6. Differential structure of images

"*If I had more time, I would have written you a shorter letter*", Pascal (1623-1662)

6.1 The differential structure of images

In this chapter we will study the differential structure of discrete images in detail. This is the structure described by the local multi-scale derivatives of the image. We start with the development of a toolkit for the definitions of heightlines, local coordinate systems and independence of our choice of coordinates.

```
<< FrontEndVision`FEV`; Off[General::spell];
Show[Import["Spiral CT abdomen.jpg"], ImageSize -> 170];
```

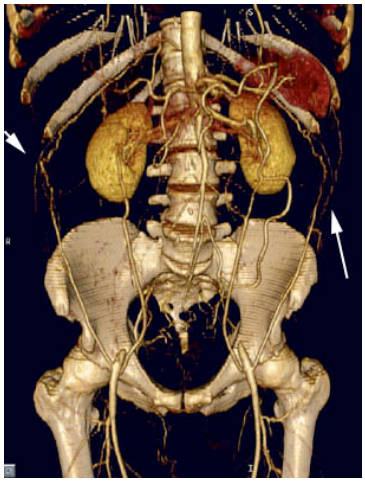

Figure 6.1 An example of a need for segmentation: 3D rendering of a spiral CT acquisition of the abdomen of a patient with Leriche's syndrome (EuroRAD case #745, authors R. Brillo, A. Napoli, S. Vagnarelli, M. Vendola, M. Benedetti Valentini, 2000, www.eurorad.org).

We will use the tools of *differential geometry*, a field designed for the structural description of space and the lines, curves, surfaces etc. (a collection known as *manifolds)* that live there.

We develop strategies for the generation of formulas for the detection of particular *features*, that detect special, semantically circumscribed, local meaningful structures (or properties) in the image. Examples are edges, corners, T-junctions, monkey-saddles and many more. We develop operational detectors in *Mathematica* for all features described.

One can discriminate local and multi-local methods in image analysis. We specifically discuss here local methods, at a particular local neighborhood (pixel). In later chapters we look at multi-local methods, and enter the realm of how to connect local features, both by studying similarity in properties with neighboring pixels ('perceptual grouping'), relations

over scale ('deep structure') and relations given by a particular *model*. We will discuss the use of the local features developed in this chapter into 'geometric reasoning'.

Why do we need to go in detail about local image derivatives? Combinations of derivatives into expressions give nice feature detectors in images. It is well known that $\sqrt{\left(\frac{\partial L}{\partial x}\right)^2 + \left(\frac{\partial L}{\partial y}\right)^2}$ is a good edge detector, and $\left(\frac{\partial L}{\partial y}\right)^2 \frac{\partial^2 L}{\partial x^2} - 2 \frac{\partial L}{\partial x} \frac{\partial L}{\partial y} \frac{\partial^2 L}{\partial x \partial y} + \left(\frac{\partial L}{\partial x}\right)^2 \frac{\partial^2 L}{\partial y^2}$ is a good corner detector. But how do we come to such formula's? We can make an infinite number of such expressions. What constraints can/should we impose to come to a reasonably small set of *basis* descriptors? Is there such a *basis*? It turns out there is, and in this chapter we will derive a formal complete set of such descriptive elements.

A very important constraint in the development of tools for the description of image structure is to be independent of the choice of coordinates. We will discuss coordinate transformations, like translations, rotations, zooming, in order to find a way to detect features *invariant* to such coordinate transformations. In fact, we will discuss three 'languages' in which it is easy to develop a general strategy to come up with quite complex image structure detectors:

gauge coordinates, Cartesian tensors, and algebraic polynomial invariants. All these methods have firm roots in mathematics, specifically differential geometry, and form an ideal substrate for the true understanding of image structure.

We denote the function that describes our landscape (the image) with $L(x, y)$ throughout this book, where L is the physical property measured in the image. Examples of L are luminance, T1 or T2 relaxation time (for MRI images), linear X-ray absorption coefficient (for CT images), depth (for range images) etc. In fact, it can be any scalar value. The coordinates x, y are discrete in our case, and denote the locations of the pixel. If the image is 3-dimensional, e.g. a stack of images from an MRI or CT scanner, we write $L(x, y, z)$. A scale-space of images, observed at a range of scales σ is written as $L(x, y; \sigma)$. We write a semicolon as separator to highlight the fact that σ is *not* just another spatial variable. If images are a function of time as well, we write e.g. $L(x, y, z; t)$ where t is the time parameter. In chapter 17 we will develop scale-space theory for images sampled over time. In chapter 15 we study the extra dimension of color in images and derive differential features in color-space, and in chapter 13 we derive methods for the extraction of motion, a vectorial property with a magnitude and a direction. We firstly focus on static, spatial images.

6.2 Isophotes and flowlines

Lines in the image connecting points of equal intensity are called *isophotes*. They are the heightlines of the intensity landscape when we consider the intensity as 'height'. Isophotes in 2D images are curves, and in 3D surfaces, connecting points with equal luminance.

(Greek: isos ($\iota\sigma o\varsigma$) = equal, photos ($\phi o\tau o\varsigma$) = light): $L(x, y) = \text{constant}$ or $L(x, y, z) = \text{constant}$. This definition however is for a continuous function. But the scale-space paradigm solves this: in discrete images isophotes exist because these are *observed*

images, and thus *continuous* (which means: infinitely differentiable, or C^{∞}). Lines of constant value in 2D are **Contours** in *Mathematica*, which can be plotted with **ContourPlot**. Figure 6.2 illustrates this for a blurred version of a 2D image.

```
im = Import["mr128.gif"][[1, 1]];
Block[{$DisplayFunction = Identity, dp, cp},
 dp = ListDensityPlot[gD[im, 0, 0, #]] & /@ {1, 2, 3};
 cp = ListContourPlot[gD[im, 0, 0, #],
      ContourStyle → List /@ Hue /@ (.1 Range[10])] & /@ {1, 2, 3};
 pa = MapThread[Show, {dp, cp}]]; Show[GraphicsArray[pa],
 ImageSize -> 400];
```

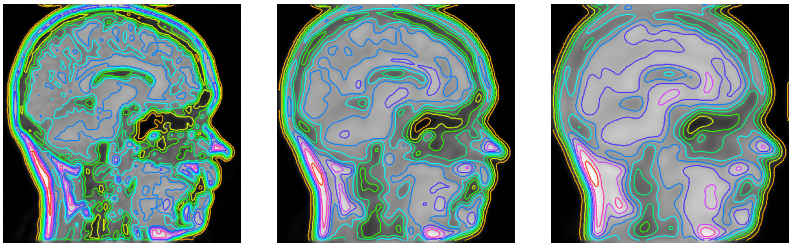

Figure 6.2 Isophotes of an image at various blurring scales: from left to right: $\sigma = 1$, $\sigma = 2$ and $\sigma = 3$ pixels. Image resolution 128^2. Ten isophotes are plotted in each image, equidistant over the available intensity range. Each is shown in a different color, superimposed over the grayvalues. Notice that the isophotes get more 'rounded' when we blur the image. When we consider the intensity distribution of a 2D image as a landscape, where the height is given by the intensity, isophotes are the heightlines.

Isophotes are important elements of an image. In principle, all isophotes together contain the same information as the image itself. The famous and often surprisingly good working segmentation method by thresholding and separating the image in pixels lying within or without the isophote at the threshold luminance is an example of an important application of isophotes. Isophotes have the following properties:

■ isophotes are closed curves. Most (but not all, see below) isophotes in 2D images are a so-called Jordan curve: a non-self-intersecting planar curve topologically equivalent to a circle;
■ isophotes can intersect themselves. These are the critical isophotes. These always go through a saddlepoint;
■ isophotes do not intersect other isophotes;
■ any planar curve is completely described by its *curvature*, and so are isophotes. We will define and derive the expression for isophote curvature in the next section.

■ isophote shape is independent of grayscale transformations, such as changing the contrast or brightness of an image.

A special class of isophotes is formed by those isophotes that go through a *singularity* in the intensity landscape, thus through a minimum, maximum or saddle point. At these places the intensity landscape is horizontal, the local spatial derivatives are all zero. Only at saddle points isophotes intersect themselves, and just above and below this intersection its neighbor

isophotes have different *topology*: they have split from one curve into two, or merged from two curves into one.

```
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];
```

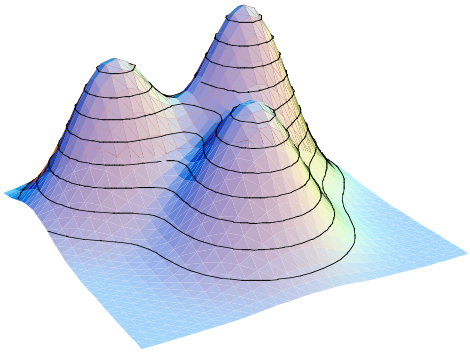

Figure 6.3 Isophote on a 2D 'landscape' image of 3 Gaussian blobs, depicted as heightlines. The height is determined by the intensity. The height plot is depicted slightly lower (-0.0002) in order to show the full extent of the isophotes.

At a minimum or maximum the isophote has shrunk to a point, and going to higher or lower intensity gives rise to the creation or disappearance of isophotes. This is best illustrated with an example of an image where only three Gaussian 'blobs' are present (see figure 6.3). The saddle points are in between the blobs. Isophotes through saddles and extrema are called *critical isophotes*.

We show the dynamic event of a 'split' and a 'merge' of an isophote by the behaviour of a two-parameter family of curves, the Cassinian ovals: $(x^2 + y^2 + a^2) - b^2 - 4\,a^2\,x^2 = 0$.

Famous members of Cassini functions are the circle (**`cassini[x,y,a=0,b]`**) and the lemniscate of Bernouilli (**`cassini[x,y,a=b,b]`**). The limaçon function, a generalization of the cardioid function, shows how we can get self-intersection where the new loop is formed within the isophote's inner domain. Here are the plots:

```
cassini[x_, y_, a_, b_] := (x^2 + y^2 + a^2)^2 - b^2 - 4 a^2 x^2;
DisplayTogetherArray[{
   ImplicitPlot[cassini[x, y, #, 4] == 0, {x, -5, 5}] & /@ {1.99, 2., 2.01},
   ParametricPlot[(2 Cos[t] + #) {Cos[t], Sin[t]}, {t, 0, 2 π}] & /@
    {3, 2., 1}}, ImageSize -> 400];
```

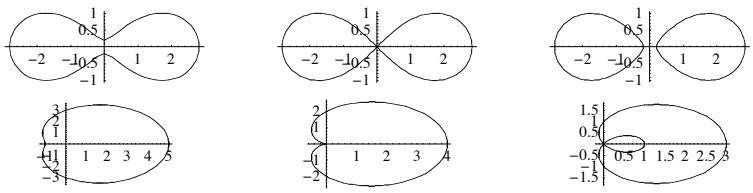

Figure 6.4 Top row: Split and merge of an isophote just under, at and above a saddle point in the image, simulated with a Cassini curve. Bottom row: Self intersection with an inner loop, simulated with the limaçon function. Examples taken from the wonderful book by Alfred Gray [Gray1993].

Isophotes in 3D are surfaces. Here is an example of the plotting of 4 isophote surfaces of a discrete dataset. We use the versatile OpenGL viewer MathGL3d developed by Jens-Peer Kuska: http://phong.informatik.uni-leipzig.de/~kuska/mathgl3dv3/

```
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];
```

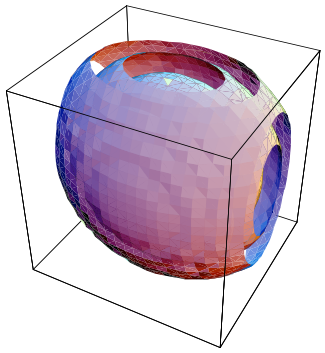

Figure 6.5 Isophotes in 3D are surfaces. Shown are the isophotes connecting all voxels with the values 0.1, 1, 10 and 100 in the discrete dataset of two neighboring 3D Gaussian blobs.

The calculations with the native command **`ListContourPlot3D`** take take much longer.

Flowlines are the lines everywhere perpendicular to the isophotes. E.g. for a Gaussian blob the isophotes are circles, and the flowlines are radiating lines from the center. Flowlines are the *integral curves* of the gradient, made up of all the small little gradient vectors in each point integrated to a smooth long curve. In 2D, the flowlines and the isophotes together form a *mesh* or *grid* on the intensity surface.

Figure 6.6 shows such a grid of the isophotes and flowlines of a 2D Gaussian blob (we have left out the singularity).

```
DisplayTogether[
  ShadowPlot3D[-gauss[x, 5] gauss[y, 5], {y, -15, 15}, {x, -15, 15}],
CartesianMap[Exp, { -π, π}, { -π, π}],
  ImageSize -> 200, AspectRatio -> 1];
```

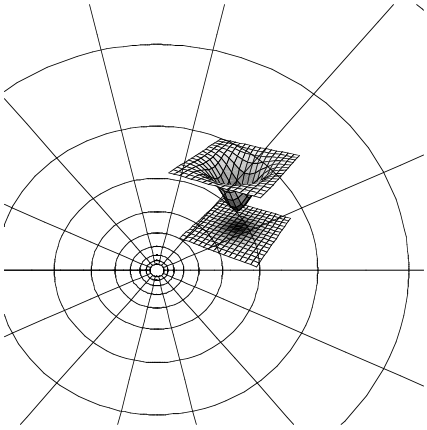

Figure 6.6 Isophotes and flowlines on the slope of a Gaussian blob. The circles are the isophotes, the flowlines are everywhere perpendicular to them. Inset: The height and intensity map of the Gaussian blob.

Just as in principle all isophotes together completely describe the intensity surface, so does the set of all flowlines. Flowlines are the *dual* of isophotes, isophotes are the *dual* of flowlines. One set can be calculated from the other. Just as the isophotes have a singularity at minima and maxima in the image, so have flowlines a singularity in direction in such points.

