16. Deblurring Gaussian blur

16.1 Deblurring

To discuss an application where really high order Gaussian derivatives are applied, we study the deblurring of Gaussian blur by inverting the action of the diffusion equation, as originally described by Florack et al. [Florack et al. 1994b, TerHaarRomeny1994a].

Gaussian degradation, as deblurring with a Gaussian kernel is also coined, occurs in a large number of situations. E.g.: the point-spread-function of the human lens e.g. has a close to Gaussian shape (for a 3 mm pupil its standard deviation is about 2 minutes of arc); the atmospheric turbulence blurs astronomical images in a Gaussian fashion; and the thickness profile of thin slices made by modern spiral-CT scanners is about Gaussian, leading to Gaussian blur in a multiplanar reconstructed image such as in the sagittal or coronal plane. Surely, deblurring is of immediate importance for image restoration.

Due to the central limit theorem, stating that a concatenation of any type of transformation gives a Gaussian shape when the number of sequential transformations goes to infinity, many physical processes involving sequential local degradations show a close-to-Gaussian blurring.

There is an analytical solution for the inversion of Gaussian blur. But the reconstruction can never be exact. Many practical solutions have been proposed, involving a variety of enhancing filters (e.g. high-pass or Wiener) and Fourier methods. Analytical methods have been proposed by Kimia, Hummel and Zucker [Kimia1986, Kimia1993, Hummel1987] as well as Réti [Réti 1995a]. They replaced the Gaussian blur kernel by a highly structured Toeplitz matrix and deblurred the image by the analytical inverse of this matrix. Martens deblurred images with polynomial transforms [Martens 1990].

16.2 Deblurring with a scale-space approach

If we consider the stack of images in the scale-space, we see the images gradually blur when we increase the scale. Indeed, the diffusion equation $\frac{\partial L}{\partial t} = \frac{\partial^2 L}{\partial x^2} + \frac{\partial^2 L}{\partial y^2}$ tells us that the change ∂L in L when we increase the scale t with a small increment ∂t is equal to the local value of the Laplacian $\frac{\partial^2 L}{\partial x^2} + \frac{\partial^2 L}{\partial y^2}$. From the early chapters we remember that a scale-space is infinitely differentiable due to the regularization properties of the observation process.

A natural step is to look what happens if we go to negative scales. Due to the continuity we are allowed to construct a Taylor expansion of the scale-space in any direction, including the negative scale direction. We create a Taylor series expansion of our scale-space $L(x, y, t)$ with *Mathematica*'s command **Series**, e.g. to third order around the point $t = 0$:

```
<< FrontEndVision`FEV`;
```

```
L =.; Series[L[x, y, t], {t, 0, 3}]
```

$$L[x, y, 0] + L^{(0,0,1)}[x, y, 0]\, t + \frac{1}{2} L^{(0,0,2)}[x, y, 0]\, t^2 + \frac{1}{6} L^{(0,0,3)}[x, y, 0]\, t^3 + O[t]^4$$

The derivatives to t are recognized as e.g. $L^{(0,0,1)}$. It is not possible to directly calculate the derivatives to t. But here the diffusion equation rescues us. We can replace the derivative of the image to scale with the Laplacian of the image, and that can be computed by application of the Gaussian derivatives on the image. Higher order derivatives to t have to be replaced with the repeated Laplacian operator. E.g., the second order derivative to t has to be replaced by the Laplacian of the Laplacian. To shorten our notations, we define Δ to be the Laplacian operator:

```
Δ := (∂x,x # + ∂y,y #) &
```

Here the construct of a 'pure function' in *Mathematica* is used: e.g. `(#^3) &` is a function without name that raises its argument to the third power. The repeated Laplacian operator is made with the function **Nest**:

```
Nest[f, x, 3]
```

```
f[f[f[x]]]
```

