Computer Vision

Lecture 2: Images

Last lecture

- About the course
- About the subject
- What CV programs do: the bottom-up pathway through vision
- 2-D and 3-D information in images (exercise)
- Alternative approaches to computer vision

This lecture

- Conclusions from the images exercise
- The optic array
- Cameras and images
- Images in the computer: arrays and neighbourhoods
- An image processing example: simple intensity edges

Image structures for 3-D and recognition

You were asked to think about the kinds of information single images carry for 3-D shape. Possible answers include

- Linear perspective. Look for fans of straight lines diverging from a *vanishing point*. **Assume** the scene actually contains parallel lines in 3-D.
- Texture gradient. Look for systematic changes in image size of similar objects. **Assume** that things with similar appearance have a fairly uniform size in the scene.
- Height in image. Higher up in the image often means further away. **Assume** that the scene lies on a *ground plane.*
- Size of known objects. The image size of a recognised object of known dimensions gives its distance. **Assume** that you're seeing the real thing, not a model for example.
- Shading. Image brightness gives information about surface orientation relative to light source. **Assume** that the brightness variation is not intrinsic to the surface; may also need assumptions about the light source and surface properties.
- Foreshortening. The image shape of a surface patch depends on its orientation — e.g. circles foreshorten to ellipses. **Assume** known true shape, or symmetry, of contour.
- Focus. Things nearer or farther away than the point of focus of the lens may be blurred. **Assume** that real objects generally have sharp boundaries.
- Occlusion. The images of nearer things cut across the image structure of farther things. **Assume** that the edges of objects are not accidentally lined up in the image ("*general viewpoint"* assumption).

You were also asked to think about the kinds of information that the image carries for object recognition, without building a 3-D description. This is more open-ended, but possibilities include:

- significant colours (flesh tones are often used for locating people);
- significant textures — e.g. repeated linear structures on buildings, intricate patterns on vegetation;
- significant shapes — e.g. circles/ellipses may be wheels on a vehicle;
- spatial relationships between shapes — two circles aligned horizontally strengthen the possibility of a vehicle.

Conclusions from the exercise

Computer Vision does not necessarily mean inferring 3-D or recognising objects. But these are extremely important sub-goals in many systems, and people are very good at these tasks. A CV system is therefore likely to need a substructure that can support these tasks.

