Computational Vision
U. Minn. Psy 5036
Daniel Kersten
Lecture 13: Edge Detection

Initialize

■ Read in Statistical Add-in packages:

```
Off[General::spell1];
<< Statistics`DescriptiveStatistics`
<< Statistics`DataManipulation`
<< Statistics`NormalDistribution`
<< Graphics`PlotField`
```

```
<< Statistics`ContinuousDistributions`
<< Statistics`MultinormalDistribution`
```

1) Define 64x64 pixel cup image (cup), or

■ Evaluate this: cup

```
width = Dimensions[ cup][[1]];
```

2) alternatively import image file

```
cup = Import[Experimental`FileBrowse[False]];
```

```
cup =  cup /. Graphics → List;
cup =  cup[[1, 1]];
width = Dimensions[ cup][[1]];
```

Outline

Last time

■ Efficient coding

Task neutral

1rs order statistics

- point-wise non-linearities, histogram equalization, receptors and information capacity

2nd order statistics

- lateral inhibition, ganglion cells and predictive coding

-opponent color processing (principal components analysis)

--cortical representation

Decorrelation, PCA

3rd or higher orders?

Today: Continue with discussion of the two views of the function of early local visual spatial coding

■ Spatial (difference) filtering as efficient coding or as part of a system of edge detectors (or both?)

■ Local image measurements that correlate well with useful surface properties

task specific--find "significant" intensity changes

edge detection

1rst and 2nd spatial derivatives (i.e. the edge and bar detectors)

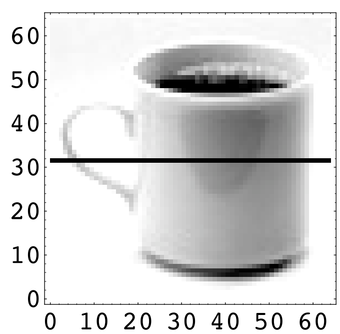
relation to center-surround and oriented receptive fields

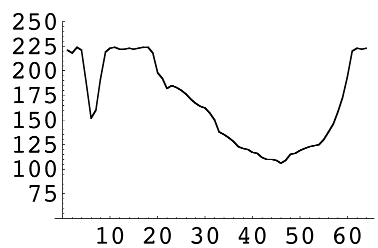
■ **Problems with edge detection**

Edge Detection

Introduction

```
new = cup;
new[[width / 2, All]] = 0;
ListDensityPlot[new, PlotRange → {50, 200}, Mesh → False];
ListPlot[ cup[[width / 2]], PlotJoined → True, PlotRange → {50, 250}];
```





Edge detection as differentiation

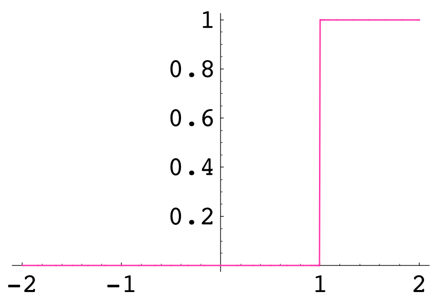
■ The Noise/Scale trade-off

The definition of edge detection is very tricky--exactly what do we want to detect? We would like to label "significant" intensity changes in the image. One definition of significant edges is that they are the ones with the biggest change in intensity. The biggest would correspond to step changes. In *Mathematica*, these can be modeled as g(x) = **UnitStep[]**. One of the first problems we encounter is that edges in an image are typically fuzzy, either due to optical blur in the imaging device, or because the scene causes of edges are not changing abruptly. Our generative model for image data f is to convolve the step with a blur function:

f(x) = $\int g(x - x')\,\mathrm{blur}(x')\,dx'$ = g*blur

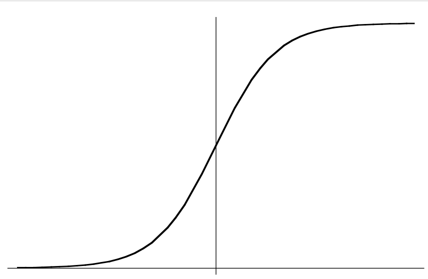
where g() is the signal to be detected or estimated. g(x) is a step function:

```
g[x_] := UnitStep[x - 1];
Plot[g[x], {x, -2, 2}, PlotStyle → Hue[0.9]];
```



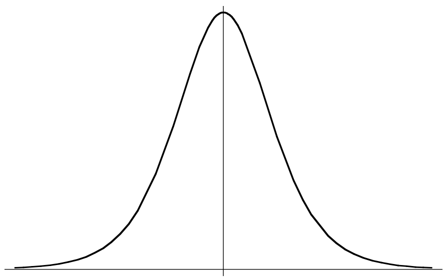
Depending on the type of blur, the image intensity profile f(x) will look more or less like:

```
edge[x_,s_] := 1/(1 + Exp[-x/s])
Plot[edge[x,.3],{x,-2,2},Ticks->None];
```



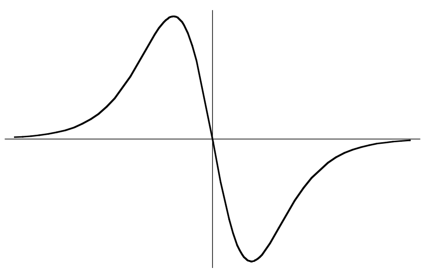
One way of locating the position of the edge in this image would be to take the first derivative of the intensity function, and then mark the edge location at the peak of the first derivative:

```
dedge[u_, s_] := D[edge[x, t], x] /. x → u /. t → s
Plot[dedge[u, .3], {u, -2, 2}, Ticks → None];
```



Alternatively, we could take the second derivative, and look for zero-crossings to mark the edge location.

```
d2edge[u_,s_] := D[dedge[x,t],x]/.x->u /.t->s
Plot[d2edge[u,.3],{u,-2,2},Ticks->None];
```



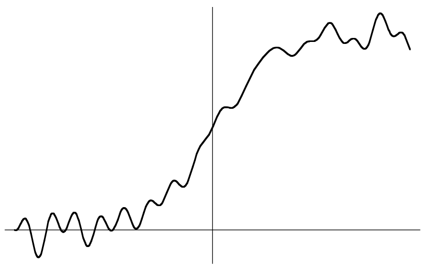
So far so good. But real images rarely have a nice smooth intensity gradation at points that we subjectively would identify as a clean edge. A more realistic generative model for intensity data would be:

$$\mathrm{f(x)} = \int g(x - x')\,\mathrm{blur}(x')\,dx' + \mathrm{noise}$$

We'll add a fixed sample of high-frequency "noise":

