Module 5 Effective data visualisation

Kimbal Marriott

Contents

Chapter 1

Effective data visualisation: Overview

By Kimbal Marriott

Updated 17 March 2017

This is the fifth module in the Data Exploration and Visualisation unit. In this module you will learn about the human visual system and how humans communicate. By understanding these you will be able to design more effective visualisations that take into account human perceptual and cognitive strengths and limitations.

1.1 Aims of this module

After completing this module you will:

- understand how the human visual system works and how this impacts on the design of effective visualisations;
- understand data visualisation as a communication act and the implications of this for effective visualisation design;
- appreciate the need for careful design when visually communicating the results of data analysis to stakeholders;
- be able to design effective data visualisations for communicating the results of data analysis;
- be able to use D3 to construct interactive visualisations.

1.2 How to study for this module

In this module we draw on books, journal and conference articles as well as material in the public domain, including quite a few videos.

In this module there is one assessment activity:

- Programming a simple interactive visualisation using D3.

Chapter 2

The human visual system

By Kimbal Marriott, Richard Cox

Updated 28 February 2019

About half (mainly the rear half) of the human brain is devoted to processing visual information. Within that, the lower (ventral) section is primarily concerned with the analysis *what* (visual features – colour, size, shape…) and the upper (dorsal) section is mainly concerned with *where* (spatial location, relative positions of objects). "…visual information and spatial information appear to be processed differently and separately from each other" (Knauff, 2013).

The visual system is the product of millions of years of evolution from simple light sensitive cells to the complexity of the human eye. Clearly human vision did not evolve for data visualisation as this is a relatively recent practice. In order to understand the visual system's strengths and weaknesses for data visualisation we need to understand the purposes for which it evolved and how it works.

2.1 Overview of visual system

In humans, vision is the primary sense for perceiving the external environment. It is used for navigation, recognising friends, locating food and identifying danger, such as a crouching tiger. The human visual system has a computationally impossible job to do: from a 2D projection on the back of each eye it must reconstruct the shape and position of objects in the 3D world. Computer vision systems are still floundering on this task. Even worse the visual system needs to work quickly: when it comes to crouching tigers a few milliseconds can make all of the difference. As a result the human vision system has inbuilt biases and heuristics for recognising objects quickly: optical illusions reveal how these biases and heuristics can be tricked into making wrong deductions. Effective data visualisation takes advantage of these heuristics to allow the human visual system to quickly perceive patterns and groups.

The human visual system has 3 main level or stages:

1. Parallel processing to extract low level properties: colour, texture, lines and movement
2. Rapid serial processing divides the visual field into regions of similar colour or texture and achieves "proto-object" recognition of surfaces, boundaries and relative depth. This is driven both top-down by visual attention and bottom-up by low level properties.
3. Visual working memory: object recognition & attention, this is under conscious control

Figure 2.1: A dangerous hidden tiger License: Public Domain

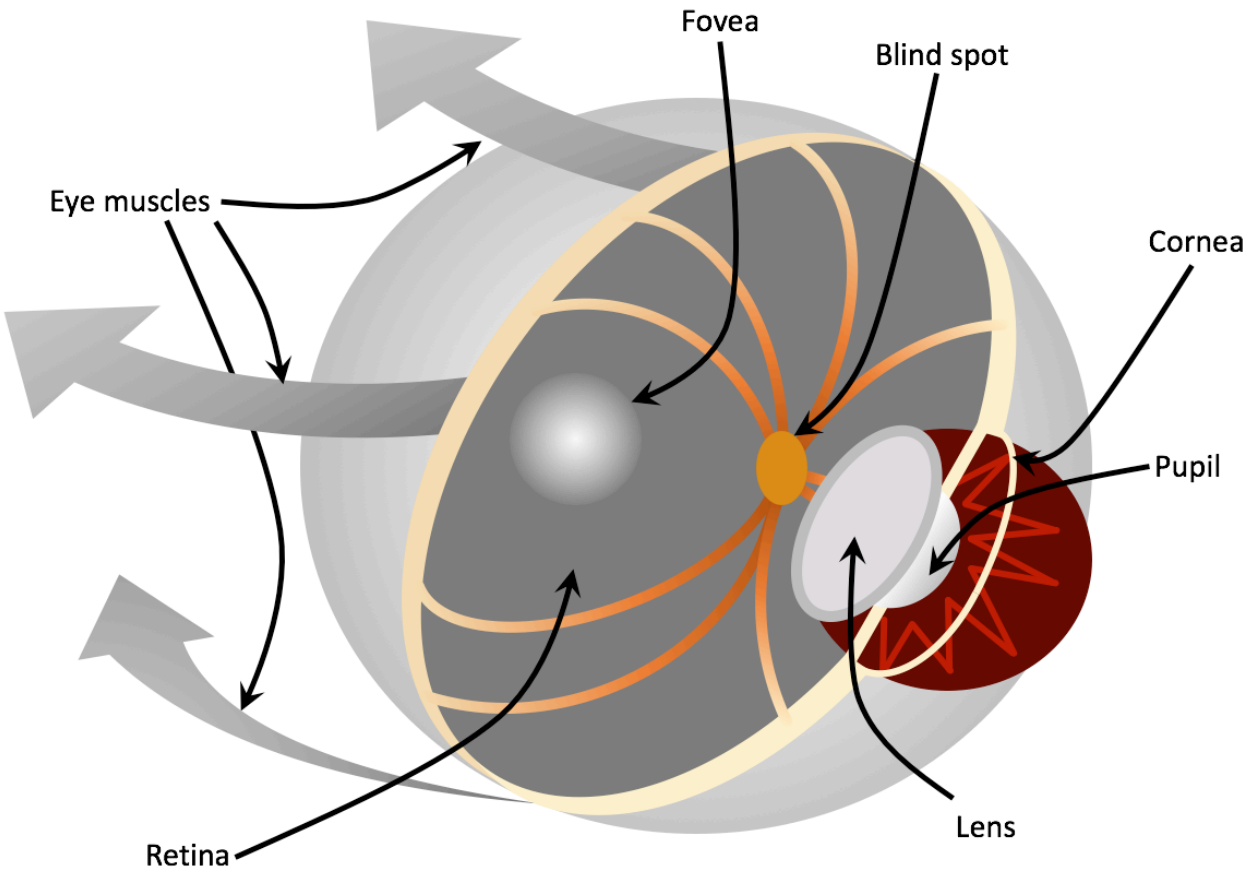


Figure 2.2: Main components of the human eye (Based on Figure 2.10 from Information Visualization – Perception for Design by Colin Ware, 2013)

2.2 The eye

Light enters the human eye through the pupil and then passes through the lens which focuses the (inverted) image onto the retina at the back of the eye. The retina contains two kinds of light sensitive cells: *rods* and *cones*. There are about 100-120 million rods and 6 million cones. Rods are very sensitive to light but only see in monochrome and are not very acute. Cones are less sensitive so do not work well at night but see colour and are more acute. Cones are primarily responsible for day-time vision.

Cones are not distributed uniformly across the retina. Most of the cones are in a small area called the *fovea* which is responsible for detailed vision.

- We see the image falling on the fovea clearly. This corresponds to about 2^o of vision which is about an area 2cm by 2cm at arms length
- The rest of the eye provides peripheral vision

While we believe that we simultaneously see all regions of a data visualisation in detail this is not really true: our eye rapidly darts around the image, fixating on a different region for a few milliseconds and then moving on. The visual system stitches these detailed images together to create an illusion that we see the whole graphic in detail. Nonetheless peripheral vision allows us to see a much larger region in coarse resolution and can direct attention to changes and movement.

2.3 Marks and channels

Graphics are made up of *marks*, the basic graphical elements such as a glyphs, lines and regions. A mark's visual appearance and spatial attributes such as position, shape and size are given by *visual variables*. Information graphics map data attributes to these visual variables. Low-level visual processing uses different neural pathways to process different visual variables. These pathways are often called *visual channels*. Different pathways are used to detect motion, orientation, texture, colour and size. This means that these channels are perceptually distinct.

Where possible different channels should be used to encode different attributes, rather than using the same channel such as colour to encode multiple attributes. It does not hurt to use redundant encoding.

Lines and shapes are recognised by specialised cells called Gabor receptors. Different receptors respond to different frequency and orientation of input lines. This means that symbols should be as distinct as possible from their background and from one another in terms of their components spatial frequency and orientation.

2.4 Colour

Colour is actually composed of three different channels. Cones provide colour vision: they come in three varieties each with a peak response to a different light frequency within the visible light spectrum. Low-level visual processing encodes these in terms of three opponent colour channels: red to green; blue to yellow and, the most important channel, black to white which encodes *luminance*.

In data visualisation it is common to think about colour in terms of the HSL colour space: H for hue-the choice of pure colour, S for saturation-the amount of white mixed with the colour, and L for lightness-the amount of black mixed with the colour. Another colour space that is common in computer graphics is the RGB system which codes colours in terms of the amount of red, green and blue. While HSL is not ideal it is a closer match to the actual perceptual system and should be used instead of RGB.

One thing to be aware of is that colours inhibit adjacent colours. This means that the same colour can appear quite different in different contexts. Boundaries between colours help this. You also need to be aware that the amount of area affects perception. Use low-saturation lighter colours for large background regions, higher-saturation darker colours for small foreground shapes or regions.

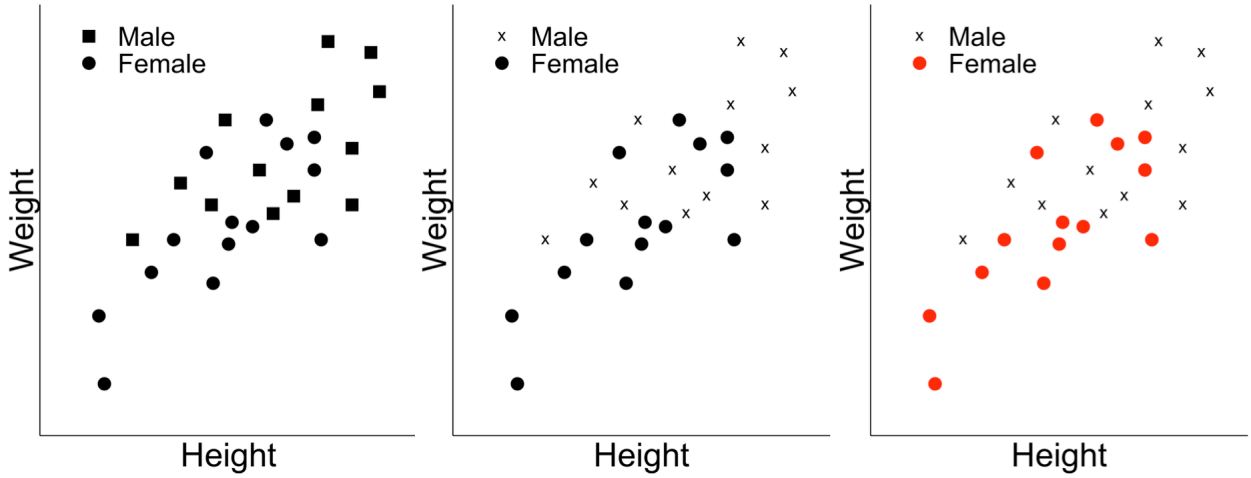


Figure 2.3: Feature channels can be used to make symbols more distinct from one another. Crosses are perceptually distinct to filled circles, so it is easier to separate males from females in the graph in the middle than the graph on the left. The graph on the right use redundant color coding in addition to more distinctive shapes making it even easier to distinguish the two sexes. (Based on Figure 5.8 from Information Visualization – Perception for Design by Colin Ware, 2013)

Colour choice is quite difficult. Luminance and saturation are automatically interpreted as ordered while hue is not. However hues that vary along only the red-green or blue-yellow channel do have a natural ordering. Fortunately many colour maps or palettes and tools have been developed to help in data visualisation design. One of the most commonly used is by Cindy Brewer. Her ColorBrewer tool enables selection of handcrafted color schemes for various tasks. Her colour schemes are also available for use in R (RColorBrewer, examples below). Her tool distinguishes between three different kinds of data: sequential (ordered but ascending from a single least value), diverging (ordered but ascending and descending around a neutral value), and qualitative (categorical). Ware (2013) also presents a number of colour maps.