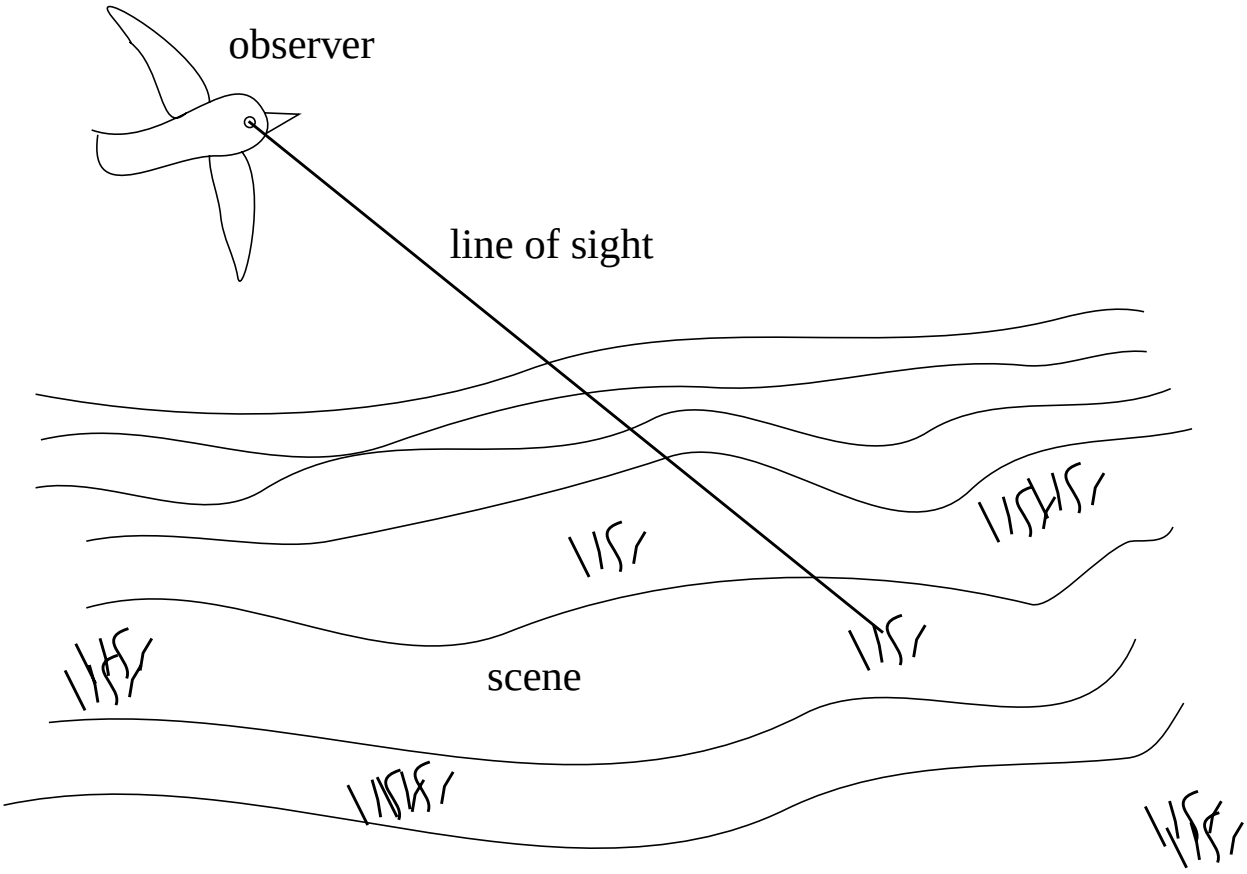
- Image processing operations will be needed to find *image structure*, including such things as
  - object boundaries
  - straight lines, circles and other geometrical shapes
  - texture descriptions.
- The ability to use *geometrical* operations to exploit the information the images carry about a 3-D world.
- Mechanisms for combining different kinds of *evidence* about location, shape or object identity.

The Optic Array

How do we describe our visual environment? A central concept is the *optic array* — a term coined by J.J. Gibson.

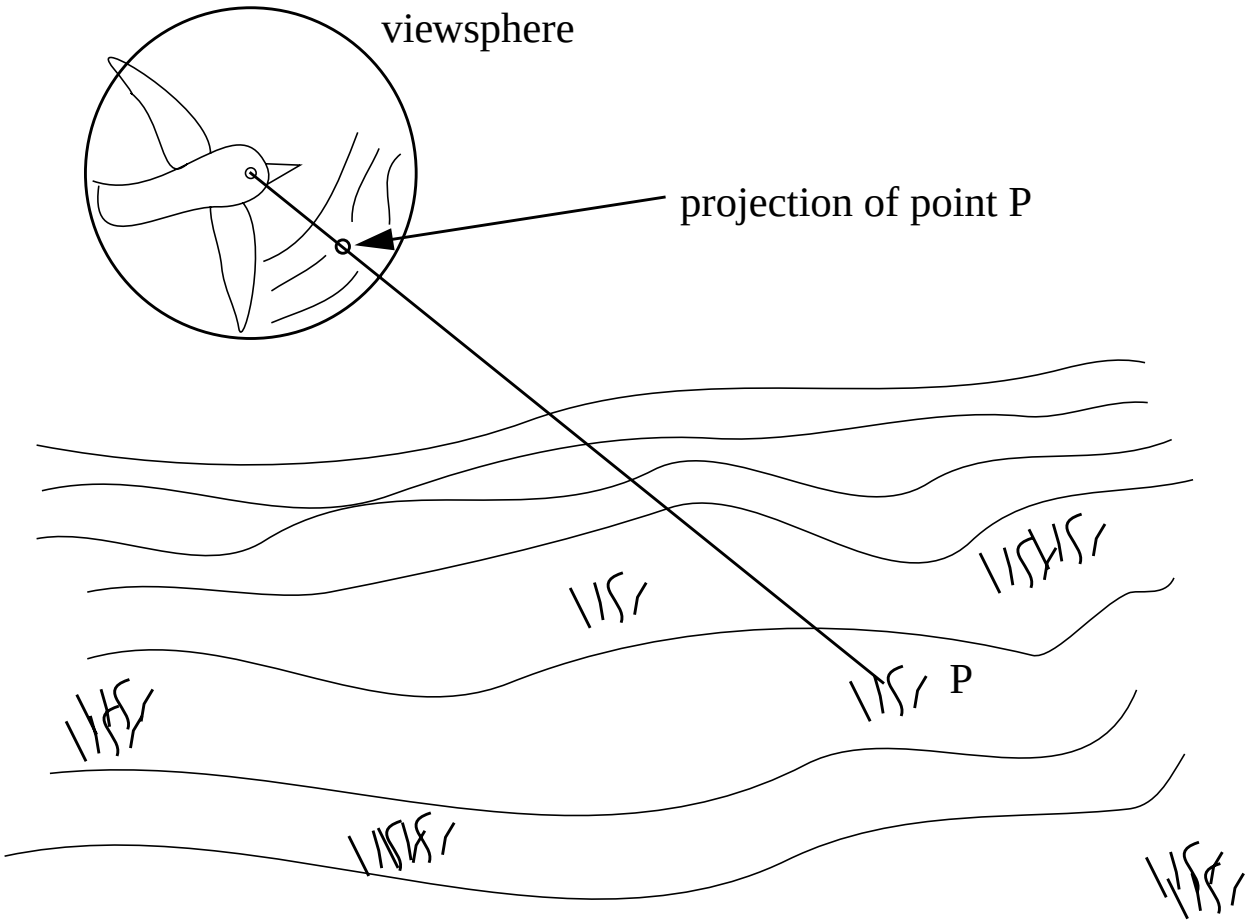
We describe the light reaching an eye or camera in terms of intensity as a function of **direction** (and of wavelength if we are interested in colour vision).

