Proceedings International Mathematica Symposium 2005
5-8 August 2005, Perth, Australia
URL: http://internationalmathematicasymposium.org/IMS2005/

Biomedical Image Analysis

Design of Algorithms with Mathematica

Prof. Bart M. ter Haar Romeny, PhD
Eindhoven University of Technology
Eindhoven, the Netherlands
Email: B.M.terHaarRomeny@tue.nl

Introduction

Medical images are the primary source for diagnosis today. A wide variety of modalities (types of imaging) exist, each with a specific application domain. Plain X-ray, diagnostic ultrasound, tomographic ('slicing') modalities such as Computed Tomography (CT), Magnetic Resonance Imaging (MRI) and Positron Emission Tomography (PET), and the imaging of nuclear uptakes are amongst the most popular.

Images are hardly made on film anymore. The majority of modern hospitals have a fully digital department. Images are made in huge quantities. A typical hospital with several hundreds of beds may take several hundred thousands of images per year. Images are stored in digital archives (called 'Picture Archive and Communication Systems, PACS'), typically of Terabyte size, and are available on diagnostic workstations in the so-called radiological 'reading room', where the diagnosis is made by the radiologist.

Medical image analysis comes to assist in this process by means of modern computer vision techniques. The applications are modules in the diagnostic workstation, and can serve of a wide variety of clinical tasks: 3D volume imaging, quantitative analysis of anatomical parameters (vessel width, pharmacon uptake, blood velocities, etc.), computer-aided diagnosis (pattern recognition), matching of different modalities, etc. The algorithms are highly dominated by mathematical theory, which makes *Mathematica* an excellent tool for the design of such algorithms. E.g. shape, texture and optic flow (motion) detection need differential geometry techniques, while pattern recognition relies on statistical and linear algebra methods.

The author started a new group in 2001 at the Dept. of Biomedical Engineering of the Eindhoven University of Technology (TU/e), and selected *Mathematica* as the primary development base for the design of algorithms. Many student projects (PhD, MSc, BSc) have now been brought to completion, with excellent response of the students and researchers.

Biomedical Engineering is the study where essentially the aspect of engineering courses is mixed with medical and biomedical courses. The acquision of skills in mathematics, programming, statistics, etc. with physiology, anatomy, epidemiology, etc. is a good basis for the new engineer that will be working in close team cooperation with medical specialists, or in medical companies. Worldwide this is a steadily growing relatively young direction.

In Eindhoven 2 Mastertracks can be followed: Biomedical Engineering (more generic techniques, BME) and Medical Engineering (more patient related techniques, ME). In the BME track there are three directions: Biomedical Imaging and Modeling, Tissue Engineering & Cardiovascular Mechanics, and Biochemical Engineering.

This keynote gives a glimpse into this large field, with a selection of examples given as *Mathematica* notebooks. The text accompanies the powerpoint slides, shown during the keynote lecture at the International *Mathematica* Symposium 2005 in Perth, 6–10 August. The slides are available at www.bmi2.bmt.tue.nl/image-analysis/People/BRomeny/publications/IMS2005/ - IMS2005Keynote.ppt (MS powerpoint format, 65 MB).

Imaging Modalities

The most important modalities for the application of computer vision analysis applications are X-Ray, CT and MRI. A plain X-ray is a projection image of a bundle of X-rays through the patient on a digital high resolution (typically 3500×0 pixels)

detector. It is widely applied for the imaging of bone fractures, vessel anatomy (after injection of a iodine-containing contrast agent that creates clear shadows by blocking the X-rays) and trauma.

A CT scanner make slices of the patient, by rotating an X-ray tube with a banana-shaped detector on the opposite side around the patient (about 2 revolutions per second). From the measured attenuation profiles the unknown pixels of the patient's 'slice' can be calculated. Typical resolution is 512^2. Today the advent of multi-slice CT scanners, with 4 to even 64 rows of detectors, has revolutionized the field. A full high resolution lung scan of over 2000 slices can be made within a single breathhold.

MRI works with a strong magnetic field. The patient is placed in a long static (most often superconducting) magnet, typically of 1.5 Tesla (this is about 30000 times stronger than the earth's magnetic field). In this field the magnetic dipole moment of the hydrogen atoms in the patient is directed along the main magnetic field. Extra magnetic coils can create (very short) excitation pulses, to steer the dipoles in perpendicular directions. During the relaxation of the atoms to their equilibrium positions, they lose their energy as radio photons, which are measured by sensitive antennas around the patient. MRI is also a tomographic (slice-forming) technique. The electronic steering of the dipoles is very versatile, many types of images can be made, e.g. techniques exist for blood flow measurement, spectroscopic analysis of different molecules, functional activity of the brain, local body temperature, etc.

3D Volume Imaging

The stack of 2D tomographic images (typically with 1 mm resolution in-plane, 1.5–2 mm between planes) gives the full 3D information. These voxels (volume pixels) are used in 3D volume rendering systems, which are now widely available as commercial systems. The images are generated (often on the fly, real-time) by calculating the rays, emanating from a virtual light source, reflected on the patient's structure of interest, towards a virtual observer position. Modern graphics cards (GPU's, Graphical Processing Units) are now well programmable, and are becoming popular for this purpose.

The 3D imaging is only possible when the objects of interest can be clearly defined. This is done by a process called 'segmentation'. E.g. in order to only visualize the blood vessels, all other structures have to made transparent, which should be programmed in the computer with computer vision techniques.

Volume visualization is now a mature field. New recent techniques include 'virtual endoscopy', where the camera virtually 'flies through' the stack of 2D CT slices of air-filled intestines of the patient to search for possible polyps, the pre-stage of colon cancer. Diffusion Tensor Imaging is the MRI technique where the Brownian motion of water is measured. This motion, normally with 3D Gaussian distribution, becomes ellipsoidal when restricted by a tubular structure, such as a nerve fibre of muscle cell. The longest Eigenvector of the diffusion tensor, measured in every voxel, gives the primary direction of the fibre. This is an example of the advent of complex valued imaging (in this case a 3×3 tensor per voxel).

Diagnostic Workstations

Every major vendor, like Philips, Siemens, GE, Toshiba, etc. has a line of diagnostic workstations. These electronic lightboxes have completely replaced the conventional lightbox, and typically carry a wide range of applications for the radiologist and the surgeon. Often used viewing functions are the cine-loop view of stacks of images, 2D slicing in many different directions other then in the original acquisition direction, and 3D volume visualization. The format for medical images is DICOM, now universally adopted. This is a complex format, with hundreds of descriptors of the image, the patient's demographic data, the acquisition technique, security checks, compression etc. *Mathematica* has now full reading and writing capability of DICOM files on board.

Such workstations typically offer a wide range of applications. Dedicated packages are offered for 3D volume visualization, cardiac analysis, virtual endoscopy, computer-aided diagnosis (see next section), peripheral vessel analysis, functional MRI, multi-modality image registration, etc.

The 3D volume visualization and the interactive manipulation of these visualizations can also be performed on modern Graphical Processing Units (GPU's, game-cards)), which are nowadays 20–30 times more powerful then state-of-the-art CPU's. See e.g. the new start-up company 3mensio (www.3mensio.com).

Computer-aided Diagnosis

The overwhelming amount of images, and the many quantitative questions about the images, call for computer-assisted analysis. Medical image analysis is an active research field, with many dedicated large conferences (see e.g. MICCAI, www.miccai.org).

Computer-aided diagnosis is mainly targeting the major diseases, as there is the larger societal gain and market. The primary areas of interest are digital mammography (finding tumors, microcalcifications and masses in breast photographs), lung screening (finding lung tumors (often seen as 'nodules'), vessel occlusions, sarcoidosis), cardiac analysis (finding stenosis of the coronary arteries, infarct areas, cardiac output; heart diseases kill most people in the Western world) and colon cancer (finding polyps).
The methods used span all areas of mathematics and statistics. Any computer vision area has applications in CAD, like shape analysis with differential geometrical methods, shape variation and segmentation with linear algebra methods, texture analysis and pattern recognition with statistical cluster analysis methods, etc.
The goal of the designer is to create a user-friendly, effective and validated method for medical specialists to be used in daily clinical practice. There is a strong collaboration between the medical imaging industry and university research centers.

Image Analysis with *Mathematica* at TU/e

The advantages of *Mathematica* for the design of new algorithms are evident. The integration of full symbolic functionality with fast numerical capability is unique. Images are big data, and since version 4 these are easily handled. Really big images, of several hundreds of megabytes, should be processed by dedicated lower-level or even hardware supported systems. *Mathematica*'s strong point lies in the design phase.
Students easily adopt the functional programming style. At Eindhoven, we give regular small 1-day courses, when a new cohort of students enters a new course or program. Reports of internships are directly written as interactive *Mathematica* notebook, giving automatically documented code to the teacher. The interpreter mode invites to 'play with mathematics', an essential skill for mastering computer vision techniques. See for a range of examples the notebooks available at www.bmi2.bmt.tue.nl/image-analysis/Education/index.html.
The code resembles the theory in the textbooks and literature. We have encountered quite a number of examples where students implemented a paper in a few days in *Mathematica*, just by entering the formulas from the paper.
At TU/e we have a strong emphasis on design-centered learning. *Mathematica* fits excellent in this endeavor. E.g. in the second year, students get the task to analyze microscopy images from blood cells, to find the cancer cells. They do this in *Mathematica*, which is their first encounter with the program. They brainstorm to develop their own techniques, recognizing cells by their shape ('what is shape'?), size, number, color etc. Ten groups of 8 students each work for 8 weeks on the problem, and present their results in a common seminar, which is always an exciting and competitive event. This course is given high ranks by the students.
At TU/e every student receives a 50% sponsored high-end laptop (we have now over 10.000 in total on the TU/e campus). The campus premium license for *Mathematica* allows full use for students and staff alike.
We are developing a *Mathematica*-based library of advanced, multi-scale, computer vision algorithms, called *MathVision-Tools* [2]. We invite interested laboratories for a possible collaboration.

Mathematica Kernel Server

At TU/e we have installed two dedicated remote *Mathematica* kernel servers. The first one is a cluster of high-end Linux PC's, 2.8 GHz, 2 GB RAM. See math1.bmt.tue.nl. The second one is a Tyan-motherboard TX46 based 64-bit Linux *Mathematica* server with 4 AMD Opteron 848 CPU's of 3.2 GHz and 32 GB DDR 400 MHz ECC RAM (16 x 2048 MB), 8MB per CPU, all fully addressable from any CPU (for details see www.tyan.com/products/html/barebone.html.
Both systems are very popular. It enables especially BME students with older laptops to run powerful remote kernels, while the *Mathematica* frontend runs on a modest computer. The 32GB memory server is highly popular for large number-crunching tasks with PhD students. We fully exploit the use of Parallel *Mathematica*. Both systems can also be accessed from home by a secure VPN connection.

Biomimicking: Learning from Visual Perception Mechanisms

Computer-aided diagnosis aims to assist in finding pathologies. This is by far not an easy task. There exist hundreds of specific theories and applications, and the quest is for generic, robust techniques.
The human visual system has an amazing capability with respect to instantly recognizing (deviating) target structures. It is of substantial interest to study modern neurophysiological findings, in order to mimic these in a computer implementation. The

author has chosen this as one of the lead focus areas of research. See the textbook (written in *Mathematica*) "Front-End Vision and Multi-Scale Image Analysis" [1].

A key design feature seems to be the measurement of the images at a wide range of scales. This is already clear from the structure of the retina, which is designed to do exactly this. The stack of images at different scales ('blurring' levels) creates a 3D volume, which is known as a 'scale-space'. Image structure at a larger scale is more 'important' then fine structure, and with further blurring image structure gradually gets lost. When e.g. the paths of singular points (maxima, minima, saddle points) are followed over scale, we observe many annihilations (sometimes also creations). These so-called toppoints can be ordered into a hierarchical tree structure, which give a natural decomposition of a complex scene. We currently have many projects pursuing this important 'deep structure' of images.

Another important realization is that the primary visual cortex (in the back of our head) seems to contains many cells that take high order derivatives of the incoming retinal images, at least up to fourth order. Modeling this leads to a wide spectrum of differential geometric entities, called 'features'. They should be invariant to different types of geometric transformations, so are also called 'invariants'. Examples of how these may be constructed with *Mathematica* are given in Example #2.

Interestingly, when the process of blurring is mathematically known (isotropic blurring is governed by the linear diffusion equation, a linear second order PDE), the process of deblurring (which is well known to be ill-posed) can be approximated by inverting the process. A 'scale-space' approach to the deblurring of Gaussian blur is discussed in Example #3. A main strength of *Mathematica* becomes clear: the strong integration of symbolic and numerical capabilities with pattern matching allows the calculation of complex analytical results, and replacing the derivatives in the large polynomial expressions with numerical implementations of convolutions with gaussian derivative kernels.

Conclusion

For the design of complex mathematical algorithms for modern computer vision techniques *Mathematica* is ideal. We found high acceptance and learning rates of students at all levels, and we have accomplished a complete integration of *Mathematica* in our image analysis research. For an overview of the projects carried out, past and present, see the BMIA website at www.bmi2.bmt.tue.nl/image-analysis. Most notebooks as are available as *Mathematica* documents and as PDF files. The current development of *Mathematica* into a strong support of 64 bit architectures, fast numerical capabilities and more efficient handling of huge datasets is warmly welcomed. It justifies our strategy that we have chosen the right environment to have invested in.

Acknowledgement

The real work is done by the BME students. See some fruits of their labor at www.bmi2.bmt.tue.nl/image-analysis/index.html.

Markus van Almsick is a valuable and ever inspiring source of *Mathematica* knowledge and enthusiasm.

References

[1] B. M. ter Haar Romeny, *Front-End Vision & Multi-Scale Image Analysis*, Dordrecht, the Netherlands: Kluwer Academic Publishers, 2003. URL: library.wolfram.com/infocenter/Books/5514.

[2] B. M. ter Haar Romeny and M. A. van Almsick, "MathVisionTools," Wolfram Technology Conference 2004, Champaign. URL: library.wolfram.com/infocenter/Conferences/5383.

Appendix: Example Notebooks

Initialization

```
In[11]:= Off[General::"spell1"];
         Off[General::spell];
```

The Java-based function GetURL reads data from the internet.

In[13]:=
```
Needs["JLink`"];
GetURL[url_String, opts___?OptionQ] :=
    JavaBlock[
        Module[{u, stream, numRead, outFile, buf},
            InstallJava[];
            u = JavaNew["java.net.URL", url];
            (* This is where the error will show up if the URL is not valid.
               A Java exception will be thrown during openStream, which
               causes the method to return $Failed.
            *)
            stream = u@openStream[];
            If[stream === $Failed, Return[$Failed]];
            buf = JavaNew["[B", 5000]; (* 5000 is an arbitrary buffer size *)
            outFile = OpenTemporary[DOSTextFormat->False, CharacterEncoding->{}];
            While[(numRead = stream@read[buf]) > 0,
                WriteString[outFile, FromCharacterCode[If[# < 0, # + 256, #]&
                        Take[JavaObjectToExpression[buf], numRead]]]
            ];
            stream@close[];
            Close[outFile]   (* Close returns the filename *)]];

Unprotect[Get];
Get[s_String] :=
    Module[{tempFile, res},
        tempFile = GetURL[s];
        If[tempFile =!= $Failed,
            res = Get[tempFile];
            DeleteFile[tempFile];
            res,
        (* else *)
            $Failed
        ]
    ] /; StringMatchQ[s, "http://*"]
Protect[Get];

NotebookOpenURL[url_String] := NotebookOpen[GetURL[url]]
```

Read the package with the definitions for the Gaussian derivatives from the internet.

