UNIVERSITY OF CAMBRIDGE

Advanced Graphics and Image Processing

Computer Science Tripos Part 2
MPhil in Advanced Computer Science
Michaelmas Term 2021/2022

Department of
Computer Science
and Technology
The Computer Laboratory

William Gates Building
15 JJ Thomson Avenue
Cambridge
CB3 0FD

www.cst.cam.ac.uk

This handout includes copies of the slides that will be used in lectures and more detailed notes on the selected topics. These notes do not constitute a complete transcript of all the lectures and they are not a substitute for text books. They are intended to give a reasonable synopsis of the subjects discussed, but they give neither complete descriptions nor all the background material.

Advanced Graphics & Image Processing

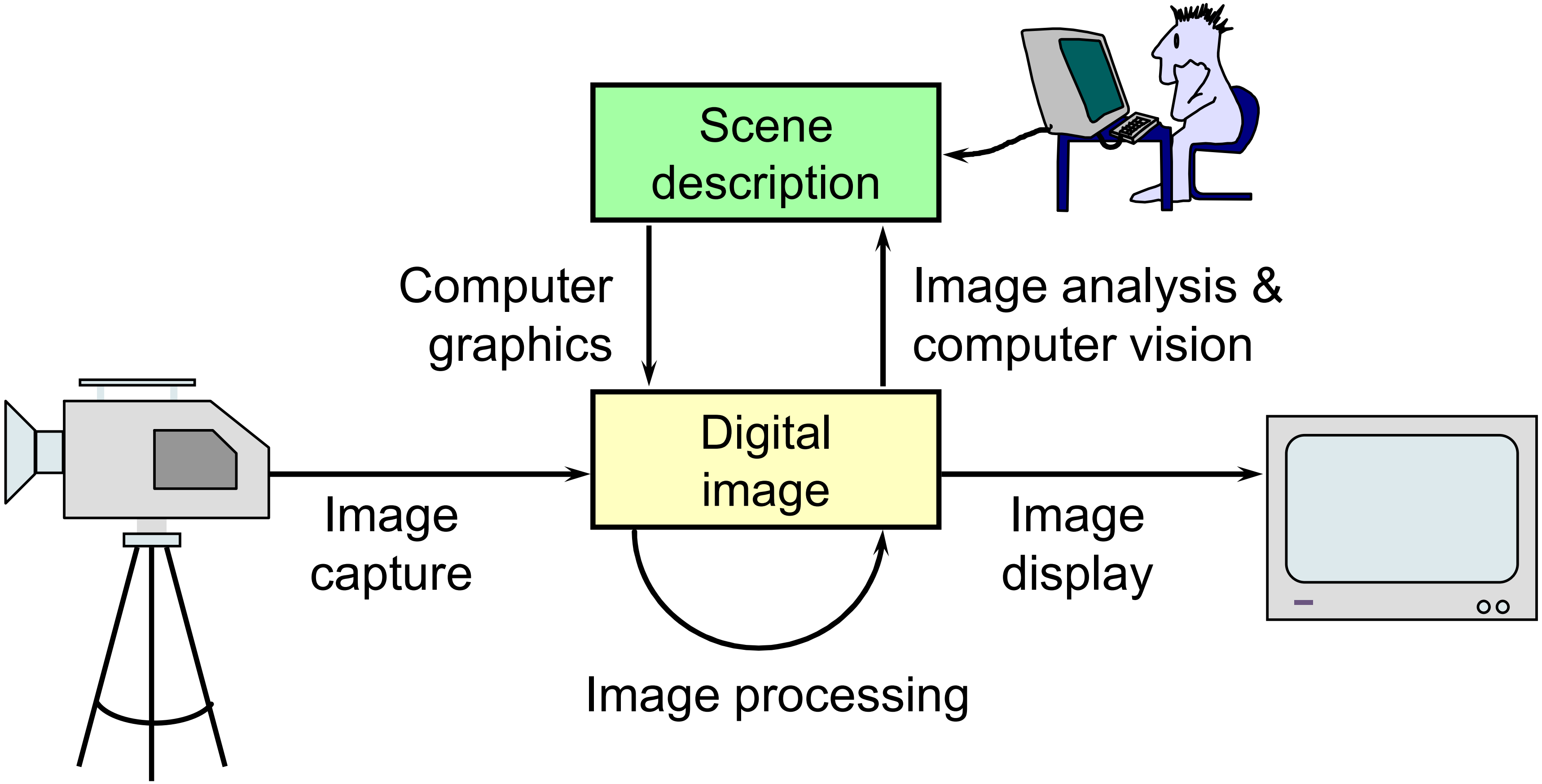
Introduction to Image Processing

Part 1/2 – Images, pixels and sampling

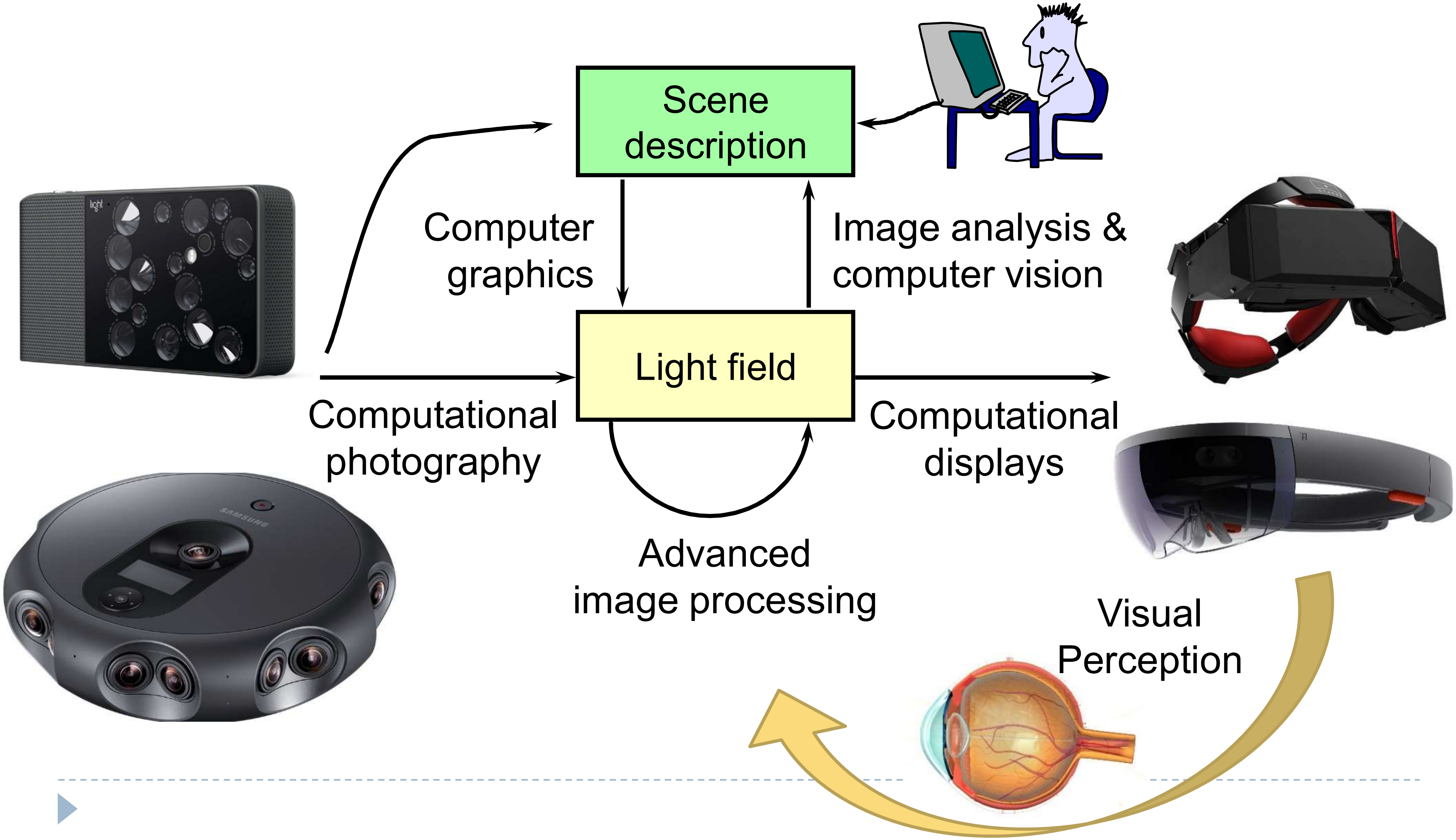
Rafał Mantiuk

