Computational Vision
U. Minn. Psy 5036
Daniel Kersten
Lecture 1

The course

Syllabus is online at courses.kersten.org

Prerequisites

Readings and lecture material

a "lab course"

Grading

General class goals are to learn

about your own visual system

an active interdisciplinary approach to a complex problem

statistical techniques for making good guesses from lots of weak information

how knowledge of vision can be applied to everyday problems

a high-level programming language

to improve your writing

Introduction to the problem of vision

Understanding visual perception is an important problem in psychology

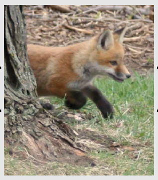
One of the great mysteries of psychology is how the human visual system determines what and where objects are just by looking. This is the problem of vision. The perception of what is out there in the world is accomplished continually, instantaneously and usually without conscious thought. The very effortlessness of perception disguises the underlying difficulty of the problem. Vision is important because it is one of the principle routes to our acquisition of knowledge, as well as guide to its utilization.

It takes just one quick glance at the picture in the Figure below to recognize the fox, a tree trunk, some grass and background twigs. But that is just the beginning of what vision enables us to do with this picture. With a few more glances, one can see, as one person describes it, that:

“It is a baby fox emerging from behind the base of a tree not too far from the viewer. It is heading right, high-stepping through short grass, and probably moving rather quickly. Its body fur is fluffy, reddish-brown, relatively light in color, but with some variation. It has darker colored front legs and a dark patch above the mouth. Most of the body hairs flow from front to back...and what a cute smile, like a dolphin.”

The ability to generate an unbounded set of descriptions from a virtually limitless number of images illustrates the extraordinary versatility of human visual processing

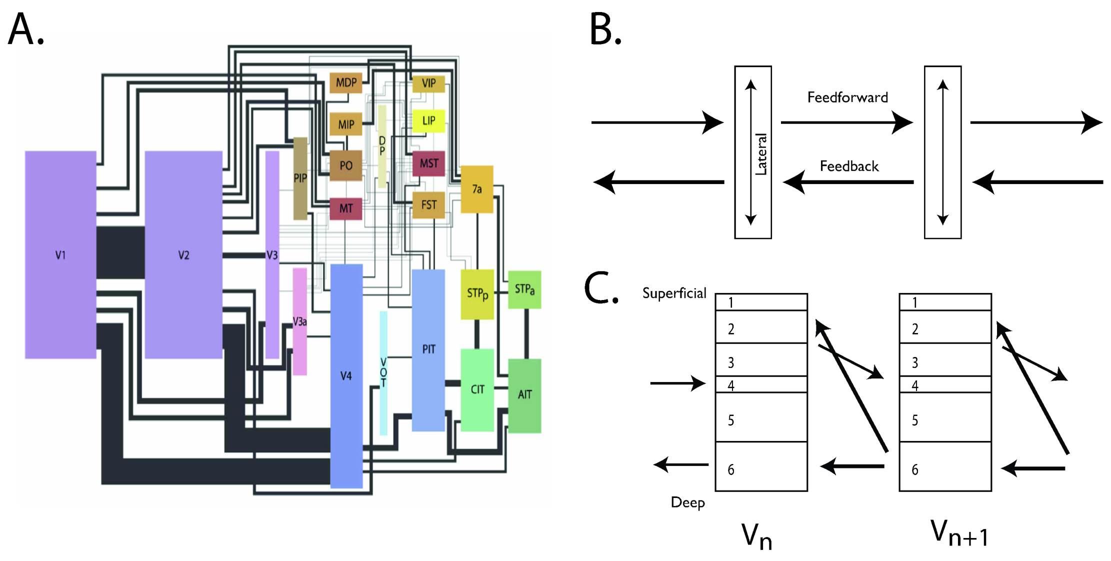
In[197]:=

```
Image[
```

```
]
```

Understanding vision is an important aspect of brain science

Visual neuroscientists currently estimate that 40 to 50% of human visual cortex (your gray-matter) is closely involved in visual processing. The general structure of cortical layers and pattern of inter-connectivity is similar across the neocortex. Thus the hope that if we can understand visual computations in the cortex, this knowledge may generalize to other cognitive domains.

Figure, panel A, below shows a subset of the visual areas of the monkey, together with their interconnections. The areas of the rectangles represents the relative proportions of neurons in each, and the thickness of the lines the proportion of feedforward neural fibers connecting them.



With about 10 million retinal receptors, the human retina makes on the order of 10 to 100 million measurements per second. These measurements are processed by about a billion plus cortical neurons. Vision is a complex process requiring mathematical modeling and programming tools...

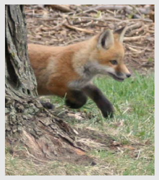
Vision is a challenging mathematical and computational problem

The problem of vision is not only important from the point of view of understanding the brain, but it is also formally challenging. Formal solutions have implications for robotics and artificial intelligence. As such vision is an active area of research for computer scientists, engineers, and mathematicians as well.

■ Why is it a hard problem?

A fundamental reason is that are no locally recognizable features in a typical "natural" image (the problem of local ambiguity)

```
In[198]:=
Manipulate[ImageApply[UnitStep[#] &,
```



```
,
  Masking → Graphics[Disk[Scaled[{x, y}], r]]], {x, 0, 1}, {y, 0, 1},
 {{r, .1}, .01, .5}]
```

No general-purpose algorithm can accurately locate, and categorize the objects in the picture. Or determine the material...skin, hair, cloth... of the various regions. Or determine the object's boundaries with the precision and accuracy of a human.

■ From images to actions, objects: Preview of the formal problem

Formally, we want to understand how to get useful actions **A**, from image measurements **I**:

I -> A

Think of **I** and **A** as multivalued descriptions (e.g. vectors) of image measurements and action parameters. Actions are general--they could be motor actions such as reach and grasp, or descriptions of the fox above. To get from **I** to **A**, vision usually requires information about objects, surfaces, and scenes and their relationships to the viewer. Properties of objects and their relationships will be called **scene** attributes, represented by a vector **S**.

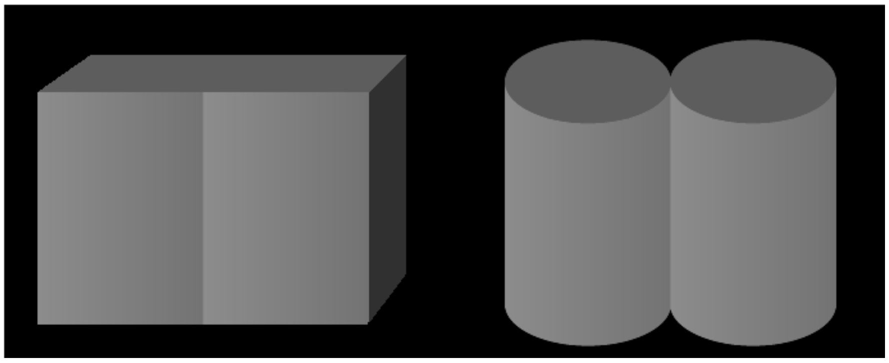
Consider the question: How can one estimate parameters of objects--their colors, shapes, materials, their relationship to other objects, to the viewer, to the viewer's hands, etc..--all from a glance? This is often referred to as the problem of *image understanding*, to emphasize that vision is a problem of perceptual inference. That is, given an image which is just a description of the light intensities at each point in space, and time (e.g. video camera or the sensors at the back of your eye), how can one infer the properties of the scene that caused the image?

