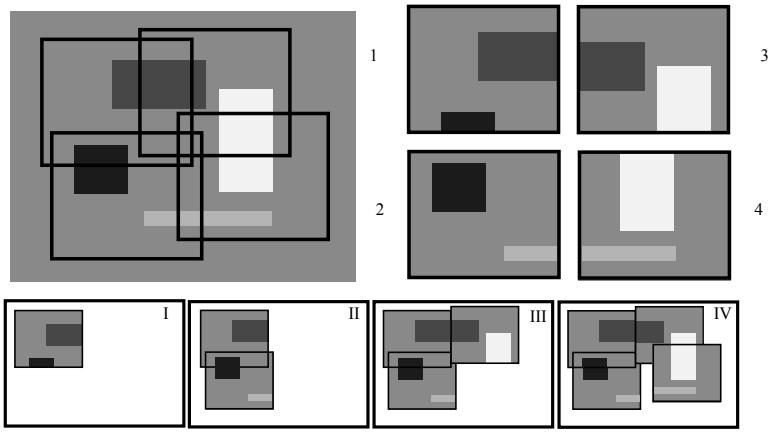

FIGURE 4.6: **Top left** *shows a simple scene with the location of four images; these are shown on the* **top right**. *The steps of creating a mosaic are sketched in the Roman numeral panes on the* **bottom**. *In this case, the translation of the fourth image with respect to the first is poorly estimated; more detail in the text.*

translating $\mathcal{I}_2$ to the right place in the mosaic coordinate system. The effect of this translation is shown in step II in Figure 4.5. Now you estimate $T_{3\rightarrow 1}$ to register $\mathcal{I}_3$ to $\mathcal{I}_1$ (Step III). Notice that the patterns in the scene have been chosen to show an exaggerated case where the registration error between $\mathcal{I}_3$ and $\mathcal{I}_1$ is likely to be large (many different horizontal translations of $\mathcal{I}_3$ will register with $\mathcal{I}_1$ well). Finally, you estimate $\mathcal{T}_{4\rightarrow 3}$ (Step IV). Notice how error has cascaded. $\mathcal{I}_3$ is poorly registered to $\mathcal{I}_1$, and $\mathcal{I}_4$ is registered to $\mathcal{I}_3$, meaning that $\mathcal{I}_4$ is poorly registered to $\mathcal{I}_1$.

Notice that this isn't necessary. Some pixels of $\mathcal{I}_4$ overlap $\mathcal{I}_2$. If these pixels contributed to the registration, $\mathcal{I}_4$ and $\mathcal{I}_3$ could be properly registered. This problem occurs quite generally – it is not just a result of an odd scene – and is often referred to as a failure of *loop closure*. This refers to the idea that if 2 is registered to 1, 3 is registered to 2, 4 is registered to 3, all the way up to N is registered to $N-1$, N may not be registered to 1 at all well – the loop does not close.

There are several procedures to prevent or control this kind of error propagation. Start by registering the images by pairs as described. The easiest strategy is now to repeat: fix all but one image, then register that image to all the others it overlaps with. This approach can work, but may be slow, because it may take a long time for improvements to propagate across all the images. The alternative, which is better but can be onerous, is to fix one image in place, then adjust all others to register in every overlap. Associate a translation $t_{i;x}, t_{j;y}$ with each image $\mathcal{I}_i$, and set $t_{1;x} = 0, t_{1;y} = 0$. Write $\mathbf{t}$ for a vector of all these translations except $t_{1;x}, t_{1;y}$, and $O_{ij}(\mathbf{t})$ for the area of the overlap between $\mathcal{I}_i$ and $\mathcal{I}_j$. Now minimize

$$\sum_{i,j \in \text{overlapping pairs of images}} \left\{ \frac{1}{O_{ij}(\mathbf{t})} \sum_{x,y \in \text{overlap}} \left[\mathcal{I}_{i;x+t_{i;x},y+t_{i;y}} - \mathcal{I}_{j;x+t_{j;x},y+t_{j;y}} \right]^2 \right\}$$

as a function of $\mathbf{t}$. This isn't a straightforward optimization problem. If you require

that $\mathbf{t}$ are integers – so that the sample points of overlapping images lie on top of one another – then searching for the right set of integer values is hard. If, instead, you treat this as a continuous optimization problem, the objective function is hard to evaluate because you will need to do a lot of bilinear interpolation. A coarse-to-fine search will work for this problem, too. While it is important to know that this difficulty is present, there is no need to resolve it in detail yet (but see Section 33.2 if you're concerned).