Tutorial 7

Gidon Rosalki

2025-12-03

Today we are going to use both dynamic programming, and network flows, as studied in algorithms. If you have not yet studied that (or have forgotten), then simply accept it as working, and feel free to cry yourself to sleep.

1 Applying a 2D Homography to Lines

Let us begin by considering a line. A line is every (x, y) that satisfies

$$ax + by + c = 0$$

As we represent 2D points in homogeneous coordinates:

$$x = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

So too we will represent lines as

$$l = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

The point x is on line l if:

$$l \Leftrightarrow l^T x = 0$$

$$\text{since } l^T x = (a\ b\ c) \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = ax + by + c = 0$$

Furthermore

$$l = p_1 \times p_2$$

Where l goes through the points p_1, p_2. This is due too

$$l^T p_1 = 0$$
$$l^T p_2 = 0$$

While homography on points is as follows:

$$x' = Hx$$

Homography on lines is

$$l' = H^{-T} l$$

This is the case since if $l^T x_1 = l^T x_2 = 0$, and $x' = Hx$, then the image line l' must satisfy the following:

$$l'^T x'_1 = 0 \implies l'^T H x_1 = 0 \implies \left(H^T l'\right)^T x_1 = 0 \tag{1}$$
$$l'^T x'_2 = 0 \implies l'^T H x_2 = 0 \implies \left(H^T l'\right)^T x_2 = 0 \tag{2}$$
$$\implies H^T l' = l \implies l' = H^{-T} l \tag{3}$$

2 Blending and Stitching

In the previous tutorial, we successfully merged images for creating panoramas, but wound up with a seam:

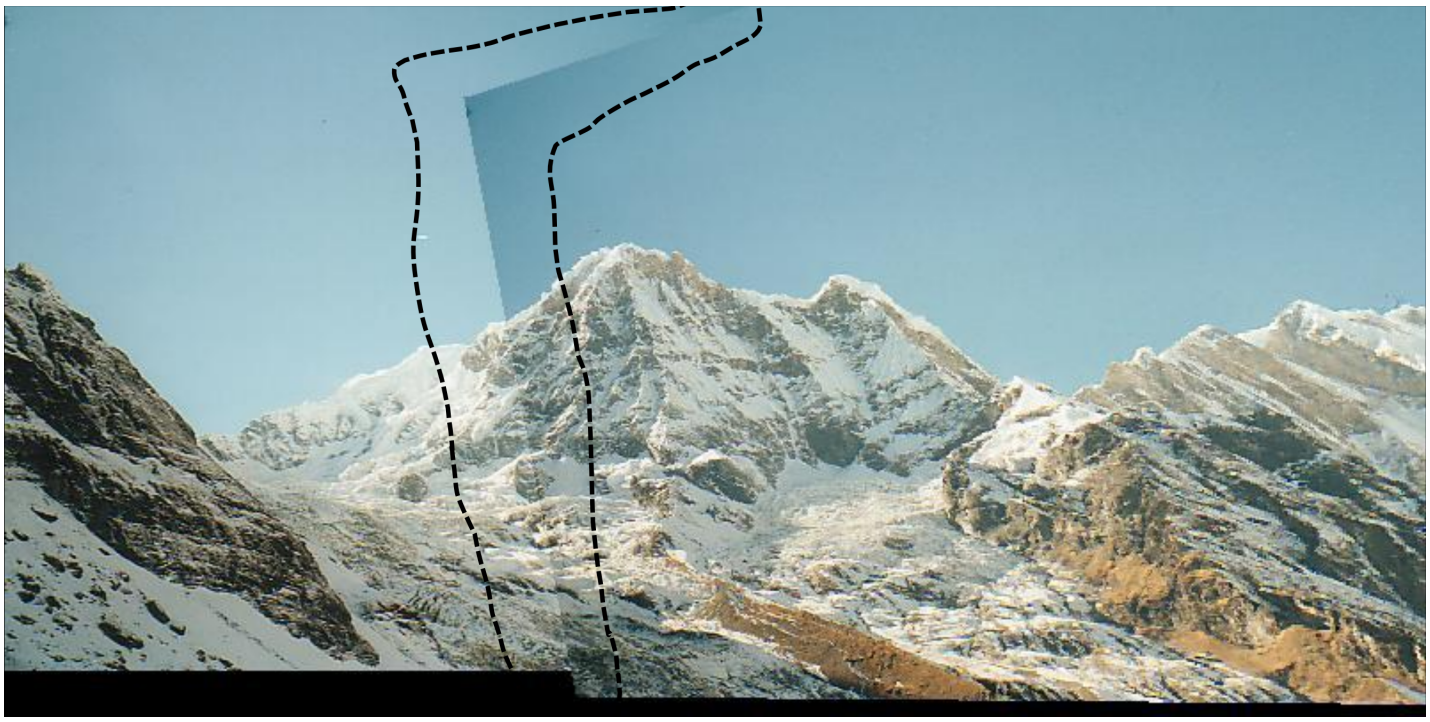

Figure 1: Seam

We (obviously) do not want that seam to be there.
To build a panorama, we need to

