Visual Computing I:

Interactive Computer Graphics and Vision

Digital Images and Colour Representation

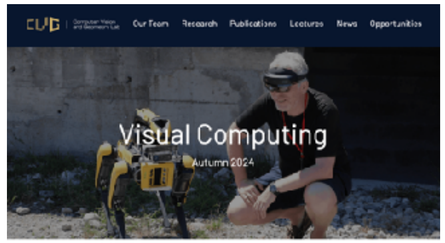
Tobias Langlotz

Last time

Design of the course

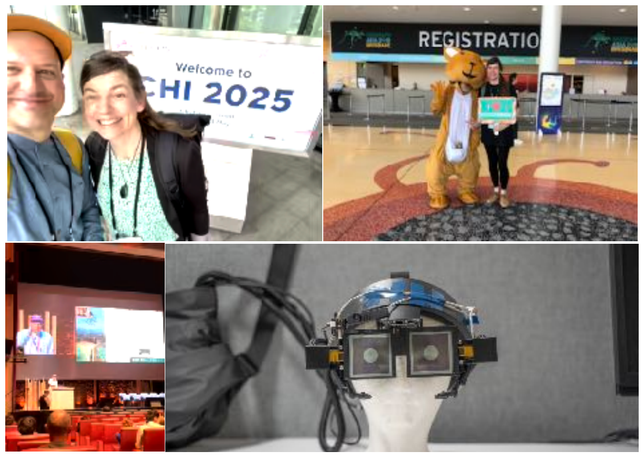
Industry Experience and Examples

Expert Societies

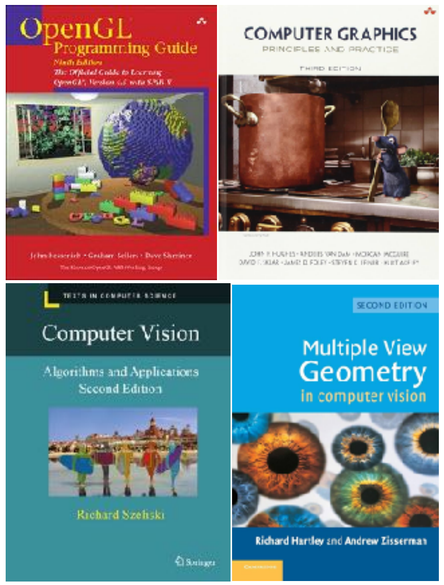


Similar Courses

Visual Computing I: Interactive Computer Graphics and Vision

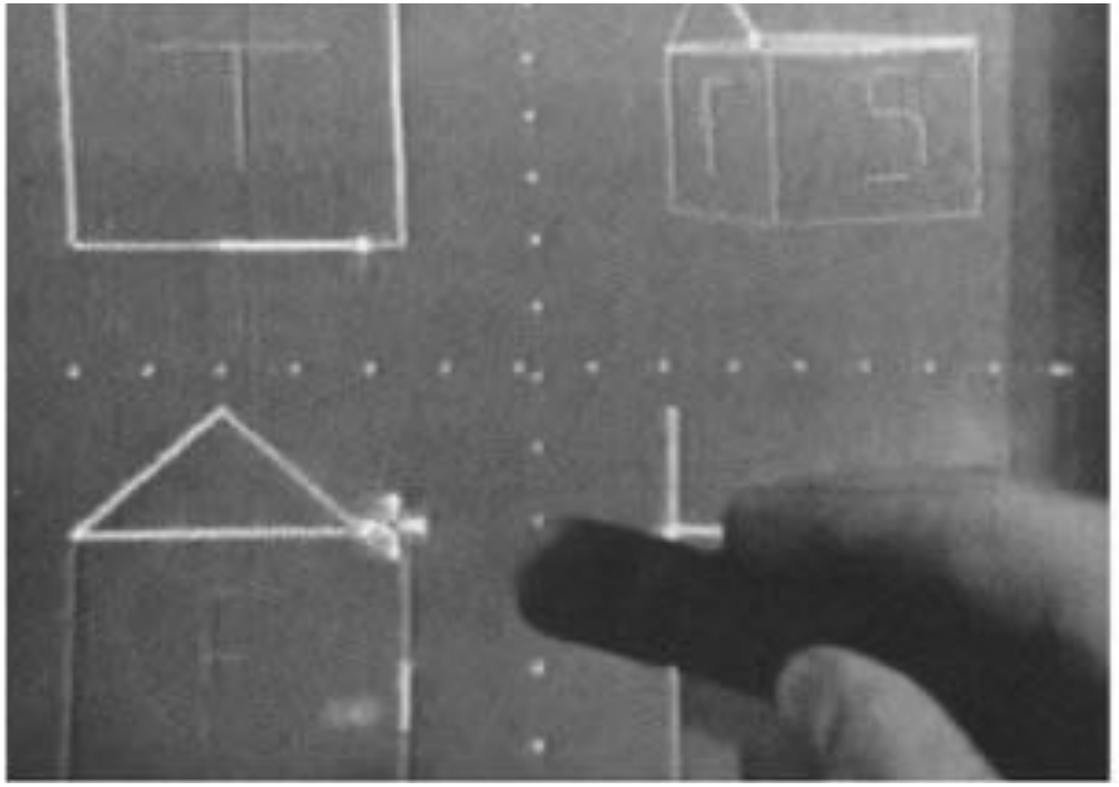


Academic Background



Literature

A Short History of Visual Computing



Ivan Sutherland, *Sketchpad, a Man–Machine Graphical Communication System*,1963

Digital Images and 2D Image Representation

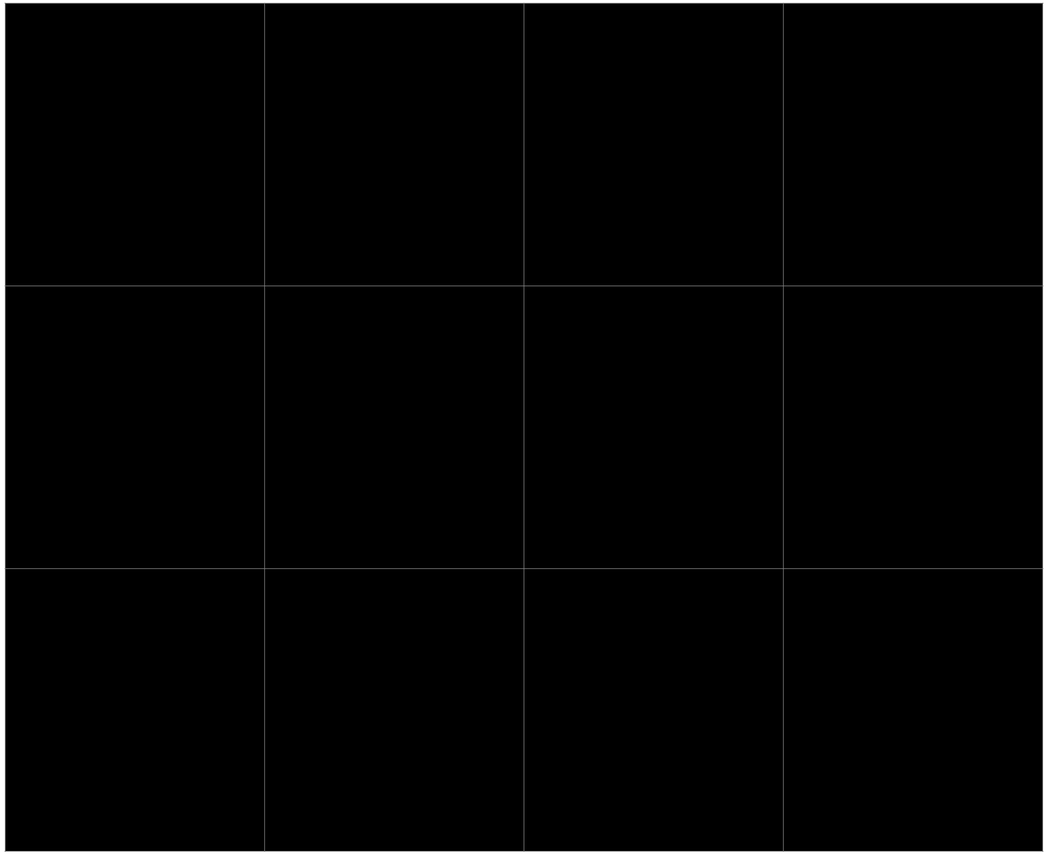
Digital Images and 2D Image Representation

- 2D array of pixels – a raster
- Pixels have a value, or list of values
  - value: usually brightness
  - values: brightness, colour, transparency

Digital Images and 2D Image Representation

- 2D array of pixels – a raster
- Pixels have a value, or list of values
  - value: usually brightness
  - values: brightness, colour, transparency

```
# Create a 2D array
image2D = create_array(height, width)

# Fill with example grayscale values
for y from 0 to height-1:
    for x from 0 to width-1:
        image2D[y][x] = 125
```



Digital Images and 2D Image Representation

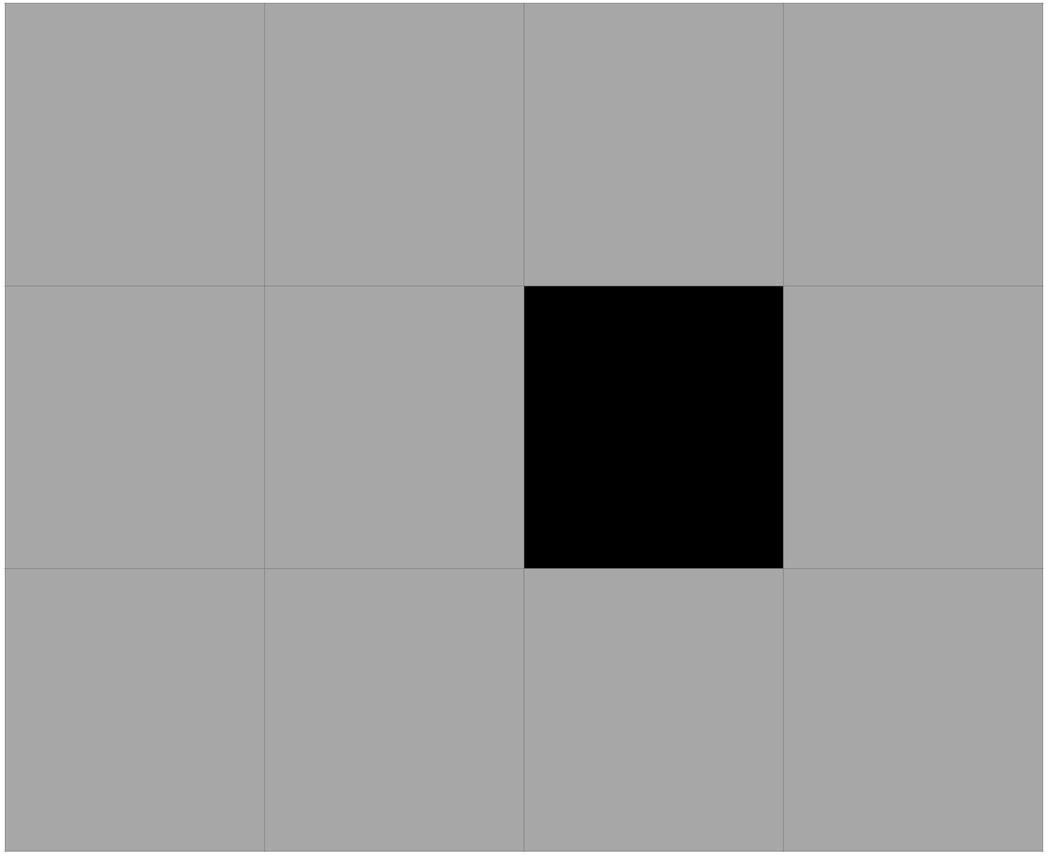
- 2D array of pixels – a raster
- Pixels have a value, or list of values
  - value: usually brightness
  - values: brightness, colour, transparency

```
# Create a 2D array
image2D = create_array(height, width)

# Fill with example grayscale values
for y from 0 to height-1:
    for x from 0 to width-1:
        image2D[y][x] = 125
```

Digital Images and 2D Image Representation

- 2D array of pixels – a raster
- Pixels have a value, or list of values
  - value: usually brightness
  - values: brightness, colour, transparency

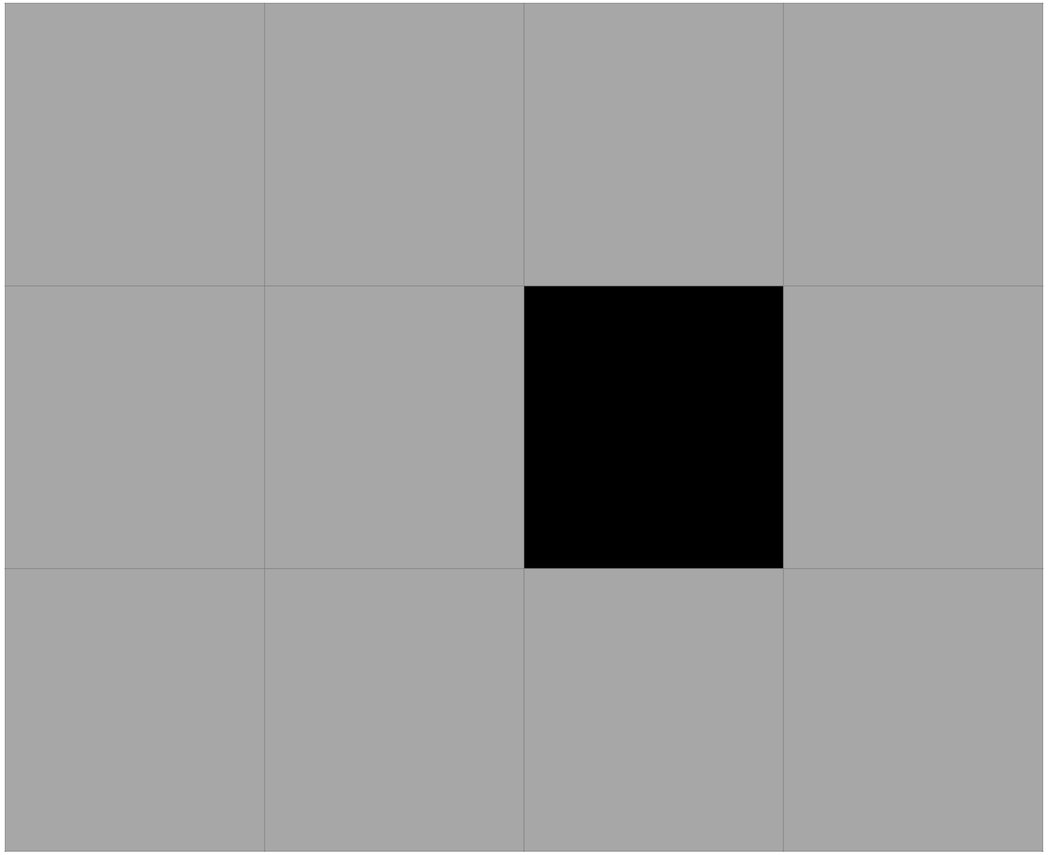
```
# Create a 2D array
image2D = create_array(height, width)

# Fill with example grayscale values
for y from 0 to height-1:
    for x from 0 to width-1:
        image2D[y][x] = 125

# Access pixel at (row=2, col=3)
image2D[1][2]= 0
```



Digital Images and 2D Image Representation

- 2D array of pixels – a raster
- Pixels have a value, or list of values
  - value: usually brightness
  - values: brightness, colour, transparency

```
# Create a 2D array
image2D = create_array(height, width)

# Fill with example grayscale values
for y from 0 to height-1:
    for x from 0 to width-1:
        image2D[y][x] = 125
image2D =
[
  [ 0, 1, 2, 3 ],      # row 0
  [ 0, 1, 2, 3 ],      # row 1
  [ 0, 1, 2, 3 ]     # row 2
]
```



Digital Images and 2D Image Representation

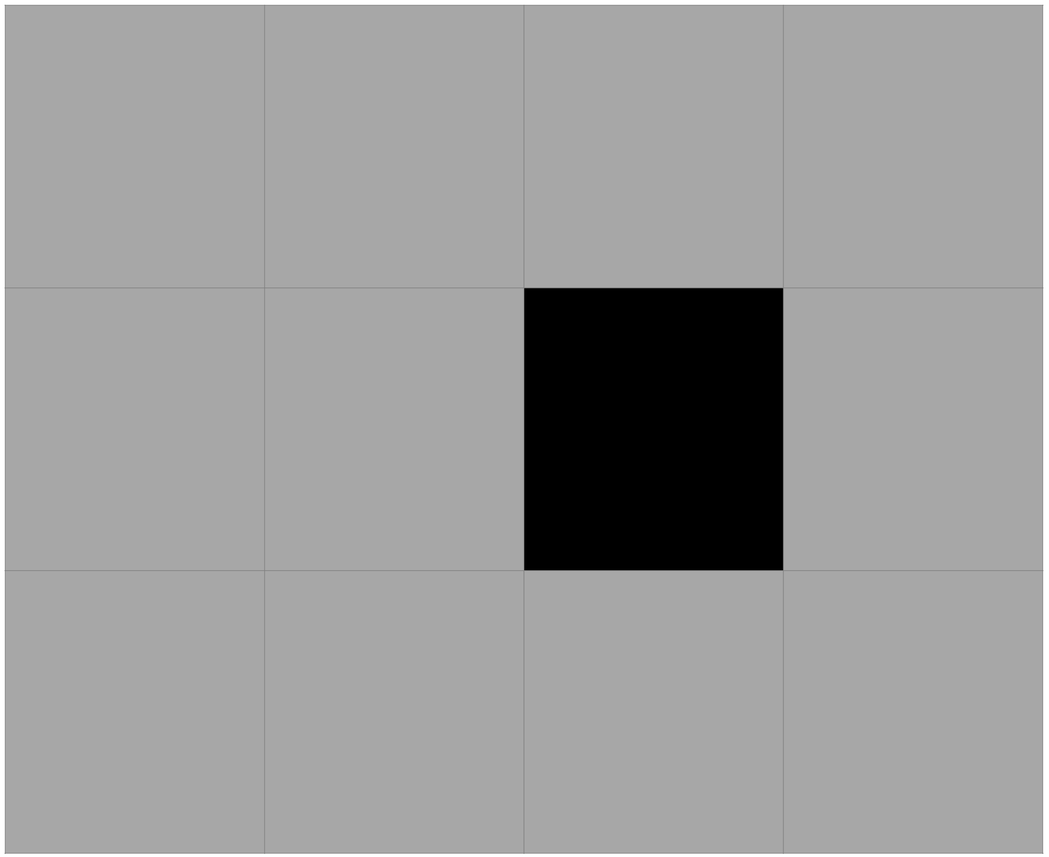
- 2D array of pixels – a raster
- Pixels have a value, or list of values
  - value: usually brightness
  - values: brightness, colour, transparency

```
# Create a 1D array
image1D = create_array(width * height)

# Fill with example grayscale values
for y from 0 to height-1:
    for x from 0 to width-1:
        index = (y * width) + x
        image1D[index]= 125
```

Digital Images and 2D Image Representation

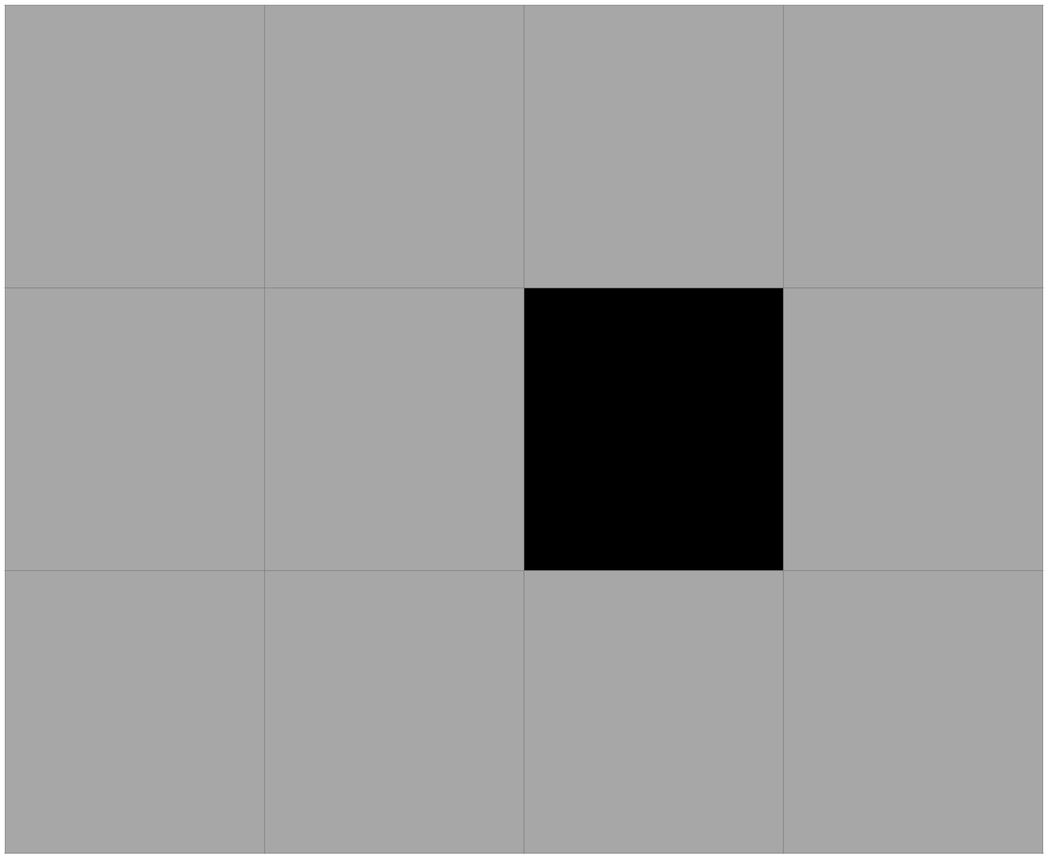
- 2D array of pixels – a raster
- Pixels have a value, or list of values
  - value: usually brightness
  - values: brightness, colour, transparency

```
# Create a 1D array
image1D = create_array(width * height)

# Fill with example grayscale values
for y from 0 to height-1:
    for x from 0 to width-1:
        index = (y * width) + x
        image1D[index]= 125

# Access pixel at (row=2, col=3)
index = (row-1 * width) + col
image1D[index]= 0
```



