Introduction

The front end visual system belongs to the best studied brain areas. Scale-space theory, as pioneered by Iijima in Japan [11, 41, 43] and Koenderink [12] has been heavily inspired by the important derivation of the Gaussian kernel and its derivatives as regularized differential operators, and the linear diffusion equation as its generating PDE. The view visual system as a ‘geometry’ engine is the inspiration of the current work, and simultaneously, the presented examples of applications of (differential) geometric operations may inspire the thinking of the visual system as a geometry engine.

Scale-space theory has developed into a serious field [34, 20]. Several comprehensive overview texts have been published in the field [15, 10, 40]. The introduction of a geometry driven conduction term in the diffusion equation, making it locally adaptive to differential geometric properties (edge strength, curvature, orientation) by Perona and Malik in the early nineties triggered a wealth of nonlinear PDE developments, which attracted the attention of the mathematical community.

So far, however, this robust mathematical framework has seen impact on the computer vision community, but there is still a gap between the more physiologically, psychologically and psychophysically oriented researchers in the vision community. One reason may be the nontrivial mathematics involved, such as group invariance, differential geometry and tensor analysis.

The last couple of years symbolic computer algebra packages, such as *Mathematica*, Maple and Matlab, have developed into a very user friendly and high level prototyping environment. Especially *Mathematica* combines the advantages of symbolic manipulation and processing with an advanced front-end text processor. This paper has been completely written in *Mathematica* version 4 as a notebook. The advantage is that this paper can be read as an interactive paper: the high level code of any function is directly visible, and can be operated directly, as well as modified or templated for own use. Students can now use the exact code rather then pseudocode. With these high level programming tools most programs can be expressed in very few lines, so it keeps the reader at a highly intuitive but practical level. *Mathematica* notebooks are portable, and run on any system equivalently. Previous speed limitations are now well overcome.
The main focus of the paper is twofold: to provide a rehearsal of the derivation of the Gaussian kernel and its derivatives as an essential class of front-end vision aperture functions, and to provide a practical tutorial for a broad audience to be able to do geometric reasoning with robust multiscale differential operators on discrete images. This may break ground for the view of the front-end visual system as a geometry-engine, or 'inference machine', rather than do a spatial frequency analysis.

This paper can only focus on a small area, the differential geometry and its features. Much research is currently underway. An important area is especially the deep structure of images, where the relations between scales are studied.

**Initialization**

We first initialize *Mathematica* with a path to the image directory, load some Graphics packages, set some often used options for some plotfunctions, turn off the spellingchecker and optimize for speed and memory.

```mathematica
$Path = Append[$Path, "d:/images/"]; << Graphics';
SetOptions[ListDensityPlot, Mesh -> False,
  PlotRange -> All, Frame -> False, AspectRatio -> Automatic];
SetOptions[ListContourPlot, PlotRange -> All, Frame -> False];
SetOptions[ListPlot3D, PlotRange -> All, Axes -> False, Mesh -> False];
SetOptions[Plot3D, PlotPoints -> 30, PlotRange -> All, BoxRatios -> {1, 1, .6},
  Boxed -> False, Shading -> False, Axes -> False, ViewPoint -> {0.950, -2.985, 1.280}];
SetOptions[Integrate, GenerateConditions -> False];
Off[Table::iterb]; Off[StringJoin::string];
Off[General::spell1]; Share[]; $HistoryLength = 10;
```

**Biological inspiration:**

**Receptive field profiles from first principles**

In mathematics objects have no scale. Points have zero size, lines have zero width, differential operators have neighborhoods shrinking to zero, making it strict local operators. In physics however, objects live on a range of scales. Humankind can observe about 50 decades of scale [18]. In physics, dimensional units are essential, there is no such thing as a physical 'point'. In front-end vision the apparatus is specifically equipped to extract multiscale information: the threshold modulation depth is constant (≈ 5%) over more then 2 decades of sizes, i.e. the visual system has a large range of sampling apertures.

One branch of biologically motivated multiscale computer vision is known as 'scale-space theory' [12]. Many axiomatic approaches to scale-space theory exist. For an overview see the well documented paper by Weickert [41]. We start this paper with a treatment of the derivation of the aperture functions in the very first stages of the visual front-end from *first principles*.

We consider the physics of the observation process: Any vision system, whether biological or artificial, has to take samples from a scene in the outside world. This is done through a sampling aperture, which has to have
a finite size in order to integrate the entity to be measured (light intensity, X-ray radiation etc). At this stage, we have no clue for what the size should be, so we leave it a free parameter.

We consider the visual system as a concatenation of steps, where the very first step is responsible for the measurement. This stage has to be uncommitted, the original data should be measured as careful as possible. In fact, at this stage we know nothing, and we have no preference whatsoever for any aspect of the data. We can establish the following first principles:

1. The measurement is done in a linear fashion: we do not allow any nonlinearities at this stage, they should need to incorporate any knowledge of some kind.

2. There is no preference for location in the visual scene: any location should be measured in the same fashion, i.e. with the same aperture function.

3. There is no preference for orientation: structures with a particular orientation, like vertical trees or a horizontal horizon, should have no preference, any orientation is just as likely. This necessitates an aperture function with a circular integration area.

4. There is no preference for size: any size of structure, object, texture etc. is at this stage just as likely. We have no reason just to look only through the finest of apertures. The visual world consists of structures at any size, and this should be measured at any size. The biological motivation comes in here: the retina and subsequent processing layers measure with receptive fields at a wide range of scales..

When we want to establish principal relations between a set of physical quantities involved in a system, such as our front-end observation system, we first study their dependence relation through dimensional analysis. Any physical quantity has a dimension, and consequently it is expressed in units of that dimension. Examples are meters, seconds etc. When we consider such a set of a physical quantities, a typically small number of dimensionless combinations can be formed. The importance of this is given by the fact that these dimensionless units can (and must) be expressed as functions of each other. This gives us a natural, physics based, starting point for the relations to derive. The Pi-theorem states that the number of such dimensionless combinations is equal to the number of variables minus the rank of the matrix \( m \) of the variables against their units.