6.3 Coordinate systems and transformations

We will now apply the complete family of well behaving differential operators developed in the first chapter for the detection of local differential structure in images. The set of derivatives taken at a particular location is a *language* from which we can make a description of a local feature. We can make assemblies of the derivatives to any order, in any combination. Local structure is the local shape of the intensity landscape, like how sloped or curved it is, if there are saddlepoints, etc. The first order derivative gives us the slope, the second order is related to how curved the landscape is, etc.

In mathematical terms the image derivatives show up in the so-called Taylor expansion of our image function.

The Taylor expansion describes the function 'a little further up': If we move a little distance (δx, δy) away from the pixel where we stand, the Taylor expansion -or Taylor series- is given by (we take the expansion in the origin (0, 0) for notational convenience):

$$L(\delta x, \delta y) = L(0, 0) + \left(\frac{\partial L}{\partial x}\,\delta x + \frac{\partial L}{\partial y}\,\delta y\right) + \frac{1}{2!}\left(\frac{\partial^2 L}{\partial x^2}\,\delta x^2 + \frac{\partial^2 L}{\partial x\,\partial y}\,\delta x\,\delta y + \frac{\partial^2 L}{\partial y^2}\,\delta y^2\right) + \frac{1}{3!}\left(\frac{\partial^3 L}{\partial x^3}\,\delta x^3 + \frac{\partial^3 L}{\partial x^2\,\partial y}\,\delta x^2\,\delta y + \frac{\partial^3 L}{\partial x\,\partial y^2}\,\delta x\,\delta y^2 + \frac{\partial^3 L}{\partial y^3}\,\delta y^3\right) + O(\delta x^4, \delta y^4)$$

We see al the partial derivatives appearing. The spatial derivatives are taken at the location (0,0), e.g. $\frac{\partial^2 L}{\partial x^2}\Big|_{(0,0)}$. The first-order, second-order and third-order terms are grouped in brackets. Such groups of all terms of a specific order together are called 'binary forms'. The list goes to infinity, so we have to cut-off somewhere. The above series is an approximation to the third order, and the final expression $O(\delta x^4, \delta y^4)$ indicates that there is more, a rest term of order 4 and higher in δx and δy. *Mathematica* has the command **Series** to make a Taylor expansion. Here is the Taylor series for $L(x, y)$ for δx to second order and then expanded to second order by δy :

```
Series[L[δx, δy], {δx, 0, 2}, {δy, 0, 2}]
```

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

This expansion says essentially that we get a good approximation of the intensity landscape a little bit (δx, δy) further away from the origin (0, 0), when we first climb up over δx and δy with a slope given by the first derivative, the tangent. Then we come close, but not exactly. We can come somewhat better approximated when we include also the second order derivative, indicating how curved locally our landscape is. Etc. Taking into account more and more higher order terms gives us a better approximation and finally with the infinite series we have an exact description.
Our most important constraint for a good local image descriptor comes from the requirement that we want to be independent of our choice of coordinates. The coordinate system used the most is the Cartesian coordinate system (invented by and named after Descartes, a brilliant French mathematician from the 18th century): this is our familiar orthogonal (x, y) or (x, y, z) coordinate system.

But it should not matter if we describe our local image structure in another coordinate system like a polar, cylindrical or rotated or translated version of our Cartesian coordinate system. Because the Cartesian system is the easiest to understand, we will deal only with changes in this coordinate system. The *frame* of the coordinate system is formed by the unit vectors pointing in the respective dimensions. What changes could occur to a coordinate system? Of course any modification is possible. We will focus on the change of orientation (rotation of the axes frame), translation (x and/or y shift of the axes frame), and zoom (multiplication of the length of the units along the axes with some factor).

The shear transformation (where the axes are no longer orthogonal) will not be discussed here; we limit ourselves to changes of the coordinates where they remain orthogonal.

```
DisplayTogetherArray[
  Show[Graphics[{Arrow[{0, 0}, #] & /@ {{1, 0}, {0, 1}},
     Red, PointSize[.04], Point[{.4, .6}]}],
   Frame -> True, Axes -> True, AspectRatio -> 1],
  Show[Graphics3D[{arrow3D[{0, 0, 0}, #, True] & /@ {{1, 0, 0}, {0, 1, 0},
       {0, 0, 1}}, Red, PointSize[.04], Point[{.4, .6, .7}]}],
   Boxed -> True, BoxRatios -> {1, 1, 1}, Axes -> True], ImageSize -> 250];
```

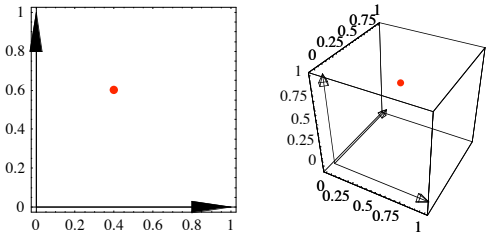

Figure 6.7 Use of graphics primitives in *Mathematica*: the coordinate unit vectors in 2D and 3D.

We call all the possible instantiations of a transformation the transformation *group*. So all rotations form the rotational group, the group of translations is formed by all translations. We now consider the transformation of the frame vectors.

Mathematically, the operation of a transformation is described by a matrix, the transformation matrix. E.g. rotation of a vector over an angle ϕ is described by the rotation matrix in 2D:

```
RotationMatrix2D[ϕ] // MatrixForm
```

$$\begin{pmatrix} \text{Cos}[\phi] & \text{Sin}[\phi] \\ -\text{Sin}[\phi] & \text{Cos}[\phi] \end{pmatrix}$$

The angle ϕ is defined as clockwise for the positive direction. In 3D it gets a little more complicated, as we have three angles to rotate over (these are called the 'Euler' angles):

```
RotationMatrix3D[ψ, Θ, ϕ]
```

```
{{Cos[ϕ] Cos[ψ] - Cos[Θ] Sin[ϕ] Sin[ψ],
   Cos[Θ] Cos[ψ] Sin[ϕ] + Cos[ϕ] Sin[ψ], Sin[Θ] Sin[ϕ]},
  {-Cos[ψ] Sin[ϕ] - Cos[Θ] Cos[ϕ] Sin[ψ],
   Cos[Θ] Cos[ϕ] Cos[ψ] - Sin[ϕ] Sin[ψ], Cos[ϕ] Sin[Θ]},
  {Sin[Θ] Sin[ψ], -Cos[ψ] Sin[Θ], Cos[Θ]}}
```

In general a transformation is described by a set of equations:

$$\begin{aligned} x'_1 &= f_1(x_1, x_2, \ldots, x_n) \\ &\vdots \\ x'_n &= f_n(x_1, x_2, \ldots, x_n) \end{aligned}$$

When we transform a space, the volume often changes, and the density of the material inside is distributed over a different volume. To study the change of a small volume we need to consider $\frac{\partial \vec{x}'}{\partial \vec{x}}$, which is the matrix of first order partial derivatives.

We have $J = \frac{\partial \vec{x}'}{\partial \vec{x}} = \begin{pmatrix} \frac{\partial (x')_1}{\partial x_1} & \cdots & \frac{\partial (x')_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial (x')_n}{\partial x_1} & \cdots & \frac{\partial (x')_n}{\partial x_n} \end{pmatrix}$. This matrix is called the *Jacobian matrix*, named after Carl Jacobi (1804-1851), a Prussian mathematician. The Jacobian can be computed in *Mathematica* with

```
jacobianmatrix[functions_List, variables_List] :=
 Outer[D, functions, variables]
```

If we consider the change of the infinitesimally small volume $d\,x'_1\,d\,x'_2 \ldots d\,x'_n = \left| \frac{\partial \vec{x}'}{\partial \vec{x}} \right| d\,x_1\,d\,x_2 \ldots d\,x_n$ we see that the determinant of the Jacobian matrix (also called the *Jacobian*) is the factor which corrects for the change in volume. When the Jacobian is unity, we call the transformation a *special* transformation.
The transformation in matrix notation is expressed as $\vec{x}' = A\,\vec{x}$, where $\vec{x}'$ is the transformed vector, $\vec{x}$ is the input vector, and $A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}$ is the transformation matrix. When the coefficients of A are constant, we have a *linear* transformation, often called an *affine* transformation. In *Mathematica* (note the dot product between the matrix and the vector):

```
Clear[x, y]; A = {{a11, a12}, {a21, a22}}; x = {x1, x2};
x' = 1/Det[jacobianmatrix[A.x, x]] A.x
```

$$\left\{\frac{a_{11}\,x_1 + a_{12}\,x_2}{-a_{12}\,a_{21} + a_{11}\,a_{22}},\ \frac{a_{21}\,x_1 + a_{22}\,x_2}{-a_{12}\,a_{21} + a_{11}\,a_{22}}\right\}$$

▲ Task 6.1 Show that the Jacobian of the transformation matrices `RotationMatrix2D[ϕ]` and `RotationMatrix3D[ϕ,ϑ,ψ]` are unity.

A rotation matrix that rotates over zero degrees is the *identity matrix* or the *symmetric tensor* or *δ-operator*:

```
δ = RotationMatrix2D[0]; δ // MatrixForm
```

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

and the matrix that rotates over 90 degrees ($\pi/2$ radians) is called the *antisymmetric tensor*, the *ϵ-operator* or the *Levi-Civita tensor*:

```
ϵ = RotationMatrix2D[π / 2]; ϵ // MatrixForm
```

$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

Let us study an example of a rotation: a unit vector under 45^0 is rotated over 110^0 clockwise:

```
v = {Sqrt[2]/2, Sqrt[2]/2}; v' = RotationMatrix2D[110 2 π/360].v // N
```

```
{0.422618, -0.906308}
```

```
Show[Graphics[{Arrow[{0, 0}, #] & /@ {v, v'}, Text["v", {.8, .8}],
    Text["v'", {.55, -.8}]}], PlotRange -> {{-1, 1}, {-1, 1}},
  Frame -> True, Axes -> True, AspectRatio -> 1, ImageSize -> 100];
```

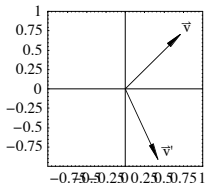

Figure 6.8 The vector $\vec{v}'$ is rotated by the action of the rotation matrix operator on the vector $\vec{v}$.

What we want is *invariance* under the transformations of translation and rotation. A function is said to be invariant under a group of transformations, if the transformation has no effect on the value of the function. The *only* geometrical entities that make physically sense are invariants. In the words of Hermann Weyl: "any invariant has a specific *meaning*", and as such they are widely studied in computer vision theories.

An example: The derivative to x is *not* invariant to rotation; if we rotate the coordinate system, or the image, we get in general a completely different value for the value of the derivative at that point. The same applies to the derivative to y. However, the combination $\sqrt{(\frac{\partial L}{\partial x})^2 + (\frac{\partial L}{\partial y})^2}$ is invariant, as can be seen from the following: We denote derivatives with a lower index: $L_x \equiv \frac{\partial L}{\partial x}$. The *length* of the *gradient vector* (L_x, L_y) is the scalar

```
Sqrt[{Lx, Ly}.{Lx, Ly}]
```

$$\sqrt{Lx^2 + Ly^2}$$

We used here again the **Dot** (.) product of vectors. When we now rotate each vector (L_x, L_y) with the rotation matrix over an arbitrary angle ϕ, we get

```
√((RotationMatrix2D[φ].{Lx, Ly}).(RotationMatrix2D[φ].{Lx, Ly}))
```

$$\sqrt{(Ly\,Cos[\phi] - Lx\,Sin[\phi])^2 + (Lx\,Cos[\phi] + Ly\,Sin[\phi])^2}$$

```
Simplify[%]
```

$$\sqrt{Lx^2 + Ly^2}$$

Invariance proved for this case. Invariants are so important, that the lower-order ones have a name. E.g. the scalar $\sqrt{(\frac{\partial L}{\partial x})^2 + (\frac{\partial L}{\partial y})^2}$ is called the *gradient magnitude*, the vector operator $\vec{\nabla} \equiv \{\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\}$ is called the *nabla operator*. So $\vec{\nabla} L$ is the gradient of L. $\vec{\nabla}.(\vec{\nabla} L) = \frac{\partial^2 L}{\partial x^2} + \frac{\partial^2 L}{\partial y^2}$ is called the *Laplacian*. Note that the gradient of the gradient $\vec{\nabla}(\vec{\nabla} L) = \begin{pmatrix} \frac{\partial^2 L}{\partial x^2} & \frac{\partial^2 L}{\partial x \partial y} \\ \frac{\partial^2 L}{\partial x \partial y} & \frac{\partial^2 L}{\partial y^2} \end{pmatrix}$ is the matrix of second order derivatives, or the *Hessian matrix* (this is not an invariant).

▲ Task 6.2 Show that the Laplacian is an invariant under rotation, in 2D and 3D.

In the sequel, we will only consider *orthonormal transformations*. These are also called Euclidean transformations. Orthonormal transformations are *special orthogonal* transformations (the Jacobian is unity). With orthogonal transformations the orthogonality of the coordinate frame is preserved. An orthonormal transformation preserves lengths of vectors and angles between vectors, i.e. it preserves a symmetric inner product $< \vec{x}, \vec{y} >$. When T is the orthogonal transformation, this means that $< \vec{x}, \vec{y} >= < T\vec{x}, T\vec{y} >$.

The transformation matrix of an orthogonal transformation is an *orthogonal matrix*. They have the nice property that they are always invertible, as the inverse of an orthogonal matrix is equal to its transpose: $A^{-1} = A^T$. A matrix *m* can be tested to see if it is orthogonal using

```
OrthogonalQ[m_List?MatrixQ] :=
  (Transpose[m].m == IdentityMatrix[Length[m]]);
```

Of course, there are many groups of transformations that can be considered, such as projective transformations (projecting a 3D world onto a 2D surface). In biomedical imaging mostly orthogonal transformations are encountered, and on those which will be the emphasis of the rest of this chapter.