Greens (sequential)

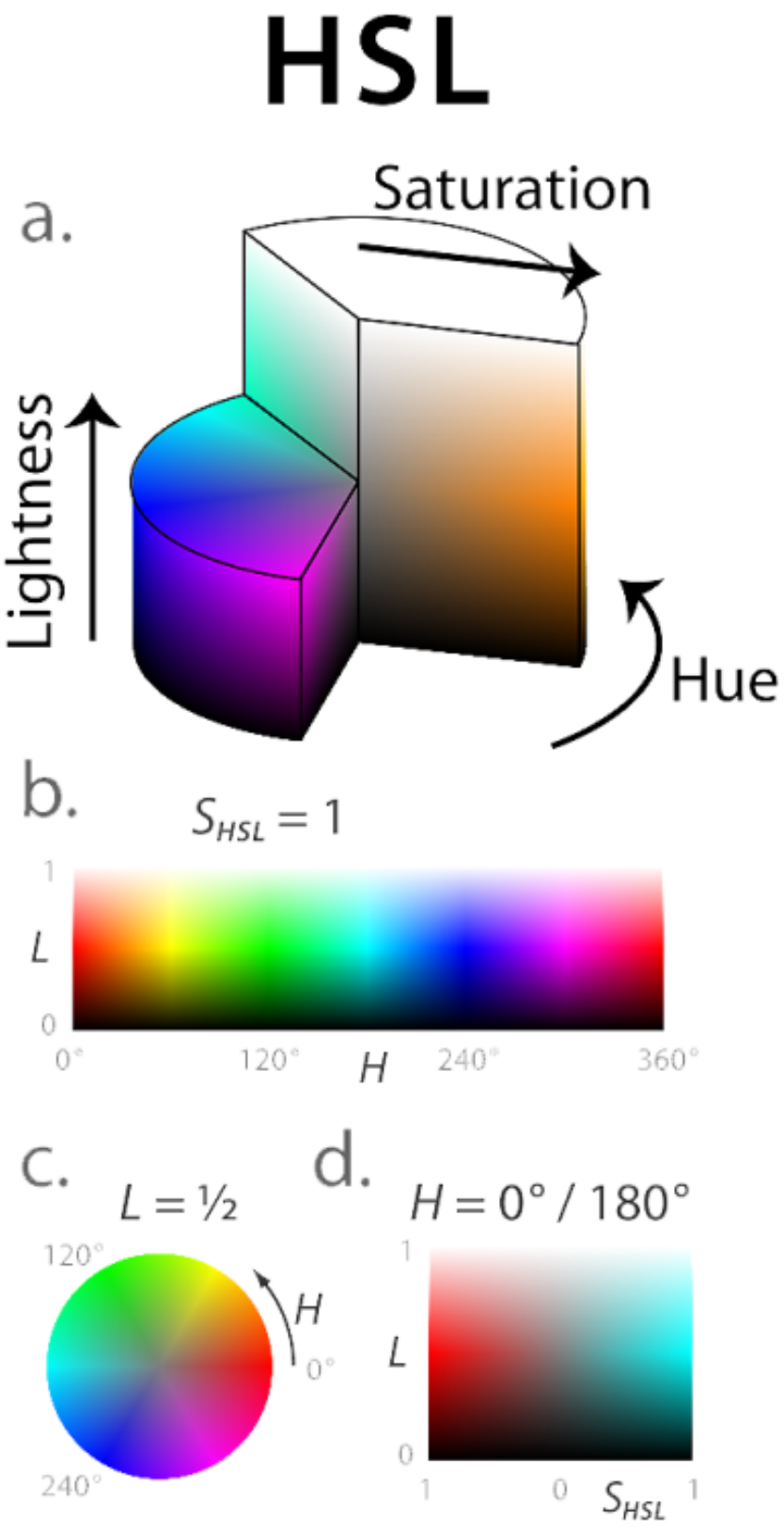


Figure 2.4: Top: cut-away 3D models of HSL (a); bottom: two-dimensional plots showing two of a model's three parameters at once, holding the other constant (b, c, d). (Based on figure 1 on HSL and HSV page of Wikipedia) License: CC Attribution 3.0 Unported (CC BY 3.0)

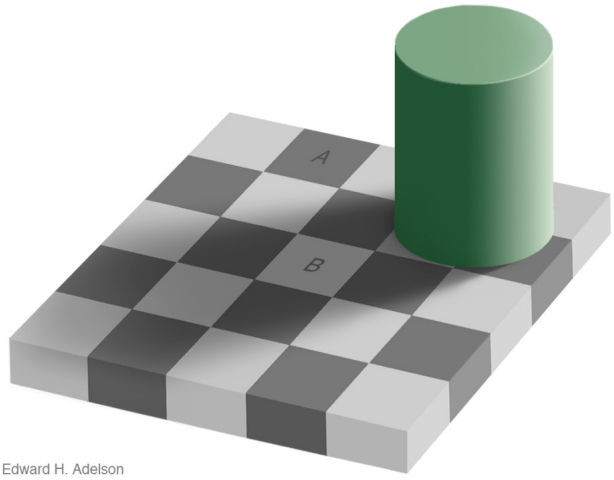


Figure 2.5: Colour inhibition: the colors of the squares labeled A and B are the same! If you don't believe this download the image and check the RGBs value in a photo editor. (This figure is from Edward H. Adelson)

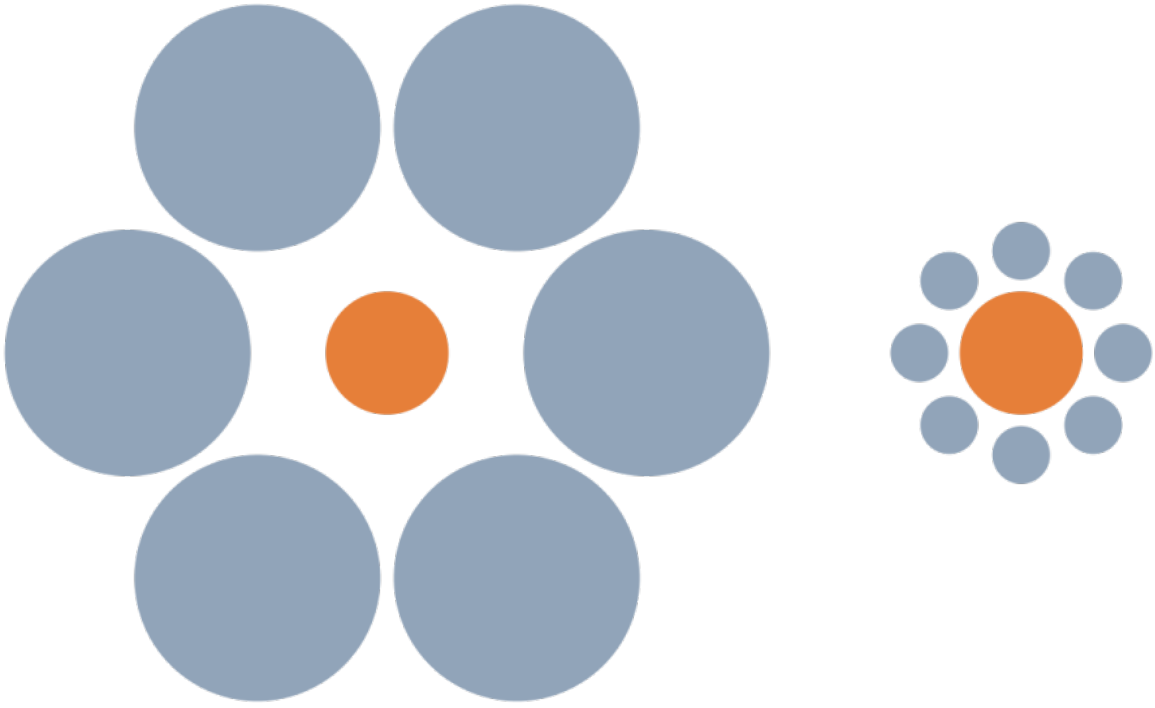
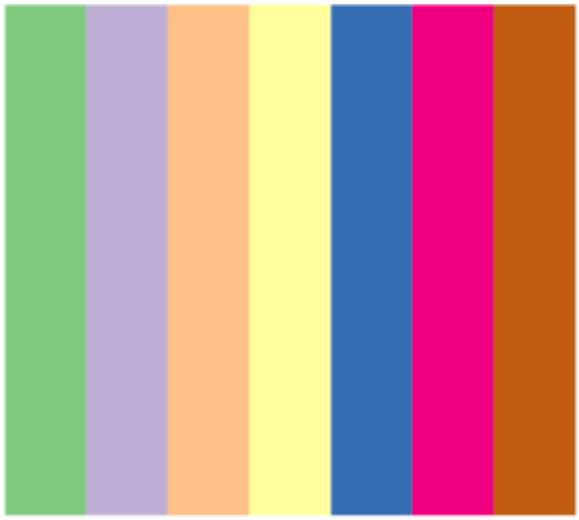


Figure 2.6: There is also size inhibition. Believe it or not the two orange circles in the center are in the same size. (figure is from Wikipedia) License: Public Domain

BrBG (divergent)



Accent (qualitative)

2.5 Colour-blindness & accessibility

One thing to aware of when using colour is that colour blindness is quite common. About 10% of males and 1% of women have some kind of colour blindness. The different colour channels explain the different kinds of colour blindness. Most commonly differentiation on the red-green channel is reduced (about 8% of men but much less common in women) while blue-yellow channel differentiation is much less common and not sex related.

When designing colour schemes the easiest strategy is to ensure that hue is not the only channel used to encode information. For categorical data choose colour maps that vary in luminance or saturation as well and if possible avoid colour maps that emphasise red-green. Sites such as http://www.color-blindness.com provide on-line tools to show what an image looks like with different kinds of colour blindness.