Computer Laboratory, University of Cambridge

What are Computer Graphics & Image Processing?



Where are graphics and image processing heading?



What is a (computer) image?

- A digital photograph? ("JPEG")
- A snapshot of real-world lighting?

Image

From computing perspective (discrete)

2D array of pixels

- To represent images in memory
- To create image processing software

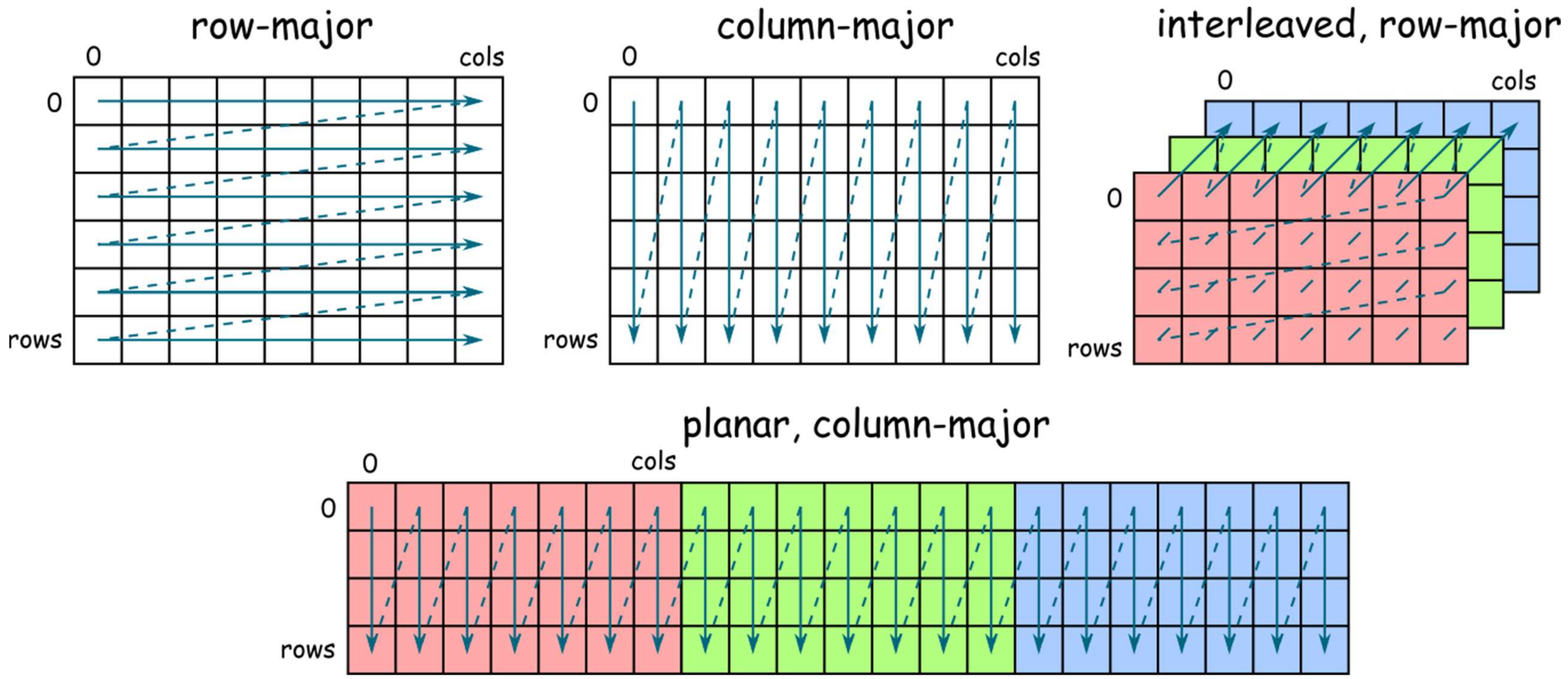
From mathematical perspective (continuous)

