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

Initialize

```
SetOptions[ArrayPlot, ColorFunction → "GrayTones",
  DataReversed → False, Frame → False, AspectRatio → Automatic,
  Mesh → False, PixelConstrained → True, ImageSize → Small];
SetOptions[ListPlot, ImageSize → Small];
SetOptions[Plot, ImageSize → Small];
SetOptions[DensityPlot, ImageSize → Small, ColorFunction → GrayLevel];
nbinfo = NotebookInformation[EvaluationNotebook[]];
dir = ("FileName" /. nbinfo /. FrontEnd`FileName[d_List, nam_, ___] ⧴ ToFileName[d]);

downsample[imaged_, f_] := Take[imaged, {1, -1, f}, {1, -1, f}]


cup = ImageData[     ];


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

Outline

Last time

Efficient coding

Task neutral
1rst 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?
    contrast normalization

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 are correlated with useful surface properties

task specific--e.g. find "significant" intensity changes, likely to belong to an object boundary
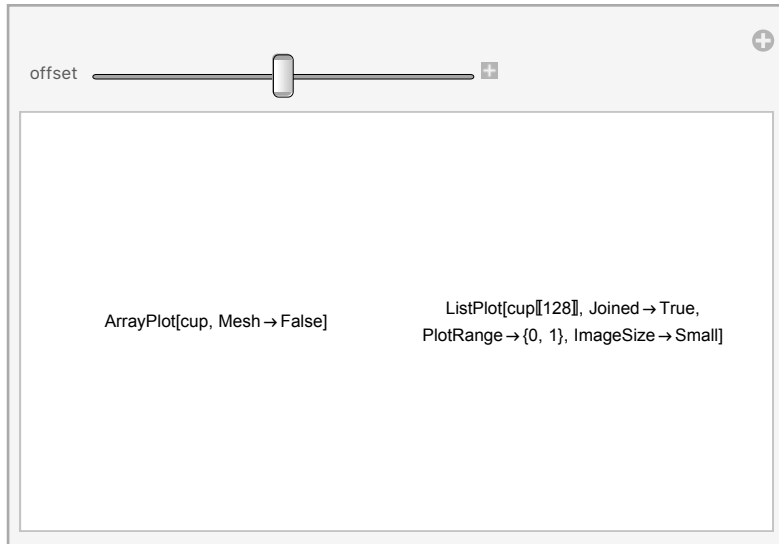edge detection
1rst and 2nd spatial derivatives (i.e. the edge and bar detectors)
relation to center-surround and oriented receptive fields

Problems with edge detection

Edge detection as differentiation

```
Manipulate[
 new = cup;
 new[[offset, All]] = 0;
 GraphicsRow[{ArrayPlot[new, Mesh → False],
   ListPlot[ cup[[offset]], Joined → True, PlotRange → {0, 1}, ImageSize → Small]}],
 {{offset, width / 2}, width / 2 - 63, width / 2 + 63, 1}]
```

offset ⬚

ArrayPlot[cup, Mesh → False]

ListPlot[cup〚128〛, Joined → True,
PlotRange → {0, 1}, ImageSize → Small]

⚫ **Set**: Part specification new〚FE`offset$$1068, All〛 is longer than depth of object.

⚫ **ArrayPlot**: Argument cup at position 1 is not a list of lists.

⚫ **Part**: Part specification cup〚128〛 is longer than depth of object.

⚫ **ListPlot**: cup〚128〛 is not a list of numbers or pairs of numbers.

⚫ **ListPlot**: cup〚128〛 is not a list of numbers or pairs of numbers.

⚫ **ListPlot**: cup〚128〛 is not a list of numbers or pairs of numbers.

⚫ **General**: Further output of ListPlot::lpn will be suppressed during this calculation.

The Noise/Scale trade-off

The definition of edge detection is 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 rapid changes in intensity. (Another is a change in texture, which we will discuss in a later lecture.) The biggest intensity changes 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. Consider a generative model for image data f as the convolution of the step with a blur function:
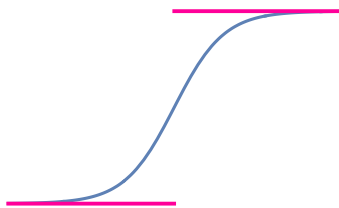
$$f(x) = \int g(x - x') \, blur(x') \, dx' = g*blur$$

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