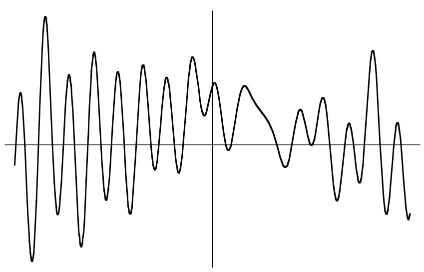
```
noisyedge[x_,s_] := edge[x,s] +
        0.01 Cos[10 x] + -0.02 Sin[10 x] + 0.03 Cos[12 x] + 0.04 Sin[12
x] +
        -0.01 Cos[13 x] + -0.03 Sin[13 x] + 0.01 Cos[14 x] + 0.01 Sin[14
x] +
        -0.04 Cos[25 x] + -0.02 Sin[25 x] + 0.02 Cos[26 x] + 0.03 Sin[26
x];

Plot[noisyedge[x,.3],{x,-2,2},Ticks->None];
```



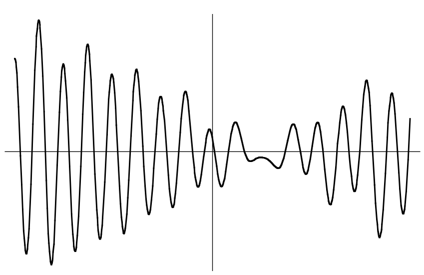
Now, if we take the derivative, there are all sorts of peaks, and the biggest isn't even where the edge is:

```
dnoisyedge[u_,s_] := D[noisyedge[x,t],x]/.x->u /.t->s
Plot[dnoisyedge[u1,.3],{u1,-2,2},Ticks->None];
```



Looking for zero-crossings looks hopeless:

```
d2noisyedge[u_,s_] := D[dnoisyedge[x,t],x]/.x->u /.t->s
Plot[d2noisyedge[u1,.3],{u1,-2,2},Ticks->None];
```



There are many spurious zero-crossings, and the higher the frequency of the noise, the bigger the problem gets. Here is the 3rd derivative of a component with frequency f:

```
D[Sin[x f], {x, 3}]
```

```
-f^3 Cos[f x]
```

■ A solution: pre-blur using convolution

As in your Assignment #2, a possible solution to the noise problem is to pre-filter the image with a convolution operation that blurs out the fine detail which is presumably due to the noise. And then proceed with differentiation. The problem is how to choose the degree of blur. Blur the image too much, and one can miss edges; don't blur it enough, and one gets false edges.

This is one edge detection dilemma: *Too much blur and we miss edges, too little and we have false alarms.*

Some biologically motivated edge detection schemes

Edge detection using 2nd derivatives: Marr-Hildreth

Assignment 2 looked at one scheme for edge detection that has received some attention for its biological plausibility. This is the Marr-Hildreth edge detector. The idea is to: 1) pre-blur with a Gaussian; 2) take second derivates of the image intensity using the Laplacian ; 3) locate zero-crossings. In short,

Find zero-crossings of: r(x,y) = $\int \nabla^2 G_\sigma(x', y')\, g(x - x', y - y')\, dx'\, dy' = (\nabla^2 G_\sigma)*g$

$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ is the Laplacian operator, which takes the second derivatives in x and y directions, and sums up the result.

As you saw in Assignment 2, the Laplacian and convolution operators are combined into the "del-squared G" operator, $\nabla^2 G_\sigma$, where

$$G_\sigma[x,y] = \frac{e^{\frac{1}{2}\left(-\frac{x^2}{\sigma} - \frac{y^2}{\sigma}\right)}}{2\pi\sigma}$$

```
Σ = {{σ, 0}, {0, σ}}; μ = {0, 0};
ndist = MultinormalDistribution[μ , Σ ];
PDF[ndist, {x, y}]
```

The order of the operators doesn't matter, so one can take the Laplacian of the Gaussian first, and then convolve this del-squared G kernel with the image, or one can blur the image first, and then take the second derivatives:

r(x,y) = $\nabla^2 (G_\sigma * g)$

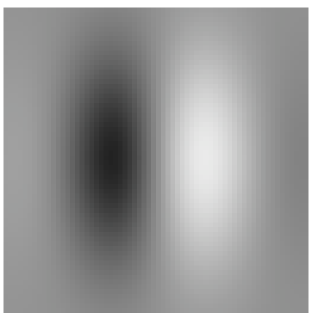
As σ approaches zero, G_σ becomes a delta function, and the $\nabla^2 G_\sigma$ becomes a Laplacian ∇^2, i.e. a second derivative operator. For small σ, the detector is sensitive to noise. For large σ, it is less sensitive to noise, but misses edges. The biological appeal of the Marr-Hildreth detector is that lateral inhibitory filters provide the ($\nabla^2 G_\sigma$) kernel. But what about the oriented filters in the cortex? One interpretation is in terms of 1rst derivatives.

One could also build zero-crossing detectors by ANDing the outputs of appropriately aligned center-surround filters effectively building oriented filters out of symmetric ganglion-cell (or LGN) like spatial filters (Marr and Hildreth).

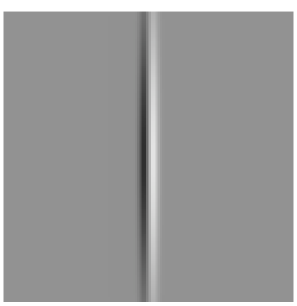
Edge detection using 1rst derivatives

Because of the orientation selectivity of cortical cells, they have sometimes been interpreted as edge detectors. We noted earlier how a sine-phase Gabor function filter (1 cycle wide) would respond well to an edge oriented with its receptive field.

```
swidth=32;
sgabor[x_,y_, fx_, fy_,sig_] :=
    N[Exp[(-x^2 - y^2)/(2 sig*sig)] Sin[
        2 Pi (fx x + fy y)]];
edgefilter = Table[sgabor[i/32,j/32,0,2/3,1/2],
    {i,-swidth,swidth-1},{j,-swidth,swidth-1}];
    ListDensityPlot[edgefilter,Mesh→
False,Frame->False,PlotRange->{-1,1}];
```



These sine-phase or odd-symmetric filters can also be viewed as 1rst order spatial derivatives.