Notice that with invariance we mean invariance for the transformation (e.g. rotation) of the coordinate system, not of the image. The value of the local invariant properties is also the same when we rotate the image. There is however an important difference between image rotation, and coordinate rotation. We specifically mean here the *local* independence of rotation, for that particular point. See also figure 6.9. If we study the rotation of the whole image, we apply the same rotation to every pixel.

Here, we want in every point a description which is independent to the rotation of the local coordinates, so we may as well rotate our coordinates in every pixel differently. Invariance for rotation in this way means something different than a rotation of the image. There would be no way otherwise to recognize rotated images from non-rotated ones!

```
Show[Import["Thatcher illusion.jpg"], ImageSize -> 330];
```

Figure 6.9 The "Thatcher illusion", created by P. Thompson [Thompson1980], shows that local rotations of image *patches* are radically different from the local coordinate rotation invariance, and that we are not used to (i.e. have no associative set in our memory) for sights that we seldomly see: faces upside down. Rotate the images 180 degrees to see the effect.

In particular, we will see that specific scalar combinations of local derivatives give descriptions of local image structure invariant under a Euclidean transformation.

6.4 Directional derivatives

The directed first order nabla operator is given in 2D by $\vec{v}.\vec{\nabla}$, where $\vec{v}$ is a unit vector pointing in the specific direction. $\vec{v}.\vec{\nabla}$ is called the *directional derivative*. Let us consider some examples. We calculate the directional derivative for $\vec{v} = \left\{-\sqrt{2}, -\sqrt{2}\right\}$ and $\vec{v} = \left\{\sqrt{3}/2, 1/2\right\}$:

```
im = Import["mip147.gif"][[1, 1]];
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];
```

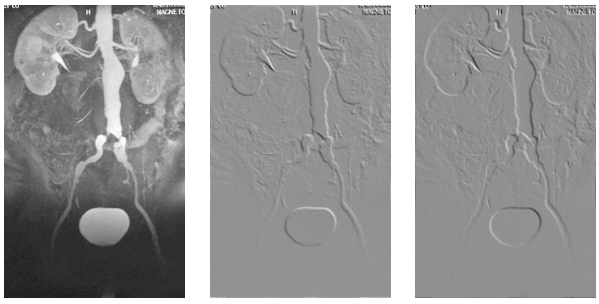

Figure 6.10 Directional derivatives. Image from the Eurorad database (www.eurorad.org), case 147.

6.5 First order gauge coordinates

We introduce the notion of *intrinsic geometry*: we like to have every point described in such a way, that if we have the same structure, or local landscape form, no matter the rotation, the description is always the same. This can be accomplished by setting up in each point a dedicated coordinate frame which is determined by some special local directions given by the landscape locally itself.

Consider yourself an ant on a surface, you can only see the direct vicinity, so the world looks locally very simple. We now fix *in each point separately* our local coordinate frame in such a way that one frame vector points to the direction of maximal change of the intensity, and the other perpendicular to it (90 degrees clockwise). The direction of maximal change of intensity is just the gradient vector $\vec{w} = \left(\frac{\partial L}{\partial x}, \frac{\partial L}{\partial y}\right)$. The perpendicular direction is $\vec{v} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} . \vec{w} = \left(\frac{\partial L}{\partial y}, -\frac{\partial L}{\partial x}\right)$. We can check: if we are on a slope going up in the y-direction only (the 'Southern' slope of a hill), we have as gradient $\left\{0, \frac{\partial L}{\partial y}\right\}$, because in the x-direction the slope is horizontal.

```
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];
```

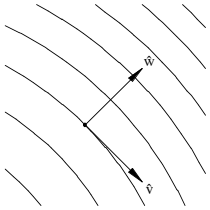

Figure 6.11 Local first order gauge coordinates $\{\hat{v}, \hat{w}\}$. The unit vector $\hat{v}$ is everywhere tangential to the isophote (line of constant intensity), the unit vector $\hat{w}$ is everywhere perpendicular to the isophote and points in the direction of the gradient vector.

We have now *fixed* locally the direction for our new intrinsic local coordinate frame $(\vec{v}, \vec{w})$. This set of local directions is called a *gauge*, the new frame forms the *gauge coordinates* and fixing the frame vectors with respect to the constant direction $\vec{w}$ is called: *fixing the gauge*. Because we discuss first order derivatives here, we call this a *first order gauge*. We can also derive a second order gauge from second order local differential structure, as we will see later.

We want to take derivatives with respect to the gauge coordinates.

As they are fixed to the object, no matter any rotation or translation, we have the following very useful result:

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.

From the derivatives with respect to the gauge coordinates, we always need to go to Cartesian coordinates in order to calculate the invariant properties on a computer. The transformation from the $(\hat{v}, \hat{w})$ from to the Cartesian $(\hat{x}, \hat{y})$ frame is done by implementing the definition of the directional derivatives. Important is that first a directional partial derivative (to whatever order) is calculated with respect to a *frozen* gradient direction. We call this direction (Lx, Ly). Then the formula is calculated which expresses the gauge derivative into this direction, and finally the frozen direction is filled in from the *calculated* gradient.

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

```
ŵ = {Lx, Ly} / Sqrt[Lx^2 + Ly^2]; v̂ = {{0, 1}, {-1, 0}}.ŵ;
```

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

```
v̂.{∂x #, ∂y #} &;
v̂.{∂x #, ∂y #} &;
```

The notation **(...#)&** is a 'pure function' on the argument **#**, e.g. **(#^2 + #^5) &** gives the sum of second and fifth power of some argument #, **D[#,x]&** (or equivalently **(∂x #) &**) takes the derivative of the variable **#** with respect to **x** (look in the Help browser to **Function** for more examples). So the construct of a pure function is the construct for an *operator*. This pure function can be applied to an argument by the familiar square brackets, e.g.

```
(#^2 + #^5) &[zz]
```

```
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:

```
Clear[f, L, Lx, Ly]; Unprotect[gauge2D];
gauge2D[f_, nv_ /; nv ≥ 0, nw_ /; nw ≥ 0] :=
  Module[{Lx, Ly, v, w}, w = {Lx, Ly} / Sqrt[Lx^2 + Ly^2];
   v = {{0, 1}, {-1, 0}}.w;
   Simplify[
    Nest[(v.{D[#1, x], D[#1, y]} &), Nest[(w.{D[#1, x], D[#1, 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 $\frac{\partial L}{\partial w}$:

```
Lw = gauge2D[L[x, y], 0, 1]
```

$$\sqrt{L^{(0,1)}[x, y]^2 + L^{(1,0)}[x, y]^2}$$

Using pattern matching with the function **shortnotation** we get more readable output:

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

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

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

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

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

0

As expected, because it is exactly what we put into the definition of $\frac{\partial L}{\partial v}$: it is the differentiation in the direction perpendicular to the gradient, so along the tangential direction of the isophote, and in this direction there is *no* change of the intensity function. But

```
Lvv = gauge2D[L[x, y], 2, 0] // shortnotation
```

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

is *not* zero, because it is constructed by first applying the directional derivative twice, and then fixing the gauge.

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

```
gauge2D[L[x, y], 0, 2] + gauge2D[L[x, y], 2, 0] // shortnotation
```

$$L_{xx} + L_{yy}$$

▲ Task 6.3 Show and explain that in the definition of the function gauge2D we cannot define $w = \{\partial_x L, \partial_y L\}$. We need to have the direction of the gauge fixed while computing the compound formula. Why?

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

```
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]]];

Show[{lp, ar}, Frame -> True, ImageSize -> 410];
```

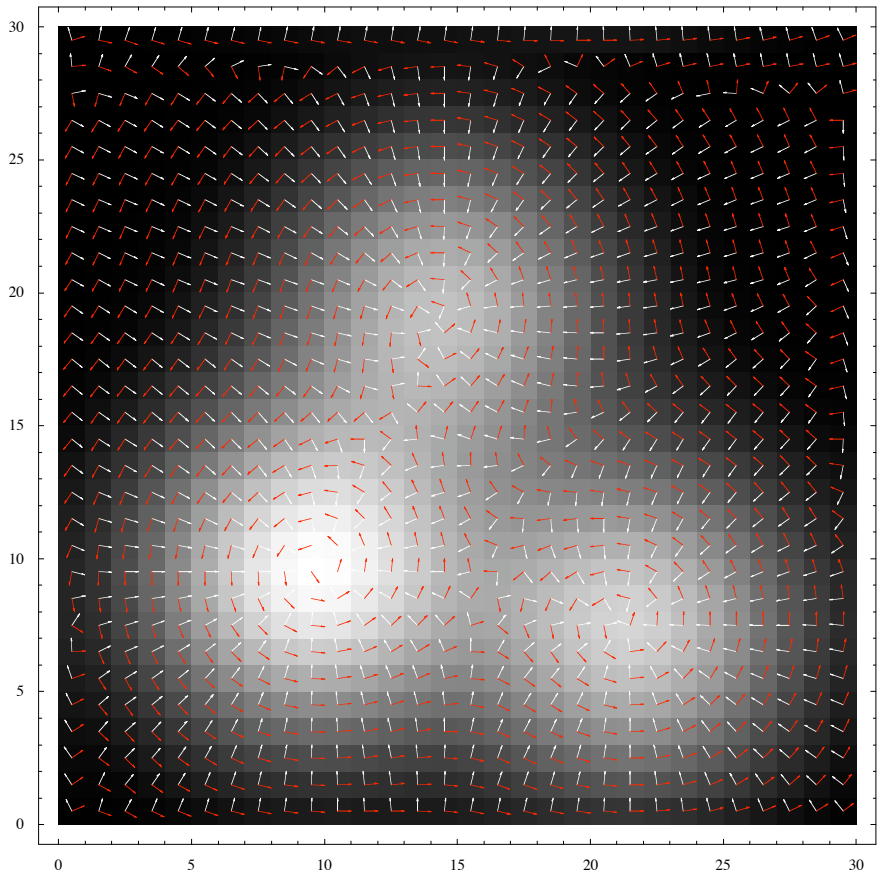

Figure 6.12 The gauge frame $\{w, v\}$ given for every pixel in a 30^2 image of three Gaussian blobs. The gradient direction w, calculated at a scale of σ=1 pixel, is indicated in white, and points always to higher intensity. They are (defined as) everywhere perpendicular to the isophotes and tangential to the flowlines. These vectors always point to extrema and saddle points. The v frame vector (in red) is $\pi/2$ radians rotated clockwise, they encircle the extrema, (defined as) tangential to the isophotes. (The boundary effects, most notably on the right, are due to the cyclic interpretation of the gradient calculation, which causes the image to be interpreted as infinitely repeated in all directions: the gradient direction changes over π, no artefact, but now well understood).

The gauge coordinates are not defined at 'horizontal points' in the intensity landscape, i.e. locations where $\sqrt{{L_x}^2 + {L_y}^2} = 0$, as is clear from the definition of the gauge coordinates. This occurs in saddle points and extrema (minima and maxima) of the intensity landscape, where both $L_x = 0$ and $L_y = 0$. In practice however this is not a problem: we have a finite number of such points, typically just a few, and we know from *Morse theory* that we can get rid of such a *singularity* by an infinitesimally small local change in the intensity landscape.

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.*

In other words: it is an invariant property where translation and/or rotation of the coordinate frame is irrelevant. This also means that polynomial combinations of these gauge derivative terms are invariant. We now have the toolkit to make invariants expressed in gauge derivatives to any order.

Here are a few other differential invariants of the image, which are now easily constructed:

```
gauge2D[L[x, y], 4, 0] // shortnotation
```

$$\frac{-4\,L_x^3\,L_{xyyy}\,L_y + 6\,L_x^2\,L_{xxyy}\,L_y^2 - 4\,L_x\,L_{xxxy}\,L_y^3 + L_{xxxx}\,L_y^4 + L_x^4\,L_{yyyy}}{(L_x^2 + L_y^2)^2}$$

```
gauge2D[L[x, y], 2, 1] // shortnotation
```

$$\frac{L_x^3\,L_{xyy} + L_x\,(L_{xxx} - 2\,L_{xyy})\,L_y^2 + L_{xxy}\,L_y^3 + L_x^2\,L_y\,(-2\,L_{xxy} + L_{yyy})}{(L_x^2 + L_y^2)^{3/2}}$$

In conclusion of this section, we have found a *complete family* of differential invariants, that are invariant for rotation and translation of the coordinate frame. They are called differential invariants, because they consist of polynomials with as coefficients partial derivatives of the image. In the next section we discuss some important members of this family. Only the lowest order invariants have a *name*, the higher orders become more and more exotic.

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:

```
Unprotect[gauge2DN];
gauge2DN[im_, nv_, nw_, σ_ /; σ > 0] :=
  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:

```
Clear[im, σ]; gauge2DN[im, 2, 0, 2]

(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:

```
im = Import["thorax02.gif"][[1, 1]];
DisplayTogetherArray[
  ListDensityPlot /@ {im, -gauge2DN[im, 0, 1, 1], -gauge2DN[im, 2, 0, 4]},
  ImageSize -> 400];
```

Figure 6.13 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.