We now look for each occurrence of a derivative to t. This is the term `L^(0,0,n_) [x, y, 0]` where `n_` is anything, named `n`, the order of differentiation to t. The underscore `_` or `Blank[]` is the *Mathematica* representation for any single expression). With *Mathematica*'s powerful technique of *pattern matching* (`/.` is the **Replace** operator) we replace each occurrence of `L^(0,0,n_) [x, y, 0]` with an `n`-times-nested Laplacian operator as follows:

```
expr = Normal[Series[L[x, y, t], {t, 0, 3}]] /.
  L^(0,0,n_) [x, y, 0] :> Nest[Δ, L[x, y, 0], n]
```

$$L[x, y, 0] + t\,(L^{(0,2,0)}[x, y, 0] + L^{(2,0,0)}[x, y, 0]) + \frac{1}{2} t^2\,(L^{(0,4,0)}[x, y, 0] + 2\, L^{(2,2,0)}[x, y, 0] + L^{(4,0,0)}[x, y, 0]) + \frac{1}{6} t^3\,(L^{(0,6,0)}[x, y, 0] + 3\, L^{(2,4,0)}[x, y, 0] + 3\, L^{(4,2,0)}[x, y, 0] + L^{(6,0,0)}[x, y, 0])$$

To get the formulas better readable we apply the function **shortnotation** (defined in chapter 6, section 5), which replaces the formal notations of the derivatives by a shortform expressed in a (luminance) function `L` with appropriate subscripts through *pattern matching*:

```
expr // shortnotation
```

$$L[x, y, 0] + t\,(L_{xx} + L_{yy}) + \frac{1}{2} t^2\,(L_{xxxx} + 2\, L_{xxyy} + L_{yyyy}) + \frac{1}{6} t^3\,(L_{xxxxxx} + 3\, L_{xxxxyy} + 3\, L_{xxyyyy} + L_{yyyyyy})$$

High order of spatial derivatives appear. The highest order in this example is 6, because we applied the Laplacian operator 3 times, which itself is a second order operator. With *Mathematica* we now have the machinery to make Taylor expansions to any order, e.g. to 5:

```
expr = Normal[Series[L[x, y, t], {t, 0, 5}]] /.
  L^(0,0,n_)[x, y, 0] :> Nest[Δ, L[x, y, 0], n];
expr // shortnotation
```

$$L[x, y, 0] + t\,(L_{xx} + L_{yy}) + \frac{1}{2}\,t^2\,(L_{xxxx} + 2\,L_{xxyy} + L_{yyyy}) +$$
$$\frac{1}{6}\,t^3\,(L_{xxxxxx} + 3\,L_{xxxxyy} + 3\,L_{xxyyyy} + L_{yyyyyy}) +$$
$$\frac{1}{24}\,t^4\,(L_{xxxxxxxx} + 4\,L_{xxxxxxyy} + 6\,L_{xxxxyyyy} + 4\,L_{xxyyyyyy} + L_{yyyyyyyy}) + \frac{1}{120}\,t^5$$
$$(L_{xxxxxxxxxx} + 5\,L_{xxxxxxxxyy} + 10\,L_{xxxxxxyyyy} + 10\,L_{xxxxyyyyyy} + 5\,L_{xxyyyyyyyy} + L_{yyyyyyyyyy})$$

No matter how high the order of differentiation, the derivatives can be calculated using the multi-scale Gaussian derivatives. So, as a final step, we replace by pattern matching (**/.**) the spatial derivatives in the formula above by Gaussian derivatives (**HoldForm** assures we see just the formula for **gD[]**, of which evaluation is 'hold'; **ReleaseHold** removes the hold):

```
corr =
 expr /. Derivative[n_, m_, 0][L][x, y, a_] → HoldForm[gD[im, n, m, 1]]
```

$$t\,(gD[im, 0, 2, 1] + gD[im, 2, 0, 1]) +$$
$$\frac{1}{2}\,t^2\,(gD[im, 0, 4, 1] + 2\,gD[im, 2, 2, 1] + gD[im, 4, 0, 1]) + \frac{1}{6}\,t^3$$
$$(gD[im, 0, 6, 1] + 3\,gD[im, 2, 4, 1] + 3\,gD[im, 4, 2, 1] + gD[im, 6, 0, 1]) +$$
$$\frac{1}{24}\,t^4\,(gD[im, 0, 8, 1] + 4\,gD[im, 2, 6, 1] +$$
$$6\,gD[im, 4, 4, 1] + 4\,gD[im, 6, 2, 1] + gD[im, 8, 0, 1]) +$$
$$\frac{1}{120}\,t^5\,(gD[im, 0, 10, 1] + 5\,gD[im, 2, 8, 1] + 10\,gD[im, 4, 6, 1] +$$
$$10\,gD[im, 6, 4, 1] + 5\,gD[im, 8, 2, 1] + gD[im, 10, 0, 1]) + L[x, y, 0]$$