Digital Images and 2D Image Representation

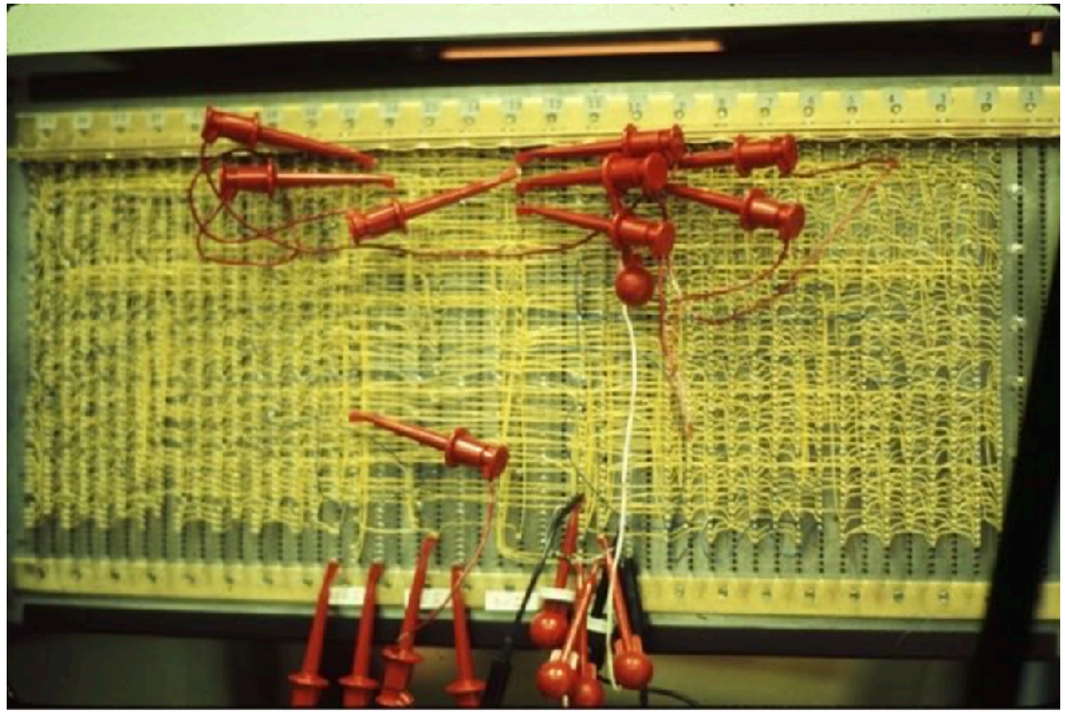
- 2D array of pixels – a raster
- Pixels have a value, or list of values
  - value: usually brightness
  - values: brightness, colour, transparency



```
# Create a 1D array
image1D = create_array(width * height)

# Fill with example grayscale values
for y from 0 to height-1:
    for x from 0 to width-1:
        index = (y * width) + x
        image1D[index]= 125
```

```
image1D = [ 0, 1, 2, 3,  4, 5, 6, 7,  8, 9, 10, 11 ]
            ↑  row0      ↑  row1      ↑  row2
```

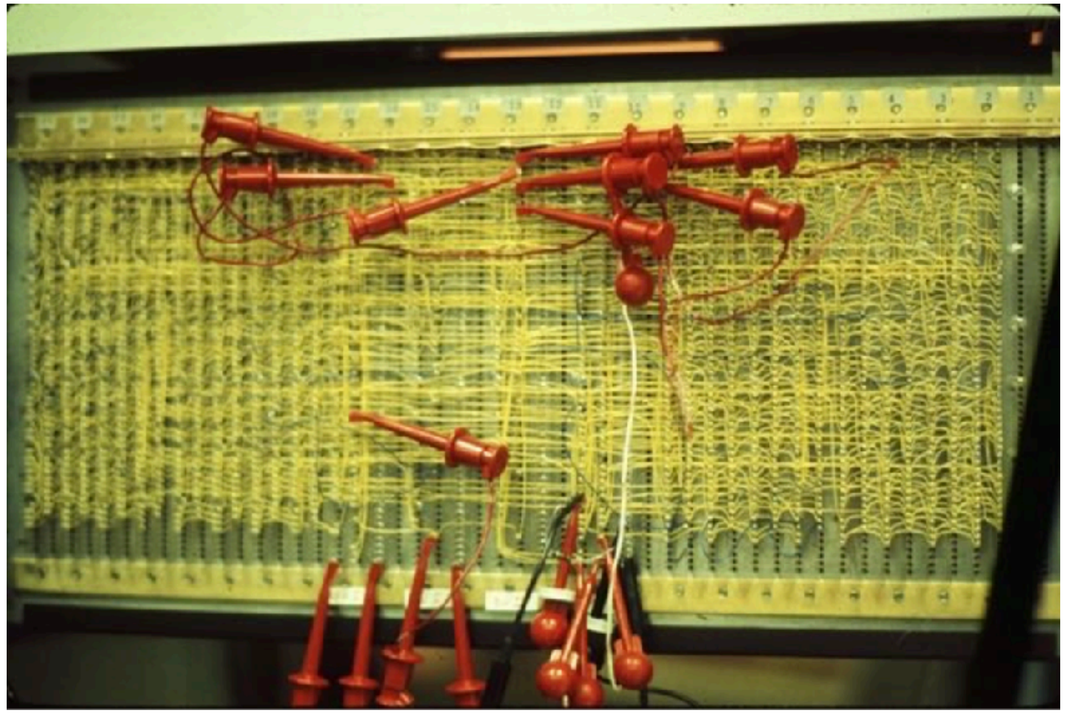
Digital Images and 2D Image Representation



Xerox Parc Super Frame, First Frame Buffer

Xerox Parc Super Frame, First Image Stored in Frame Buffer

Digital Images and 2D Image Representation



Xerox Parc Super Frame, First Frame Buffer

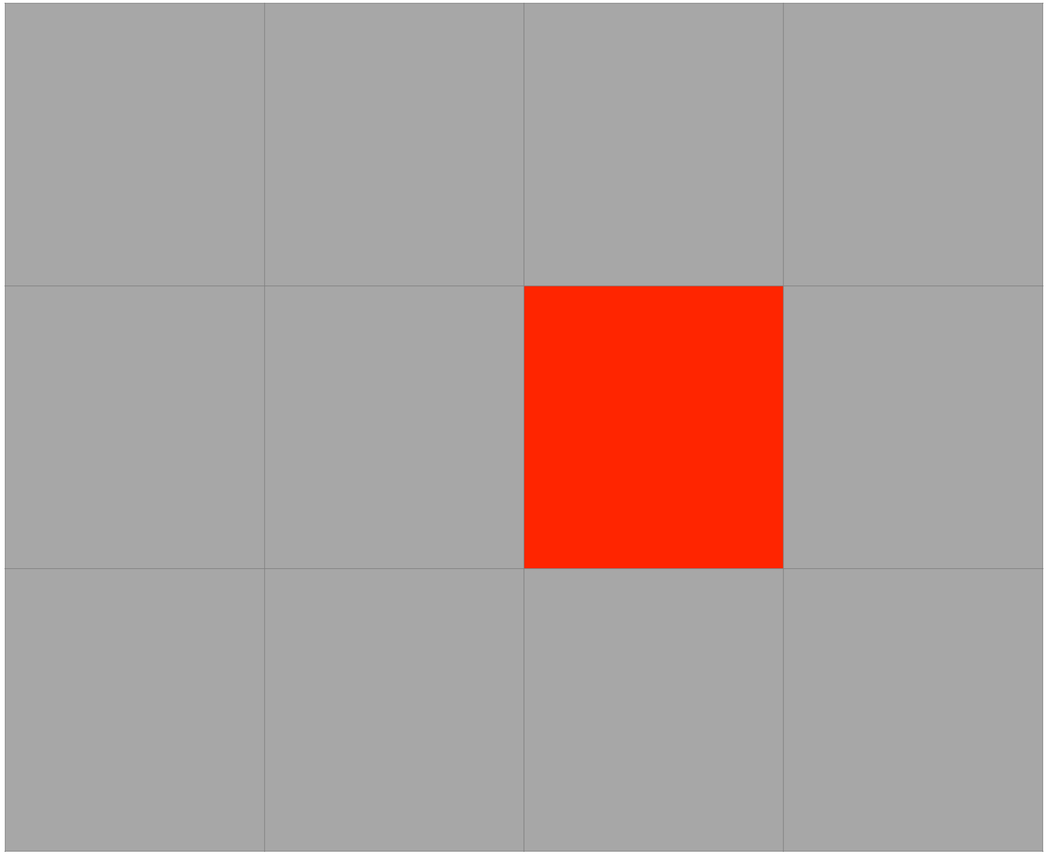
Xerox Parc Super Frame, First Image Stored in Frame Buffer

- Framebuffer: memory for a raster display

Digital Images and 2D Image Representation

- 2D array of pixels – a raster
- Pixels have a value, or list of values
  - value: usually brightness
  - values: brightness, colour, transparency

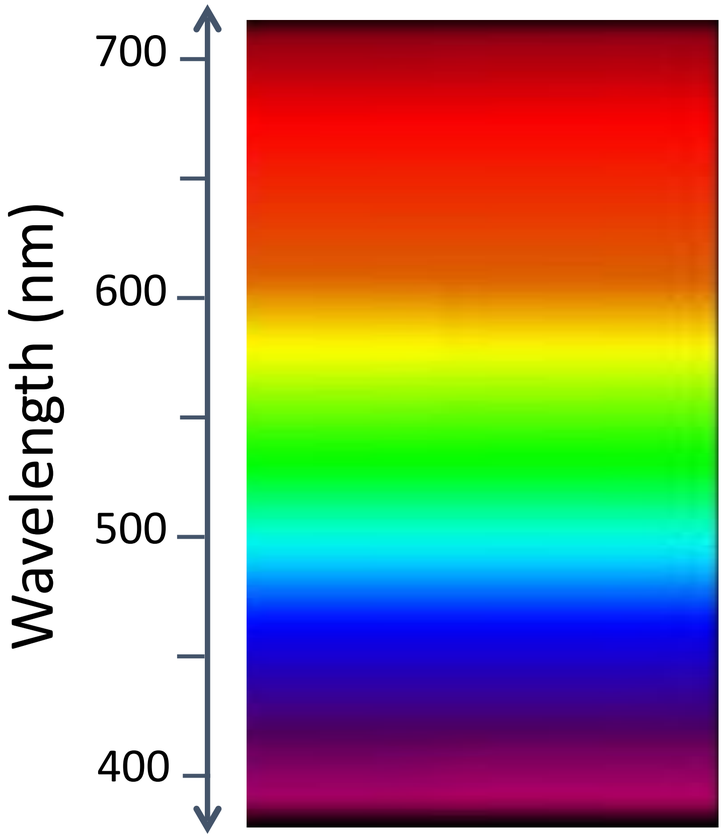
```
# Create a 2D array
image2D = create_array(height, width)

# Fill with example grayscale values
for y from 0 to height-1:
    for x from 0 to width-1:
        image2D[y][x] = 125

# Access pixel at (row=2, col=3)
image2D[1][2]= ?
```



Colour

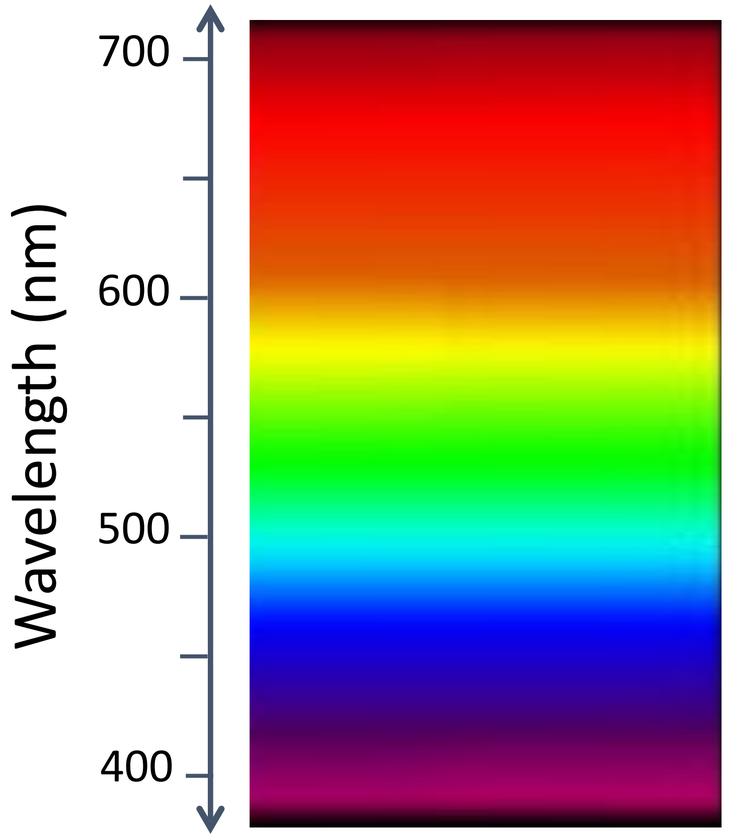
Colour as a Spectrum

- A physical interpretation of light
  - Light has dual wave/particle nature
  - Colour relates to the frequency or wavelength of photons
  - ‘White’ is a mix of all wavelengths
- This explains many phenomena
  - Rainbows appear because refractive index varies by wavelength
  - Some processes emit photons of specific colour – e.g. sodium lights, LED’s etc



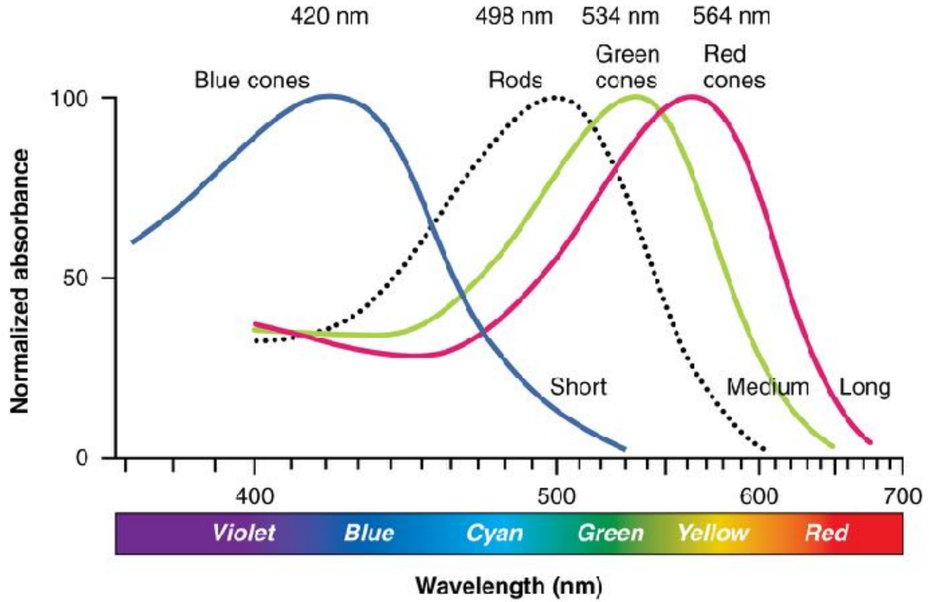
Colour as a Spectrum?

- This does not explain everything
  - Blue + Green light —> Cyan
  - Red + Green light —> Yellow
  - Red + Blue light —> Magenta
  - Blue + Yellow **paint** —> Green
  - Blue + Yellow light —> White



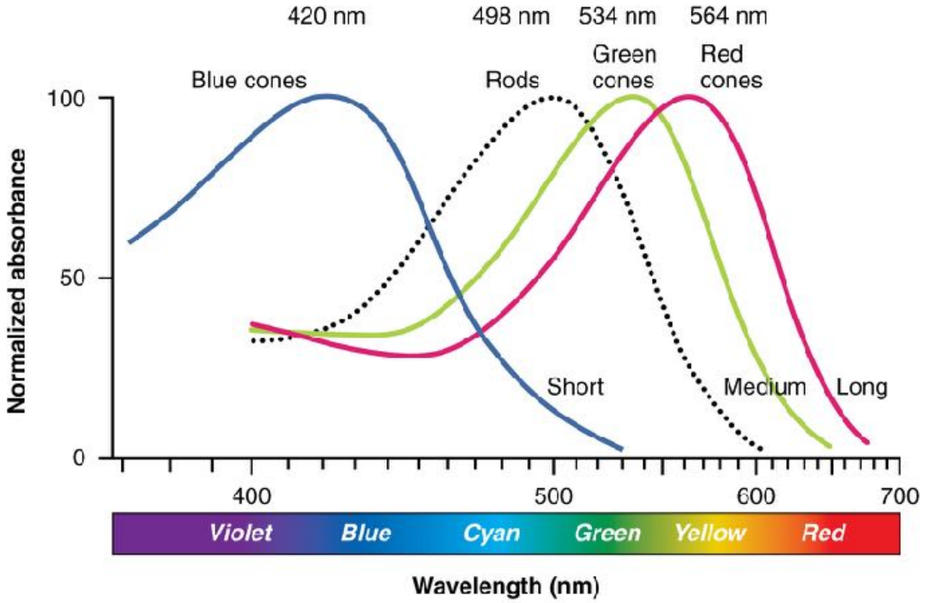
Colour Perception

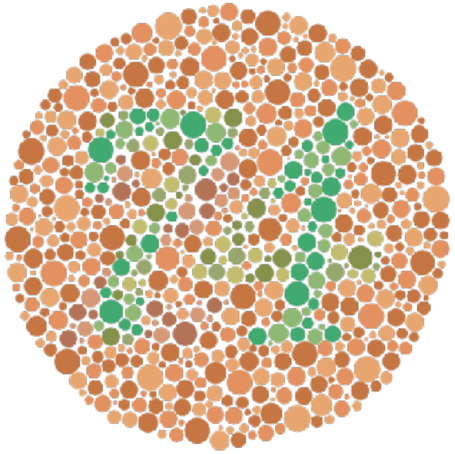
- We perceived light on our retina
  - Four types of cells (usually)
  - Rods – just sensitive to brightness
  - 3 types of Cones – sensitive to short/medium/long wavelengths (trichromatic theory of color vision)
  - Often called ‘blue’, ‘green’ or ‘red’ cones

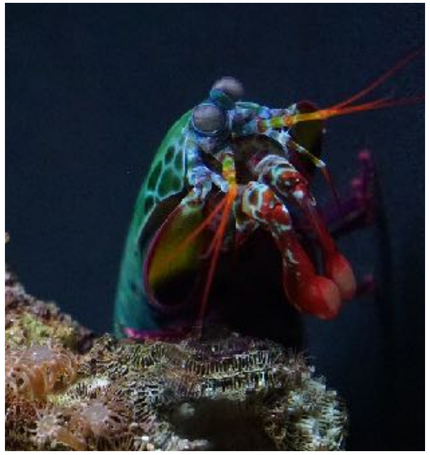


Colour Perception

- We perceived light on our retina
  - Four types of cells (usually)
  - Rods – just sensitive to brightness
  - 3 types of Cones – sensitive to short/medium/long wavelengths (trichromatic theory of color vision)
  - Often called ‘blue’, ‘green’ or ‘red’ cones