```
Clear[g];
g[x_] := UnitStep[x];
g1 = Plot[g[x], {x, -2, 2},
   PlotStyle → {Thick, Hue[0.9]}, Axes → None, ImageSize → Small]
```

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])
Show[Plot[edge[x,.3],{x,-2,2},Ticks->None, Axes->None,ImageSize→Small],g1]
```
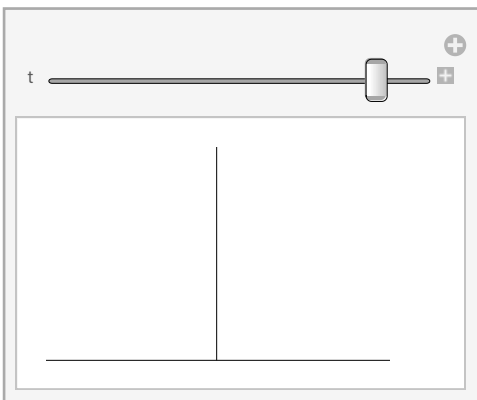
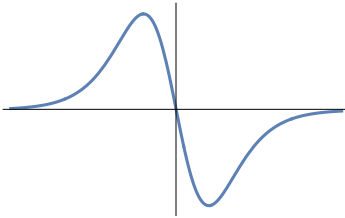(see Appendix for a closed expression solution for gaussian blur of a step function)

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:

```
Clear[dedge, x, t]
```

```
dedge[u_, s_] := D[edge[x, t], x] /. x → u /. t → s
Manipulate[Plot[dedge[u, t], {u, -2, 2},
   Ticks → None, PlotRange → {{-3, 3}, {0, 10}}], {{t, .3}, .01, .33}]
```

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]
```
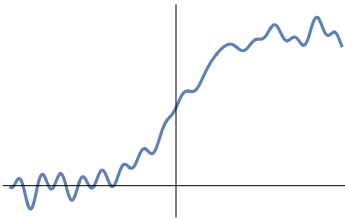

So far so good. But real images rarely have a nice smooth intensity gradation at points that we subjectively would identify as a clean edge. A more realistic generative model for intensity data would be:

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

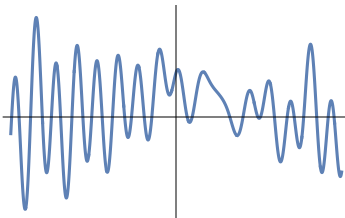Let's add a fixed sample of high-frequency "noise":

```
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]
```
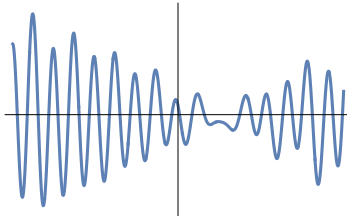

Now, if we take the first derivative, there are all sorts of peaks, and the biggest isn't even where the edge is:

```
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 even worse:

```
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.

In general, the higher the frequency of the noise, the bigger the problem gets. We can see what is going on by taking the nth derivative of the sinusoidal component with frequency parameter f. Here is the 3rd derivative of a component with frequency f:

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

$- f^3 \cos[f\,x]$

The magnitude of the output is proportional to the frequency raised to the power of the derivative. Not good.

A solution: pre-blur using convolution

As in your home work, 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.

The 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

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

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

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

$\nabla^2 G_\sigma$

$G_\sigma \qquad \dfrac{e^{\frac{1}{2}\left(-\frac{x^2}{\sigma} - \frac{y^2}{\sigma}\right)}}{}$

$$\nabla^2 \quad \frac{\partial^2}{\partial x^2} \quad \frac{\partial^2}{\partial y^2}$$

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

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

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

$$\frac{e^{\frac{1}{2}\left(-\frac{x^2}{\sigma}-\frac{y^2}{\sigma}\right)}}{2\pi\sqrt{\sigma^2}}$$

One can take the Laplacian of the Gaussian first, and then convolve this, i.e. the $\nabla^2 G$ kernel, with the image, or one can blur the image first, and then take the second derivatives. The operators, ∇^2 and $*$, are said to be associative:

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

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 $\left(\nabla^2 G_\sigma\right)$ kernel.

One could 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).

But what about the oriented filters in the cortex? An interpretation consistent with the Hubel-Wiesel edge detector interpretation of sine-phase receptive fields in V1, is in terms of 1rst derivatives. Let's see how this works.
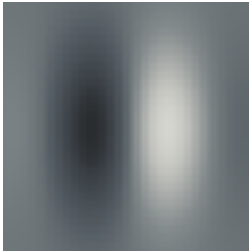
Edge detection using 1rst derivatives

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}];

ArrayPlot[edgefilter,Mesh→False,Frame->False,PlotRange->{-1,1}]
```


As the width of the gaussian envelope decreases, 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

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

```
contcup = ListInterpolation[Transpose[Reverse[cup]],
    {{1, width}, {1, width}}, InterpolationOrder → 4];
```
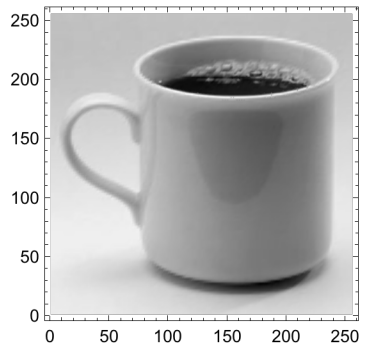
Plot the derivative in the x-direction

```
gd1 = DensityPlot[contcup[x, y],
    {x, 1, width}, {y, 1, width}, Mesh → False, PlotPoints → 128];
gd2 = DensityPlot[Evaluate[D[contcup[x, y], x]], {x, 1, width}, {y, 1, width},
    PlotPoints → width / 2, Mesh → False, Frame → False, PlotRange → {-100, 100}];
```

```
GraphicsRow[{gd1, gd2}]
```