In general, we'll represent the **image** by an array of intensities, **I**(x,y,t), varying in space and/or time. (More generally, **I** could be some derived image measurements, like the location and orientation of local edge segments). Actions **A**, such as parameters required for grasping an object, or saying "that's a dog", require information about scene attributes **S**. Thus sometimes, we'll think of **A** as a function of **S**: **A(S)**. But many times, we'll study problems in which **A ~ S,** i.e. in which an action parameter is the same as the scene property.

Then the problem is:

I -> S

An example is S = depth of an object from the viewer, or S = pigment property of the object. For example, the pattern of intensities on the left and right rectangles of the left block are the same. Further, the right pair of cylinders as the same pattern in the horizontal direction. Yet, the left rectangle appears darker than the right. But not so much for the cylinders. Why? The answer is that your visual system is trying to estimate the pigment level of the presumed surface being viewed--NOT the intensity. (example from: Knill & Kersten, 1991)



The problem of computing scene parameters from images is an example of an *inverse problem*. It is called this, because the goal is to estimate cause, such as change in pigment level, from data--the image measurements. Computing image intensities (the data) from the causes is called the forward problem, and involves specifying a “generative model”.

■ The generative model

Visual understanding is only possible because images are a structured function of what is out there, i.e. images are functions of scenes: I = function(S). This is called a forward or generative model.

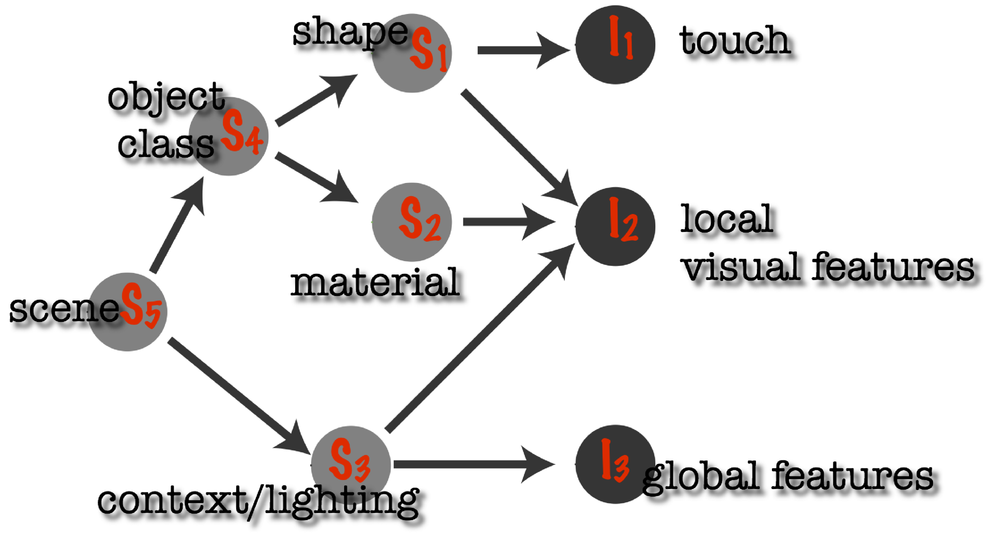
3D computer graphics is a good example of a forward problem involving generative models. Sometimes called of "forward optics" -- how to go from a description of the scene environment to the image:

S -> I

In other words, a function that describes what "out there in the world" (object shape, lighting, etc.) causes the image intensities observed. This is also the problem of creating "virtual reality".

The causal structure leading to the data (image) is well-defined, hence generative model.

The figure below shows how one might specify a set of "knobs" **S** when synthesizing or generating stimuli **I** (which could include image and haptic information) using scene descriptions:



We'll spend some time learning both about methods for generative modeling of images, as well as techniques for solving vision problems by inverse inference.

Generative models are important because: 1) they are the causes of the regularities that vision must exploit; 2) they can provide quantitative models for experimenters to manipulate visual stimuli to test models; 3) generative models may indeed by used explicitly by the brain--a topic we return to at the end of the course.

■ Vision as (statistical) inference

So to sum up so far, the formal problem of vision is:

1. An image **I** has regularities caused by an underlying structure that has useful components **S**:

I = function(**S**)

2. The visual system has various goals (make decisions, estimates, descriptions, actions...) based on components in the underyling structure (**S**). I.e.

to go from **I** to a description (or decision, estimate, etc.) **S** or more generally, an action **A.** We'll put a prime on **S**, i.e. **S'** to distinguish our estimate of a scene attribute from its actual value **S**.

I -> S'

This is the sense in which vision can be viewed as an inverse inference problem--inferring useful causes from image data.

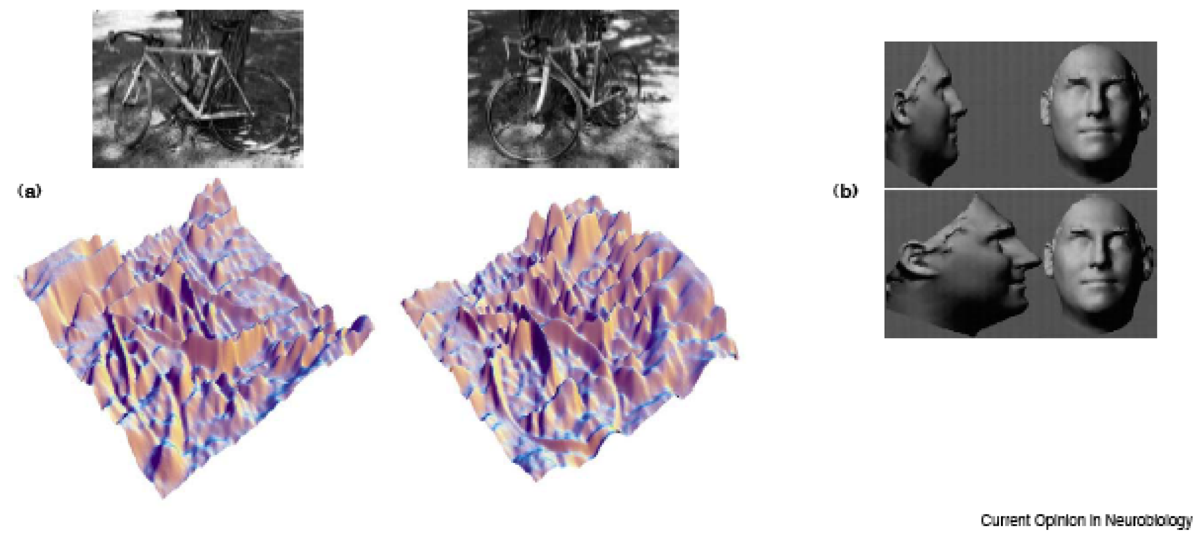
Distinguishing our estimate from the true value is important, because any estimator, including our own visual systems will make mistakes--we don't always see the correct depth. Further, we don't always see the same thing given the same image. Later we'll look at visual illusions that illustrate these points. We will see that the theory of statistical inference provides a natural framework to model under-constrained problems.

Over a century ago, Hermann Helmholtz described perception as "unconscious inference". Another way of expressing this is to view perception as a process that makes good guesses from indirect and ambiguous image input. As we go on, we will justify and amplify on the Helmholtz definition of perception.