1. Find alignment between overlapping images
    - Choose motion transformation between images
    - (This is the translation, rotation, affine, homography)
2. Choose a compositing surface for warping
3. Warp
4. Seamlessly blend the edges

There are some further problems with creating these mosaics. Consider a non-static scene. If someone moves across the seams while they are being photographed, and so will appear in numerous parts of the image (and sometimes, only parts of himself):

Figure 2:

Stitching the images together can be a challenge for many reasons:

- Exposure differences
- Vignetting
- Blurring (due to misregistration)
- Ghosting (moving objects)

Our goal is thus to have invisible seams between the images in the panorama. This means that there should be a minimal amount of seam artefacts, avoiding the creation of edges that did not appear in the original image.

2.1 Seam location - Naive

In order to reduce problems with the edges, we take the middle strip of the image, and align based off that, and insert that into the image. This reduces the problems from vignetting on the sides of the photo. This is discussed further in Lecture 7.

2.2 Blend the transition - Feathering (alpha blending) and Pyramid blending

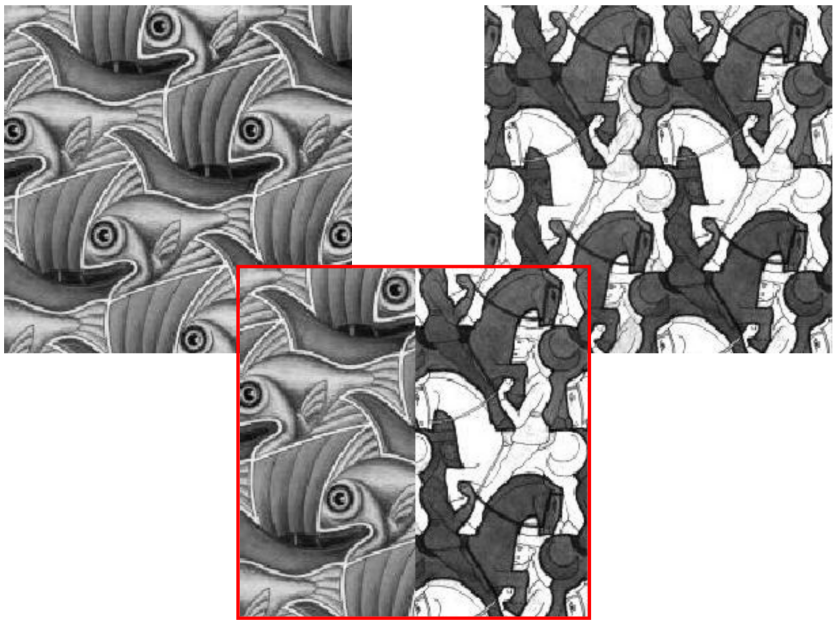

Figure 3:

This is a cut between to images, which we can see, does not convincingly merge together to the two images. We may instead mask them together, and blend them together much more smoothly with a linear transformation in the alpha (transparency) channel.

$$I_{left}(x,y) = (\alpha R, \alpha G, \alpha B)\, I_{left}(x,y) = ((1-\alpha)R, (1-\alpha)G, (1-\alpha)B)\, I_{blend} = I_{left} + I_{right} \tag{4}$$

Figure 4:

A larger window size will result in ghosting across the seam (not shown here), where a smaller window size will result in a sharp line, so this is naturally, imperfect. We may instead do pyramid blending, like we did in exercise 3, which results in much better transitions.

There is also Fourier Domain blending, where we instead take the FT of the two images, and apply the mask such that for one we only take the low frequencies (the middle), and the other we take the high frequencies, merge them together, and take the inverse Fourier to recreate the image.

2.3 Optimal Seam

What if instead we take a strip from each image, but instead of predetermining a mask, we take the optimal mask at each point? Well, let the algo begin! In order to stitch moving objects, we do **not** want to blend, but rather to *cut*. Moving images become ghosts, which looks bad. Instead of blending between the images, we may instead find the optimal seam. Considering the difference between the two images, we may take the point where there is the greatest agreement between the two images, and place the cut there, such that there should be the least difference.
Davis in 1998 suggested segmenting the mosaic, such that there is a single source image per segment, and we avoid artefacts along the boundaries.