Functions from the Book "Front-End Vision & Multi-Scale Image Analysis"

In[19]:=
```
Get[ GetURL["http://www.bmi2.bmt.tue.nl/image-analysis/People/BRomeny/FEV/-
FEV.m"]]
```

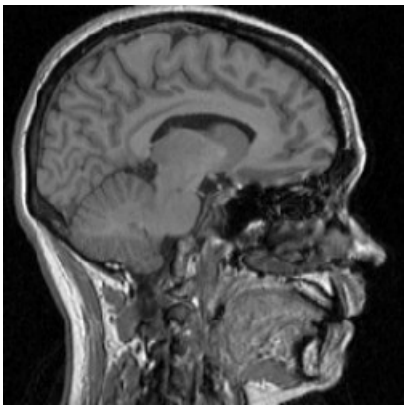
FEV package version 2.0, for *Mathematica* 5.2

FEV package version 2.0, for *Mathematica* 5.1

Example 1: Image Operations are Fast

Read an Image from the Net:

In[20]:=
```
imageFile = GetURL["http://www.bmi2.bmt.tue.nl/image-analysis/People/-
BRomeny/FEV/Images/mr256.gif"];
im = Import[imageFile,"GIF"][[1,1]];
DeleteFile[imageFile];
p1=ListDensityPlot[im];
```



The Gradient of an Image $\sqrt{\left(\frac{\partial L}{\partial x}\right)^2+\left(\frac{\partial L}{\partial y}\right)^2}$

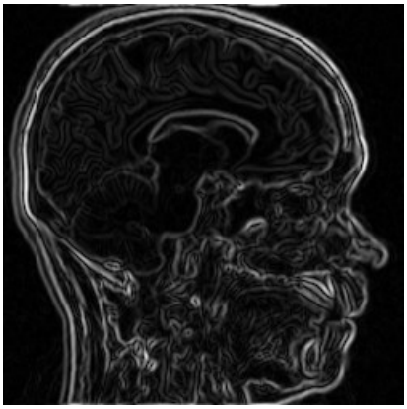
Derivatives of discrete images can be calculated in a robust way by convolution with Gaussian derivative kernels. The function gD implements this convolution by means of ListConvolve (see the package FEV.m in the initialization).

In[24]:=
```
? gD
```

```
gD[im,nx,ny,σ,options] calculates the  Gaussian derivative of a 2D image
  by spatial convolution. It is optimized for  speed by 1D convolutions
  per dimension. The image is considered cyclic in each  direction.
     im = 2D input image [List]
     nx = order of  differentiation to x [Integer]
     ny = order of differentiation to y [Integer]
      σ = scale [in pixels]
     options = <optional>kernelSampleRange: range  of kernel
  sampled in multiples of σ. Default: kernelSampleRange->{-6,6}
```

This shows the gradient (edge magnitude):

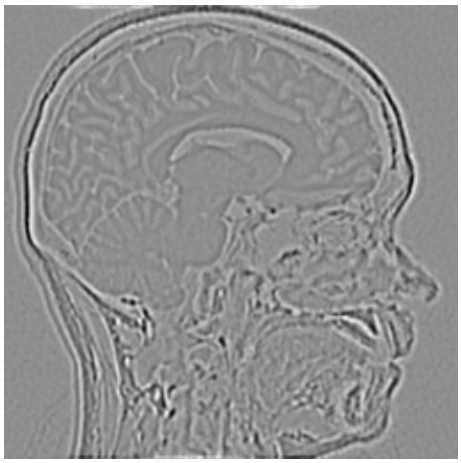
In[25]:=
```
σ = 1;
ListDensityPlot[√(gD[im, 1, 0, σ]^2 + gD[im, 0, 1, σ]^2)];
```



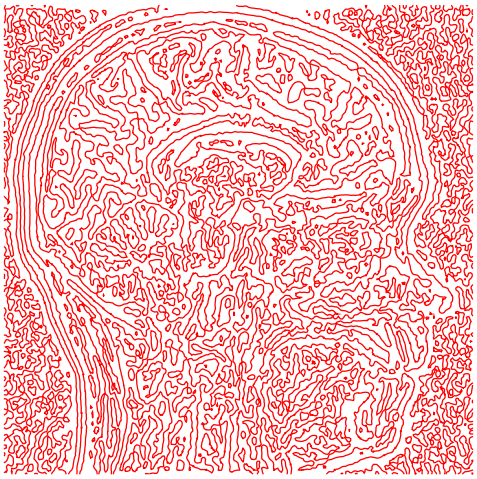
Zerocrossings of the Laplacian $L_{xx} + L_{yy}$

The maximum value of the first order edges is attained at the zerocrossings of the second derivative of the image. The proper second order derivative is the second order directional derivative in the gradient direction (see example 2). The Laplacian $L_{xx} + L_{yy}$ is a good and often used approximation, and it is easy to calculate.

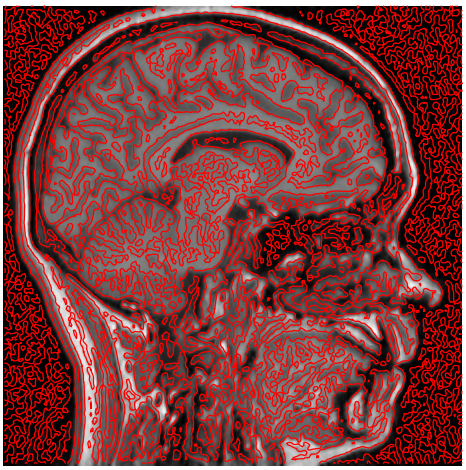
In[27]:=
```
σ = 1;
laplacian = gD[im, 2, 0, σ] + gD[im, 0, 2, σ];
ListDensityPlot[laplacian];
```



In[30]:=
```
contours = ListContourPlot[laplacian, Contours → {0}, ContourStyle → Red];
```



In[31]:=
```
Show[{p1, contours}];
```



Magnify the image to study the contours and their underlying greyvalues. Play with different scales σ.

Example 2: The Differential Structure of Images

This is an excerpt of text from the *Mathematica* textbook "Front-End Vision and Multi-Scale Image Analysis" by the author [1].

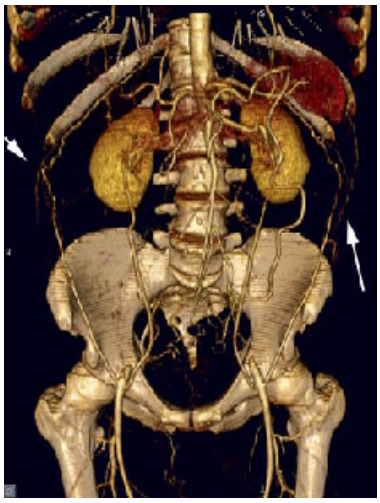
Initialization

Use the initialization of example 1.

Image Structure

The structure is described by the local multi-scale derivatives of the image.

In[32]:=
```
imgFile = GetURL["http://www.bmi2.bmt.tue.nl/image-analysis/People/-
BRomeny/FEV/Images/SpiralCTAbdomen.jpg"];
image = Import[imgFile,"JPEG"];
DeleteFile[imgFile];
Show[image];
```



An example of a need for segmentation: 3D rendering of a spiral CT acquisition of the abdomen of a patient with Leriche's syndrome.

We will use the tools of *differential geometry*.

Why is $\sqrt{\left(\frac{\partial L}{\partial x}\right)^2+\left(\frac{\partial L}{\partial y}\right)^2}$ a good edge detector?

And $\left(\frac{\partial L}{\partial y}\right)^2\frac{\partial^2 L}{\partial x^2}-2\frac{\partial L}{\partial x}\frac{\partial L}{\partial y}\frac{\partial^2 L}{\partial x\partial y}+\left(\frac{\partial L}{\partial x}\right)^2\frac{\partial^2 L}{\partial y^2}$ a good corner detector?

How do we come to such formulas?

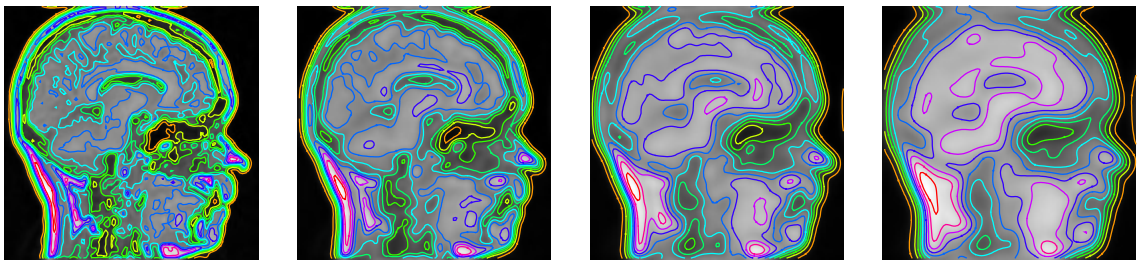
We want to detect features *invariant* to coordinate transformations, e.g. translations, rotations.

$\frac{\partial L}{\partial x}$ is *not* invariant, $\sqrt{\left(\frac{\partial L}{\partial x}\right)^2+\left(\frac{\partial L}{\partial y}\right)^2}$ is invariant.

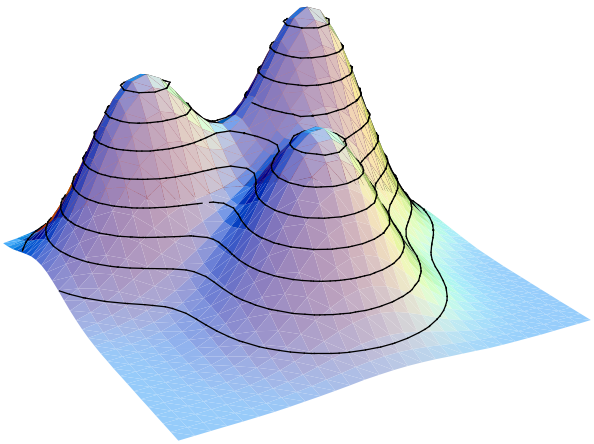
Isophotes and Flowlines

```
In[36]:= imgFile = GetURL["http://www.bmi2.bmt.tue.nl/image-analysis/People/-
         BRomeny/FEV/Images/mr128.gif"];
         im = Import[imgFile,"GIF"][[1,1]];
         DeleteFile[imgFile];
```

```
In[39]:= Block[{$DisplayFunction = Identity, dp, cp},
           dp = ListDensityPlot[gD[im, 0, 0, #]] & /@ {1, 2, 3, 4};
           cp = ListContourPlot[gD[im, 0, 0, #],
                ContourStyle → List /@ Hue /@ (.1 Range[10])] & /@ {1, 2, 3, 4};
           pa = MapThread[Show, {dp, cp}]]; Show[GraphicsArray[pa], ImageSize → 500];
```



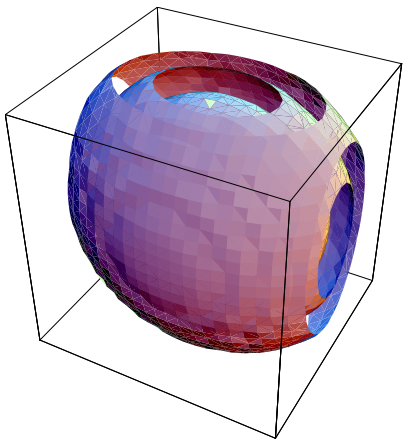
```
In[40]:= blob[x_, y_, μx_, μy_, σ_] := 1/(2 π σ^2) Exp[-((x - μx)^2 + (y - μy)^2)/(2 σ^2)];
         blobs[x_, y_] := blob[x, y, 10, 10, 4] + .7 blob[x, y, 15, 20, 4] + 0.8 blob[x, y, 22, 8, 4];
         Block[{$DisplayFunction = Identity}, p1 = Plot3D[blobs[x, y] - .00008,
             {x, 0, 30}, {y, 0, 30}, PlotPoints → 30, Mesh → False, Shading → True];
           c = ContourPlot[blobs[x, y], {x, 0, 30}, {y, 0, 30}, PlotPoints → 30, ContourShading → False];
           c3d = Graphics3D[Graphics[c][[1]] /. Line[pts_] :> (val = Apply[blobs, First[pts]];
                 Line[Map[Append[#, val] &, pts]])]];
         Show[p1, c3d, ViewPoint → {1.393, 2.502, 1.114}, ImageSize → 250];
```



Isophote on a 2D 'landscape' image of three Gaussian blobs, depicted as heightlines. The height is determined by the intensity.

Isophotes in 3D are surfaces (the 3D OpenGL viewer for *Mathematica* by Jens-Peer Kuska can be downloaded from phong.informatik.uni-leipzig.de/~kuska/mathgl3dv3):

```
In[44]:= Get["MathGL3d`OpenGLViewer`"]; isos =
          Compile[{}, 10^3 Table[Exp[-x^2/18 - y^2/8 - z^2/18], {z, -10, 10}, {y, -10, 10}, {x, -10, 10}]];
         MVListContourPlot3D[isos[], Contours → {.1, 1, 10}, ImageSize → 150];
```

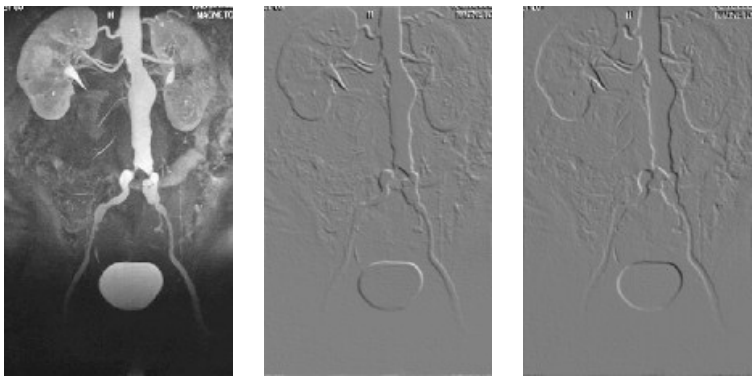


Isophotes in 3D are surfaces. Shown are the isophotes connecting all voxels with the values 0.1, 1, 10 and 100 in the discrete dataset of two neighboring 3D Gaussian blobs.

Directional Derivatives

The *directional* first order derivative in the direction $\vec{v}$ is given by $\vec{v}.\{\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\}$.

```
In[46]:= imgFile = GetURL["http://www.bmi2.bmt.tue.nl/image-analysis/People/-
         BRomeny/FEV/Images/mip147.gif"];
         im = Import[imgFile,"GIF"][[1,1]];
         DeleteFile[imgFile];

In[49]:= northeast[im_, σ_] := {-√2, -√2}.{gD[im, 1, 0, σ], gD[im, 0, 1, σ]};
         southsouthwest[im_, σ_] := {√3 / 2, 1 / 2}.{gD[im, 1, 0, σ], gD[im, 0, 1, σ]};
         DisplayTogetherArray[ListDensityPlot /@
           {im, northeast[im, 1], southsouthwest[im, 1]}, ImageSize → 300];
```



Directional derivatives. Image from the Eurorad database (www.eurorad.org), case 147.

```
In[52]:= Table[ListDensityPlot[{Cos[ϕ], Sin[ϕ]}.{gD[im, 1, 0, 1], gD[im, 0, 1, 1]}],
           {ϕ, 0, 2 π, π / 8}];
```

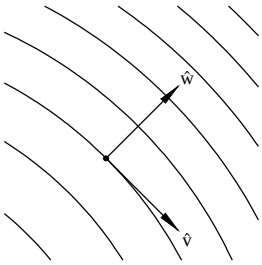
First Order Gauge Coordinates

We change from *extrinsic geometry* to *intrinsic geometry.*

We fix *in each point separately* our local coordinate frame: the gradient vector $\vec{w} = \left(\frac{\partial L}{\partial x}, \frac{\partial L}{\partial y}\right)$; the perpendicular direction is $\vec{v} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} . \vec{w} = \left(\frac{\partial L}{\partial y}, -\frac{\partial L}{\partial x}\right)$.

```
In[53]:= ContourPlot[x^2 + y^2, {y, 2, 4.5},
           {x, 2, 4.5}, Contours → Range[2, 100, 4], Epilog →
```

```
{PointSize[.02], Point[{3, 3}], Arrow[{3, 3}, {3 + .5 √2, 3 - .5 √2}],
 Arrow[{3, 3}, {3 + .5 √2, 3 + .5 √2}], Text["v̂", {3.8, 2.2}],
 Text["ŵ", {3.8, 3.8}]}, Frame → False, ImageSize → 100];
```



Local first order gauge coordinates $\{\hat{v}, \hat{w}\}$. The unit vector $\hat{v}$ is everywhere tangential to the isophote (line of constant intensity), the unit vector $\hat{w}$ is everywhere perpendicular to the isophote and points in the direction of the gradient vector.
This set of local directions is called a *gauge*, the new frame forms the *gauge coordinates*.
We want to take derivatives with respect to the gauge coordinates.