2D function

- To express image processing as a mathematical problem
- To develop (and understand) algorithms

Image

- 2D array of pixels
- In most cases, each pixel takes 3 bytes: one for each red, green and blue
- But how to store a 2D array in memory?



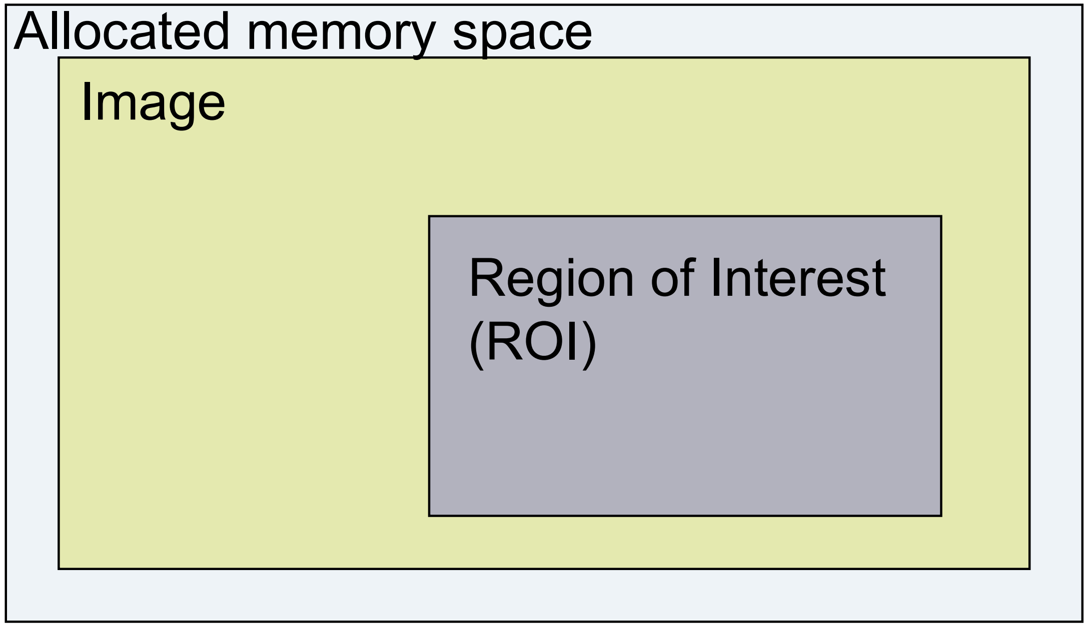
Stride

- Calculating the pixel component index in memory
  - For row-major order (grayscale)
    $$i(x, y) = x + y \cdot n_{cols}$$
  - For column-major order (grayscale)
    $$i(x, y) = x \cdot n_{rows} + y$$
  - For interleaved row-major (colour)
    $$i(x, y, c) = x \cdot 3 + y \cdot 3 \cdot n_{cols} + c$$
  - General case
    $$i(x, y, c) = x \cdot s_x + y \cdot s_y + c \cdot s_c$$