More on the problem of ambiguity for objects

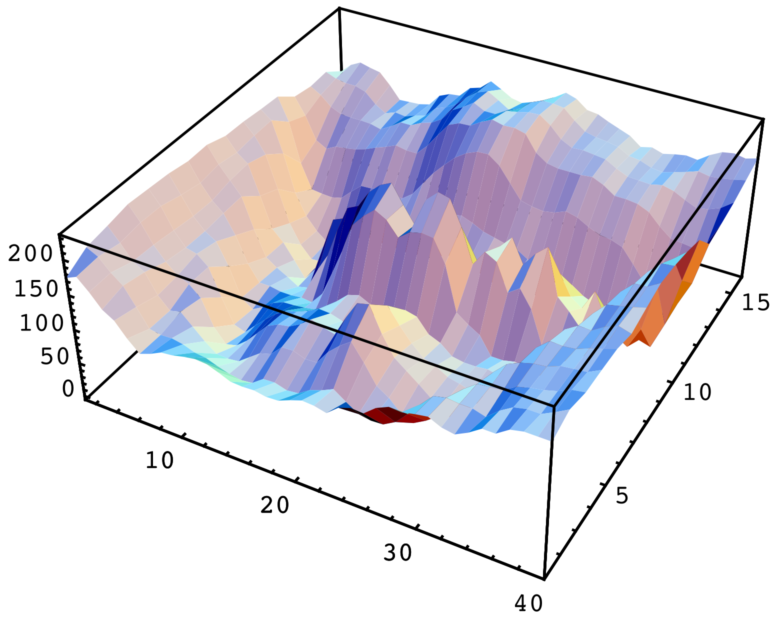
Let's take a closer look to see reasons why vision can be challenging problem from a formal point of view.

Not only can the same object can give rise to different images (panel a below), but different objects can give rise to the same image (panel b below).



■ An example: a "mystery image"

The following section has a "mystery" image **I**, which is just a 2D list (matrix) of light intensity values. Let's make a plot that represents intensity as height:



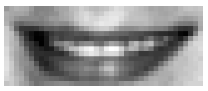
This is a plot of **I** as a function of position, (x,y).

Suppose our goal is to estimate depth at each position, i.e. obtain **S'(x,y)** from **I**. One idea is to assume that **I** is proportional to depth, **S** at each location. Although naive, it isn't a terrible idea--there is some correlation between intensity and depth. Given this assumption, we'd conclude that the high middle ridge is closer than the bottom ridge. But as we will see in a moment, the dominant middle ridge that we see here does not correspond to near depths.

Consider another perceptual inference goal. What if we want to estimate the surface color (pigmentation or "paint") of the image? Let **S''** be the surface color, where **S''** is big for white surfaces, and low for black surfaces. Now it seems even more plausible that **I** would correlate very well with **S''.** But how well?

Let's represent the mystery image in a form where your own visual system can judge...

■ Intensity representation of same data in mystery image:



First, note that the high intensity areas of the teeth are poor predictors of "nearness" in depth. They seem to be better measures of "whiteness". But what about the highlight on the lower lip? This produces a bump in the surface height plot above, but this a highlight due to the glossiness of the lip, and the pigmentation of the actual lip surface is not any lighter than other areas of the lip.

Information about shape and pigment is ambiguous, and the two are confounded in the simple image intensity measurements.

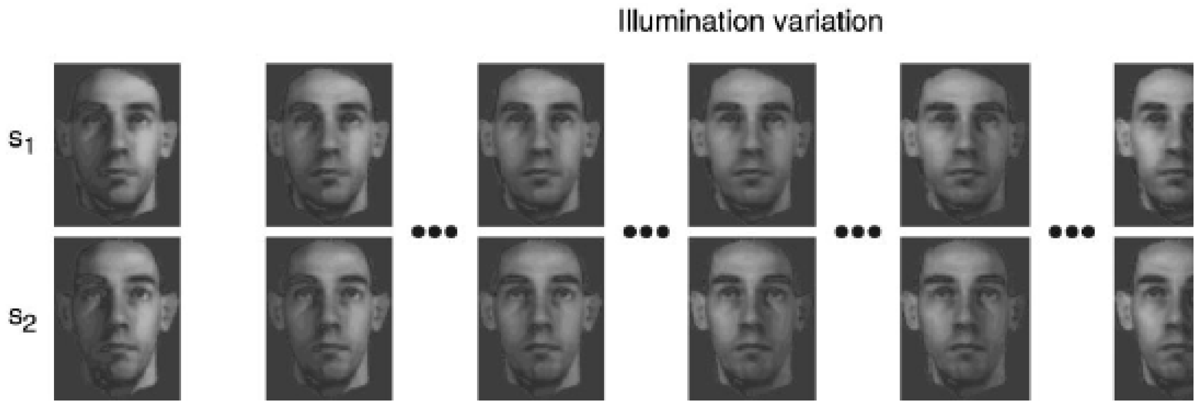
The problem of task

An important recurring theme that will appear throughout this course is the importance of carefully characterizing the task of the person or "agent". Some kinds of scene or object information are more important to estimate the other kinds depending on the task of the agent.

Here's example of two tasks: Face identity vs. illumination direction. In the first task, given an image, decide whether it belongs to the George (s1) or Jim (s2).

Alternatively, there could be another task: given the same image, decide whether it is illuminated from the left or right.

The figure below illustrates the kinds of image variation that can result from different facial shapes (just two in this case), and different illumination conditions:



There are HUGE objective changes in the image going from left to right ("variability in illumination") that human perceptual judgments give very little weight to. The identity of a face is more important than which direction the light is coming from.

Where does the field of computational vision stand today?

Scientists, despite several decades of research have yet to produce a machine that can solve the general recognition problem: identify objects in natural images from arbitrary viewpoints, illumination conditions and in arbitrary contexts. As pointed out above, a central mathematical problem is the multiplicity of possible scene or object configurations that could have caused a particular set of image measurements. Richard Feynman compared the problem of vision to deciding what jumped into a swimming pool just by measuring the bobbing water height as a function of time using a ruler in the corner of the pool. There are lots of ways of in which a given pattern of water heights could arise. Further, as with water height, image measurements are very indirectly related to useful scene information--like was it a beach ball or infant that fell in? An understanding of image formation and optics does not sufficiently constrain the number of possible scene descriptions, **S** that could have given rise to any one image, **I**. This is one of the defining characteristics of inverse problems in general--the data underconstrain the solution.

One of the major contributions of computer vision has been to define the mathematical problems of vision and to show that these can be quite difficult to solve. Historically, the problem of chess was considered a prototypical problem for Artificial Intelligence. Today we have machines that can beat most of us at chess. A lot of their power comes from high speed brute force search, likely quite different from the brain processes of the grand masters. Nevertheless, they can beat us, whereas there is no current system that can pick out the chess pieces from the box (because of variability over viewpoint, lighting, material, and style), and set them up on the board in the right places.

In this course we will study how the visual system deals with variability, such as due to over illumination, viewpoint and material. In addition to the problem that different illumination conditions (e.g. light source coming from above left or above right), and different viewpoints produce different images of the same object, vision has to cope with occlusion of one piece by another. Like fonts, different chess sets have different styles. To some extent style variation can be modeled in terms of geometric variation. But chess piece styles can be determined by symbols having to do with the formation of concepts.