How can we combine oriented filters to signal an edge? The first-derivative operation takes the gradient of the image. From calculus, you learned that the gradient of a 2D function evaluated at (x,y) is a vector that points in the direction of maximum change. So taking the gradient of an image should produce a vector field where the vectors are perpendicular to the edges.The length of the gradient is a measure of the steepness of the intensity gradient.

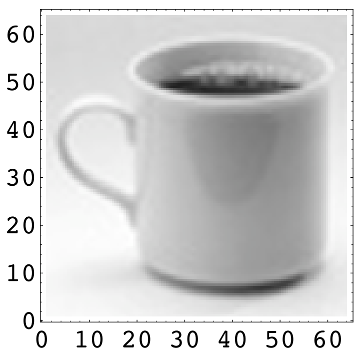
■ The gradient of a function

$$\nabla g = \left(\frac{\partial g(x, y)}{\partial x}, \frac{\partial g(x, y)}{\partial y} \right)$$

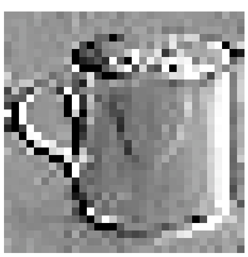
Let's try it with the cup image.

```
contcup = ListInterpolation[Transpose[cup], {{1, width}, {1, width}}];
```

```
DensityPlot[contcup[x, y], {x, 1, 64}, {y, 1, 64}, Mesh → False,
  PlotPoints → 128];
```




```
DensityPlot[Evaluate[D[contcup[x, y], x]], {x, 1, width}, {y, 1, width},
  PlotPoints → width / 2, Mesh → False, Frame → False];
```



Let's take the derivatives in the x and y directions:

```
fxcontcup[x_, y_] := D[contcup[x1, y1], x1] /. { x1 → x, y1 → y};
fycontcup[x_, y_] := D[contcup[x1, y1], y1] /. { x1 → x, y1 → y};
```

Now let's put the x and y directions together and compute the squared gradient magnitude:

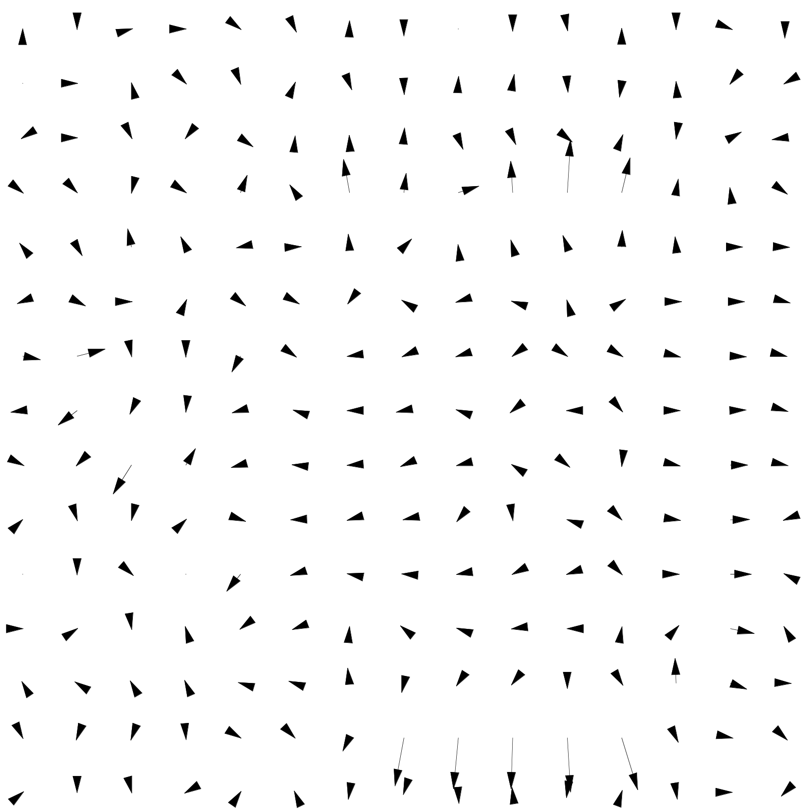
```
fcontcup[x_, y_] := D[contcup[x1, y1], x1]^2 + D[contcup[x1, y1], y1]^2 /.
   { x1 → x, y1 → y};
```

```
DensityPlot[fcontcup[x, y], {x, 1, width}, {y, 1, width},
  PlotPoints → width / 2, Mesh → False, Frame → False];
```

Doesn't look too bad, but it isn't clean and some of our satisfaction is premature and the result of our visual system effectively fitting the edge representation above into the interpretation of a cup.

Let's try a different representation and plot up the gradient vector field for the cup image:

```
PlotVectorField[{fxcontcup[x, y], fycontcup[x, y]}, {x, 1, width},
  {y, 1, width}];
```



Imagine trying to link up points along an edge with this information---You get a better idea of how much variability remains in terms of both direction and magnitude.

If we took many pictures of the same cup under different illumination conditions, one could measure how much variability (at a point) is in the magnitude vs. diretion of the gradient. Chen et al. (2000) did this and showed that there is much more variability in the magnitude than the direction of the gradient.

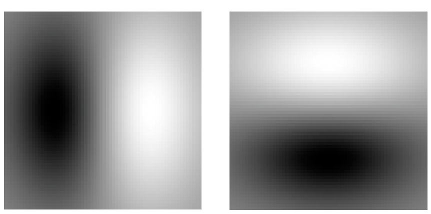
■ Combining a smoothing pre-blur with 1rst derivatives

So the idea is to blur the image, and then take the first derivates in the x and y directions, square each and add them up, the 1rst derivative analog to the 2nd derivative ∇^2G operator.