Let's take the derivatives in both 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};
```
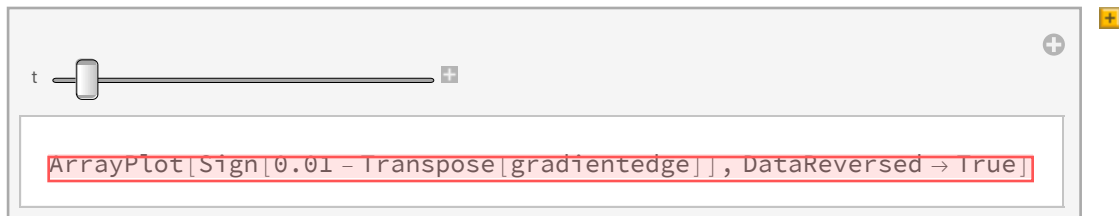
```
gradientedge = Table[N[fcontcup[x, y]], {x, 1, width}, {y, 1, width}];
```

```
ArrayPlot[Transpose[Sqrt[gradientedge]^.5, DataReversed → True] // ImageAdjust
```


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. Further, we haven't specified a blur level, or a criterion for the threshold. We haven't put a measure of confidence on the edges.
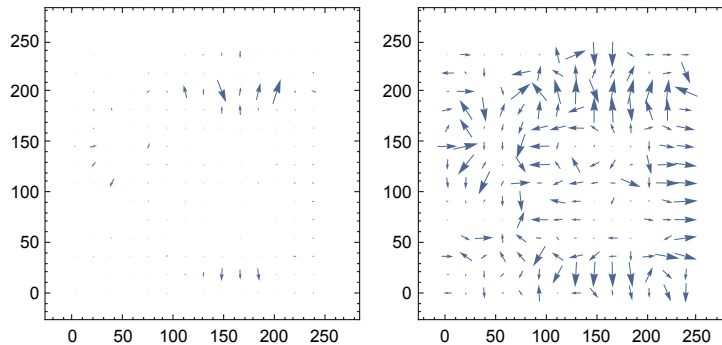
```
Manipulate[ArrayPlot[Sign[t - Transpose[gradientedge]], DataReversed → True],
  {{t, .01}, 0, Max[gradientedge]}]
```



```
ArrayPlot[Sign[0.01 - Transpose[gradientedge]], DataReversed → True]
```

ArrayPlot: Argument Sign[0.01 – Transpose[gradientedge]] at position 1 is not a list of lists.

There is also useful information in the direction of the gradient vectors:

```
gd3 = VectorPlot[{fxcontcup[x, y], fycontcup[x, y]},
   {x, 1, width}, {y, 1, width}, ImageSize → Small];
gd4 = VectorPlot[{fxcontcup[x, y], fycontcup[x, y]} / Sqrt[.0001 + fxcontcup[x, y] ^ 2 +
     fycontcup[x, y] ^ 2], {x, 1, width}, {y, 1, width}, ImageSize → Small];
GraphicsRow[
 {gd3,
  gd4}]
```


Imagine trying to link up points along an edge with the information in the left panel---You get a better idea of how much uncertainty 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. direction 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. This suggests that for illumination-invariant recognition, one should rely more on orientation than contrast magnitude.

▶ 1. Replace the cup with an annulus, and run the above edge detection cells again

```
annulus =
   ImageData[ColorConvert[Graphics[Annulus[], ImageSize → 256], "Grayscale"]];
annulus = .25 * annulus + .375;
```

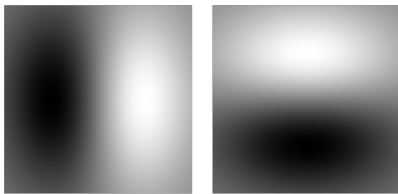## Summing up: Combining a smoothing pre-blur with 1rst derivatives

As with the 2nd derivative zero-crossing detector, the idea is to blur the image, and but then take the first derivatives in the x and y directions, square each and add them up. The x and y components of the gradient of the blur kernel :