To further sober (and challenge you), the remarkable limitation of our understanding of visual inference is underscored by the fact that there are no machine systems that can solve the patently simple problem of deciding whether a surface has a light or dark pigment under general illumination. I.e., given a white or black chess piece, what color is it? The problem is that a black piece in bright light can have the same average intensity as a white piece in dim light. This is a problem that we will return to later in the context of human material and lightness perception.

On the positive side, there has been considerable theoretical and empirical progress in understand the problems of vision, how to solve them, and how the brain enables us to see with such remarkable competence. Hopefully this course will give you a useful and exciting introduction to the field.

Computational vision: combining disciplines

Combining approaches from psychology, neuroscience, and computation

Vision is a part of cognitive science -- an interdisciplinary effort to understand the nature of knowledge, its acquisition, storage and utilization. It is also part of Cognitive Neuroscience—the study of the relationship between brain and cognitive behavior. I'd like to spend some time motivating the importance of an interdisciplinary study of vision.

I think some motivation may be required here because of the nature of the course. In this course, we will study vision from a computational point of view. The topics should be exciting because the course involves integrating knowledge across disciplines. But it can be frustrating because although it involves some math and computation, it isn't like most quantitative disciplines that have a structured sequence, and you will have to revisit content you've studied before, fill in gaps in your knowledge, and learn new concepts to make the interdisciplinary picture come into focus.

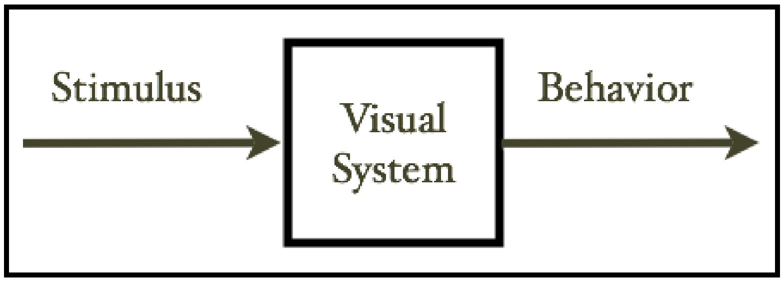
What is Computational Vision? It is the study of how to compute useful scene information, object properties, and action parameters from image measurements. And central to this course, it is in particular the study of how we as humans accomplish this computation. Thus, we will study human visual behavior to understand what information is and is not used for visual inference. And we will study neural systems because we would like to understand how the machinery of the eye and brain enables actions and the inference of properties of scenes in the world from images.

What does it mean "to understand visual perception"?

If we think we understand human vision, then we should be able to build or simulate a machine "which sees like we do". But what does this mean? There are several levels of abstraction that have to be considered, and the answer to this question requires careful thought. Robot vision, even if excellent, would not necessarily work the same way as human vision even at a very general level. For example, a vision system could rely on the reception of natural image patterns, or could actively send out signals to see how the environment modulates them. An example of the latter case would be to project alternating stripes of light and dark on surfaces, and then use the systematic distortion of the stripes in the image which to decode the shape. In fact, this latter principle is used (laser painted stripes) in commercial applications to measure shapes (e.g. Cyberware scans of the human face). But biological vision (in contrast to echo location) processes nature's patterns without active modulation of its images.

The study of cognitive science, in particular vision as we've pointed out, is necessarily interdisciplinary involving behavioral, biological and mathematical approaches. Let's take a closer look at the methods of three specific areas within each of these general approaches: Psychophysics, Neuroscience and Computer Science. We will gain some familiarity with all three of these disciplines in this course, but let us first have a preview of their respective contributions and see how they relate to different levels of analysis.

Behavior and Psychophysics: Black-box approach to construct a model that "sees like we do"



This approach has a *major goal to qualitatively and quantitatively describe visual behavior*. Psychologists, ethologists and behavioral and systems neuroscientists all study behavior...but as we've noted, even physicists and mathematicians get in the act. Careful description is an essential first step in any science (e.g. Mendel, and genetics).

The study of behavior can be of at least two types.

First, we need to *know and understand the visual functions of an organism*. Human vision is used to identify objects, to read, to walk, to drive, to steady oneself, to reach and grasp, to throw, to plan, to judge beauty, and the list goes on. Different tasks require different kinds of image processing and inference.

This brings us to the second type of behavioral study, *psychophysical analysis*. Psychophysics measures the behavioral consequence of physical (or informational) variations in the image stimulus with a goal towards understanding underlying neural mechanisms. Examples are: the measurement of apparent brightness as a function of physical light intensity; just discriminable differences in light intensity; changes in sensitivity as a function of adaptation; changes in recognition performance as a function of viewpoint. Clever psychophysical experiments can reveal not only the diverse visual processing requirements, but also test hypotheses about alternative accounts of a given process.

Psychophysics goes beyond mere description and historically has made some striking predictions about the nature of the underlying biology. For example, the psychophysics of color matching in the 19th century anticipated three physiological cone receptor types and their relative spectral sensitivites as a function of wavelength. This so-called "trichromacy" theory of was not physiologically established at the cell level until the 1960's. We will see later how psychophysics in the 1940's showed that photoreceptors (rods) in the eye could transduce single photons into an electrical signal. Certain brightness illusions discovered in the 19th century suggested patterns of neural connectivity and spatial interaction (called "lateral inhibition") that were not put on a firm physiological and neural foundation until the 1950's.

A psychophysical approach has clear limits in its ability to give an account of how we see. One reaches a point where too many theories of what is inside the box give the same input/output relation in the psychophysical data. For example, computing y as: $y = x(x - 1)$ gives the same mathematical relationship as a different computation in which x is subtracted from its square : $y = x^2 - x$. The fact that different combinations of wavelengths of light appear the same could have many neural explanations. Researchers eventually "go inside the box", or a animal "model" of the box (like a frog, cat or monkey) to find out what was going on at a finer level of analysis. This brings us to the methods of neuroscience, such as anatomical tracing, electrophysiologal recording from single neurons, and brain imaging.

Neuroscience: Going inside the box

What happened when physiologists and anatomists looked at the biological basis of the psychophysical descriptions? Indeed, as discovered using microspectrophotometry in the 1960's, there are 3 distinct types of cone photoreceptors in the retina at the back of the eye. Electrophysiological recordings showed that their spectral sensitivities were remarkably similar, but not identical, to those inferred from psychophysics. There are neural circuits (lateral inhibition) in the retina that behave like Ernst Mach predicted to account for certain brightness illusions.

And as we will see in the next lectures, 1940s psychophysical predictions were by verified neuroscientists in the 1970's: photoreceptors can transduce single photons.

Later, we will see examples of more recent neural accounts of psychophysical observations that go beyond retinal processing to other parts of the brain, accounts that are being tested using both electrophysiological and brain imaging techniques.

In the 1950's and 1960's there was a tremendous excitement that we could understand the brain's function and in particular visual perception in terms of single neurons....but neurobiologists had probably been particularly lucky...at least they were more fortunate than if the brain had been designed like a modern digital computer. In the 1970's, it became increasingly apparent that understanding how biological systems discriminated light and pattern was only scratching the surface of the problem of vision. Computer vision was beginning to show that competent vision was truly a problem of sophisticated inference and estimation. The number of visual areas discovered in the cortex of the brain grew.

Vision was becoming more complicated and harder than expected.