Directions are specified relative to some **frame of reference** fixed to the eye or camera.



The optic array can be thought of as describing the intensity of light reaching an observation point along every possible line of sight.

Another way to view this is as a *projection* of the world onto a sphere centred on the observation point, and moving and rotating with the eye or camera.



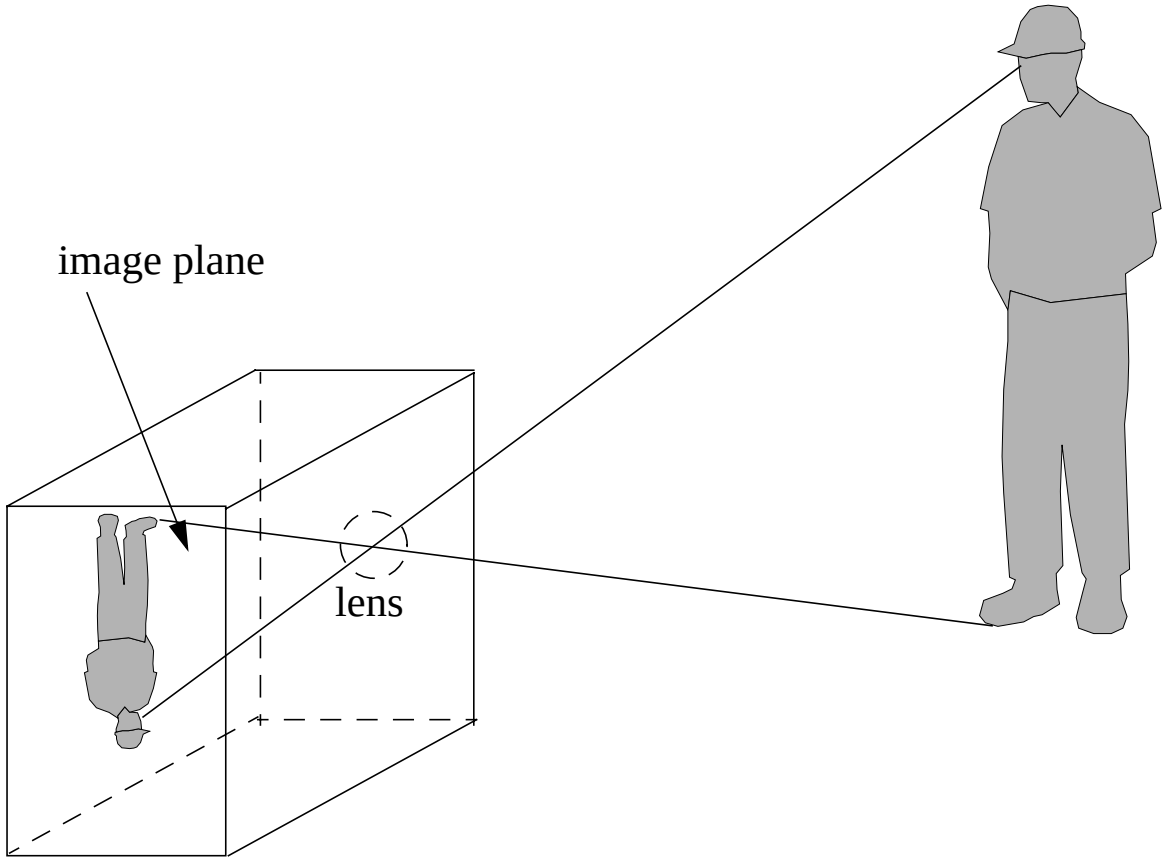
This description of the optical input lends itself to theoretical analyses. It is often used when general properties of vision, independent of specific visual systems, are considered. Both abstract qualitative analysis (as in Gibson's work) and formal mathematical models are based on this idea.

