Advanced Graphics and Image Processing - Lecture notes

Rafał Mantiuk

Michaelmas term 2026/27

1 Contrast- and gradient-based methods

Many problems in image processing are easier to solve or produce better results if operations are not peformed directly on image pixel values but on differences between pixels. Instead of altering pixels, we can transform an image into gradient field and then edit the values in the gradient field. Once we are done with editing, we need to reconstruct an image from the modified gradient field.

A few examples of gradient-based methods are shown in Figures 1 and 2.

In one common case such differences between pixels represent gradients: for image I, a gradient at a pixel location (x, y) is computed as:

$$\nabla I_{x,y} = \begin{bmatrix} I_{x+1,y} - I_{x,y} \\ I_{x,y+1} - I_{x,y} \end{bmatrix}. \tag{1}$$

The equation above is obviously a discrete approximation of a gradient, as we are dealing with discrete pixel values rather than a continous function. This particular approximation is called forward difference, as we take the difference between the next and current pixel. Other choices include backward differences (current minus previous pixel) or central differences (next minus previous pixel).

Once a gradient field is computed, we can start modifying it. Usually better effects are achieved if the magnitude of gradients is modified and the orientation of each gradient remains unchanged. This can be achieved by

(a) Original image

(b) Details enhanced

(c) Cartoonized image

Figure 1: Two examples of gradient-based processing. Texture details in the original image were enhanced to produce the result shown in (b). Contrast was removed everywhere except at the edges to produced a cartoonized image in (c).

multiplying gradients by the gradient editing function $f()$:

$$G_{x,y} = \nabla I_{x,y} \cdot \frac{f\left(||\nabla I_{x,y}||\right)}{||\nabla I_{x,y}|| + \epsilon} \tag{2}$$

where $||\cdot||$ operator computes the magnitude (norm) of the gradient and ϵ is a small constant that prevents division by 0.

We try to reconstruct pixel values, which would result in a gradient field that is the closest to our modifed gradient field $G = [G^{(x)} \quad G^{(y)}]'$. In particular, we can try to minimize the squared differences between gradients in actual image and modified gradients:

$$\arg\min_{I} \sum_{x,y} \left[\left(I_{x+1,y} - I_{x,y} - G^{(x)}_{x,y}\right)^2 + \left(I_{x,y+1} - I_{x,y} - G^{(y)}_{x,y}\right)^2 \right], \tag{3}$$

(a) Naive image copy & paste (b) Gradient-domain copy & paste

Figure 2: Comparison of naive and gradient domain image copy & paste.

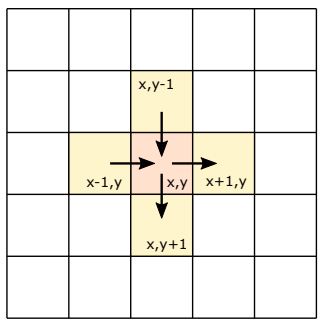

Figure 3: When using forward-differences, a pixel with the coordinates (x, y) is referred to in at most four partial derivates, two along x-axis and two along y-axis.

where the summation is over the entire image. To minimize the function above, we need to equate its partial derivatives to 0. As we optimze for pixel values, we need to compute partial derivates with respect to $I_{x,y}$. Fortunately, most terms in the sum will become 0 after differentiation, as they do not contain the differentiated variable $I_{x,y}$. For a given pixel (x, y), we need to consider only 4 partial derivates: two belonging to the pixel (x, y), x-derivative for the pixel on the left $(x-1, y)$ and y-derivative for the pixel in the top $(x, y-1)$, as shown in Figure 3. This gives us:

$$\frac{\delta F}{\delta I_{x,y}} = -2(I_{x+1,y} - I_{x,y} - G^{(x)}_{x,y}) - 2(I_{x,y+1} - I_{x,y} - G^{(y)}_{x,y}) + \tag{4}$$

$$2(I_{x,y} - I_{x-1,y} - G^{(x)}_{x-1,y}) + 2(I_{x,y} - I_{x,y-1} - G^{(y)}_{x,y-1}) \,. \tag{5}$$

After rearanging the terms and equating $\frac{\delta F}{\delta I_{x,y}}$ to 0, we get:

$$I_{x-1,y} + I_{x+1,y} + I_{x,y-1} + I_{x,y+1} - 4I_{x,y} = G^{(x)}_{x,y} - G^{(x)}_{x-1,y} + G^{(y)}_{x,y} - G^{(y)}_{x,y-1} \,. \quad (6)$$

In these few steps we derived a discrete Poisson equation, which can be found in many engineering problems. The Poisson equation is often written as:

$$\nabla^2 I = \mathrm{div} G \,, \quad (7)$$

where $\nabla^2 I$ is the discrete Laplace operator:

$$\nabla^2 I_{x,y} = I_{x-1,y} + I_{x+1,y} + I_{x,y-1} + I_{x,y+1} - 4I_{x,y} \,, \quad (8)$$

and $\mathrm{div}G$ is the divergence of the vector field:

$$\mathrm{div} G_{x,y} = G^{(x)}_{x,y} - G^{(x)}_{x-1,y} + G^{(y)}_{x,y} - G^{(y)}_{x,y-1} \,. \quad (9)$$

We can also write the equation using discrete convolution operators:

$$I * \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = G^{(x)} * \begin{bmatrix} 0 & 1 & -1 \end{bmatrix} + G^{(y)} * \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix} . \quad (10)$$

Note that the covolution flips the order of elements in the kernel, thus the row and column vectors on the right hand side are also flipped.

