Computational Vision
U. Minn. Psy 5036

Assignment #4
Curvature-based interest operators, more spatial frequency filtering, and illusions

Goals:
1) use symbolic processing to calculate a Hessian interest operator
2) explore the information in the amplitude and phase spectra in natural images
3) practice 2D graphics and animation programming and thinking about possible perception experiments for final projects

If you use ArrayPlot instead of Image[], these options may be useful:

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

▸ 1. The Hessian, "interest operators", and saliency

```
icats = ColorConvert[
```

```
, "Grayscale"];
```

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:

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

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 at each point 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".

```
cats = ImageData[icats];
{width, height} = Dimensions[cats]

kernel = N[{{1, 1, 1}, {1, 1, 1}, {1, 1, 1}}];
filtercats = ListConvolve[kernel, cats];
catsFunction = ListInterpolation[filtercats, {{-1, 1}, {-1, 1}}];
```

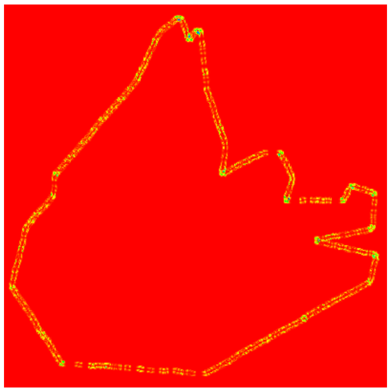
```
{256, 256}
```

Write a function to calculate the hessian which takes catsFunction as the argument. Depending on you you do it, you may need to force Evaluation with the Evaluation[] function.

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 initial fixations immediately after an image is presented.

Calculate the Absolute value of the Determinant of the Hessian at each point. Use GraphicsRow[{}] to show the original cats image, a representation of the “saliency” as determined by the determinant of the Hessian.

For example, your output could look something like below. But feel free to improve on the visualization.



For 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

▶ 2. Importance of the phase spectrum in visual recognition

In the early days of pattern vision research, before it was possible to easily calculate fourier transforms on images, it was not often appreciated that the phase spectrum carries the significant information for recognition. This exercise demonstrates the relative roles of the spectra.

```
img256 =

  ImageResize[ImageAdjust@Image[ColorConvert[
```

```
, "Grayscale"]], 256];
```

```
astro = ImageData[img256];
astroft = Fourier[astro];
astrospectrum = Chop[Abs[astroft]];
astrophase = Chop[Arg[astroft]];
{width,height}=Dimensions[astro];
```

Reconstruct the original image using astrospectrum and astrophase.

```
astrowithcorrectphase =
 Chop[?];
   Image[astrowithcorrectphase]
```

It should look like this:

Make some uniformly distributed white noise of the same size as the original “astro” image. Then extract the amplitude and phase spectra.

randomphase = Chop[Arg[?]];
randomspectrum = Chop[Abs[?]];

By taking the inverse Fourier transform, show what an image looks like with the original amplitude spectrum of astro, but with random phases, i.e. randomphase.

ImageAdjust@Image[?]

Again taking the inverse Fourier transform, show what “astro” looks like if we substitute the above white noise spectrum (randomspectrum) for the original amplitude spectrum from astro.

ImageAdjust@Image[?]

▶ 3. Random Fractals

Random fractals have been proposed as statistical models for the (“1/f”) amplitude spectra of natural images. The function below, **fractalfilter2[D_,size_]**, is one way of generating the envelope for a fractal amplitude spectrum. You will need to multiply this envelope by a randomly generated white noise sample to generate the fractal amplitude spectrum. **fractalfilter2[D_,size_]** multiplies the noise spectra so that low frequencies are systematically suppressed relative to the higher ones. The log of the Fractalfilter2[D_,size_] values along, for example, the horizontal direction should drop off linearly with spatial frequency, indexed by i and j below.

The value of D, below should be between 3 and 4. We first make the filter centered in the middle, and then adjust it so that it is symmetric with respect to the four corners.

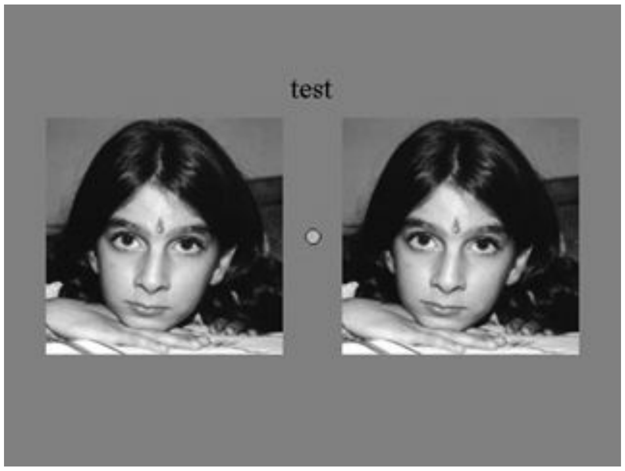
```
fractalfilter2[D_,size_] :=
Module[ {q,i,j,mat},
    q = 4 - D;
    mat = Table[If[(i != 0 || j!= 0),
        1/(i^2 + j^2)^q, 1/(2)^(q)],
    {i,-size/2,(size/2) - 1},{j,-size/2,(size/2) - 1}];
    mat = RotateRight[Transpose[
        RotateRight[mat,size/2]],size/2];
    Return[mat];
    ];
```

Generate a fractal noise spectrum with D = 3.2, and combine this amplitude spectrum with original astro phase spectrum, and show the resulting image.