Another application is in texture synthesis. We take a set of blocks, and place them randomly on top of each other to create a texture (like say, the ground). This will appear bad.

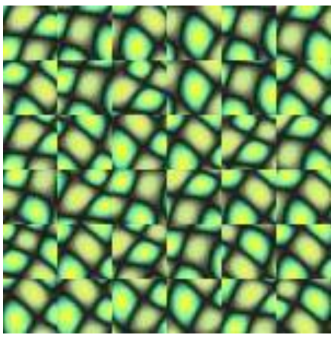

Figure 5: Random placement

We may instead restrict our neighbouring blocks to those that have the best overlap. This will still be imperfect, since just because there is good overlap, does not mean that there is a good alignment:

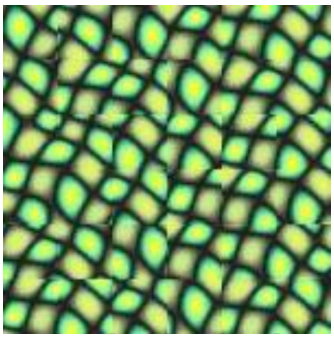

Figure 6: Constrained by overlap

A good solution is to find the minimal error, by cutting along the boundary. This will minimise the changes, since we can find the point of perfect overlap between the blocks.

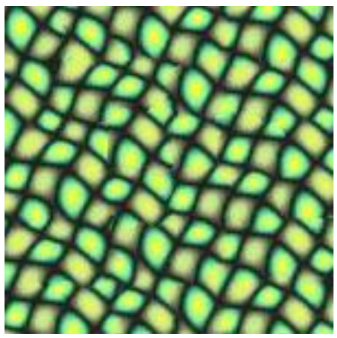

Figure 7: Boundary cut

2.3.1 Dynamic Programming

We may stitch with dynamic programming (hooray!). We can do this by scanning the image row by row, and computing a cumulative minimum squared difference E for all paths.

$$E[i,j] = e(i,j) + \min\{E[i-1,j-1], E[i-1,j], E[i-1,j+1]\}$$

$$\text{Such that } e = (I_1 - I_2)^2$$

The optimal path can be obtained by tracing back with a minimal cost from bottom to top. Dynamic programming can be computed in $O(n)$ where n is the number of pixels in the overlap area.

2.3.2 Graph Cut

Remember flow networks? Well, ready or not, it is time to return to them. Consider a weighted, directed graph, with a source, and a sink. We want to find the largest flow, that can be sent from the source, to the sink, along the edges (the weights are the capacities). Wee also have the min cut problem, such that the edges are the weights. Here we

want to find the cheapest cut in the graph, the cut that completely separates the source from the sink, with the lowest weight. In 1962, Ford and Fulkerson established that the maximum flow saturates the edges edges along the minimum cut, and provided the first polynomial time algorithm for the globally optimum solution.

So how does this relate to us in image processing? Well, we want to find the cut between two images, with the lowest cost. If we can define a cost between pairs of pixels, we may say that there is infinite flow between each pixel, and the source / sink, and find the minimum cut in the photo. The selected path will then run between pairs of pixels.

Let the graph nodes be the pixels $p = (x, y)$. The graph edges are the 4 adjacency relations between each pair of adjacent pixels. The edge weights (flow capacities) are defined to be the colour differences between the edge pixels in both images:

$$W(p_1, p_2, A, B) = ||A(p_1) - = B(p_1)|| + ||A(p_2) - B(p_2)||$$

Where p_1, p_2 are two adjacent pixel locations, and $A(p_1)$ and $B(p_1)$ are the pixel colours at location p_1 in pictures A and B respectively. The source and target are pixels that we define explicitly to be taken from either A or B. The cut location determines the seam between the two images.

There are two cases for the edge weights:

1. The pair of pixels in image B has similar values to the pair in image A. As a result, the weight is very small, and the cut "prefers" this edge.

2. The pair of pixels in image B has different values from the pair in image A. Thus the weight is large, and the cut "avoids" this edge.

Note that since we are cutting *between* the pixels, we do not need to alpha blend, or pyramid, or anything, since we are blending between the pixels.

To sum this together, the simple seam location and blending is less suitable when there is a misalignment, or moving objects. In contrast, the optimal seam search is less suitable when there are differences in global intensity.