Any derivative expressed in gauge coordinates is an orthogonal invariant. E.g. it is clear that $\frac{\partial L}{\partial w}$ is the derivative in the gradient direction, and this is just the gradient itself, an invariant.

And $\frac{\partial L}{\partial v} \equiv 0$, as there is no change in the luminance as we move tangentially along the isophote, and we have chosen this direction by definition.

We can only calculate derivatives to x and y. So we need to go from $\{v, w\}$ to $\{x, y\}$.

In *Mathematica*: The frame vectors $\hat{w}$ and $\hat{v}$ are defined as

```
In[54]:= ŵ = {Lx, Ly} / √(Lx^2 + Ly^2); v̂ = ({{0, 1}, {-1, 0}}).ŵ;
```

The directional differential operators $\hat{v}.\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)$ and $\hat{w}.\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)$ are defined as:

```
In[55]:= v̂.{∂x#, ∂y#} &;
         ŵ.{∂x#, ∂y#} &;
```

The notation **(...#)&** is a 'pure function' on the argument **#**:

```
In[57]:= (#^2 + #^5) &[zz]

Out[57]= zz^2 + zz^5
```

Higher order derivatives are constructed through nesting multiple first order derivatives, as many as needed. The total transformation routine is now:

```
In[58]:= Unprotect[gauge2D]; Clear[f, L, Lx, Ly, gauge2D];

         gauge2D[f_, nv_, nw_] := Module[{Lx, Ly, v, w},
           w = {Lx, Ly} / √(Lx^2 + Ly^2); v = {{0, 1}, {-1, 0}}.w;
           Simplify[
            Nest[(v.{∂x#, ∂y#} &), Nest[(w.{∂x#, ∂y#} &), f, nw], nv] /.
             {Lx → D[f, x], Ly → D[f, y]}]];
```

where f is a symbolic function of x and y, and n_w and n_v are the orders of differentiation with respect to w resp v. Here is an example of its output: the gradient $\frac{\partial L}{\partial w}$:

```
In[60]:= Lw = gauge2D[L[x, y], 0, 2]

Out[60]= (L^(0,1)[x, y]^2 L^(0,2)[x, y] + 2 L^(0,1)[x, y] L^(1,0)[x, y] L^(1,1)[x, y] +
    L^(1,0)[x, y]^2 L^(2,0)[x, y]) / (L^(0,1)[x, y]^2 + L^(1,0)[x, y]^2)
```

Using pattern matching with the function shortnotation (see FEV.m) we get more readable output:

```
In[61]:= Lw = gauge2D[L[x, y], 0, 1] // shortnotation

Out[61]//DisplayForm=
```

$$\sqrt{L_x^2 + L_y^2}$$

```
In[62]:= Lww = gauge2D[L[x, y], 0, 2] // shortnotation

Out[62]//DisplayForm=
```

$$\frac{L_x^2 L_{xx} + 2 L_x L_{xy} L_y + L_y^2 L_{yy}}{L_x^2 + L_y^2}$$

```
In[63]:= Lv = gauge2D[L[x, y], 1, 0] // shortnotation

Out[63]//DisplayForm=
0

In[64]:= Lvv = gauge2D[L[x, y], 2, 0] // shortnotation

Out[64]//DisplayForm=
```

$$\frac{-2 L_x L_{xy} L_y + L_{xx} L_y^2 + L_x^2 L_{yy}}{L_x^2 + L_y^2}$$

This calculates the Laplacian in gauge coordinates, $L_{vv} + L_{ww}$ (what do you expect?):

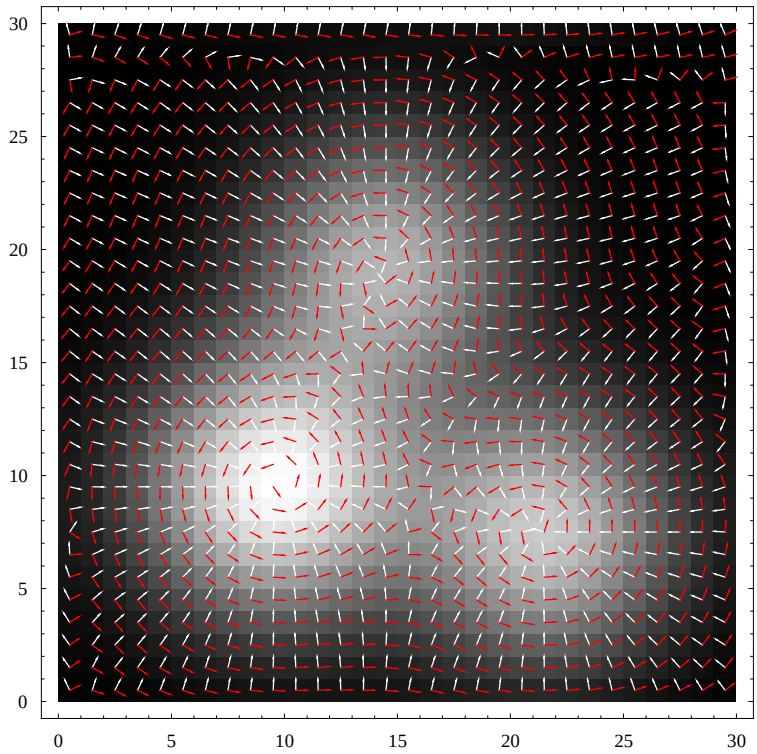
```
In[65]:= gauge2D[L[x, y], 0, 2] + gauge2D[L[x, y], 2, 0] // shortnotation

Out[65]//DisplayForm=
Lxx + Lyy
```

The next figure shows the $\{\hat{v}, \hat{w}\}$ gauge frame in every pixel of a simple 32^2 image with 3 blobs:

```
In[66]:= blob[x_, y_, μx_, μy_, σ_] := 1/(2 π σ^2) Exp[-((x - μx)^2 + (y - μy)^2)/(2 σ^2)];
         blobs[x_, y_] :=
           blob[x, y, 10, 10, 4] + .7 blob[x, y, 15, 20, 4] + 0.8 blob[x, y, 22, 8, 4];
         im = Table[blobs[x, y], {y, 30}, {x, 30}];
         Block[{$DisplayFunction = Identity, gradient, norm, σ, frame},
           norm = (# / Sqrt[#.#]) &;
         σ = 1; gradient = Map[norm,
             Transpose[{gD[im, 1, 0, σ], gD[im, 0, 1, σ]}, {3, 2, 1}], {2}];
```

```
frame = Graphics[{White, Arrow[#2 - .5, #2 - .5 + #1], Red,
      Arrow[#2 - .5, #2 - .5 + {#1[[2]], -#1[[1]]}]}] &;
ar = MapIndexed[frame, gradient / 2, {2}];
  lp = ListDensityPlot[gD[im, 0, 0, σ]]];
```

In[70]:=
```
Show[{lp, ar}, Frame → True, ImageSize → 410];
```



Due to the fixing of the gauge by removing the degree of freedom for rotation (that is why $L_v \equiv 0$), we have an important result: *every derivative to v and w is an orthogonal invariant.*
The final step is the operational implementation of the gauge derivative operators for discrete images. This is simply done by applying *pattern matching*:

- first calculate the symbolic expression
- then replace any derivative with respect to *x* and *y* by the numerical derivative gD[im,n_x,n_y,σ]
- and then insert the pixeldata in the resulting polynomial function;

as follows:

In[71]:=
```
Unprotect[gauge2DN]; Clear[gauge2DN];
gauge2DN[im_, nv_, nw_, σ_] := Module[{im0},
   gauge2D[L[x, y], nv, nw] /.
     Derivative[nx_, ny_][L_][x_, y_] → gD[im0, nx, ny, σ] /. im0 → im];
```

This *writes our numerical code automatically*. Here is the implementation for L_{vv}. If the image is not defined, we get the formula returned:

In[73]:=
```
Clear[im, σ]; gauge2DN[im, 2, 0, 2]
```

```
Out[73]= (gD[im, 0, 2, 2] gD[im, 1, 0, 2]^2 -
           2 gD[im, 0, 1, 2] gD[im, 1, 0, 2] gD[im, 1, 1, 2] +
           gD[im, 0, 1, 2]^2 gD[im, 2, 0, 2]) / (gD[im, 0, 1, 2]^2 + gD[im, 1, 0, 2]^2)
```

If the image is available, the invariant property is calculated in each pixel:

```
In[74]:= imgFile = GetURL["http://www.bmi2.bmt.tue.nl/image-analysis/People/-
         BRomeny/FEV/Images/thorax02.gif"];
         im = Import[imgFile,"GIF"][[1,1]];
         DeleteFile[imgFile];

In[77]:= DisplayTogetherArray[ListDensityPlot /@
           {im, -gauge2DN[im, 0, 1, 1], -gauge2DN[im, 2, 0, 4]}, ImageSize → 400];
```

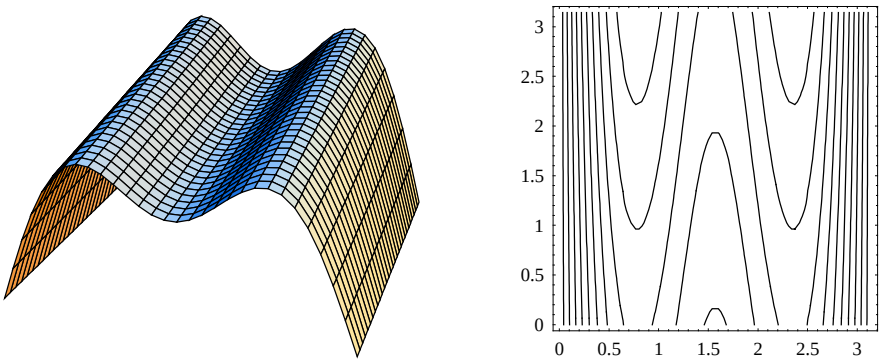
The gradient L_w (middle) and L_{vv}, the second order directional derivative in the direction tangential to the isophote (right) for a 256^2 X-thorax image at a small scale of 0.5 pixels. Note the shadow of the coins in the pocket of his shirt in the lower right.

Gauge Coordinate Invariants: Examples

Ridge detection

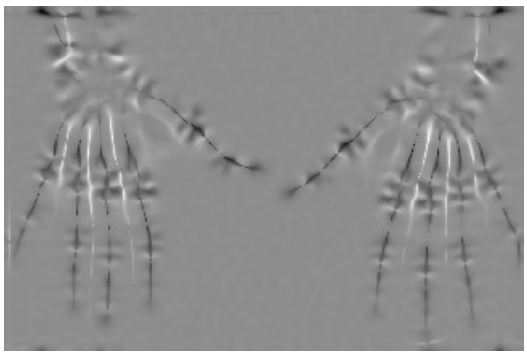
L_{vv} is a good ridge detector, since at ridges the curvature of isophotes is large.

```
In[78]:= f[x_, y_] := (Sin[x] + 1/3 Sin[3 x]) (1 + .1 y);
         DisplayTogetherArray[Plot3D[f[x, y], {x, 0, π}, {y, 0, π}],
          ContourPlot[f[x, y], {x, 0, π}, {y, 0, π}, PlotPoints → 50], ImageSize → 370];
```



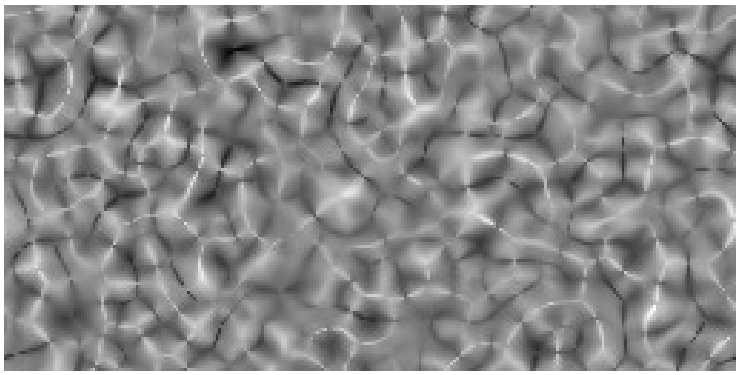
Isophotes are much more curved at the top of ridges and valleys then along the slopes of it. Left: a slightly sloping artificial intensity landscape with two ridges and a valley, at right the contours as isophotes.
Let us test this on an X-ray image of fingers and calculate L_{vv} scale $\sigma = 3$.

```
In[79]:= imgFile = GetURL["http://www.bmi2.bmt.tue.nl/image-analysis/People/-
         BRomeny/FEV/Images/hands.gif"];
         im = Import[imgFile,"GIF"][[1,1]];
         DeleteFile[imgFile];

In[82]:= Lvv = gauge2DN[im, 2, 0, 3];
         DisplayTogetherArray[ListDensityPlot /@ {im, Lvv}, ImageSize → 450];
```



The invariant feature L_{vv} is a ridge detector. Here applied on an X-ray of two hands at $\sigma = 3$ pixels. Image resolution: 361×239 pixels.
Noise has structure too. Here are the ridges of uniform white noise:

```
In[84]:= im = Table[Random[], {128}, {256}]; ListDensityPlot[gauge2DN[im, 2, 0, 4]];
```



The invariant feature L_{vv} detects the ridges in white noise here, $\sigma = 4$ pixels, image resolution: 256 x 128 pixels.

Isophote Curvature in Gauge Coordinates

Isophote curvature κ is defined as the change $w'' = \frac{\partial^2 w}{\partial v^2}$ of the tangent vector $w' = \frac{\partial w}{\partial v} = v$ in the gradient-gauge coordinate system. The definition of an isophote is: $L(v, w) =$ Constant, and $w = w(v)$. So, in *Mathematica* we implicitly differentiate the equality (**==**) to v:

```
In[85]:= Clear[v, w];
         L[v, w[v]] == Constant;
         ∂v (L[v, w[v]] == Constant)

Out[87]= w'[v] L^(0,1)[v, w[v]] + L^(1,0)[v, w[v]] == 0
```

We know that $L_v \equiv 0$ by definition of the gauge coordinates, so $w' = 0$, and the curvature $\kappa = w''$ is found by differentiating the isophote equation again and solving for w'':

```
In[88]:= Solve[∂v,v (L[v, w[v]] == Constant) /. w'[v] → 0, w''[v]]
```

$$\text{Out[88]= } \left\{\left\{w''[v] \to -\frac{L^{(2,0)}[v, w[v]]}{L^{(0,1)}[v, w[v]]}\right\}\right\}$$

So $\kappa = -\frac{L_{vv}}{L_w}$. In Cartesian coordinates we recognize the well-known formula:

```
In[89]:= im =.; κ = -gauge2D[L[x, y], 2, 0] / gauge2D[L[x, y], 0, 1]; κ // shortnotation
```

Out[89]//DisplayForm=

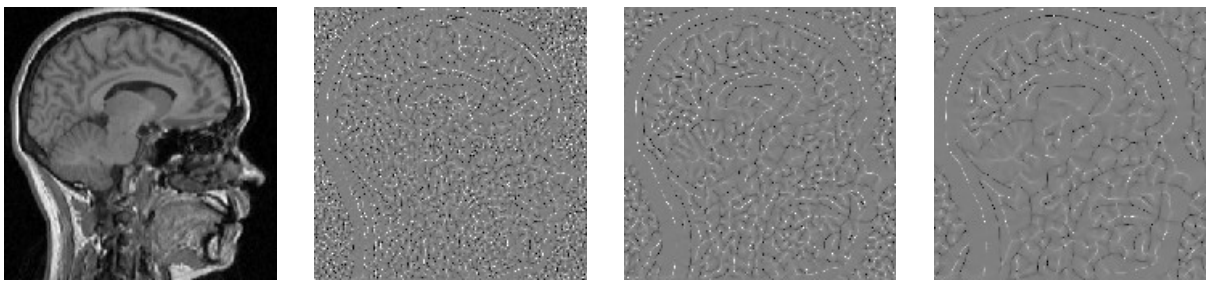
$$-\frac{-2\,L_x\,L_{xy}\,L_y + L_{xx}\,L_y^2 + L_x^2\,L_{yy}}{(L_x^2 + L_y^2)^{3/2}}$$

Here is an example of the isophote curvature at a range of scales for a sagittal MR image:

```
In[90]:= imgFile = GetURL["http://www.bmi2.bmt.tue.nl/image-analysis/People/-
         BRomeny/FEV/Images/mr256.gif"];
         im = Import[imgFile,"GIF"][[1,1]];
         DeleteFile[imgFile];
```

```
In[93]:= κplot[σ_] := ListDensityPlot[-gauge2DN[im, 2, 0, σ] / gauge2DN[im, 0, 1, σ], PlotRange → {-5, 5}];
```

```
In[94]:= DisplayTogetherArray[
           {ListDensityPlot[im], κplot[1], κplot[2], κplot[3]}, ImageSize → 600];
```



The isophote curvature κ is a rotationally and translationally invariant feature. It takes high values at extrema. Image resolution: 256^2 pixels.