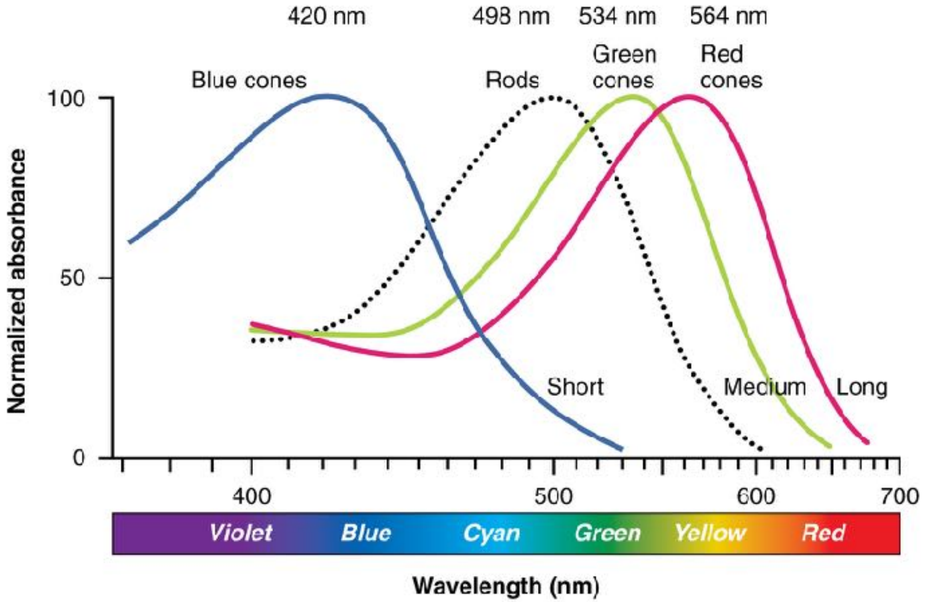


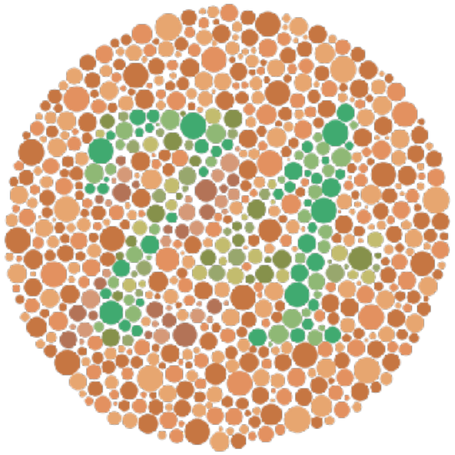




Colour Perception

- We perceived light on our retina
  - Four types of cells (usually)
  - Rods – just sensitive to brightness
  - 3 types of Cones – sensitive to short/medium/long wavelengths (trichromatic theory of color vision)
  - Often called ‘blue’, ‘green’ or ‘red’ cones

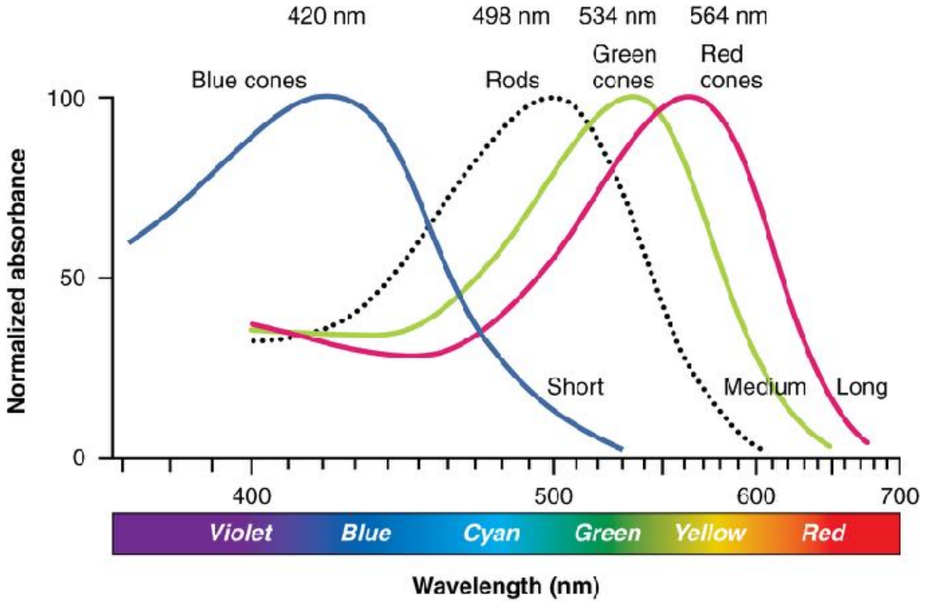




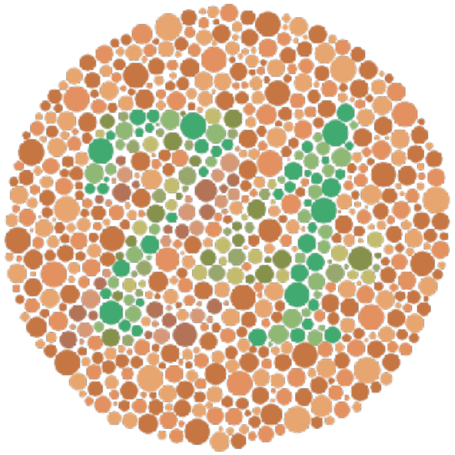
Trichromacy

Colour Perception

- We perceived light on our retina
  - Four types of cells (usually)
  - Rods – just sensitive to brightness
  - 3 types of Cones – sensitive to short/medium/long wavelengths (trichromatic theory of color vision)
  - Often called ‘blue’, ‘green’ or ‘red’ cones



Monochromacy



Dichromacy

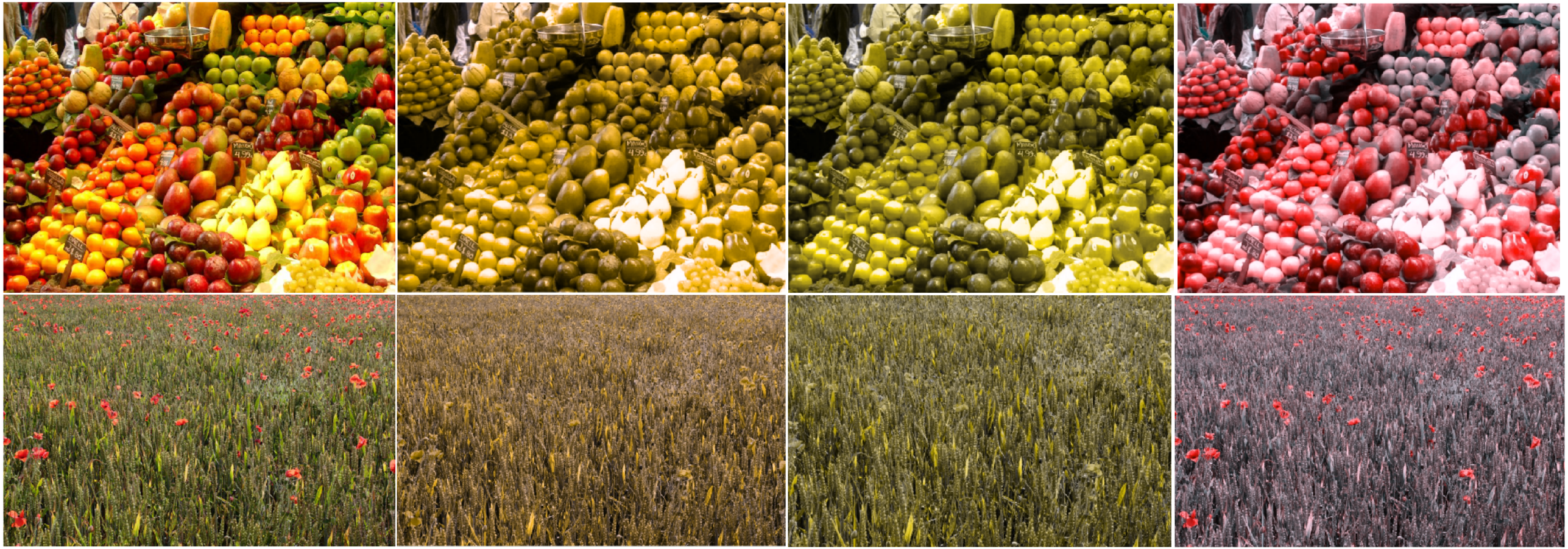
Trichromacy

Tetrachromacy

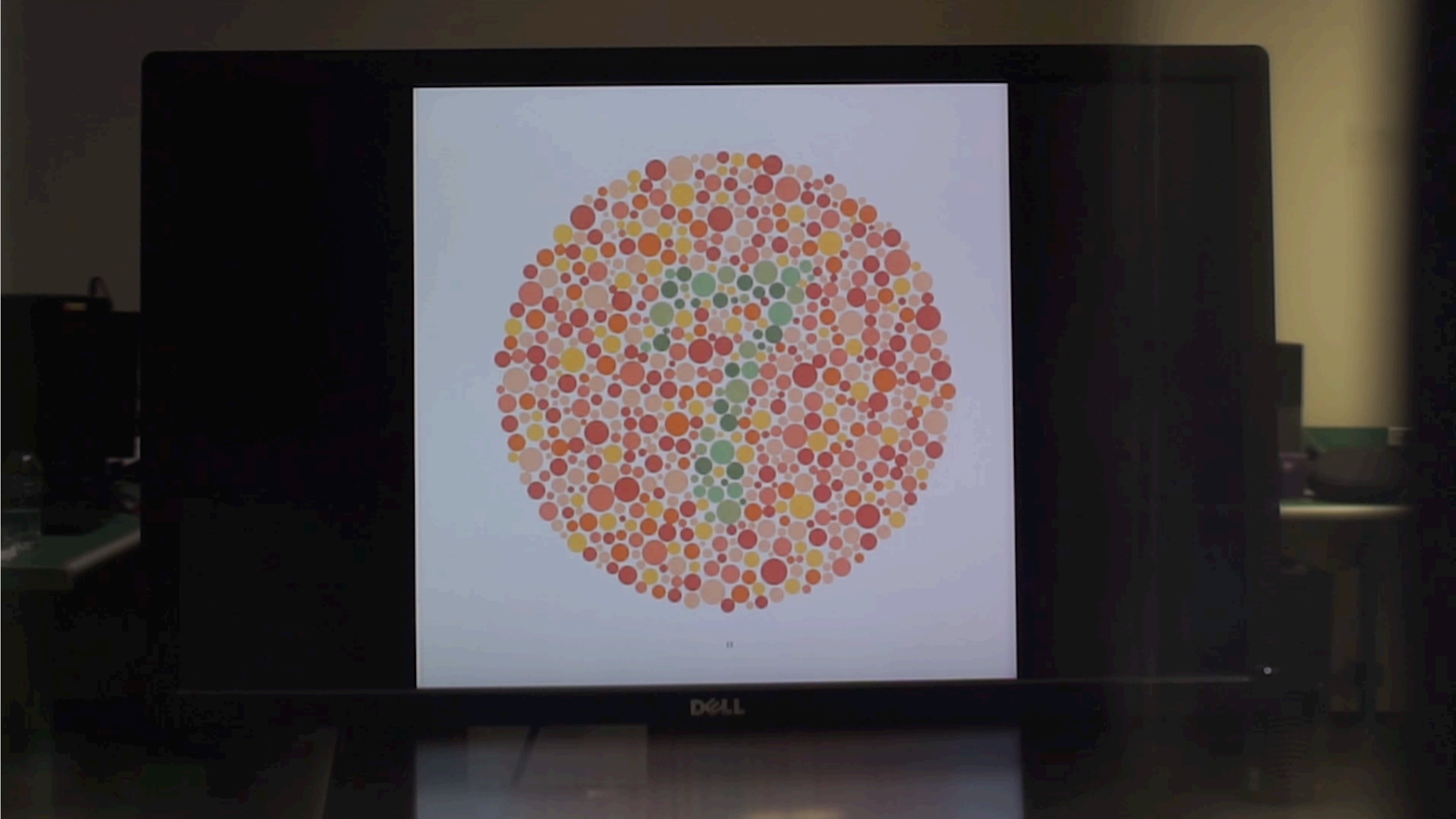
Dodecachromacy to Hexadecachromacy

Sidetrack:
Some info on Dichromacy and Colour Vision Deficiency

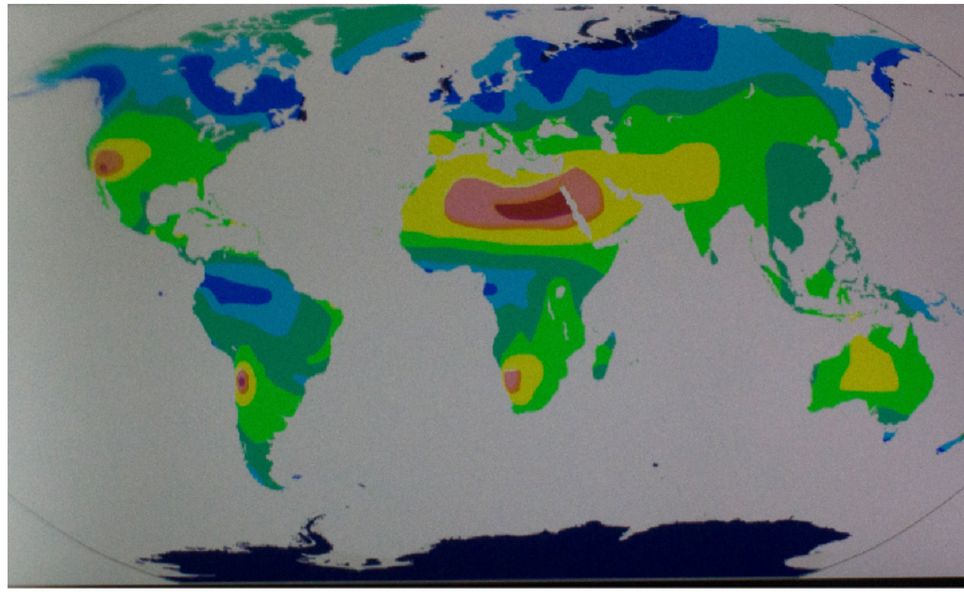
Colour Vision Deficiency (CVD)

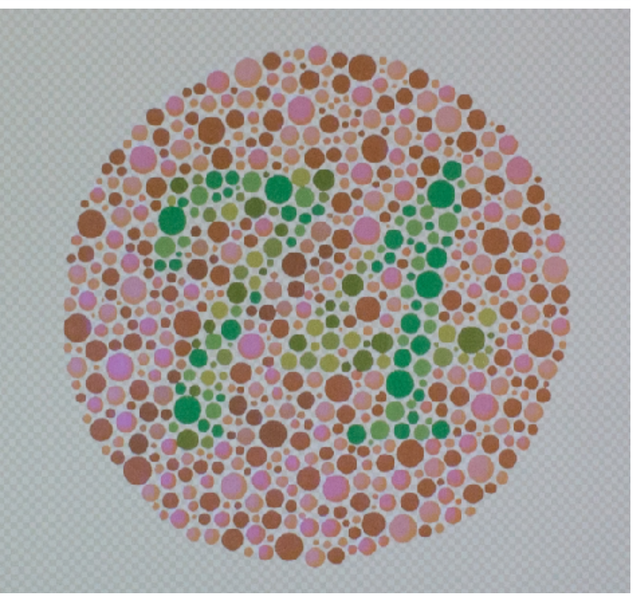


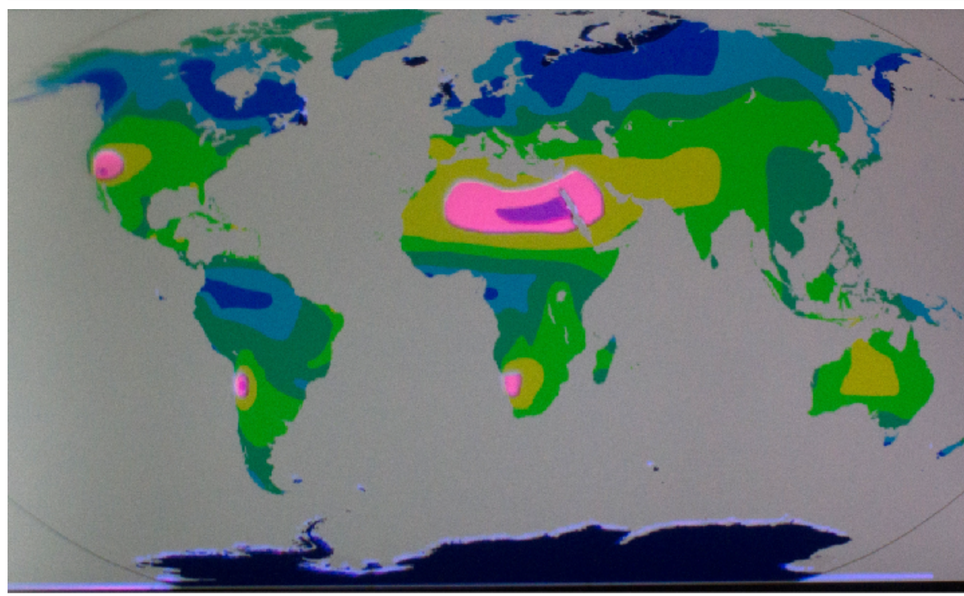
Original | Deuteranopia (green blind) | Protanopia (red blind) | Tritanopia (blue blind)



Results

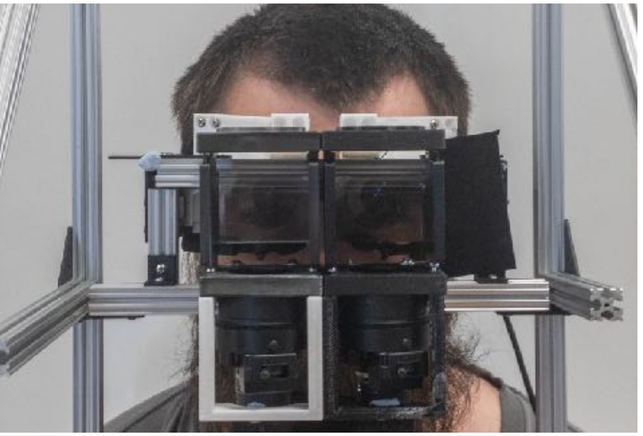




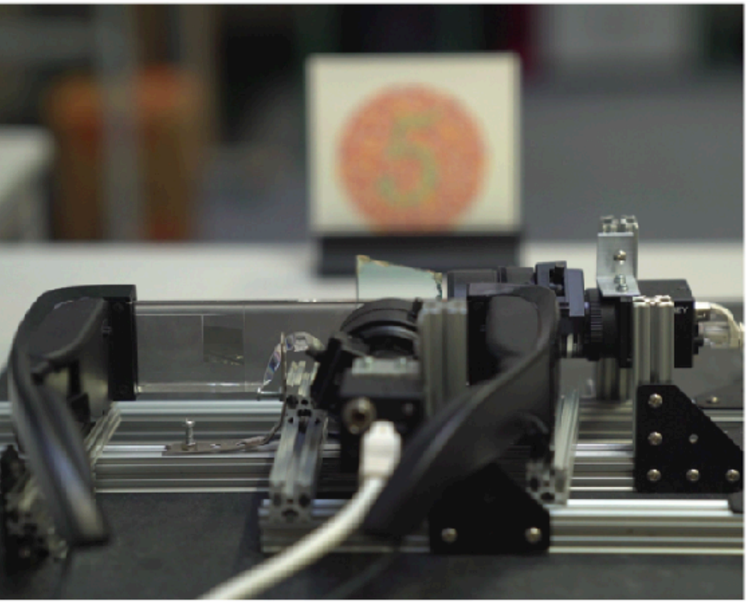


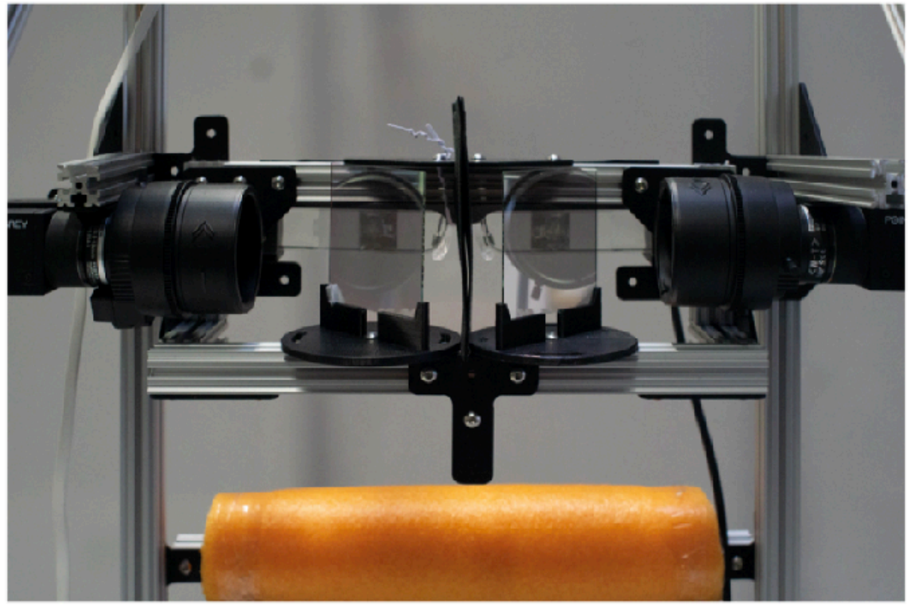
Jonathan Sutton, Tobias Langlotz, Alexander Plopski (2022)
Seeing Colours: Addressing Colour Vision Deficiency with Vision Augmentations using Computational Glasses,
ACM Transactions on Human-Computer Interaction (ACM TOCHI), 2022