Because we *de*blur, we take for $t = \frac{1}{2}\,\sigma^2$ a negative value, given by the estimated amount of blurring σ_{est} we expect we have to deblur. However, applying Gaussian derivatives on our image increases the inner scale with the scale of the applied operator, i.e. blurs it a little necessarily. So, if we calculate our repeated Laplacians say at scale $\sigma_{\text{operator}} = 4$, we need to deblur the effect of both blurrings. Expressed in t, the total deblurring 'distance' amounts to $t_{\text{deblur}} = -\frac{\sigma^2_{\text{est}} + \sigma^2_{\text{operator}}}{2}$.
We assemble our commands in a single deblurring command which calculates the amount of correction to be added to an image to deblur it:

```
deblur[im_, σest_, order_, σ_] := Module[{expr}, Δ = ∂x,x # + ∂y,y # &;
  expr = Normal[Series[L[x, y, t], {t, 0, order}]] /.
     L_^(0,0,l_)[x_, y_, t_] :> Nest[Δ, L[x, y, t], l] /. t → -(σest^2 + σ^2)/2;
  Drop[expr, 1] /. L^(n_,m_,0)[x, y, t_] → HoldForm[gD[im, n, m, σ]]]
```

and test it, e.g. for first order:

```
im =.; deblur[im, 2, 1, σ]
```

$$\frac{1}{2}\,(-4 - \sigma^2)\,(gD[im, 0, 2, \sigma] + gD[im, 2, 0, \sigma])$$

It is a well known fact in image processing that subtraction of the Laplacian (times some constant depending on the blur) sharpens the image. We see here that this is nothing else than the first order result of our deblurring approach using scale-space theory. For higher order deblurring the formulas get more complicated and higher derivatives are involved:

```
deblur[im, 2, 3, σ]
```

$$\frac{1}{2}\left(-4-\sigma^2\right)\left(\text{gD}[\text{im}, 0, 2, \sigma] + \text{gD}[\text{im}, 2, 0, \sigma]\right) + \frac{1}{8}\left(-4-\sigma^2\right)^2$$
$$\left(\text{gD}[\text{im}, 0, 4, \sigma] + 2\,\text{gD}[\text{im}, 2, 2, \sigma] + \text{gD}[\text{im}, 4, 0, \sigma]\right) + \frac{1}{48}\left(-4-\sigma^2\right)^3$$
$$\left(\text{gD}[\text{im}, 0, 6, \sigma] + 3\,\text{gD}[\text{im}, 2, 4, \sigma] + 3\,\text{gD}[\text{im}, 4, 2, \sigma] + \text{gD}[\text{im}, 6, 0, \sigma]\right)$$

We generate a test image blurred with $\sigma = 2$ pixels:

```
im = Import["mr128.gif"][[1, 1]]; DisplayTogetherArray[
 ListDensityPlot /@ {im, blur = gDf[im, 0, 0, 2]}, ImageSize -> 360];
```

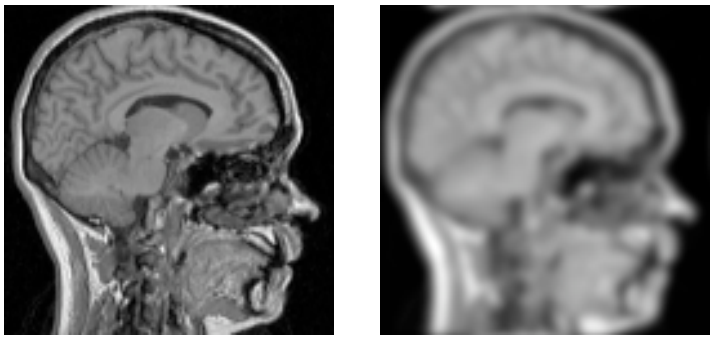

Figure 16.1 Input image for deblurring, blurred at $\sigma = 2$ pixels., resolution 128^2 .