The optic array is the real starting point for vision.

The digital image

In practice, we often start from the more concrete idea of the image formed in a camera. This represents a sampling of part of the optic array.

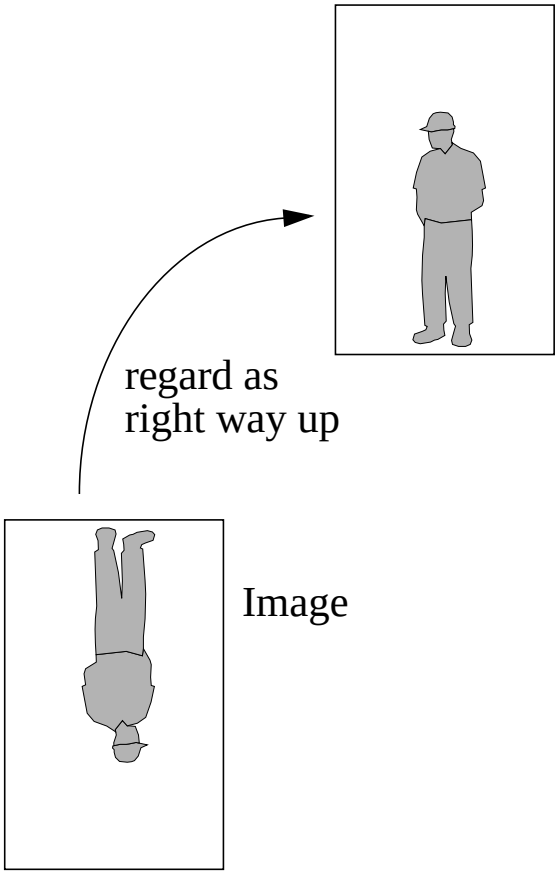
A simple **camera model** is usually used.



The lens is modelled as a pinhole — straight lines passing through its centre join corresponding points on object and image.

Simple mathematical formulae describe this model — we will come to them later in the course.

For Computer Vision, the image is digitised by suitable hardware. This is usually done on a uniform rectangular grid of samples (sometimes called a *raster-scan*).



12	15	12	18	19	20	19	17	16	15
13	14	61	19	11	9	10	12	14	19
13	15	18	17	19	20	23	11	10	12
14	15	14	14	17	20	21	10	9	12
12	13	13	19	30	35	36	15	19	15
15	16	17	18	45	40	38	16	15	12
16	17	33	17	50	76	50	25	8	13
17	18	17	46	66	87	43	12	17	17
20	19	18	39	67	88	76	65	13	12
11	18	17	42	76	81	54	64	10	17

In much research, colour is ignored. The numbers represent average brightnesses in small regions of the image. (In this idealisation, the regions are square and non-overlapping.)