Maureen Stone an expert in the use of colour (who now works for Tableau) introduced the slogan *Get it Right*

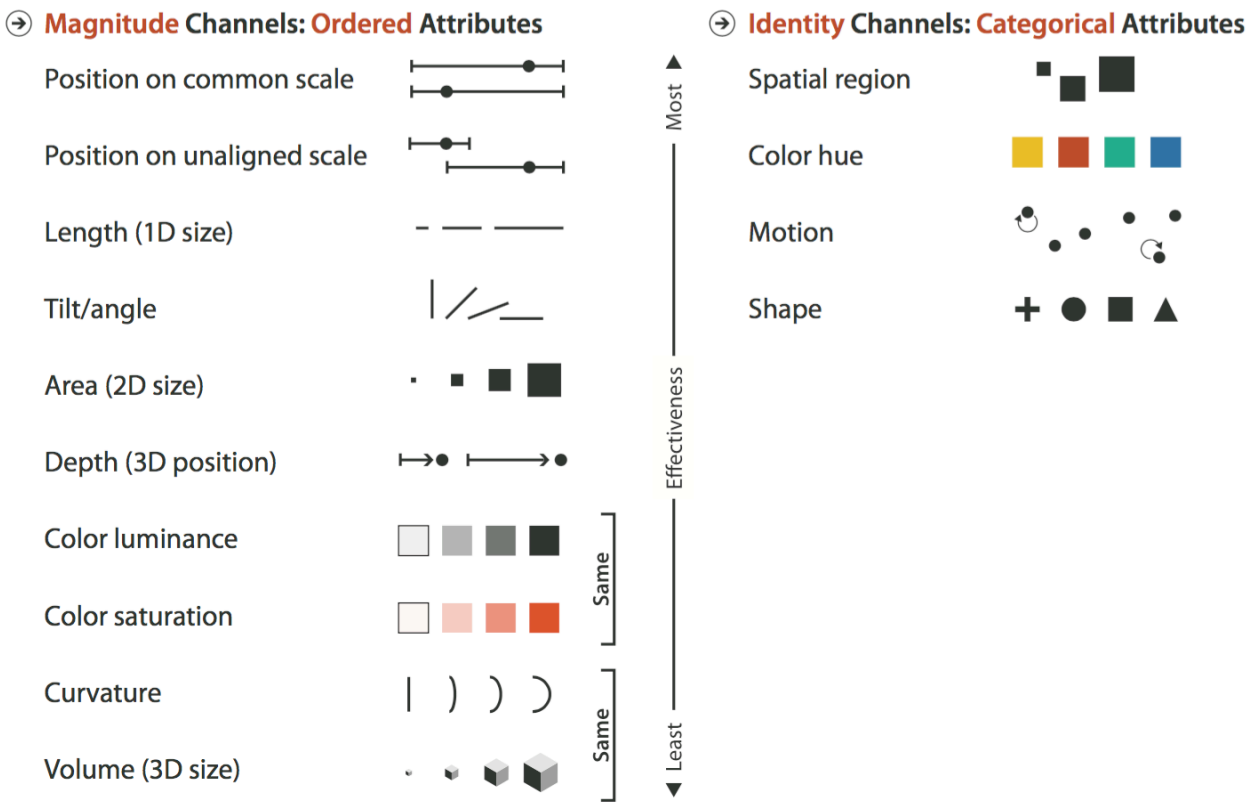


Figure 2.7: Effectiveness of different visual channels. (Fig 5.1 from Visualization Analysis and Design by Tamara Munzner, 2014) License: CC Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)

in Black and White. She suggests that you should develop visualisations in black and white first, ensuring that the important aspects of the visualisation are still legible when the image is rendered in greyscale. Hue and saturation, i.e. colour, is added later, to provide redundant or secondary information.

2.6 Which visual variable should I use?

Visual variables are not interchangeable: the same data attribute encoded using different visual variables will not be perceived as effectively. Different variables vary in terms of

- *salience* — how quickly they are noticed. For instance movement is more salient than orientation.
- *discriminability* — how many distinct values can you encode without confusion to the user
- *accuracy* — how easily can you compare different values.

Experiments have shown that the visual variables commonly used for encoding quantitative or categorical data vary greatly in accuracy and discriminability . The following figure summarises their effectiveness

What this means is that you should use bar charts rather than pie charts or doughnut charts and that if occlusion is not a problem then a prism map is more effective than a choropleth map for ungrouped data. And you should never, never use a 3D pie-chart!

Related to discriminability is the degree of "visual popout"– that is how well target items standout from the other items. Visual pre-attentive processing occurs in parallel and objects that are pre-attentively distinct from the other items are are found quickly and the time taken is independent of the number of non-target items. Without visual popout target items must be found using a conscious serial search through all items

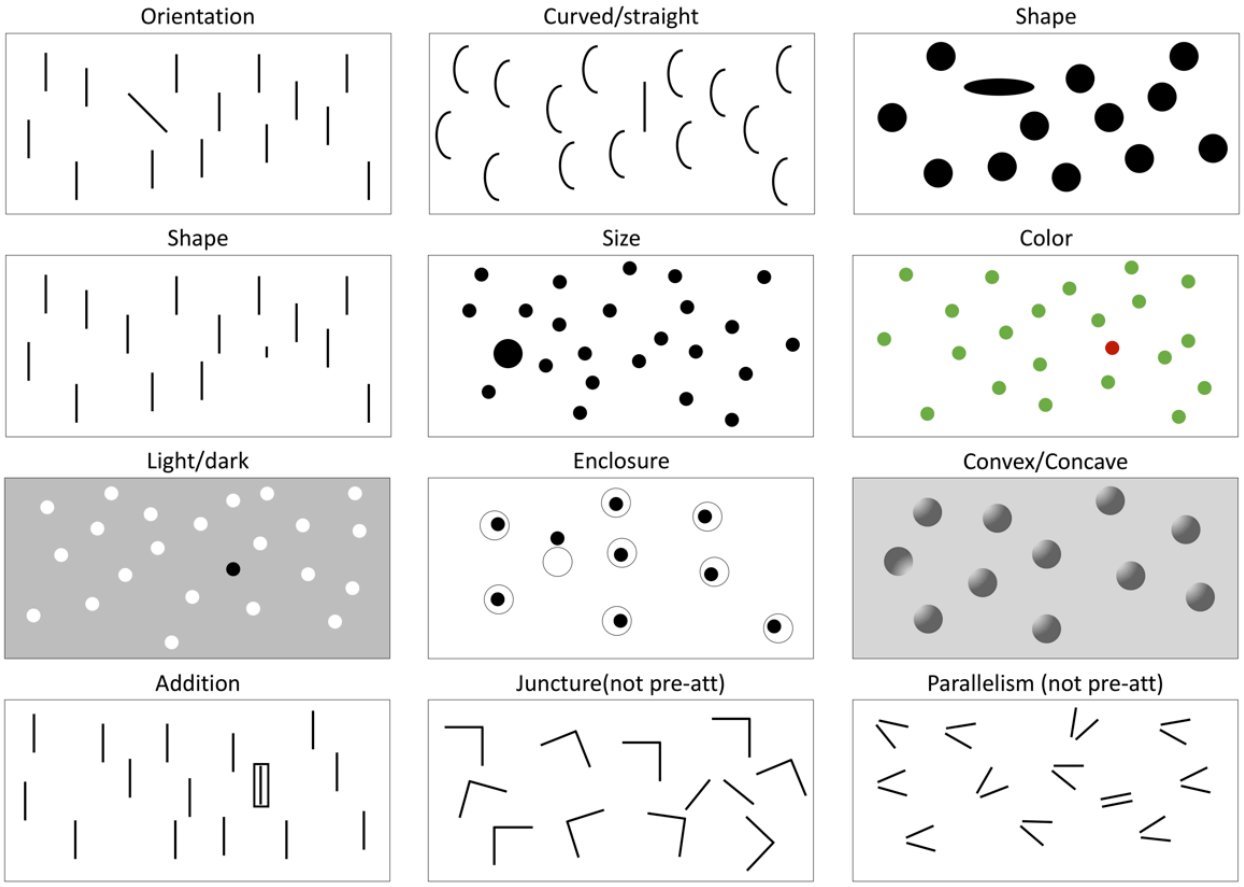


Figure 2.8: Examples of glyphs showing different kinds of visual pop-out. (Based on Figure 5.11 from Information Visualization – Perception for Design by Colin Ware, 2013)

and so the time taken to find target items depends upon the number of non-target items. Many channels support visual popout, at least to some extent. They include line orientation, length and width, size, curvature, spatial grouping, blur, annotation, color, motion and position. Pre-attentive cues should be used when directing or attracting attention or to show search results.

You cannot choose visual variables independently of one another as the underlying visual channels interfere with other to varying degrees. At one extreme are visual variables like colour and location that are *separable* in the sense that they have very little interference and at the other are variables like the red-green and yellow-blue colour channels that are integral and have high interference. All things being equal you should use visual channels that interfere as little as possible.

2.7 Grouping

Colour, line orientation and frequency, stereoscopic depth and motion are identified in the first stage of visual processing. In the next stage contours, regions and foreground and background are identified. This is the stage in which pattern perception is used to extract objects from low-level visual features.

Perception of visual patterns was first seriously studied in the early 20th century by a group of German psychologists who identified a set of laws of pattern perception which were called the *Gestalt laws* since Gestalt is the German word for pattern. Based on this research we now know there are many ways in which people automatically organise element

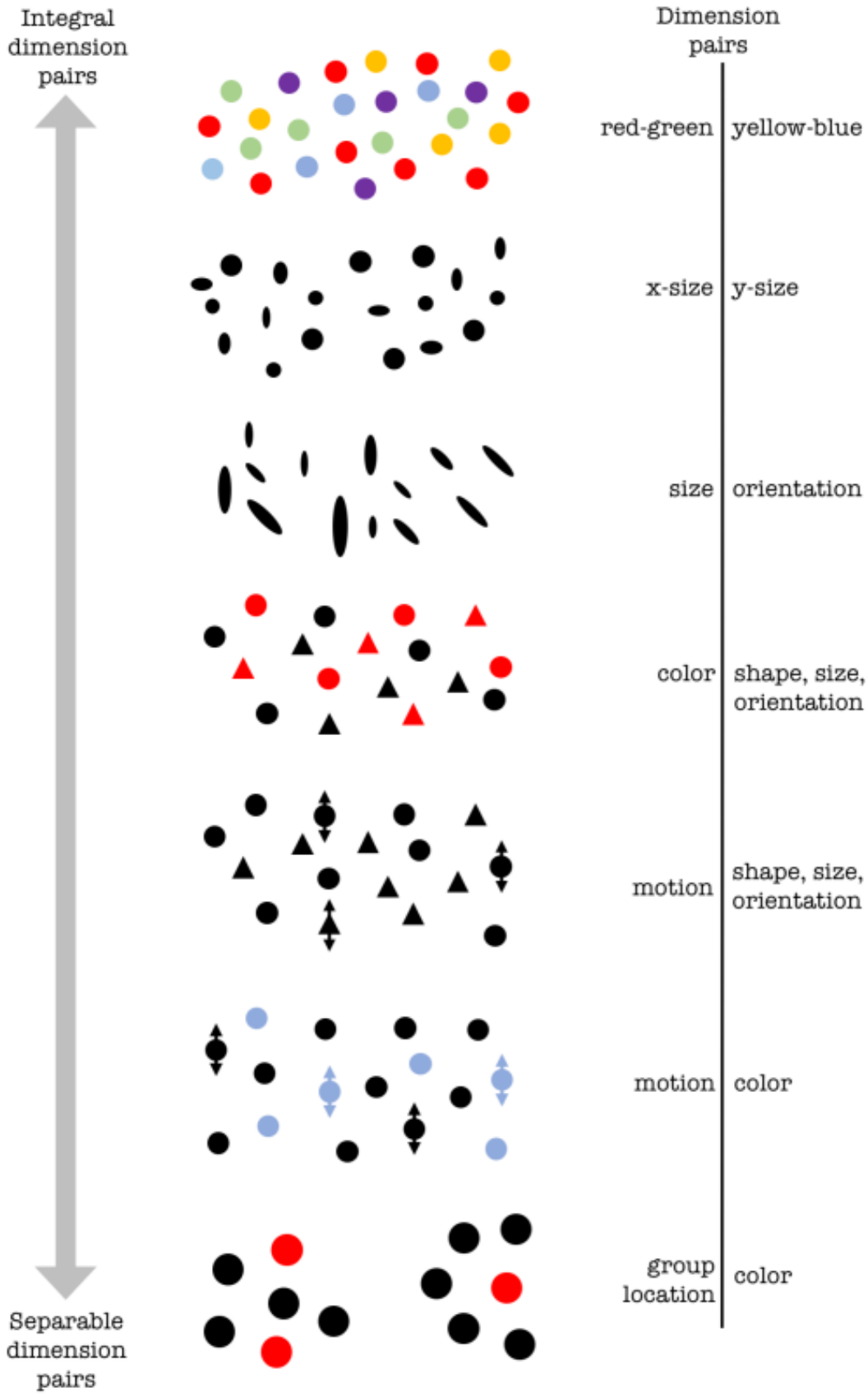
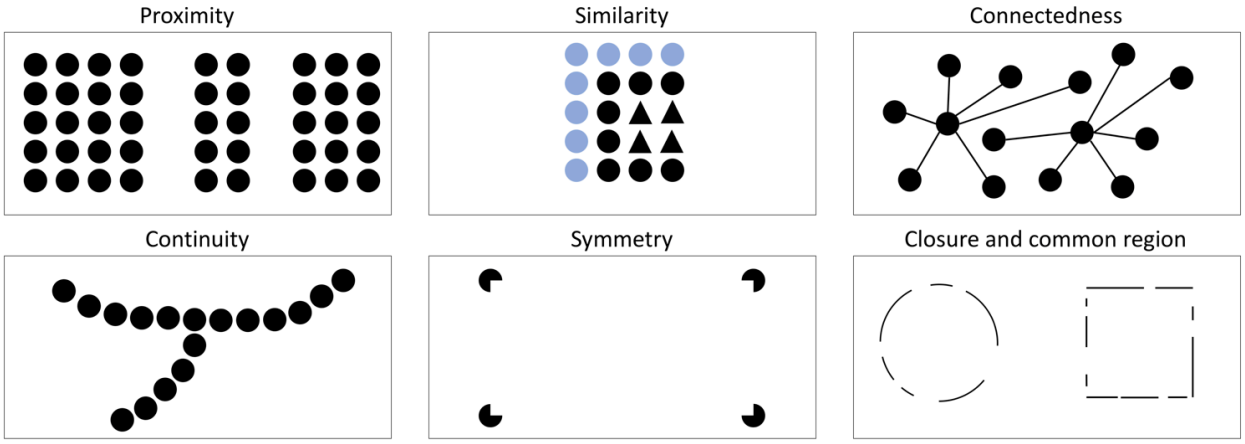