  where s_x, s_y and s_c are the strides for the x, y and colour dimensions

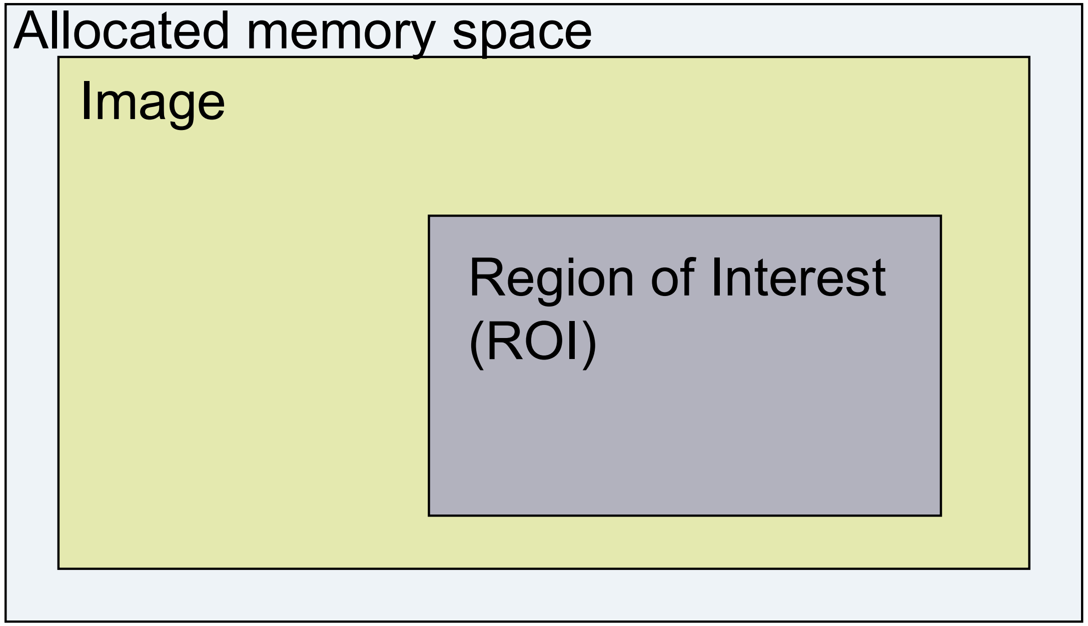
Padded images and stride

- Sometimes it is desirable to "pad" image with extra pixels
  - for example when using operators that need to access pixels outside the image border
- Or to define a region of interest (ROI)



- How to address pixels for such an image and the ROI?

Padded images and stride



$$i(x, y, c) = i_{first} + x \cdot s_x + y \cdot s_y + c \cdot s_c$$

- For row-major, interleaved
  - $s_x = ?$
  - $s_y = ?$
  - $s_c = ?$

Pixel (PIcture ELement)

- Each pixel (usually) consist of three values describing the color

(red, green, blue)

- For example
  - (255, 255, 255) for white
  - (0, 0, 0) for black
  - (255, 0, 0) for red
- Why are the values in the 0-255 range?
- Why red, green and blue? (and not cyan, magenta, yellow)
- How many bytes are needed to store 5MPixel image? (uncompressed)

Pixel formats, bits per pixel, bit-depth

- Grayscale – single **color channel**, 8 bits (1 byte)
- Highcolor – 2^{16}=65,536 colors (2 bytes)

Sample Length:	5	6	5
Channel Membership:	Red	Green	Blue
Bit Number:	15 14 13 12 11	10 9 8 7 6 5	4 3 2 1 0
RGBAX	R. G. B. A. X		
Sample Length Notation:	**5.6.5.0.0**		

