Deblurring Gaussian Blur

Deblurring with a scale-space approach

- **Initialization**

In[1]:=
```
<< MathVisionTools`
SetOptions[RasterPlot, Frame → False];
```

- **Deblurring**

In the scale-space the images gradually blur when we increase the scale.

The diffusion equation $\frac{\partial L}{\partial t} = \frac{\partial^2 L}{\partial x^2} + \frac{\partial^2 L}{\partial y^2}$ governs the process.

A scale-space is infinitely differentiable due to the regularization properties of the observation process.

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:

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

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

The derivatives to *t* are recognized as e.g. $L^{(0,0,1)}$. It is not possible to directly calculate the derivatives to *t*. 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 orders derivatives to *t* have to be replaced with the repeated Laplacian operator Δ.

In[5]:=
```
Δ := (∂x,x#1 + ∂y,y#1) &
```

In[6]:=
```
Δ[f[x, y]]
```

Out[6]=
$$f^{(0,2)}[x, y] + f^{(2,0)}[x, y]$$

The repeated Laplacian operator is made with the function **Nest**:

In[7]:=
```
Nest[f, x, 3]
```

Out[7]=
$$f[f[f[x]]]$$

With *pattern matching* we replace all derivatives of *L* with respect to *t* with the nested Laplacian operator Δ:

In[26]:=
```
expr =
 Normal[Series[L[x, y, t], {t, 0, 3}]] /. L^(0,0,n_)[x, y, 0] :> Nest[Δ, L[x, y, 0], n]
```

Out[26]=
$$L[x, y, 0] + t\left(L^{(0,2,0)}[x, y, 0] + L^{(2,0,0)}[x, y, 0]\right) + \frac{1}{2} t^2 \left(L^{(0,4,0)}[x, y, 0] + 2 L^{(2,2,0)}[x, y, 0] + L^{(4,0,0)}[x, y, 0]\right) + \frac{1}{6} t^3 \left(L^{(0,6,0)}[x, y, 0] + 3 L^{(2,4,0)}[x, y, 0] + 3 L^{(4,2,0)}[x, y, 0] + L^{(6,0,0)}[x, y, 0]\right)$$

In order to get the formulas better readable for humans, we apply pattern matching again: we change the complex notations of derivatives into a more compact representation, where a higher order derivative is indicated by a list of dimensional indices:

In[24]:=
```
short[expr_] := expr /. Derivative[n_, m_, l_][L][x_, y_, z_] ->
  SubscriptBox[L, Table["x", {n}] <> Table["y", {m}] <> Table["z", {l}]] //
   DisplayForm
```

In[27]:=
```
expr // short
```

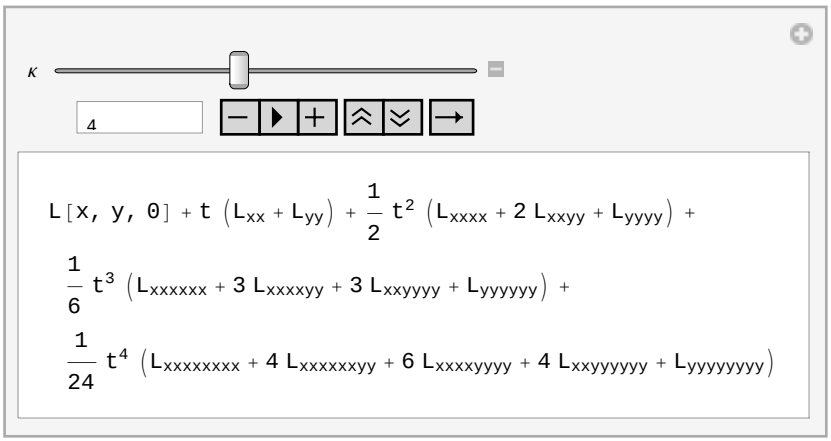
Out[27]//DisplayForm=

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

Indeed, 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 8:

In[11]:=
```
Manipulate[expr = Normal[Series[L[x, y, t], {t, 0, κ}]] /.
    L^(0,0,n_)[x, y, 0] :> Nest[Δ, L[x, y, 0], n] // short, {κ, 1, 8, 1}]
```