Affine Invariant Corner Detection

Corners are defined as locations with high isophote curvature and high intensity gradient. An elegant reasoning for an affine invariant corner detector was proposed by Blom [Blom1991a], then a PhD student of Koenderink. We reproduce it here using *Mathematica*. Blom proposed to take the *product* of isophote curvature $-\frac{L_{vv}}{L_w}$ and the gradient L_w raised to some (to be determined) power n:

$\Theta^{[n]} = -\frac{L_{vv}}{L_w} L_w{}^n = \kappa L_w{}^n = -L_{vv} L_w{}^{n-1}$.

An obvious advantage is invariance under a transformation that changes the opening angle of the corner. Such a transformation is the *affine* transformation. An affine transformation is a *linear* transformation of the coordinate axes:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \frac{1}{a\,d - b\,c} \begin{pmatrix} a & b \\ c & d \end{pmatrix} (x \quad y) + (e\ f)$$

We omit the translation term $(e\,f)$ and study the affine transformation proper. The term $\frac{1}{ad - bc}$ is the determinant of the transformation matrix, and is called the *Jacobian*. Its purpose is to adjust the amplitude when the area changes.
A good example of the effect of an affine transformation is to study the projection of a square from a large distance. Rotation over a vertical axis shortens the *x*-axis. Changing both axes introduces a *shear*, where the angles between the sides change. The following example illustrates this by an affine transformation of a square:

```
In[95]:= square = {{0, 0}, {1, 0}, {1, 1}, {0, 1}, {0, 0}};
         affine = {{5, 2}, {0, .5}}; afsquare = affine.# & /@ square;
         DisplayTogetherArray[
           Graphics[Line[#], AspectRatio → 1] & /@ {square, afsquare}, ImageSize → 200];
```

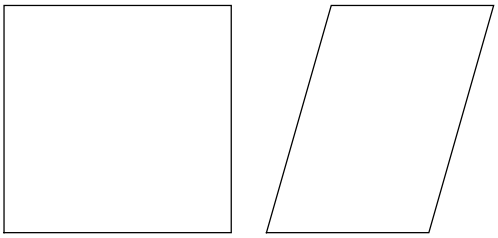


Figure 6.11. Affine transformation of a square, with transformation matrix $\begin{pmatrix} 5 & 2 \\ 0 & .5 \end{pmatrix}$ mapped on each point.

The derivatives transform as $\begin{pmatrix} \partial_{x'} \\ \partial_{y'} \end{pmatrix} = \frac{1}{ad - bc} \begin{pmatrix} a & b \\ c & d \end{pmatrix} (\partial_x \quad \partial_y)$. We put the affine transformation $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ into the definition of affinely transformed gauge coordinates:

```
In[98]:= Clear[a, b, c, d];
         gauge2Daffine[f_, nv_, nw_] := Module[{Lx, Ly, v, w, A = {{a, b}, {c, d}}},
           w = {Lx', Ly'} / Sqrt[Lx'^2 + Ly'^2]; v = {{0, 1}, {-1, 0}}.w; Simplify[
            Nest[v.(1 / Det[A] A.{∂x#, ∂y#}) &, Nest[w.(1 / Det[A] A.{∂x#, ∂y#}) &, f, nw],
             nv] /. {Lx' → (a Lx + b Ly) / Det[A], Ly' → (c Lx + d Ly) / Det[A]} /. {Lx → ∂x f, Ly → ∂y f}]];
```

The equation for the affinely distorted coordinates $-L_{v_a v_a} L_{w_a}^{\,n-1}$ now becomes:

```
In[99]:= -gauge2Daffine[L[x, y], 2, 0] gauge2Daffine[L[x, y], 0, 1]^(n-1) // Simplify //
          shortnotation
```

Out[99]//DisplayForm=

$$-\frac{\left(\frac{(a^2+c^2)\,L_x^2 + 2\,(a\,b+c\,d)\,L_x\,L_y + (b^2+d^2)\,L_y^2}{(b\,c-a\,d)^2}\right)^{\frac{1}{2}(-3+n)} (-2\,L_x\,L_{xy}\,L_y + L_{xx}\,L_y^2 + L_x^2\,L_{yy})}{(b\,c - a\,d)^2}$$

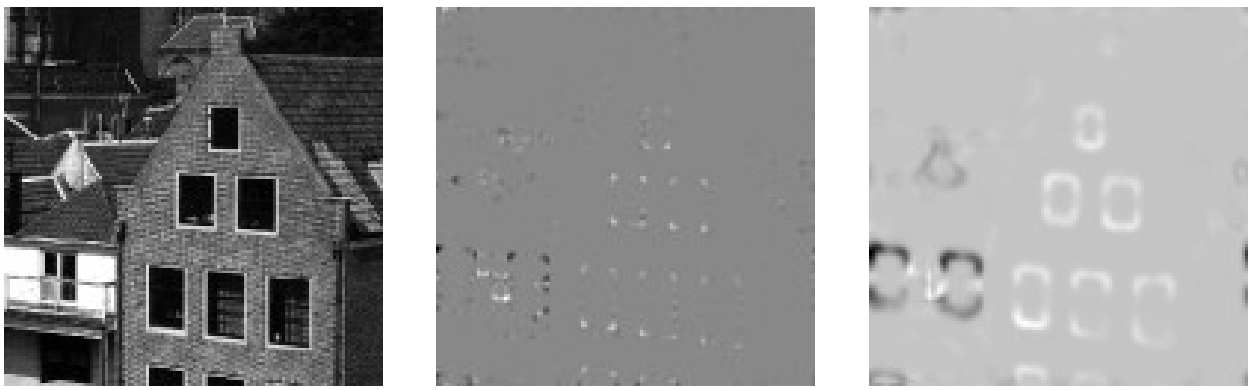
Very interesting: when $n = 3$ and for an affine transformation with unity Jacobean ($ad - bc = 1$, a so-called *special* transformation) we are independent of the parameters *a*, *b*, *c* and *d*! This is the affine invariance condition.

So the expression $\Theta = \frac{L_{vv}}{L_w} L_w^3 = L_{vv} L_w^2 = 2L_x L_{xy} L_y - L_{xx} L_y^2 - L_x^2 L_{yy}$ is an *affine invariant corner detector*. This feature has the nice property that it is not singular at locations where the gradient vanishes, and through its affine invariance it detects corners at all 'opening angles'.
We show corner detection at two scales on an image:

```
In[100]:= imgFile = GetURL["http://www.bmi2.bmt.tue.nl/image-analysis/People/-
          BRomeny/FEV/Images/Utrecht256.gif"];
          im = Import[imgFile,"GIF"][[1,1]];
          DeleteFile[imgFile];
```

```
In[103]:= im = SubMatrix[im, {1, 128}, {128, 128}];
          corner1 = gauge2DN[im, 2, 0, 1] gauge2DN[im, 0, 1, 1]^2;
          corner3 = gauge2DN[im, 2, 0, 3] gauge2DN[im, 0, 1, 2]^2;
```

```
In[106]:= DisplayTogetherArray[
            ListDensityPlot /@ {im, corner1, corner3}, ImageSize → 500];
```



Corner detection with the $L_{vv} L_w^2$ operator. Left: original image, dimensions 128^2. Middle: corner detection at σ = 1 pixel; right: corner detection at σ = 3 pixels. Isophote curvature is signed, so note the positive (convex, light) and negative (concave, 2dark) corners.

Second Order Structure

The second order structure of the intensity landscape is rich.

```
In[107]:= s = Series[L[x, y], {x, 0, 2}, {y, 0, 2}] // Normal // shortnotation
```

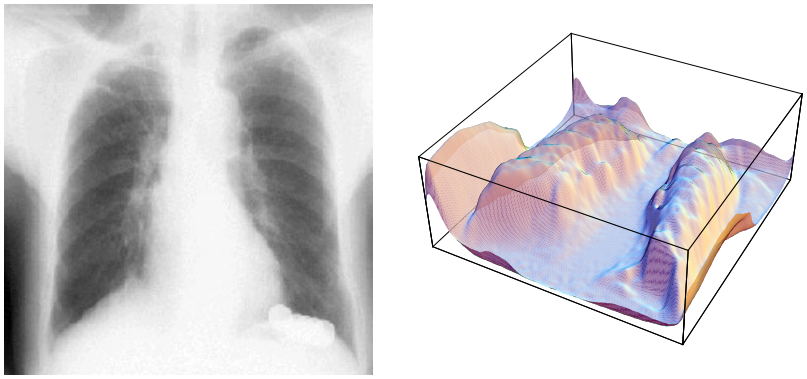
Out[107]//DisplayForm=

$$L[0, 0] + x\, L_x + \frac{x^2 L_{xx}}{2} + y \left(\frac{x^2 L_{xxy}}{2} + x\, L_{xy} + L_y \right) + \frac{1}{4} y^2 \left(x^2 L_{xxyy} + 2 \left(x\, L_{xyy} + L_{yy} \right) \right)$$

The second order term is $\frac{1}{2} L_{xx} x^2 + L_{xy} xy + \frac{1}{2} L_{yy} y^2$. The second order derivatives are the coefficients in the quadratic polynomial that describes the second order landscape.

```
In[108]:= imgFile = GetURL["http://www.bmi2.bmt.tue.nl/image-analysis/People/-
          BRomeny/FEV/Images/thorax02.gif"];
          im = Import[imgFile,"GIF"][[1,1]];
          DeleteFile[imgFile];
```

```
In[111]:= DisplayTogetherArray[ListDensityPlot[im],
            ListPlot3D[-gD[im, 0, 0, 2], Mesh → False], ImageSize → 320];
```



```
In[112]:= 1

Out[112]= 1
```

Left: An X-thorax image (resolution 256^2) and its 'intensity landscape' at $\sigma = 2$ pixels (right).

The Shape Index

When the principal curvatures κ_1 and κ_2 are considered coordinates in a 2D 'shape graph', we see that all different second order shapes are represented. Each shape is a point on this graph. The following list gives some possibilities:
When both curvatures are zero we have the *flat* shape.

When both curvatures are positive, we have *concave* shapes.

When both curvatures are negative, we have *convex* shapes.

When both curvatures the same sign and magnitude: *spherical* shapes.

When the curvatures have opposite sign: *saddle* shapes.

When one curvature is zero: *cylindrical* shapes.
Koenderink proposed the *shape index*. It is defined as:

$$\text{shapeindex} \equiv \frac{2}{\pi}\arctan\frac{\kappa_1+\kappa_2}{\kappa_1-\kappa_2},\ \kappa_1 \geq \kappa_2.$$

The expression for $\frac{\kappa_1+\kappa_2}{\kappa_1-\kappa_2}$ can be markedly cleaned up:

```
In[113]:= Simplify[(κ1 + κ2)/(κ1 - κ2)] // shortnotation
```

Out[113]//DisplayForm=

$$\frac{\left(-\frac{-2\,L_x\,L_{xy}\,L_y+L_{xx}\,L_y^2+L_x^2\,L_{yy}}{(L_x^2+L_y^2)^{3/2}}\right)_1+\left(-\frac{-2\,L_x\,L_{xy}\,L_y+L_{xx}\,L_y^2+L_x^2\,L_{yy}}{(L_x^2+L_y^2)^{3/2}}\right)_2}{\left(-\frac{-2\,L_x\,L_{xy}\,L_y+L_{xx}\,L_y^2+L_x^2\,L_{yy}}{(L_x^2+L_y^2)^{3/2}}\right)_1-\left(-\frac{-2\,L_x\,L_{xy}\,L_y+L_{xx}\,L_y^2+L_x^2\,L_{yy}}{(L_x^2+L_y^2)^{3/2}}\right)_2}$$

so we get for the shape index:

$$\text{shapeindex} \equiv \frac{2}{\pi}\arctan\left(\frac{-L_{xx}-L_{yy}}{\sqrt{{L_{xx}}^2+4{L_{xy}}^2-2L_{xx}L_{yy}+{L_{yy}}^2}}\right).$$

The length of the vector is the *curvedness*:

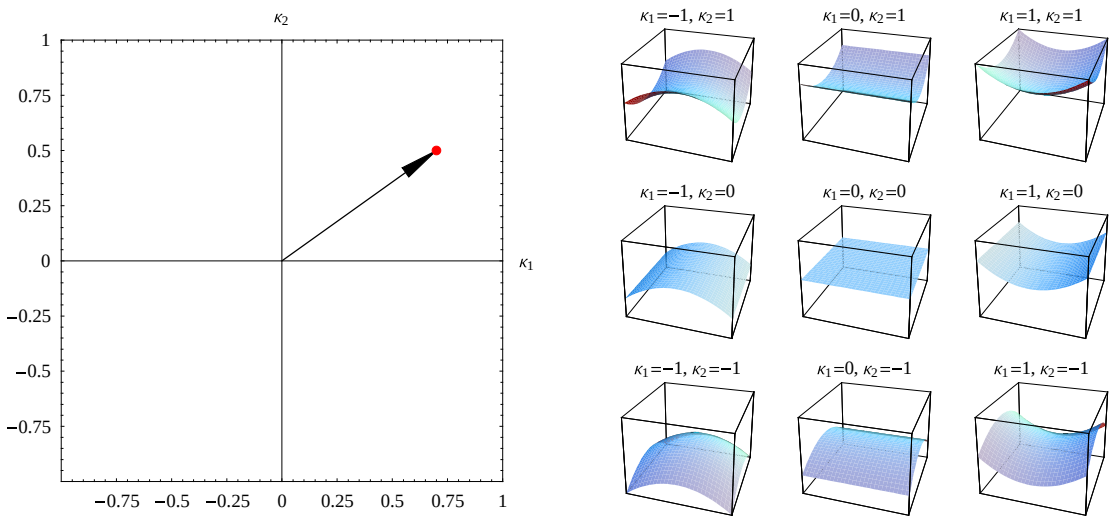
$$\text{curvedness} \equiv \frac{1}{2}\sqrt{\kappa_1^2 + \kappa_2^2}\,.$$

```
In[114]:= 1/2 Sqrt[κ1^2 + κ2^2] // Simplify // shortnotation
```

Out[114]//DisplayForm=

$$\frac{1}{2}\sqrt{\left(-\frac{-2\,L_x\,L_{xy}\,L_y + L_{xx}\,L_y^2 + L_x^2\,L_{yy}}{(L_x^2 + L_y^2)^{3/2}}\right)_1^2 + \left(-\frac{-2\,L_x\,L_{xy}\,L_y + L_{xx}\,L_y^2 + L_x^2\,L_{yy}}{(L_x^2 + L_y^2)^{3/2}}\right)_2^2}$$

```
In[115]:= shapes = Table[GraphicsArray[
            Table[Plot3D[κ1 x^2 + κ2 y^2, {x, -3, 3}, {y, -3, 3}, PlotRange → {-18, 18},
              PlotLabel → "κ1=" <> ToString[κ1] <> ", κ2=" <> ToString[κ2],
              AspectRatio → 1, DisplayFunction → Identity,
              Boxed → True, Mesh → False],
             {κ2, 1, -1, -1}, {κ1, -1, 1}]]];

In[116]:= Show[GraphicsArray[
            {Graphics[{Arrow[{0, 0}, {.7, .5}], Red, PointSize[.02], Point[{.7, .5}]},
              PlotRange → {{-1, 1}, {-1, 1}}, Frame → True, Axes → True,
              AxesLabel → {"κ1", "κ2"}, AspectRatio → 1], shapes}], ImageSize → 450];
```



Left: Coordinate space of the shape index. Horizontal axis: maximal principal curvature κ_1, vertical axis: minimal principal curvature κ_2. The angle of the position vector determines the shape, the length the curvedness. Right: same as middle set of figure 6.22.