Computational Vision: The need for tools to deal with the complexities of perceptual inference

Imagine the following example. Sometime in the distant future, Martian scientists have acquired a Terran computer device that plays an ancient video game, say Super Mario Brothers. Now consider the various ways these scientists might go about trying to understand this device.

First, they could adopt technique adapted from a neuroscience, "anesthetize" the computer (i.e. just take away the screen, so there is no external output), and begin using a volt meter or a logic probe to figure out what the box is doing. This is like doing neuroscience without psychology or psychophysics. The scientists might learn about logic gates, shift registers, and RAM, etc.. But, what are the chances of figuring out that the machine was even designed to play a game? Pretty slim.

But now give the scientists a working system complete with screen and the controls, but minus the logic probe. This is like doing behavioral science--psychophysics. With some careful experiments, they could begin to figure out the rules of the game. (Although, they might be left with questions forever unanswerable, like "why was this machine built in the first place"!). If asked to "build a machine" that does the same thing, the Martian scientists might still have a hard time. They might be able to build a copy that mimics the behavior of the game, but even if the scientists had a solid body of results using the logic probe and observing the functioning system as a whole with the screen on, there is still something missing. They would have missed the point that the essential structure of the game is not the hardware, nor the input-output relations, but rather a highly complex computer program. They need an understanding of the intermediate level -- the software (firmware), the algorithms, and how that these are related to the hardware that supports it.

In short, what is missing is an understanding of how the pieces fit together to solve a specific information processing task-- a task that involves getting magic mushrooms, escaping turtles, smashing brick ceilings to get coins, jumping up flag poles, etc. Without this knowledge, they would be unable build new video games, like Mario Brothers. True understanding of vision should result in the generalization, e.g. the capability of building a machine that sees like us, but which may differ from the original in ways that we can understand.

This example illustrates the need for a **computational approach to vision**.

The computational level involves understanding how image patterns are formed (generative models), and how scene inferences can be drawn from image patterns. To handle the complexity of natural patterns requires the tools of computer programming, and has been increasingly emphasized in recent years, the mathematics of statistical inference.

Although the computational approach grew out of the early communications theory, cybernetic, and artificial intelligence studies of the late 1940's, 1950's and 1960's (e.g. Turing, von Neumann, Wiener, Shannon), one of the chief protagonists of this approach for the study of visual perception was the late David Marr from MIT in the early 1980’s.

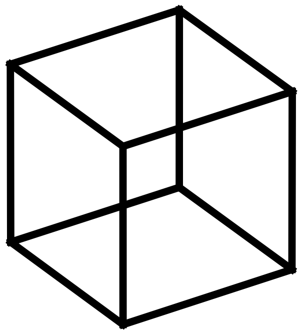
Levels of analysis

Although Marr made many specific contributions to understanding human vision, he is well-known for his elucidation of a computational approach as consisting of multiple levels of explanation for puzzles of perception. Computational vision also requires study at several *levels of analysis or abstraction*. Let's see what this means.

■ Functional ("computational") Theory. What is the goal of a computation?

Why is it appropriate? What strategy can carry it out? Both psychology and theoretical analysis help to answer these questions. For example, the Necker cube (below) perceptually flips because the goal of the visual computation is to represent the 3D structure of the objects causing the 2D retinal image. But, there are two equally plausible 3D interpretations.

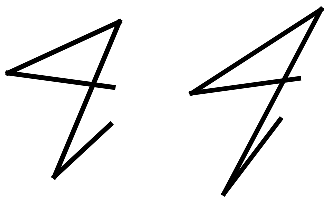
Necker cube



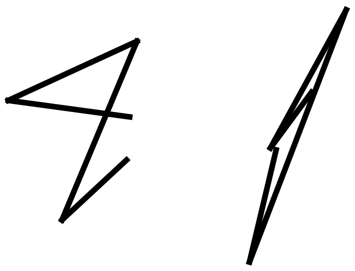
■ Representation and algorithm. How to represent input and output? How to get from input to output?

Psychology and computer program models help answer these questions. For example, mental rotation experiments done with human observers give clues as to how visual information is represented and processed. The time to decide whether a sample figure on the right is a rotated version of the one on the left or not, increases monotonically with the actual angle required to check the match (over a certain range and conditions).

The right figure is an image of the same object as the one on the left, but is rotated by 30 degrees about the vertical axis:



In the next figure below, the right figure is rotated by 80 degrees:



■ Implementation or hardware. How to build it with actual components?

An algorithm can be implemented with pencil and paper, devices with mechanics, vacuum tubes, silicon chips, or neurons. Neurobiology and neural network computer simulation help to understand the particular characteristics of neural computation. For example, after-images are a well-known perceptual phenomenon. The after-image of a flash from a camera is an effect of visual perception that has to do with how human vision implements transduction in the retina, and the receptors in particular.

In future lectures we will see the relationships between human behavior (psychophysics), physiological mechanisms, and computational theory. Sometimes the computational theory comes first, but sometimes we will work backward from an experiment or visual phenomenon to the theory. In the early part of the course we will look at:

o quantum limits to vision-- What are the theoretical limits to light discrimination? -> Computational theory of discrimination.

o lateral inhibition in neural networks-- for detecting edges? or to reduce redundancy? -> Computational theory of neural image coding.

In the next lecture we will begin by studying one of the simplest of vision problems: How well can we detect and discriminate light intensity? What are the limits to this ability? Quantum nature of light? What is the computational theory for brightness discrimination?

Getting started with *Mathematica*

Mathematica vs. Matlab

Introduction

- **To get an overview with audio, go to the screencast:**

http://www.wolfram.com/broadcast/screencasts/handsonstart/

or for version 9:

http://www.wolfram.com/support/learn/get-started-with-mathematica/

- **Go to the Help menu in *Mathematica*. Go to Documentation Center, and from there to Getting Started Videos or Find Your Learning Path.**
- **You can read *Mathematica* files free with Mathematica Player.**

- **Front-end and Notebooks: Organize, outline, document. Program, evaluations, data all in one place Kernel: Separate program does the calculations**

Some practice

- **Numerical Calculations.** You can do arithmetic. For example, type 5+7 as shown in the cell below, and then hit the "enter" key. Note that if you
  try division, e.g. 2/3, you get the exact answer back. To get a decimal approximation, type N[2/3].

```
5+7
```

```
12
```

```
2 / 3
```

$$\frac{2}{3}$$

```
N[2/3]
```

```
0.666667
```

You can go back and select an expression by clicking on the brackets on the far right. These brackets serve to organize text and calculations into a Notebook with outlining features. You can group or ungroup cells for text, graphs, and expressions in various ways to present your calculations. Explore these options under Cell in the menu. You can see the possible cell types under the **Style** menu.

Try some other operations, 5^3, 4*3 (note that 4 3, where a space separates the digits is also interpreted as multiplication.

Evaluate 4*3

Compare with 4 3 (i.e. 4 followed by a space, and then 3).

Seems pretty dumb so far...but you can see that *Mathematica*'s default handling of arithmetic is special:

Compare the square roots of: 12345678987654321 and 12345678987654321.0

Compare (2^.000000000001)^1000000000000 with (2^(1/1000000000000))^1000000000000

If you don't explicitly tell *Mathematica* that your expressions involving floating point numbers, it will treat the numbers as true integers. And in general, *Mathematica* will do symbolic processing as a default, and assume you want as general an answer as possible. Because symbolic processing demands considerably more computer resources than numerical processing, this result in annoying consequences, like slow or no responsivity. So remember to include a decimal point somewhere, or use **N[]** to force a floating point representation, and consequently numerical computations.