We try a deblurring for orders 4, 8, 16 and 32 (fig. 16.2 next page): A good result. Compare with figure 16.1. *Mathematica* is reasonably fast: the deblurring to 32^{nd} order involved derivatives up to order 64 (!), in a polynomial containing 560 calls to the **gD** derivative function.

The 4 calculations take together somewhat more than one minute for a 128^2 image on a 1.7 GHz 512 MB Pentium 4 under Windows 2000 (the 32^{nd} order case took 50 seconds). This counts the occurrences of **gD** in the 32^{nd} order deblur polynomial, i.e. how many actual convolutions were needed:

```
dummy =.; Length[Position[deblur[dummy, 2, 32, 4], gD]]
```

```
560
```

```
Remove[p];
p[i_] := ListDensityPlot[blur + ReleaseHold[deblur[blur, 2, i, 4]],
  PlotLabel → "order = " <> ToString[i]];
DisplayTogetherArray[{{p[4], p[8]}, {p[16], p[32]}}, ImageSize -> 450];
```

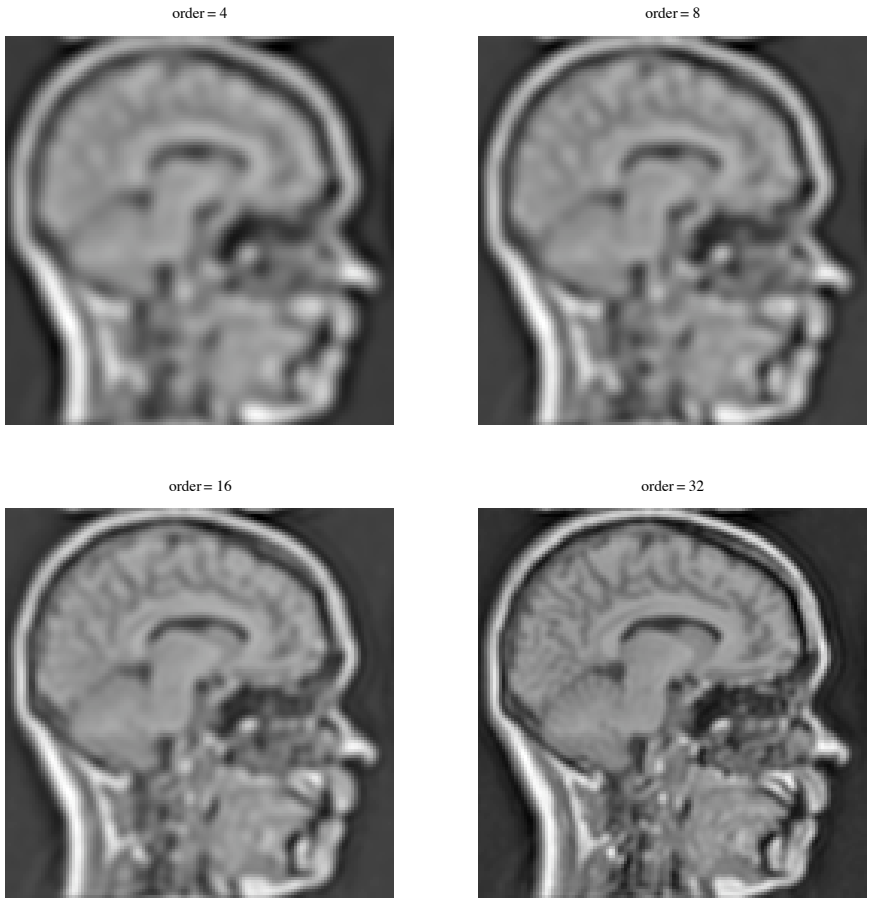

Figure 16.2 Deblurring of a blurred image (128^2 pixels, $\sigma_{blur} = 2$ pixels, left column) with different orders of approximation. The 32^{nd} order (bottom right) result comes close to the original (figure 16.1, left).