Prototypes










Tobias Langlotz, Jonathan Sutton, Stefanie Zollmann, Yuta Itoh, and Holger Regenbrecht (2018) *ChromaGlasses: Computational Glasses for Compensating Colour Blindness ACM CHI'18 Conference on Human Factors in Computing Systems* (ACM CHI 2018), Montreal, 2018,

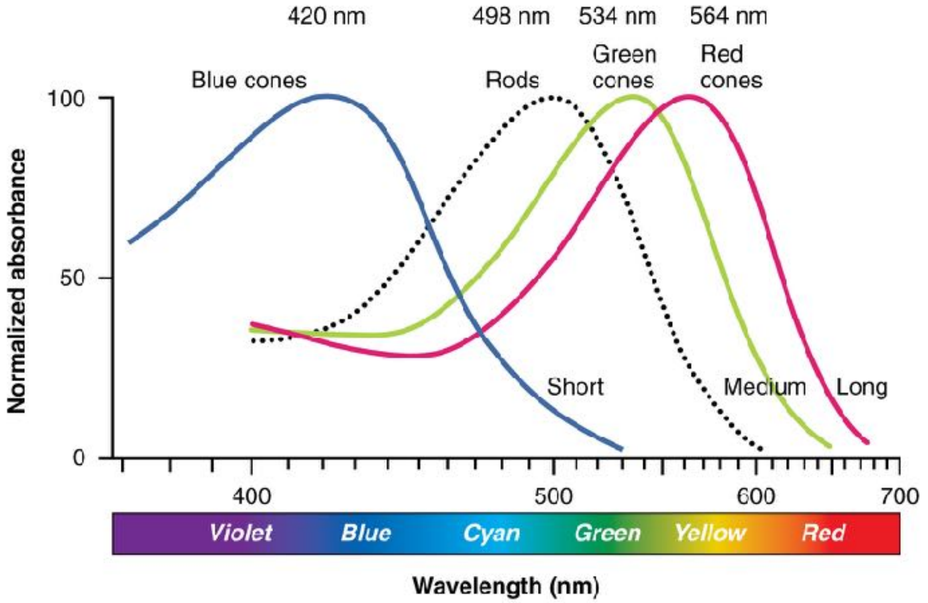
Honourable Mention Award

Jonathan Sutton, Tobias Langlotz, Alexander Plopski (2022)
Seeing Colours: Addressing Colour Vision Deficiency with Vision Augmentations using Computational Glasses,
ACM Transactions on Human-Computer Interaction (ACM TOCHI), 2022

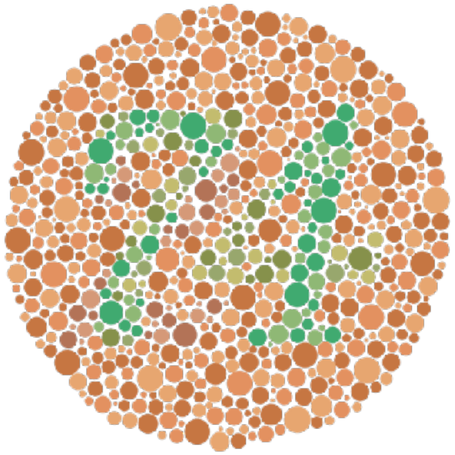
Back to topic….

Colour Perception

- We perceived light on our retina
  - Four types of cells (usually)
  - Rods – just sensitive to brightness
  - 3 types of Cones – sensitive to short/medium/long wavelengths (trichromatic theory of color vision)
  - Often called ‘blue’, ‘green’ or ‘red’ cones



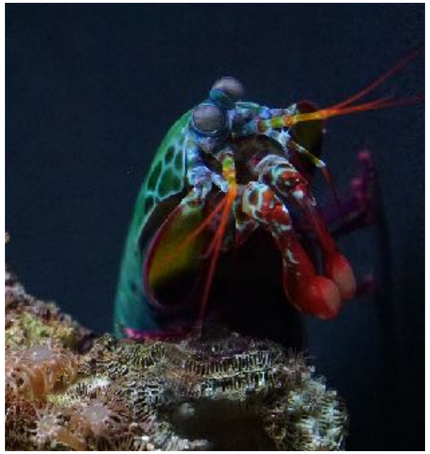
Monochromacy



Dichromacy

Trichromacy

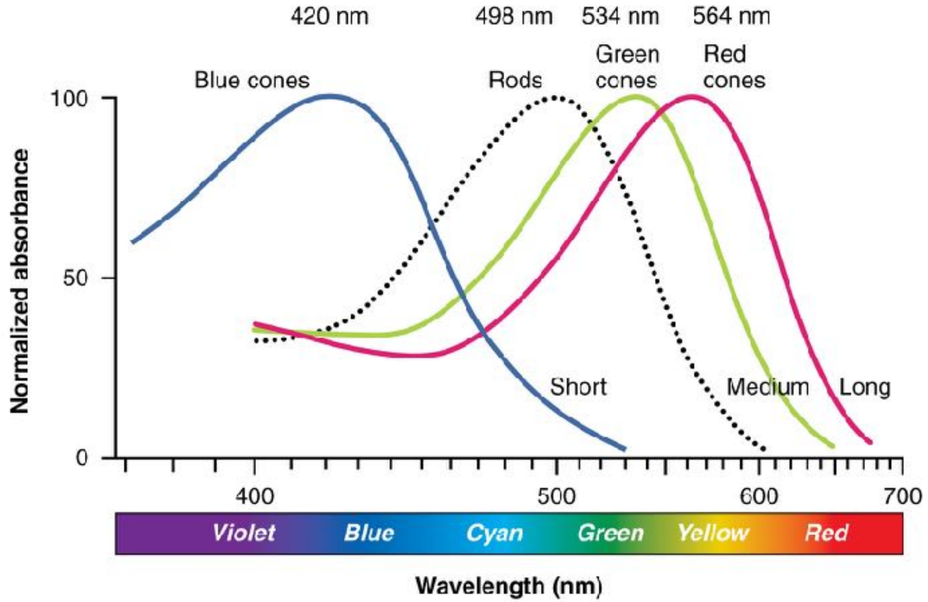
Tetrachromacy



Dodecachromacy to Hexadecachromacy

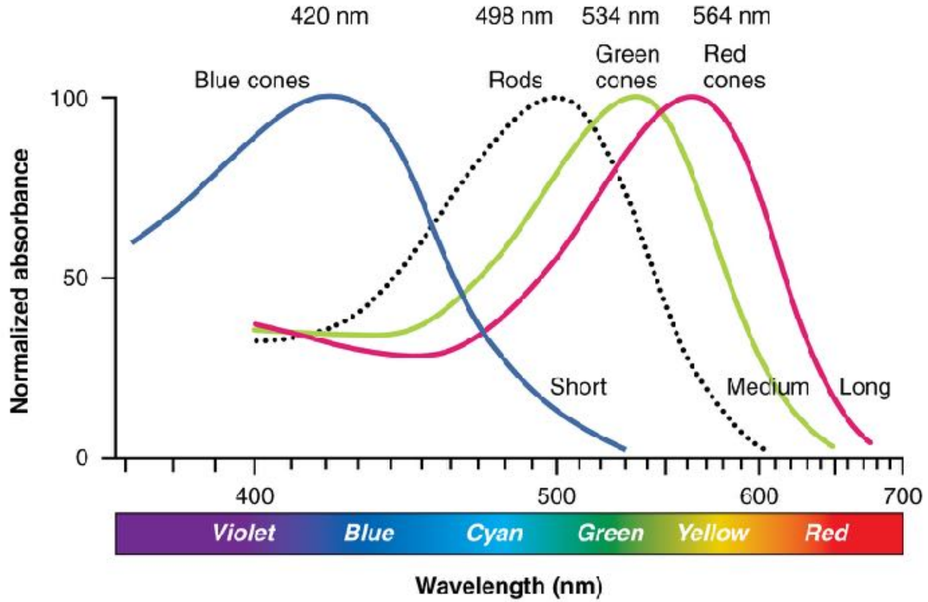
Colour Perception

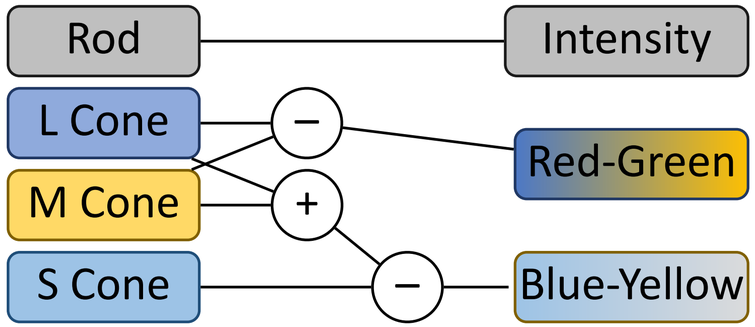
- We perceived light on our retina
  - Four types of cells (usually)
  - Rods – just sensitive to brightness
  - 3 types of Cones – sensitive to short/medium/long wavelengths (trichromatic theory of color vision)
  - Often called ‘blue’, ‘green’ or ‘red’ cones



Colour Perception

- We perceived light on our retina
  - Four types of cells (usually)
  - Rods – just sensitive to brightness
  - 3 types of Cones – sensitive to short/medium/long wavelengths (trichromatic theory of color vision)
  - Often called ‘blue’, ‘green’ or ‘red’ cones
- Brain does not receive raw signal
  - Rather, differences in cone responses
  - One axis is $L - M$, or red–green
  - The other is $S - (L + M)$, or blue–yellow

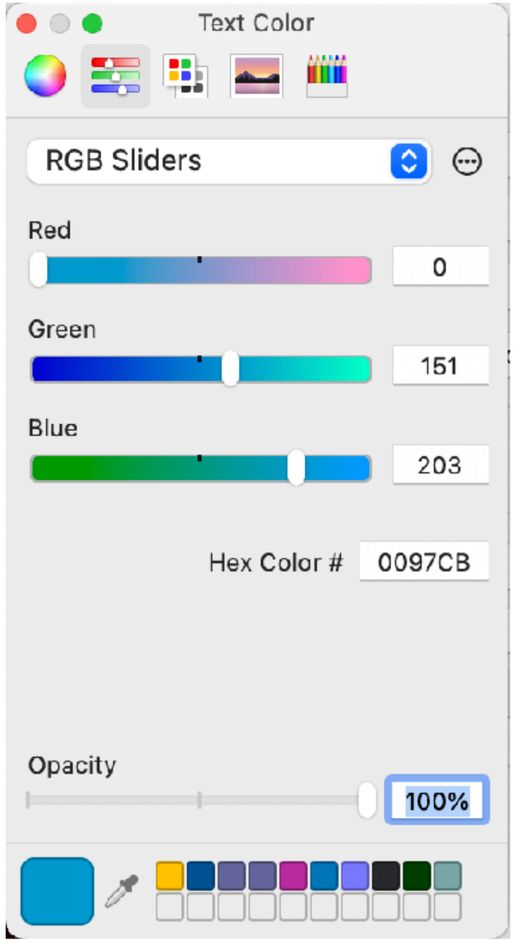


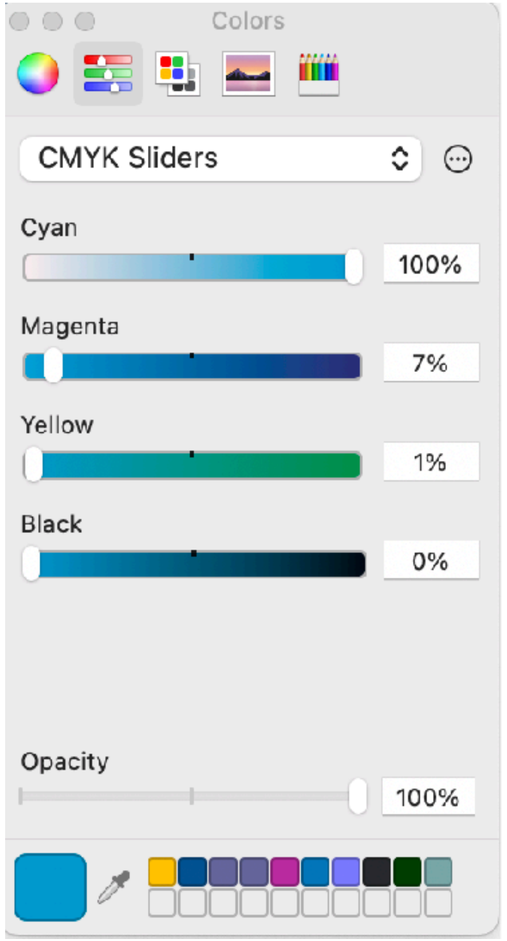


Representing Colour

- We perceive 3 colour dimensions
  - Short, medium, long cones
  - Intensity, Red-Green, Blue-Yellow

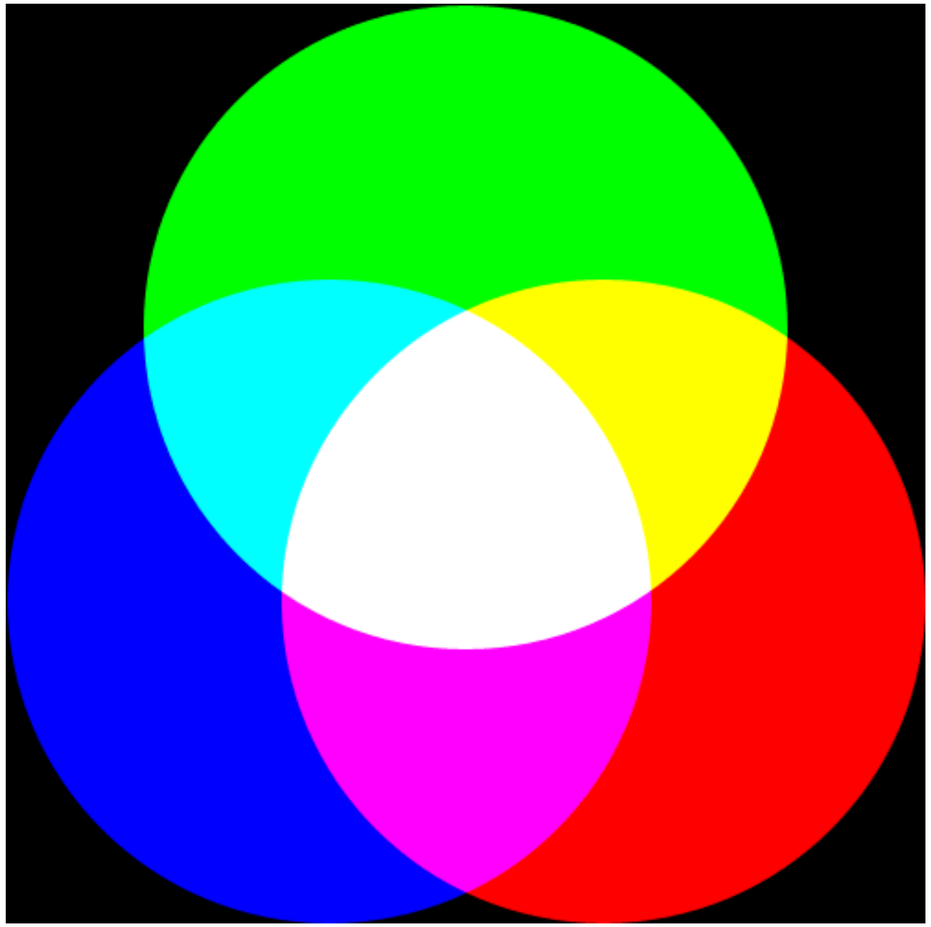
- Colour representations are usually 3 dimensional
  - Red-Green-Blue -> RGB
  - Cyan-Magenta-Yellow -> CMY (+K)
  - Hue-Saturation-Value ->HSV or HSL
  - And others (e.g. YUV, CIE)





RGB Model

- Most common in computing
  - Most monitors and projectors
  - Additive colour model
  - Not perceptually uniform
- Record/transmit RGB channels
  - Aligns to the types of cone in the eye
  - Mixes three primary light colours
  - Can represent most colours (not all?)
  - Typically within the range [0,255] or [0,1]
    - E.g. Red as (255,0,0) or (1,0,0)
  - Can be abbreviated as hex values:
    - (183, 24, 92) -> #B7185C



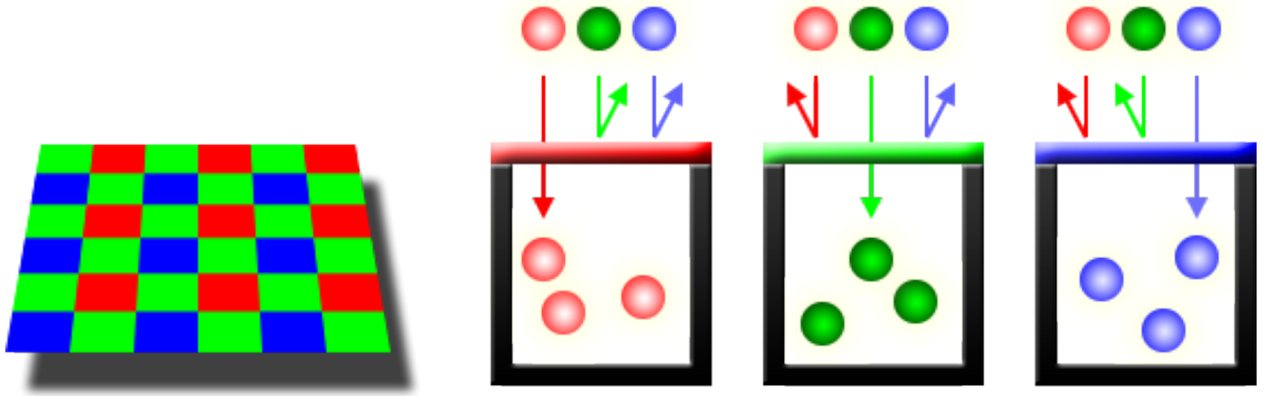
RGB Model

- Most common in computing
  - Most monitors and projectors
  - Additive colour model
  - Not perceptually uniform
- Record/transmit RGB channels
  - Aligns to the types of cone in the eye
  - Mixes three primary light colours
  - Can represent most colours (not all?)
  - Typically within the range [0,255] or [0,1]
    - E.g. Red as (255,0,0) or (1,0,0)
  - Can be abbreviated as hex values:
    - (183, 24, 92) -> #B7185C