6.6 Gauge coordinate invariants: examples

6.6.1 Ridge detection

L_{vv} is a good ridge detector, since at ridges the curvature of isophotes is large (see figure 6.13).

```
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];
```

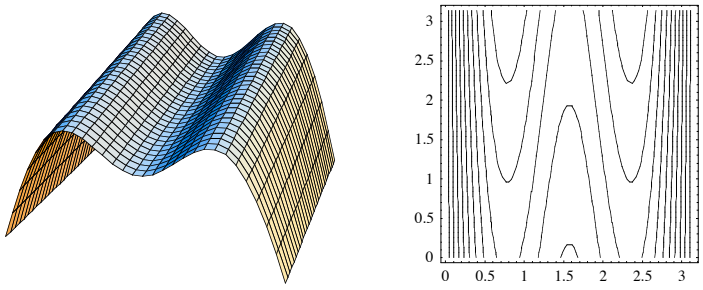

Figure 6.14 Isophotes are much more curved at the top of ridges and valleys then along the slopes of it. Left: a slightly sloping artificial intensity landscape with two ridges and a valley, at right the contours as isophotes.

Let us test this on an X-ray image of fingers and calculate L_{vv} scale $\sigma = 3$.

```
im = Import["hands.gif"][[1, 1]]; Lvv = gauge2DN[im, 2, 0, 3];
DisplayTogetherArray[ListDensityPlot /@ {im, Lvv}, ImageSize → 450];
```

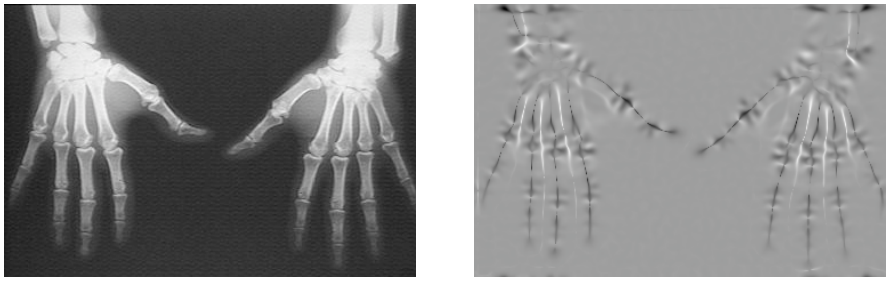

Figure 6.15 The invariant feature L_{vv} is a ridge detector. Here applied on an X-ray of two hands at $\sigma = 3$ pixels. Image resolution: 361 x 239 pixels.

▲ Task 6.4 Study the ridges L_{vv} of the fingers at different scales, and note the scale-dependent interpretation.

Noise has structure too. Here are the ridges of uniform white noise:

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

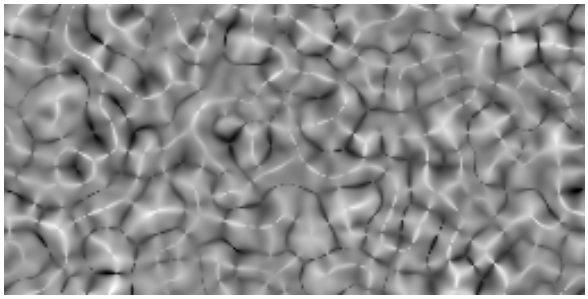

Figure 6.16 The invariant feature L_{vv} detects the ridges in white noise here, $\sigma = 4$ pixels, image resolution: 256 x 128 pixels.

▲ Task 6.5 Study in the same way the gradient of white noise at a range of scales. Do you see the similarity with a brain surface at larger scales?

We will encounter the second order gauge derivative L_{vv} in chapter 19 in the 'fundamental' equation of Alvarez et al. [Alvarez1992a, Alvarez1993], a *nonlinear* (geometry driven) diffusion equation: $\frac{\partial L}{\partial t} = L_{vv}$.

This equation is used to evolve the image in a way that locally adapts the amount of blurring to differential invariant structure in the image in order to do e.g. edge-preserving smoothing. We discuss this in detail in chapter 21.

Detection of ridges is an active topic in multi-scale feature detection [Koenderink1993a, Maintz1996a, Eberly1993, Eberly1994, Eberly1994a, Eberly1994b, Damon1999, Lindeberg1998b, Lopéz1999], as it focuses on the *dual* of boundaries.

6.6.2 Isophote and flowline curvature in gauge coordinates

The derivation of the formula for isophote curvature is particularly easy when we express the problem 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: $L(v, w) = \text{Constant}$, and $w = w(v)$. So, in *Mathematica* we implicitly differentiate the equality (==) to v:

```
L[v, w[v]] == Constant;
v =.; w =.; D[L[v, w[v]] == Constant, v]
```

$$w'[v]\, L^{(0,1)}[v, w[v]] + L^{(1,0)}[v, w[v]] == 0$$

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'':

```
κ = w''[v] /. Solve[D[L[v, w[v]] == Constant, {v, 2}] /. w'[v] -> 0, w''[v]]
```

$$\left\{-\frac{L^{(2,0)}[v, w[v]]}{L^{(0,1)}[v, w[v]]}\right\}$$

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

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

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

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

```
im = Import["mr256.gif"][[1, 1]];
κplot =
  ListDensityPlot[-gauge2DN[im, 2, 0, #] / gauge2DN[im, 0, 1, #], PlotRange -> {-5, 5}] &;
```

```
DisplayTogetherArray[
  {ListDensityPlot[im], κplot[1], κplot[2], κplot[3]}, ImageSize -> 470];
```

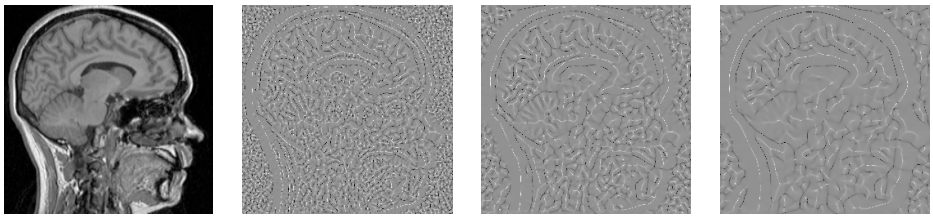

Figure 6.17 The isophote curvature κ is a rotationally and translationally invariant feature. It takes high values at extrema. Image resolution: 256^2 pixels.

The reason we see extreme low and high values is due to the singularities that occur at intensity extrema, where the gradient $L_w = 0$.

▲ Task 6.6 Why was not in a single pixel infinite isophote curvature encountered? There are many maxima and minima in the image.

Lopéz et al. [Lopéz2000b] defined a robust multi-scale version of a local curvature measure, which they termed level set extrinsic curvature, based on the divergence of the gradient field, integrated over a path (with a certain are: the scale) around the point of interest.
The *perception of curvature* is influenced by its context, as is clear from the Tolansky's curvature illusion (see figure 6.18).

```
Show[
  Graphics[{Thickness[.01], Circle[{0, 0}, 10, {0, π}], Circle[{0, -4},
     10, {π / 4, 3 π / 4}], Circle[{0, -8}, 10, {3 π / 8, 5 π / 8}]}],
  AspectRatio → Automatic, ImageSize -> 260];
```

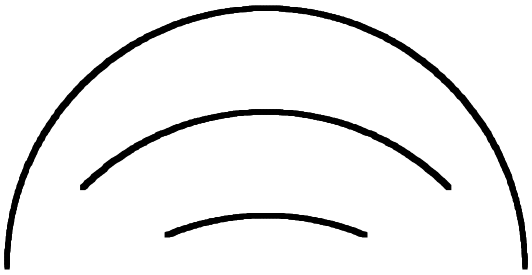

Figure 6.18 Tolansky's curvature illusion. The three circle segments have the same curvature 1/10.

We remember the *flowlines* as the integral curves of the gradient. In figure 6.6 they were depicted together with their *duals*, the isophotes. In that particular case, for such circular objects flowlines are straight lines with curvature zero. In figure 6.6 the isophote curvature at the top of the blob goes to infinity and is left out for that reason.

▲ Task 6.7 Prove, with the methodology sketched above, that the flowline curvature expressed in first order gauge coordinates is: $\mu = -\frac{L_{vw}}{L_w}$.

The third (and last) member of the set of second order derivatives in gauge coordinates is L_{ww}. This is the derivative of the gradient in the gradient direction. So when we want to find the maximum of the gradient, we can inspect zeros of L_{ww}.
Historically, much attention is paid to the zerocrossings of the Laplacian due to the groundbreaking work of Marr and Hildreth. As a rotational isotropic filter, and its close analogy to the retinal receptive fields, its zerocrossings were often interpreted as the maxima of a rotational invariant edge detector. The zerocrossings are however displaced on curved edges.

Note that with the compact expression for isophote curvature $\kappa = -\frac{L_{vv}}{L_w}$ we can establish a relation between the Laplacian and the second order derivative in the gradient direction we want to investigate for zerocrossings: L_{ww}. From the expression of the Laplacian in gauge coordinates $\Delta L = L_{ww} + L_{vv} = L_{ww} - \kappa L_w$ we see immediately that there is a deviation term κL_w which is directly proportional to the isophote curvature κ. Only on a straight edge with local isophote curvature zero the Laplacian is numerically equal to L_{ww}. Without gauge coordinates, this is much harder to prove. It took Clark two full pages in PAMI to show this [Clark1989]!

```
im = Import["thorax02.gif"][[1, 1]];
Block[{$DisplayFunction = Identity},
  p1 = ListDensityPlot[im];
  p2 = ListContourPlot[gauge2DN[im, 0, 2, 4], Contours -> {0}];
  p3 = ListContourPlot[gD[im, 2, 0, 4] + gD[im, 0, 2, 4], Contours -> {0}]];

DisplayTogetherArray[{Show[{p1, p2}], Show[{p1, p3}]}, ImageSize -> 380];
```

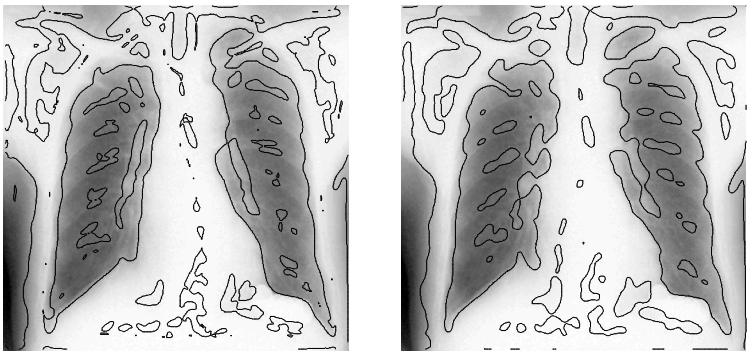

Figure 6.19 Contours of $L_{vv} = 0$ (left) and $\Delta L = 0$ (right) superimposed on the X-thorax image for $\sigma = 4$ pixels.

The term $\nu = -\frac{L_{ww}}{L_w}$ is not a curvature, but can be interpreted as a *density* of isophotes.

Notice that the isophote curvature $\kappa = -\frac{L_{vv}}{L_w}$ and flowline curvature $\mu = -\frac{L_{vw}}{L_w}$ have equal dimensionality for the intensity in both nominator and denominator. This leads to the desirable property that these curvatures do not change when we e.g. manipulate the contrast or brightness of an image. In general, these curvatures are said to be *invariant under monotonic intensity transformations*. In section 6.7 we elaborate on this special case of invariance.

6.6.3 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:

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

We omit the translation term $(e\ f)$ and study the affine transformation proper. The term $\frac{1}{a\,d - b\,c}$ 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:

```
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];
```

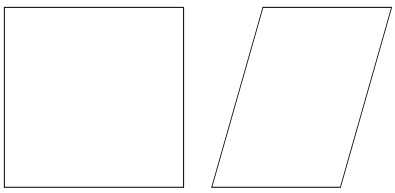

Figure 6.20 Affine transformation of a square, with transformation matrix $\begin{pmatrix} 5 & 2 \\ 0 & .5 \end{pmatrix}$ mapped on each point.

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

```
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:

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

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

Very interesting: when $n = 3$ and for an affine transformation with unity Jacobean ($a\,d - b\,c = 1$, 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 $\Theta = \frac{L_{v\,v}}{L_w}\,L_w{}^3 = L_{v\,v}\,L_w{}^2 = 2\,L_x\,L_{x\,y}\,L_y - L_{x\,x}\,L_y{}^2 - L_x{}^2\,L_{y\,y}$ 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 the 'Utrecht' image:

```
im = SubMatrix[Import["Utrecht256.gif"][[1, 1]], {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;
```

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

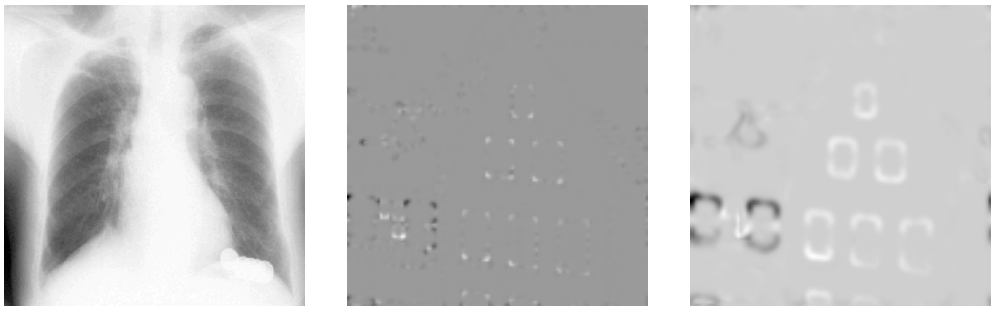

Figure 6.21 Corner detection with the $L_{vv} L_w^2$ operator. Left: original image, dimensions 128^2. Middle: corner detection at σ = 1 pixel; right: corner detection at σ = 3 pixels. Isophote curvature is signed, so note the positive (convex, light) and negative (concave, dark) corners.

▲ Task 6.8 Show why the compound spike response, where an rotationally invariant operator is applied on a spike image (discrete delta function), leads to a rotationally symmetric response. An example is given below:

```
spike = Table[0, {128}, {128}]; spike[[64, 64]] = 100;
gradient = gauge2DN[spike, 0, 1, 15];
cornerness = -gauge2DN[spike, 2, 0, 15] gauge2DN[spike, 0, 1, 15]^2;
DisplayTogetherArray[
  ListDensityPlot /@ {spike, gradient, cornerness}, ImageSize -> 400];
```

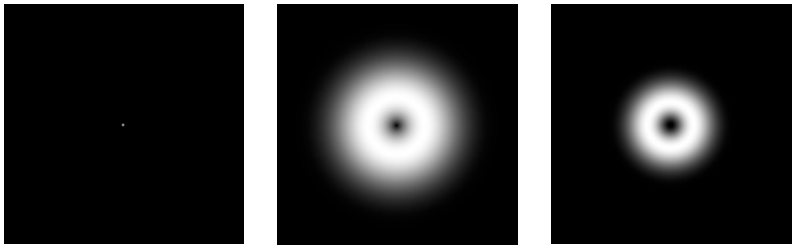

Figure 6.22 Convolution of a spike (Delta function) image with a kernel gives the kernel itself as result. Left: spike image, middle: response to the gradient kernel assembly, right: response to the cornerness kernel assembly. Scale σ = 15 pixels, resolution image 128^2.

6.7 A curvature illusion

A particular visual illusion shows the influence of the *multi-scale* perception of a local property, like curvature. In figure 6.23 the lines appear curved, though they are really straight.