We exemplify this statement for the visual front-end. We do the reasoning in the Fourier domain, as this turns out to be easier and leads to smaller equations. We give the theory for 2D. We will see that expansion to other dimensionalities is straightforward. We use scripted symbols for variables in the Fourier domain. We consider 'looking through an aperture'. The matrix \( m \) becomes:

\[
\begin{pmatrix}
\sigma & \omega & L_0 & L \\
\text{meter} & 1 & -1 & 0 & 0 \\
\text{cd/m}^2 & 0 & 0 & 1 & 1 \\
\end{pmatrix}
\]

were \( \sigma \) is the size of the aperture, \( \omega \) the spatial coordinate (frequency in the Fourier domain), \( L_0 \) the luminance of the outside world, and \( L \) the luminance as processed in our system. We have four physical entities (\( \sigma, \omega, L_0 \) and \( L \)), and the rank of the matrix is two:

\[
\text{TensorRank}[[\{1, -1, 0, 0\}, \{0, 0, 1, 1\}]]
\]

2
so we may expect two independent dimensionless numbers to be extracted.

The two dimensionless numbers are given by the nullspace of our matrix, i.e. the list of basis vectors that satisfy the matrix equation \( m \tilde{x} = \tilde{0} \).

\[
\text{NullSpace}[\{\{1, -1, 0, 0\}, \{0, 0, 1, 1\}\}] // \text{MatrixForm}
\]

\[
\begin{pmatrix}
0 & 0 & -1 & 1 \\
1 & 1 & 0 & 0
\end{pmatrix}
\]

So from the two rows (for the two dimensionless combinations) we find \( -\frac{k}{L_0} \) and \( \sigma \omega \) as the two basis dimensionless entities. They can therefore be expressed into each other: \( -\frac{k}{L_0} = G(\sigma \omega) \), where \( G \) is the kernel (filter, aperture) function in the Fourier domain to be found. We now plug in our first principles, one by one.

No preference for location, together with the prerequisite for linearity, leads to the recognition of the process as a convolution. The aperture function is shifted over the whole image domain, with no preference for location: any location is measured (‘filtered’, ‘observed’) with the same aperture function (kernel, template, filter, receptive field: all the same thing). This is written as:

\[
L(x, y) = L_0(x, y) \otimes G(x, y) \equiv \int_{-\infty}^{\infty} L_0(u, v) G(x - u, y - v) \, du \, dv
\]

where \( L(x, y) \) is the luminance distribution obtained, \( L(x, y) \) is the luminance distribution in the outside world to be measured, and \( G(x, y) \) is our aperture function in the spatial domain. In the Fourier domain, a convolution of functions translates to a regular product between the Fourier transforms of the functions:

\[
\mathcal{L}(\omega_x, \omega_y) = L_0(\omega_x, \omega_y) \cdot \mathcal{G}(\omega_x, \omega_y)
\]

The axiom of isotropy translates into the fact that we now only have to consider the length of our spatial vector: \( \{ \omega_x, \omega_y \} = \sqrt{\omega} \rightarrow ||\omega|| \).

The axiom of scale-invariance is the core of the reasoning: when we observe (blur) an observed image again, we get an image which is blurred with the same but wider kernel: \( \mathcal{G}(\omega \sigma_1) \mathcal{G}(\omega \sigma_2) = \mathcal{G}(\omega \sigma_1 + \omega \sigma_2) \).

A general solution of this equation is: \( \mathcal{G}(\omega \sigma) = \exp((\alpha \omega \sigma)^p) \). We must raise the argument here to the power of \( p \) because we are dealing with the dimensionless parameter \( \omega \sigma \).

The dimensions are independent, thus separable: \( ||\sqrt{\omega} \sigma|| = (\omega_1 \sigma) \tilde{e}_1 + (\omega_1 \sigma) \tilde{e}_2 + ... \) where \( \tilde{e}_i \) are the basis unit coordinate vectors.

The magnitude of \( ||\sqrt{\omega} \sigma|| \) is calculated by means of Pythagoras from the projections along \( \tilde{e}_i \), so we add the squares, i.e. \( p = 2 \). We further demand the solution to be real, so \( a^2 \) is real. We notice that when we open the aperture fully, we blur everything out, so \( \lim_{\sigma \rightarrow 0} \mathcal{G}(\omega \sigma) = 0 \). This means that \( a^2 \) must be negative. We choose \( a^2 = -\frac{1}{2} \) and finally get the answer: \( \mathcal{G}(\tilde{\omega}, \sigma) = \exp(-\frac{1}{2} \sigma^2 \omega^2) \), which is in the spatial domain:

\[
G(\tilde{x}, \sigma) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{\tilde{x}^2}{2\sigma^2}\right)
\]

This is the Gaussian kernel, which is the Green’s function of the linear, isotropic diffusion equation

\[
\frac{\partial^2 L}{\partial x^2} + \frac{\partial^2 L}{\partial y^2} = L_{xx} + L_{yy} = \frac{\partial L}{\partial s}, \quad \text{where} \quad s = 2 \sigma^2 \quad \text{is the variance}. \quad \text{Note that the derivative to scale is here the derivative to} \ \sigma^2, \quad \text{which also immediately follows from a considerations of the dimensionality of the equation.}
All partial derivatives of the Gaussian kernel are solutions too of the diffusion equation.

So the first important result is that we have found the Gaussian kernel and all of its partial derivatives as the unique kernel for a front-end visual system that satisfies the constraints "no preference for location, scale and orientation" and linearity. We have found a one-parameter family of kernels, where the scale \( \sigma \) is the free parameter. This is a general feature of the biological visual system: the exploitation of ensembles of aperture functions, which are mathematically modeled by families of kernels for a free parameter, e.g. for all scales, derivative order, orientation, stereo disparity, motion velocity etc.

The Gaussian kernel is the unique kernel that generates no spurious resolution (e.g. the squares so familiar with zooming in on pixels). It is the physical point operator, the Gaussian derivatives are the physical derivative operators.