```
G[x_, y_, σx_, σy_] := e^(1/2 (- x²/σx - y²/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, ImageSize → Tiny];
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, ImageSize → Tiny];
GraphicsRow[
 {xg,
  yg}]
```


--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:

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

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

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

We'll encounter this idea later when we extend detecting edges in space to detecting edges in space-time in order to make motion measurements.

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 a line--i.e. a pair of edges close together. A line-drawing is an example.

The Appendix and homework show 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". Quadrature pairs will also crop up later in the context of motion detection.

The intuition conveyed by a measure of "contrast energy" is that the response shouldn't care about the sign of a contrast change, i.e. white to black or black to white. See exercise in the Appendix.

Some challenges for edge detection

Problems with interpreting a V1 simple/complex cell as an edge detector

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 several 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. One way of combining information efficiently across scales is to use a Laplacian image pyramid (See supplementary links on class web page). Oriented information can be computed across scales using a steerable pyramid. 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.

Later when we study hierarchical neural networks for object classification, it may not be important to precisely and accurately localize edges.

Segmentation & why edge detection is hard

A key problem is to get from intensity edges to object boundaries.

The problem of texture and background

The above analysis assumed that an edge detection should be the solution to an image-based generative problem: Given intensity f as a function of x, estimate a step-function g(x):
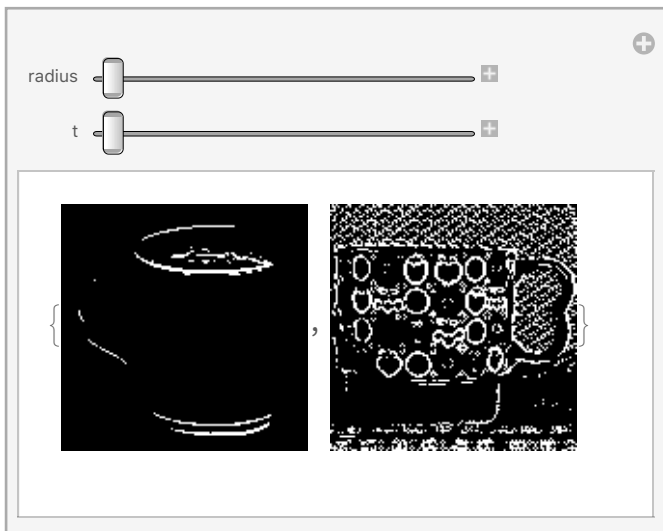
$$f(x) = \int g(x - x') \, blur(x') \, dx' + noise$$

We used the cup image to illustrate how scale and noise (represented by the blur and noise processes)

confound estimates of g(). But the cup image had a fairly uniform figure and background. Consider the more typical case of a patterned cup against a non-uniform background:

```
camo = ImageData[ColorConvert[        , "Grayscale"]];
```

```
camod = downsample[camo, 2];
cupd = downsample[cup, 2];
images = {Image[cupd], Image[camod]};
Manipulate[
 Pane[
  Map[Binarize[ImageAdjust[GradientFilter[#, radius], {.7, .6}], t] &, images],
   {300, 150}], {radius, 1, 10, 1}, {t, 0, 1, .1}, SaveDefinitions → True]
```


From: http://www.flickr.com/photos/96221617@N00/280637989/

The above example illustrates the problem of misleading edges both at the boundary between the object and background, but also between texture elements in the object and its boundary. One needs to take into account texture as well as intensity in determining object boundaries (see Malik et al, 2001).

The problem of edge cause: the same intensity gradient means different things depending on context

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}];
ArrayPlot[picture,Frame->False,Mesh->False,
              PlotRange->{0,1}, AspectRatio->Automatic]
```
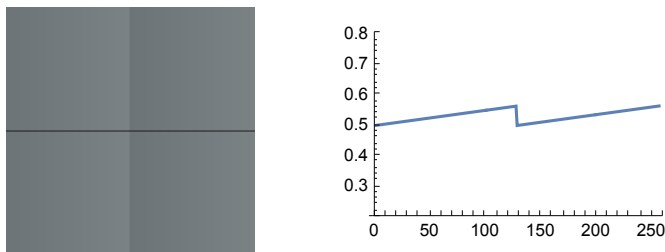
The left half looks lighter than the right half. But, let's plot the intensity across a horizontal line:

```
new =  picture;
new[[128, All]] = 0;
GraphicsRow[{ArrayPlot[new, PlotRange → {0, 1}],
   g0 = ListPlot[ picture[[128]], Joined → 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 about human contrast sensitivity as a function of spatial frequency--in terms of a single-channel model?

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
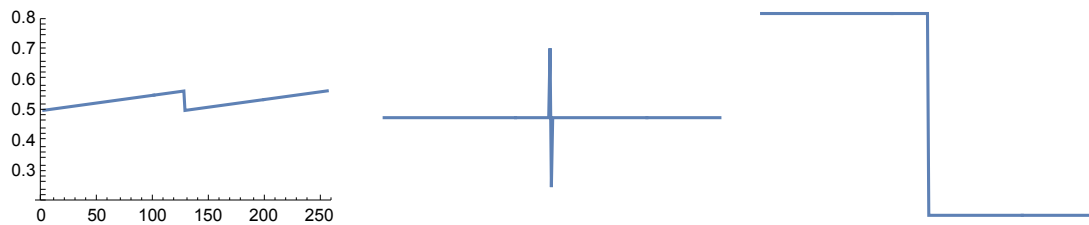
```
filter = {-1, 2, -1};