16.3 Less accurate representation, noise and holes

The method is reasonably robust to the accuracy or representation of the data. Of course, it is essential to retain as much information as possible during the blurring process. Close to precise representation (as high precision real floating point numbers) was the case in the above example. When we store the image to disk as a typical unsigned byte per pixel representation, we throw away much information. We can study the effect of such round-off by rounding each pixelvalue of the blurred image (making them integers), and do the same deblurring again:

```
roundedblur = Round[blur]; Block[{$DisplayFunction = Identity},
 p = Table[corr = deblur[roundedblur, 2, 2^i, 4] // ReleaseHold;
   ListDensityPlot[roundedblur + corr,
    PlotLabel -> "order = " <> ToString[2^i]], {i, 2, 5}]];
Show[GraphicsArray[Partition[p, 2]], ImageSize -> 450];
```

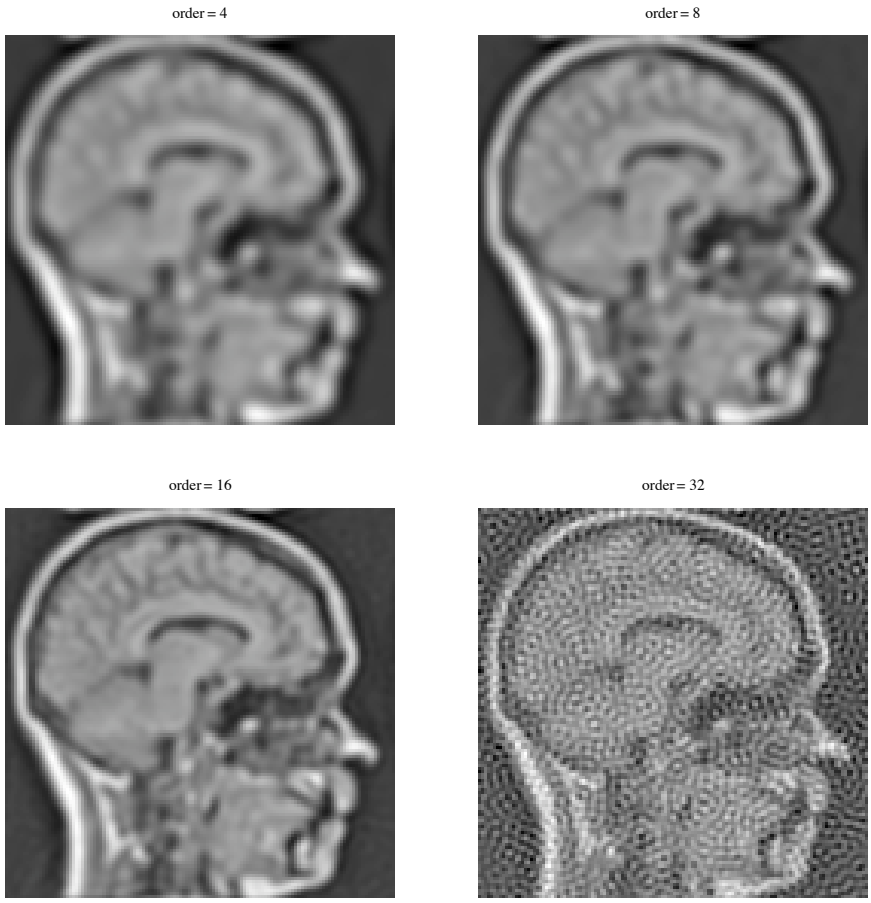

Figure 16.3 Deblurring results when the blurred image is stored as integers (intensity range of this particular image is [2-186]). Note that only the deblur results are shown. The deblurring order is indicated with each result.

Clearly the deblurring now fails for the very high order, but the results are still good till 16^{th} order.

Noise is a disaster. When we add Gaussian distributed noise with zero mean and a standard deviation of 5 intensity units, we get the following results:

```
<< Statistics`ContinuousDistributions`;
noisyblur = blur + Table[Random[NormalDistribution[0, 5]], {128}, {128}];
Block[{$DisplayFunction = Identity},
  p1 = ListDensityPlot[noisyblur, PlotLabel -> "noisyblur"];
  p2 = Table[corr = deblur[noisyblur, 2, 2^i, 4] // ReleaseHold;
    ListDensityPlot[noisyblur + corr,
     PlotLabel -> "order = " <> ToString[2^i]], {i, 2, 4}]];
Show[GraphicsArray[Prepend[p2, p1]], ImageSize -> 470];
```