**Gaussian partial derivative kernels**

Here are the receptive field sensitivity structures of some members of the Gaussian derivative family:

```math
\begin{align*}
g[\mathbf{x}, \mathbf{y}, \sigma] := \frac{1}{2 \pi \sigma^2} \exp \left[-\frac{x^2 + y^2}{2 \sigma^2}\right];
\end{align*}
```

```math
Block[{DisplayFunction = Identity},
p1 = Plot3D[g[\mathbf{x}, \mathbf{y}, 1], \{\mathbf{x}, -3.5, 3.5\}, \{\mathbf{y}, -3.5, 3.5\}];
p2 = Plot3D[Evaluate[D[g[\mathbf{x}, \mathbf{y}, 1], \mathbf{x}]], \{\mathbf{x}, -3.5, 3.5\}, \{\mathbf{y}, -3.5, 3.5\}];
p3 = Plot3D[Evaluate[D[g[\mathbf{x}, \mathbf{y}, 1], \mathbf{y}]], \{\mathbf{x}, -3.5, 3.5\}, \{\mathbf{y}, -3.5, 3.5\}];
laplacean[\mathbf{x}, \mathbf{y}, \sigma] := \nabla^2 g[\mathbf{x}, \mathbf{y}, \sigma], \{\mathbf{x}, 2\} + \nabla^2 g[\mathbf{x}, \mathbf{y}, \sigma], \{\mathbf{y}, 2\};
p4 = Plot3D[Evaluate[laplacean[\mathbf{x}, \mathbf{y}, 1]], \{\mathbf{x}, -3.5, 3.5\}, \{\mathbf{y}, -3.5, 3.5\}];
Show[GraphicsArray[{{p1, p2}, {p3, p4}}], ImageSize -> \{300, 240\}];
```
Upper left: the Gaussian kernel as the zero-th order operator; upper right: \( \frac{\partial^2 G}{\partial x^2} \); lower left: \( \frac{\partial^2 G}{\partial x \partial y} \); lower right: \( \frac{\partial^2 G}{\partial y^2} \) of the Gaussian kernel.

The receptive fields in the primary visual cortex closely resemble Gaussian derivatives, as was first noticed by Young [Young 1984, 1986] and Koenderink [Koenderink 1984], and they may accomplish a double simultaneous task: observation and differentiation. These RF’s come at a wide range of sizes, and at all orientations.

Below two examples are given of the measured receptive field sensitivity profile of a cortical simple cell (left) and a Lateral Geniculate Nucleus (LGN) center-surround cell, as measured by DeAngelis, Ohzawa and Freeman [4], [http://totoro.berkeley.edu/].

Left: cortical simple cell, well modeled by a first order Gaussian derivative kernel. Right: center-surround LGN cell, well modeled by the Laplacean of a Gaussian. From [4].

Through the center-surround structure at the very first level of measurement on the retina the Laplacean of the input image can be seen to be taken. The linear diffusion equation states that this Laplacean is equal to the first derivative to scale: \( L_{xx} + L_{yy} = \frac{\partial L}{\partial s} \). One conjecture for its presence at this level could be that the visual system actually might measure \( \frac{\partial L}{\partial s} \), i.e. the slight change in signal \( \partial L \) when the aperture is changed with \( \partial s \); at homogeneous areas there is no output, at highly textured areas there is much output. Integrating both sides of \( \partial L = (L_{xx} + L_{yy}) \partial s \) over all scales gives the measured intensity in a robust fashion.

**Derivatives of sampled, i.e. observed data**

The derivative of the observed data \( L_0(x, y) \otimes G(x, y; \sigma) \) is given by \( \frac{\partial}{\partial s} (L_0(x, y) \otimes G(x, y; \sigma)) \), which can be written as \( L_0(x, y) \otimes \frac{\partial}{\partial s} G(x, y; \sigma) \). The commutation of the convolution and the derivative operators is possible because of their linearity, which is easily shown in the Fourier domain. From this we can see the following important results:

→ Differentiation and observation can be done in a single step: convolution with a Gaussian derivative kernel.

→ Differentiation is now done by *integration*, i.e. by the convolution integral.

→ The Gaussian kernel is the physical analogon of a mathematical point, the Gaussian derivative kernels are the physical analogons of the mathematical differential operators. Equivalence is reached for the limit when
the scale of the Gaussian goes to zero: \( \lim_{\sigma \to 0} G(x, \sigma) = \delta(x) \), where \( \delta(x) \) is the Dirac delta function, and
\[
\lim_{\sigma \to 0} \frac{\partial G(x, \sigma)}{\partial \sigma} = \delta_x.
\]
→ Any differentiation blurs the data somewhat, with the amount of the scale of the differential operator. There is no way out this increase of the inner scale, we can only try to minimize the effect.

The Gaussian kernel has by definition a strong regularizing effect. It was shown by Schwartz [39] that differentiation of distributions of data (such as sampled data) has to be accomplished by convolution with a smooth test function. It is important to realize that the process of observation is the regularizer. Recently some interesting papers have shown the complete equivalence of Gaussian scale space regularization with a number of other methods for regularization [19, 38].

There have been published many formulations to derive the front-end aperture function as the Gaussian kernel and its derivatives. For an overview see Weickert [41] and Lindeberg [17].

### Gabor kernels

The derivation given below required first principles be plugged in that essentially stated “we know nothing” (at this stage of the observation). Of course, we can relax these principles, and introduce some knowledge.

When we want to derive a set of apertures tuned to a specific spatial frequency \( \tilde{k} \) in the image, we add this physical quantity to the matrix of the dimensionality analysis:

\[
\begin{pmatrix}
\sigma & \omega & L_0 & L & \tilde{k} \\
\text{meter} & 1 & 0 & 0 & -1 \\
\text{cd/m}^2 & 0 & 1 & 1 & 1
\end{pmatrix}
\]

Following the exactly similar line of reasoning, we end up from this new set of constraints with a new family of kernels, the Gabor family of receptive fields, with are given by a sinusoidal function (at the specified spatial frequency) under a Gaussian window:

\[
gabor[x, \sigma] := \text{Sin}[x] \frac{1}{\sqrt{2 \pi \sigma^2}} \exp \left[ -\frac{x^2}{2 \sigma^2} \right];
\]

Note the similarity between Gabor and Gaussian derivative kernels. They can be made to look very similar by an appropriate choice of parameters:
The essential difference is that Gabor functions have an infinite number of zero crossings, the Gaussian
derivatives as many as the order of differentiation. By relaxing or modifying other constraints, we might find
other families of kernels. We conclude this section by the realization that the front-end visual system at the
retinal level has a task to be uncommitted, no feedback from higher levels is at stake, so the Gaussian kernel
seems a good candidate to start exploring with at this level. The extensive feedback loops from the primary
visual cortex to LGN may give rise to ’geometry-driven diffusion’ [30], nonlinear scale-space theory, where
the early differential geometric measurements through e.g. the simple cells may modify the kernels at other
levels. Nonlinear scale-space theory will be extensively treated in a forthcoming interactive paper.