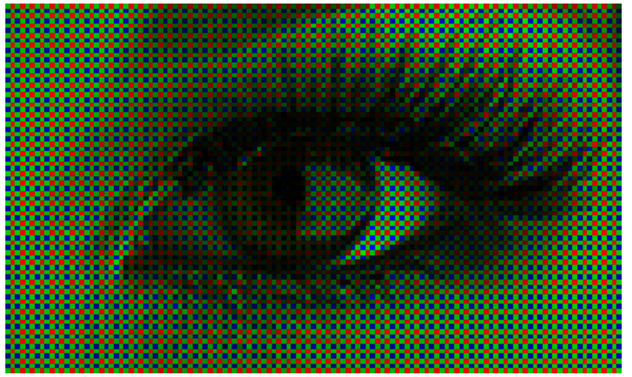
www.iphonearena.com

RGB Model

- Most common in computing
  - Most monitors and projectors
  - Additive colour model
  - Not perceptually uniform
- Record/transmit RGB channels
  - Aligns to the types of cone in the eye
  - Mixes three primary light colours
  - Can represent most colours (not all?)
  - Typically within the range [0,255] or [0,1]
    - E.g. Red as (255,0,0) or (1,0,0)
  - Can be abbreviated as hex values:
    - (183, 24, 92) -> #B7185C



www.red.com

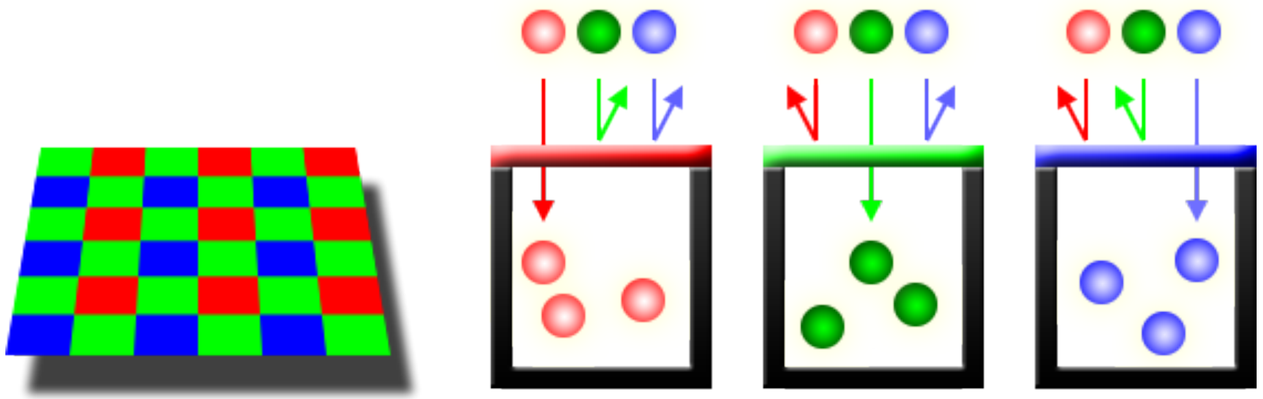


Bayer Image

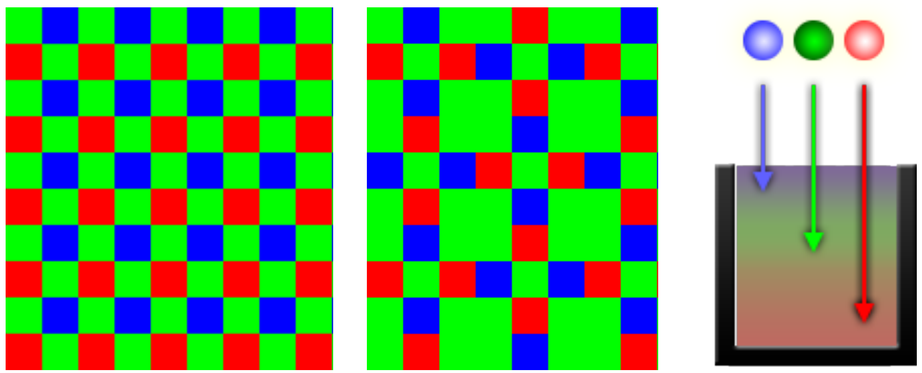
After Demosaicing

RGB Model

- Most common in computing
  - Most monitors and projectors
  - Additive colour model
  - Not perceptually uniform
- Record/transmit RGB channels
  - Aligns to the types of cone in the eye
  - Mixes three primary light colours
  - Can represent most colours (not all?)
  - Typically within the range [0,255] or [0,1]
    - E.g. Red as (255,0,0) or (1,0,0)
  - Can be abbreviated as hex values:
    - (183, 24, 92) -> #B7185C



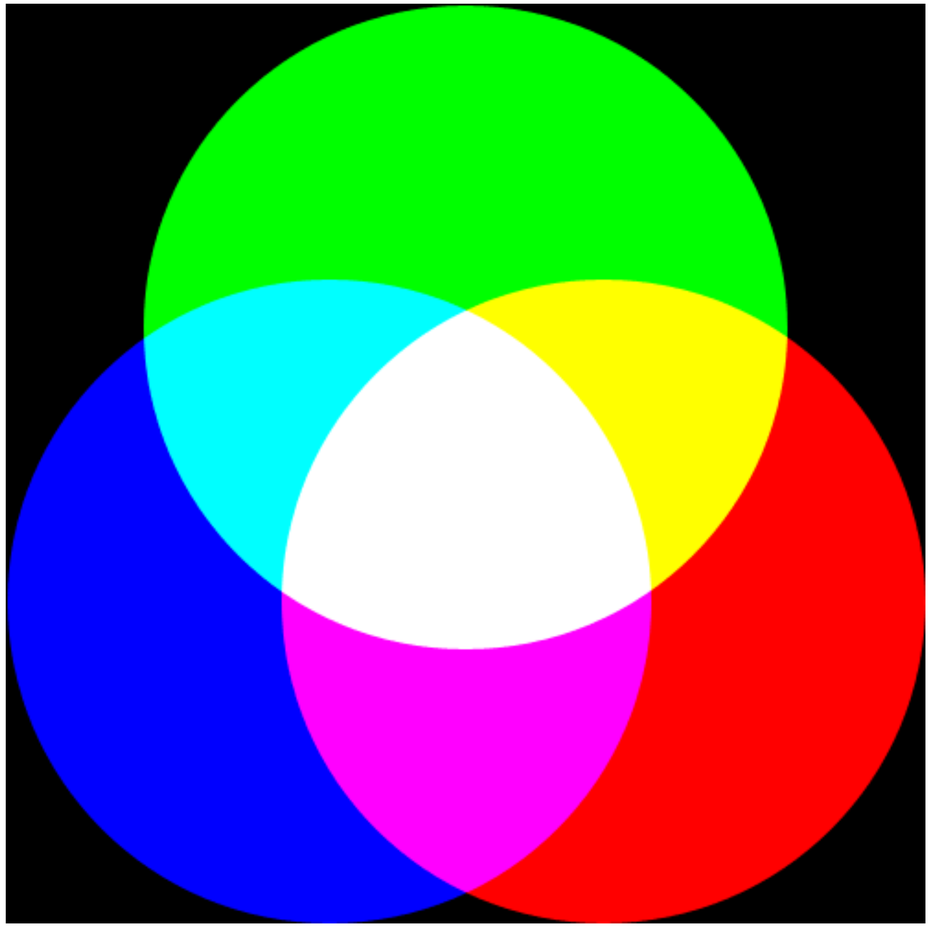
www.red.com



Bayer Pattern X Trans (Fuji) Foveon/Sigma

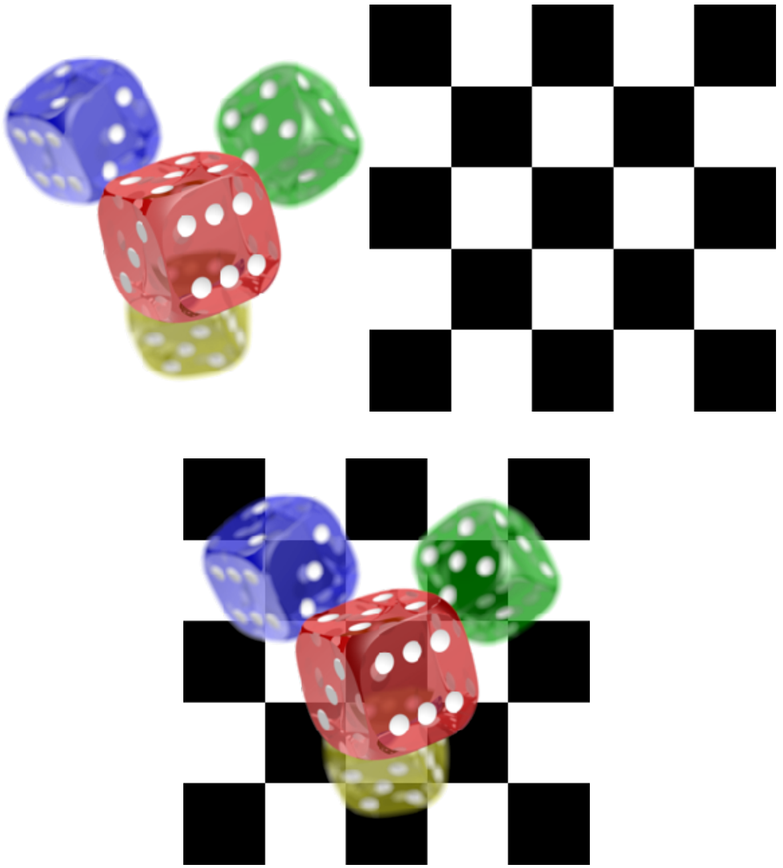
RGB Model

- Most common in computing
  - Most monitors and projectors
  - Additive colour model
  - Not perceptually uniform
- Record/transmit RGB channels
  - Aligns to the types of cone in the eye
  - Mixes three primary light colours
  - Can represent most colours (not all?)
  - Typically within the range [0,255] or [0,1]
    - E.g. Red as (255,0,0) or (1,0,0)
  - Can be abbreviated as hex values:
    - (183, 24, 92) -> #B7185C



RGBA - Transparency

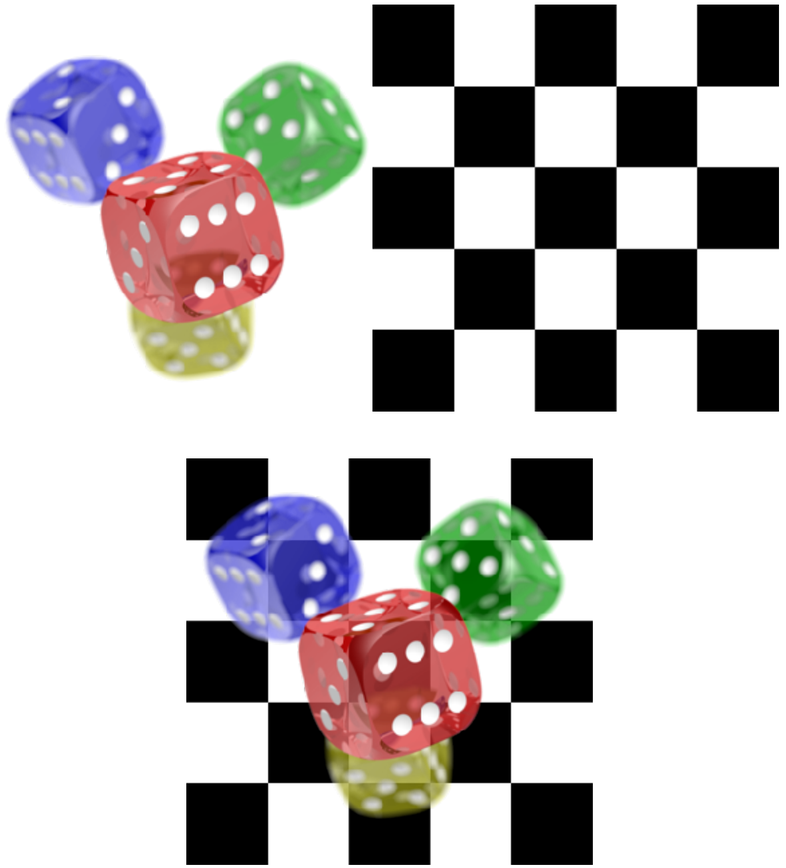
- Adds a fourth ‘colour’ channel
- Typically called the ‘alpha channel’ α
- Combining images is alpha-blending
- transparent $\alpha = 0$, opaque $\alpha = 1$
- ARGB or RGBA common byte orders



RGBA - Transparency

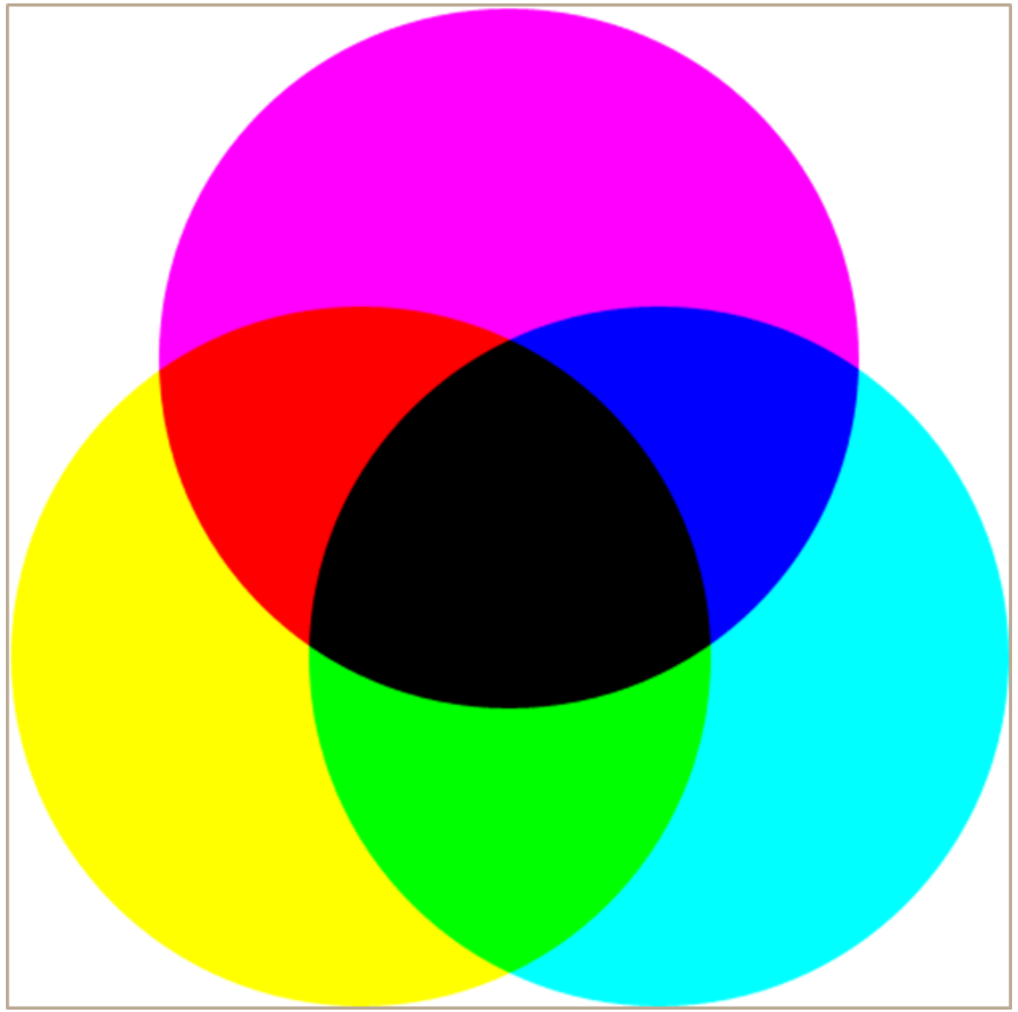
- How do we combine colours?
  - Foreground/front: $(\alpha_f, r_f, g_f, b_f)$
  - Background/back: $(\alpha_b, r_b, g_b, b_b)$
  - Output colour is:

$$r_o = \left(\alpha_f r_f + \left(1 - \alpha_f\right)\alpha_b r_b\right)/\alpha_o$$
$$g_o = \left(\alpha_f g_f + \left(1 - \alpha_f\right)\alpha_b g_b\right)/\alpha_o$$
$$b_o = \left(\alpha_f b_f + \left(1 - \alpha_f\right)\alpha_b b_b\right)/\alpha_o$$
$$\alpha_o = \alpha_f + \left(1 - \alpha_f\right)\alpha_b$$



CMY Colour Space

- Most common for printing
  - Subtractive Colour Model
  - Not perceptually uniform
- CMY is the inverse of RGB
  - Cyan Magenta Yellow (Black)
  - Cyan pigment absorbs red light
  - Magenta pigment absorbs green
  - Yellow pigment absorbs blue
- In theory C+M+Y = Black
  - In practice a ‘true Black’ ink is used
  - Black is represented by K (Key)



CMY Colour Space

- Most common for printing
  - Subtractive Colour Model
  - Not perceptually uniform
- CMY is the inverse of RGB
  - Cyan Magenta Yellow (Black)
  - Cyan pigment absorbs red light
  - Magenta pigment absorbs green
  - Yellow pigment absorbs blue
- In theory C+M+Y = Black
  - In practice a ‘true Black’ ink is used
  - Black is represented by K (Key)

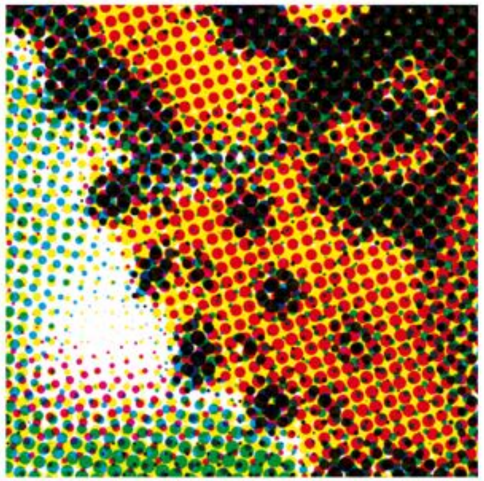
CYAN

MAGENTA

YELLOW

BLACK

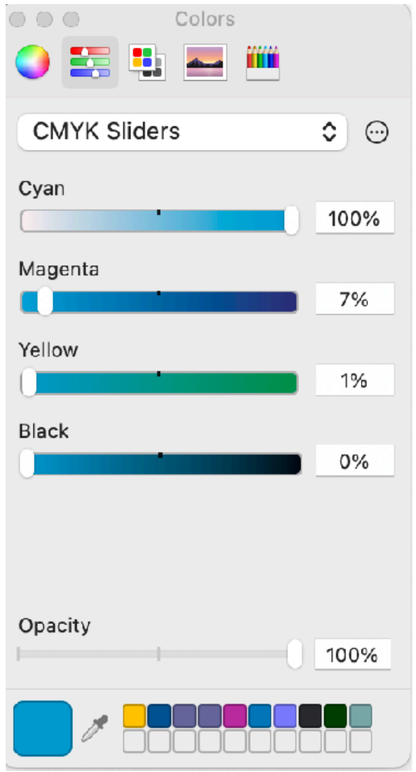
FINAL CMYK



DETAIL VIEW

CMY Colour Space

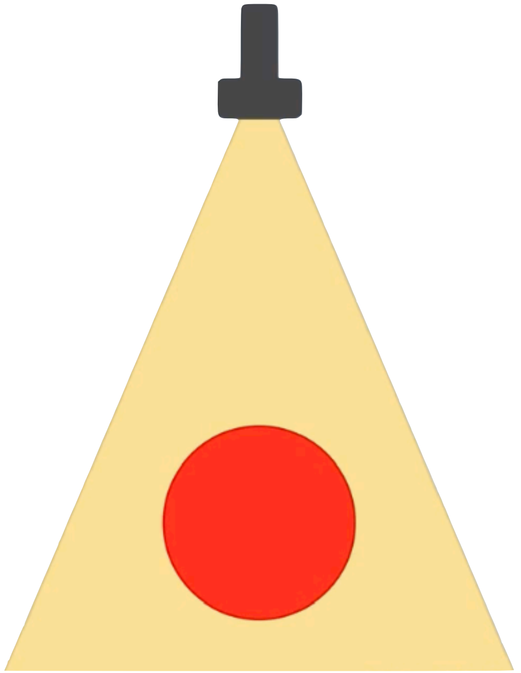
- Most common for printing
  - Subtractive Colour Model
  - Not perceptually uniform
- CMY is the inverse of RGB
  - Cyan Magenta Yellow (Black)
  - Cyan pigment absorbs red light
  - Magenta pigment absorbs green
  - Yellow pigment absorbs blue
- In theory C+M+Y = Black
  - In practice a ‘true Black’ ink is used
  - Black is represented by K (Key)



Color	Color name	(R,G,B)	Hex	(C,M,Y,K)
	Black	(0,0,0)	#000000	(0,0,0,1)
	White	(255,255,255)	#FFFFFF	(0,0,0,0)
	Red	(255,0,0)	#FF0000	(0,1,1,0)
	Green	(0,255,0)	#00FF00	(1,0,1,0)
	Blue	(0,0,255)	#0000FF	(1,1,0,0)
	Yellow	(255,255,0)	#FFFF00	(0,0,1,0)
	Cyan	(0,255,255)	#00FFFF	(1,0,0,0)
	Magenta	(255,0,255)	#FF00FF	(0,1,0,0)

Coloured Light on Coloured Object

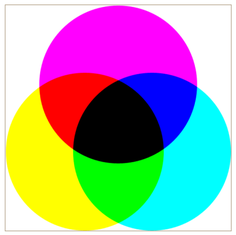
- What do I see if I shine a yellow light on a red ball?