(*Take the second derivative at each location*)
fspicture = ListConvolve[filter, picture[[128]]];
g1 = ListPlot[fspicture, Joined → True, PlotRange → {-0.1, .1}, Axes → False];

(*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];

g2 = ListPlot[integratefspicture2, Joined → True, Axes → False];
GraphicsRow[{g0, g1, g2}, ImageSize → Large]
```


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];
```
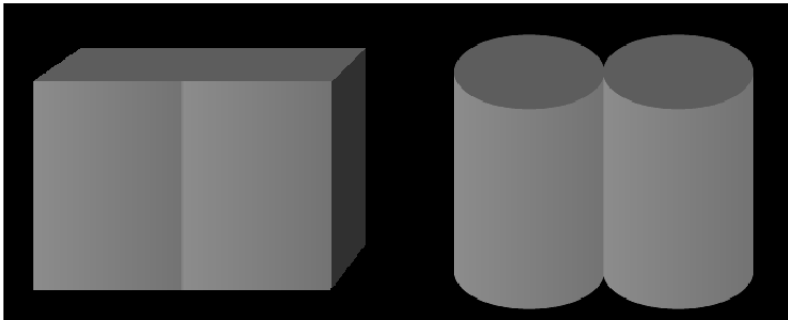
Or one can take just the first derivative, followed by the threshold function
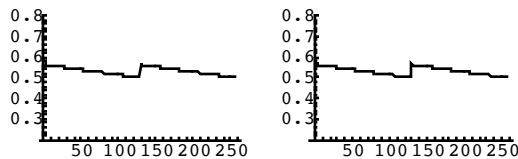
"Two cylinders and no happening"

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. (See Knill and Kersten, 1991).
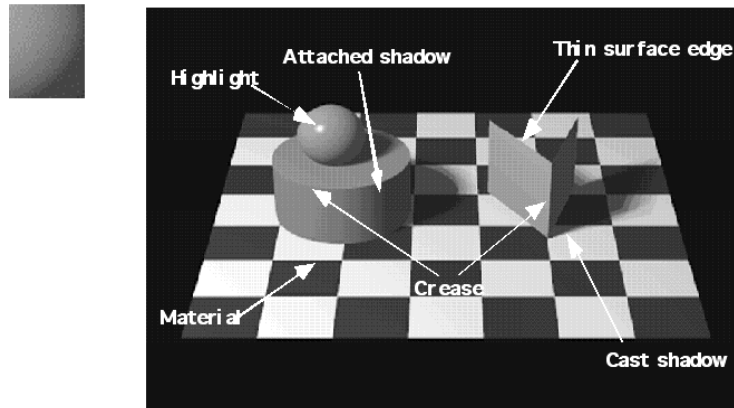
Edge interpretation depends on knowing context

E.g. famous "Mooney image" -- the Dalmation dog.

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. Even when there is little or no noise, local measurements of contrast change say very little about the physical cause of the gradient. And a "smart" visual system takes the causes into account.
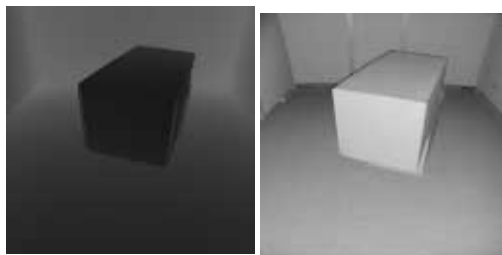
So on the one hand, it still makes sense to interpret lateral inhibitory filters and oriented cortical filters as possible components of an edge detection system, but we have to allow for considerable uncertainty in the significance of their outputs--i.e. a local edge detector typically has a low signal-to-noise ratio for a variety of ways of defining signals, e.g., whether the causes are geometric or photometric.

For tasks such as object recognition, vision places a higher utility on surface and material edges than on other types, such as shadow edges. 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.

Combining signal detection theory with edge detection

Canny (1986).

- 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 measurement is on or off of an edge.