The x and y components of the gradient of the blur kernel :

```
G[x_, y_, σx_, σy_] := ⅇ^(1/2 (-x^2/σx - y^2/σy)) / (2 π Sqrt[(σx^2 + σy^2)]);
dGx[x_, y_] := D[G[x1, y1, 1, 2], x1] /. {x1 → x, y1 → y};
xg = DensityPlot[-dGx[x, y], {x, -2, 2}, {y, -2, 2}, Mesh → False,
   Frame → False, PlotPoints → 64, DisplayFunction → Identity];
dGx[x_, y_] := D[G[x1, y1, 2, 1], y1] /. {x1 → x, y1 → y};
yg = DensityPlot[-dGx[x, y], {x, -2, 2}, {y, -2, 2}, Mesh → False,
   Frame → False, PlotPoints → 64, DisplayFunction → Identity];
Show[GraphicsArray[xg, yg], DisplayFunction → $DisplayFunction];
```

```
Show[GraphicsArray[{xg, yg}], DisplayFunction → $DisplayFunction];
```



--2D smoothing operator followed by a first order directional derivatives in the x and y directions.

If one takes the outputs of two such cells, one vertical and one horizontal, the sum of the squares of their outputs correspond to the squared magnitude of the gradient of the smoothed image:

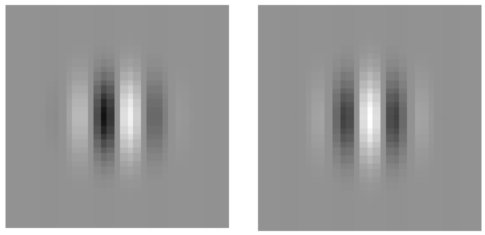
$(r_x(\mathrm{x,y}), r_y(\mathrm{x,y})) = (\frac{\partial G_\sigma(x,y)}{\partial x}\ ,\ \frac{\partial G_\sigma(x,y)}{\partial y}) * \mathrm{g(x,y)} = \nabla G_\sigma * \mathrm{g(x,y)} = (\frac{\partial G_\sigma(x,y)}{\partial x} * g(x,\ y)\ ,\ \frac{\partial G_\sigma(x,y)}{\partial y} * g(x,\ y))$

Then to get a measure of strength, compute the squared length:

$$|\nabla G_\sigma * g(x,\ y)\ |^2 = \ |\ \nabla\ (G_\sigma * \mathrm{g}\ (\mathrm{x,\ y}))\ \ |^2 = r_x(\mathrm{x,y})^2 + r_y(\mathrm{x,y})^2$$

Morrone & Burr edge detector--combining even and odd filters

The Marr-Hildreth 2nd derivative operation is similar to the even-symmetric cosine-phase gabor or "bar detector". The 1rst derivative gradient operator is similar to the odd-symmetric sine-phase gabor. Any reason to combine them?



Sometimes the important "edge" is actually line--i.e. a pair of edges close together. A line-drawing is an example.

The appendix shows how one can combine both sine and cosine phase filters to detect both edges and lines. A sine and cosine phase pair are sometimes called "quadrature (phase) pairs". The summed squared outputs can be interpreted as "local contrast energy".

We'll encounter the idea of quadrature pairs later in the context of motion detection, when we consider derivatives in time as well as space.

Combining signal detection theory with edge detection

Canny (1986).

Konishi et al. (2003)

Possible project idea: Build a Bayesian edge detector using gradient statistics measured on and off of real edges, and using methods from signal detection theory to decide theory to decide whether a given measure ment is on or off of an edge.

Problems with interpreting V1 simple/complex cells as edge detectors

Although one can build edge detectors from oriented filters, simple cells cannot uniquely signal the presence of an edge for several reasons. One is that their response is a function of many different parameters. A low contrast bar at an optimal orientation will produce the same response as a bar of higher contrast at a non-optimal orientation. There is a similar trade-off with other parameters such as spatial frequency and temporal frequency. In order to make explicit the location of an edge from the responses of a population of cells, one would have to compute something like the "center-of-mass" over the population, where response rate takes the place of mass. Another problem is that edge detection has to take into account a range of spatial scales. We discussed evidence earlier that the cortical basis set does encompass a range of spatial scales, and in fact may be "self-similar" across these scales. See Koenderink (1990) for a theoretical discussion of "ideal" receptive field properties from the point of view of basis elements.

Konishi et al. (2003) used signal detection theory and real images to show that there is an advantage in combining information across scales when doing edge detection.

Segmentation & Why edge detection is hard

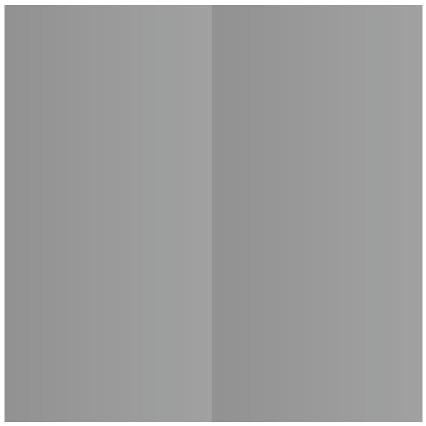
T*he same intensity gradient means different things depending on context*

The finding of Knill & Kersten

■ Land & McCann's "Two squares and a happening"

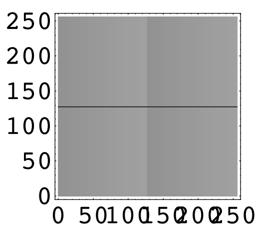
```
size = 256;Clear[y]; slope = 0.0005;
y[x_] := slope x +0.5 /;x<1*size/2
y[x_] := slope (x-128) +0.5  /; x>=1*size/2
```

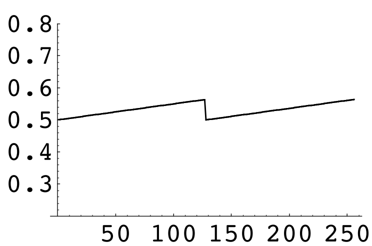
```
picture = Table[Table[y[i],{i,1,size}],{i,1,size}];
ListDensityPlot[picture,Frame->False,Mesh->False,
                 PlotRange->{0,1}, AspectRatio->Automatic];
```



The left half looks lighter than the right half. But, let's plot the intensity across a horizontal line:

```
new = picture;
new[[128, All]] = 0;
ListDensityPlot[new, Mesh → False, PlotRange → {0, 1}];
ListPlot[ picture[[128]], PlotJoined → True, PlotRange → {0.2, .8}];
```





The two ramps are identical...tho' not too surprising in that that is how we constructed the picture. How can we explain this illusion based on what we've learned so far?