The digitised image is then often called a **grey-level array**. Usually bigger numbers mean brighter. A typical range for grey levels is integers from 0 to 255. Other ranges may be used — e.g. real numbers from 0.0 to 1.0.

The grey-level array

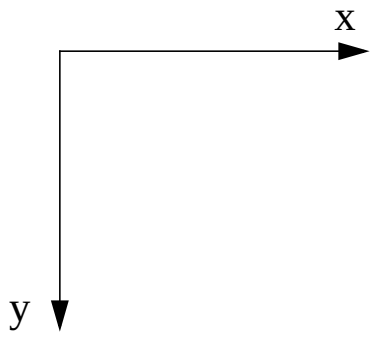
Each element of the array is called a **pixel.** It is identified by its **column** and **row** in the array.

		Column number									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Row number	**1**	12	15	12	18	19	20	19	17	16	15
	2	13	14	61	19	11	9	10	12	14	19
	3	13	15	18	17	19	20	23	11	10	12
	4	14	15	14	14	17	20	21	10	9	12
	5	12	13	13	19	30	35	36	15	19	15
	6	15	16	17	18	45	40	38	16	15	12
	7	16	17	33	17	50	76	50	25	8	13
	8	17	18	17	46	66	87	43	12	17	17
	9	20	19	18	39	67	88	76	65	13	12
	10	11	18	17	42	76	81	54	64	10	17

When specifying a pixel, column number normally comes first. (Note for mathematicians: this is the opposite to the matrix convention.)

So the pixel at (2,9) has grey level 19.

Column and row are often represented by the symbols **x** and **y** respectively. (Note for mathematicians — this is a left-handed coordinate system.)



The central property of the grey-level array:

> **Each pixel corresponds to a direction in space, fixed relative to the camera.**

The mapping from row and column to pixel direction is done via the camera model.

In **calibrated cameras** the mapping is known.

In **uncalibrated cameras** the mapping may be unknown or partially known.

In all camera models, it holds that:

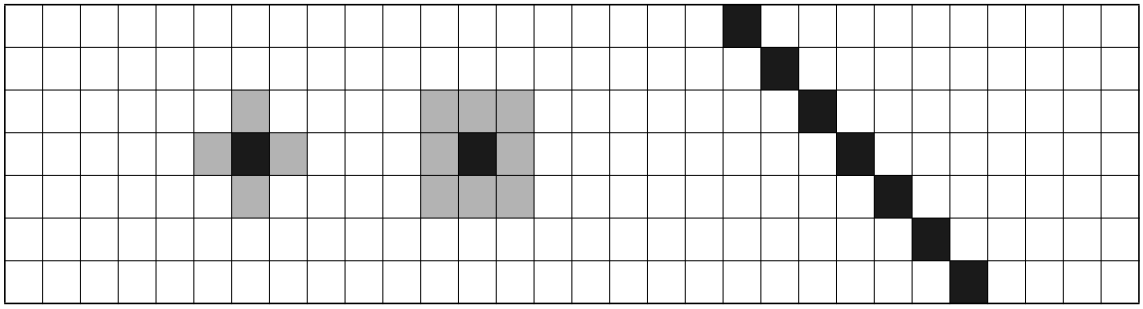
> **Neighbouring pixels correspond to neighbouring directions in space.**

Usually, neighbouring directions in space generally intersect objects at neighbouring positions. The exceptions are at **occluding contours.**

These properties are the basis for most low-level image processing operations.

Neighbourhoods

In a rectangular grid, pixels may be regarded as having 4 or 8 neighbours:

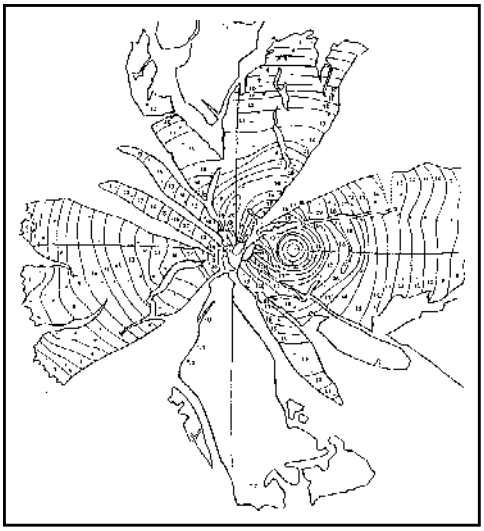


Note that the example on the right means the background must have different connectivity to the foreground.