```
star = Graphics[Table[
    Line[{{Cos[ϕ], Sin[ϕ]}, {-Cos[ϕ], -Sin[ϕ]}}], {ϕ, 0, π, π / 20}]];
lines = Graphics[{Thickness[.015], DarkViolet,
    Line[{{-1, .1}, {1, .1}}], Line[{{-1, -.1}, {1, -.1}}]}];
Show[{star, lines}, PlotRange → {{-.4, .4}, {-.2, .2}},
  AspectRatio → Automatic, ImageSize -> 300];
```

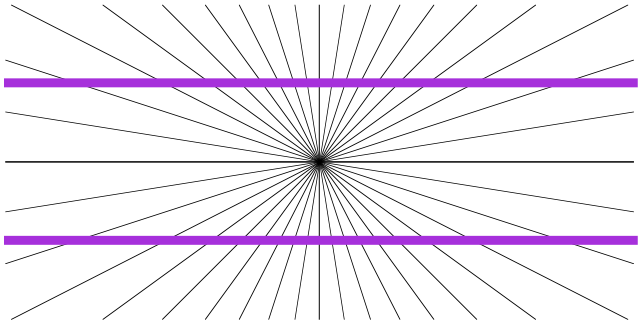

Figure 6.23 The straight lines appear curved due to the surrounding pattern.

When we calculate the isophote curvature $\kappa = \frac{-L_{vv}}{L_w}$ for this figure at a coarse scale, we see that the curvature is not constant along the horizontal lines, but changes when moving from the center. Figure 6.24 shows the curvature and the profile along the center of the horizontal line.

```
curvill = Show[{star, lines}, PlotRange → {{-.4, .4}, {-.2, .2}},
  AspectRatio → Automatic, ImageSize -> 432, DisplayFunction -> Identity];
Export["curvillusion-star.jpg", curvill];
im1 = Import["curvillusion-star.jpg"][[1, 1]] /. {a_, b_, c_} -> (a + b + c) / 3;
DeleteFile["curvillusion-star.jpg"];

DisplayTogetherArray[
  {ListDensityPlot[κ1 = -gauge2DN[im1, 2, 0, 20] / gauge2DN[im1, 0, 1, 20], PlotRange -> {-.1, .1},
    Epilog -> {Red, Line[{{110, 161}, {320, 161}}]}],
ListPlot[Take[κ1, {161, 161}, {110, 320}] // Flatten,
    AspectRatio -> .4, AxesLabel -> {"", "κ1"}]}, ImageSize -> 450];
```

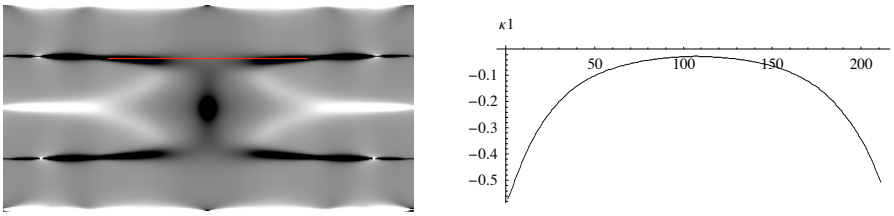

Figure 6.24 Left: Isophote curvature κ at a scale of $\sigma = 20$ pixels for the pattern in figure 6.23, dimensions image 216 x 432 pixels. Right: profile of curvature along the central portion of the top horizontal line (to avoid boundary effects only the central portion is shown, indicated by the red line in the left figure).

6.8 Second order structure

The second order structure of the intensity landscape is rich. To describe and to represent it, we will develop a precise mathematical formulation in order to do a proper analysis.

Let us first develop some intuitive notions by visual inspection. Figure 6.25 shows a blurred version of an X-thorax image is depicted as a height plot. We see hills and dales, saddle points, ridges, maxima and minima. Clearly curvature plays an important role.

The second order structure of the intensity landscape $L(x, y; \sigma)$ in a point $L(x_0, y_0; \sigma)$ is described by the second order term in the local Taylor expansion around the point (x_0, y_0). Without any loss of generalization we take $(x_0, y_0) = (0, 0)$:

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

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

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

```
im = Import["thorax02.gif"][[1, 1]];
DisplayTogetherArray[ListDensityPlot[im],
  ListPlot3D[-gD[im, 0, 0, 2], Mesh → False], ImageSize → 320];
```

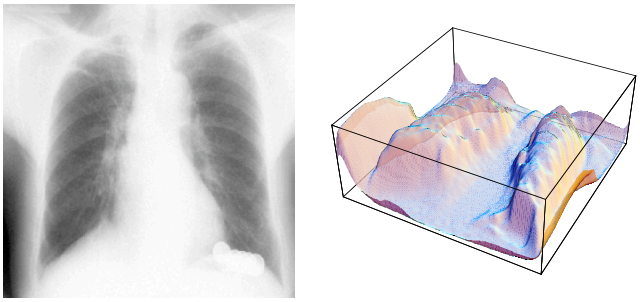

Figure 6.25 Left: An X-thorax image (resolution 256^2) and its 'intensity landscape' at $\sigma = 2$ pixels (right).

We investigate the role of the coefficients in this second order polynomial. In the graph below we vary all three coefficients. In the three groups of 9 plots the value of the mixed coefficient $L_{x\,y}$ has been varied (value -1, 0 and 1). In each group the 'pure' order terms $L_{x\,x}$ and $L_{y\,y}$ are varied (values -1, 0 and +1). In the middle group we see concave, convex, cylindrical and saddle shapes.

```
Show[GraphicsArray[
  Table[GraphicsArray[Table[Plot3D[Lxx/2 x^2 + Lxy x y + Lyy/2 y^2, {x, -3, 3},
    {y, -3, 3}, PlotRange → {-18, 18}, AspectRatio → 1,
    DisplayFunction → Identity, Boxed → True, Mesh → False],
   {Lxx, -1, 1}, {Lyy, -1, 1}], Frame → True], {Lxy, -1, 1}]], ImageSize → 480];
```

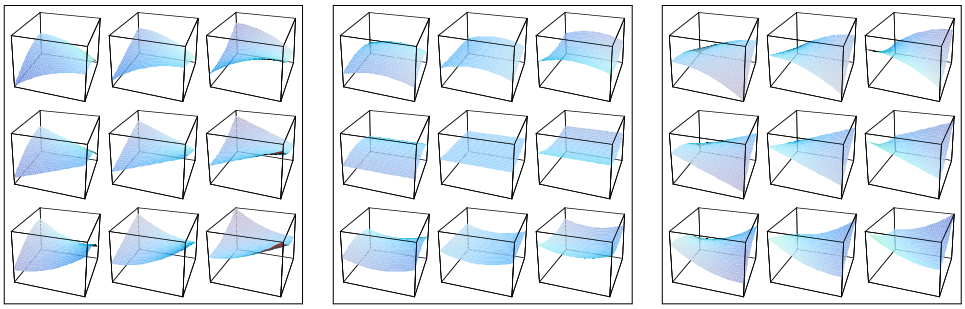

Figure 6.26 Plots of $\frac{L_{xx}}{2} x^2 + L_{xy}\, x\, y + \frac{L_{yy}}{2} y^2$. Left: $L_{xy} = -1$. Middle: $L_{xy} = 0$. Right: $L_{xy} = 1$. In each frame: upper row: $L_{xx} = 1$, middle row: $L_{xx} = 0$, lower row: $L_{xx} = -1$, left column: $L_{yy} = -1$, middle column: $L_{yy} = 0$, right row: $L_{yy} = 1$.

When three variables are at steak, and a visual impression may give valuable insight, one can exploit the trichromacy of our vision. We employ the *invariant* second order derivatives, L_{vv}, L_{vw} and L_{ww}. This shows the triple $\{L_{vv}, L_{vw}, L_{ww}\}$ as RGBColor$[L_{vv}, L_{vw}, L_{ww}\}$ color directive settings in each pixel. The color coefficients for this function need to be scaled between 0 and 255.

```
im = Import["thorax02.gif"]; σ = 5; impix = im[[1, 1]]; imcolor = im;
min = Min[color = Transpose[{gauge2DN[impix, 2, 0, σ],
      gauge2DN[impix, 1, 1, σ], gauge2DN[impix, 0, 2, σ]}, {3, 1, 2}]];
max = Max[color - min]; imcolor[[1, 1]] = N[(color - min)/max 255];
imcolor[[1, 4]] = ColorFunction → RGBColor;
DisplayTogetherArray[Show /@ {im, imcolor}, ImageSize → 400];
```

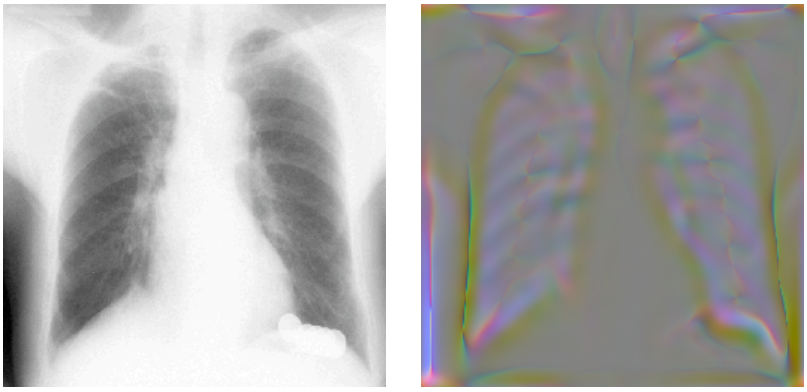

Figure 6.27 Left: An X-thorax image (resolution 256^2) and a mapping of the triple of invariant second order derivatives $\{L_{vv}, L_{vw}, L_{ww}\}$ on the RGB coordinates in each pixel.

6.8.1 The Hessian matrix and principal curvatures

At any point on the surface we can step into an infinite number of directions away from the point, and in each direction we can define a curvature. So in each point an infinite number of curvatures can be defined. It runs out that the curvatures in opposite directions are always the same. Secondly, when we smoothly change direction, there are two (opposite) directions where the curvature is maximal, and there are two (opposite) directions where the curvature is minimal. These directions are perpendicular to each other, and the extremal curvatures are called the *principal curvatures*.

The Hessian matrix is the gradient of the gradient vectorfield. The coefficients form the *second order structure matrix*, or the *Hessian matrix*, also known as the *shape operator* [Gray1993]. The Hessian matrix is a square, symmetric matrix:

```
hessian2D = ( ∂x,x L[x, y]   ∂x,y L[x, y] )
            ( ∂x,y L[x, y]   ∂y,y L[x, y] );
```

The Hessian matrix is square and symmetric, so we can bring it in diagonal form by calculating the Eigenvalues of the matrix and put these on the diagonal elements:

```
DiagonalMatrix[Eigenvalues[hessian2D]] // shortnotation
```

$$\left\{\left\{\frac{1}{2}\left(L_{xx}+L_{yy}-\sqrt{L_{xx}^2+4\,L_{xy}^2-2\,L_{xx}\,L_{yy}+L_{yy}^2}\right),\ 0\right\},\right.$$
$$\left.\left\{0,\ \frac{1}{2}\left(L_{xx}+L_{yy}+\sqrt{L_{xx}^2+4\,L_{xy}^2-2\,L_{xx}\,L_{yy}+L_{yy}^2}\right)\right\}\right\}$$

These special values are the *principal curvatures* of that point of the surface. In the diagonal form the Hessian matrix is rotated in such a way, that the curvatures are maximal and minimal. The principal curvature *directions* are given by the Eigenvectors of the Hessian matrix, found by solving the *characteristic equation* $|H-\kappa I|=0$ for κ, where $|\ldots|$ denotes the determinant, and I is the identity matrix (all diagonal elements are 1, rest zeros).