Figure 2.9: Examples of glyphs coded according to two display attributes. At the top are more integral coding pairs. At the bottom are more separable coding pairs. (Based on Figure 5.23 from Information Visualization – Perception for Design by Colin Ware, 2013)

- *Proximity*: Elements that are close together form groups.
- *Similarity*: Elements that are similar in some way such as colour or shape form a group
- *Connectedness*: Connection by lines is a powerful way of grouping elements
- *Continuity*: We tend to group regions and lines to form smooth and continuous shapes.
- *Symmetry*: We are good at recognising bilateral symmetry, especially around a horizontal or vertical axis and group the symmetric lines together to form an object.
- *Closure and common region*: we like to see closed contours and will mentally extend lines to close them. Being "inside" a closed contour is a very powerful grouping principle
- *Shared fate*: Elements that move together are grouped together.



diagrams_datasets/section2/shared-fate.1.gif

These principles capture the heuristic rules that the human visual system uses to group the lines and regions of similar colour and texture in 2D in order to segment the image into foreground and background and into different objects so that they can understand what they are really looking at in the 3D world.

Information graphics take advantage of these heuristic rules to help us see patterns etc. For example, scatter plots make use of proximity, node link diagrams make use of connectedness, Venn diagrams of common region and paired bar charts make use of symmetry.

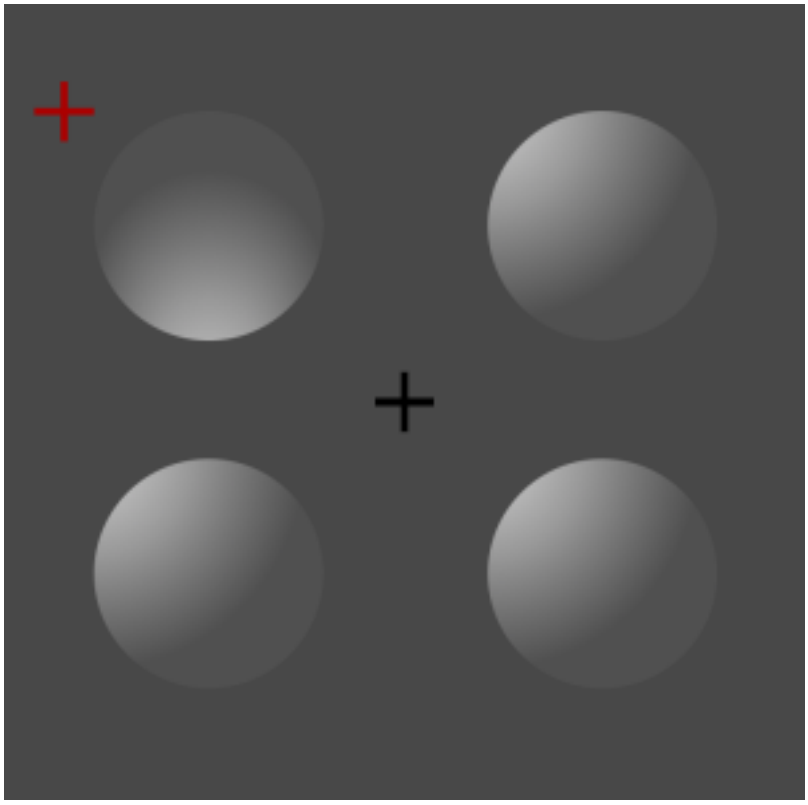
2.8 Perceiving 3D

An important part of understanding our 3D environment is the way in which the visual system extracts information about depth from what are essentially two 2D visual images, one for each eye. A wide number of different depth cues are used, most of which are now used to create lifelike immersive 3D visualisations.

Monocular static cues

- *Occlusion*: this is the most important depth cue–objects in front obscure those behind.
- *Linear perspective*: foreshortening, parallel lines converging to a point. We see the sides of the road converge and that people get smaller in the distance.
- *Shape-from-shading*: We see this illustrated below – here light from above is suggested by the shading. Concave or convex dimple shapes are suggested by shading. In the Angkor image above it used to show the shape of the elephant's head.

Figure 2.10: Scene from 3D animation constructed by Monash academic Tom Chandler showing how the Cambodian temple complex of Angkor may have looked. License: Copyright © Monash University, unless otherwise stated. All Rights Reserved.



- *Shape-from-texture distortion*: Wire frames make use of this to show shape

- *Cast shadows*: Cast shadows give a clue about height above the object on which the shadow is cast.
- *Familiar size*: Familiar objects allow us to judge distance because we know how big they actually are.
- *Depth of focus*: our eyes change focus to bring the image of the object we are looking at into sharp focus on the fovea. Objects that are closer or further away are blurred giving an ambiguous clue as to their depth.

Monocular dynamic (moving picture)

- *Structure from motion*: rotation and movement of an object relative to the observer allowing them to see it from different points of view is an extremely important depth cue.

Binocular

- *Vergence angle*: When the eyes look object at an object at a certain depth the visual system can make use of the difference in angle between the line-of-sight vectors of the two eyes to measure depth of objects that are close by (roughly within arm's length)
- *Stereoscopic depth*: The visual system can make use of small differences between the images on each eye to see depth. 3D TVs and displays provide stereoscopic vision. While stereoscopic vision (in combination with the other cues) can provide a sense of truly immersive 3D it is only one of many depth cues and is actually not that important. Something like 20% of people do not have stereoscopic vision and many never notice its absence.

Not all of these depth cues are needed to create realistic 3D graphics and may not be needed at all in some tasks. Ware (2013) provides a more detailed analysis. Occlusion is the most important depth cue. I think structure-from-motion is the next most important and can also mitigate the disadvantage of occlusion hiding information. Ware recommends that if structure-from-motion is used then so should occlusion, linear perspective and texture-distortion or else it looks strange.

Data visualisations also make use of more artificial depth cues to show depth. These include showing gridded ground and side planes to show perspective distortion. An extra cue is to projecting the 3D data onto these planes. In the case of 3D scatter plots it is common to drop lines to the ground plane so that the points look like pins.

An important question is when to 2D or not 2D? The general rule is that you should use as few dimensions as is required. Thus if if you are simply comparing the magnitude of a single attribute use only a single dimension and plot the values on a uniform scale. There is no need for 2D in this case.

In the case of 3D it should be used when visualising inherently three-dimensional structures such as buildings and other physical objects and flows. This is why immersive 3D is so important in scientific visualisation.

The use of 3D for abstract data visualisation is less easy to justify and by default you should use 2D. The disadvantage of 3D is that occlusion hides information and the perspective distorts size, making it difficult to compare magnitudes. Interaction is also more difficult. For this reason 3D bar charts are a very bad idea. However my view is that 3D will be used more frequently in abstract data visualisation when low-cost 3D visualisation technologies, such as the HTC Vive, Oculus Rift or zSpace, that allow the user to naturally vary their viewpoint become available. By allowing the observer to move relative to the graphic the problems of occlusion and perspective distortion are mitigated. They will be useful when looking at actual 3D visualisations like 3D scatter plots, prism maps, space-time cubes and congruent 2D surfaces drawn in 3D (sometimes called 2 1/2 D).

2.9 Visual attention and working memory

In the third and highest level of visual processing, visual objects are held in working memory while the viewer performs some task such as finding the shortest route between two cities. At this level processing is conscious and sequential. Only a few objects are held in memory at one time.

We use several types of memory in visual processing: *Iconic memory* (aka visual cache) which is essentially a very short-term snapshot of the image on the retina; *visual short-term memory (STM)* which holds the visual features of objects of immediate attention; *spatial STM* which holds the position/location of the objects; and *long-term memory (LTM)* which holds memories retained from previous experiences. There are similar kinds of memory for other modalities such as echoic and verbal working memory for sound. (e.g. Baddeley, 2007).

Visual working memory holds visual objects from long-term memory as well as those on the screen. Actually visual working memory is probably not distinct from long-term memory, it is simply the current activated long-term memories. We can also think of the visualisation on the screen as a different kind of memory: *external visual memory.*

One of the most surprising finds of psychologists has been how few objects can be held in our working memory, somewhere between 3 and 5, depending upon task. And we only remember 3-5 objects if we are concentrating, usually only 1 or 2 objects are remembered.

To most people this limited capacity seems extraordinary, as we feel as if we have a rich internal representation of the world we are seeing. This is however not true. *Inattentional change blindness* is a powerful demonstration of our lack of memory capacity. Because we remember so little, we do not notice large changes between what we see in one view and the next. Change blindness is graphically shown in this video. The experiment is about 1:40 into the video but I encourage you to watch the entire video.

The limited capacity of working memory has strong implications for visualisation. In particular it means that we are better to encode multiple attributes into one visual object rather than using separate visual objects for each attribute, since, if multiple data is integrated into a single object, more information can be held in visual working memory. For instance if we are examining wind direction, temperature and wind strength we are better off encoding this in a single glyph such as an arrow whose orientation gives the direction, colour the temperature and length or width the strength rather using three different glyphs.

Visual attention is the key to understanding how information flows between the different visual processing stages. As the viewer performs the task their attention turns to different parts of the image. When they move their eye to focus on a new region, subconscious parallel processing of stage 1 extracts low-level properties. Visual attention guides stage 2 processing to extract the surfaces and features of the objects that the viewer is now looking at and stage 3 processing recognises these objects and places those being attended to in working memory. However, visual attention may also be driven by stage 1: if a light blinks in peripheral vision, this will be noticed subconsciously ad the viewers attention will be drawn to it.

Controlling attention is a key-part of effective visualisations. You need to direct attention to the salient parts of the display.

A recent theory suggests that for tasks involving visual reasoning, the spatial aspects of a display are the most important – too much visual detail reduces performance (*visual-impedance hypothesis*, Knauff & Johnson-Laird, 2002). This is consistent with calls from some information visualisation designers (e.g. Tufte) to produce clean, minimalist displays from which irrelevant detail ('chartjunk') is eliminated.