\textbf{Differential geometry and invariance}

It is essential to work with descriptions that are \textit{independent of the choice of coordinates}. This was Ein-
stein’s impetus in his development of the general theory of relativity. This means, that when we apply a
transformation on our coordinates, we like our local image properties to be independent of this transfor-
mation. E.g. if we rotate our \{x,y\} coordinate frame, we do not want locale measures as edge strength or curva-
ture to be changed. Coordinate transformations can be divided in \textit{groups}. E.g. all coordinate transformations
that leave the axes of the coordinates perpendicular (e.g. rotations, translations, mirroring and scaling) form
the group of the orthogonal transformations. Another group is the group of the \textit{affine transformations}, where
the new coordinates \{x’,y’\} are acquired through a linear transformation applied to the original coordinates
\{x,y\}, with a, b, c and d constants:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Affine transformations occur when we view objects obliquely at a relatively large distance. At shorter dis-
tances such views are described with \textit{perspective transformations}.
Entities that do not change under a group of coordinate transformations are called *invariants* under that particular group. 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.

In this paper we only study orthogonal and affine invariants, as they form an important basic group and are often encountered in computer vision.

### Multiscale derivatives: implementations

In order to get some feeling of the interactive use of *Mathematica*, we start in this section with three implementations of convolution with a Gaussian derivative kernel (in 2D): implementation in the Fourier domain, in the spatial domain with a 2D kernel, and in the spatial domain exploiting the separability property through two 1D kernel convolutions. Blurring, i.e. convolution with the plain Gaussian kernel, is done through convolution with the zero order Gaussian derivative.

The function $gDf[im, nx, ny, \sigma]$ implements the convolution of the image with the Gaussian derivative for 2D data in the Fourier domain. This is an exact function, no approximations other then the finite periodic window in both domains. We explicitly give the code of the functions here, so you see *how* it is implemented, the reader may make modifications as required. For *Mathematica* novices: all information on (capitalized) internal functions is on board in the Help Browser (highlight+keyF1).

Variables:  

- $im = 2D$ image (as a list structure)
- $nx, ny = \text{order\ of\ differentiation\ to\ x\ resp.\ y}$
- $\sigma = \text{scale\ of\ the\ kernel,\ in\ pixels}$

```math

gDf[im_, nx_, ny_, \sigma_] := Module[{xres, yres, gdkernel},
{xres, yres} = Dimensions[im];
gdkernel = N[Table[Evaluate[D[1/(2 \pi \sigma^2) \text{Exp}[-\frac{x^2+y^2}{2 \sigma^2}], \{x, nx\}, \{y, ny\}]],
{y, -(yres - 1)/2, (yres - 1)/2}, {x, -(xres - 1)/2, (xres - 1)/2}]]; Chop[N[\sqrt{xres \ yres} \text{InverseFourier}[\text{Fourier}[im]
\text{Fourier}[\text{RotateLeft}[gdkernel, \{yres/2, xres/2\}]]]]];
```

This function is rather slow, but is exact. Use it for 64x64 and 128x128 images only.

The function $gDc[im, nx, ny, \sigma]$ implements the same function in the spatial domain. The parameters are the same as above. This function is much faster, as it exploits the internal function *ListConvolve*, and applies Gaussian derivative kernels with a width truncated to +/- 4 standard deviations, which of course can freely be changed.
The fastest implementation exploits the separability of the Gaussian kernel, and this implementation is mainly used in the sequel:

```
gDc[im_, nx_, ny_, σ_] := Module[{x, y, kernel},
  kernel = N[Table[Evaluate[D[1/(2 π σ²) Exp[-x^2/2 σ²], {x, nx}, {y, ny}]],
                 {y, -4 σ, 4 σ}, {x, -4 σ, 4 σ}];
  ListConvolve[kernel, im, Ceiling[Dimensions[kernel]/2]]]
```

Some examples:

Convolving an image with a single point (a delta function) with the Gaussian derivative kernels, gives the kernels themselves, i.e. the point spread function. E.g. the second order to \(x\), first order to \(y\) (at \(σ=10\) pixels) and the eight order to \(x\):

```
spike = Table[0., {128}, {128}]; spike[[64, 64]] = 1.;
Block[{$DisplayFunction = Identity},
  p1 = ListDensityPlot[gD[spike, 2, 1, 10.]]; 
  p2 = ListDensityPlot[gD[spike, 8, 0, 10.]]; 
  Show[GraphicsArray[{{p1, p2}}]]; 
```

The construction with `$DisplayFunction` is necessary to calculate but not display the plots. We read an image with `Import` and only use the first element `[[1,1]]` of the returned structure as this contains the pixel data.
We start with just blurring at a scale of $\sigma=4$ pixels and show the result as image and height plot:

```mathematica
Block[{$DisplayFunction = Identity},
   p1 = ListDensityPlot[gD[im, 0, 0, 4.]]; 
   p2 = ListPlot3D[gD[255 - im, 0, 0, 4]]; ]; 
Show[GraphicsArray[{{p1, p2}}, ImageSize -> {440, 220}];
```

A movie of a linear intensity scale-space is made with the \textbf{Table} function for $\sigma$ exponentially running from 1 to $e^{2.5}$ pixels in steps of $e^{0.25}$ pixel. Double-clicking one of the resulting images starts the animation. Controls are on the bottom windowbar.

```mathematica
im = Import["mr128.gif"][[1, 1]]; 
ss = Table[
   ListDensityPlot[gD[im, 0, 0, Exp[\[sigma]]], ImageSize -> Dimensions[im], 
   \[sigma], 0, 2.5, .25]]; 
```
The sequence can be saved as an animated GIF movie (e.g. for use in webpages) with:
The gradient $\sqrt{L_x^2 + L_y^2}$ on a scale $\sigma = 1$ pixel:

\[
\text{grad} = \text{ListDensityPlot}\left[\sqrt{\text{gD}[\text{im}, 1, 0, 1]^2 + \text{gD}[\text{im}, 0, 1, 1]^2}, \text{ImageSize} \rightarrow \{128, 128\}\right];
\]