Out[11]=



$$L[x, y, 0] + t\left(L_{xx} + L_{yy}\right) + \frac{1}{2} t^2 \left(L_{xxxx} + 2\, L_{xxyy} + L_{yyyy}\right) + \frac{1}{6} t^3 \left(L_{xxxxxx} + 3\, L_{xxxxyy} + 3\, L_{xxyyyy} + L_{yyyyyy}\right) + \frac{1}{24} t^4 \left(L_{xxxxxxxx} + 4\, L_{xxxxxxyy} + 6\, L_{xxxxyyyy} + 4\, L_{xxyyyyyy} + L_{yyyyyyyy}\right)$$

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

In[45]:=
```
corr = expr /. Derivative[n_, m_, 0][L][x, y, a_] → HoldForm[gD[im, n, m, 1]]
```

Out[45]//DisplayForm=

$$L[x, y, 0] + t\left(L_{xx} + L_{yy}\right) + \frac{1}{2} t^2 \left(L_{xxxx} + 2\, L_{xxyy} + L_{yyyy}\right) + \frac{1}{6} t^3 \left(L_{xxxxxx} + 3\, L_{xxxxyy} + 3\, L_{xxyyyy} + L_{yyyyyy}\right) + \frac{1}{24} t^4 \left(L_{xxxxxxxx} + 4\, L_{xxxxxxyy} + 6\, L_{xxxxyyyy} + 4\, L_{xxyyyyyy} + L_{yyyyyyyy}\right)$$

Because we *de*blur, we take for $t = \frac{1}{2}\sigma^2$ a negative value, given by the amount of blurring $\sigma_{\text{estimated}}$ 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 $t_{\text{deblur}} = \frac{\sigma^2_{\text{estimated}} + \sigma^2_{\text{operator}}}{2}$. We assemble our commands in a single deblurring command which calculates the amount of correction to be added to an image to deblur it:

```
In[13]:= deblur[im_, σest_, order_, σ_] :=
           Module[{expr},
             Δ=D[#1,{x,2}]+D[#1,{y,2}]&;
             expr = Normal[Series[L[x, y, t], {t, 0, order}]]/.
               Derivative[0, 0, l_][L_][x_, y_, t_] :>
             Nest[Δ, L[x, y, t], l] /. t -> -(σest^2+σ^2)/2;
             Drop[expr,1]/.Derivative[n_,m_,0][L][x,y,t_] →
               HoldForm[gD[im,n,m,σ]]]
```

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

```
In[14]:= im=.; deblur[im, 2, 1, 2]

Out[14]= -4 (gD[im, 0, 2, 2] + gD[im, 2, 0, 2])
```

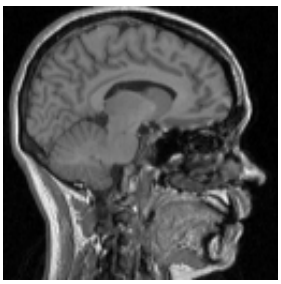
It is a well known fact in image processing that subtraction of the Laplacian (times some constant depending on the blur) sharpens the image. We see here that this is nothing else then the first order result of our deblurring approach using scale-space theory. For higher order deblurring the formulas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

```
In[15]:= deblur[im, 2, 3, 2]

Out[15]= -4 (gD[im, 0, 2, 2] + gD[im, 2, 0, 2]) +
         8 (gD[im, 0, 4, 2] + 2 gD[im, 2, 2, 2] + gD[im, 4, 0, 2]) -
         32/3 (gD[im, 0, 6, 2] + 3 gD[im, 2, 4, 2] + 3 gD[im, 4, 2, 2] + gD[im, 6, 0, 2])
```

We generate a testimage blurred with σ=2 pixels and display it both below as in a new window for later easy comparison. We read an image from the internet:

```
In[37]:= im = ImageData[ColorConvert[Import["mr128.gif"], "Grayscale"], "Byte"];

In[39]:= RasterPlot[im]
```