2.10 Summary

In this topic we have investigated how the human visual system works. We have seen how it has 3 main stages: low-level feature extraction; region, depth and boundary recognition; and visual working memory and object recognition.

The design of good visualisations needs to take into account the perceptual and cognitive limitations of the visual system.

- Different visual channels should be used to encode different attributes and the choice of channel is important
- Low-level feature recognition occurs in parallel and supports "visual popout".
- Colour schemes and interfaces should be designed to cater for colour blindness.

- Pattern matching and grouping makes use of visual heuristics
- Use 1D in preference to 2D and 2D in preference to 3D unless there is a strong reason not to.
- We have extremely limited working memory: attention should be directed to salient parts of the display.

FURTHER READING

The material in this topic is mostly based on

Munzner, Tamara. *Visualization Analysis and Design.* CRC Press, 2014.

Ware, Colin. *Information visualization: perception for design (3rd Ed.).* Elsevier, 2013.

Further reading is

- Chapter 10 of Munzner, 2014.

Chapter 3

Visual communication

By Kimbal Marriott, Richard Cox

Updated 9 May 2017

Understanding the human visual system is one part of understanding how to create effective data visualisations. Another component to this is understanding human language and how humans communicate. This aspect is important when designing data visualisations for communicating what you have found to stake holders as they will subconsciously understand the data visualisation as a "communication act." In this topic we look at this aspect of visualisation design.

3.1 Effective communication

Most cognitive psychologists and linguists now believe that humans have an innate instinct to learn and use spoken language. All human languages share a common "deep" structure and children who are deaf and so not exposed to spoken language will, if they are given the opportunity, spontaneously invent sign languages so that they can communicate. Amazingly although sign language is understood visually and produced using hand gestures, the same parts of the brain are used for understanding and producing sign language as are used for spoken language. Humans really want to be able to talk to one another.

When we communicate with each other there are a number of assumptions that are true for spoken language, written language, graphic novels, movies and sign language:

- *Relevance*: The information that is provided is relevant to whatever is being discussed and that neither too much or too little information will be provided.
- *Appropriate knowledge*: We tailor the communication to the listener or reader, taking into account their knowledge and also cultural expectations.
- *Directing and holding attention*: The discussion is sequential, with clear indications as to what is important and when there has been a change in topic.

Paul Grice, a philosopher of language, proposed several 'conversational maxims':

- the maxim of quality – provide information that is true, do not say that which you know to be false or for which you lack sufficient evidence
- maxim of quantity – provide enough information as is required but not more than is required
- maxim of relevance – ensure that the information you provide is relevant
- maxim of manner – avoid obscurity, ambiguity, be brief and be orderly

These points hold equally true for the visual language of data graphics. They mean that you need to be very clear about what message you are trying to convey when designing a data graphic for communication and who your intended audience is. It means only showing data that is relevant to that message and choosing

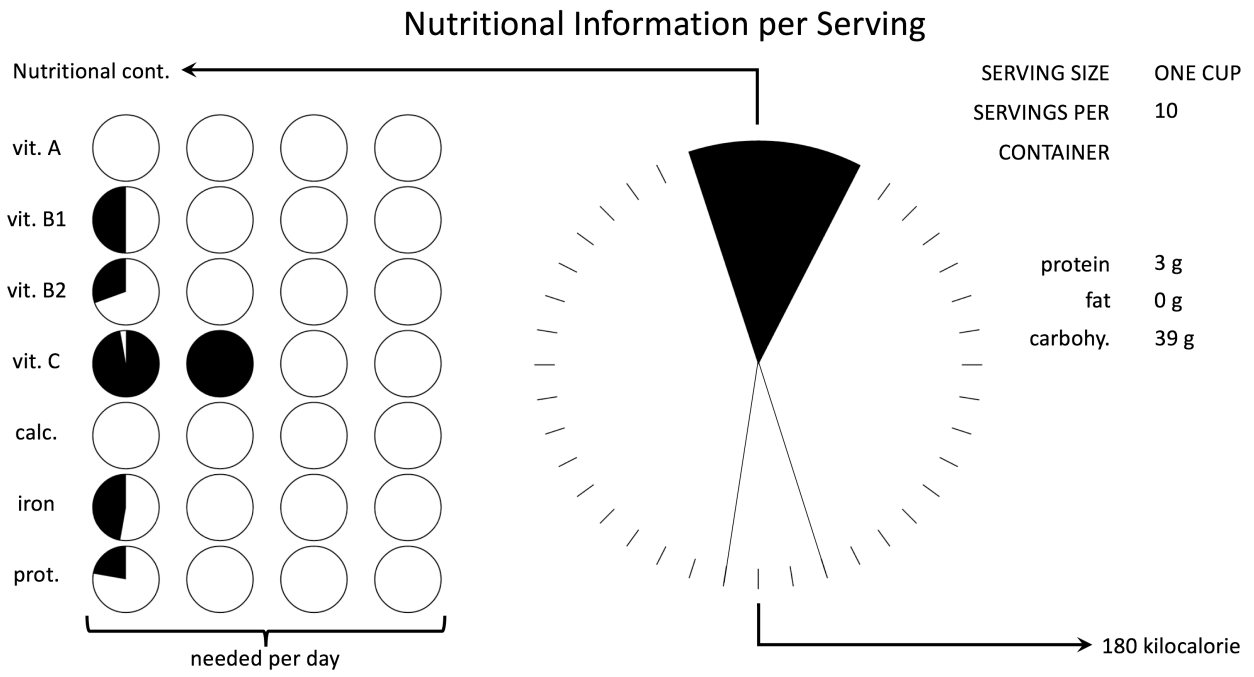


Figure 3.1: Proposed nutritional display graphic. (Based on Figure 1.8 from Graph Design for the Eye and Mind by Stephen M. Kosslyn, 2006)

charts and graphics that the user is familiar with, or at least ones which they can easily understand and providing titles and legends etc that explain what the graphic is about. and how to read it. It means using visual tricks to direct attention to the important part of the graphic. The reader's attention will be drawn to the most visually striking components of the graphic so these should be the most important parts of the message, i.e. highly salient. We have seen from the last topic how to make parts of a display pop-out. Another technique is to use annotations to direct attention. Readers expect changes in appearance from, say interaction, to be relevant and informative.

In many situations a mix of graphics and text is required in order to communicate more complex messages. Relevance, appropriate knowledge and narrative need to be considered for the overall mix, with clear visual signals guiding the reader on which are the most important elements and in what order to read them.

When designing an effective data visualisation it is crucial to take into account the limitations of the human visual system and cognitive system discussed in The human visual system. In particular you should choose visual variables and representations that facilitate

- *Discriminability*: allow the reader to easily discriminate and compare data
- *Perceptual organisation*: encourage the perceptual system to group data in meaningful ways, such as by using Gestalt principles and different channels
- *Compatibility*: The conventions used are compatible with what they represent, e.g. use line graphs for continuous data, bar charts for discrete measurements and "more" of a visual variable such as length means "more" of the underlying quantity being conveyed.
- *Capacity limitations*: take into account limitations in working memory and processing limits.

Kosslyn (2006) presents a graphic that was designed to present nutritional information to consumers on food packaging. Apparently the US seriously considered making the use of this display mandatory. Please take a look at the display and see if you can understand it.

He identifies the following problems

1. The reader is likely to think that the small circles are pie charts but in this case it is the group of 4

circles that shows the whole. This violates the assumption that the reader has appropriate knowledge.
2. The circles are arranged so that they are perceptually grouped into 4 columns instead of 7 rows when in fact the rows are the logical way to group the circles.
3. The center panel also violates the assumption of appropriate knowledge as it is a graphic that the user has almost certainly never seen before and probably hopes never to see again. While it looks like a pie chart it is not and actually consists of two separate graphics: a black wedge at the top giving the nutritional content (vitamins+minerals) and white wedge at the bottom giving the number of calories. Even though the two wedges are aligned and on the same background and so perceptually grouped together that are in fact logically independent. It is impossible to understand the scale being used for each wedge and it is exceedingly difficult to know how many tick marks the black wedge subsumes.
4. It is also very difficult to know how to read the display and which pieces of information are the most important.

In terms of the Grician maxims, the graphic violates those of manner (it is obscure) and quantity (it provides more information than is required).

3.2 Narrative visualisations

Online interactive graphics are increasingly used by journalists and data scientists to communicate the patterns and trends they have found in data. Publishing to the web is made easy by tools like Tableau and with D3 it is possible to create powerful and sophisticated visualisations. Such graphics blur the distinction between exploration and communication and data visualisation researchers and graphic designers are still exploring different ways of using interactive graphics for communicating their findings.

Good examples of these new kind of data visualisation are those being created by the New York Times. Take a look at Budget Forecasts, Compared With Reality from 2010. This explores quite a complex topic: budget forecasting and the reasons why it mostly goes wrong. It uses what is essentially a slide show with interactive slides to guide the reader through the story. Linked text and visualisation along with the linear slide show make it clear in what order to read the visual elements and what is important. Interaction allows the reader to explore the data but only in a carefully controlled way.

Segel and Heer (2010) provide an overview of such *narrative visualisations*. They identified seven genres: magazine style, annotated chart, partitioned poster, flow chart, comic strip, slide show, and film/video/animation. These differ in the number of frames and the ordering of the frames. The genres can be combined. For instance, the Budget Forecasts example uses annotated graphs within a slide show format.

They discussed the many different kinds of narrative tactics used in these visualisation, some of which are borrowed from movie making. These visual devices help the reader navigate the visualisation and to appropriately focus attention. They include highlighting to direct attention to salient parts of the display, transition guidance to help the reader move between the different elements in the visualisation without disorientation such as smooth animations, camera motion and visual structuring that helps the reader understand the overall structure and where they are in the narrative.

The final aspect that Segel and Heer (2010) considered is the visualisation structure. This varies in how much the author directs the story and how much freedom the reader has to explore the data to create their own interpretation. The Budget Forecast example is called an *interactive slideshow with single-frame interactivity*. In this structure the overall structure is linear but the reader can explore the data in a single side. They are also free to step backwards and forwards through the narrative. Another well-known example of this kind of visual narrative is Gapminder Human Development, 2005.

Another visualisation structure is the *martini glass*. This begins with questions, observations, or written articles to introduce the visualization but then the narrative expands to allow the reader to freely explore the data (hence the amount of exploration allowed expands like the shape of a martini glass). An interactive chart with associated text is a common example of this kind of narrative.

The final structure they identified was the *drill-down-story*. This presents a general theme but allows the reader to interactively choose different aspects to the story and drill down into the data. Interactive posters or maps often have this structure. Comparison of Bear Markets, The New York Times, 2008 is an example.

3.3 Animation & interaction

It is easy to believe that animation is better than multiple static frames at showing changes. However, it turns out to be more complex. Studies referenced by Ware (2009) suggest that, at least for showing how mechanical devices work, like a flushing toilet, snapshots of the key stages with careful annotations directing attention to changes and direction of movement work more effectively than animations. This is probably because: (1) the snapshots carefully direct and guide the reader's attention, (2) it is easy for the reader to compare different steps and to move backwards and forwards between these, and (3) they encourage the viewer to "mentally animate" components between the different steps which helps deeper understanding,

As a result of these studies, visualisation designers now realise that they need to be more careful when designing animations and changes resulting from interaction. They need to carefully direct attention, use visual continuity to help the reader preserve their mental map of the visualisation, stagger changes so that not everything changes at once and use carefully designed animations to show how the elements move. These tricks are the kinds of visual narrative tactics identified by Segel and Heer.