To change the window/level (contrast/brightness) settings one must change the displayed range of intensity values:

\[
\text{Show}[\text{grad}, \text{PlotRange} \rightarrow \{0, 30\}];
\]

### Accuracy of differential operators

When we decrease the size of the kernel in the spatial domain, it becomes increasingly difficult to fit the Gaussian derivative kernel with its zerocrossings. For a given order of differentiation we find that there is a limiting scale-size below which the results are no longer exact. E.g. when we study the derivative of a ramp with slope 1, we expect the outcome to be correct. Let us look at the observed derivative at the center of the image for a range of scales ($0.4 < \sigma < 1.2$ in steps of 0.1):
im = Table[x, {y, 64}, {x, 1, 64}];
b = Table[{σ, gDF[im, 1, 0, σ][{32, 32}]}, {σ, .4, 1.2, .1}];
ListPlot[b, PlotJoined -> True, PlotStyle -> Thickness[0.01],
  PlotRange -> {All, {0.8, 1.4}}, AxesLabel -> {"σ", "∂x L"}, AxesOrigin -> {1, .8}];

The value of the derivative starts to deviate for scales smaller than say $\sigma = 0.6$.

There is a fundamental relation between the order of differentiation, scale of the operator and the accuracy required. We will derive now this relation.

The Fourier transform of a Gaussian kernel is again a Gaussian:

$$
\text{gauss}[x_\sigma, \sigma_\_] := \frac{1}{\sqrt{2 \pi \sigma}} \exp \left[ -\frac{x^2}{2 \sigma^2} \right];
$$

$$
\text{fftgauss}[\omega_\sigma, \sigma_\_] = \text{FourierTransform}[\text{gauss}[x, \sigma], x, \omega]
$$

$$
\frac{e^{-\frac{1}{2} \sigma^2 \omega^2}}{\sqrt{2 \pi}}
$$

The Fourier transform of the $n$-th derivative of a function is $(i\omega)^n$ times the Fourier transform of the function:

$$
\text{fftgaussD}[\omega_\sigma, \sigma_\_] = \text{FourierTransform}[D[\text{gauss}[x, \sigma], x], x, \omega]
$$

$$
-\frac{1}{\sqrt{2 \pi}} \frac{\sigma^2 \omega^2}{\sqrt{2 \pi}}
$$

A smaller kernel in the spatial domain gives rise to a wider kernel in the Fourier domain, as shown below for a range of widths of first order derivative Gaussian kernels (in 1D):
We plot the Fourier spectrum of a kernel that shows aliasing:

\[
\text{error}[n_\omega, \sigma_] = 100 \frac{\int_{-\pi}^{\pi} (I \omega)^2 \cdot \text{fftgauss}[\omega, \sigma]^2 d\omega}{\int_0^{2\pi} (I \omega)^2 \cdot \text{fftgauss}[\omega, \sigma]^2 d\omega}
\]

\[
100 \left( (1 + 2n) \text{Gamma}\left[\frac{1}{2} + n\right] - 2 \text{Gamma}\left[\frac{1}{4} + n\right] + (1 + 2n)^2 \text{Gamma}\left[\frac{1}{2} + n, \pi^2 \sigma^2\right]\right)
\]

\[
(1 + 2n)^2 \text{Gamma}\left[\frac{1}{2} + n\right]
\]
We plot this Gammafunction for scales between $\sigma = 0.2$ and order of differentiation from 1 to 10, and we insert the 5% error line in it (we have to lower the plot somewhat to make the line visible):

```math
Block[{DisplayFunction = Identity},
    p1 = Plot3D[error[n, \sigma] - 6, {\sigma, .2, 2}, {n, 1, 10}, PlotRange -> All,
             AxesLabel -> {"\sigma", "n", "error \%"}, Boxed -> True, Axes -> True];
    p2 = ContourPlot[error[n, \sigma], {\sigma, .2, 2}, {n, 1, 10},
                     ContourShading -> False, Contours -> 5];
    c3d = Graphics3D[Graphics[p2][[1]] /. Line[pts_] :> ({Thickness[.01],
              val = Apply[error, First[pts]]; Line[Map[Append[#, val] &, pts]]});]
    Show[p1, c3d];
```

The lesson from this section is that we should never make the scale of the operator, the Gaussian kernel, too small. The lower limit is indicated in the graph above. A similar reasoning can be set up for the outer scale, when the aliasing occurs in the spatial domain.

---

**Natural coordinates**

The intensity of images and invariant features at larger scale decreases fast. This is due to the non-scaleinvariant use of the differential operators. For, if we consider the transformation $x \rightarrow \hat{x}$, then $\hat{x}$ is dimensionless. At every scale now distances are measured in a distance yardstick with is scaled with the scale itself, i.e. scale-invariant. The dimensionless coordinate is termed the *natural* coordinate. This implies that the derivative operator in natural coordinates has a scaling factor: $\frac{\partial}{\partial \sigma} \rightarrow \sigma^n \frac{\partial}{\partial \sigma}$.

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 $[0,40]$:
Clearly the gradient expressed in the natural coordinates keeps its average output range. For a Laplacean scale-space stack in natural coordinates we need to multiply the Laplacean with $\sigma^2$:

$$\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} = \sigma^2 \left( \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right),$$

and so on for higher order derivative operators in natural coordinates.

**Discrete Gaussian Kernels**

Lindeberg [Lindeberg 1990] derived the optimal kernel for the case when the Gaussian kernel was discretized and came up with the "modified Bessel function of the first kind". In *Mathematica* this function is available as `BesselI`.