The left panel of the figure below shows "range data", where geometric depth from the camera is represented by graylevel, with dark meaning close, and light meaning far. The right panel shows intensity data. The data in the left can be used to define measures of "geometric ground truth", and one can devise edge detectors based on a signal detection theory analysis of how well the intensity changes on the right predicts geometrical changes on the left. In other words, what edge detector provides the best ROC performance? (See Konishi et al., 2003).
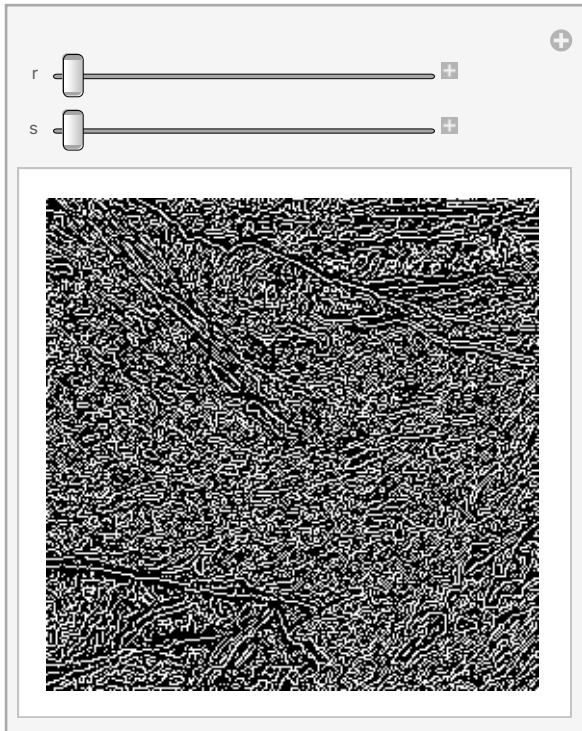

Natural images & segmentation

Where to draw the important contours? Here's the output from *Mathematica*'s built-in EdgeDetect[]

function (its default is Canny edge detection, Canny (1986)). The sliders allow you to select r -- the range of pixels over which to blur, and t --- the threshold to drop edges.

```
Manipulate[EdgeDetect[        , r, s], {r, 1, 4}, {s, 0, 1}]
```


How can one go from the imperfect output of a low-level edge detector to a clean "line-drawing" representing the true boundaries of an object? Grouping local measurements that are similar is one step. This can be at the edge or region level, i.e. grouping local edge measurements into longer lines and grouping features within a region. Grouping processes are sometimes called "intermediate-level" because they don't rely on specific knowledge about particular object classes, but just on how contours typically go, or how features are typically related (i.e. similar orientations, colors, motion direction,...). Perceptual grouping or similarity principles were studied by Gestalt psychologists in the early 1900s.

In addition, the visual system seems able to solve, i.e. invert, a generative model that is more scene-based than image-based--it cares about the type of edge, and the types of objects and arrangements likely to be encountered. This will be the focus of the next few lectures.
Even with intermediate-level grouping, and edge selection based on scene-based filtering, finding the boundaries of objects requires more specific knowledge or memory about the possible shapes of previously seen objects. The dalmation dog illusion, above, illustrates the human visual system brings experience and high-level knowledge to bear on the problem of segmenting an object.

Finding useful segmentations is an image parsing problem. It is a non-trivial computer vision problem. For 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

Given the problems of edge detection in the absence of context, it seems more appealing to interpret the spatial filtering properties of V1 as efficient encoding. However, if one thinks of V1 oriented cell responses as representing tentative "edges", perhaps with a representation of confidence, then one can begin to understand how high-level "models" may be used to select the edges that belong, and reject those that don't (Yuille and Kersten, 2006). How these tentative edges might be extracted from both intensity and textural transitions, and how high-level information might constrain these remains a challenging area of research.

Other than edges: Curvature, interest points & saliency

Human vision does more than seek out edges as a basis for object detection and recognition. One basic function is to direct the eyes to so-called salient points in an image. In 1954, Fred Attneave pointed out that people's eye fixations were attracted to some features of an image more than others. In "Attneave's cat", shown below, eye movements tend to go to points of high curvature. The Appendix shows one way to extend derivative operators to amplify corners using the Hessian.

There is a line of research to model first fixations in natural images (Itti & Koch, 2001; Torralba et al., 2006, Zhang et al., 2008). The idea is that first-fixations may be driven by fairly low-level image properties. The key idea is that eye movements go to regions that have low probability (high information) given either the current image context, or a generic natural image context. These points are said to have high *saliency*.

Subsequent fixations are more difficult to model because they are dominated by the task the person is trying to accomplish, rather than by bottom-up novelty or "surprise". See: Tatler et al., 2011.

Zhang et al (2012) provide evidence and argument that primary visual cortex (V1) is provides the basis for a spatial saliency map.
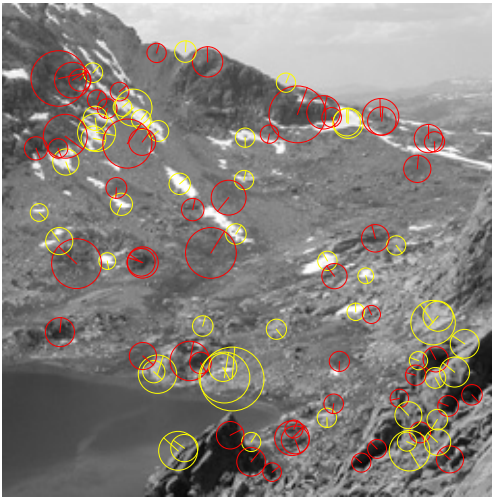
Recognition models in computer vision often rely on the detection of salient or "interest" points, using detectors chosen to be robust over local variations in position or lighting in complex natural images. (Later we'll talk about the problems of recognition). One popular method is to use SIFT operators (Lowe, 2004), http://en.wikipedia.org/wiki/Scale-invariant_feature_transform, or HOG filters -- "Histograms of Oriented Gradients".