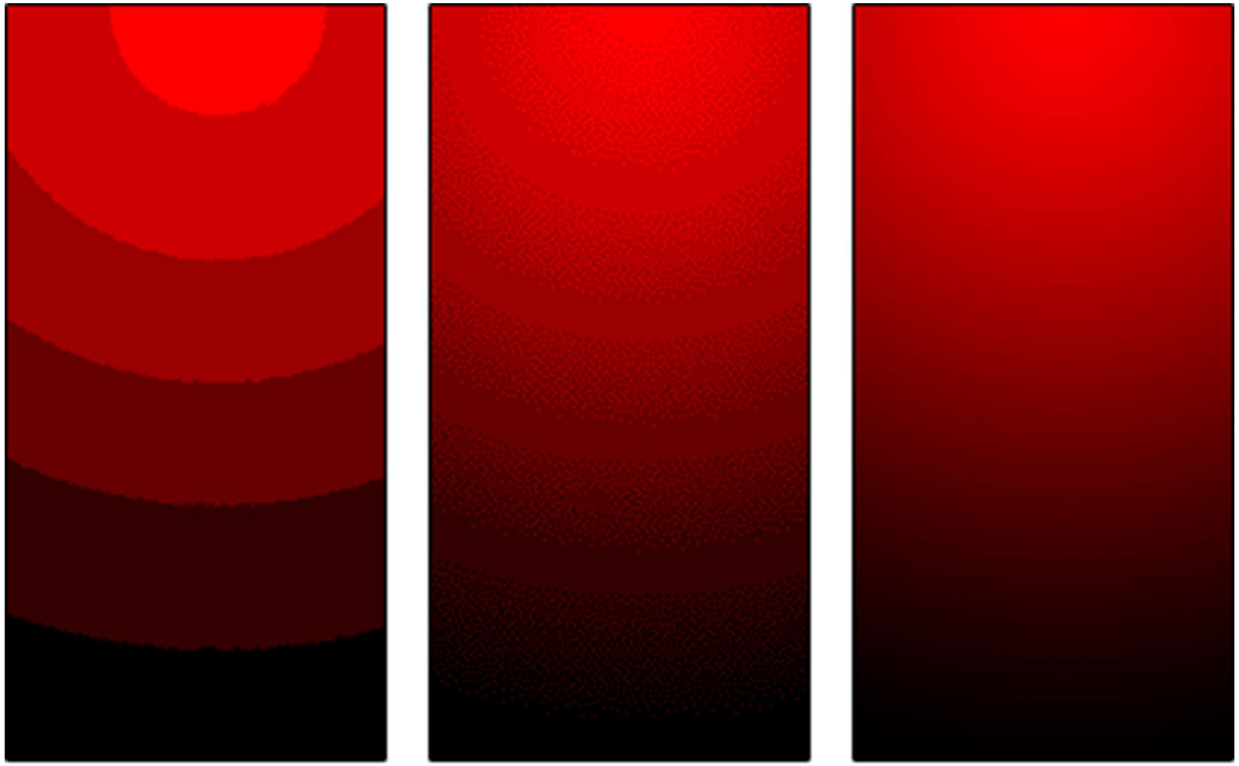
- Truecolor – 2^{24} = 16,8 million colors (3 bytes)
- Deepcolor – even more colors (>= 4 bytes)

Sample Length:	2	10	10	10
Channel Membership:	None	Red	Green	Blue
Bit Number:	31 30	29 28 27 26 25 24 23 22 21 20	19 18 17 16 15 14 13 12 11 10	9 8 7 6 5 4 3 2 1 0
RGBAX	R . G . B . A . X			
Sample Length Notation:	**10.10.10.0.2**			