■ Front-end stuff

There's a lot to explore stylistically in Notebooks, but one of the most common things you will do is to select and manipulate the brackets on the far right. You can go back and select an expression by clicking on the brackets on the far right. These brackets are features of the user interface and serve to organize text and calculations into a Notebook with outlining features. You can group or ungroup cells for text, graphs, and expressions in various ways to present your calculations. Explore these options under **Cell** in the menu. You can see the possible cell types under the **Style** menu.

■ By ending an expression with ; you can suppress the output.

```
(3/4)/6;
(3 4)/6;
```

In[176]:=
```
a = Table["hi", {50 000}]
```

Out[176]=
```
A very large output was generated. Here is a sample of it:
{hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi,
 hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, <<49 940>>,
 hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi,
 hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi, hi}
Show Less  Show More  Show Full Output  Set Size Limit...
```

The most recent result of a calculation is given by %, the one before by %%, and so forth. Try it on the previous two outputs

---

Built-in functions

Mathematica has a very large library of built-in functions. They all begin with an uppercase letter and the arguments are enclosed by square brackets. Knowing that, you can often guess the form of a function.

Try taking the logarithm of 8.0

Did it return log to the base 10 or e? Check the definition by typing "?Log" or by typing "Log[E]" and "Log[10]"

You can get information more about a function, by clicking on the resulting link >>

Try Log[2,8]

You can get information about a function, e.g. for the exponential of a function, or for plotting graphs by selecting the function (e.g. Exp) and going to "Find Selected Function" in the Help menu. Or you can enter:

```
?Exp
```

Exp[z] gives the exponential of z. ≫

and then click on >> to take you to the documentation page.

```
?Plot
```

Plot[f, {x, x_{min}, x_{max}}] generates a plot of f as a function of x from x_{min} to x_{max}.
Plot[{f_1, f_2, …}, {x, x_{min}, x_{max}}] plots several functions f_i. ≫

If you type two question marks before a function, **??Plot**, you'll get more information. Try it. What does the **Random** function do?

Lists

We will uses lists a lot. Lists are general in *Mathematica*. Their elements can be numbers, symbols, other lists, and a list can have mixed types.

In[191]:=
```
{1.1, "hi", {Pi, 33 / 2}}
```

Out[191]=
$\left\{1.1, \text{hi}, \left\{\pi, \frac{33}{2}\right\}\right\}$

Mostly we will use lists to represent vectors and matrices. You can quickly make a list using the Table function:

```
avector = Table[i^2, {i, 1, 6}]
```

Out[181]=

```
{1, 4, 9, 16, 25, 36}
```

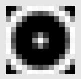
```
amatrix = Table[N[Cos[Sqrt[i^2 + j^2]]], {i, -6, 6}, {j, -6, 6}];
```

Remove the ; above to see what the list looks like in output form.

We will use matrices (a list of lists of pixels) to represent images. We can visualize what this 2D list looks like using Image[]

```
Image[amatrix]
```

Out[186]=



Defining your own functions

Let's illustrate function definition by building a really simple model of a neuron. We'll see the justification later. Suppose a neuron takes a set of inputs, say **x1**, **x2**, **x3**, (e.g. from the outputs of three other neurons in the network) and produces an output signal, call it **y**. For a so-called linear model, the output is the weighted sum of the inputs. We'll assume the weights are fixed and given by w1, w2, w3.

```
w1 = -1; w2 = 2; w3 = -1;

y[x1_, x2_, x3_] := w1 * x1 + w2 * x2 + w3 * x3;
```

(This code is pretty primitive and not very general--don't worry, we'll get more sophisticated later).

The underscore in **x1_** is important because it tells Mathematica that **x1** represents a slot for a variable, not an expression.

■ := vs. =

Note that when defining a function, we used a colon followed by equals (:=) instead of just an equals sign (=). When you use an equals sign, the value is calculated and assigned immediately. When there is a colon in front of the equals, the value is calculated only when called on later. So we use := for function definition because we need to define the function for later use and evaluation, when we may have new values for its arguments.

A double equals (==) has yet another meaning and is used to represent a symbolic equation which evaluates to True if left and right hand sides are identical.

Let's define xv using :=, and xf using =

```
xv := RandomInteger[10];
xf = RandomInteger[10];
```

Now evaluate r1 and r2 three times each. What is the difference between the two definitions?

```
{xv, xf}
```

```
{6, 3}
```

..but if we evaluate xf = RandomInteger[10] again, we re-initialize it:

```
xf = RandomInteger[10]
```

```
7
```

```
xf = RandomInteger[10];
y[xf, xf, xf]
```

```
0
```

```
y[xv, xv, xv]
```

```
-5
```

Why does our neuron model always output a zero when the inputs all have the same value? What other family of inputs will all produce zero output?
Hint: The weights we defined can be thought of as a discrete approximation to a 2nd derivative operator in differential calculus.

Graphics & more function definitions

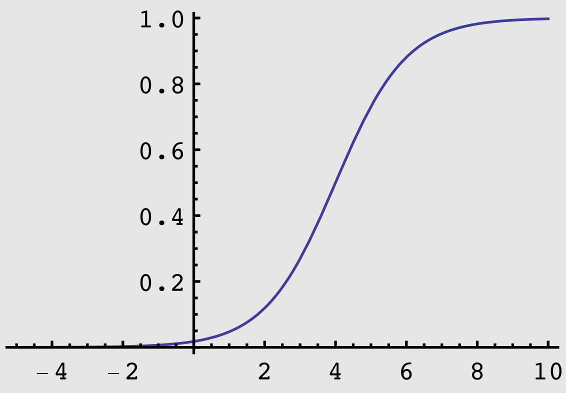
Later on we'll require defining a function that suppresses small outputs and "squashes" or clamps large outputs to a maximum level. Here is an example:

```
squash[x_] := N[1/(1 + Exp[-x+4])];
```

Also note that our squashing function was defined with **N[]**. Again, remember that *Mathematica* trys to keep everything exact as long as possible and thus will try to do symbol manipulation if we don't explicitly tell it that we want numerical representations and calculations.

Let's plot a graph of the squash function using the syntax we discovered above for -5<x<10

```
Plot[squash[x], {x, -5, 10}]
```



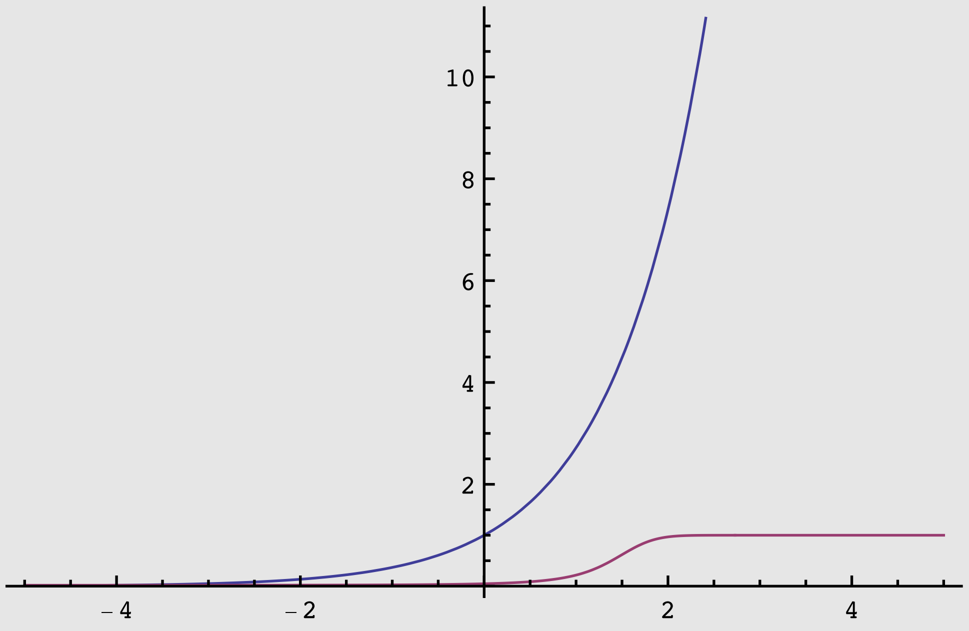
We'll use this and similar functions to model the small-signal compression and large signal saturation characteristics of neural output.