The "modified Bessel function of the first kind" `BesselI` is almost equal to the Gaussian kernel for $\sigma > 1$, as we see below. Note that the Bessel function has to be normalized by its value at $x = 0$. For larger $\sigma$ the kernels become rapidly very similar.
Gauge coordinates

Gauge coordinates are a very useful tool in computer vision, and in understanding the possible geometric functionality of the front-end visual system. Gauge coordinates are connected with the isophotes, lines of equal brightness in 2D images (surfaces in 3D). Here are 10 equidistant isophotes of an image in 10 different colors (\@ stands for Map):

```mathematica
im = Import["mr256.gif"][[1, 1]]; max = Max[im];
ListContourPlot[im/max, ContourShading -> False, Contours -> 10,
ContourStyle -> List /@ Hue /@ (Range[10]/10), ImageSize -> {128, 128}];
```

In order to establish differential geometric properties it is easiest to exploit intrinsic geometry. This means that we will define a new coordinate frame for our geometric explorations which is related to the local isophote structure, so it is different in every different point. A straightforward definition of a new local coordi-
nate frame in 2D is where we cancel the degree of freedom of rotation by defining gauge coordinates: we locally ‘fix the gauge’. The 2D unit vector frame of gauge coordinates \([v, w]\) is defined as follows: \(w\) is the unit vector in the gradient direction, i.e. the direction in which the intensity changes fastest, \(v\) is defined perpendicular to \(w\), i.e. tangential to the intensity isophote. The new framevectors are drawn below:

\[
\begin{align*}
\partial_v &= \frac{-L_x \delta_x + L_y \delta_y}{\sqrt{L_x^2 + L_y^2}} = \frac{L_x \delta_i \partial_i}{\sqrt{L_x l_i}} \\
\partial_w &= \frac{L_x \delta_x + L_y \delta_y}{\sqrt{L_x^2 + L_y^2}} = \frac{L_y \delta_i \partial_i}{\sqrt{L_y l_i}}.
\end{align*}
\]

The derivatives to \(v\) and \(w\) are by definition features that are invariant under orthogonal transformations, i.e. rotation and translation. To apply these gauge derivative operators on images, we have to convert to the Cartesian \([x, y]\) domain. The derivatives to \(v\) and \(w\) are defined as:

\[
\begin{align*}
\partial_v &= \frac{-L_x \delta_x + L_y \delta_y}{\sqrt{L_x^2 + L_y^2}} = \frac{L_x \delta_i \partial_i}{\sqrt{L_x l_i}} \\
\partial_w &= \frac{L_x \delta_x + L_y \delta_y}{\sqrt{L_x^2 + L_y^2}} = \frac{L_y \delta_i \partial_i}{\sqrt{L_y l_i}}.
\end{align*}
\]

We can alternatively see the derivatives to \([v, w]\) as rotated over an angle \(\alpha\), with rotation matrix

\[
\begin{pmatrix}
\cos(\alpha) & \sin(\alpha) \\
-\sin(\alpha) & \cos(\alpha)
\end{pmatrix} = \begin{pmatrix} L_x & L_y \\ -L_y & L_x \end{pmatrix}.
\]

The second formulation uses tensor notation, where the index \(i\) or \(j\) stands for the range of dimensions. So \(L_i \equiv \{L_x, L_y\}\) in 2D and \(L_i \equiv \{L_x, L_y, L_z\}\) in 3D. Likewise \(\delta_j\) is the nabla operator \(\{\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\}\). The constant tensors \(\delta_{ij}\) and \(\epsilon_{ij}\) are the symmetric Kronecker tensor \(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\) and the antisymmetric Levi-Civita tensor \(\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}\) respectively (in 2D). With this notation, we see that the derivative operator \(\partial_w\) is defined as the derivative operator \(\partial_j\) rotated in the direction (through \(\delta_{ij}\)) of the unit length gradient vector \(\frac{L_i}{\sqrt{L_x l_i}}\), the derivative operator \(\partial_e\) is defined as the derivative operator \(\partial_j\) rotated in the perpendicular direction (through \(\epsilon_{ij}\)) of the unit length gradient vector \(\frac{L_i}{\sqrt{L_x l_i}}\). So we encounter 3 types of notation for our geometrical exer-
cises: Cartesian coordinate notation \([x, y]\), tensor notation \(L_i L_j\), and gauge coordinate notation \([v, w]\). They are essentially equivalent. In this paper we will elaborate mainly on gauge coordinates.

The definitions above are easily accomplished in Mathematica:

\[
\delta^2 = \text{IdentityMatrix}[2]
\]

\[
\begin{bmatrix} 1, 0 \end{bmatrix}, \begin{bmatrix} 0, 1 \end{bmatrix}
\]

\[
\epsilon^2 = \text{Table}[\text{Signature}[[i, j]], [j, 2], [i, 2]]
\]

\[
\begin{bmatrix} 0, -1 \end{bmatrix}, \begin{bmatrix} 1, 0 \end{bmatrix}
\]

\[
\text{jacobian} = \sqrt{L_x^2 + L_y^2} ;
\]

\[
dv = \frac{1}{\text{jacobian}} \{L_x, L_y\} . \epsilon^2 . [D[#], D[#]] &
\]

\[
\frac{\{L_x, L_y\} . \epsilon^2 . \{\partial_x \#1, \partial_y \#1\}}{\text{jacobian}} &
\]

\[
dw = \frac{1}{\text{jacobian}} \{L_x, L_y\} . \delta^2 . [D[#], D[#]] &
\]

\[
\frac{\{L_x, L_y\} . \delta^2 . \{\partial_x \#1, \partial_y \#1\}}{\text{jacobian}} &
\]

The notation \((...\#) &\) is a 'pure function' on the argument \#, e.g. \((#^2 + #^5) &\) gives the sum of second and fifth power of some argument, \(D [x, \#] &\) takes the derivative. This function can be applied to an argument by the familiar square brackets: \((#^2 + #^5) & [zz]\). Look in the Help browser to the function \text{Function} for examples.