Third Order Image Structure: T-junction Detection

```
In[117]:= imgFile = GetURL["http://www.bmi2.bmt.tue.nl/image-analysis/People/-
          BRomeny/FEV/Images/blankcheque.jpg"];
          im = Import[imgFile,"JPEG"];
          DeleteFile[imgFile];
```

In[120]:=
```
Show[im, ImageSize → 210];
```

The painting ‘the blank cheque’ by the famous Belgian surrealist painter René Magritte (1898–1967).

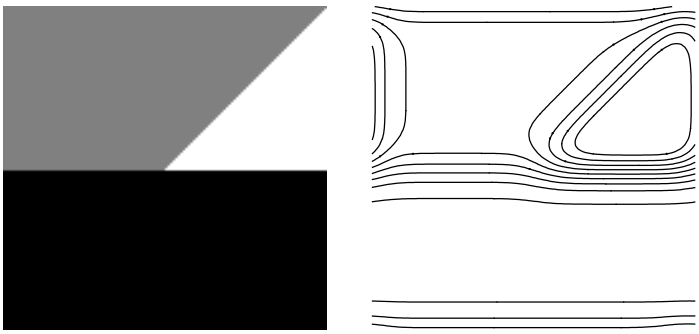
In[121]:=
```
imgFile = GetURL["http://www.bmi2.bmt.tue.nl/image-analysis/People/-
BRomeny/FEV/Images/blocks.gif"];
blocks = Import[imgFile,"GIF"][[1,1]];
DeleteFile[imgFile];
```

In[124]:=
```
ListDensityPlot[blocks,
  Epilog → (circles = {Circle[{221, 178}, 13], Circle[{157, 169}, 13],
      Circle[{90, 155}, 13], Circle[{148, 56}, 13],
      Circle[{194, 77}, 13], Circle[{253, 84}, 13]}), ImageSize → 300];
```

T-junctions often emerge at occlusion boundaries. The foreground edge is most likely to be the straight edge of the "T", with the occluded edge at some angle to it. The circles indicate some T-junctions in the image.

Let us zoom in on a T-junction of an observed image:

```
In[125]:= im = Table[If[y < 64, 0, 1] + If[y < x && y > 63, 2, 1], {y, 128}, {x, 128}];
          DisplayTogetherArray[ListDensityPlot[im], ListContourPlot[gD[im, 0, 0, 7],
            Contours → 15, PlotRange → {-0.3, 2.8}], ImageSize → 280];
```



The isophote structure (right) of a simple idealized and observed (blurred) T-junction (left) shows that isophotes strongly bend at T-junctions when we walk through the intensity landscape.

It seems to make sense to study $\frac{\partial \kappa}{\partial w}$:

We recall that the isophote curvature κ is defined as $\kappa = -\frac{L_{vv}}{L_w}$:

```
In[127]:= ? gauge2D

          gauge2D[L[x,y],nv,nw] calculates the  Gaussian derivatives of
            the function L[x,y] in the gauge coordinates {v,w}. v  is the
            direction tangential to the isophote, w is the gradient direction.
                L[x,y] = 2D input function
               nv = order of differentiation to v [Integer,  >= 0]
               nw = order of differentiation to w [Integer, >= 0]
           Example: gauge2D[L[x,y],2,0]//shortnotation
```

```
In[128]:= κ = gauge2D[L[x, y], 2, 0] / gauge2D[L[x, y], 0, 1]; κ // Simplify // shortnotation
```

Out[128]//DisplayForm=

$$\frac{-2\,L_x\,L_{xy}\,L_y + L_{xx}\,L_y^2 + L_x^2\,L_{yy}}{\left(L_x^2 + L_y^2\right)^{3/2}}$$

The derivative of the isophote curvature in the direction of the gradient, $\frac{\partial \kappa}{\partial w}$ is quite a complex third order expression. The formula is derived by calculating the *directional derivative* of the curvature in the direction of the normalized gradient. We define the gradient (or *nabla*: ∇) operator with a pure function:

```
In[129]:= grad = {∂x#, ∂y#} &;
          dκdw = grad[L[x, y]] / Sqrt[grad[L[x, y]].grad[L[x, y]]].grad[κ];
          dκdw // Simplify // shortnotation
```

Out[131]//DisplayForm=

$$\frac{1}{(L_x^2 + L_y^2)^3}$$
$$(L_{xxy} L_y^5 + L_x^4 (-2 L_{xy}^2 + L_x L_{xyy} - L_{xx} L_{yy}) - L_y^4 (2 L_{xy}^2 - L_x (L_{xxx} - 2 L_{xyy}) + L_{xx} L_{yy}) +$$
$$L_x^2 L_y^2 (-3 L_{xx}^2 + 8 L_{xy}^2 + L_x (L_{xxx} - L_{xyy}) + 4 L_{xx} L_{yy} - 3 L_{yy}^2) +$$
$$L_x^3 L_y (6 L_{xy} (L_{xx} - L_{yy}) + L_x (-2 L_{xxy} + L_{yyy})) +$$
$$L_x L_y^3 (6 L_{xy} (-L_{xx} + L_{yy}) + L_x (-L_{xxy} + L_{yyy})))$$

To avoid singularities at vanishing gradients through the division by $(L_x{}^2 + L_y{}^2)^3 = L_w{}^6$ we use as our T-junction detector $\tau = \frac{\partial \kappa}{\partial w} L_w{}^6$:

In[132]:=
```
tjunction = dκdw (grad[L[x, y]].grad[L[x, y]])^3;
tjunction // shortnotation
```

Out[133]//DisplayForm=

$$L_x^5 L_{xyy} + L_y^4 (-2 L_{xy}^2 + L_{xxy} L_y - L_{xx} L_{yy}) + L_x^3 L_y (6 L_{xx} L_{xy} + L_{xxx} L_y - L_{xyy} L_y - 6 L_{xy} L_{yy}) +$$
$$L_x L_y^3 (-6 L_{xx} L_{xy} + L_{xxx} L_y - 2 L_{xyy} L_y + 6 L_{xy} L_{yy}) - L_x^4 (2 L_{xy}^2 + 2 L_{xxy} L_y + L_{xx} L_{yy} - L_y L_{yyy}) +$$
$$L_x^2 L_y^2 (-3 L_{xx}^2 + 8 L_{xy}^2 - L_{xxy} L_y + 4 L_{xx} L_{yy} - 3 L_{yy}^2 + L_y L_{yyy})$$

Finally, we apply the T-junction detector on our blocks at a rather fine scale of $\sigma = 2$ (we plot $-$tjunction to invert the contrast):

In[134]:=
```
σ = 2; ListDensityPlot[
 tjunction /. Derivative[nx_, ny_][L][x, y] → gD[im0, nx, ny, σ] /.
  im0 → blocks, Epilog → circles, ImageSize → 230];
```

Detection of T-junctions in the image of the blocks. The same circles have been drawn as in the figure above.

Example 3: Deblurring Gaussian Blur

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[135]:=
```
L =.;
Series[L[x, y, t], {t, 0, 3}]
```

```
Out[136]= L[x, y, 0] + L^(0,0,1)[x, y, 0] t + 1/2 L^(0,0,2)[x, y, 0] t^2 + 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[137]:= Δ := (∂x,x# + ∂y,y#) &

In[138]:= Δ[f[x, y]]

Out[138]= (1 + 0.1 y) (-Sin[x] - 3 Sin[3 x])
```

The repeated Laplacian operator is made with the function Nest:

```
In[139]:= Nest[f, x, 3]

Out[139]= f[f[f[x]]]
```

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

```
In[140]:= 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[140]= L[x, y, 0] + t (L^(0,2,0)[x, y, 0] + L^(2,0,0)[x, y, 0]) +
          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]) +
          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])
```

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

In[142]:= expr // short
```

Out[142]//DisplayForm=

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

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

Out[143]//DisplayForm=

$$L[x, y, 0] + t\,(L_{xx} + L_{yy}) + \frac{1}{2}\,t^2\,(L_{xxxx} + 2\,L_{xxyy} + L_{yyyy}) +$$
$$\frac{1}{6}\,t^3\,(L_{xxxxxx} + 3\,L_{xxxxyy} + 3\,L_{xxyyyy} + L_{yyyyyy}) +$$
$$\frac{1}{24}\,t^4\,(L_{xxxxxxxx} + 4\,L_{xxxxxxyy} + 6\,L_{xxxxyyyy} + 4\,L_{xxyyyyyy} + L_{yyyyyyyy}) + \frac{1}{120}\,t^5$$
$$(L_{xxxxxxxxxx} + 5\,L_{xxxxxxxxyy} + 10\,L_{xxxxxxyyyy} + 10\,L_{xxxxyyyyyy} + 5\,L_{xxyyyyyyyy} + L_{yyyyyyyyyy}) +$$
$$\frac{1}{720}\,t^6\,(L_{xxxxxxxxxxxx} + 6\,L_{xxxxxxxxxxyy} + 15\,L_{xxxxxxxxyyyy} + 20\,L_{xxxxxxyyyyyy} +$$
$$15\,L_{xxxxyyyyyyyy} + 6\,L_{xxyyyyyyyyyy} + L_{yyyyyyyyyyyy}) + \frac{1}{5040}$$
$$(t^7\,(L_{xxxxxxxxxxxxxx} + 7\,L_{xxxxxxxxxxxxyy} + 21\,L_{xxxxxxxxxxyyyy} + 35\,L_{xxxxxxxxyyyyyy} +$$
$$35\,L_{xxxxxxyyyyyyyy} + 21\,L_{xxxxyyyyyyyyyy} + 7\,L_{xxyyyyyyyyyyyy} + L_{yyyyyyyyyyyyyy})) +$$
$$\frac{1}{40320}\,(t^8\,(L_{xxxxxxxxxxxxxxxx} + 8\,L_{xxxxxxxxxxxxxxxy} + 28\,L_{xxxxxxxxxxxxyyyy} +$$
$$56\,L_{xxxxxxxxxxyyyyyy} + 70\,L_{xxxxxxxxyyyyyyyy} + 56\,L_{xxxxxxyyyyyyyyyy} +$$
$$28\,L_{xxxxyyyyyyyyyyyy} + 8\,L_{xxyyyyyyyyyyyyyy} + L_{yyyyyyyyyyyyyyyy}))$$

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'; Release-Hold removes the hold):

In[144]:=
```
corr = expr /. Derivative[n_, m_, 0][L][x, y, a_] → HoldForm[gD[im, n, m, 1]]
```

Out[144]//DisplayForm=

$$L[x, y, 0] + t\,(L_{xx} + L_{yy}) + \frac{1}{2}\,t^2\,(L_{xxxx} + 2\,L_{xxyy} + L_{yyyy}) +$$
$$\frac{1}{6}\,t^3\,(L_{xxxxxx} + 3\,L_{xxxxyy} + 3\,L_{xxyyyy} + L_{yyyyyy}) +$$
$$\frac{1}{24}\,t^4\,(L_{xxxxxxxx} + 4\,L_{xxxxxxyy} + 6\,L_{xxxxyyyy} + 4\,L_{xxyyyyyy} + L_{yyyyyyyy}) + \frac{1}{120}\,t^5$$
$$(L_{xxxxxxxxxx} + 5\,L_{xxxxxxxxyy} + 10\,L_{xxxxxxyyyy} + 10\,L_{xxxxyyyyyy} + 5\,L_{xxyyyyyyyy} + L_{yyyyyyyyyy}) +$$
$$\frac{1}{720}\,t^6\,(L_{xxxxxxxxxxxx} + 6\,L_{xxxxxxxxxxyy} + 15\,L_{xxxxxxxxyyyy} + 20\,L_{xxxxxxyyyyyy} +$$
$$15\,L_{xxxxyyyyyyyy} + 6\,L_{xxyyyyyyyyyy} + L_{yyyyyyyyyyyy}) + \frac{1}{5040}$$
$$(t^7\,(L_{xxxxxxxxxxxxxx} + 7\,L_{xxxxxxxxxxxxyy} + 21\,L_{xxxxxxxxxxyyyy} + 35\,L_{xxxxxxxxyyyyyy} +$$
$$35\,L_{xxxxxxyyyyyyyy} + 21\,L_{xxxxyyyyyyyyyy} + 7\,L_{xxyyyyyyyyyyyy} + L_{yyyyyyyyyyyyyy})) +$$
$$\frac{1}{40320}\,(t^8\,(L_{xxxxxxxxxxxxxxxx} + 8\,L_{xxxxxxxxxxxxxxxy} + 28\,L_{xxxxxxxxxxxxyyyy} +$$
$$56\,L_{xxxxxxxxxxyyyyyy} + 70\,L_{xxxxxxxxyyyyyyyy} + 56\,L_{xxxxxxyyyyyyyyyy} +$$
$$28\,L_{xxxxyyyyyyyyyyyy} + 8\,L_{xxyyyyyyyyyyyyyy} + L_{yyyyyyyyyyyyyyyy}))$$

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[145]:=
```
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:

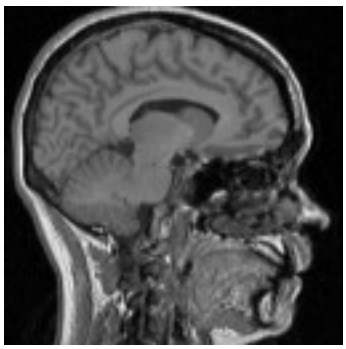
```
In[146]:= im =.; deblur[im, 2, 1, 2]

Out[146]= -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 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:

```
In[147]:= deblur[im, 2, 3, 2]

Out[147]= -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 test image 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[148]:= imgFile = GetURL["http://www.bmi2.bmt.tue.nl/image-analysis/People/-
          BRomeny/FEV/images/mr128.gif"];
          image = Import[imgFile,"GIF"];
          DeleteFile[imgFile];
          Show[image];
```



```
In[152]:= im = image[[1, 1]]; blur = gDf[im, 0, 0, 2];
          ListDensityPlot[blur, ImageSize → 128];
```

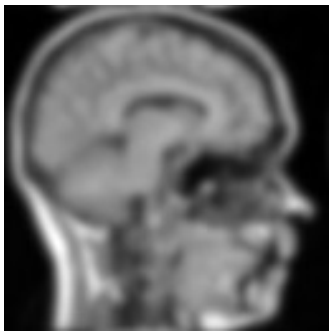
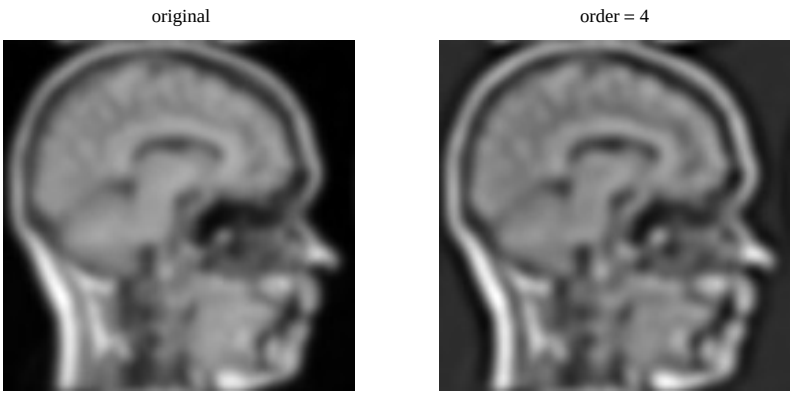
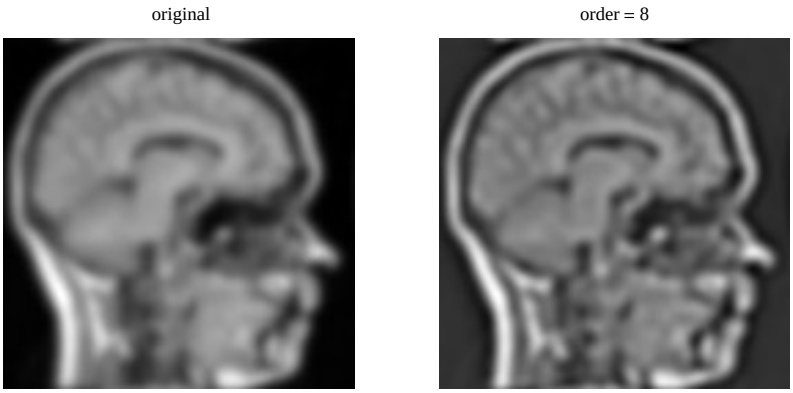


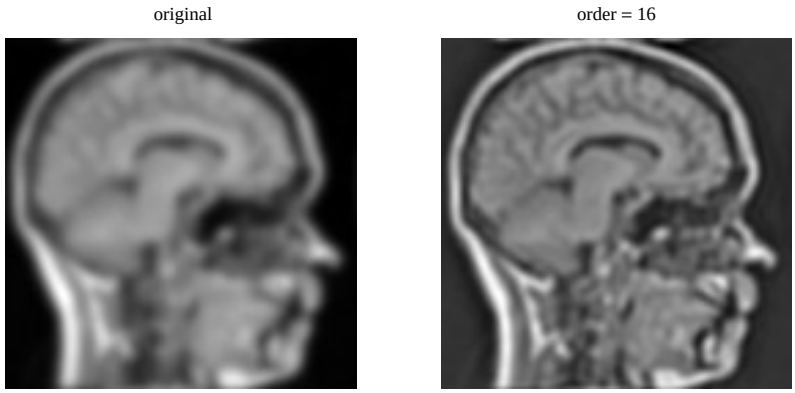
Figure 19. Input image for deblurring, blurred at σ = 2 pixels. Image resolution 128^2.

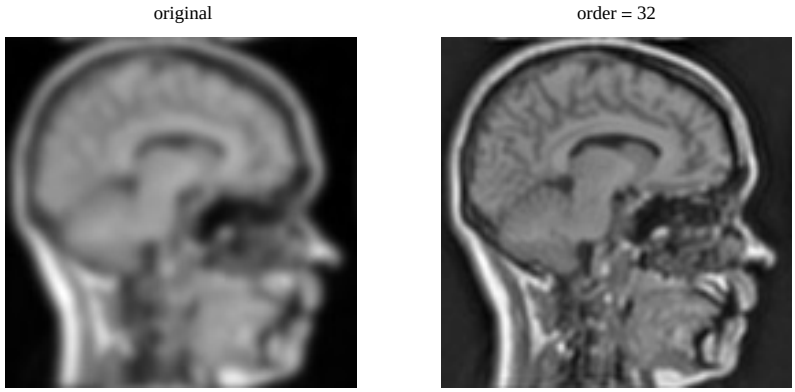
We try a deblurring for orders 4, 8, 16 and 32:

```
In[154]:= Timing[Do[
            corr = deblur[blur, 2, 2^i, 4] // ReleaseHold;
          Block[{$DisplayFunction = Identity},
            p1 = ListDensityPlot[blur, PlotLabel → "original"];
            p2 =
             ListDensityPlot[blur + corr, PlotLabel → "order = " <> ToString[2^i]]];
          Show[GraphicsArray[{p1, p2}], ImageSize → 330];,
          {i, 2, 5}];][[1]]
```










```
Out[154]= 68.625 Second
```

Result of deblurring to 32nd order.
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 4.5 minutes for a 128^2 image on a 500 MHz 128 MB Pentium III under Windows 98 (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[155]:= dummy =.; Length[Position[deblur[dummy, 2, 32, 4], gD]]

Out[155]= 560
```

Example 4: Detection of Granular Structures in 3D Macrophages

Initialization

Use the initialization of example 1.
Read the TIFF file with the 50 slices from the author's homepage's image directory (NB: 12.8 MB):

```
In[156]:= imgFile = GetURL["http://www.bmi2.bmt.tue.nl/image-analysis/People/-
          BRomeny/FEV/images/29-4-2004-2_raw02.tif"];
          im = Import[imgFile,"TIFF"];
          DeleteFile[imgFile];
```

Take from all 50 slices the pixels (in the [[1,1]] element), and take the green channel only of the color RGB images (the other channels are zero):

```
In[159]:= tmp = im[[All, 1, 1]];
          Dimensions[im3D = tmp[[All, All, All, 2]]]

Out[160]= {50, 512, 512}

          {50, 512, 512}
```

Take a submatrix with the macrophage in each of the 50 images:

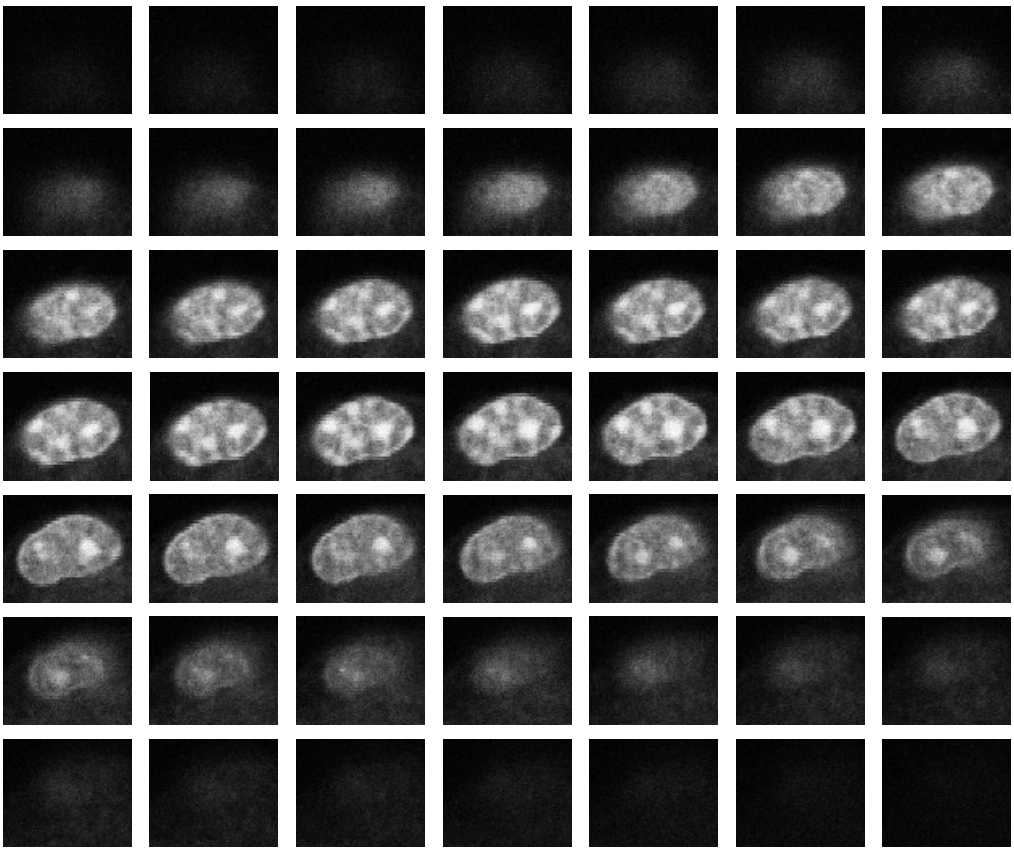
```
In[161]:= imn = Take[im3D, All, {211, 320}, {286, 417}];
          {max, min} = {Max[imn], Min[imn]};
          {zdim, ydim, xdim} = Dimensions[imn]

Out[163]= {50, 110, 132}

          {50, 110, 132}
```

The z-scale is multiplied by α to include the anisotropy of the voxels:

```
In[164]:= α = 1;
```

Check the slices:

```
In[165]:= Show[GraphicsArray[Partition[ListDensityPlot[#,
                DisplayFunction → Identity, PlotRange → {min, max}] & /@ imn, 7]],
            DisplayFunction → $DisplayFunction, ImageSize → 600];
```



3D Viewer

```
In[166]:= Get["MathGL3d`OpenGLViewer`"];

          The OpenGLViewer is running.

          LinkObject::linkd :
           LinkObject[C:\Documents and Settings\All Users\Applicat
               … L3d\Binaries\Windows\mathview3d.exe -mathlink, 158, 6]
             is closed; the connection is dead. More…
```

This shows the outline (isosurface of value 100) of the macrophage in 3D in a separate window:

```
In[167]:= MVClear[];
          g = MVListContourPlot3D[gDn[imn, {0, 0, 0}, {1, 1, 1}],
             Contours → {100}, LightSources → {{{1, 0, 1}, RGBColor[.6, .6, .6]},
               {{1, 0, 1}, RGBColor[.6, .6, .6]}, {{0, 0, 1}, RGBColor[.6, .6, .6]}},
             ContourStyle → {Banana}, DisplayFunction → Identity,
             MVReducePolygons → {0, Automatic}];

          LinkObject::linkn :
           Argument LinkObject[C:\Documents and Settings\All Users\Applicat
               … L3d\Binaries\Windows\mathview3d.exe -mathlink, 158, 6] in
             LinkWrite[LinkObject[C:\Documents and Settings\All Users\A …
                aries\Windows\mathview3d.exe -mathlink, 158, 6], <<1>>]
             has an invalid LinkObject number; the link may be dead. More…
```

```
LinkObject::linkn :
 Argument LinkObject[C:\Documents and Settings\All Users\Applicat
    … L3d\Binaries\Windows\mathview3d.exe -mathlink, 158, 6] in
  LinkWrite[LinkObject[C:\Documents and Settings\All Users\A …
     aries\Windows\mathview3d.exe -mathlink, 158, 6], <<1>>]
  has an invalid LinkObject number; the link may be dead. More…
```

And this is the wireframe:

```
In[169]:= wireframe = Show[WireFrame[g],
            DisplayFunction → $DisplayFunction, ImageSize → 550, BoxRatios → {1, 1, 1}];
```

Blur the 3D data a little with $\sigma = 3$ pixels in the *x*, *y* and *z* dimension:

```
In[170]:= ? gDn

          gDn[im,{...,ny,nx},{...,σy,σx},optio ns] calculates the Gaussian
            derivative of an N-dimensional image by  approximated spatial
            convolution. It is optimized for speed by 1D convolutions per
            dimension. The image is considered cyclic in each direction.
            Note the  order of the dimensions in the parameter lists.
               im = N-dimensional  input image [List]
               nx = order of differentiation to x [Integer, nx  ≥ 0]
               σx = scale in x-dimension [in pixels, σ >  0]
               options = <optional> kernelSampleRange: range of kernel
            sampled in  multiples of σ. Default: kernelSampleRange->{-6,6}

           Example: gDn[im,{0,0,1},{2,2,2}] calculates the x-
            derivative of a 3D image at  an isotropic scale of σz=σy=σx=2.


In[171]:= imnblurred = gDn[imn, {0, 0, 0}, {3, 3, 3}];
```

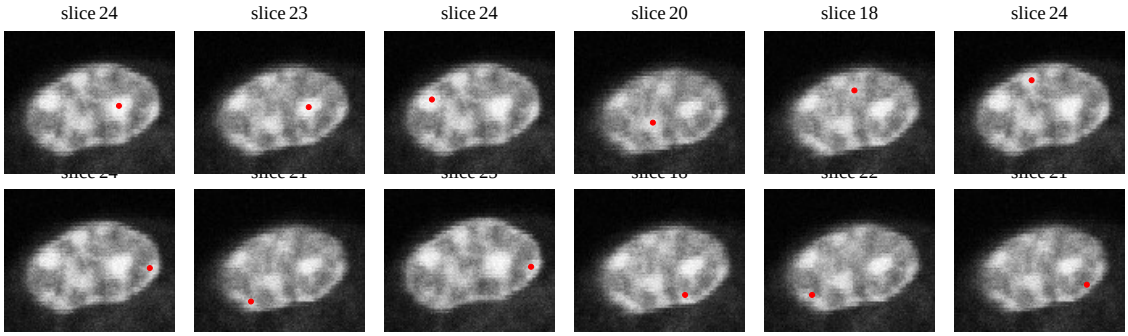
Find the *n* largest maxima in *N*-dimensions:

```
In[172]:= nMaxima[im_, n_] := Module[{l, d = Depth[im] - 1},
            p = Times @@ Table[(Sign[im - Map[RotateLeft, im, {i}]] + 1)
                (Sign[im - Map[RotateRight, im, {i}]] + 1), {i, 0, d - 1}];
            l = Length[Position[p, 4^d]];
            Take[
             Reverse[Union[{Extract[im, #], #} & /@ Position[p, 4^d]]], If[n < l, n, l]]];
```

We return the 3D positions of the 12 largest maxima:

```
In[173]:= maximumpositions = Last[#] & /@ nMaxima[imnblurred, 12]

Out[173]= {{24, 52, 89}, {23, 51, 89}, {24, 57, 37}, {20, 39, 61},
           {18, 64, 70}, {24, 72, 60}, {24, 49, 113}, {21, 23, 44},
           {25, 50, 114}, {18, 28, 86}, {22, 28, 37}, {21, 36, 103}}


          {{24, 52, 89}, {23, 51, 89}, {24, 57, 37}, {20, 39, 61},
           {18, 64, 70}, {24, 72, 60}, {24, 49, 113}, {21, 23, 44},
           {25, 50, 114}, {18, 28, 86}, {22, 28, 37}, {21, 36, 103}}
```

```
In[174]:= Show[GraphicsArray[
            Partition[ListDensityPlot[imn[[First[#]]], PlotRange → {min, max},
                PlotLabel → "slice " <> ToString[#[[1]]], DisplayFunction → Identity,
                Epilog → {Red, PointSize[0.03], Point[Reverse[Drop[#, 1]]]}] & /@
              maximumpositions, 6], DisplayFunction → $DisplayFunction,
            ImageSize → 600, GraphicsSpacing → -.1]];
```



Granulae Shape Detection

We sample the intensity along a star of rays in each granule, starting at the location of its maximum value. We interpolate with a cubic spline (3rd order) function:

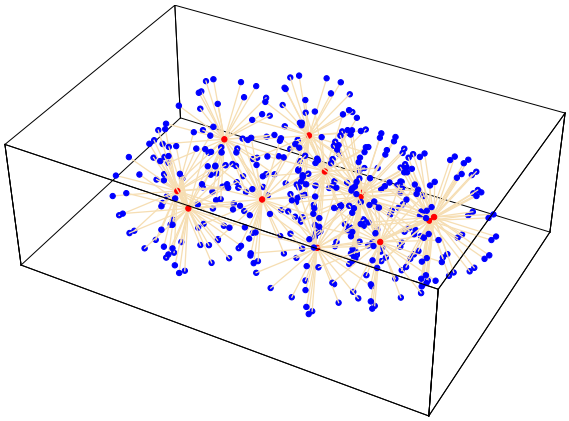
```
In[175]:= interpolation = ListInterpolation[imn];
```

Create a star of rays with each 20 sampling points 1 pixel apart starting from the maximum position {*x*, *y*, *z*} in 7 directions of θ (tils) and 5 directions of ϕ (slant), in total 35 directions:

```
In[176]:= δθ = π / 8; δϕ = 2 π / 5;
          rays[z_, y_, x_] := Module[{ϕ, θ, r}, Table[N[
               interpolation[z + r Cos[ϕ] Cos[θ], y + r Sin[ϕ] Cos[θ], x + r Sin[θ]]],
              {θ, -π / 2 + δθ, π / 2 - δθ, δθ}, {ϕ, 0, 2 Pi - δϕ, δϕ}, {r, 1, 20}]];
```

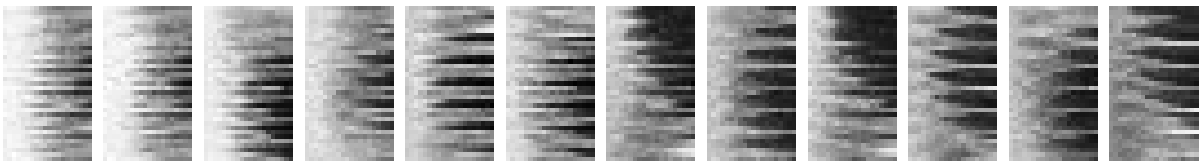
Just to check the result visually, we display the star of sampling rays:

```
In[178]:= star[z_, y_, x_] := Module[{ϕ, θ, r},
            Graphics3D[{Red, Point[{x, y, α z}], Table[{Wheat, Line[{{x, y, α z},
                  N[{x + r Cos[ϕ] Cos[θ], y + r Sin[ϕ] Cos[θ], α z + r Sin[θ]}]}], Blue,
                Point[N[{x + r Cos[ϕ] Cos[θ], y + r Sin[ϕ] Cos[θ], α z + r Sin[θ]}]]},
              {θ, -π / 2 + δθ, π / 2 - δθ, δθ}, {ϕ, 0, 2 π - δϕ, δϕ}, {r, 20, 20}]}]];
          stars = Show[Apply[star, maximumpositions, 2]];
```



We sample the intensity tracks along the 35 rays outbound from the 8 maxima, and inspect them lined up in a single figure:

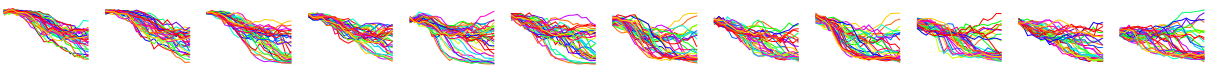
```
In[180]:= Off[InterpolatingFunction::"dmval"];

In[181]:= DisplayTogetherArray[ListDensityPlot[#, PlotRange → {min, max}] & /@
            (tracks = (Flatten[rays @@ #, 1] & /@ maximumpositions)), ImageSize → 600];
```



```
In[182]:= Dimensions[tracks]

Out[182]= {12, 35, 20}

          {12, 35, 20}
```

Here are the tracks per granule:

```
In[183]:= DisplayTogetherArray[
            MultipleListPlot[#, PlotRange → {min, max}, SymbolShape → None, Axes → False,
              PlotStyle → Hue /@ Range[0, 1, 1 / 15]] & /@ tracks, ImageSize → 600];
```



The edges are very weak, in a very noisy environment. Therefore we will use the signature function and edge focusing to detect the location of the largest edge in the track.