Mathematica has a built-in function, ImageKeyPoints[] based on a related method (SURF; Bay et al.,

2006).

```
img =      ;
```

```
points = ImageKeypoints[img,
    {"Position", "Scale", "Orientation", "ContrastSign"}, MaxFeatures → 100];
Show[img, Graphics[Table[{{If[p[[4]] == 1, Yellow, Red],
      Circle[p[[1]], p[[2]] * 2.5], Line[
        {p[[1]], p[[1]] + p[[2]] * 2.5 * {Cos[p[[3]]], Sin[p[[3]]]}}]}}, {p, points}]]]]
```


► 2. *Try replacing the above mountain image with Attneave's cat*

```
img2 =             ;
```

► 3. *Try using ImageCorners[] and HighlightImage[] applied to Attneave's cat: HighlightImage[*
    img,ImageCorners[img]]

► 4. *Compare the results with ImageSaliencyFilter[] applied to the cat: ImageSaliencyFilter[img]*
    //ImageAdjust

For more information on saliency benchmarks, see: http://saliency.mit.edu

Next time

Beyond V1: Extra-striate visual cortex

Surfaces from images

Scene-based modeling

Appendices

Symbolic convolution solution for 1D gaussian blur of a step function

```
blur = Exp[-(x / s)^2 / 2] / (Sqrt[2 Pi] s);
Convolve[blur, UnitStep[x], x, y]
```

$$\frac{\frac{1}{\sqrt{\frac{1}{s^2}}} + s \, Erf\left[\frac{y}{\sqrt{2} \, s}\right]}{2 \, s}$$

$$Manipulate\left[Plot\left[\frac{\frac{1}{\sqrt{\frac{1}{s^2}}} + s \, Erf\left[\frac{y}{\sqrt{2} \, s}\right]}{2 \, s}, \{y, -2, 2\}\right], \{s, .1, 4\}\right];$$

The Hessian, "Interest operators", and saliency.

```
SetOptions[ArrayPlot, ColorFunction → "GrayTones",
   DataReversed → True, Frame → False, AspectRatio → Automatic,
   Mesh → False, PixelConstrained → True, ImageSize → Small];
```

The input 64x64 image: face

```
width = Dimensions[face]〚1〛; size = width;
hsize = width / 2;
hfwidth = hsize;
height = Dimensions[face]〚2〛;
face;
gface = ArrayPlot[face];
```

Computing both the first and second derivatives of image intensity can be thought of as filters to pick out regions of an image that have "salient", i.e. rapid, intensity changes. A natural extension is to look

at all four combinations of second derivatives.

Calculating the Hessian of an image using function interpolation.

The Hessian of a function f, H(f(x_1, x_2, ..., x_n)) is given by:

$$
H(f) = \begin{bmatrix}
\frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \, \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \, \partial x_n} \\[2mm]
\frac{\partial^2 f}{\partial x_2 \, \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \, \partial x_n} \\[2mm]
\vdots & \vdots & \ddots & \vdots \\[2mm]
\frac{\partial^2 f}{\partial x_n \, \partial x_1} & \frac{\partial^2 f}{\partial x_n \, \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2}
\end{bmatrix}
$$

For our purposes, {x_1,x_2} = {x,y}, so the Hessian of an image returns a 2x2 matrix at each point (x,y) that represents the four combinations of second derivatives in the x and y directions. The determinant of each of the 2x2 matrices provides a scalar which is a measure of the "area" of each 2x2 matrix. The area can be used as a rough measure of saliency or local "interest", which takes into account rates of change in x and y, for example at "corners".

Let filterface = f, where we' ve blurred out face a little to reduce quantization artifacts :

```
kernel = N[{{1, 1, 1}, {1, 1, 1}, {1, 1, 1}}];
filterface = ListConvolve[kernel, face];

faceFunction = ListInterpolation[Transpose[filterface], {{-1, 1}, {-1, 1}}];

hessian[x_, y_] := Evaluate[D[faceFunction[x, y], {{x, y}, 2}]];
```