Now we can calculate any derivative to \(v\) or \(w\) by applying the operator \(dw\) or \(dv\) repeatedly. Note that the \(L_x\) and \(L_y\) are constant terms, in fact the combination \(L_i e_i\) is precisely the rotation matrix to rotate the local coordinate frame to the definitions of \(v\) and \(w\). So after the application of \(dv\) and \(dw\) we need to substitute \(L_x\) and \(L_y\) with the image derivatives with the command / (which is the substitute command). \text{Mathematica} needs to know what are the differentiation variables, so this function works when we explicitly state a function of \(x\) and \(y\), such as \(L[x, y]\) \((L\) for luminance).

\[
Lw = dw[L[x, y]] / . \{Lx \rightarrow \partial_x L[x, y], Ly \rightarrow \partial_y L[x, y]\} // \text{Simplify}
\]

\[
\sqrt{L[0, 1][x, y]^2 + L[1, 0][x, y]^2}
\]
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, i.e. an invariant property where translation 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 gauge derivatives to any order.

Applications of multiscale invariants on discrete images: 'looking through the simple cells'

The definitions for the gauge differential operators $\partial_v$ and $\partial_w$ need to have their regular differential operators be replaced by Gaussian derivative operators. To just show the textual formula, we do not yet evaluate the derivative by using temporarily HoldForm:

So here is the Cartesian (in $\{x,y\}$) expression for $L_{vw}$:

Every Gaussian derivative is evaluated as a separate image, and the invariant is the polynomial concatenation.
Ridge detection

$L_{vv}$ is a ridge detector. Let us test this on an X-ray image of fingers and calculate $L_{vv}$ on a scale $\sigma = 2$:

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

With the function ReleaseHold we release the Hold function, so now gD is not just displayed as name, but actually called and calculated:

```math
L_{vv} = gauge2D[im, 2, 0, 3] // ReleaseHold;
```

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

```math
im = Table[Random[], {128}, {256}];
nosieridges = gauge2D[im, 2, 0, 3] // ReleaseHold;
ListDensityPlot[nosieridges, ImageSize -> {256, 128}];
```

We also recognize $L_{vv}$ in the ‘fundamental’ equation of Alvarez et al. [2], a nonlinear geometry driven diffusion equation: $\frac{\partial L_{vv}}{\partial \sigma} = L_{vv}$. 

Isophote curvature in gauge coordinates

Isophote curvature $\kappa$ is defined as the change $w''$ of the tangent vector $w'$ in the gauge coordinate system. When we differentiate the definition of the isophote ($L = \text{Constant}$) to $v$, we get:

$$D[L[v, w[v]] = \text{Constant}, v]$$

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

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:

$$\kappa = w''[v] / \text{Solve}[D[L[v, w[v]] = \text{Constant}, \{v, 2\}] / \text{w'}[v] \rightarrow 0, w'[v]]$$

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

So $\kappa = -\frac{L_{xx}}{L_{uu}}$. QED.

In Cartesian coordinates we recognize the well-known formula from classical textbooks:

$$\kappa = -\frac{\partial^2 L[x, y]}{\partial x \partial y}$$

To see this more clearly we write the previous expression of partial Cartesian derivatives in a more readable notation by a pattern matching operation ($\rightarrow$) where the Derivative function is replaced by $L$ with a string of $x$'s and $y$'s:

$$\kappa = -\frac{\text{HoldForm}[gD/im, n, m]}{\text{HoldForm}[gL[im, n, m]]}$$

Here is an example of the isophote curvature at a range of scales for a sagittal MR image:
The reason we see extreme low and high values is due to the singularities that occur at intensity extrema, where the gradient $L_w = 0$.

In a similar fashion we can derive and study the curvature of *flowlines*, which are defined as the curves everywhereperpendicular to isophotes. The flowline curvature $\mu$ is given by $\mu = -\frac{L_{xx}}{L_w}$.

Zerocrossings of the Laplacean have historically received much attention, due to the work of Marr and Hildreth. The zerocrossings are however displaced on curved edges. Note that with the compact expression for isophote curvature $\kappa = -\frac{L_{xx}}{L_w}$ we can establish a relation between the Laplacean and the proper second order derivative to study for zerocrossings: $L_{ww}$. From the expression of the Laplacean in gauge coordinates $L_{ww} + L_{yy} = L_{ww} - \kappa L_w$ we see that there is a deviation which is directly proportional to the curvature $\kappa$. 

```math
Block[{$DisplayFunction = Identity},
   p1 = ListDensityPlot[im];
   p2 = Table[ListDensityPlot[k // ReleaseHold, PlotRange -> {-5, 5}, {σ, 1, 3}];
   Show[GraphicsArray[Partition[Prepend[p2, p1], 2]], ImageSize -> {400, 400}];
```
Affine invariant corner detection

Corners can be defined as locations with high isophote curvature and high intensity gradient. It was proposed by Blom [3]:

\[ \Theta_{in} = - \frac{L_{xy}}{L_{ww}} L_n = \kappa L_n. \]

An obvious advantage is invariance under as large a group as possible. Blom calculated \( \Theta_{in} \) for invariance under the affine transformation

\[ \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}. \]

The derivatives transform as

\[ \begin{pmatrix} \frac{\partial}{\partial x'} \\ \frac{\partial}{\partial y'} \end{pmatrix} = \begin{pmatrix} a & c \\ b & d \end{pmatrix} \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix}. \]

The corner detectors \( \Theta_{in} \) transform as

\[ \Theta_{in} = (a d - b c)^2 \left[ (a L_x + c L_y)^2 + (b L_x + d L_y)^2 \right]^{n-2/3} \left[ 2 L_x L_y L_x'y' - L_y^2 L_x'y - L_x^2 y - L_y^2 y' \right]. \]

This is a relative affine invariant of order 2 if \( n = 3 \) with the determinant \( D = (a d - b c) \) of the affine transformation as order parameter. We consider here special affine transformations \( (D = 1) \). So a good corner-detector is \( \Theta = \frac{L_{xy}}{L_{ww}} L_n^3 = L_{xy} L_n^2. \) This feature has the nice property that is is not singular at locations where the gradient vanishes, and through its affine invariance it detects corners at all ‘opening angles’.