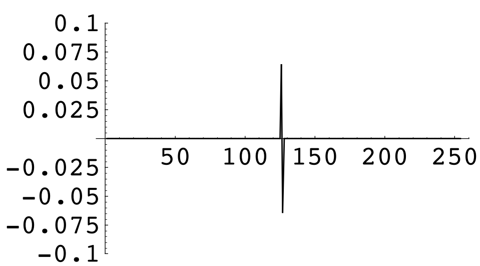
One explanation is that the visual system takes a spatial derivative of the intensity profile. Recall from calculus that the second derivative of a linear function is zero. So a second derivative should filter out the slowly changing linear ramp in the illusory image. We approximate the second derivative with a discrete kernel (-1,2,-1).

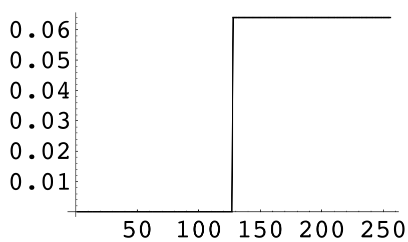
The steps are: 1) take the second derivative of the image; 2) threshold out

```
filter = {-1, 2, -1};
(*Take the second derivative at each location*)
fspicture = ListConvolve[filter, picture[[128]]];
ListPlot[fspicture, PlotJoined → True, PlotRange → {-0.1, .1}];

(*Now integrate twice-- to undo the the second derivative and
  "restore" the picture*)
integratefspicture = FoldList[Plus, fspicture[[1]], fspicture];
integratefspicture2 = FoldList[Plus, integratefspicture[[1]],
   integratefspicture];

ListPlot[integratefspicture2, PlotJoined → True];
```





To handle gradients that aren't perfectly linear, we could add a threshold function to set small values to zero before re-integrating:

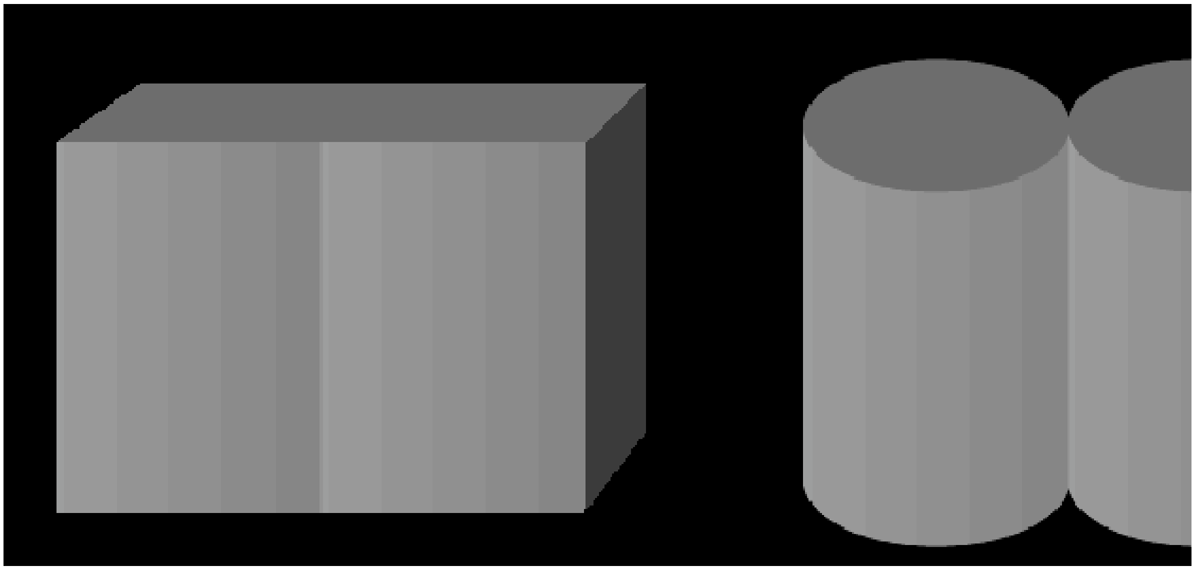
```
threshold[x_, τ_] := If[x > τ, x, 0]; SetAttributes[threshold, Listable];
fspicture = threshold[fspicture, 0.025];
```

Or one can take just the first derivative, followed by the threshold function

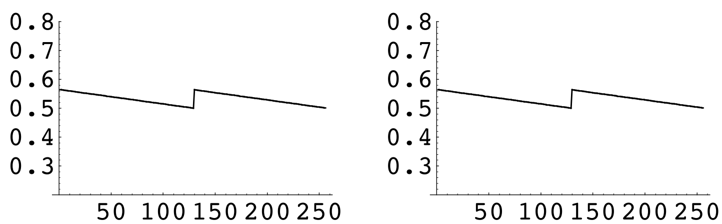
■ Knill & Kersten's "Two cylinders and no happening"

But is edge enhancement and and spatial filtering a good way to explain the lightness effect? Up until the early 1990's many people thought so, and this was a standard textbook explanation of these kinds of lightness illusions.

What if we measure the intensity across a horizontal line in the "slab" on the left, and the "two-cylinders" on the right?



They are also the same! They would both look something like this:

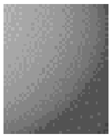


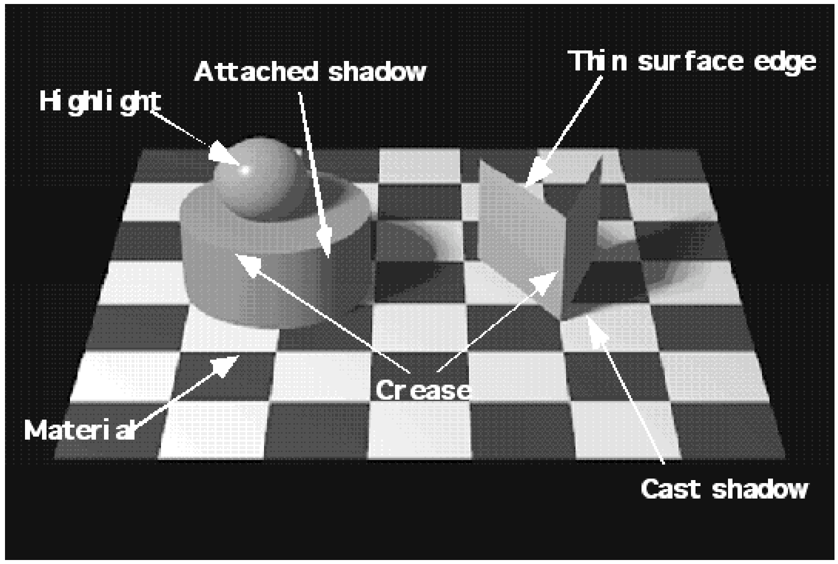
But the perceived lightness contrast for the slabs is significantly stronger than it is for the two cylinders. A spatial convolution/derivative model would predict the same for both. The spatial convolution operation won't work as an explanation!