When equation 6 is satisfied for every pixel, it forms a system of linear equations:

$$A \cdot \begin{bmatrix} I_{1,1} \\ I_{2,1} \\ \dots \\ I_{N,M} \end{bmatrix} = b \quad (11)$$

Here we represent an image as a very large column vector, in which image pixels are stacked column-after-column (in an equivalent manner they can be stacked row-after-row). Every row of matrix A contains the Laplace operator for a corresponding pixel. But the matrix also needs to account for the boundary conditions, that is handle pixels that are at the image edge and therefore do not contain neighbour on one of the sides. Matrix A for a tiny

3x3 image looks like this:

$$A = \begin{bmatrix} -2 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -3 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -3 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -3 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -3 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -2 \end{bmatrix} \tag{12}$$

Obviously, the matrix is enormous for normal size images. However, most matrix elements are 0, so it can be easily stored using a sparse matrix representation. Note that only the pixel in the center of the image (5th row) contains the full Laplace operator; all other pixels are missing neighbours so the operator is adjusted accordingly. Accounting for all boundary cases is probably the most difficult and error-prone part in formulating gradient-field reconstruction problem. The column vector b corresponds to the right hand side of equation 6.

2 Solving linear system

There is a large number of methods and software libraries, which can solve a sparse linear problem given in Equation 11. The Poisson equation is typically solved using multi-grid methods, which iteratively update the solution at different scales. Those, however, are rarther difficult to implement and tailored to one particular shape of a matrix. Alternatively, the solution can be readily found after transformation to the frequency domain (discrete cosine transform). However, such a method does not allow introducing weights, importance of which will be discussed in the next section. Finally, conjugate gradient and biconjugate gradient [1, sec. 2.7] methods provide a fast-converging iterative method for solving sparse systems, which can be very memory efficient. Those methods require providing only a way to compute multiplication of the matrix A and its transpose with an arbitrary vector. Such operation can be realized in an arbitrary way without the need to store the sparse matrix (which can be very large even if it is sparse). The conjugate gradient requires fewer operations than the biconjugate gradient method, but

(a) Uniform weights

(b) Higher weights at low contrast

Figure 4: The solution of gradient field reconstruction often contain "pinching" artefacts, such as shown in figure (a). The artefacts can be avoided if small gradient magnitudes are weighted more than large magnitudes.

it should be used only with positive definite matrices. Matrix A is not positive definite so in principle the biconjugate gradient method should be used. However, in practice, conjugate gradient method converges equally well.

3 Weighted reconstruction

An image resulting from solving Equation 11 often contains undesirable "pinching" artefacts, such as those shown in Figure 4a. Those artefacts are inherent to the nature of gradient field reconstruction — the solution is just the best approximation of the desired gradient field but it hardly ever exactly matches the desired gradient field. As we minimize squared differences, tiny inaccuracies for many pixels introduce less error than large inaccuracies for few pixels. This in turn introduces smooth gradients in the areas, where the desired gradient field is inconsistent (cannot form an image). Such gradients produce "pinching" artefacts.

The problem is that the error in reconstructed gradients is penalized the same regardless of whether the value of the gradient is small or large. This is opposite to how the visual system perceives differences in color values: we are more likely to spot tiny difference between two similar pixel values than the same tiny difference between two very different pixel values. We could account for that effect by introducing some form of non-linear metric, however, that would make our problem non-linear and non-linear problems are in general much slower to solve. However, the same can be achieved by introducing weights to our objective function:

$$\arg\min_{I} \sum_{x,y} \left[w_{x,y}^{(x)} \left(I_{x+1,y} - I_{x,y} - G_{x,y}^{(x)} \right)^2 + w_{x,y}^{(y)} \left(I_{x,y+1} - I_{x,y} - G_{x,y}^{(y)} \right)^2 \right] , \tag{13}$$

where $w_{x,y}^{(x)}$ and $w_{x,y}^{(y)}$ are the weights or importance we assign to each gradient, for horizontal and vertical partial derivatives respectively. Usually the weights are kept the same for both orientations, i.e. $w_{x,y}^{(x)} = w_{x,y}^{(y)}$. To account for the contrast perception of the visual system, we need to assign a higher weight to small gradient magnitudes. For example, we could use the weight:

$$w_{x,y}^{(x)} = w_{x,y}^{(y)} = \frac{1}{||G_{x,y}|| + \epsilon} \tag{14}$$

where $||G_{x,y}||$ is the magnitude of the desired (target) gradient at pixel (x, y) and ϵ is a small constant (0.0001), which prevents division by 0.