Note the positive (convex) and negative (concave) corners. We show corner detection at two scales:

\[
\text{im} = \text{N[Import["utrecht256.gif"]][[1, 1]]];
\]

\[
\text{corner1} = (\text{gauge2D[im, 2, 0, 1]} \text{gauge2D[im, 0, 1, 1]}^2) / \text{ReleaseHold};
\]

\[
\text{corner2} = (\text{gauge2D[im, 2, 0, 3]} \text{gauge2D[im, 0, 1, 3]}^2) / \text{ReleaseHold};
\]

\[
\text{Block[{$DisplayFunction = \text{Identity}},
\text{p1} = \text{ListDensityPlot[im]};
\text{p2} = \text{ListDensityPlot[corner1]};
\text{p3} = \text{ListDensityPlot[corner2]};
\text{Show[GraphicsArray[\{p1, p2, p3\}, \text{ImageSize} -> \{436, 136\}];}
\]
**T-junction detection**

An example of third order geometric reasoning in images is the detection of T-junctions [24]. T-junctions in the intensity landscape of natural images occur typically at occlusion points. In the figure below the circles indicate a few particular T-junctions:

```math
blocks = Import["blocks.gif"][[1, 1]]; Block[{$DisplayFunction = Identity}, p1 = ListDensityPlot[blocks, ImageSize -> {317, 204}]; p2 = Graphics[{Circle[{221, 178}, 13], Circle[{157, 169}, 13], Circle[{90, 155}, 13], Circle[{148, 56}, 13], Circle[{194, 77}, 13], Circle[{253, 84}, 13]}]; Show[{p1, p2}, AspectRatio -> Automatic];
```

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 \( \kappa \) 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 \( \kappa \):
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 $\frac{\partial \omega}{\partial w}$. We saw before that the isophote curvature $\kappa$ is defined as $\kappa = -\frac{L_x}{L_w}$. So the Cartesian expression for the T-junction detector becomes

$$\kappa = -\frac{\partial^2 v}{\partial w^2} \left( \begin{array}{c} L_x \nabla^2 L \end{array} \right) \bigg|_{x,y}$$

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 $\tau = \frac{\partial \omega}{\partial w} L_w^6$, the derivative of the curvature in the direction perpendicular to the isophotes (an affine invariant?):

$$\tau = \text{Simplify}[\frac{\partial \omega}{\partial w} L[x, y] / (L_x^2 + L_y^2)^3]$
Fourth order 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. [37].

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

\[
pol4 = \frac{1}{4!} (Lxxxx x^4 + 4 Lxxxy x^3 y + 6 Lxxyy x^2 y^2 + 4 Lyxxy x y^3 + Lyyyy y^4);
\]

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 by Hilbert that the 'coincidencesness' of the roots, i.e. how well all roots coincide, is a particular invariant condition. From algebraic theory it is known that this 'coincidencesness' is given by the discriminant, defined below (see also [1]):

\[
\text{Discriminant}[p_, x_] :=
\text{With}[[m = \text{Exponent}[p, x]],
\frac{(-1)^{\frac{1}{2} m (m-1)}}{\text{Coefficient}[p, x, m]} \text{Resultant}[p, \partial_x p, x]]
\]
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:

\[
\text{Discriminant} \left[ \frac{1}{2} (Lxx x^2 + 2 Lxy x y + Lyy y^2), x \right] / \{y \to 1\}
\]

\[Lxy^2 - Lxx Lyy\]

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:

\[
\text{Discriminant} [\text{pol4}, x] / \{y \to 1\}
\]

\[
\frac{1}{746496} (36 Lxxxy^2 Lxxyy^2 Lxyyy^2 - 54 Lxxxx Lxxyy Lxyyy^2 - \\
64 Lxxxxy^3 Lxyyy^3 + 108 Lxxxx Lxxyy Lxyyy^2 - 27 Lxxxx^2 Lxyyy^4 - \\
54 Lxxxx^2 Lxxyy^3 Lyyyy + 81 Lxxxx Lxxyy^4 Lyyyy + 108 Lxxxx^3 Lxxyy Lxyyy Lyyyy - \\
180 Lxxxx Lxxyy Lxyyy^2 Lyyyy - 6 Lxxxx Lxxyy^2 Lxyyy^2 Lyyyy + \\
54 Lxxxx^2 Lxxyy^2 Lyyyy^2 Lyyyy - 27 Lxxxx^4 Lyyyy^2 + 54 Lxxxx Lxxxyy^2 Lxxyy Lyyyy^2 - \\
18 Lxxxx^2 Lxxxyy^2 Lyyyy^2 - 12 Lxxxx^2 Lxxxyy Lyyyy Lyyyy^2 + Lxxxx^4 Lyyyy^3)
\]

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 change all coefficients into scaled Gaussian derivatives:

\[
discr4[im, _\sigma] := \\
\text{Discriminant} [\text{pol4}, x] / \{y \to 1, Lxxxx \to \text{gD}[\text{im}, 4, 0, \sigma], Lxxxy \to \text{gD}[\text{im}, 3, 1, \sigma], \\
Lxxyy \to \text{gD}[\text{im}, 2, 2, \sigma], Lxyyy \to \text{gD}[\text{im}, 1, 3, \sigma], Lyyyy \to \text{gD}[\text{im}, 0, 4, \sigma]\}
\]

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:
The detection clearly is rotation invariant, robust to noise, and there is no detection at corners:

```
ListDensityPlot[discr4[noisycheck, 5], ImageSize -> {200, 100}];
```

---

**Conclusion**

Biologically motivated Gaussian derivative kernels provide a solid framework for differential geometric analysis in computer vision. In this paper a practical overview is given of some results up to fourth order of differentiation. This multiscale analysis is applicable to all fields of computer vision, including nonlinear geometry-driven diffusion \[2, 30, 42\], optic flow, stereo disparity analysis, orientation analysis, scale-time \[13\], deep structure of images etc. Space constraints do not allow to elaborate on these issues in this paper. This paper is an excerpt from a forthcoming book \[35\], where many of the issues above are treated in an interactive way.

This paper has been written as a notebook in *Mathematica* 4.0, giving the possibility to the reader to experiment with every treated subject himself. The high level of functions and speed of code and hardware now available (this full notebook runs in 12 minutes on a Pentium II PC, 266 MHz, 128K memory, Win95) and the easy interactive visualization possibilities makes the combination of textbook text and code a highly tutorial toolkit on the desktop.
Acknowledgement

The authors thank the members of the Image Sciences Institute, particularly the members of the TGV ('Tools of Geometry in Vision') team for their contributions and discussions. Special thanks to Max Viergever, director of ISI.

References