```
κ =.; Solve[Det[hessian2D - κ IdentityMatrix[2]] == 0, κ] // shortnotation
```

$$\left\{\left\{\kappa\to\frac{1}{2}\left(L_{xx}+L_{yy}-\sqrt{L_{xx}^2+4\,L_{xy}^2-2\,L_{xx}\,L_{yy}+L_{yy}^2}\right)\right\},\right.$$
$$\left.\left\{\kappa\to\frac{1}{2}\left(L_{xx}+L_{yy}+\sqrt{L_{xx}^2+4\,L_{xy}^2-2\,L_{xx}\,L_{yy}+L_{yy}^2}\right)\right\}\right\}$$

The command to calculate Eigenvalues is built into *Mathematica*:

```
{κ1, κ2} = Eigenvalues[hessian2D] // FullSimplify;
{κ1, κ2} // shortnotation
```

$$\left\{\frac{1}{2}\left(L_{xx}-\sqrt{4\,L_{xy}^2+(L_{xx}-L_{yy})^2}+L_{yy}\right),\ \frac{1}{2}\left(L_{xx}+\sqrt{4\,L_{xy}^2+(L_{xx}-L_{yy})^2}+L_{yy}\right)\right\}$$

The two principal curvatures are equal when $4\,L_{xy}{}^2+(L_{yy}-L_{xx})^2$ is zero. This happens in so-called *umbilical points*. In umbilical points the principal directions are undefined. The surface is locally *spherical*. The term $4\,L_{xy}{}^2+(L_{yy}-L_{xx})^2$ can be interpreted as 'deviation from sphericalness'.

6.8.2 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 to call the angle, of where the shape vector points to, 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:

```
Simplify[(κ1 + κ2)/(κ1 - κ2)] // shortnotation
```

$$\frac{-L_{xx} - L_{yy}}{\sqrt{4\,L_{xy}^2 + (L_{xx} - L_{yy})^2}}$$

so we get for the shape index:

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

The shape index runs from -1 (*cup*) via the shapes *trough*, *rut*, and *saddle rut* to zero, the saddle (here the shape index is undefined), and the goes via *saddle ridge*, *ridge*, and *dome* to the value of +1, the *cap*.

The length of the vector defines how curved a shape is, which gives Koenderink's definition of *curvedness*:

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

```
1/2 Sqrt[κ1^2 + κ2^2] // Simplify // shortnotation
```

$$\frac{1}{2}\sqrt{L_{xx}^2 + 2\,L_{xy}^2 + L_{yy}^2}$$

```
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}]]];
```

```
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];
```

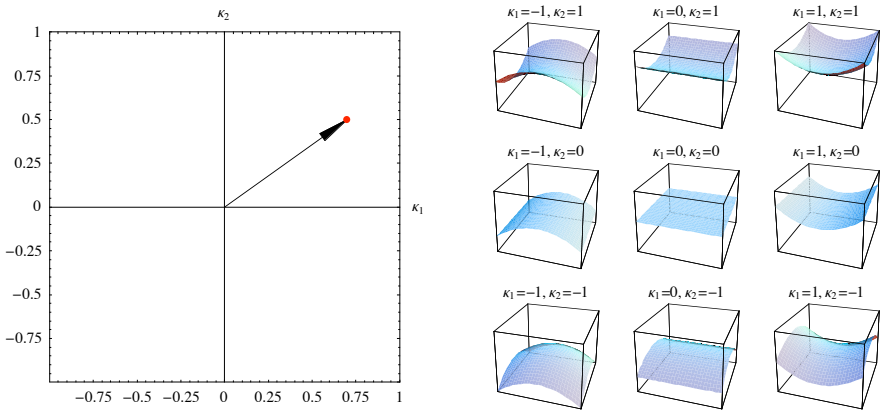

Figure 6.28 Left: Coordinate space of the shape index. Horizontal axis: maximal principal curvature κ_1, vertical axis: minimal principal curvature κ_2. The angle of the position vector determines the shape, the length the curvedness. Right: same as middle set of figure 6.22.

Here is the shape index calculated and plotted for every pixel on our familiar MR image at a scale of σ=3 pixels:

```
im = Import["mr128.gif"][[1, 1]];
shapeindex[im_, σ_] := 2/π ArcTan[(-gD[im, 2, 0, σ] - gD[im, 0, 2, σ]) /
    (√(gD[im, 2, 0, σ]^2 + 4 gD[im, 1, 1, σ]^2 -
        2 gD[im, 2, 0, σ] gD[im, 0, 2, σ] + gD[im, 0, 2, σ]^2))];
DisplayTogetherArray[ListDensityPlot[shapeindex[im, #]] & /@ Range[5],
  ImageSize → 400];
```

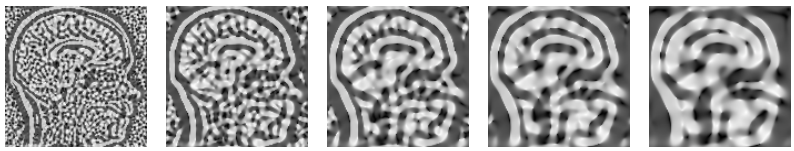

Figure 6.29 Shape index of the sagittal MR image at σ = 1, 2, 3, 4 and 5 pixels.

```
curvedness[im_, σ_] :=
  √(gD[im, 2, 0, σ]^2 + 2 gD[im, 1, 1, σ]^2 + gD[im, 0, 2, σ]^2);
DisplayTogetherArray[ListDensityPlot[curvedness[im, #]] & /@ Range[4],
  ImageSize → 400];
```

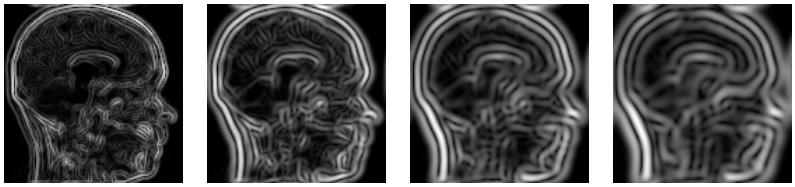

Figure 6.30 Curvedness of the sagittal MR image at σ = 1, 2, 3 and 4 pixels.

6.8.3 Principal directions

The principal curvature directions are given by the Eigenvectors of the Hessian matrix:

```
{vκ1, vκ2} = Eigenvectors[hessian2D]; {vκ1, vκ2} // shortnotation
```

$$\left\{\left\{-\frac{-L_{xx}+L_{yy}+\sqrt{L_{xx}^2+4\,L_{xy}^2-2\,L_{xx}\,L_{yy}+L_{yy}^2}}{2\,L_{xy}},\ 1\right\},\right.$$
$$\left.\left\{\frac{L_{xx}-L_{yy}+\sqrt{L_{xx}^2+4\,L_{xy}^2-2\,L_{xx}\,L_{yy}+L_{yy}^2}}{2\,L_{xy}},\ 1\right\}\right\}$$

The Eigenvectors are perpendicular to each other, there inner product is zero:

```
vκ1.vκ2 // Simplify
```

```
0
```

The local principal direction vectors form locally a *frame*. We inspect how the orientations of such frames are distributed in an image. We orient the frame in such a way that the largest Eigenvalue (maximal principal curvature) is one direction, the minimal principal curvature direction is $\pi/2$ rotated clockwise.

```
plotprincipalcurvatureframes[im_, σ_] :=
 Module[{hessian, frame, frames},
  hessian = ( gD[im, 2, 0, σ]  gD[im, 1, 1, σ]
              gD[im, 1, 1, σ]  gD[im, 0, 2, σ] );
frame = {Green, Arrow[#2 - .5, #2 - .5 + First[#1]],
     Red, Arrow[#2 - .5, #2 - .5 + Last[#1]]} &;
frames = MapIndexed[frame, .5 Map[Eigenvectors,
      Transpose[hessian, {4, 3, 2, 1}], {2}], {2}];
plot = ListDensityPlot[gD[im, 0, 0, σ], Epilog → frames]]
im = Import["mr32.gif"][[1, 1]];
```

```
plotprincipalcurvatureframes[im, 1];
```

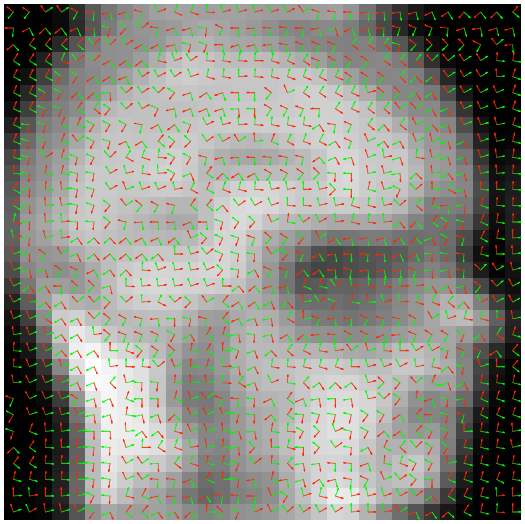

Figure 6.31 Frames of the normalized principal curvature directions at a scale of 1 pixel. Image resolution 32^2 pixels. Green: maximal principal curvature direction; red: minimal principal curvature direction.

The principal curvatures have been employed by Niessen, ter Haar Romeny and Lopéz in studies to the 2D and 3D structure of trabecular bone [TerHaarRomeny1996f, Niessen1997b, Lopéz200a]. The local structure was defined as *flat* when the two principal curvatures of the iso-intensity surface in 3D were are both small, as *rod-like* if one of the curvatures was small and the other high, giving a local cylindrical shape, and *sphere-like* if two principal curvatures were both high. See also Task 19.8.

6.8.4 Gaussian and mean curvature

The Gaussian curvature $\mathcal{K}$ is defined as the product of the two principal curvatures: $\mathcal{K} = \kappa_1 \kappa_2$.

```
𝒦 = κ1 κ2 // Simplify // shortnotation
```

$$-L_{xy}^2 + L_{xx}\, L_{yy}$$

The Gaussian curvature is equal to the determinant of the Hessian matrix:

```
Det[hessian2D] // shortnotation
```

$$-L_{xy}^2 + L_{xx}\, L_{yy}$$

The sign of the Gaussian curvature determines if we are in a concave / convex area (positive Gaussian curvature) or in a saddle-like area (negative Gaussian curvature). This shows saddle-like areas as dark patches:

```
im = Import["mr256.gif"][[1, 1]];
σ = 5; 𝒦 = -gD[im, 1, 1, σ]^2 + gD[im, 2, 0, σ] gD[im, 0, 2, σ];
DisplayTogetherArray[Append[ListDensityPlot /@ {𝒦, Sign[𝒦]},
   ListContourPlot[𝒦, Contours → {0}]], ImageSize → 390];
```

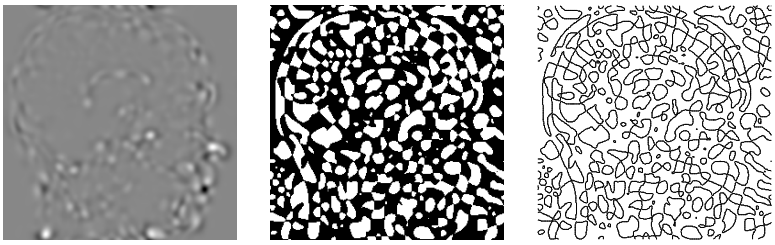

Figure 6.32 Left: Gaussian curvature $\mathcal{K}$ for a 256^2 sagittal MR image at a scale of 5 pixels. Middle: sign of $\mathcal{K}$. Right: zerocrossings of $\mathcal{K}$.

The locations where the Gaussian curvature is zero, are characterized by the fact that at least one of the principal curvatures is zero. The collection of locations where the Gaussian curvature is zero is known as the *parabolic lines*. It was shown by Koenderink that these lines play an important role in reflection and shape-from-shading.

The mean curvature is defined as the arithmetic mean of the principal curvatures: $\mathcal{H} = \frac{\kappa_1 + \kappa_2}{2}$.

The mean curvature is related to the trace of the Hessian matrix:

```
ℋ = (κ1 + κ2)/2 // Simplify // shortnotation
```

$$\frac{1}{2}\left(L_{xx} + L_{yy}\right)$$

```
Tr[hessian2D] // shortnotation
```

$$L_{xx} + L_{yy}$$

The relation between the mean curvature $\mathcal{H}$ and the Gaussian curvature $\mathcal{K}$ is given by $\kappa^2 - 2\mathcal{H}\kappa + \mathcal{K} = 0$, which has solutions:

```
ℋ =.; 𝒦 =.; Solve[κ^2 - 2 ℋ κ + 𝒦 == 0, κ]
```

$$\left\{\left\{\kappa \to \mathcal{H} - \sqrt{\mathcal{H}^2 - \mathcal{K}}\right\}, \left\{\kappa \to \mathcal{H} + \sqrt{\mathcal{H}^2 - \mathcal{K}}\right\}\right\}$$

The mean curvature $\mathcal{H}$ and the Gaussian curvature $\mathcal{K}$ are well defined in umbilical points.

The directional derivative of the principal curvature in the direction of the principal direction is called the *extremality* [Monga1995].

Because there are two principal curvatures, there are two extremalities, $\overrightarrow{\nabla\kappa_1}.\vec{\nabla}\kappa_1$ and $\overrightarrow{\nabla\kappa_2}.\vec{\nabla}\kappa_2$:

```
<< Calculus`VectorAnalysis`;
```

```
e1 = vκ1.Take[Grad[{κ1, 0, 0}, Cartesian[x, y, z]], 2] // FullSimplify;
e1 // shortnotation
```

$$\left\{-\frac{1}{4\,L_{xy}}\left(\left(L_{xxx}+L_{xyy}+\frac{-4\,L_{xxy}\,L_{xy}-(L_{xxx}-L_{xyy})\,(L_{xx}-L_{yy})}{\sqrt{4\,L_{xy}^2+(L_{xx}-L_{yy})^2}}\right)\left(-L_{xx}+\sqrt{4\,L_{xy}^2+(L_{xx}-L_{yy})^2}+L_{yy}\right)\right),\right.$$

$$\left.-\frac{1}{4\,L_{xy}}\left(\left(-L_{xx}+\sqrt{4\,L_{xy}^2+(L_{xx}-L_{yy})^2}+L_{yy}\right)\left(L_{xxy}+\frac{-4\,L_{xy}\,L_{xyy}-(L_{xx}-L_{yy})\,(L_{xxy}-L_{yyy})}{\sqrt{4\,L_{xy}^2+(L_{xx}-L_{yy})^2}}+L_{yyy}\right)\right),\,0\right\}$$

```
e2 = vκ2.Take[Grad[{κ2, 0, 0}, Cartesian[x, y, z]], 2] // FullSimplify;
e2 // shortnotation
```

$$\left\{-\frac{1}{4\,L_{xy}}\left(\left(L_{xxx}+L_{xyy}+\frac{4\,L_{xxy}\,L_{xy}+(L_{xxx}-L_{xyy})\,(L_{xx}-L_{yy})}{\sqrt{4\,L_{xy}^2+(L_{xx}-L_{yy})^2}}\right)\left(-L_{xx}-\sqrt{4\,L_{xy}^2+(L_{xx}-L_{yy})^2}+L_{yy}\right)\right),\right.$$

$$\left.-\frac{1}{4\,L_{xy}}\left(\left(-L_{xx}-\sqrt{4\,L_{xy}^2+(L_{xx}-L_{yy})^2}+L_{yy}\right)\left(L_{xxy}+\frac{4\,L_{xy}\,L_{xyy}+(L_{xx}-L_{yy})\,(L_{xxy}-L_{yyy})}{\sqrt{4\,L_{xy}^2+(L_{xx}-L_{yy})^2}}+L_{yyy}\right)\right),\,0\right\}$$

The lines defined by the zerocrossings of each of these two extremalities are called the *extremal lines* [Thirion1995a, Thirion1996]. There are 4 types of these lines:
- lines of maximum largest principal curvature (these are called *crest lines*);
- lines of minimum largest principal curvature;
- lines of maximum smallest principal curvature;
- lines of minimum smallest principal curvature.

The product of the two extremalities is called the *Gaussian extremality* $\mathcal{G} = e_1.e_2$, a true local invariant [Thirion1996]. The boundaries of the regions where the Gaussian extremality changes sign, are the extremal lines.