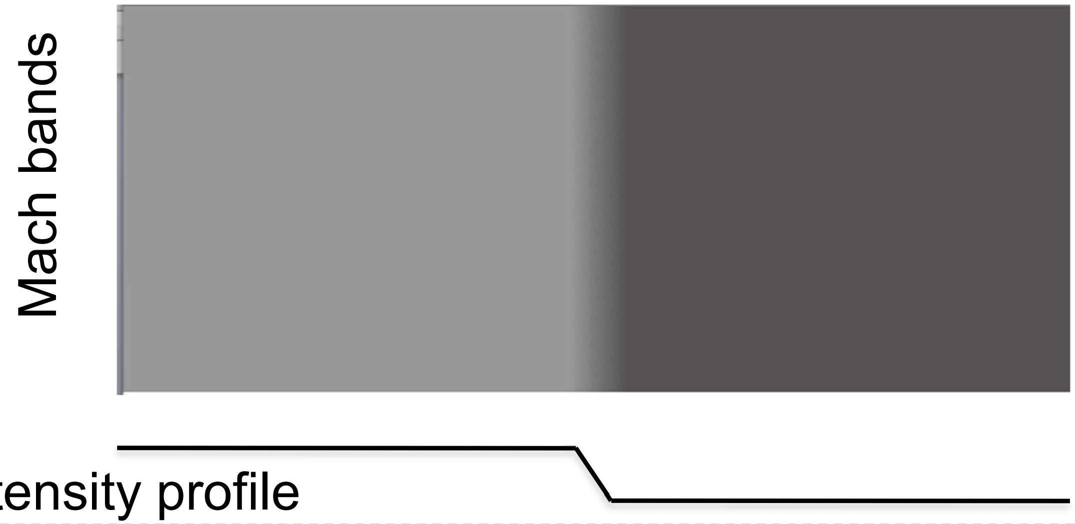
- But why?

Color banding

- If there are not enough bits to represent color
- Looks worse because of the **Mach band** illusion
- Dithering (added noise) can reduce banding
  - Printers
  - Many LCD displays do it too



8-bit gradient | 8-bit gradient, dithered | 24-bit gradient



What is a (computer) image?

- A digital photograph? ("JPEG")
- A snapshot of real-world lighting?

Image

From computing perspective (discrete)

2D array of pixels

•To represent images in memory

•To create image processing software

From mathematical perspective (continuous)

2D function

•To express image processing as a mathematical problem

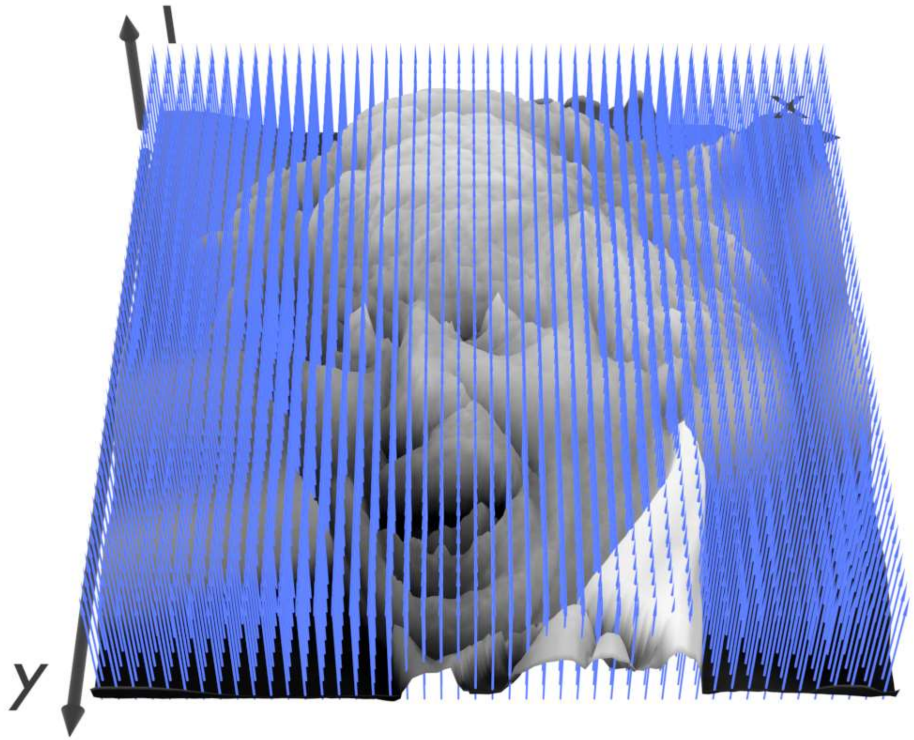
•To develop (and understand) algorithms

Image – 2D function

- Image can be seen as a function I(x,y), that gives intensity value for any given coordinate (x,y)