One interpretation of this observation is *that the visual system has knowledge of the type of edge*--i.e. whether it is due to pigment or to self-occlusion/contact.

Edge classification: Some causes of edges are more important than others: task-dependence

We’ve seen that uncertainty due to noise and spatial scale confound reliable edge detection. But the above demonstrates another reason why edge detection is hard--local intensity gradients can have several possible meanings.





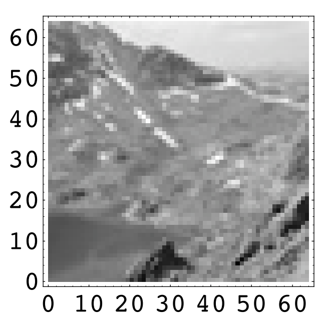
So on the one hand, it still makes sense to interpret lateral inhibitory filters and oriented cortical filters as possible components of an edge detection system, but we have to allow for considerable uncertainty in the significance of their outputs--i.e. a local edge detector typically has a low signal-to-noise ratio for a variety of ways of defining signals. To translate edge detection into useful segmentations is a non-trivial computer vision problem. For state-of-the-art work on this, see Malik et al. (2001) and Tu and Zhu (2002), and a preprint: http://www.stat.ucla.edu/~sczhu/papers/IJCV_parsing.pdf

For tasks such as object recognition, vision places a higher utility on surface and material edges than on other types. Surface edges are used differently from material edges. Shadow edges are variable yet potentially important for stereo. Specular edges are variable, but problematic for stereo because they are at different surface positions for the two eyes.

Natural images & segmentation

■ Lonesome peak

Where to draw the important contours?



Early spatial filtering seems more appealing as efficient encoding.

Next time

- **Mid-term exam**

Next lecture

- **Image-based modeling: geometry, contours & long lines**

- **Surfaces from images**

- **Scene-based modeling**

Appendices

Derivative filters

See http://gandalf.psych.umn.edu/~kersten/kersten-lab/courses/Psy5036W2004/Lectures/8.%20Spatial%20filters/-Fourier_neural_image.nb.

```
gauss[x_, y_, σ_] := 1/(2 π σ^2) Exp[-(x^2 + y^2)/(2 σ^2)];
```

$$\texttt{gauss[x\_, y\_, }\sigma\texttt{\_] := } \frac{1}{2\,\pi\,\sigma^2}\ \texttt{Exp}\left[\frac{-(x^2+y^2)}{2\,\sigma^2}\right];$$

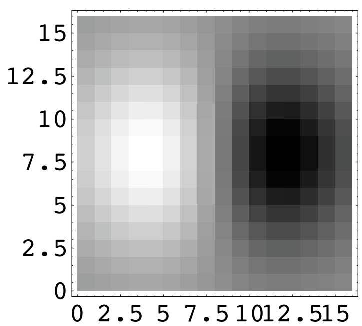
```
derivativeGauss[x_, y_, j_, k_, σ_] := D[gauss[x, y, σ], {x, j}, {y, k}];
```

```
derivativeGausskernel[j_, k_, σ_, xwidth_, ywidth_] :=
  N[Table[Evaluate[derivativeGauss[x, y, j, k, σ]],
    {y, -(ywidth - 1)/2, (ywidth - 1)/2}, {x, -(xwidth - 1)/2, (xwidth - 1)/2}]];
```

```
gaussD[image_, j_, k_, σ_] :=
    Module[{xwidth, ywidth, kernel},
        {ywidth, xwidth} = Dimensions[image];
        kernel =
    RotateLeft[derivativeGausskernel[j, k, σ, xwidth, ywidth],
     {ywidth / 2, xwidth / 2}];
        Chop[N[InverseFourier[Fourier[image] Fourier[kernel]]]]];
```

■ First derivative filter in x direction

```
xwidth = 16; ywidth = 16;
ListDensityPlot[derivativeGausskernel[1, 0, 4, xwidth, ywidth],
  Mesh → False];
```

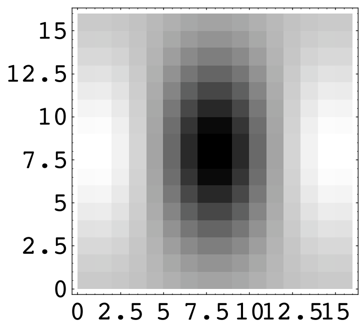


Take first x derivative of gaussian filtered image:

```
ListDensityPlot[gaussD[face, 1, 0, 1], Mesh → False, Frame → False];
```

■ Second derivative filter in x direction

```
xwidth = 16; ywidth = 16;
ListDensityPlot[derivativeGausskernel[2, 0, 4, xwidth, ywidth],
  Mesh → False];
```



■ Laplacian revisited

Recall that the ∇^2G operator is given by the Laplacian of a gaussian, G(x,y):

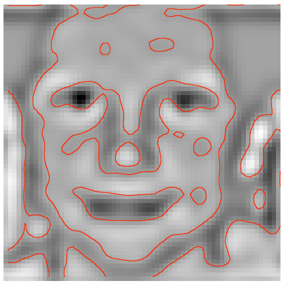
$$\nabla^2 G = \frac{\partial^2}{\partial x^2} G(x,y) + \frac{\partial^2}{\partial y^2} G(x,y) \tag{1}$$

Thus, the convoloution, ∇^2G*face, is given by:

```
laplacian[image_, σ_] := -gaussD[image, 2, 0, σ] - gaussD[image, 0, 2, σ];
```

```
gdelsqimage = ListDensityPlot[delsqimage = laplacian[face, 2],
   Mesh → False, Frame → False];
```

```
contours = ListContourPlot[delsqimage, Contours -> {0},
   ContourShading -> False,
   ContourStyle -> {Thickness[0.003], RGBColor[1, 0, 0]},
   DisplayFunction → Identity];
Show[{gdelsqimage, contours}];
```