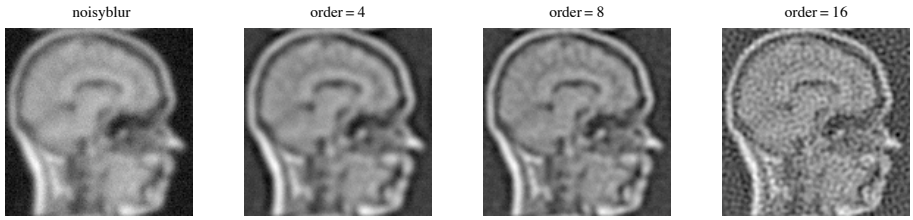

Figure 16.4 Deblurring results when the blurred image is disturbed by Gaussian additive noise (mean = 0, $\sigma_{\text{intensity}} = 5$). The deblurring order is indicated with each result.

And to conclude, we study the effect of 25 random pixels being 'blanked out', i.e. set to zero:

```
coords = Table[Random[Integer, {1, 128}], {50}, {2}];
holesblur = ReplacePart[blur, 0, coords];
Block[{$DisplayFunction = Identity},
  p1 = ListDensityPlot[holesblur];
  p2 = Table[corr = deblur[holesblur, 2, 2^i, 4] // ReleaseHold;
    ListDensityPlot[holesblur + corr,
     PlotLabel -> "order = " <> ToString[2^i]], {i, 2, 4}]];
Show[GraphicsArray[Prepend[p2, p1]], ImageSize -> 470];
```

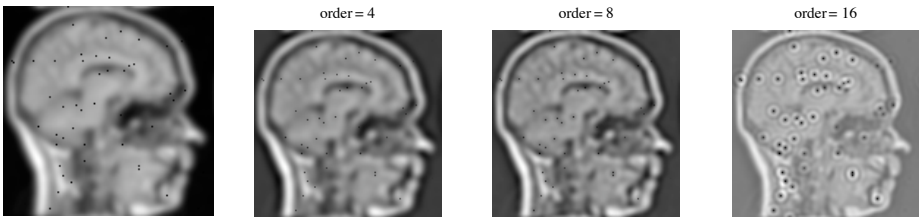

Figure 16.5 Deblurring results when the blurred image is disturbed by setting a random selection of 25 pixels to zero. The deblurring order is indicated with each result. Note that up to order 8 the 'blanked' points reconstruct well. At order 16 an overshoot occurs.

▲ Task 16.1 Experiment with deblurring images that are blurred with another kernel than the Gaussian.

▲ Task 16.2 Experiment with blurred images from an external source, e.g. find unsharp speed ticket camera images on the internet, digitize your unsharp home pictures, etc.

▲ Task 16.3 Motion blur may be simulated with anisotropic Gaussian blur, i.e. where the σ is rather different for the x and y direction. It may also be at any angle (see also chapter 19 where we discuss Gaussian kernels at arbitrary directions). Make such a blurred test image, and come up with a deblurring scheme for it.

▲ Task 16.4 In chapter 21 we discuss nonlinear diffusion equations. After having studied this chapter, it is interesting to consider how these nonlinear diffusion equations might be applied in the framework presented in this chapter, and what is the type of degradation .

16.4 Summary of this chapter

The regularization property of the Gaussian kernel makes the scale-space continuous, which means infinitely differentiable in both the spatial as the scale domain. It was proposed by Florack to expand the scale-space of a blurred image into the negative scale direction by means of a Taylor expansion. The high order derivatives to scale in this expansion can be expressed in spatial Laplacians of the image, due to the constraint of the isotropic diffusion equation. *Mathematica* turns out to be an efficient tool to do the analytic calculations of the high order Taylor expansion polynomial, in which the derivatives can be replaced by scaled Gaussian derivatives. We show some examples to real high order.

Deblurring is instable, and can only be carried out analytically when no data is lost, for example through finite intensity representation (8 bit), noise of other pixel errors. The message of this chapter is that the taking of very high order derivatives is feasible, that computer algebra is a suitable mean for implementing these calculations, and gives an example of deblurring from Gaussian blur.