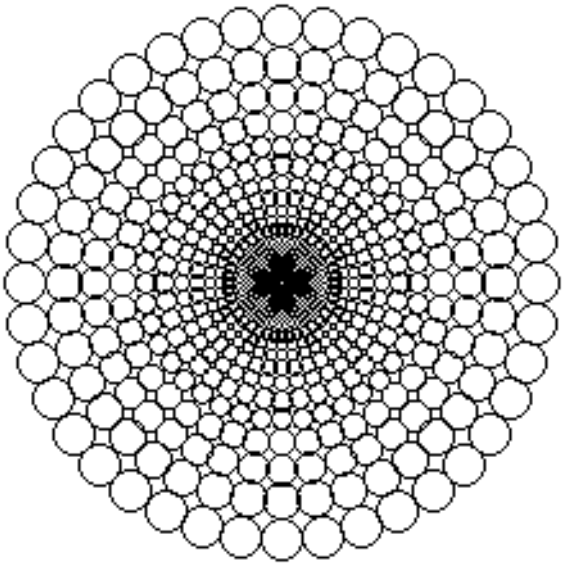
Alternatives to the raster scan

Our own eyes (and other animals') are not structured much like the digitising array of a Computer Vision setup.

A map of the human retina (Østerberg, 1935), showing cone density:



Such **foveal** layouts of image samples are better represented by **log-polar** image sampling, sometimes used in Computer Vision:



An image processing example

Finding vertical edges

An **edge** is a junction between two pixels where there is a significant change in grey-level.

A simple operation to find vertical edges is just to subtract each pixel's grey-level from that on its right. If changes bigger than, say, 10 grey levels are regarded as significant, then we can produce a **binary image** indicating where there are edges.

For example

	4	**5**	**6**	**7**
4	14	17	20	21
5	19	30	35	36
6	18	45	40	38

→

5	**6**	**7**
3	3	1
11	5	1
27	-5	-2

→

5	**6**	**7**
0	0	0
1	0	0
1	0	0

Note that we have to make an arbitrary choice to put the result of subtracting (4,4) from (5,4) into (5,4).

The second operation is called **thresholding**. Choosing the threshold (in this case 10) is a problem in its own right.

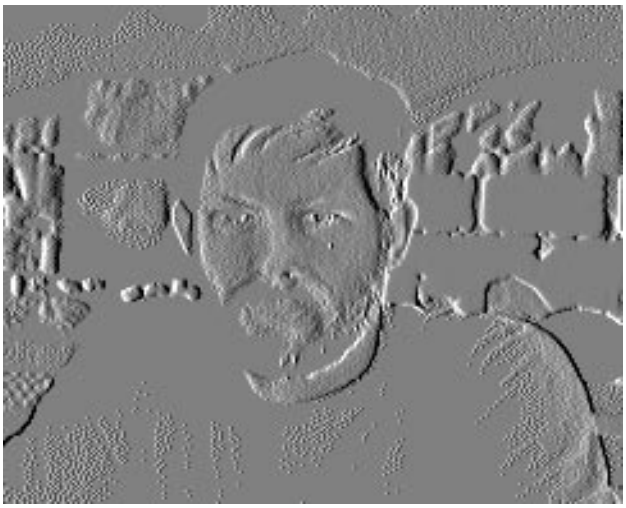
If we want to detect all vertical edges, then we need to threshold so that we record a "1" wherever the difference is greater than 10 **or** less than -10.

These two operations, though simple, in fact conform to the structure used by a very wide range of visual systems.

On a real image, the results look like this:

Original image.

Black = 0,
white = 255



Differenced image.

Black negative,
mid-grey = 0,
white positive

Binary image.

Black = 1,
white = 0