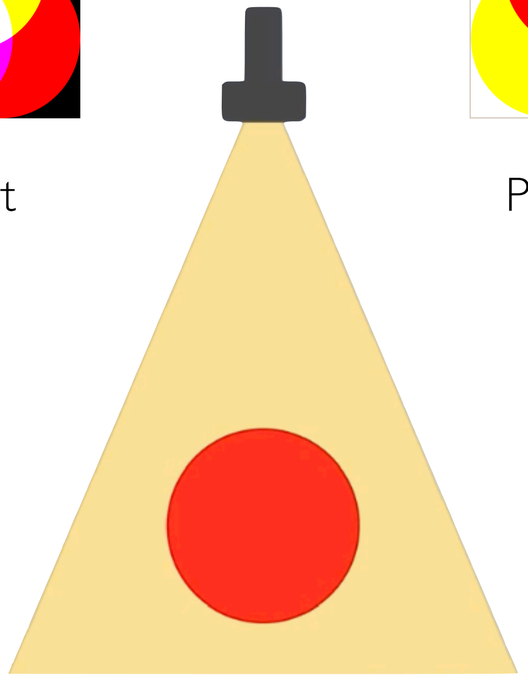
Coloured Light on Coloured Object

- What do I see if I shine a yellow light on a red ball?
  - Yellow light = Red + Green
  - Red ball = Magenta + Yellow pigment
  - There is no Blue light -> B = 0
  - Green light is absorbed by Magenta
  - Red light is not absorbed -> reflected
  - So we see Red

Light

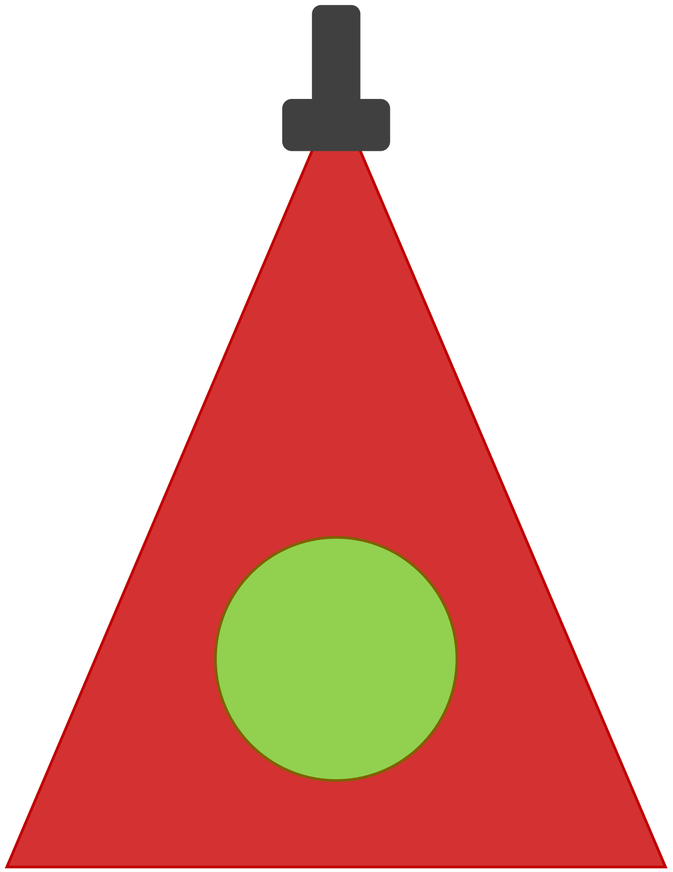


Pigment



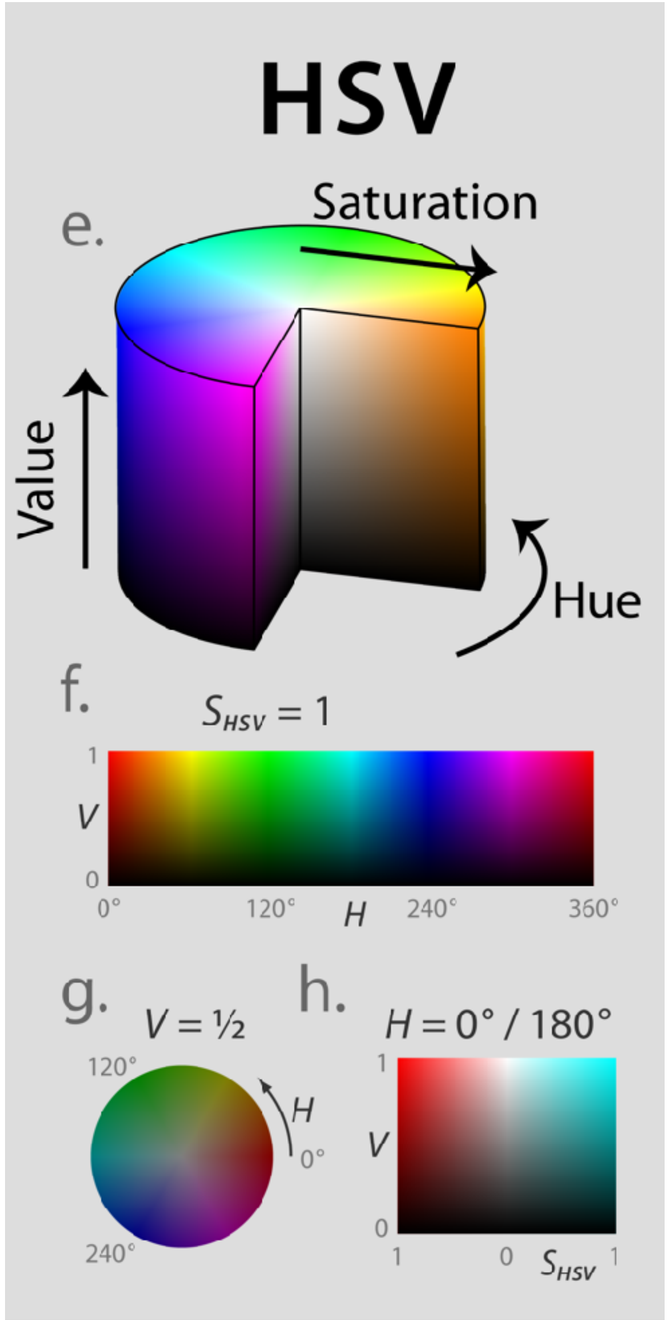
Coloured Light on Coloured Object

- Think about
  - What colour would you see for:
  - Shining a red light on a green object?
  - A cyan light on a yellow object?
  - Does the order matter when alpha blending colours?



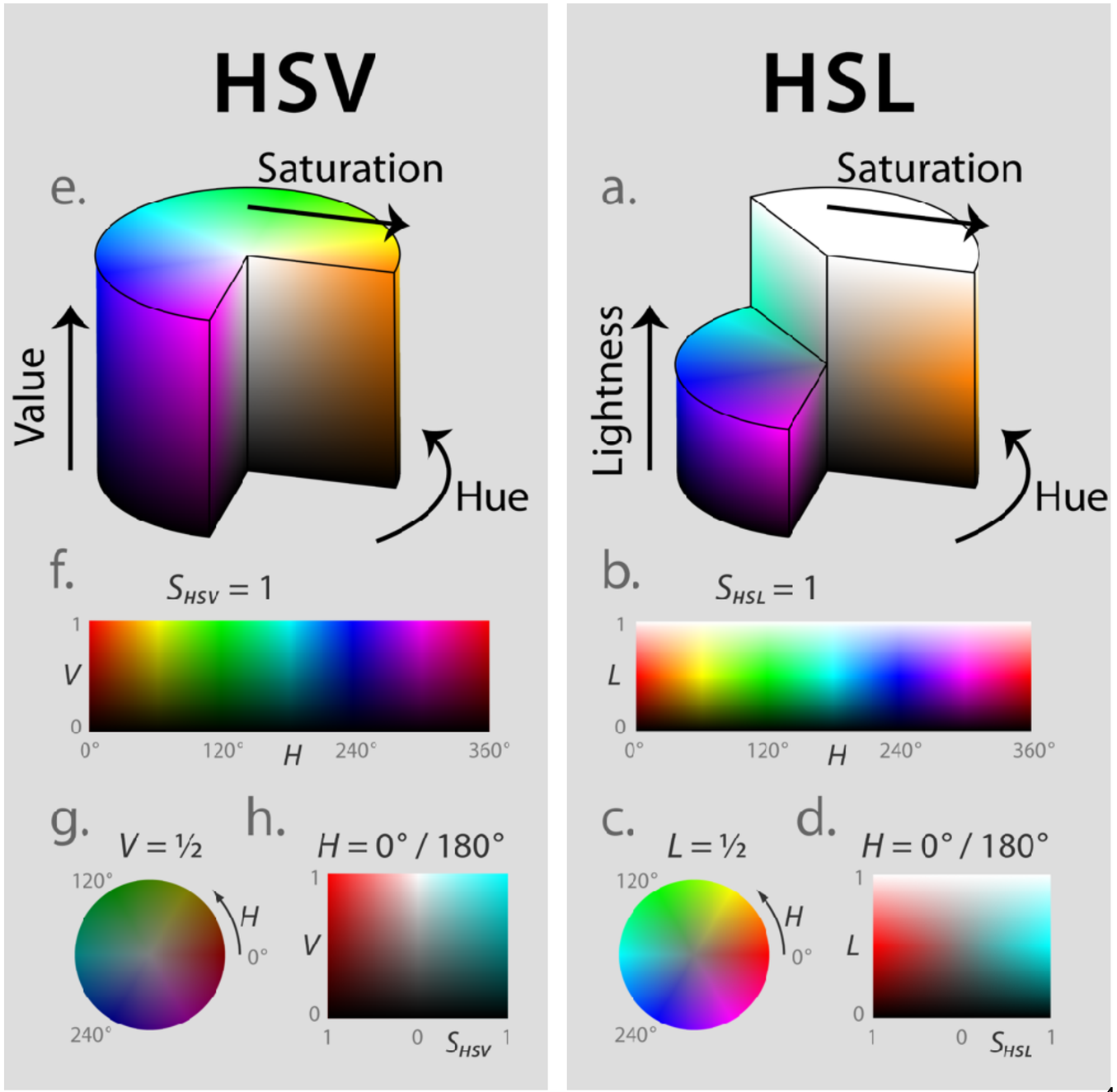
HSV Colour Space

- How do we describe colour?
  - The sky is a bright blue
  - The trees are a dark green
  - That shirt is a bold colour
- HSV is closer to this:
  - Hue – what general colour is it
  - Saturation – how strong the colour is (distance from grey)
  - Value – how light or dark it is
  - Not perceptually uniform
- Also HSL, L = lightness/luminance
  - from dark to light



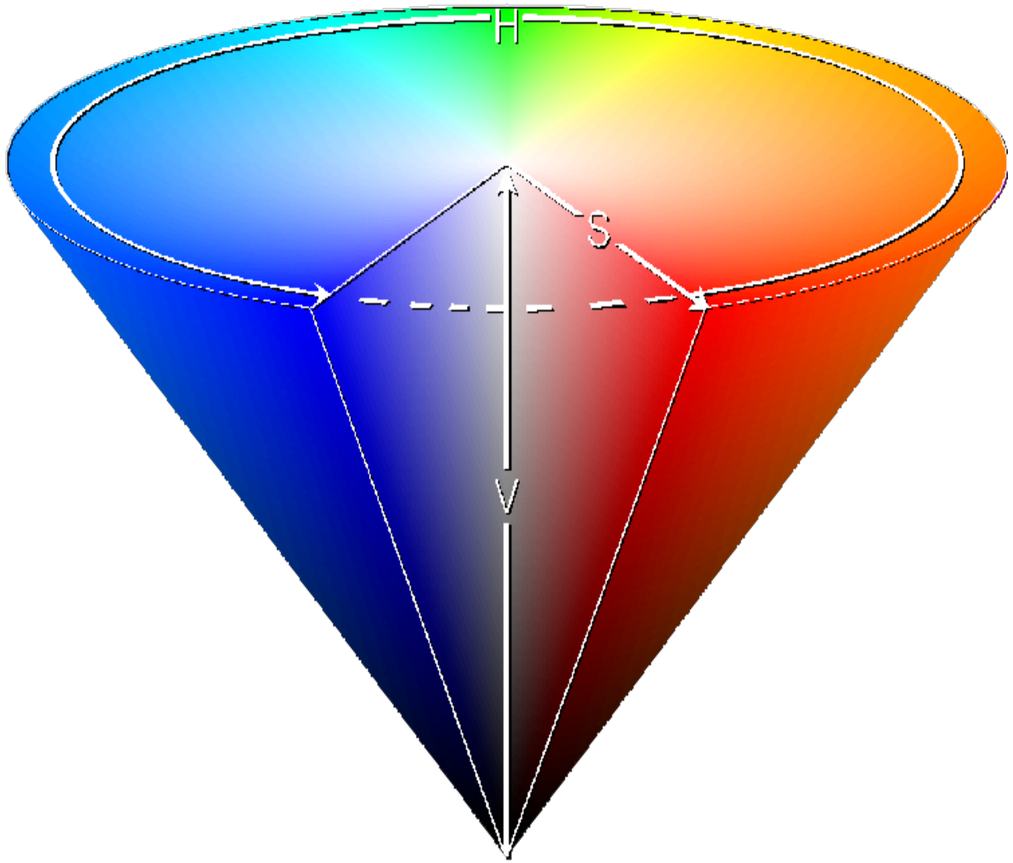
HSV Colour Space

- How do we describe colour?
  - The sky is a bright blue
  - The trees are a dark green
  - That shirt is a bold colour
- HSV is closer to this:
  - Hue – what general colour is it
  - Saturation – how strong the colour is (distance from grey)
  - Value – how light or dark it is
  - Not perceptually uniform
- Also HSL, L = lightness/luminance
  - from dark to light



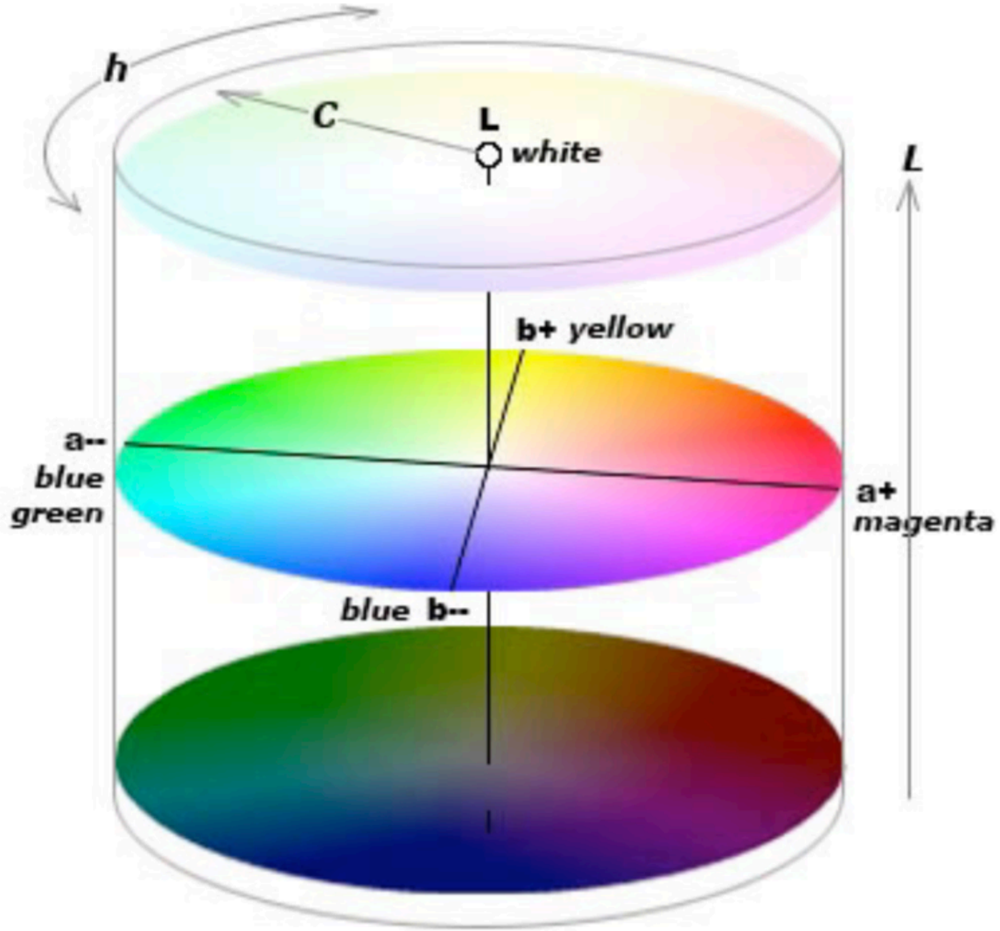
HSV Colour Space

- How do we describe colour?
  - The sky is a bright blue
  - The trees are a dark green
  - That shirt is a bold colour
- HSV is closer to this:
  - Hue – what general colour is it
  - Saturation – how strong the colour is (distance from grey)
  - Value – how light or dark it is
  - Not perceptually uniform
- Also HSL, L = lightness/luminance
  - from dark to light



CIELAB Colour Space

- Also LAB or L*a*b*
- Defined by International Commission on Illumination (CIE)
- Perceptually uniform space:
  - Numerical change corresponds to a similar perceived change in colour
- L* (lightness) coordinate normally ranges from [0,100]
- a* (red-greed) and b* (yellow blue) coordinates commonly clamped to [-128,127] or [0,255]

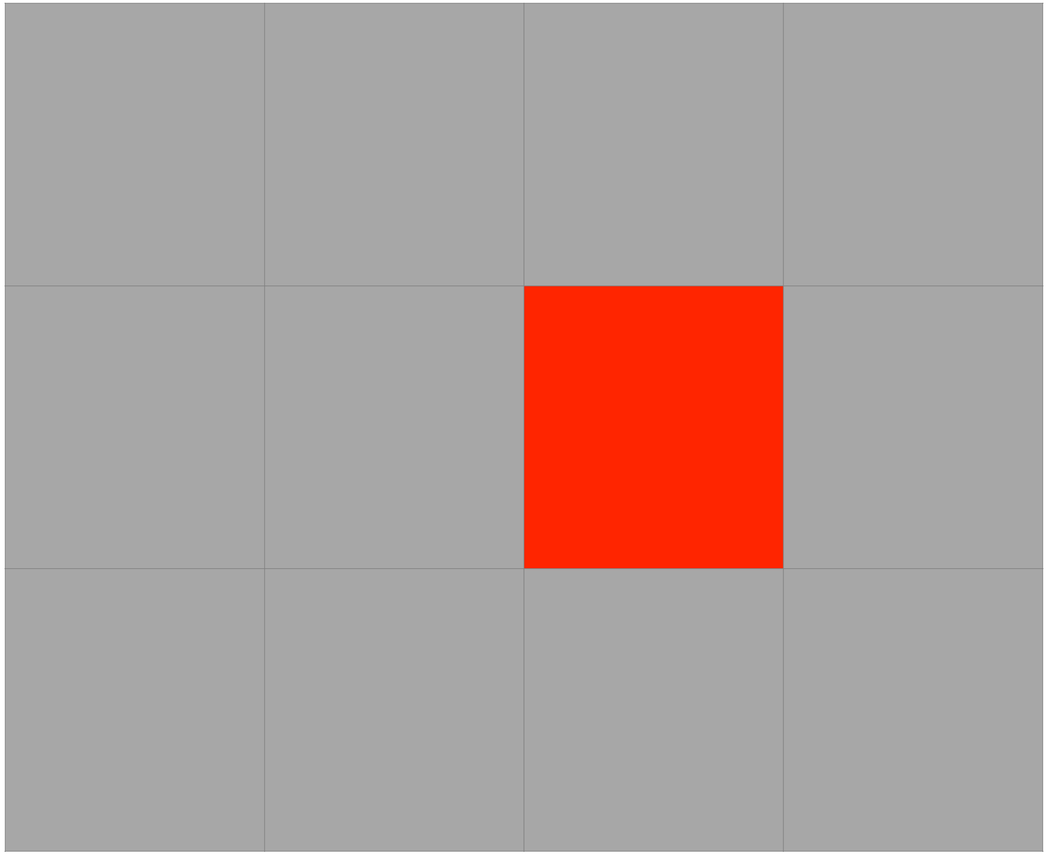


Back to
2D Image Representation

Digital Images and 2D Image Representation

- 2D array of pixels – a raster
- Pixels have a value, or list of values
  - value: usually brightness
  - values: brightness, colour, transparency