3.4 Summary

The design of effective visualisation for communication is quite difficult and requires understanding the human visual system as well as conventions used in communication. The following summarises the design process:

- The key first step is to to clearly identify what message you wish to communicate and to whom.
- The second step is to decide on the genre, the narrative structure and the presentation technology.
- The third step is to design an appropriate visualisation. Do mock-ups of different designs to explore the design space. Critique the designs taking into account: relevance, appropriate knowledge, directing and holding attention, discriminability, perceptual organisation, compatibility and human capacity limitations.
- The last step is to implement the visualisation and actually check that it is effective by testing it, if possible, with members of the target audience and refining the design

---

FURTHER READING

This topic is based on

Ware, Colin. *Information visualization: perception for design (3rd Ed.)*. Elsevier, 2013.

Kosslyn, Stephen M. *Graph design for the eye and mind*. Oxford University Press, 2006. The principles identified in human communication for good data visualisation design are based on his eight principles of effective graphics.

Segel, Edward, and Jeffrey Heer. Narrative visualization: Telling stories with data. *IEEE Transactions on Visualization and Computer Graphics*, 16, no. 6 : 1139-1148, 2010.

Further reading

- Segel & Heer, 2010.

Chapter 4

Activity: Basic web development skills

By Minyi Li, Yalong Yang

Updated 28 February 2019

To start coding with D3, you need to be equipped with some web developing skills, which includes HTML, CSS, SVG, DOM, Javascript.

We will start with some basic information about these first before getting to D3.

4.1 Prepare the tools

You need a text editor to write the code. http://brackets.io/ is a good option, as it has the "live preview" function, which saves you the trouble of setting up a static server. Of course, feel free to use other text editors, like Sublime Text.

Another option is to just use Firefox browser instead of Chrome, as the later versions of Firefox can access local files directly.

4.2 HTML

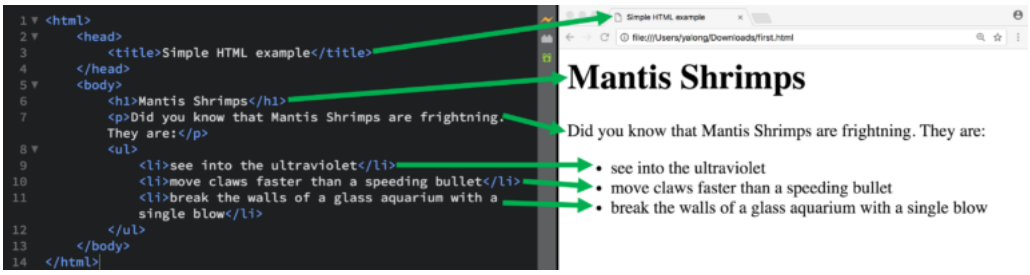
HTML is short for Hypertext Markup Language. It is used to structure content for web browsers. HTML elements (represented by tags) are the building blocks of HTML pages. Let's start with an example:

1. Open your text editor, create a new file using the menu (usually should located at left-top): **File -> New**.
2. Save the file as html using the menu: **File -> Save As**, let's use the name first.html.
3. Copy and paste the following code into your text editor, and save it by using the menu: **File -> Save**.
4. If you are using Brackets, click the "**live preview**" button to look at the very first HTML page.

```
<html>
    <head>
        <title>Simple HTML example</title>
    </head>
    <body>
        <h1>Mantis Shrimps</h1>
```

```
        <p>Did you know that Mantis Shrimps are frightning. They are:</p>
        <ul>
            <li>see into the ultraviolet</li>
            <li>move claws faster than a speeding bullet</li>
            <li>break the walls of a glass aquarium with a single blow</li>
        </ul>
    </body>
</html>
```

Now, let's link our code to the content of the page in your browser.



Actually, in this page, we used quite a lot HTML elements:

- title -> page title
- h1 -> header with large font size, try to play around with h2, h3 and etc.
- p -> paragraph
- ul -> bullet points list
- li -> list items

There are many other HTML elements there: https://developer.mozilla.org/en-US/docs/Web/HTML/Element, you do not need to know all of them at the moment.

All HTML elements can be assigned attributes using **property/value** pairs in the opening tag.

Let look at another very useful HTML element – link(**a** element):

```
<a href="http://d3js.org/">The D3 Website</a>
```

In the **a** element, we use a **property/value** pair to define a link in the webpage href **property**, with the **value** of http://d3js.org/.

Try to add this to the end of the **ul** element of our first HTML page, and save the page. If you are using Brackets with "live preview", you should notice the change once you save the file.

Different HTML elements have different properties, you can look up them on the previous link when you need them. All elements have the **id** and **class** properties.

The **id** property specifies a **unique** id for an HTML element (i.e., the value must be unique within a HTML document among all HTML elements).

The **class** property specifies one or more **classnames** for an element (usually used together with css which we will introduce later). Different HTML elements can be assigned the same class to present they are in the same group for css processing.

4.3 CSS

CSS is short for Cascading Style Sheets, and is designed to describe how the contents of a web page should be presented.

At the very beginning of web, there was no CSS, but people still needed to specify the style of a web page. So they wrote everything in the HTML file with the "style" property for HTML element. This works all right for small web pages, but for complex web pages such as a website with thousands of web pages, it is very hard work to maintain all of the style properties.

So, W3C invented CSS to allow the **style** and **content** of web pages to be managed separately.

Let's make our HTML page a bit more colorful using our first CSS file:

1. Using **File -> New** to create a new file, and save it in the same folder with **File -> Save As**, let's use the name first.css.
2. Copy paste the css code into your CSS file, and save it: **File -> Save**

```
h1,p { color: pink;
    font-family: Arial;
}
li {
    color: green
}
```

3. Link your style file in your HTML page (usually within the **head** section)

```
<html>
    <head>
        <title>Simple HTML example</title>
        <link rel="stylesheet" href="first.css">
    </head>
    <body>
        <h1>Mantis Shrimps</h1>
        <p>Did you know that Mantis Shrimps are frightning. They are:</p>
        <ul>
            <li>see into the ultraviolet</li>
            <li>move claws faster than a speeding bullet</li>
            <li>break the walls of a glass aquarium with a single blow</li>
        </ul>
        <a href="http://d3.js.org/">The D3 Website</a>
    </body>
</html>
```

4. Refresh your web page, you should see the result as the following image:

Now, it is the time to explain how it works.

There are two parts of a CSS rule: **selector** and **properties**.

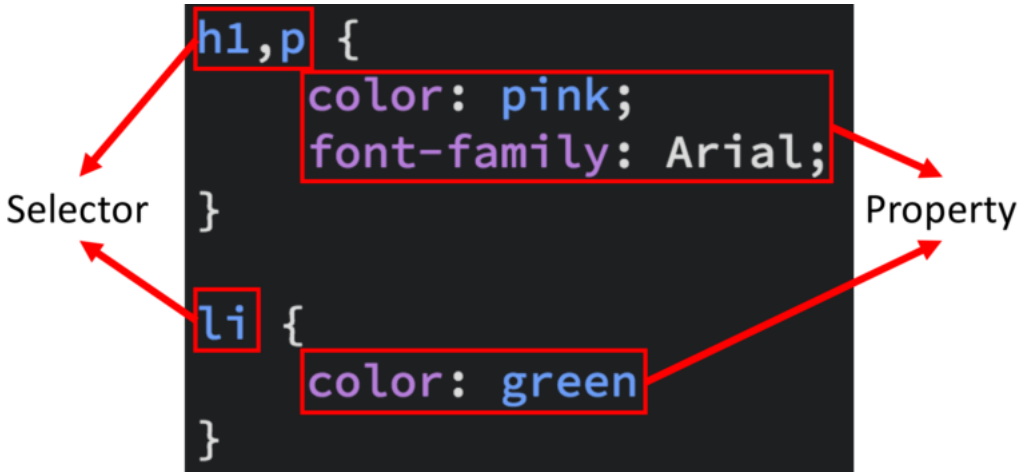
Selector, basically chooses the HTML elements to operate. There are many types of selectors to help you select the HTML elements you want easily. For example:

Type selectors	h1 /* selects all level 1 headings*/

Descendent selectors:	p em /* selects all emphasized text in a paragraph*/
Class selectors:	.axis /* selects all elements with class axis*/
Class selectors:	.axis.y /* selects all elements with class axis and class y */
ID selectors:	#L1 /* selects element with ID "L1 */

For detailed explaination of CSS selectors, please refer to http://www.w3schools.com/cssref/css_selectors.asp.

Properties, basically defines the styles of HTML elements. Different HTML elements also have different styles you can change. Usually you can play around with margin, padding, color, size, positions and etc.



There are also some existing CSS libraries to help you style your HTML more efficiently:

https://getbootstrap.com/

https://purecss.io/

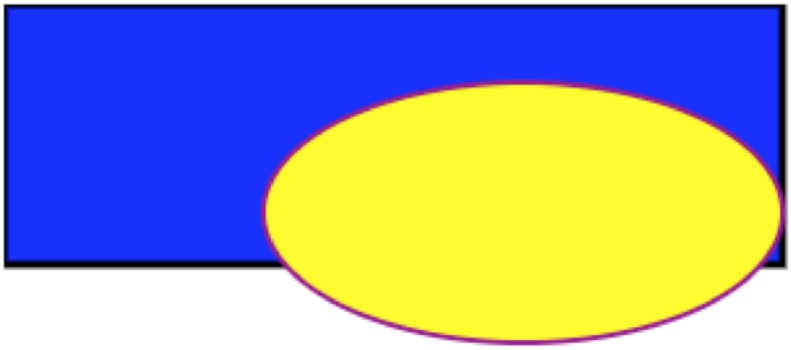
In the above example, we use CSS file and HTML separately, however, there are ways to keep them in just HTML file. As the original purpose of CSS is to keep responsibility clear for each other, we would recommend you to keep them separate. You may find many online examples with them in one HTML though as this is makes things easier to explain.

4.4 SVG

Scalable Vector Graphics (SVG) is the web vector graphics format. It is designed to work with both HTML and CSS, and so it can be directly included in the document. You can also consider SVG as a type of HTML element.

A SVG canvas is the container of standard graphic primitives: **line, circle, text, ellipse, rect, path**. A more comprehensive list is at: https://developer.mozilla.org/en-US/docs/Web/SVG/Element .

The coordinate system of SVG canvas has origin (0,0) at top left corner, with the positive x-axis pointing towards the right, the positive y-axis pointing down, and one unit in the initial coordinate system equals one "pixel". Following is an example SVG, where a blue rectangle overlapped with a yellow ellipse is coded into a HTML page:



```
<!DOCTYPE html>
<html>
    <body>
        <svg width="400" height="110">
            <rect width="300" height="100" style="fill:rgb(0,0,255);stroke-width:3;stroke:rgb(0,0,0)"/>
            <ellipse cx="200" cy="80" rx="100" ry="50" style="fill:yellow;stroke:purple;stroke-width:2"
        </svg>
    </body>
</html>
```

Some of the styles (e.g. fill, stroke, stroke-width) of SVG are assigned with the **property/value** HTML way. For more SVG styles, please refer to https://www.w3.org/TR/SVG/styling.html

4.5 DOM

The Document Object Model (DOM) is a cross-platform and language independent API. In DOM, each bracketed tag is an element. Elements are defined as objects. Each element could have an relationship to another element, which could be expressed in human terms: parent, child, sibling, ancestor, and descendant.

It is the standard for Web browsers to parse HTML and make sense of a page content.

DOM also defines:

- ***Properties*** of all HTML elements;
- ***Methods*** to access all HTML elements; and
- ***Events*** for all HTML elements.

4.6 Javascript

JavaScript is a scripting language widely supported by web browsers. Javascript has a full API to access the DOM, so it can dynamically operate (deleting, updating, inserting) HTML elements. It also monitors user interactions in the webpage, e.g. hovering on an element, click on an element etc.