Signatures

The signature function is calculated in the Fourier domain to prevent accuracy errors at larger scales.

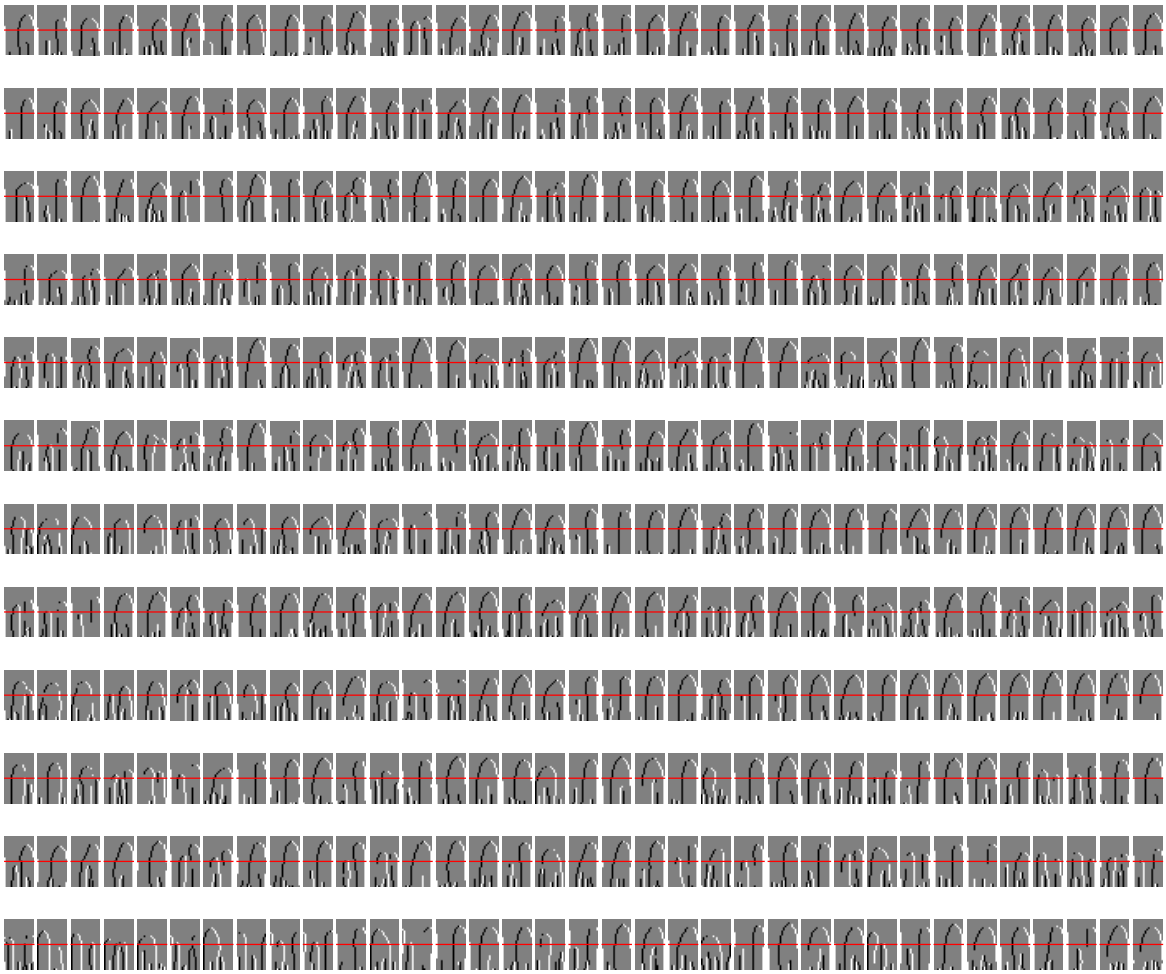
```
In[184]:= Clear[signature];
          signature[track_] := Module[{ss},
            ss = Table[gDf1D[track, 2, Eτ], {τ, 0, 2, .06}];
            (RotateRight[#] - #) & /@ Sign[ss]];
```

Calculate the 20-pixel signatures for 34 levels of scale for all 35 tracks around all 8 maxima with convolution in the Fourier domain:

```
In[186]:= signatures = Map[signature, tracks, {2}];
          Dimensions[signatures]

Out[187]= {12, 35, 34, 20}
```

Display all signature functions with a level line at 17:

```
In[188]:= level = 17;
          DisplayTogetherArray[
              ListDensityPlot[#, Epilog → {Red, Line[{{0, level}, {20, level}}]}] & /@ #,
              ImageSize → 600] & /@ signatures;
```



Edge Focusing

The function edgefocus takes a signature and a startlevel, and a direction (upgoing edge: dir= 2, downgoing edge: dir = -2).

```
In[190]:= edgefocus[signature_, startlevel_, dir_] := Module[{a, b, c},
            out = 0. signature; a = Position[signature[[startlevel]], dir];
            Do[b = Position[signature[[i]], dir];
             c = Select[b, (Position[a, # - 1] ≠ {} ||
                  Position[a, #] ≠ {} || Position[a, # + 1] ≠ {}) &];
```

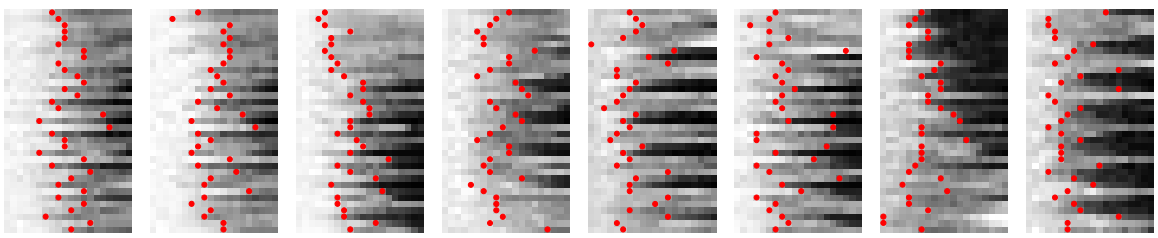
```
    out[[i]] = ReplacePart[out[[i]], -1, c]; b = c; a = b,
    {i, startlevel - 1, 1, -1}];
  Position[First[out], -1][[1, 1]]];
```

In[191]:=
```
edgelocations = Map[edgefocus[#, 17, -2] &, signatures, {2}]
```

Out[191]=
```
{{11, 14, 7, 11, 13, 9, 13, 9, 11, 14, 8, 13, 6, 10, 10, 8, 17,
  6, 16, 9, 8, 12, 10, 13, 12, 10, 9, 13, 13, 9, 10, 10, 10, 9, 8},
 {12, 12, 9, 10, 7, 9, 16, 9, 10, 14, 8, 13, 7, 8, 10, 8, 17, 7, 15,
  11, 8, 13, 10, 12, 11, 10, 12, 13, 13, 12, 13, 12, 13, 4, 8},
 {9, 13, 8, 8, 7, 7, 14, 6, 13, 9, 7, 15, 9, 11, 10, 7, 9, 9,
  12, 12, 11, 9, 11, 11, 8, 6, 7, 6, 5, 6, 6, 9, 5, 4, 5},
 {17, 8, 10, 9, 9, 9, 7, 5, 13, 8, 7, 8, 11, 11, 6, 8, 10, 13,
  12, 9, 8, 14, 13, 12, 6, 8, 11, 11, 15, 7, 7, 6, 8, 9, 11},
 {6, 5, 13, 7, 11, 13, 7, 7, 8, 13, 5, 7, 6, 9, 4, 5, 7, 6, 8, 3, 5, 6, 7, 8,
  5, 5, 13, 10, 14, 1, 6, 8, 9, 8, 7}, {7, 9, 8, 6, 7, 4, 11, 8, 17, 6, 4, 13,
  7, 15, 4, 4, 16, 9, 16, 8, 7, 8, 10, 8, 7, 9, 8, 4, 18, 6, 9, 6, 3, 6, 7},
 {7, 1, 1, 8, 5, 5, 11, 4, 10, 5, 6, 7, 7, 7, 14, 7, 7, 13,
  12, 10, 8, 8, 10, 10, 7, 9, 10, 5, 5, 5, 8, 8, 5, 6, 7},
 {7, 7, 15, 9, 8, 5, 7, 11, 8, 4, 12, 6, 6, 6, 9, 6, 4, 6, 7, 9, 5, 4, 15, 6,
  9, 15, 5, 7, 8, 11, 5, 5, 4, 4, 13}, {6, 2, 1, 2, 7, 5, 10, 4, 13, 7, 5, 7,
  3, 17, 7, 5, 7, 13, 12, 8, 8, 7, 11, 5, 8, 7, 10, 6, 6, 6, 7, 7, 5, 5, 5},
 {6, 8, 9, 4, 5, 5, 6, 13, 11, 8, 14, 9, 12, 9, 6, 10, 1, 9, 7,
  5, 10, 4, 11, 7, 6, 10, 6, 12, 8, 7, 13, 3, 6, 11, 9},
 {12, 8, 7, 8, 10, 7, 5, 8, 8, 10, 7, 4, 6, 10, 8, 13, 3, 6, 7,
  10, 5, 3, 6, 12, 12, 5, 2, 3, 13, 15, 10, 3, 3, 7, 4},
 {3, 1, 1, 1, 1, 3, 1, 18, 4, 5, 13, 1, 3, 12, 8, 11, 5, 15,
  10, 5, 7, 18, 13, 9, 5, 8, 1, 13, 7, 4, 10, 9, 15, 5, 5}}
```

Let us visually check the correctness of the detected edges in the noise:

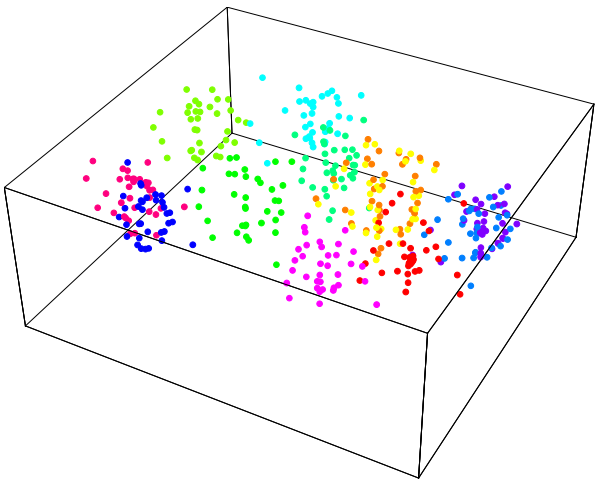
In[192]:=
```
DisplayTogetherArray[Table[ListDensityPlot[tracks[[i]],
    Epilog → {Red, PointSize[0.04], MapIndexed[Point[{#1, #2[[1]]} - .5] &,
       edgelocations[[i]]]}], {i, 1, 8}], ImageSize → 600];
```



In[193]:=
```
r =.
pr[{z_, y_, x_}] :=
  Flatten[Table[N[{x + r Cos[ϕ] Cos[θ], y + r Sin[ϕ] Cos[θ], α z + r Sin[θ]}],
    {θ, -π / 2 + δθ, π / 2 - δθ, δθ}, {ϕ, 0, 2 π - δϕ, δϕ}], 1];
```

In[195]:=
```
edgepoints3D =
  MapThread[(#1 /. r → #2) &, {pr /@ maximumpositions, edgelocations}, 2];
```

In[196]:=
```
Show[Graphics3D[MapThread[List, {Hue /@ (Range[Length[edgepoints3D]] / Length[edgepoints3D]),
    Map[Point, edgepoints3D, {2}]}]]];
```



Fit Spherical Harmonic Functions to Second Order

In[197]:=
```
order = 2;
fitfunctions =
 Flatten[Table[SphericalHarmonicY[l, m, θ, ϕ], {l, 0, order}, {m, -l, l, 1}]]
```

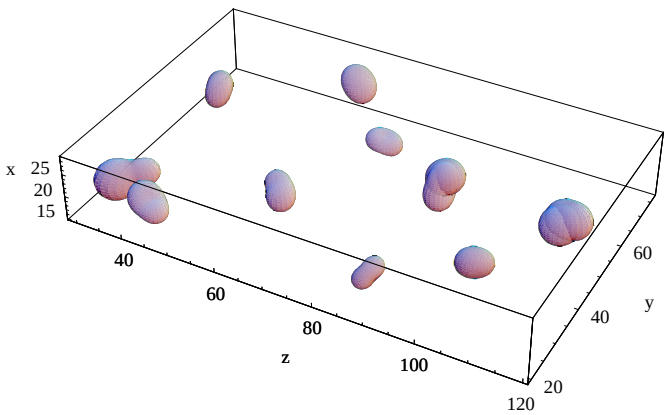
Out[198]=
$$\left\{\frac{1}{2\sqrt{\pi}},\ \frac{1}{2}e^{-i\phi}\sqrt{\frac{3}{2\pi}}\sin[\theta],\ \frac{1}{2}\sqrt{\frac{3}{\pi}}\cos[\theta],\ -\frac{1}{2}e^{i\phi}\sqrt{\frac{3}{2\pi}}\sin[\theta],\right.$$
$$\frac{1}{4}e^{-2i\phi}\sqrt{\frac{15}{2\pi}}\sin[\theta]^2,\ \frac{1}{2}e^{-i\phi}\sqrt{\frac{15}{2\pi}}\cos[\theta]\sin[\theta],$$
$$\left.\frac{1}{4}\sqrt{\frac{5}{\pi}}\,(-1+3\cos[\theta]^2),\ -\frac{1}{2}e^{i\phi}\sqrt{\frac{15}{2\pi}}\cos[\theta]\sin[\theta],\ \frac{1}{4}e^{2i\phi}\sqrt{\frac{15}{2\pi}}\sin[\theta]^2\right\}$$

In[199]:=
```
fitresults = Chop[ExpToTrig[Fit[#, fitfunctions, {θ, ϕ}]]] & /@ edgepoints3D
```