Morrone & Burr: polarity sensitive & polarity insensitive

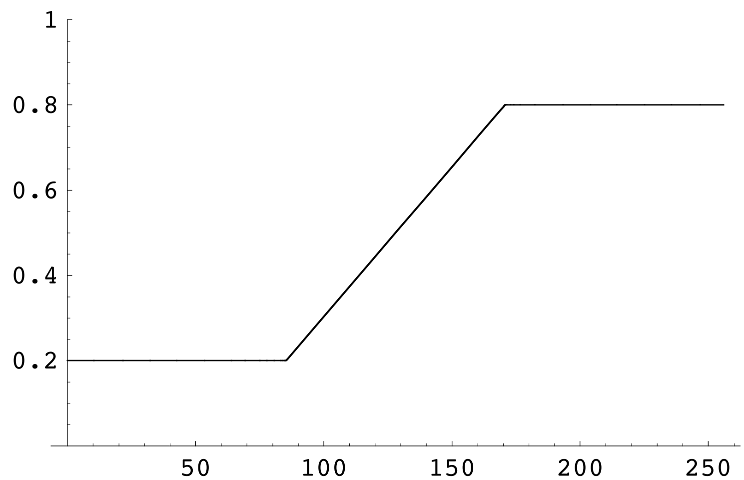
■ Morrone and Burr edge/bar detectors

Suppose we convolve an input signal with an even filter (e.g. Gaussian enveloped cosine-wave) to produce reponse Re, and then convolve the same input with an odd filter (say, a Gaussian enveloped sine-wave) to produce response Ro. The filters are orthogonal to each other, and so are the responses. Re will tend to peak at "bars" in the image whose size is near half the period of the cosine-wave. Ro will tend to peak near edges.

The local contrast "energy" is defined to be: Sqrt[Re^2 + Ro^2]. Morrone and Burr showed that the local energy peaks where the Fourier components of an image line up with zero-phase--i.e. at points where the various Fourier components are all in sine-phase. These points are edges. But it also peaks near bar features, arguably also interesting image features where the phase coherence is at 90 degrees. In addition to its neurophysiological appeal, a particularly attractive feature of this model is that if one adds up responses over multiple spatial scales, evidence accumulates for edges because the local energy peaks coincide there. They also showed how their model could be used to explain Mach bands.

■ Mach bands & the Morrone & Burr edge detector

```
size = 256;Clear[y];
low = 0.2; hi = 0.8;
y[x_] := low /; x<size/3
y[x_] :=
    ((hi-low)/(size/3)) x + (low-(hi-low)) /; x>=size/3 && x<2*size/3
y[x_] := hi /; x>2*size/3
Plot[y[x],{x,0,256},PlotRange->{0,1}];
```

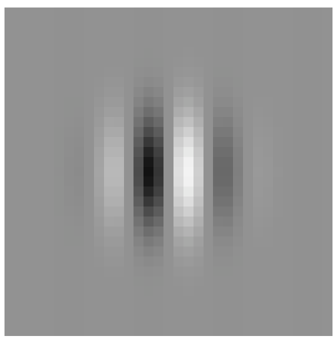



```
picture = Table[Table[y[i],{i,1,size}],{i,1,size}];
ListDensityPlot[picture,Frame->False,Mesh->False,
                PlotRange->{0,1}, AspectRatio->Automatic];
```

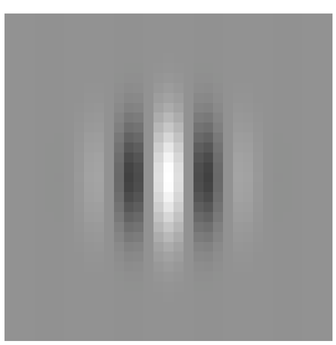
■ Gabor filters

```
sgabor[x_, y_, fx_, fy_, sig_] :=
  N[Exp[(-x^2 - y^2) / (2 sig * sig)] Sin[2 Pi (fx x + fy y)]];
cgabor[x_, y_, fx_, fy_, sig_] :=
  N[Exp[(-x^2 - y^2) / (2 sig * sig)] Cos[2 Pi (fx x + fy y)]];
```

```
fsize = 32;
sfilter = Table[sgabor[(i - fsize / 2), (j - fsize / 2), 0, 1 / 8, 4],
  {i, 0, fsize}, {j, 0, fsize}];
sfilter = Chop[sfilter];
g10 = ListDensityPlot[sfilter, Mesh → False, PlotRange → {-1, 1},
   Frame → False];
```

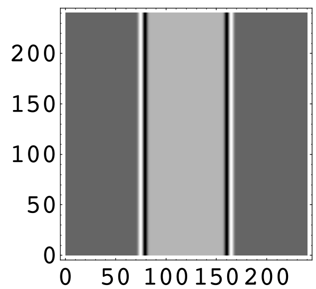


```
fsize = 32;
cfilter = Table[cgabor[(i - fsize / 2), (j - fsize / 2), 0, 1 / 8, 4],
  {i, 0, fsize}, {j, 0, fsize}];
cfilter = Chop[cfilter];
g11 = ListDensityPlot[cfilter, Mesh → False, PlotRange → {-1, 1},
   Frame → False];
```



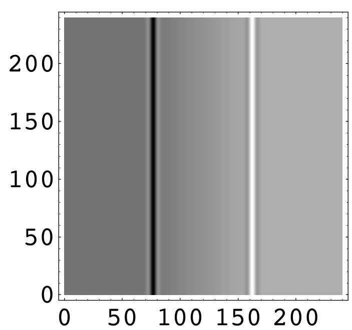
■ **Apply odd (sine) filter**

```
fspicture = ListConvolve[sfilter, picture];
ListDensityPlot[fspicture, Mesh → False];
```