```
e1.e2 // Simplify // shortnotation
```

$$-(L_{xy}^2\,(L_{xxx}^2+2\,L_{xxx}\,L_{xyy}-3\,(L_{xxy}^2+L_{xyy}^2))+L_{xxx}\,L_{xyy}\,(L_{xx}-L_{yy})^2+2\,L_{xxx}\,L_{xxy}\,L_{xy}\,(-L_{xx}+L_{yy})+(L_{xx}^2\,L_{xxy}-2\,L_{xy}\,L_{xyy}\,L_{yy}+2\,L_{xx}\,(L_{xy}\,L_{xyy}-L_{xxy}\,L_{yy})+L_{xxy}\,(2\,L_{xy}^2+L_{yy}^2))\,L_{yyy}+L_{xy}^2\,L_{yyy}^2)\,/\,(L_{xx}^2+4\,L_{xy}^2-2\,L_{xx}\,L_{yy}+L_{yy}^2)$$

```
DisplayTogetherArray[
  ListDensityPlot[Sign[e1.e2 /. Derivative[nx_, ny_][L][x_, y_] →
        gD[im0, nx, ny, #] /. im0 → im]] & /@ {2, 6, 10}, ImageSize → 400];
```

Figure 6.33 Left: Gaussian extremality $\mathcal{G} = e_1 e_2$ for a 256^2 sagittal MR image at a scale of 2 pixels (left), 6 pixels (middle) and 10 pixels (right).

The mesh that these lines form on an iso-intensity surface in 3D is called the *extremal mesh*. It has been applied for 3D image registration, by extracting the lines with a dedicated 'marching lines' algorithm [Thirion1996].

```
Show[Import["extremal mesh - Thirion.jpg"], ImageSize → 250];
```

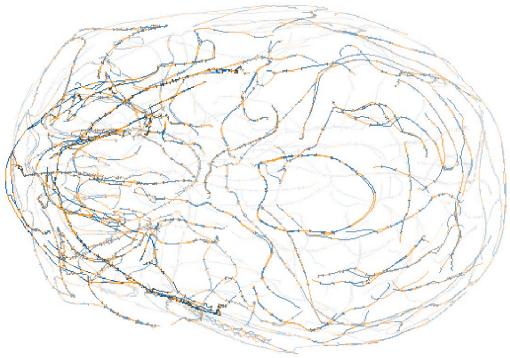

Figure 6.34 Extremal mesh on a 3D skull from a 3D CT dataset. The extremal lines are found with the marching lines algorithm. From [Thirion1993].

6.8.5 Minimal surfaces and zero Gaussian curvature surfaces

Surfaces that have everywhere mean curvature zero, are called *minimal surfaces*. There are many beautiful examples of such surfaces (see e.g. the Scientific Grahics Project, http://www.msri.org/publications/sgp/SGP/indexc.html. Soap bubbles are famous and much studied examples of minimal surfaces.

From the wonderful interactive book by Alfred Gray [Gray1993] (written in *Mathematica*) we plot two members of the family of zero Gaussian curvature manifolds that can be constructed by a moving straight line through 3D space:

```
heltocat[t_][u_, v_] := Cos[t] {Sinh[v] Sin[u], -Sinh[v] Cos[u], u} +
   Sin[t] {Cosh[v] Cos[u], Cosh[v] Sin[u], v};
moebiusstrip[u_, v_] := {Cos[u] + v Cos[u / 2] Cos[u],
   Sin[u] + v Cos[u / 2] Sin[u], v Sin[u / 2]};
DisplayTogetherArray[{ParametricPlot3D[
    Evaluate[heltocat[0][u, v]], {u, -π, π}, {v, -π, π},
    PlotPoints -> 30, Axes -> None, BoxRatios -> {1, 1, 1},
    PlotRange -> {{-13, 13}, {-13, 13}, {-π, π}}],
ParametricPlot3D[moebiusstrip[u, v] // Evaluate,
    {u, 0, 2 Pi}, {v, -.3, .3}, PlotPoints -> {30, 4},
Axes -> None]}, ImageSize → 390];
```

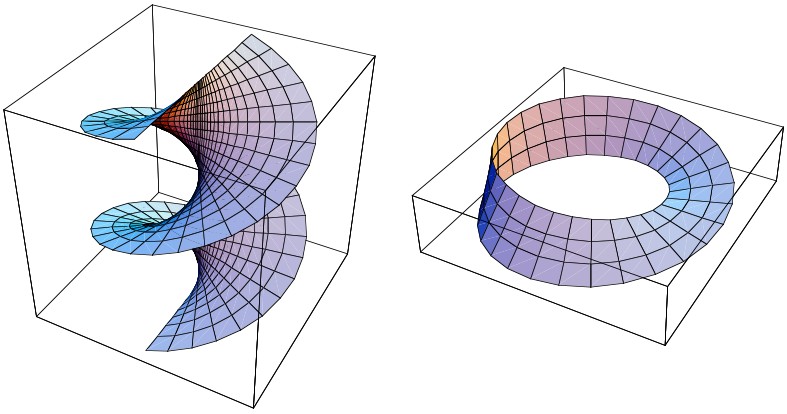

Figure 6.35 Surfaces with zero Gaussian curvature. Left the helicoid, a member of the heltocat family. Right the Moebius strip. Both surfaces can be constructed by a moving straight line. From [Gray1993].

▲ Task 6.9 Which surfaces have constant mean curvature? And which surfaces have constant Gaussian curvature?

▲ Task 6.10 If I walk with my principal coordinate frame over an egg, something goes wrong when I walk through an umbilical point. What?

6.9 Third order image structure: T-junction detection

An example of third order *geometric reasoning* in images is the detection of T-junctions [TerHaarRomeny1991a, TerHaarRomeny1993b]. T-junctions in the intensity landscape of natural images occur typically at occlusion points. Occlusion points are those points where a contour ends or emerges because there is another object in front of the contour. See for an artistic example the famous painting 'the blank cheque' by Magritte.

```
Show[Import["blank cheque.jpg"], ImageSize -> 210];
```

Figure 6.36 The painting 'the blank cheque' by the famous Belgian surrealist painter René Magritte (1898 - 1967). Source: Paleta (www.paletaworld.org).

In this section we develop a third order detector for "T-junction-likeniness". In the figure below the circles indicate a few particular T-junctions:

```
blocks = Import["blocks.gif"][[1, 1]];
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];
```

Figure 6.37 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.

When we zoom in on the T-junction of an observed image and inspect locally the isophote structure at a T-junction, we see that at a T-junction the derivative of the isophote curvature κ

in the direction perpendicular to the isophotes is high. In the figure below the isophote landscape of a blurred T-junction illustrates the direction of maximum change of κ:

```
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];
```

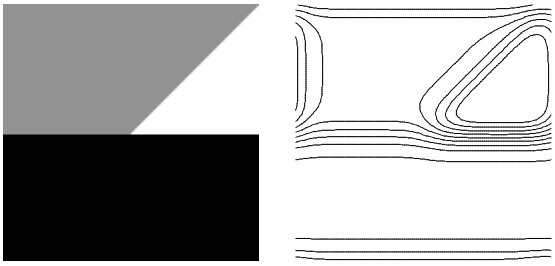

Figure 6.38 The isophote structure (right) of a simple idealized and observed (blurred) T-junction (left) shows that isophotes strongly bend at T-junctions when we walk through the intensity landscape.

When we study the curvature of the isophotes in the middle of the image, at the location of the T-junction, we see the isophote 'sweep' from highly curved to almost straight for decreasing intensity. So the geometric reasoning is the "the isophote curvature changes a lot when we traverse the image in the w direction". It seems to make sense to study $\frac{\partial \kappa}{\partial w}$:

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

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

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

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:

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

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

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 $\tau = \frac{\partial \kappa}{\partial w} L_w{}^6$:

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

```
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):

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

Figure 6.39 Detection of T-junctions in the image of the blocks with the detector $\tau = \frac{\partial \kappa}{\partial w} L_w{}^6$. The same circles have been drawn as in figure 6.32.

Compare the detected points with the circles in the input image. Note that in medical *tomographic* images (CT, MR, PET, SPECT, US) there is no occlusion present. One can however use third order properties in any *geometric reasoning* scheme, as the 'change of a second order property'.

▲ Task 6.11 Investigate if the expression for the T-junction $\tau = \frac{\partial \kappa}{\partial w} L_w{}^6$ is affine invariant.

▲ Task 6.12 Another definition for a T-junction detector might be the magnitude of the gradient of the curvature: $\tau = \sqrt{(\frac{\partial \kappa}{\partial w})^2 + (\frac{\partial \kappa}{\partial w})^2}\, L_w{}^6$, or the derivative of the curvature in the *v* direction: $\frac{\partial \kappa}{\partial v} L_w{}^6$. Study and explain the differences.

6.10 Fourth order image structure: junction detection

As a final fourth order example, we give an example for a detection problem in images at high order of differentiation from algebraic theory. Even at orders of differentiation as high as 4, invariant features can be constructed and calculated for discrete images through the biologically inspired scaled derivative operators. Our example is to find in a checkerboard the crossings where 4 edges meet. We take an algebraic approach, which is taken from Salden et al. [Salden1999a].

When we study the fourth order local image structure, we consider the fourth order polynomial terms from the local Taylor expansion:

```
pol4 = Lxxxx x^4 + 4 Lxxxy x^3 y + 6 Lxxyy x^2 y^2 + 4 Lxyyy x y^3 + Lyyyy y^4;
```

The main theorem of algebra states that a polynomial is fully described by its roots: e.g. $ax^2 + bx + c = (x - x_1)(x - x_2)$. It was shown more than a century ago by Hilbert [Hilbert1890] that the 'coincidencesness' of the roots, or how well all roots coincide, is a particular invariant condition. From algebraic theory it is known that this 'coincidenceness' is given by the *discriminant*, defined below (see also [Salden1999a]):

```
Discriminant[p_, x_] :=
 With[{m = Exponent[p, x]}, Cancel[(-1)^(1/2 m (m-1)) Resultant[p, ∂_x p, x] / Coefficient[p, x, m]]]
```

The resultant of two polynomials a and b, both with leading coefficient one, is the product of all the differences $a_i - b_j$ between roots of the polynomials. The resultant is always a number or a polynomial. The discriminant of a polynomial is the product of the squares of all the differences of the roots taken in pairs. We can express our function in two variables $\{x, y\}$ as a function in a single variable $\frac{x}{y}$ by the substitution $y \to 1$. Some examples:

```
Discriminant[Lxx x^2 + 2 Lxy x y + Lyy y^2, x] /. {y → 1}
```

```
-4 (-Lxy^2 + Lxx Lyy)
```

The discriminant of second order image structure is just the determinant of the Hessian matrix, i.e. the Gaussian curvature. Here is our fourth order discriminant:

```
Discriminant[pol4, x] /. {y → 1} // Simplify
```

```
256 (-27 Lxxxy^4 Lyyyy^2 + Lxxxy^3 (-64 Lxyyy^3 + 108 Lxxyy Lxyyy Lyyyy) -
    12 Lxxxx Lxxxy Lxyyy (-9 Lxxyy Lxyyy^2 + 15 Lxxyy^2 Lyyyy + Lxxxx Lyyyy^2) -
    6 Lxxxy^2 (-6 Lxxyy^2 Lxyyy^2 + 9 Lxxyy^3 Lyyyy +
       Lxxxx Lxyyy^2 Lyyyy - 9 Lxxxx Lxxyy Lyyyy^2) +
    Lxxxx (-54 Lxxyy^3 Lxyyy^2 + 81 Lxxyy^4 Lyyyy + 54 Lxxxx Lxxyy Lxyyy^2 Lyyyy -
       18 Lxxxx Lxxyy^2 Lyyyy^2 + Lxxxx (-27 Lxyyy^4 + Lxxxx Lyyyy^3)))
```

It looks like an impossibly complicated polynomial in fourth order derivative images, and it is. Through the use of Gaussian derivative kernels each separate term can easily be

calculated. We replace (with the operator **/.**) all the partial derivatives into scaled Gaussian derivatives:

```
discr4[im_, σ_] := Discriminant[pol4, x] /.
  {y → 1, Lxxxx → gD[im, 4, 0, σ], Lxxxy → gD[im, 3, 1, σ],
   Lxxyy → gD[im, 2, 2, σ], Lxyyy → gD[im, 1, 3, σ], Lyyyy → gD[im, 0, 4, σ]}
```

Let us apply this high order function on an image of a checkerboard, and we add noise with twice the maximum image intensity to show its robustness, despite the high order derivatives (see figure 6.40).