1. Let's create another file in the text editor, called **first.js**. Save it.
2. Link this first javascript file in our previous HTML page. Now the HTML should look like:

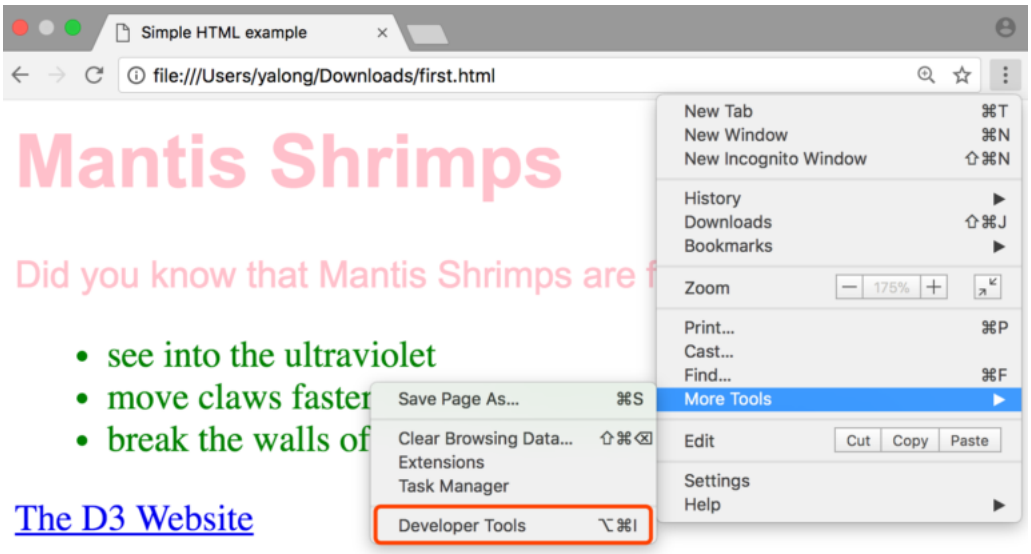
```html
<html>
    <head>
        <title>Simple HTML example</title>
        <link rel="stylesheet" href="first.css">
    </head>
    <body>
        <h1>Mantis Shrimps</h1>
        <p>Did you know that Mantis Shrimps are frightning. They are:</p>
        <ul>
            <li>see into the ultraviolet</li>
            <li>move claws faster than a speeding bullet</li>
            <li>break the walls of a glass aquarium with a single blow</li>
        </ul>
        <a href="http://d3.js.org/">The D3 Website</a>

        <script type="text/javascript" src="first.js"></script>
    </body>
</html>
```

3. Start with a simple test, it is now time to introduce you a very important event in web development **window.onload**, which fires after the whole web page is loaded, somtimes this is the time you want to start your script. Open first.js, and add the following code to the file:

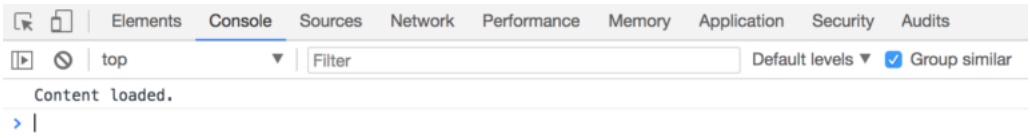
```
window.onload = function (){
    console.log("Content loaded.");
}
```

The Brackets editor may promote you that there are errors with the above code. However, this is because they cannot recognize the pre-defined variables in browser. The errors appear in the Console of web developer tool (we are going to introduce next) are real errors. Some of the errors in the text editor can be ignored.

4. After refresh the page, you may find nothing happened, well, that's true on the webpage, but something happened at the backend. We need to see the backend as a programmer, if you are using Chrome with Brackets, you can open the web developer tool by:



5. Switch to the Console tab, and you should see out text there (Console is similar to *print in python* and *System.out.println in JAVA*, and it is one of the most important tools to debug your code):



6. Let's try something more interesting, like hovering on the title to change the text content.

```
window.onload = function (){
    // Using DOM to Find the title HTML element
    // More details available at: https://developer.mozilla.org/en-US/docs/Web/API/Document
    var titleElement = document.getElementsByTagName("h1")[0];

    // Change the text to FIT5147 when mouse hovering on the h1 title
    titleElement.onmouseover = function(){
        titleElement.innerHTML = "FIT5147";
    };

    // Change the text back to Mantis Shrimps when mouse leaves the h1 title
    titleElement.onmouseout = function(){
        titleElement.innerHTML = "Mantis Shrimps";
    };
}
```

There are only two events in the above example, there are many more events there: https://developer.mozilla.org/en-US/docs/Web/Events, web user interactions are basically based on all these events.

If you want to know more details about Javascript, the W3C school is a good start: https://www.w3schools.com/js/default.asp.

Now its time to learn D3!

Chapter 5

Activity: Creating visualisations with D3

By Minyi Li, Yalong Yang

Updated 1 May 2018

5.1 An Introduction to D3

As we already introduced in the preceding Activity. D3 is a very powerful Javascript library for creating online interactive visualisations.

We would like to share the visualisations you can create with D3 again: https://github.com/mbostock/d3/wiki/Gallery and the free comprehensive online D3 tutorial by Scott Murray.

Again, you need a text editor for writing the scripts, and you need to figure out a way to access local files (more details in the previous activity).

5.2 Creating visualisations with D3

2a. Drawing SVG primitives

1. Create a folder to store our files, let's name it "second".
2. Let's create three files: second.html, second.css, second.js.
3. We should start with the HTML file, do you still remember how to link css and javascript file in HTML?

```
<html>
    <head>
        <title>D3 SVG Primitives</title>
        <link rel="stylesheet" href="second.css">
    </head>
    <body>
        <svg></svg>
        <script type="text/javascript" src="second.js"></script>
    </body>
</html>
```

4. Although it is difficult, now it is the time to start with D3….We should link D3 library first, to make it simple we link to the official online library (alternatively, you can download the javascript library locally and link it):

```
<html>
    <head>
        <title>D3 SVG Primitives</title>
        <link rel="stylesheet" href="second.css">
    </head>
    <body>
        <svg></svg>

        <script src="https://d3js.org/d3.v4.min.js"></script>
        <script type="text/javascript" src="second.js"></script>
    </body>
</html>
```

As we need to use D3 in our own script, put D3 library in front of our own script.

5. The preparation is ready, we could start code our own script, open second.js, we should start by creating our own SVG canvas for drawing:

```
window.onload = function(){
    var svgCanvas = d3.select("svg")
        .attr("width", 960)
        .attr("height", 540);
}
```

6. Open it in the browser (if you are using bracket, open the second.html and click the live preview button)! Nothing special…Well, this is because the canvas is white by default. It is now time to play with CSS (second.css):

```
.svgCanvas {
    border: solid 1px
}
```

Here we added a solid border for the canvas, and we also need to update our Javascript code:

```
window.onload = function(){
    var svgCanvas = d3.select("svg")
        .attr("width", 960)
        .attr("height", 540)
        .attr("class", "svgCanvas");
}
```

7. We should see a good looking canvas now! Let's try to draw some primitives (basic shapes) on this canvas, first a rectangle:

```
window.onload = function(){
    var svgCanvas = d3.select("svg")
        .attr("width", 960)
        .attr("height", 540)
        .attr("class", "svgCanvas");

    svgCanvas.append("rect")
        .attr("x", 100)
        .attr("y", 100)
        .attr("width", 100)
```

```
        .attr("height", 50);
}
```

Maybe you want it to have rounded-corners:

```
window.onload = function(){
    var svgCanvas = d3.select("svg")
        .attr("width", 960)
        .attr("height", 540)
        .attr("class", "svgCanvas");

    svgCanvas.append("rect")
        .attr("x", 100)
        .attr("y", 100)
        .attr("width", 100)
        .attr("height", 50)
        .attr("rx", 15)
        .attr("ry", 10);
}
```

You can also make it blue:

```
window.onload = function(){
    var svgCanvas = d3.select("svg")
        .attr("width", 960)
        .attr("height", 540)
        .attr("class", "svgCanvas");

    svgCanvas.append("rect")
        .attr("x", 100)
        .attr("y", 100)
        .attr("width", 100)
        .attr("height", 50)
        .attr("rx", 15)
        .attr("ry", 10)
        .attr("fill", "lightblue");
}
```

Let's add a few more shapes (there are more SVG shapes than this example, please refer to the link in the previous SVG section):

```
window.onload = function(){
    var svgCanvas = d3.select("svg")
        .attr("width", 960)
        .attr("height", 540)
        .attr("class", "svgCanvas");

    svgCanvas.append("rect")
        .attr("x", 100)
        .attr("y", 100)
        .attr("width", 100)
        .attr("height", 50)
        .attr("rx", 15)
        .attr("ry", 10)
        .attr("fill", "lightblue");
```

```
    svgCanvas.append("circle")
        .attr("cx", 300)
        .attr("cy", 200)
        .attr("r", 30)
        .attr("fill", "lightgreen");

    svgCanvas.append("ellipse")
        .attr("cx", 300)
        .attr("cy", 400)
        .attr("rx", 30)
        .attr("ry", 60)
        .attr("fill", "lightgreen");

    svgCanvas.append("line")
        .attr("x1", 300)
        .attr("y1", 400)
        .attr("x2", 300)
        .attr("y2", 200)
        .attr("stroke", "black");
}
```

Technically, all visualisations are made by those primitives, so once you know how to create primitives, you are already able to create all visualisations with D3.

2b. Creating visualisations from data

D3 is actually short for *Data Driven Document*. Obviously it means Data plays an important role in creating visualisations. Let's create a simple visualisation based on some data.

1. Create a folder to store our files, let's name it "third".
2. Let's create three files:third.html, third.css, third.js.
3. Edit the third.html with:

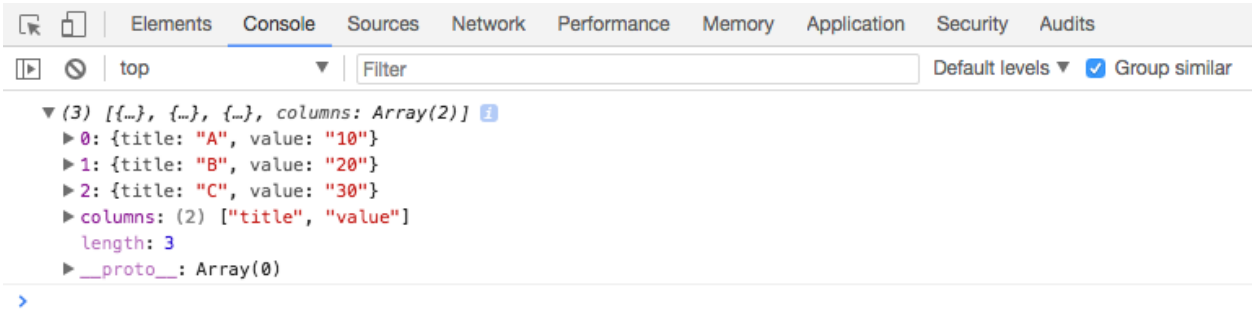
```
<html>
    <head>
        <title>Creating visualisations from data</title>
        <link rel="stylesheet" href="third.css">
    </head>
    <body>
        <svg></svg>

        <script src="https://d3js.org/d3.v4.min.js"></script>
        <script type="text/javascript" src="third.js"></script>
    </body>
</html>
```

4. Download our data to the same folder.
5. Now we can start loading the data with D3, D3 provide functions to load data in variety of formats, we will use d3.csv to load our data (in third.js):

```
window.onload = function(){
    d3.csv("third.csv", function(d){
        console.log(d);
    });
}
```

If you are using brackets, open the third.html file and click the live preview button.