Calculate and plot each of the components of the Hessian at each image point

```
dxxtemp =
   Table[hessian[x, y][[1, 2]], {x, -1, 1, 0.005}, {y, -1, 1, 0.005}];

GraphicsRow[{gface, ArrayPlot[Transpose[dxxtemp]], Histogram[Flatten[dxxtemp]]}]
```
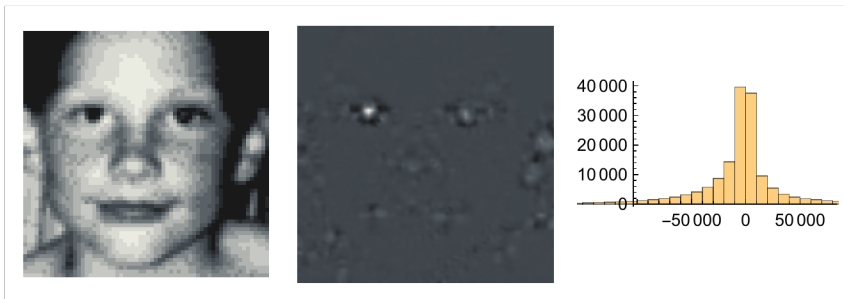

The determinant of the Hessian provides a simple measure of "salience". Better models take into account how unexpected local features are relative to the background or likely backgrounds (See Torralba et al., 2006, Itti & Koch, 2001, and Zhang et al., 2008.) These models have been applied to predicting human visual eye movements.)

```
htemp = Table[Det[hessian[x, y]], {x, -1, 1, 0.005}, {y, -1, 1, 0.005}];
```

```
GraphicsRow[{gface, ArrayPlot[Transpose[htemp]], Histogram[Flatten[htemp]]}]
```


For current computer vision work on local feature detection, see papers by Lowe in the references, and http://en.wikipedia.org/wiki/Scale-invariant_feature_transform

Also see: http://en.wikipedia.org/wiki/Interest_point_detection

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.

▶ 5. Use the model to show that the response to a white dot on a black background is the same as to black dot on a white background.

▶ 6. Calculate the response to a vertical edge that goes from white to black, and compare it to the response of an edge that goes from black to white.

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}];
ArrayPlot[picture,Frame->False,Mesh->False,
                PlotRange->{0,1}, AspectRatio->Automatic];

picture2

picture2
```

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 = ArrayPlot[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 = ArrayPlot[cfilter, Mesh → False, PlotRange → {-1, 1}, Frame → False];
```

Apply odd (sine) filter

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

Apply even (cosine) filter

```
fcpicture = ListConvolve[cfilter, picture];
ArrayPlot[fcpicture, Mesh → False];
```
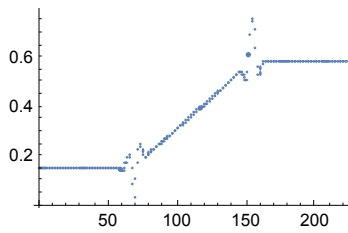
Look for peaks in local contrast energy

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

ArrayPlot[ss, Mesh → False]
```

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


Two cylinders, no illusion

```
twoc = ImageData[▨▨▨];
```
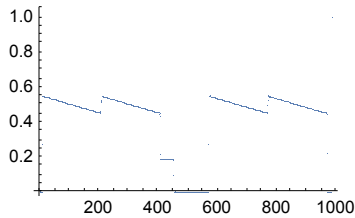
```
reds = twoc[[All, All, 2]];
Dimensions[reds]
```

{376, 984}

```
ArrayPlot[reds, DataReversed → False]
ListPlot[reds[[Dimensions[reds][[1]] / 2]]]
```


References

Attneave, F. (1954). Some informational aspects of visual perception. Psychol Rev, 61(3), 183-193.

Bay, H., Tuytelaars, T., Gool, L.V., "SURF: Speeded Up Robust Features", Proceedings of the ninth European Conference on Computer Vision, May 2006.

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.

Itti, L., & Koch, C. (2001). Computational modelling of visual attention. Nat Rev Neurosci, 2(3), 194-203.

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.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 60(2), 91–110.

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.

Tatler, B. W., Hayhoe, M. M., Land, M. F., & Ballard, D. H. (2011). Eye guidance in natural vision: Reinterpreting salience. Journal of Vision, 11(5), 5–5. http://doi.org/10.1167/11.5.5

Torralba, A., Oliva, A., Castelhano, M. S., & Henderson, J. M. (2006). Contextual guidance of eye movements and attention in real-world scenes: the role of global features in object search. Psychol Rev, 113(4), 766-786.

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.

Yuille, A., & Kersten, D. (2006). Vision as Bayesian inference: analysis by synthesis? Trends Cogn Sci, 10(7), 301-308. Biological Cybernetics, 61, 183-194.

Zhang, L., Tong, M. H., Marks, T. K., Shan, H., & Cottrell, G. W. (2008). SUN: A Bayesian framework for saliency using natural statistics. J Vis, 8(7), 32 31-20.

Zhang, X., Zhaoping, L., Zhou, T., & Fang, F. (2012). Neural Activities in V1 Create a Bottom-Up Saliency Map. Neuron, 73(1), 183–192. http://doi.org/10.1016/j.neuron.2011.10.035