Define a new function squashedExp[ ] that applies squash to an exponentiated value (i.e. takes Exp[x] as the argument of squash[ ])

```
squashedExp[x_] := squash[Exp[x]];
```

Plot squashedExp[ ] for x going from -5 to 5

```
Plot[{Exp[x], squashedExp[x]}, {x, -5, 5}]
```



Even though Exp grows exponentially fast with x (by definition!), squash “keeps a lid” on it.

Ask *Mathematica* for the definition of squashedExp[ ]

It can be important to check your definitions like this. One reason is that, as we will see later, *Mathematica* definitions can be built up with multiple constraints. And sometimes you might add to a function unwittingly and it appears to misbehave. You can check your defintion by asking *Mathematica* for it.

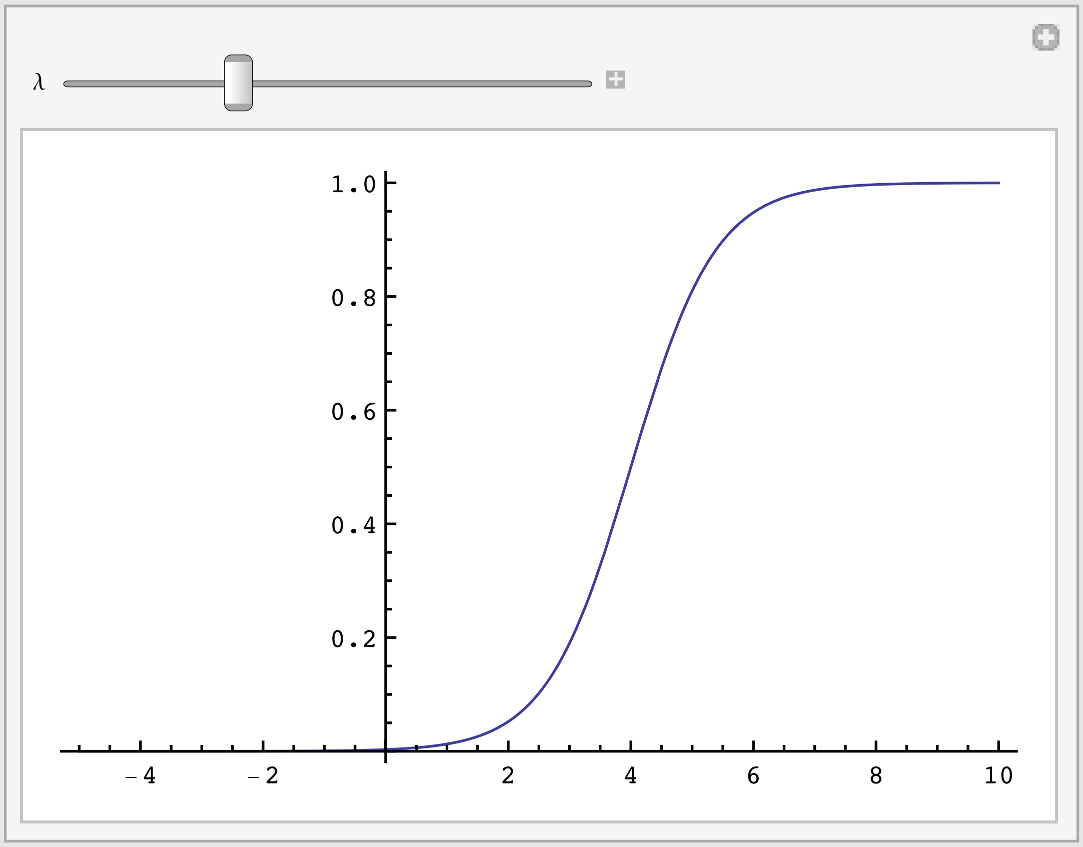
Define a new version of the squash function to include a steepness term λ, and use Manipulate to control λ

Mathematica 7 & 8 have very useful and easy-to-program GUI tools. One of the most common is **Manipulate[]**

In[192]:=

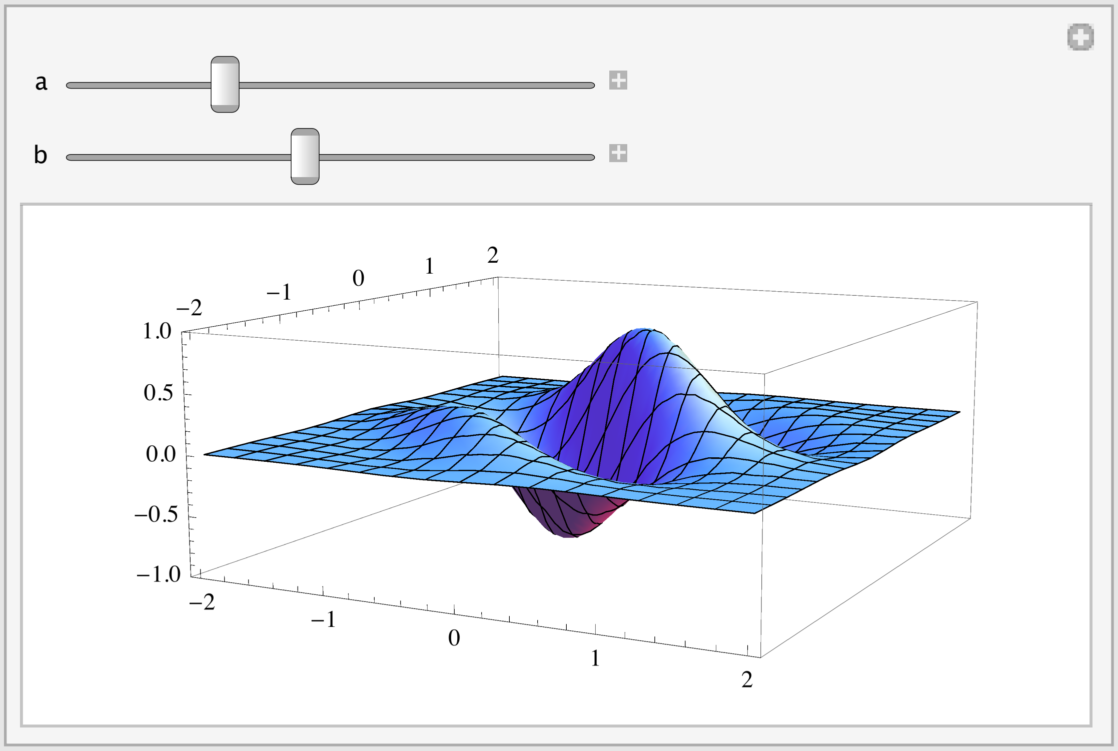
```
dsquash[x_,λ_] := N[1/(1 + Exp[(-x+4)/λ])];
Manipulate[Plot[dsquash[x,λ], {x, -5, 10}],{{λ,1},0.1,2}]
```

Out[193]=



Here's a 3D surface plot representing possible selectivities in the receptive field of a simple cell in the visual cortex:

In[196]:=
```
Manipulate[Plot3D[Exp[-x^2 - y^2] * Sin[a * x + b * y], {x, -2, 2},
  {y, -2, 2}, PlotRange → {-1, 1}], {a, 1, 6}, {b, 1, 6}]
```

Out[196]=


Try using your mouse and mouse button to rotate the above plot. What does the Option or Alt key do?