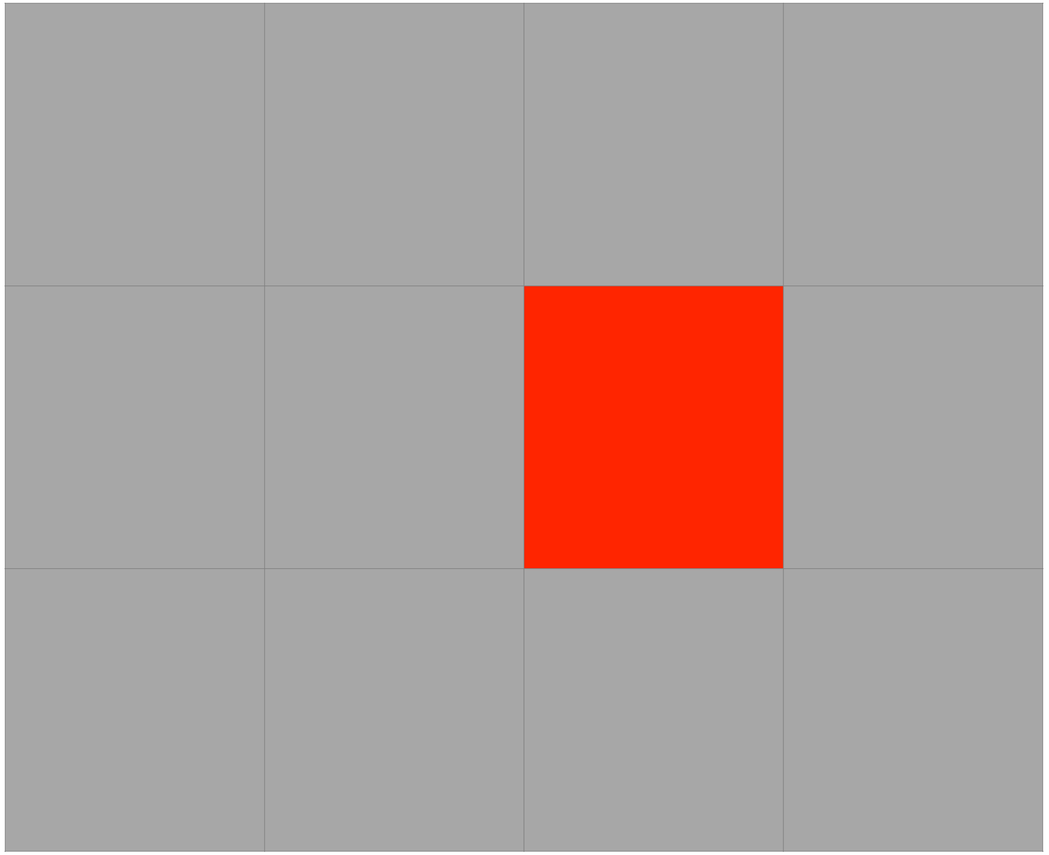
```
# Create a 2D array
image2D = create_array(height, width)

# Fill with example grayscale values
for y from 0 to height-1:
    for x from 0 to width-1:
        image2D[y][x] = 125

# Access pixel at (row=1, col=2)
image2D[1][2]= ?
```



Digital Images and 2D Image Representation

- 2D array of pixels – a raster
- Pixels have a value, or list of values
  - value: usually brightness
  - values: brightness, colour, transparency

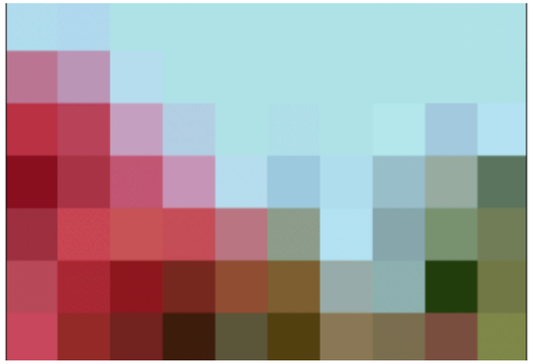
```
# Create a 2D array
image2D = create_array(height, width)

# Fill with example grayscale values
for y from 0 to height-1:
    for x from 0 to width-1:
        image2D[y][x] = 125

# Access pixel at (row=1, col=2)
image2D[1][2]= (255,0,0)
```

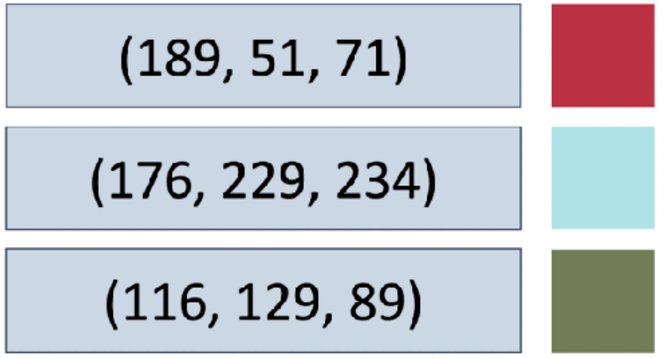


Digital Images and 2D Image Representation

- 2D array pixels – a raster
- Pixels have a value, or list of values

- One value – greyscale image
- Colour images – typically 3 values (RGB)
- Values may be integers or floats
  - Integers in the range [0,255] (8bit / 1byte)
  - Floats in the range [0,1]
- High Dynamic Range:
  - Higher bit depth of 10,12,14bit or more per colour channel





Digital Images and 2D Image Representation

- Images can be very large
- ‘Raw’ images are ~3 bytes per pixel
- 4K image (3840 × 2160) is ~24 MB
- Video is worse – 1 hour HD @ 25 fps:
  1920 × 1080 × 3 × 25 × 60 × 60 ≈ 560 GB
  3840 × 2160 × 3 × 25 × 60 × 60 ≈ 2.3 TB

- High Dynamic Range?
- 4K image (3840 × 2160) is ~36MB (at 12bit)

Digital Images and 2D Image Representation

- Images can be very large
- ‘Raw’ images are ~3 bytes per pixel
- 4K image (3840 × 2160) is ~24 MB
- Video is worse – 1 hour HD @ 25 fps:
  1920 × 1080 × 3 × 25 × 60 × 60 ≈ 560 GB
  3840 × 2160 × 3 × 25 × 60 × 60 ≈ 2.3 TB

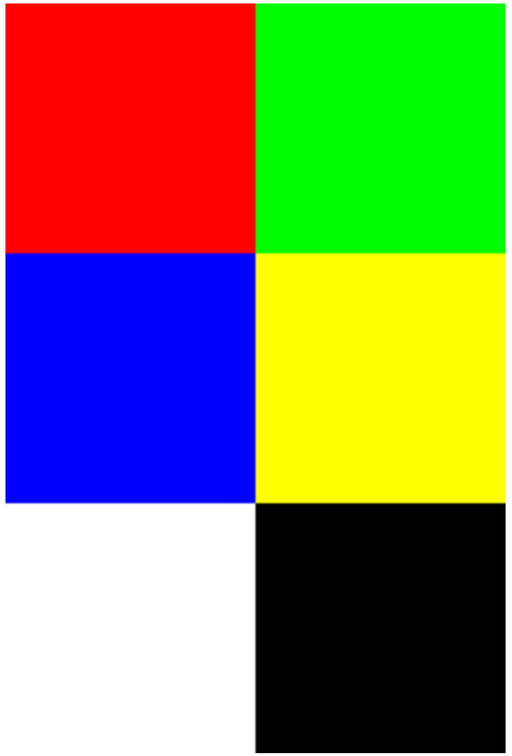
- Compression reduces file sizes
- Lossless compression
- Lossy compression
- Lossless video compression is rare

Image File Formats

Format	*Compression*	*Notes*
JPEG / JPG	Lossy	Widely used for photos; high compression with quality trade-offs; supports EXIF metadata
WebP	Lossy or Lossless	Google-developed; better compression than JPEG/PNG; supports transparency and animation
AVIF	Lossy or Lossless	Based on AV1 codec; excellent compression and quality; supports HDR and transparency; growing browser support
PNG	Lossless	Ideal for graphics with sharp edges and transparency; universally supported; open format
GIF	Lossless (palette-based)	Supports animation; limited to 256 colors; largely replaced by APNG or video formats for animation
BMP	Lossless	Simple, uncompressed or RLE; mostly obsolete except for niche or legacy uses
PNM (PBM/PGM/PPM)	None	Very simple, raw or ASCII; mainly for research or tool pipelines
SVG	None (vector)	Scalable vector format for icons, illustrations, and UI; can embed raster images; supports CSS and scripting
High Efficiency Video Coding (HEVC)	Lossy	Used by High Efficiency Video Coding (H.265) but also in Apple HEIF or HEVC image and video files (e.g. heic)

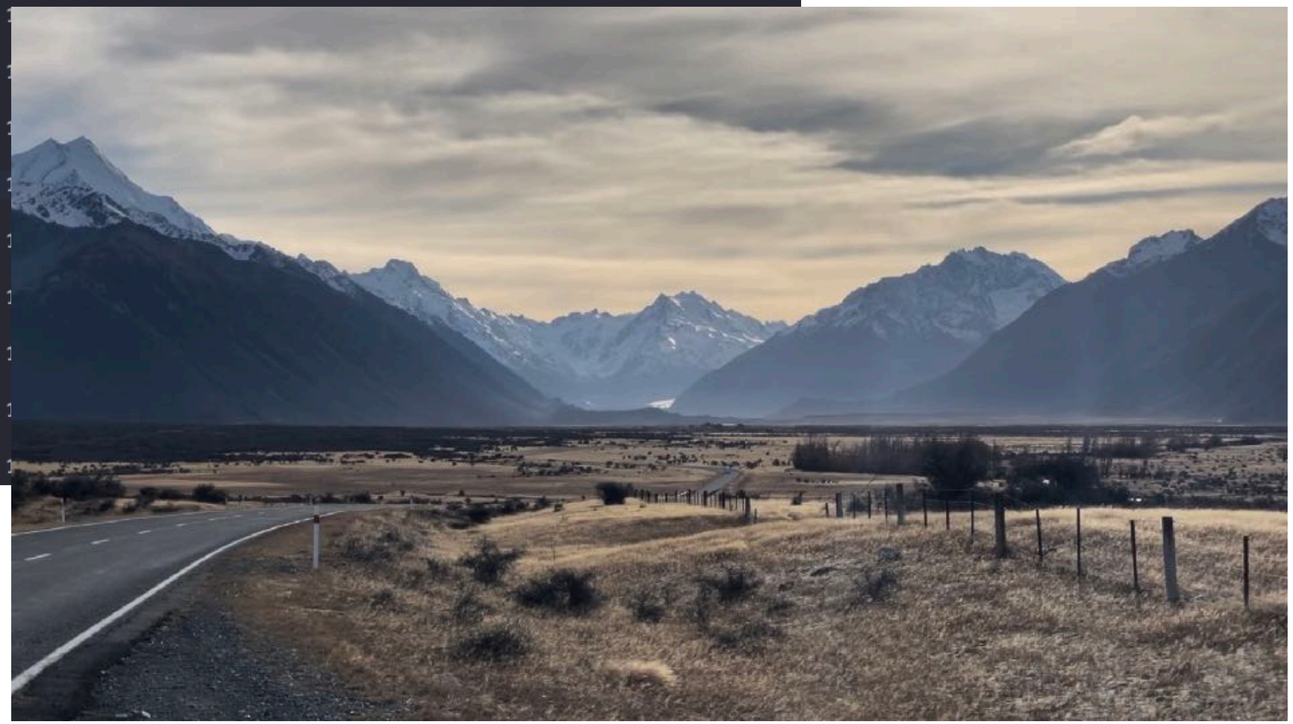
Sample PPM File

```
P3
# ASCII colour file
# 2x3 image, max value 255
2 3 255
# Content as RGB triples
255   0   0       0 255   0
  0   0   255     255 255   0
255 255 255       0   0   0
```



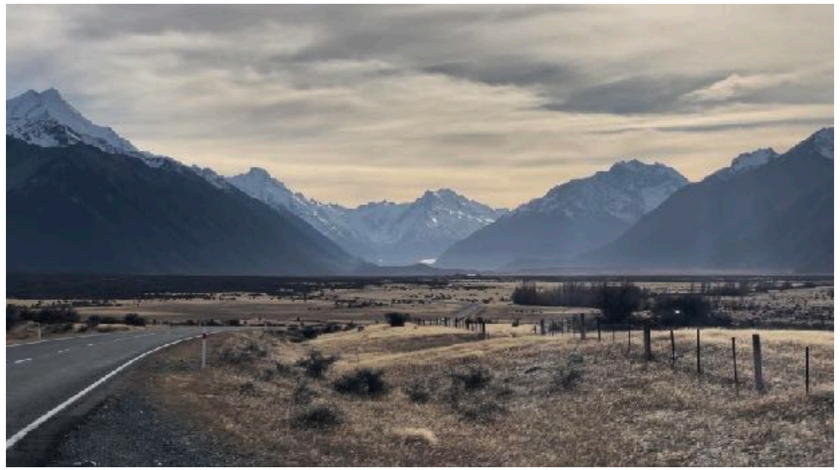
Sample PPM File

```
P3
3433 1902
255
 195 194 192   195 194 192   195 194 192   195 194 192   194 193 191   194 193 191   194 193 191   194 192 193   194 192 193   194 192 193   194 192 193   194 192 193   194
 190   190 190 190   190 190 190   190 190 190   190 188 189   190 188 189   190 188 189   191 189 190   191 189 190   191 189 190   191 189 190   191 189 190   191 189 190
 192 191 189   192 191 189   192 191 189   192 191 189   193 192 190   193 192 190   193 192 190   194 193 191   194 193 191   194 193 191   194 193 191   194 193 191   194
 191   196 192 191   196 192 191   196 192 191   195 191 190   196 192 191   196 192 191   196 192 191   196 192 191   195 191 190   195 191 190   195 191 190   195 191 190
 192 191 189   192 191 189   192 191 189   192 191 189   192 191 189   192 191 189   192 191 189   193 192 190   193 192 190   193 192 190   193 192 190   193 192 190   193
 190   195 194 190   194 193 189   194 193 189   194 193 189   194 193 189   195 194 190   195 194 190   194 193 189   194 193 189   194 193 189   193 192 188   193 192 188
 186 185 183   186 185 183   186 185 183   186 185 183   187 186 184   187 186 184   187 186 184   187 186 184   186 185 183   186 185 183   186 185 183   185 184 182   185
 184   182 182 184   182 182 184   181 181 183   181 181 183   181 181 183   181 181 183   181 181 183   181 181 183   180 180 182   180 180 182   180 180 182
 179 179 181   178 178 180   178 178 180   178 178 180   178 178 180   178 178 180   178 178 180   178 178 180   178 178 180   178 178 180   177 177 179   177 177 179   177
 178   176 176 178   176 176 178   176 176 178   175 176 178   175 176 178   175 176 178   175 176 178   176 176 178   175 175 177   175 175 177   175 175 177   175 175 175
 174 174 176   174 174 176   175 175 177   175 175 177   175 175 177   175 175 177   175 175 177   174 174 176   174 174 176   174 174 176   174 174 176   174 174 176   173
 166   164 164 166   164 164 166   164 164 166   164 164 166   163 163 165   163 163 165   163 163 165   162 162 164   162 162 164   162 162 164   162 162 164   162 162 164
 158 159 163   158 159 163   158 159 163   158 159 163   157 158 162   157 158 162   157 158 162   157 1
 160   155 156 160   155 156 160   155 156 160   155 156 160   154 155 159   154 155 159   154 155 159
 152 153 157   152 153 157   152 153 157   152 153 157   152 153 157   152 153 157   152 153 157   152 1
 154   151 152 154   151 152 154   151 152 154   151 152 156   151 152 156   151 152 156   151 152 156   151 152 156
 151 152 156   151 152 156   151 152 156   151 152 156   151 152 156   151 152 156   151 152 156   151 1
 156   151 152 156   151 152 156   151 152 156   151 152 156   151 152 156   151 152 156   151 152 156   151 152 156
 154 153 158   154 153 158   154 153 158   154 153 158   154 153 158   154 153 158   154 153 158   154 1
 154   152 152 154   152 152 154   152 152 154   152 152 154   152 151 156   152 151 156   153 152 157
 151 150 155   151 150 155   151 150 155   151 150 155   151 150 155   151 150 155   151 150 155   152 1
 152   148 147 152   148 147 152   148 147 152   148 147 152   148 147 152   148 147 152   148 147 152
 144 145 149   144 145 149   144 145 149   144 145 149   144 145 149   144 145 149   144 145 149   144 1
 148   143 144 148   143 144 148   143 144 148   143 144 148   143 144 148   143 144 148   143 144 148
 144 143 148   144 143 148   144 143 148   143 144 146   143 144 146   143 144 146   143 144 146   143 1
 147   145 145 147   145 145 147   145 145 147   145 145 147   145 145 147   145 145 147   145 145 147
 149 149 151   149 149 151   149 149 151   149 149 151   149 149 151   149 149 151   149 149 151   149 1
 153   151 151 153   151 151 153   151 151 153   152 152 154   152 152 154   152 152 154   152 152 154
 153 153 153   153 153 153   153 153 153   153 153 153   153 153 153   154 154 154   154 154 154   153 1
```



JPEG Compression Artifacts

Quality factor 100

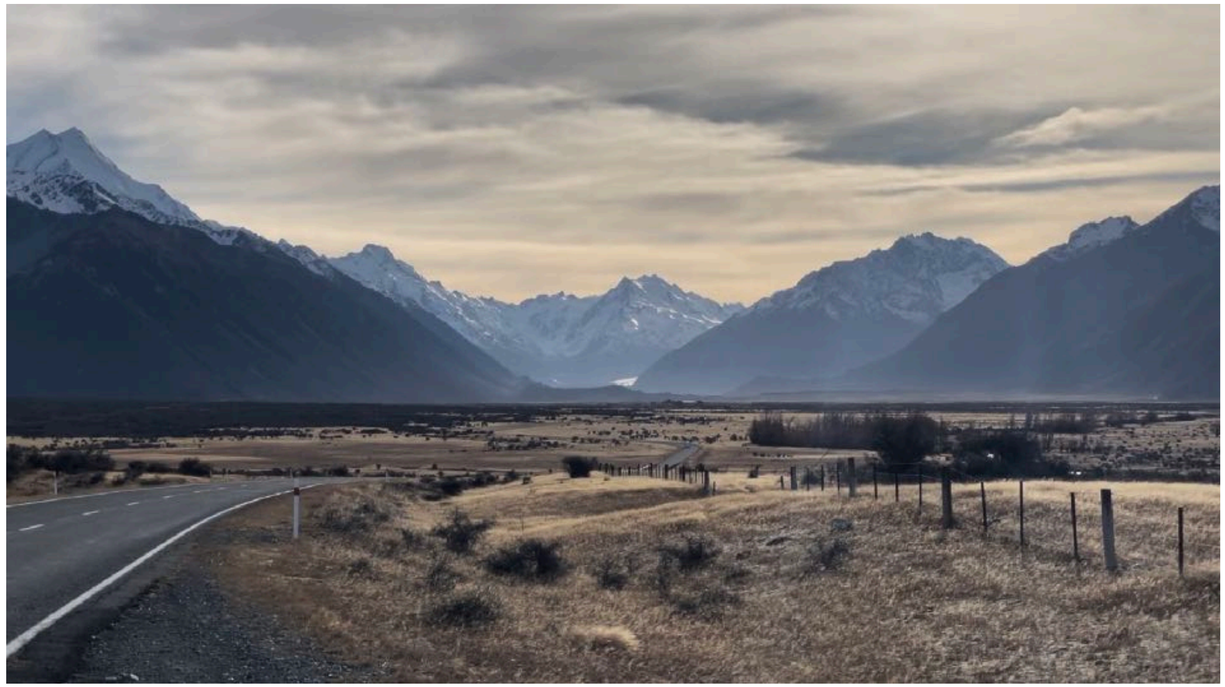


Quality factor 50

Quality factor 10

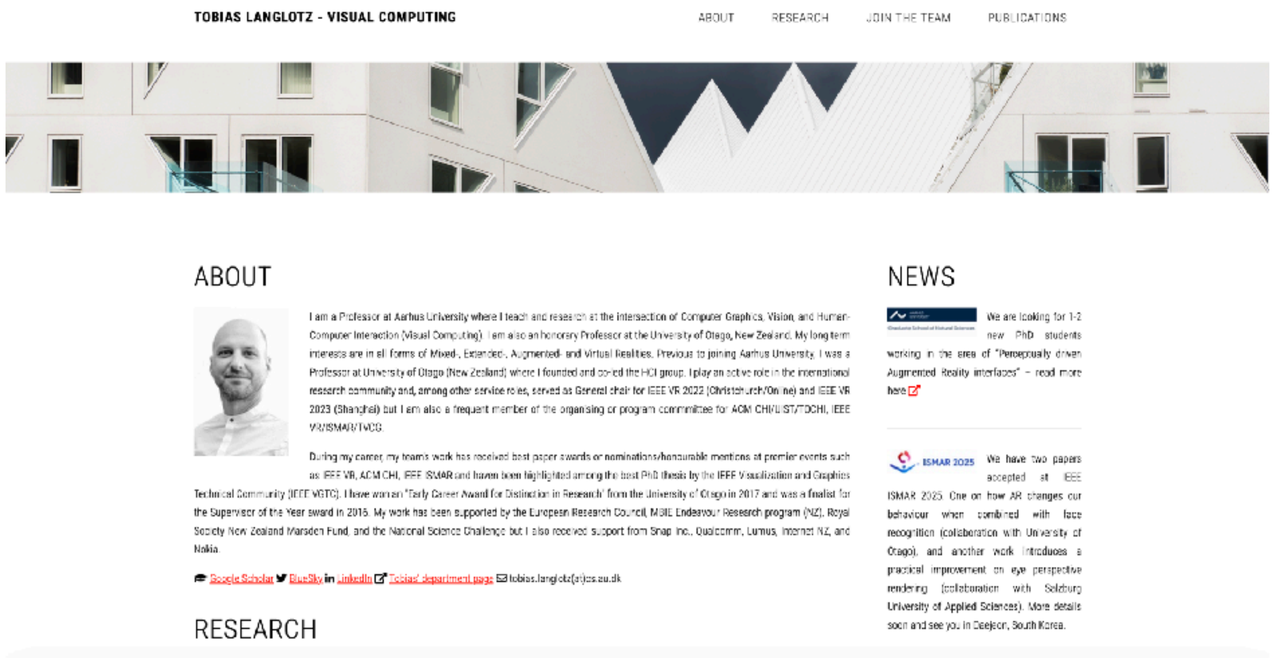
Format Size Comparison

Format	*Size*
JPEG (QF 90)	*995 KB*
JPEG (QF 50)	*379 KB*
JPEG (QF 10)	*205 KB*
GIF	*2.3 MB*
PNG	*4.9 MB*
BMP	*19.5 MB*
PPM (Binary)	*19.7 MB*
PPM (ASCII)	*91.4 MB*



Task

- Think about
  - What file format(s) would be best
    - For photographs on a website?
    - For logos?
    - For professional photo editing?
  - For scientific / medical imaging?
  - What about videos?
    - Is a series of JPEGs good enough?
    - What could help us do better?