Note that we have a highly symmetric situation here: the four edges that come together at the checkerboard vertex cut the angle in four. The symmetry can be seen in the complex expression for discr4: only pure partial derivatives of fourth order occur. For a less symmetric situation we need a detector which incorporates in its expression also the lower order partial derivatives. For details see [Salden1999a].

```
t1 = Table[If[(x > 50 && y > 50) || (x ≤ 50 && y ≤ 50), 0, 100] + 200 * Random[],
    {x, 100}, {y, 100}];
t2 = Table[If[(x + y - 100 > 0 && y - x < 0) || (x + y - 100 ≤ 0 && y - x ≥ 0),
      0, 100] + 200 * Random[], {x, 100}, {y, 100}];

noisycheck = Transpose[Join[t1, t2]];
DisplayTogetherArray[ListDensityPlot /@
   {noisycheck, discr4[noisycheck, 5]}, ImageSize -> 400];
```

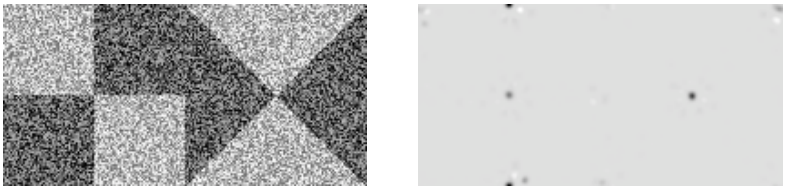

Figure 6.40 Left: A noisy checkerboard detail at two orientations. Right: the output of the 4th order discriminant. The detection clearly is rotation invariant, robust to noise, and there is no detection at corners (e.g. center of the image).

6.11 Scale invariance and natural coordinates

The intensity of images and invariant features at larger scale decreases fast. This is due to the non-scale-invariant use of the differential operators. For, if we consider the transformation $\frac{x}{\sigma} \rightarrow \hat{x}$, then $\hat{x}$ is dimensionless. At every scale now distances are measured with a distance yardstick with is scaled relative to the scale itself. This is the scale-invariance.

The dimensionless coordinate is termed the *natural* coordinate. This implies that the derivative operator in natural coordinates has a scaling factor: $\frac{\partial^n}{\partial \hat{x}^n} \rightarrow \sigma^n \frac{\partial^n}{\partial x^n}$.

Here we generate a scale-space of the intensity gradient. To study the absolute intensities, we plot every image with the same intensity plotrange of {0,40}:

```
im = Import["mr128.gif"][[1, 1]]; Block[{$DisplayFunction = Identity},
 p1 = Table[grad = √(gD[im, 1, 0, σ]^2 + gD[im, 0, 1, σ]^2);
   ListDensityPlot[#, PlotRange -> {0, 40}] & /@ {grad, σ grad}
   , {σ, 1, 5}]]; Show[GraphicsArray[Transpose[p1]], ImageSize → 450];
```

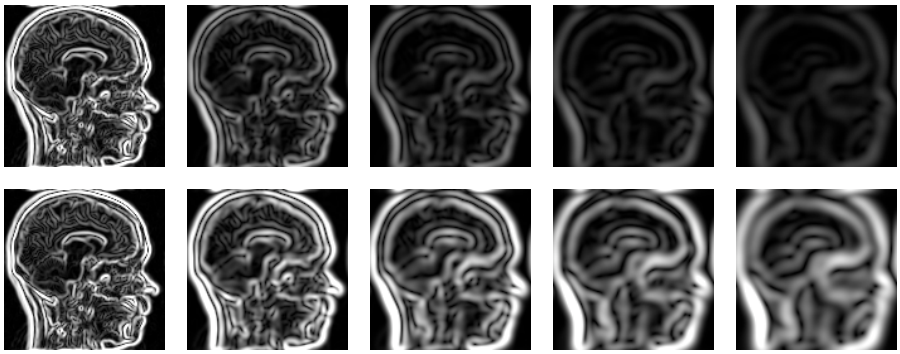

Figure 6.41 The gradient of a 128^2 image plotted at 5 scales, for σ = 1, 2, 3, 4 and 5 pixels respectively. All images (in both rows) are plotted at a fixed intensity range {0,40}. Top row shows the regular gradient, clearly showing the decrease in intensity for larger blurring. Bottom row: the gradient in natural coordinates (multiplied by σ). The intensity dynamic range is now kept more or less constant.

Clearly the gradient magnitude expressed in the natural coordinates keeps its average output range. For a Laplacian scale-space stack in natural coordinates we need to multiply the Laplacian with σ^2: $\frac{\partial^2}{\partial \hat{x}^2} + \frac{\partial^2}{\partial \hat{y}^2} = \sigma^2\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)$, and so on for higher order derivative operators in natural coordinates.

```
Block[{$DisplayFunction = Identity},
 p1 = Table[lapl = gD[im, 2, 0, σ] + gD[im, 0, 2, σ];
   ListDensityPlot[#, PlotRange -> {-90, 60}] & /@ {lapl, σ^2 lapl}
   , {σ, 1, 5}]]; Show[GraphicsArray[Transpose[p1]], ImageSize → 450];
```

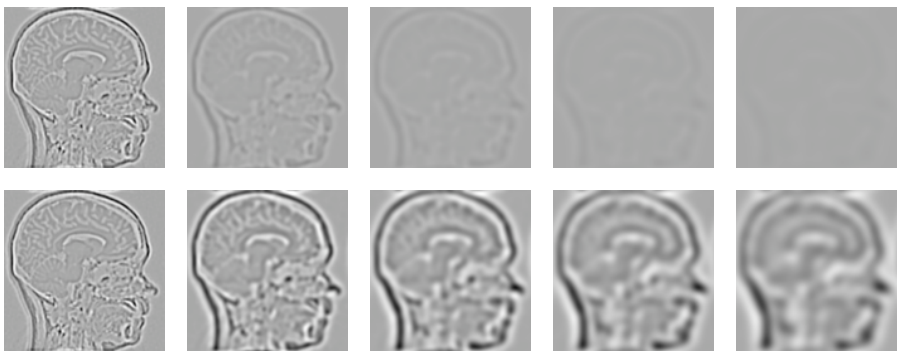

Figure 6.42 The Laplacian of a 128^2 image plotted at 5 scales, for σ = 1, 2, 3, 4 and 5 pixels respectively. Top row: Laplacian in regular coordinates. Bottom row: Laplacian in natural coordinates. Top and bottom rows at fixed intensity range of {-90,60}.

6.12 Irreducible invariants

Invariant differential features are independent of changes in specific groups of coordinate transformations. Note that the transformations of the *coordinates* are involved as the basic physical notion, as the particular choice of coordinates is just a mean to describe the world, the real situation should be independent of this choice. This is often misunderstood, e.g. when rotation invariance is interpreted as that all results are the same when the image itself is rotated. Rotation invariance is a *local* property, and as such a coordinate rotation and an image rotation are only the same when we consider a single point in the image.

For medical imaging the most important groups are the orthogonal transformations, such as translations, rotations, mirroring and scaling, and the affine transformations, such as shear. There are numerous other groups of transformations, but it is beyond the scope of this book to treat this. The differential invariants are the natural building blocks to express local differential structure.

It has been shown by Hilbert [Hilbert1893] that *any invariant of finite order can be expressed as a polynomial function of a set of irreducible invariants*. This is an important result. For e.g. scalar images these invariants form the *fundamental set of image primitives* in which all local intrinsic properties can be described. In other words: any invariant can be expressed in a polynomial combination of the irreducible invariants.

Typically, and fortunately, there are only a small number of irreducible invariants for low order. E.g. for 2D images up to second order there are only 5 of such irreducibles. We have already encountered one mechanism to find the irreducible set: gauge coordinates. We found the following set:

Zeroth order	L
First order	L_w
Second order	L_{vv}, L_{vw}, L_{ww}
Third order	L_{vvv}, L_{vvw}, L_{vww}, L_{www}
etc.	

Each of these irreducible invariants cannot be expressed in the others. Any invariant property to some finite order can be expressed as a combination of these irreducibles. E.g. isophote curvature, a second order local invariant feature, is expressed as: $\kappa = -L_{vv}/L_w$.
Note that the first derivative to v is missing. But $L_v \equiv 0$ is just the gauge condition! There is always that one degree of freedom to rotate the coordinate system in such a way that the tangential derivative vanishes. This gives a way to estimate the number of irreducible invariants for a given order: It is equal to the number of partial derivative coefficients in the local Taylor expansion, minus 1 for the gauge condition. E.g. for the 4^{th} order we have the partial derivatives L_{vvvv}, L_{vvvw}, L_{vvww}, L_{vwww}, and L_{wwww}, so in total we have $1 + 1 + 3 + 4 + 5 = 14$ irreducible invariants for the 4^{th} order.
These irreducibles form a *basis* for the differential invariant structure. The set of 5 irreducible grayvalue invariants in 2D images has been exploited to classify local image structure by Schmidt et al. [Schmidt1996a, Schmidt1996b] for statistical object recognition.

This assigns the three RGB channels of a color image to the irreducible invariants $\{L,\ L_w$ and $L_{v\,v} + L_{w\,w}\}$ of a scalar grayvalue image for $\sigma = 2$ pixels:

```
im = Import["mr256.gif"]; px = im[[1, 1]]; σ = 2;
r = gD[px, 0, 0, σ]; g = √(gD[px, 1, 0, σ]^2 + gD[px, 0, 1, σ]^2);
b = gD[px, 2, 0, σ]^2 + gD[px, 0, 2, σ]^2;
g = g 255/Max[g]; b = b 255/Max[b]; imtr = Transpose[{r, g, b}, {3, 1, 2}];
im[[1, 1]] = imtr; im[[1, 4]] = ColorFunction → RGBColor; Show[im, ImageSize -> 150];
```

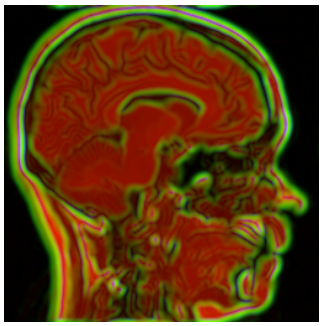

Figure 6.43 RGB color coding with the triplet of differential invariants $\{L,\ L_i\,L_i,\ L_{i\,i}\}$.

Intermezzo: Tensor notation

There are many ways to set up an irreducible basis, but it is beyond the scope of this introductory book to go in detail here. We just give one example of another often used scheme to generate irreducible invariants: tensor notation (see for details e.g. [Florack1993a]). Here tensor *indices* denote partial derivatives and run over the dimensions, e.g. L_i denotes the vector $\{L_x, L_y\}$, $L_{i\,j}$ denotes the second order matrix (the Hessian) $\begin{pmatrix} L_{x\,x} & L_{x\,y} \\ L_{y\,x} & L_{y\,y} \end{pmatrix}$, etc.

When indices come in pairs, summation over the dimensions is implied (the so-called Einstein summation convention, or *contraction*): $L_{i\,i} = \sum_{i=x}^{D} L_{i\,i} = L_{x\,x} + L_{y\,y}$, etc. So we get:

Zeroth order	L	
First order	$L_i\,L_i$	$(= L_x\,L_x + L_y\,L_y$, the gradient)
Second order	$L_{i\,i}$	$(= L_{x\,x} + L_{y\,y}$, the Laplacian)
	$L_{i\,j}\,L_{j\,i}$	$(= {L_{x\,x}}^2 + 2\,L_{x\,y} + {L_{y\,y}}^2$, the 'deviation from flatness'),
	$L_i\,L_{i\,j}\,L_j$	$(= {L_x}^2\,L_{x\,x} + 2\,L_x\,L_y\,L_{x\,y} + {L_y}^2\,L_{y\,y}$, 'curvature')

etc.

Some statements by famous physicists:
- "Gauge invariance is a classic case of a good idea which was discovered before its time." (K. Moriyasu, An Elementary Primer for Gauge Theory, World Scientific, 1984).
- "The name 'gauge' comes from the ordinary English word meaning 'measure'. The history of the use of this name for a class of field theories is very roundabout, and has little to do

with their physical significance as we now understand it." (S. Weinberg, "The Forces of Nature", Am. Scientist, 65, 1977).
- "As far as I see, all a priori statements in physics have their origin in symmetry." (H. Weyl, Symmetry, 1952).

6.13 Summary of this chapter

Invariant differential feature detectors are special (mostly) polynomial combinations of image derivatives, which exhibit invariance under some chosen group of transformations. We only discussed invariance under translations and rotations, the most common groups, especially for medical images. The derivatives are easily calculated from the image through the multi-scale Gaussian derivative kernels.

The notion of invariance is crucial for geometric relevance. Non-invariant properties have no value in general feature detection tasks. A convenient paradigm to calculate features invariant under Euclidean coordinate transformations is the notion of *gauge coordinates*. For first order in 2D they are defined as a local frame with one unit vector $\vec{w}$ pointing in the direction of the gradient, the other perpendicular unit vector $\vec{v}$ pointing in the direction tangential to the isophote. Any combination of derivatives with respect to v and w is invariant under Euclidean transformations. We discussed the second order examples of isophote and flowline curvature, cornerness and the third order example of T-junction detection in this framework.

Mathematica offers a particularly attractive framework, in that it combines the analytical calculation of features under the Euclidean invariance condition with a final replacement of the analytical derivatives with numerical Gaussian derivatives. In this way even high order (up to order 4) examples could be discussed and calculated.