■ **Apply even (cosine) filter**

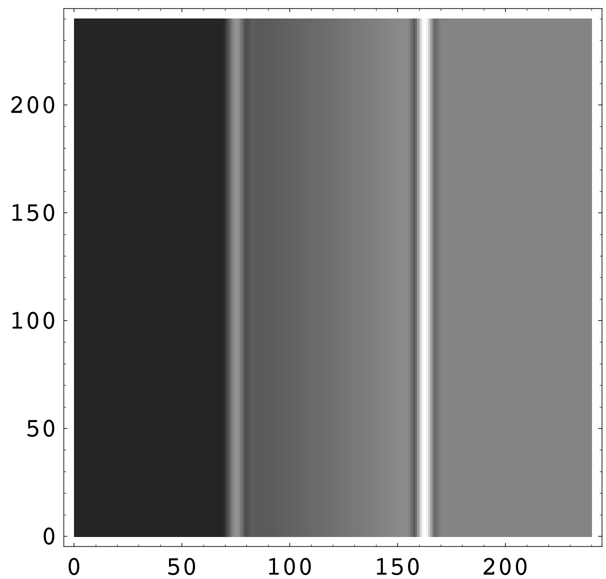
```
fcpicture = ListConvolve[cfilter, picture];
ListDensityPlot[fcpicture, Mesh → False];
```



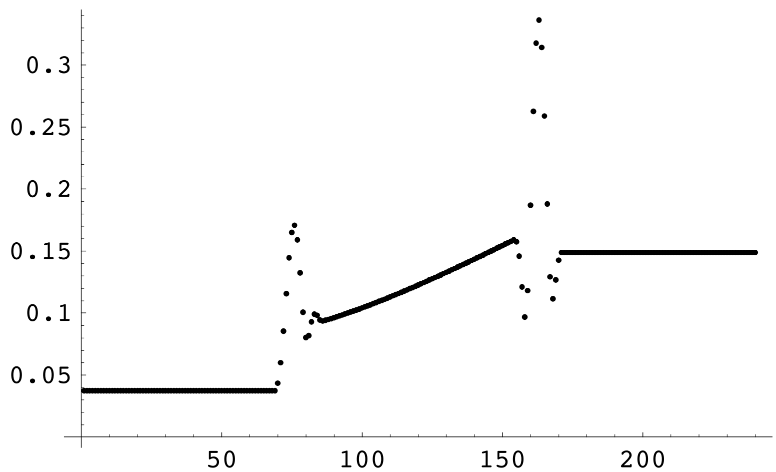
■ **Look for peaks in local contrast energy**

```
ss = Sqrt[fspicture^2 + fcpicture^2];
```

```
ListDensityPlot[ss, Mesh → False];
```




```
ListPlot[ss[[128]]];
```




```
- Graphics -
```

References

Buchsbaum, G., & Gottschalk, A. (1983). Trichromacy, Opponent Colour Coding and Optimum Information Transmission in the Retina. *Proc. Roy. Soc. Lond. B, 220*, 89-113.

Carandini, M., Heeger, D. J., & Movshon, J. A. (1997). Linearity and normalization in simple cells of the macaque primary visual cortex. J Neurosci, 17(21), 8621-44.

Canny, J. F. (1986). A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 8((6)), 679-698.

Chen, H. F., Belhumeur, P. N., & Jacobs, D. W. (2000). In search of illumination invariants. *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1*, 254--261.

Hallinan, P. W. (1994). *A low-dimensional lighting representation of human faces for arbitrary lighting conditions*. Paper presented at the IEEE Conf. on Computer Vision and Pattern Recognition, Seattle.

Marr, D., & Hildreth, E. (1980). Theory of Edge Detection. Proceedings of the Royal Society of London, B207, 187-217.

Morrone, M. C., & Burr, D. (1988). Feature detection in human vision: a phase dependent energy model. Proceedings of the Royal Society, London, 235, 221-245.

Gaska, J. P., Pollen, D. A., & Cavanagh, P. (1987). Diversity of complex cell responses to even- and odd-symmetric luminance profiles in the visual cortex of the cat. Exp Brain Res, 68(2), 249-59.

Field, D. J., & Tolhurst, D. J. (1986). The structure and symmetry of simple-cell receptive-field profiles in the cat's visual cortex. Proc. R. Soc. Lond., 228(B), 379-400.

Foldiak, P. (1989). *Adaptive network for optimal linear feature extraction*. Paper presented at the IEEE/ INNS International Joint Conference on Neural Networks, Washington D. C.

Horn, B. K. P. (1986). *Robot Vision*. Cambridge MA: MIT Press.

Kammen, D. M., & Yuille, A. L. (1988). Spontaneous Summetry-Breaking Energy Functions and the Emergence of Orientation Selective Cortical Cells. *Biological Cybernetics, 59*, 23-31.

Knill, D. C., & Kersten, D. (1991). Apparent surface curvature affects lightness perception. *Nature, 351*, 228-230.

Konishi, S. M., Yuille, A. L., Coughlan, J. M., & Zhu, S. C. (2003). Statistical edge detection: Learning and evaluating edge cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 25*(1), 57-74.

Liu, Z., Gaska, J. P., Jacobson, L. D., & Pollen, D. A. (1992). Interneuronal interaction between members of quadrature phase and anti- phase pairs in the cat's visual cortex. Vision Res, 32(7), 1193-8.

Malik, J., Belongie, S., Leung, S. H., & Shi, J. (2001). Contour and Texture Analysis for Image Segmentation. *International Journal of Computer Vision, 43*(1), 7–27.

Olshausen, B. A., & Field, D. J. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature, 381*, 607-609.

Sanger, T. (1989). Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks, 2*, 459-473.

Sanger, T. D. (1990). Analysis of the Two-Dimensional Receptive Fields Learned by the Generalized Hebbian Algorithm in Response to Random Input. *Biological Cybernetics, 63*(MIT), 221-228.

Tu, Z., & Zhu, S.-C. (2002). Image Segmentation by Data-Driven Markov Chain Monte Carlo. *IEEE Trans. on Pattern Analysis and Machine Intelligence, 24*(5), 657-673.

Watt, R. J., & Morgan, M. J. (1983). The Recognition and Representation of Edge Blur: Evidence for Spatial Primitives in Human Vision. Vision Research, 23(12), 1465-1477.

Yuille, A. L., Kammen, D. M., & Cohen, D. S. (1989). Quadrature and the development of orientation selective cortical cells by Hebb rules. *Biological Cybernetics, 61*, 183-194.

© 2004, 2006 Daniel Kersten, Computational Vision Lab, Department of Psychology, University of Minnesota.
kersten.org