Out[199]=
```
{24.5375 + 1.36246 Cos[θ] - 0.346201 Cos[θ]^2 +
  0.371463 Cos[ϕ] Sin[θ] - 1.34494 Cos[θ] Cos[ϕ] Sin[θ] -
  0.998947 Cos[2 ϕ] Sin[θ]^2 + 0.162299 Sin[θ] Sin[ϕ] +
  4.52912 Cos[θ] Sin[θ] Sin[ϕ] - 1.16299 Sin[θ]^2 Sin[2 ϕ],
 19.0396 - 1.70754 Cos[θ] + 10.9077 Cos[θ]^2 - 0.516394 Cos[ϕ] Sin[θ] +
  6.48349 Cos[θ] Cos[ϕ] Sin[θ] + 5.37368 Cos[2 ϕ] Sin[θ]^2 -
  3.1673 Sin[θ] Sin[ϕ] - 1.76275 Cos[θ] Sin[θ] Sin[ϕ] + 2.59704 Sin[θ]^2 Sin[2 ϕ],
 21.8964 + 1.19997 Cos[θ] + 1.99265 Cos[θ]^2 + 0.800959 Cos[ϕ] Sin[θ] +
  0.859479 Cos[θ] Cos[ϕ] Sin[θ] + 2.82458 Cos[2 ϕ] Sin[θ]^2 +
  1.03126 Sin[θ] Sin[ϕ] + 4.91885 Cos[θ] Sin[θ] Sin[ϕ] - 3.62613 Sin[θ]^2 Sin[2 ϕ],
 21.1501 + 0.873112 Cos[θ] - 2.52097 Cos[θ]^2 - 0.47186 Cos[ϕ] Sin[θ] +
  7.48707 Cos[θ] Cos[ϕ] Sin[θ] - 0.323233 Cos[2 ϕ] Sin[θ]^2 +
  4.93223 Sin[θ] Sin[ϕ] - 4.44066 Cos[θ] Sin[θ] Sin[ϕ] - 4.29129 Sin[θ]^2 Sin[2 ϕ],
 22.2647 + 0.269869 Cos[θ] - 8.27813 Cos[θ]^2 +
```

```
  3.40177 Cos[ϕ] Sin[θ] - 3.74178 Cos[θ] Cos[ϕ] Sin[θ] -
  1.31886 Cos[2 ϕ] Sin[θ]^2 - 0.444734 Sin[θ] Sin[ϕ] -
  5.44111 Cos[θ] Sin[θ] Sin[ϕ] + 0.624648 Sin[θ]^2 Sin[2 ϕ],
 24.3495 - 2.0098 Cos[θ] - 0.337587 Cos[θ]^2 + 2.70311 Cos[ϕ] Sin[θ] -
  1.30374 Cos[θ] Cos[ϕ] Sin[θ] + 0.75009 Cos[2 ϕ] Sin[θ]^2 -
  0.160252 Sin[θ] Sin[ϕ] - 5.75551 Cos[θ] Sin[θ] Sin[ϕ] -
  1.37843 Sin[θ]^2 Sin[2 ϕ], 25.4283 + 0.438578 Cos[θ] - 2.15445 Cos[θ]^2 -
  0.316226 Cos[ϕ] Sin[θ] + 0.431729 Cos[θ] Cos[ϕ] Sin[θ] +
  0.464945 Cos[2 ϕ] Sin[θ]^2 - 0.478331 Sin[θ] Sin[ϕ] +
  0.643766 Cos[θ] Sin[θ] Sin[ϕ] - 0.327531 Sin[θ]^2 Sin[2 ϕ],
 24.6763 + 0.00750891 Cos[θ] - 7.07093 Cos[θ]^2 + 2.09091 Cos[ϕ] Sin[θ] +
  4.2939 Cos[θ] Cos[ϕ] Sin[θ] - 2.98321 Cos[2 ϕ] Sin[θ]^2 - 4.34821 Sin[θ] Sin[ϕ] -
  8.54925 Cos[θ] Sin[θ] Sin[ϕ] - 2.83875 Sin[θ]^2 Sin[2 ϕ],
 26.5489 - 0.656594 Cos[θ] - 1.99621 Cos[θ]^2 + 1.01398 Cos[ϕ] Sin[θ] -
  0.837318 Cos[θ] Cos[ϕ] Sin[θ] - 3.41477 Cos[2 ϕ] Sin[θ]^2 +
  1.12681 Sin[θ] Sin[ϕ] + 4.22902 Cos[θ] Sin[θ] Sin[ϕ] - 3.06597 Sin[θ]^2 Sin[2 ϕ],
 16.9595 - 0.172721 Cos[θ] + 1.63283 Cos[θ]^2 -
  0.248931 Cos[ϕ] Sin[θ] + 3.33861 Cos[θ] Cos[ϕ] Sin[θ] +
  2.64507 Cos[2 ϕ] Sin[θ]^2 + 0.642694 Sin[θ] Sin[ϕ] +
  8.11311 Cos[θ] Sin[θ] Sin[ϕ] + 2.34011 Sin[θ]^2 Sin[2 ϕ],
 29.0653 - 1.16871 Cos[θ] - 6.70905 Cos[θ]^2 - 5.18537 Cos[ϕ] Sin[θ] +
  3.70162 Cos[θ] Cos[ϕ] Sin[θ] - 4.78784 Cos[2 ϕ] Sin[θ]^2 -
  7.12255 Sin[θ] Sin[ϕ] + 10.9624 Cos[θ] Sin[θ] Sin[ϕ] + 10.7787 Sin[θ]^2 Sin[2 ϕ],
 20.8489 + 0.389885 Cos[θ] + 0.901003 Cos[θ]^2 - 1.12934 Cos[ϕ] Sin[θ] -
  5.26629 Cos[θ] Cos[ϕ] Sin[θ] - 1.44079 Cos[2 ϕ] Sin[θ]^2 -
  1.57621 Sin[θ] Sin[ϕ] + 1.75637 Cos[θ] Sin[θ] Sin[ϕ] + 2.4043 Sin[θ]^2 Sin[2 ϕ]}
```

In[200]:=
```
Off[ParametricPlot3D::"ppcom"];
granulae = Show[MapThread[ParametricPlot3D[Reverse[
       #1 {α, 1, 1} + .15 #2 { Cos[θ], Sin[θ] Cos[ϕ], Sin[θ] Sin[ϕ]}], {θ, 0, π},
      {ϕ, 0, 2 π}, DisplayFunction → Identity] &, {maximumpositions,
    fitresults}] /. Polygon[q_] → {EdgeForm[], Polygon[q]},
  DisplayFunction → $DisplayFunction, ImageSize → 250,
  AxesLabel → {"z", "y", "x"}, Lighting → True];
```



Show the granules in a separate Java viewer window to play with the 3D dataset interactively:

```
<< JLink`;
InstallJava[];
liveApplet = JavaNew["Live"];
liveFrame = JavaNew["com.wolfram.jlink.MathAppletFrame",
 liveApplet, {"INPUT=" <> ToString[InputForm[N[granulae]]],
 "WIDTH=800", "HEIGHT=800"}];

liveFormWrite["c:/tmp/Lysosomes.dat", granulae];
```

Acknowledgement

The author thanks Dr. Marc van Zandvoort and Dr. Wim Engels of Maastricht University for the kind supply of the 2-photon microscopy images, and discussions on the problem.

Example 5: Eigenpatches

Gaussian Derivatives and Eigen-images

It has been shown that the so-called Eigen-images of a large series of image small patches

have great similarity to partial Gaussian derivative functions [Olshausen1996, Olshausen1997]. The resulting images are also often modeled as Gabor patches and wavelets.

In[201]:=
```
imageFile = GetURL["http://www.bmi2.bmt.tue.nl/image-analysis/People/-
BRomeny/FEV/images/mr256.gif"];
im = Import[imageFile,"GIF"][[1,1]];
DeleteFile[imageFile];
```

In[204]:=
```
δ = 12;
ListDensityPlot[im, Epilog →
   {Gray, Table[Line[{{i, j}, {i + δ, j}, {i + δ, j + δ}, {i, j + δ}, {i, j}}],
     {j, 2, 256, 15}, {i, 2, 256, 15}]}, ImageSize → 300];
```

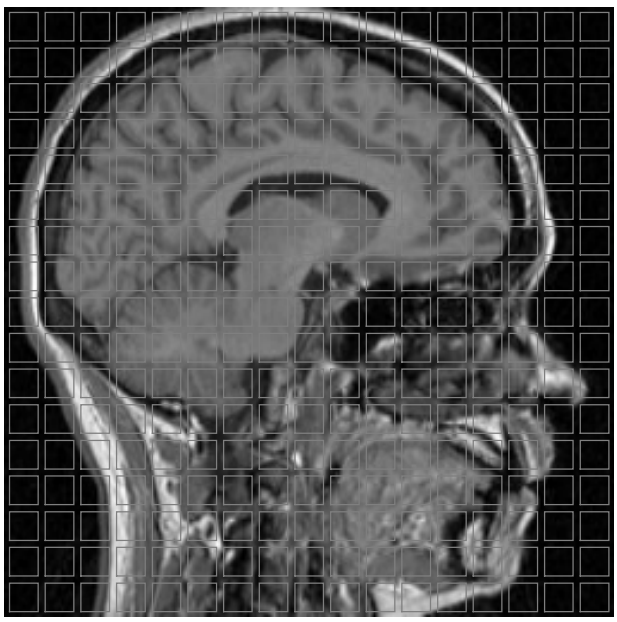


Figure 21. Location of the 289 small 12×12 pixel patches taken from a 256^2 image of a forest scene.

The small 12×12 images are sampled with SubMatrix:

```
In[206]:= set = Table[SubMatrix[im, {j, i}, {δ, δ}], {j, 2, 256, 15}, {i, 2, 256, 15}];
          Dimensions[set]

Out[206]= {17, 17, 12, 12}
```

and converted into a matrix m with 289 rows of length 144. We multiply each small image with a Gaussian weighting function to simulate the process of observation, and subtract the global mean:

```
In[207]:= σ = 4; g = Table[Exp[-(x^2 + y^2)/(2 σ^2)], {x, -5.5, 5.5}, {y, -5.5, 5.5}];
          set2 = Map[g # &, set, {2}];
          m = Flatten[Map[Flatten, set2, {2}], 1]; mean = Plus @@ # / Length[#] &;
          m = N[m - mean[Flatten[m]]]; Dimensions[m]

Out[210]= {289, 144}
```

We calculate $m^T m$, a 144^2 matrix with the Dot product, and check that it is a square matrix:

```
In[211]:= Dimensions[mTm = N[Transpose[m].m]]

Out[211]= {144, 144}
```

The calculation of the 144 Eigen-values of a 144^2 matrix goes fast in *Mathematica*. Essential is to force the calculations to be done numerically with the function N[]. Because mTm is a symmetric matrix, built from two 289×144 size matrices, we have 144 (nonzero) Eigen-values:

```
In[212]:= Short[Timing[evs = eigenvalues = Eigenvalues[mTm]], 5]

Out[212]//Short=
          {0.109 Second, {2.94085×10^7, 8.22104×10^6, 3.41456×10^6, 2.04713×10^6,
            866307., 774802., <<133>>, 12.4795, 10.693, 9.79994, 8.49766, 6.89998}}
```

We calculate the Eigenvectors of the matrix mTm and construct the first Eigen-image by partitioning the resulting 144×1 vector 12 rows. All Eigen-vectors normalized: unity length.

```
In[213]:= eigenvectors = Eigenvectors[mTm];
          eigenimages = Table[Partition[eigenvectors[[i]], δ], {i, 1, 8}];
          DisplayTogetherArray[ListDensityPlot /@ eigenimages, ImageSize → 350];
          DisplayTogetherArray[ListPlot3D /@ eigenimages, ImageSize → 350];
```

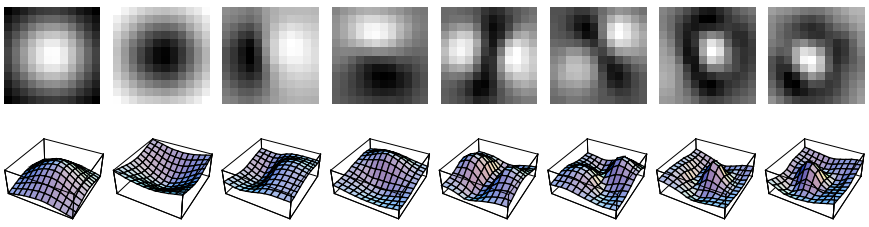


Figure 12.22. The first 8 Eigen-images of the 289 patches from figure 12.10.

```
In[216]:= noise = Table[Random[], {256}, {256}]; δ = 12;
          set = Table[SubMatrix[noise, {j, i}, {δ, δ}], {j, 3, 256, 15}, {i, 3, 256, 15}];
          m = Flatten[Map[Flatten, set, {2}], 1];
          m = N[m - mean[Flatten[m]]]; mTm = N[Transpose[m].m];
          {eigenvaluesn, eigenvectorsn} = Eigensystem[mTm];
          eigenimagesn = Table[Partition[eigenvectorsn[[i]], δ], {i, 1, 8}];

          DisplayTogetherArray[ListDensityPlot /@ eigenimagesn, ImageSize → 350];
```

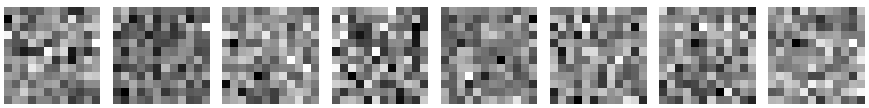


Figure 12.23. The first 8 Eigen-images of 289 patches of 12×12 pixels of white noise. Note that none of the Eigen-images contains any structure.

Note that the distribution of the Eigen-values for noise are much different from those of a structured image. They are much smaller, and the first ones are markedly less pronounced. Here we plot both distributions:

```
In[223]:= DisplayTogether[
            LogListPlot[evs, PlotJoined → True, PlotRange → {.1, Automatic}],
            LogListPlot[eigenvaluesn, PlotJoined → True], ImageSize → 250];
```

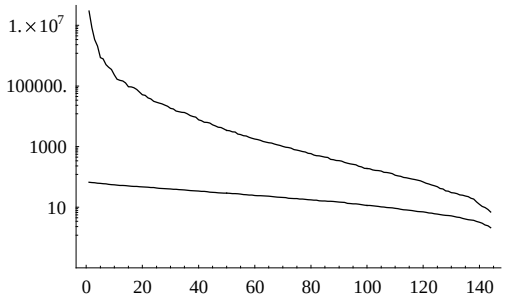


Figure 12.24. Nonzero Eigen-values for a structured image (upper) and white noise (lower).

When we extract 49×49 = 2401 small images of 12×12 pixels at each 5 pixels, so they slightly overlap, we get better statistics.

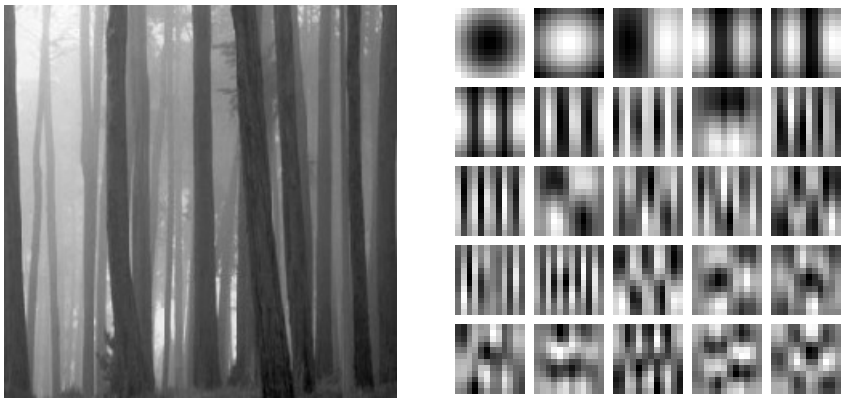
A striking result is obtained when the image contains primarily vertical structures, like trees. We then obtain Eigenpatches resembling the horizontal high order Gaussian derivatives / Gabor patches (see figure 12.25).

```
In[224]:= imageFile = GetURL["http://www.bmi2.bmt.tue.nl/image-analysis/People/-
          BRomeny/FEV/images/forest02.gif"];
          im = Import[imageFile,"GIF"][[1,1]];
          DeleteFile[imageFile];
```

```
In[227]:= δ = 12;
          set = Table[SubMatrix[im, {j, i}, {δ, δ}], {j, 2, 246, 5}, {i, 2, 246, 5}];
          δδ = (δ - 1) / 2; σ = δδ; g = Table[N[Exp[-(x^2 + y^2)/(2 σ^2)]], {x, -δδ, δδ}, {y, -δδ, δδ}];
          set2 = Map[g # &, set, {2}];
          m = Flatten[Map[Flatten, set2, {2}], 1]; mean = Plus @@ # / Length[#] &;
```

```
m = N[m - mean[Flatten[m]]]; mTm = N[Transpose[m].m];
eigenvectors = Eigenvectors[mTm];
eigenimages = Table[Partition[eigenvectors[[i]], δ], {i, 1, 25}];
```

In[232]:=
```
Block[{$DisplayFunction = Identity}, p1 = ListDensityPlot[im];
 p2 = Show[GraphicsArray[Partition[ListDensityPlot /@ eigenimages, 5]]];]
Show[GraphicsArray[{p1, p2}, ImageSize → 350]];
```



Eigen-images for 2401 slightly overlapping patches of 12×12 pixels from the image of the vertical trees. Note that the first Eigenpatches resemble the high order horizontal derivatives.