Image processing

You can drag an image into the argument slot

```
Image[
```

```
];
```

You can select, copy, and the paste the image into other function, e.g. to extract the data in list form:

```
idata = ImageData[
```

```
];
```

Images are often really big, so we can check the format and size of the image data using Dimensions[]

In[187]:=
```
Dimensions[idata ]
```

Out[187]=
```
{107, 85, 3}
```

So it isn't a simple matrix. The first two entries tell us the number of rows and columns respectively. Note that although an image might look black and white, it's pixels have still three values, one for each color channel. This is because it is in color format. We can convert it to black and white using ColorConvert[]

In[188]:=
```
ColorConvert[
```

```
, "Grayscale"]
```

Out[188]=

In[190]:=

```
Dimensions[ImageData[ColorConvert[
```

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

Out[190]=

```
{107, 85}
```

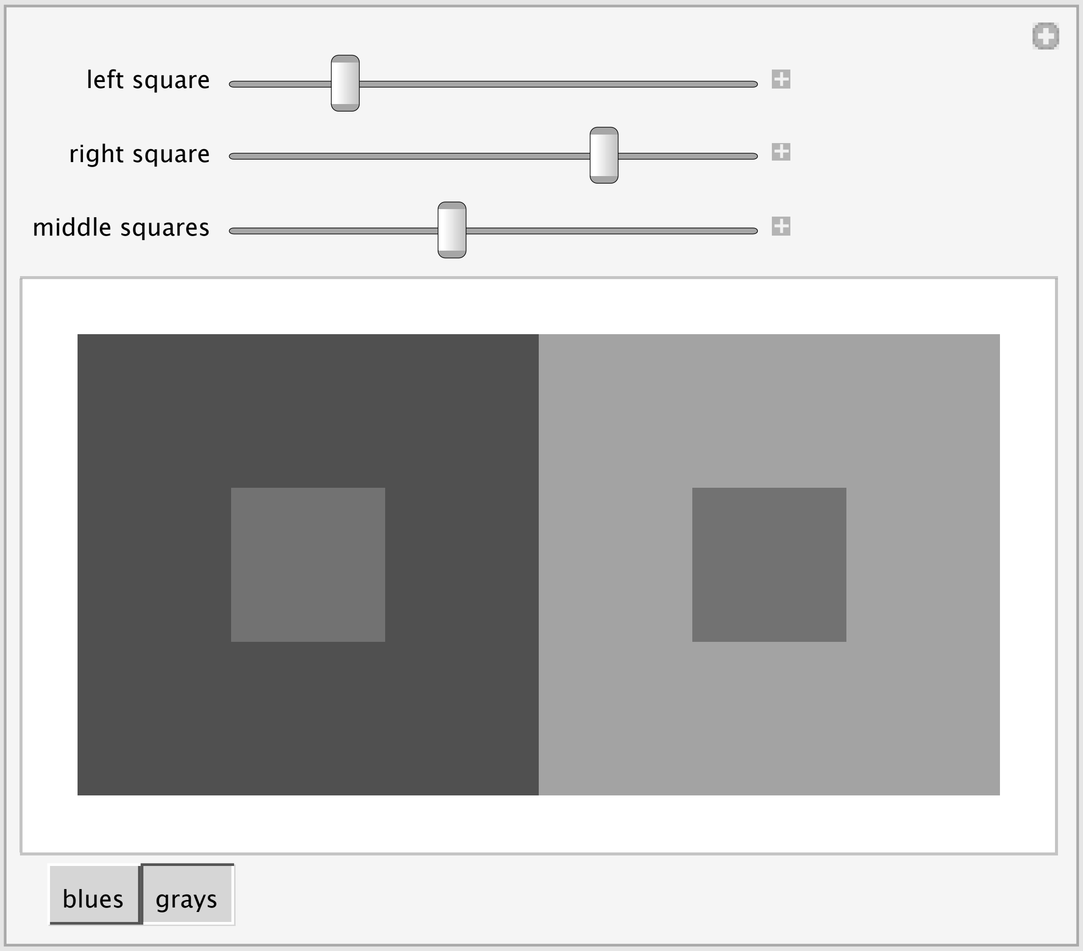
Try dragging a new image into one of the above image functions.

Mathematica Demonstrations project

Check out the Demonstrations center at the Wolfram *Mathematica* site for some cool examples, such as:

```
Manipulate[Module[{bluecol, colfunc},
  bluecol[n_] := Blend[{Black, Blue, White}, n];
  If[usecol, colfunc = bluecol, colfunc = GrayLevel];

  Graphics[{colfunc[left], Rectangle[{0, 0}, {3, 3}], colfunc[right],
    Rectangle[{3, 0}, {6, 3}], colfunc[mid], Rectangle[{1, 1}, {2, 2}],
    Rectangle[{4, 1}, {5, 2}]}]], {{left, .35, "left square"}, .2, .8},
 {{right, .75, "right square"}, .2, .8},
 {{mid, .5, "middle squares"}, .2, .8},
 {{usecol, True, ""}, {True → "blues", False → "grays"}},
  ControlPlacement → Bottom}]
```



"The Simultaneous Contrast Effect" from The Wolfram Demonstrations Project http://demonstrations.wolfram.com/TheSimultaneousContrastEffect/

References

Helmholtz, H. v. (1867). Handbuch der physiologischen optik . Leipzig: L. Voss.

Hoffman, D. D. (1998). Visual Intelligence . New York: W. W. Norton & Company.

Kersten, D. High-level vision as statistical inference. (1999) *The New Cognitive Neurosciences*, 2nd Edition, Gazzaniga (Ed.). MIT Press.

Kersten, D., & Yuille, A. (2003). Bayesian models of object perception. *Current Opinion in Neurobiology, 13*(2), 1-9.

Knill, D. C., & Richards, W. (1996). Perception as Bayesian Inference . Cambridge: Cambridge University Press.

Marr, D. (1982). Vision: A Computational Investigation into the Human Representation and Processing of Visual Information. . San Francisco, CA: W.H. Freeman and Company.

Mumford, D. (1995). Pattern theory: A unifying perspective. In D. C. Knill, & R. W. (Ed.), Perception as Bayesian Inference (Chapter 2). Cambridge: Cambridge University Press.

Poggio, T. (1984). Vision by Man and Machine. Scientific American, 250, 106-115.

Zeki, S. (1993). A Vision of the Brain . Oxford: Blackwell Scientific Publications.