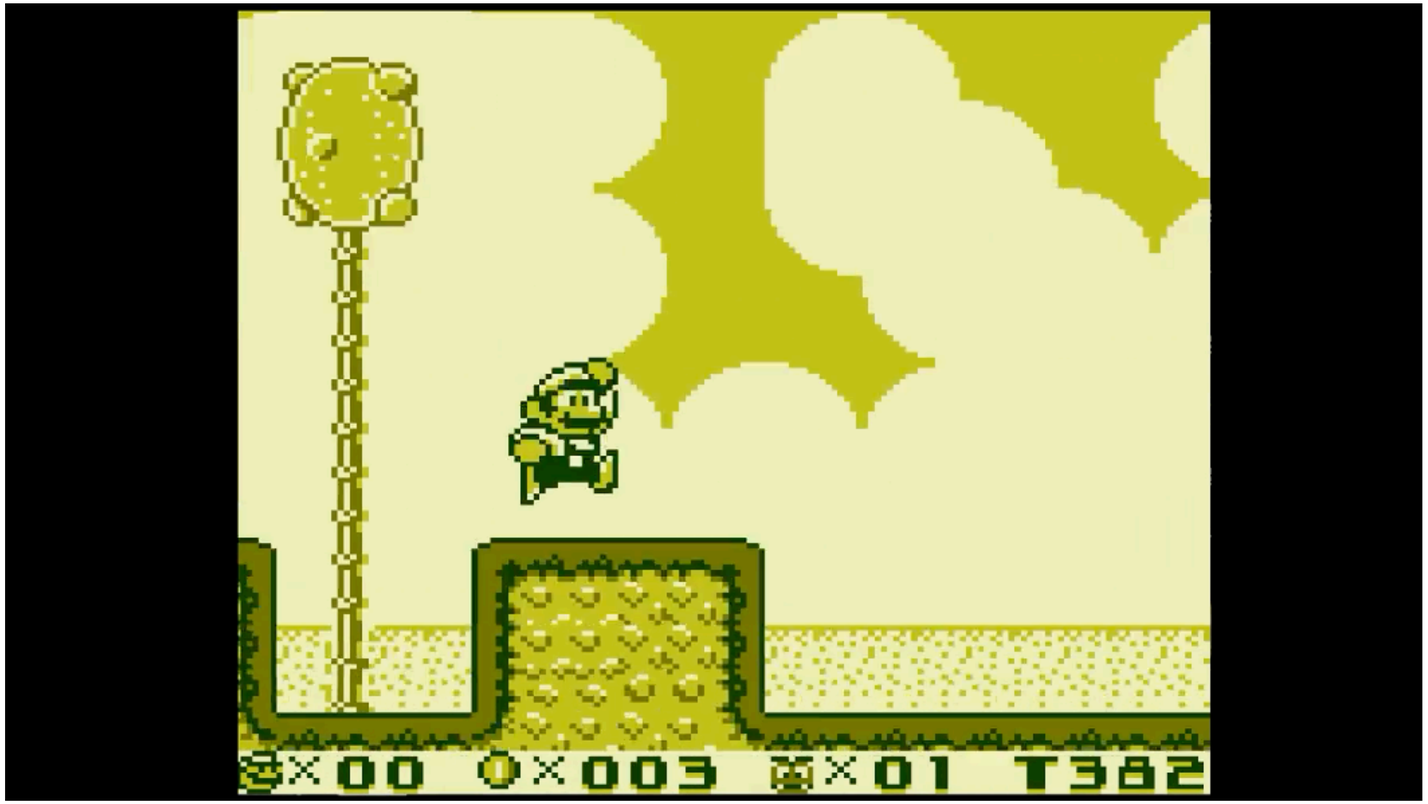


AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Sidetrack:
Can you do something cool with it?

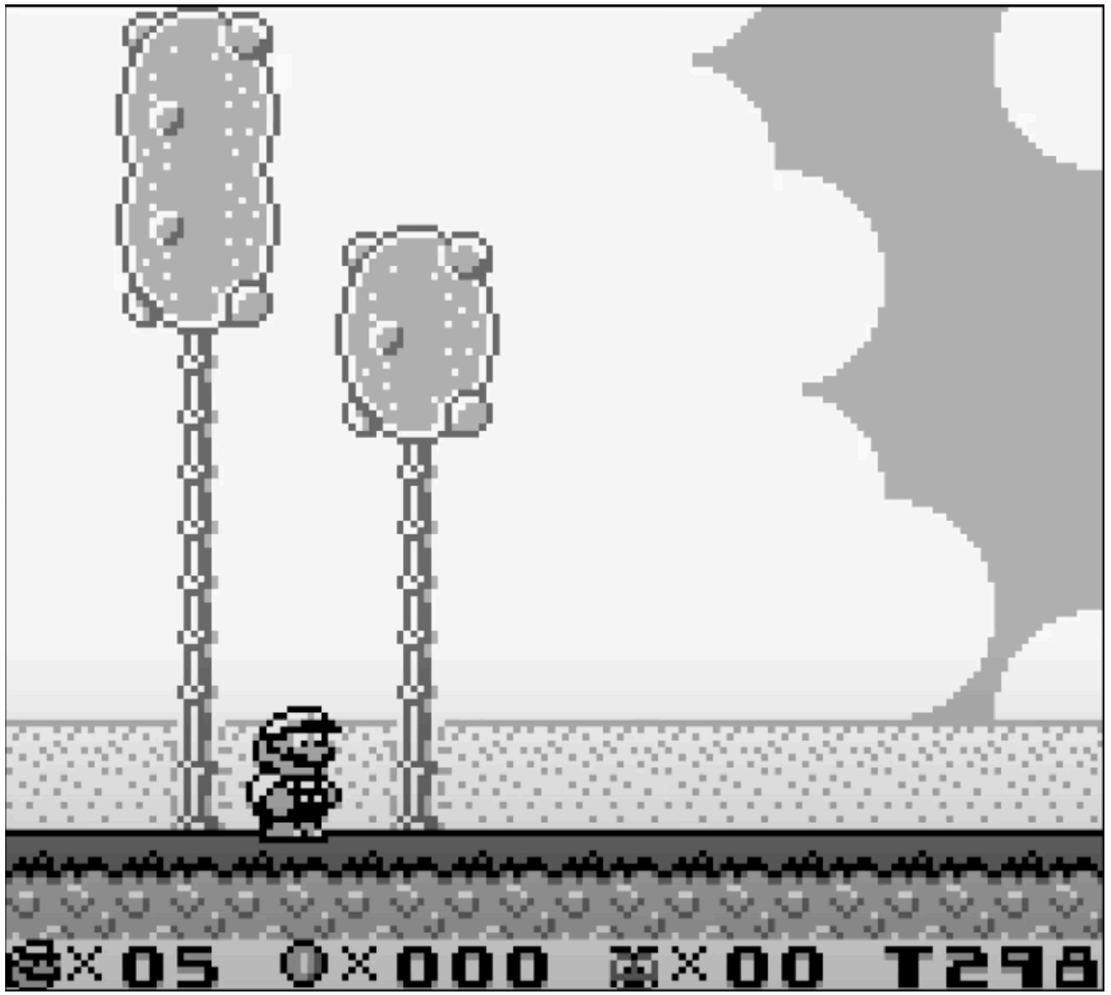
Case study: Retro Gaming / Gameboy

- Context:
  - Implement a 2D game using only 2D graphics (images)



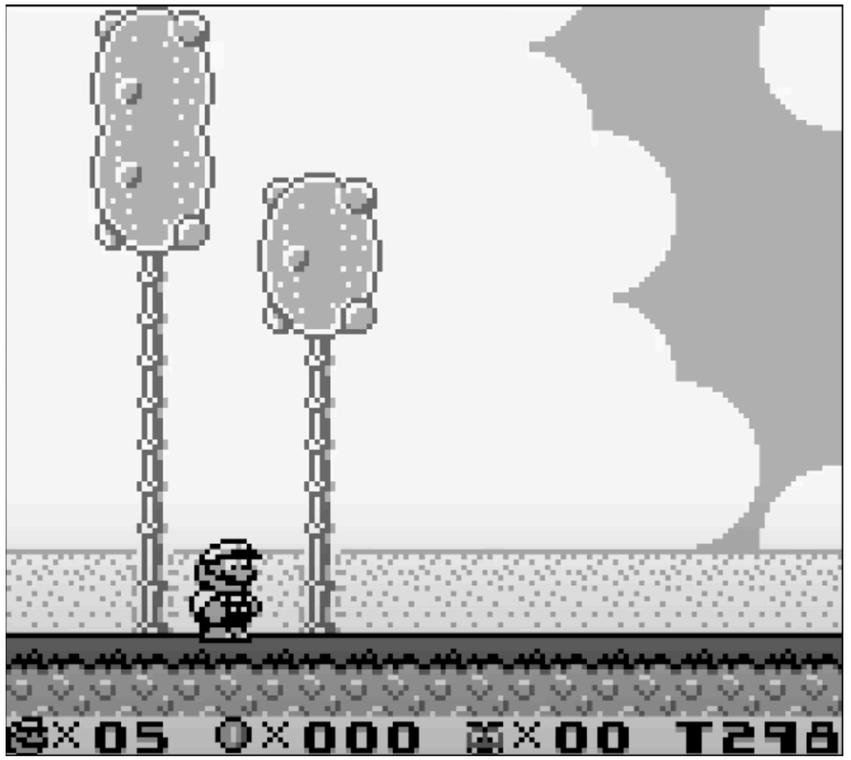
Case study: Retro Gaming / Gameboy

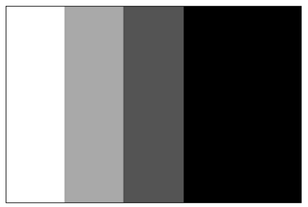
- Context:
  - No 3D graphics
  - Slow hardware
  - Grey scale (2 bits, 4 shades)
  - 160x144 pixel LCD screen
  - Tile-based rendering (images are constructed from small 8x8 pixel tiles, total 20x18 tiles)
  - Three layers: background, window, and objects (sprites)

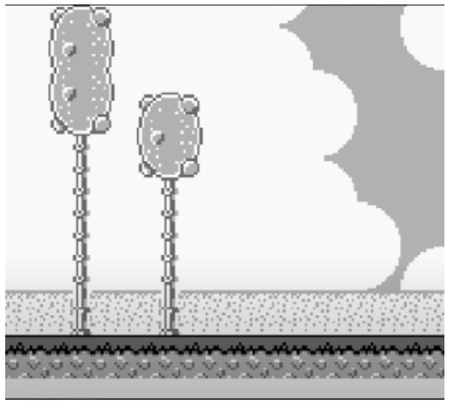


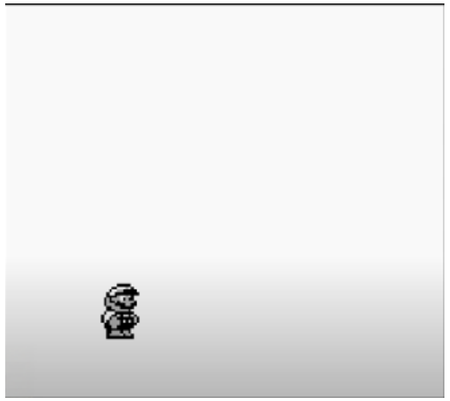
Case study: Retro Gaming / Gameboy

- Context:
  - No 3D graphics
  - Slow hardware
  - Grey scale (2 bits, 4 shades)
  - 160x144 pixel LCD screen
  - Tile-based rendering (images are constructed from small 8x8 pixel tiles, total 20x18 tiles)
  - Three layers: background, window, and objects (sprites)



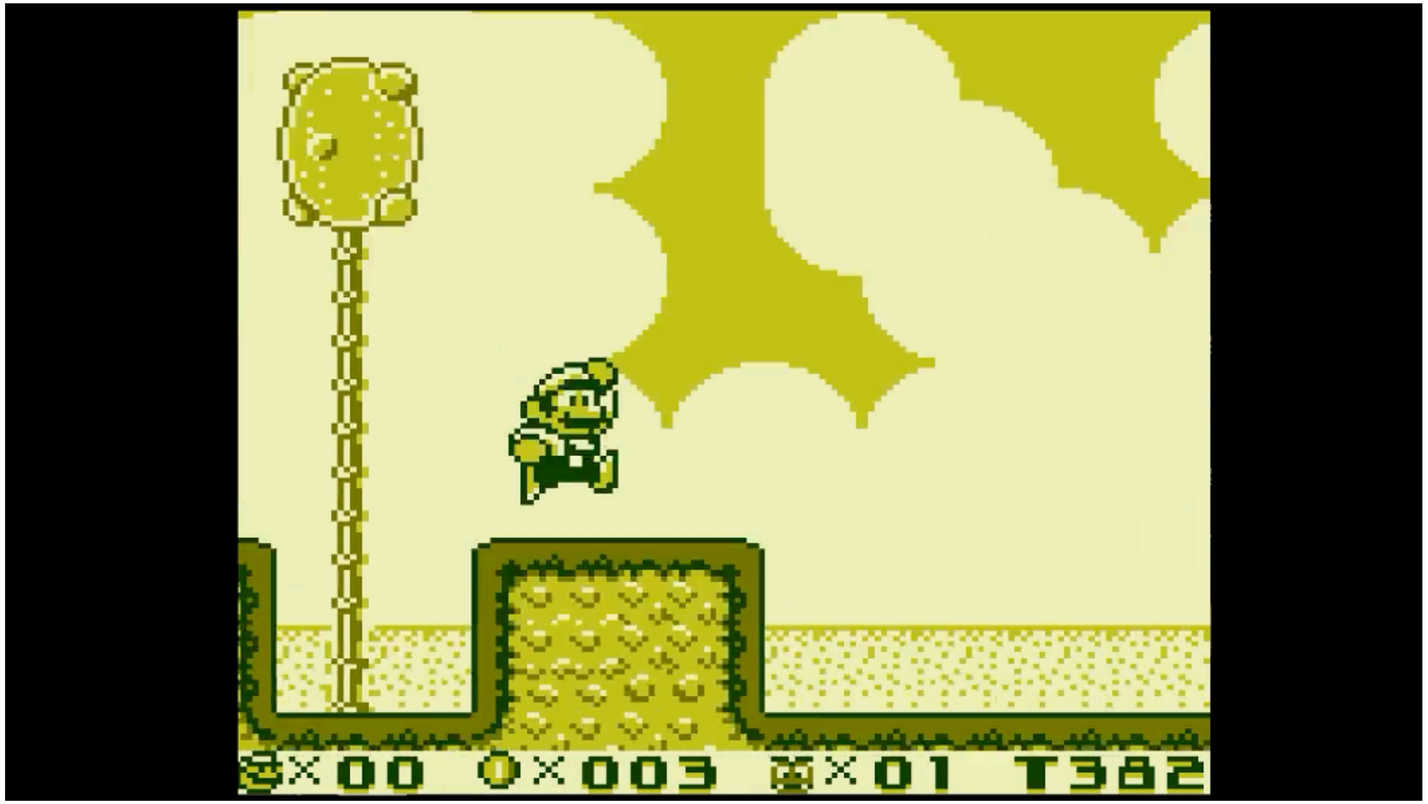






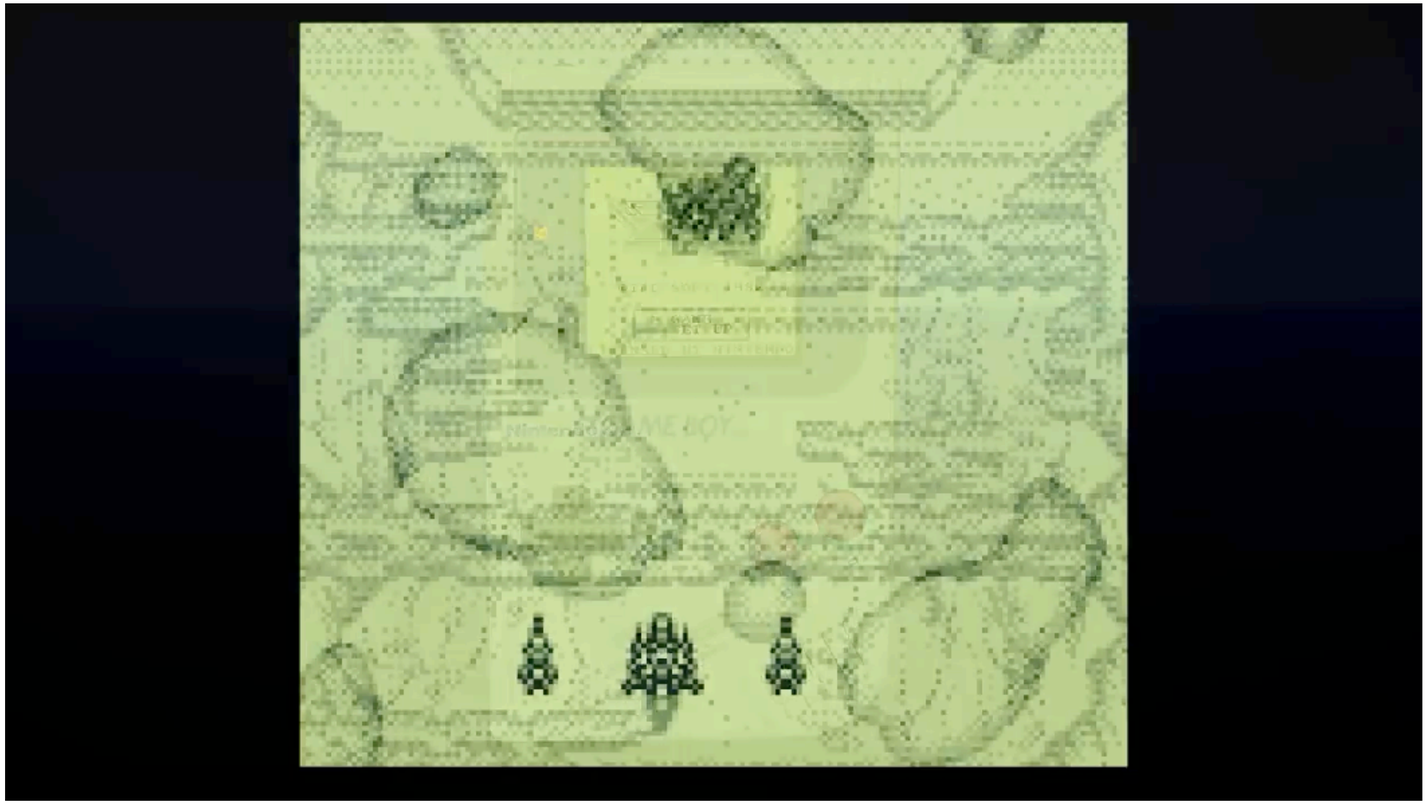
Case study: Retro Gaming / Gameboy

- Context:
  - No 3D graphics
  - Slow hardware
  - Grey scale (2 bits, 4 shades)
  - 160x144 pixel LCD screen
  - Tile-based rendering (images are constructed from small 8x8 pixel tiles, total 20x18 tiles)
  - Three layers: background, window, and objects (sprites)



Case study: Retro Gaming / Gameboy

- Context:
  - No 3D graphics
  - Slow hardware
  - Grey scale (2 bits, 4 shades)
  - 160x144 pixel LCD screen
  - Tile-based rendering (images are constructed from small 8x8 pixel tiles, total 20x18 tiles)
  - Three layers: background, window, and objects (sprites)



What do we miss?
2D Transformations / Animations

Next time :)

The end!