4 Matrix notation

We could follow the same procedure as in the previous section and differentiate Equation 13 to find the linear system that minimizes our objective. However, the process starts to be tedious and error-prone. As the objective functions gets more and more complex, it is worth switching to the matrix notation. Let us consider first our original problem without the weights $w_{x,y}$, which we will add later. Equation 3 in the matrix notation can be written as:

$$\arg\min_{I} \left|\left| \begin{bmatrix} \nabla_x \\ \nabla_y \end{bmatrix} I - \begin{bmatrix} G^{(x)} \\ G^{(y)} \end{bmatrix} \right|\right|^2 . \tag{15}$$

In the equation I, $G^{(x)}$ and $G^{(y)}$ are stacked column vectors, representing columns of the resulting image or desired gradient field. The square brackets

denote vertical concatenation of the matrices or vectors. Operator $||\cdot||^2$ is the L_2-norm, which squares and sums the elements of the resulting column vector. ∇_x and ∇_y are differential operators, which are represented as $N \times N$ matrices, where N is the number of pixels. Each row of those sparse matrices tells us which pixels need to be subtracted from one another to compute forward gradients along horizontal and vertical directions. For a tiny 3×3 pixel image those operators are:

$$\nabla_x = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{16}$$

$$\nabla_y = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{17}$$

Note that the rows contain all zeros for pixels on the boundary, for which no gradient can be computed: the last column of pixels for ∇_x and the last row of pixels for ∇_y.

Equation 15 is in the format $||Ax - b||^2$, which can be directly solved by some sparse matrix libraries, such as `SciPy.sparse` or the "\" operator in matlab Matlab. However, to reduce the size of the sparse matrix and to speed-up computation, it is worth taking one more step and transform the least-square optimization into a linear problem. For overdetermined systems, such as ours, the solution of the optimization problem:

$$\arg\min_x ||Ax - b||^2 \tag{18}$$

can be found by solving a linear system (refer to Section 5 if you are unsure about this step):

$$A^\top A x = A^\top b\,. \tag{19}$$

Note that $^\top$ denotes a matrix transpose and $A^\top A$ is a square matrix. If we replace A and b with the corresponding operators and gradient values from our problem, we get the following linear system:

$$\begin{bmatrix}\nabla_x^\top & \nabla_y^\top\end{bmatrix} \begin{bmatrix}\nabla_x \\ \nabla_y\end{bmatrix} I = \begin{bmatrix}\nabla_x^\top & \nabla_y^\top\end{bmatrix} \begin{bmatrix}G^{(x)} \\ G^{(y)}\end{bmatrix}, \tag{20}$$

which, after multiplying stacked matrices, gives us:

$$\left(\nabla_x^\top \nabla_x + \nabla_y^\top \nabla_y\right) I = \nabla_x^\top G^{(x)} + \nabla_y^\top G^{(y)}\,. \tag{21}$$

Weights can be added to such a system by inserting a sparse diagonal matrix W. For simplicity we use the same weights for vertical and horizontal derivatives:

$$\left(\nabla_x^\top W \nabla_x + \nabla_y^\top W \nabla_y\right) I = \nabla_x^\top W G^{(x)} + \nabla_y^\top W G^{(y)}\,. \tag{22}$$

The above operations can be performed using a sparse matrix library (or SciPy/Matlab), thus saving us effort of computing operators by hand.

There is still one problem remaining: our equation does not have a unique solution. This is because the target gradient field contains relative information about differences between pixels, but it does not say what the absolute value of the pixels should be. For that reason, we need to constrain the absolute value, for example by ensuring that a value of a first reconstructed pixel is the same as in the source image (I_{src}):

$$\begin{bmatrix}1 & 0 & \dots & 0\end{bmatrix} I = I_{src}(1,1)\,. \tag{23}$$

If we denote the vector on the left-hand side of the equation as C, the final linear problem can be written as:

$$\left(\nabla_x^\top W \nabla_x + \nabla_y^\top W \nabla_y + C^\top C\right) I = \nabla_x^\top W G^{(x)} + \nabla_y^\top W G^{(y)} + C^\top I_{src}(1,1)\,. \tag{24}$$

The resulting equation can be solved using a sparse solver in Python or Matlab.

5 Moore–Penrose inverse

If you unsure how the optimization from Eq. 18 becomes a linear system in Eq. 19, you can easily derive it. We can rewrite our optimised function:

$$\arg\min_{x} ||Ax - b||^2 \tag{25}$$

as:

$$\begin{aligned} ||Ax - b||^2 &= (Ax - b)^\top (Ax - b) = x^\top A^\top Ax - x^\top A^\top b - b^\top AX + b^\top b \\ &= x^\top A^\top Ax - 2\, x^\top A^\top b + b^\top b \end{aligned} \tag{26}$$

The function will be minimized, when all its partial derivatives are 0:

$$\nabla_x ||Ax - b||^2 = 2\, A^\top Ax - 2\, A^\top b = 0\,, \tag{27}$$

hence

$$A^\top Ax = A^\top b\,. \tag{28}$$

References

[1] S. A. Teukolsky, B. P. Flannery, W. H. Press, and W. T. Vetterling. *Numerical recipes in C.* Cambridge University Press, Cambridge, vol. 2 edition, 1992.