Out[39]=



In[40]:=
```
blur = gDf[im, 0, 0, 2];
RasterPlot[blur, ImageSize → 128]
```

Out[41]=

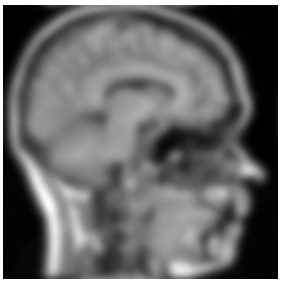
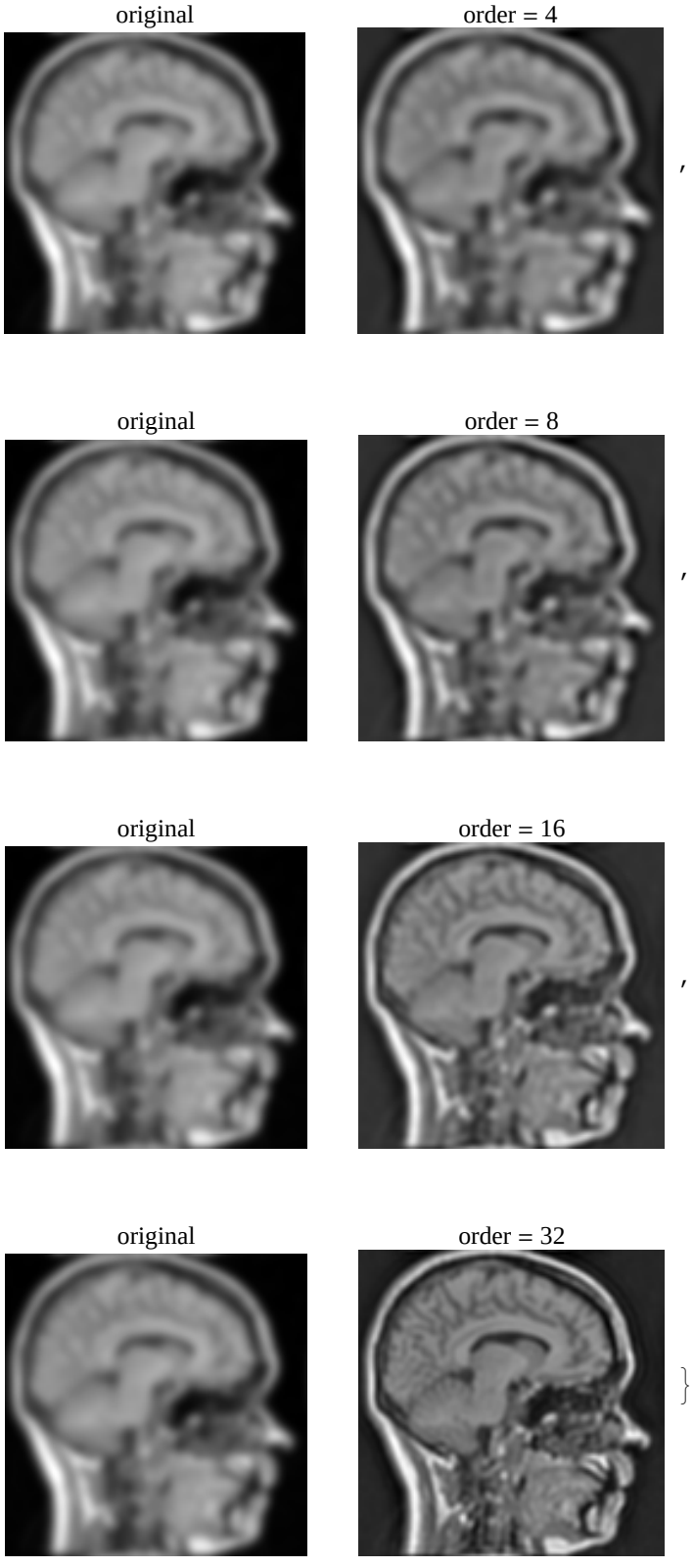


Figure 11.1. Input image for deblurring, blurred at $\sigma = 2$ pixels. Image resolution 128^2.

We try a deblurring for orders 4, 8, 16 and 32:

In[42]:=
```
Table[corr = ReleaseHold[deblur[blur, 2, 2^i, 4]];
 p1 = RasterPlot[blur, PlotLabel → "original"];
 p2 = RasterPlot[blur + corr, PlotLabel → "order = " <> ToString[2^i]];
 GraphicsGrid[{{p1, p2}}, ImageSize -> 330], {i, 2, 5}]
```

Out[42]=



Not bad.

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 above take together about 30 seconds for a 128^2 image on a 500 MHz 128 MB Pentium III under Windows XP (the 32^{nd} order case took 3.5 minutes). This counts the occurrences of **gD** in the 32^{nd} order deblur polynomial, i.e. how many actual convolutions of the image were needed:

```
In[44]:= dummy =.; Length[Position[deblur[dummy, 2, 32, 4], gD]]
Out[44]= 560
```