6. We want to have a look at whether we loaded the correct data, so we print the detailed data in the console, now we can open the console to have a look (still remember the way to open developer tools in Chrome?), we should have:



We exactly had our values.

7. Let's set up our canvas and start drawing (do not to have the svgCanvas class in your CSS file):

```
window.onload = function(){
    var svgCanvas = d3.select("svg")
        .attr("width", 960)
        .attr("height", 540)
        .attr("class", "svgCanvas");

    d3.csv("third.csv", function(d){
        console.log(d);

        svgCanvas.selectAll("circle")
            .data(d).enter() // create place hodlers if the data are new
            .append("circle") // create one circle for each
            .attr("cx", function(thisElement, index){
              // calculate the centres of circles
                return 150 + index * 150;
            })
            .attr("cy", 300)
            .attr("r", function(thisElement, index){
              // use the value from data to create the radius
                return thisElement["value"];
            });

    });
}
```

`selectAll("circle").data(d).enter()` is absolutely the magic function here. Basically it build a link between the HTML elements and the data. `selectAll` select the HTML element, and `data` link the html elements with the data. More details at: http://techtime.getharvest.com/blog/understanding-d3-selection-operations

8. Let's make it a bit more informative (still remember the way we make play around with color in the first week's D3 activity), do not forget to link the color library we use:

```
<html>
    <head>
        <title>Creating visualisations from data</title>
        <link rel="stylesheet" href="third.css">
```

```
    </head>
    <body>
        <svg></svg>

        <script src="https://d3js.org/d3.v4.min.js"></script>
        <script src="https://d3js.org/d3-scale-chromatic.v1.min.js"></script>
        <script type="text/javascript" src="third.js"></script>
    </body>
</html>
```

Now, we can make it colorful and also with some text there.

```
window.onload = function(){
    var svgCanvas = d3.select("svg")
        .attr("width", 960)
        .attr("height", 540)
        .attr("class", "svgCanvas");

    d3.csv("third.csv", function(d){
        console.log(d);

        var minValue = Infinity;
        var maxValue = -1;
        d.forEach(function(thisD){
            var thisValue = thisD["value"];
            minValue = Math.min(minValue, thisValue);
            maxValue = Math.max(maxValue, thisValue);
        });

        var value2range = d3.scaleLinear()
            .domain([minValue, maxValue])
            .range([0.5, 1]);
        var range2color = d3.interpolateBlues;

        svgCanvas.selectAll("circle")
            .data(d).enter() // create place hodlers if the data are new
            .append("circle") // create one circle for each
            .attr("cx", function(thisElement, index){ // calculate the centres of circles
                return 150 + index * 150;
            })
            .attr("cy", 300)
            .attr("r", function(thisElement, index){
              // use the value from data to create the radius
                return thisElement["value"];
            })
            .attr("fill", function(thisElement, index){
                return range2color(value2range(thisElement["value"]))
            });

        svgCanvas.selectAll("text")
            .data(d).enter()
            .append("text")
            .attr("x", function(thisElement, index){
                return 150 + index * 150;
```

```
            })
            .attr("y", 300 - 35)
            .attr("text-anchor", "middle")
            .text(function(thisElement, index){
                return thisElement["title"] + ": " + thisElement["value"];
            });

    });
}
```

9. Let's add some interactions:

```
window.onload = function(){
    var svgCanvas = d3.select("svg")
        .attr("width", 960)
        .attr("height", 540)
        .attr("class", "svgCanvas");

    d3.csv("third.csv", function(d){
        console.log(d);

        var minValue = Infinity;
        var maxValue = -1;
        d.forEach(function(thisD){
            var thisValue = thisD["value"];
            minValue = Math.min(minValue, thisValue);
            maxValue = Math.max(maxValue, thisValue);
        });

        var value2range = d3.scaleLinear()
            .domain([minValue, maxValue])
            .range([0.5, 1]);
        var range2color = d3.interpolateBlues;

        svgCanvas.selectAll("circle")
            .data(d).enter() // create place hodlers if the data are new
            .append("circle") // create one circle for each
            // calculate the centres of circles
            .attr("cx", function(thisElement, index){
                return 150 + index * 150;
            })
            .attr("cy", 300)
            // use the value from data to create the radius
            .attr("r", function(thisElement, index){
                return thisElement["value"];
            })
            .attr("fill", function(thisElement, index){
                return range2color(value2range(thisElement["value"]))
            })
            .on("mouseover", function(thisElement, index){
                svgCanvas.selectAll("circle")
                    .attr("opacity", 0.5); // grey out all circles
                d3.select(this) // hightlight the on hovering on
                    .attr("opacity", 1);
```

```
            })
            .on("mouseout", function(thisElement, index){
              // restore all circles to normal mode
                svgCanvas.selectAll("circle")
                    .attr("opacity", 1);
            });

        svgCanvas.selectAll("text")
            .data(d).enter()
            .append("text")
            .attr("x", function(thisElement, index){
                return 150 + index * 150;
            })
            .attr("y", 300 - 35)
            .attr("text-anchor", "middle")
            .text(function(thisElement, index){
                return thisElement["title"] + ": " + thisElement["value"];
            });

    });
}
```

2c. Making a grouped bar chart with D3

In theory, you should be able to create any visualisation you can imagine with D3 now.

However, it also seems a lot of work needs to be done to create a "normal" visualisation, like a bar chart.

Actually, there are lots of help functions from D3 to help you with this process.

We will show you an example of creating bar charts with D3.

Download our data first. To make thing easier and also as an example, we have all the files (css, javascript, html) in one file, we can name it fourth.html:

```
<html>
    <head>
        <title>Grouped bar chart in D3</title>
        <style>
            .svgCanvas {
                border: solid 1px
            }
        </style>
    </head>
    <body>
        <svg></svg>

        <script src="https://d3js.org/d3.v4.min.js"></script>
        <script>
            window.onload = function(){
                // Canvas width and height
                var width = 600;
                var height = 300;

                // Create a SVG canvas
```

```
var thisCanvas = d3.select("svg")
    .attr("width", width)
    .attr("height", height)
    .attr("class", "svgCanvas");

// We want some margin between the boundary of the canvas and the bar charts
var margin = {top: 10, right: 20, bottom: 30, left: 50};
width = width - margin.left - margin.right;
height = height - margin.top - margin.bottom;

// create the area to draw the bar chart
// g is a SVG element to group multiple SVG elements
var barChartArea = thisCanvas
    .append("g")
    .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

// Set up a helpr function to divide x axis
var x0 = d3.scaleBand()
    .rangeRound([0, width])
    .paddingInner(0.2); // padding between different groups

// two property for each group
var keys = ["income", "spend"];
// Set up a helpr function to divide x axis **within** a group
var x1 = d3.scaleBand()
    .padding(0.1); // padding between different properties within a group

// helper function to calculate the height of each bar
var y = d3.scaleLinear()
    .rangeRound([height, 0]);

d3.csv("fourth.csv", function(data){
    console.log(data);

    // for differetn groups
    x0.domain(data.map(function(d) { return d['name']; }));
    // for properties within a group
    x1.domain(keys).rangeRound([0, x0.bandwidth()]);
    // for bar height
    y.domain([0, d3.max(data, function(d) {
        return d3.max(keys, function(key) {
            return parseInt(d[key]);
        });
    })]);


    barChartArea.append("g")
        .selectAll("g")
        .data(data)
        .enter().append("g")
        // create groups
        .attr("transform", function(d) {
          return "translate(" + x0(d["name"]) + ",0)";
```

```
                        })
                    .selectAll("rect")
                    // pre-processing the data
                    .data(function(d) {
                        return keys.map(function(key) {
                            return {
                                key: key,
                                value: d[key],
                                name: d["name"]
                            };
                        });
                    })
                    .enter().append("rect")
                    // start drawing bars for each property and each group
                    .attr("x", function(d) {
                        return x1(d["key"]);
                    })
                    .attr("y", function(d) {
                        return y(d["value"]);
                    })
                    .attr("width", x1.bandwidth())
                    .attr("height", function(d) { return height - y(d["value"]); })
                    .attr("fill", function(d) {
                        if(d["key"] === "income")
                        {
                            return "lightgreen";
                        }
                        else if(d["key"] === "spend")
                        {
                            return "red";
                        }
                    });

                // add x axis with ticks
                barChartArea.append("g")
                    .attr("class", "axis")
                    .attr("transform", "translate(0," + height + ")")
                    .call(d3.axisBottom(x0));

                // add y axis with ticks
                barChartArea.append("g")
                    .attr("class", "axis")
                    .call(d3.axisLeft(y).ticks(null, "s"))
                    .append("text")
                    .attr("x", 2)
                    .attr("y", y(y.ticks().pop()) + 0.5)
                    .attr("dy", "0.32em");
            });
        }
    </script>
</body>
</html>
```

Well, not that complicated, isn't it.

Now, we leave you a job to add a legend for the colours and make it interactive!

Chapter 6

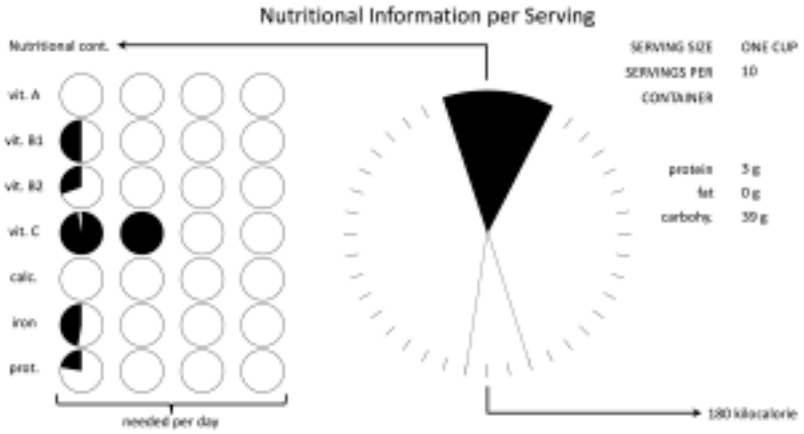
Activity: Effective graphic design

The phrase 'effective graphic design' begs the question 'effective for whom and for what?'.

Any graphic design must:

- Be usable by the user given her or his knowledge and abilities. Most people will be able to comprehend a common graphic such as a line graph or a table of data. However other forms of representation such as set diagrams (e.g. Euler's circles) are unfamiliar to many people, moreover some forms of representation are highly domain-specific e.g. cladograms are pretty much exclusive to biology and are only familiar to specialists.
- Be suited to the task the user needs to perform with the representation. There are many kinds of tasks – a user might be searching, comparing, spotting an intermittent event (vigilance), seeing trends, making inferences or deductions, discovering, assembling furniture (e.g. Chapter 3 of Munster, also Amar & Stasko, 2004; Wehrend & Lewis, 1990).

So lets go back to the proposed nutritional display graphic



What I want you to do is think about

1. Who is going to be using this graphic and what is their background?
2. For what task(s) are they going to look at the graphic and what is the context?

Now based on the material you have read

1. Come up with three or four alternative designs which you think are better. One should be black-and-white, the others can use colour. You may want to use the Five Design Sheet methodology.
2. Evaluate the quality of these designs in terms of the principles given in Effective Communication and the perceptual and cognitive characteristics given in The Human Visual System. Rank them.

FURTHER READING

Munzner, T. (2014) Visualisation, analysis and design. CRC Press

Hegarty, M. (2011) The cognitive science of visual-spatial displays: Implication for design. Topics in Cognitive Science, 3, 446-474.

Kosslyn, Stephen M. *Graph design for the eye and mind*. Oxford University Press, 2006.