ImageAdjust@Image[?]

▸ 4. Write a program to illustrate adaptation to image blur. (See "bluradaptiondemo" on class web page.)

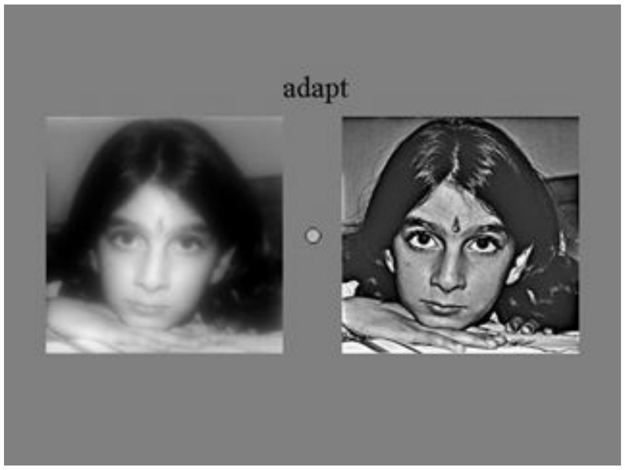
Here is an example taken from Webster et al. (2002). Use the above astro image to make a test image that has the identical source image on the left and right side of a fixation, similar to that shown below.



Now make the "adapt" image, where the left side is a low-pass filtered version of the original source image, and the right-side is the complementary high-pass filtered version.

Hint: the complementary high-pass version is simply the original image minus the blurred one.

The "adapt" image should be similar to the one shown below.



Now make an animation in which you: a) show the test image for about 3 seconds, b) followed by the adapt image for 20 seconds, c) followed by the test image again but now for at least three seconds. You can do this all in one *Mathematica* program.

You may find it convenient to use ArrayPlot[] with the options:

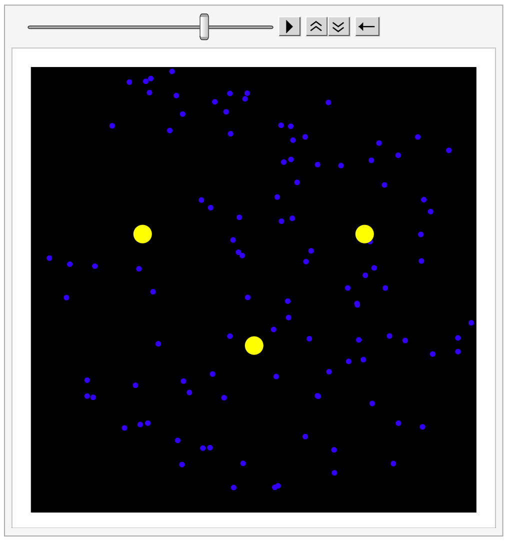
SetOptions[ArrayPlot,ColorFunction->”GrayTones”, Frame->False,AspectRatio->Automatic,Mesh-

>False,PixelConstrained->{1,1}];

You may find an alternative method, but one possibility is to composite the images using Graphics[] with Inset[].

And you can make the fixation mark with something like this: gfixation=Graphics[Disk[{0,0}, .06],PlotRange->{{-1/2,1/2},{-1,1}}];

▶ 5. Write a program that generates an animation to illustrate motion-induced blindness. (See "motion-induced-blindness demo" on class syllabus web page.)



The rationale for this exercise is to 1) practice graphical tools to make simple dynamic perceptual demonstrations; 2) experience for yourself a relatively recently discovered, and rather dramatic, perceptual illusion that is still not well understood.

You can program the animation in any way you would like. However if you would like some hints, it is possible to make the animation in four lines of code with the following functions:

Show[],Graphics[], Disk[], and ListAnimate[].

Some complexity arises in figuring out the options, so here are some more hints:

```
diskg=Show[Graphics[{{RGBColor[?], Disk[{?},?]}, {RGBColor[?,?,?], Disk[{?},?]},
{RGBColor[?,?,?],Disk[{?},?]}}, AspectRatio→Automatic, Axes→False, Background→GrayLevel[0.0]]]
```

```
gg=Graphics[disks=Table[{EdgeForm[?],Hue?],Disk[RandomReal[?,{?}],?]},{120}],Background→Black];
```

```
glist=Table[Show[Graphics[Rotate[?,theta],AspectRatio→1,PlotRange→{{0,8},{0,8}},Background→
Black],?],{theta,0,16*Pi,2*Pi/24}];
```

ListAnimate[?,10]

Optional: If you’d like to save your movie, you can use: Export[“?.avi”,glist].

You fill in the ?s.