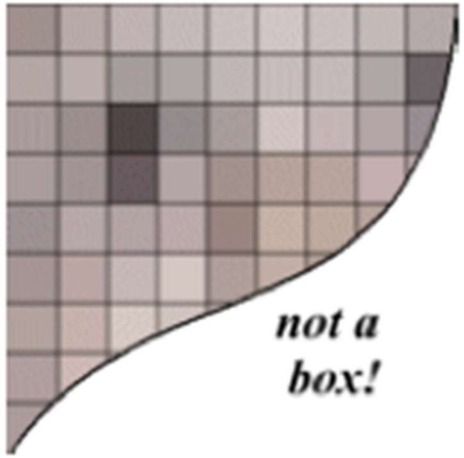
Sampling an image

- The image can be sampled on a rectangular sampling grid to yield a set of samples. These samples are pixels.

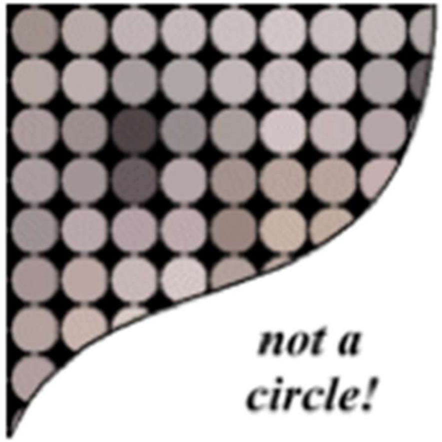


What is a pixel? (math)

- A pixel is not
  - a box
  - a disk
  - a teeny light



- A pixel is a point
  - it has no dimension
  - it occupies no area
  - it cannot be seen
  - it has coordinates

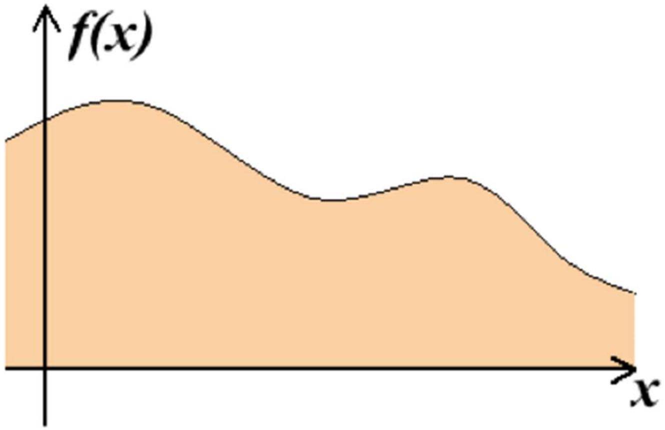


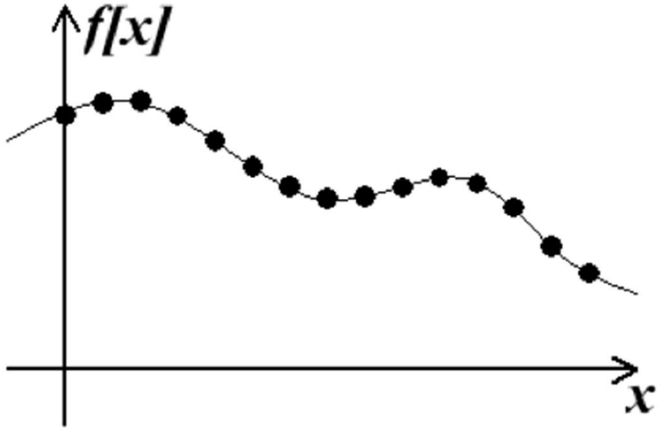
- A pixel is a **sample**

From: http://groups.csail.mit.edu/graphics/classes/6.837/F01/Lecture05/lecture05.pdf

Sampling and quantization

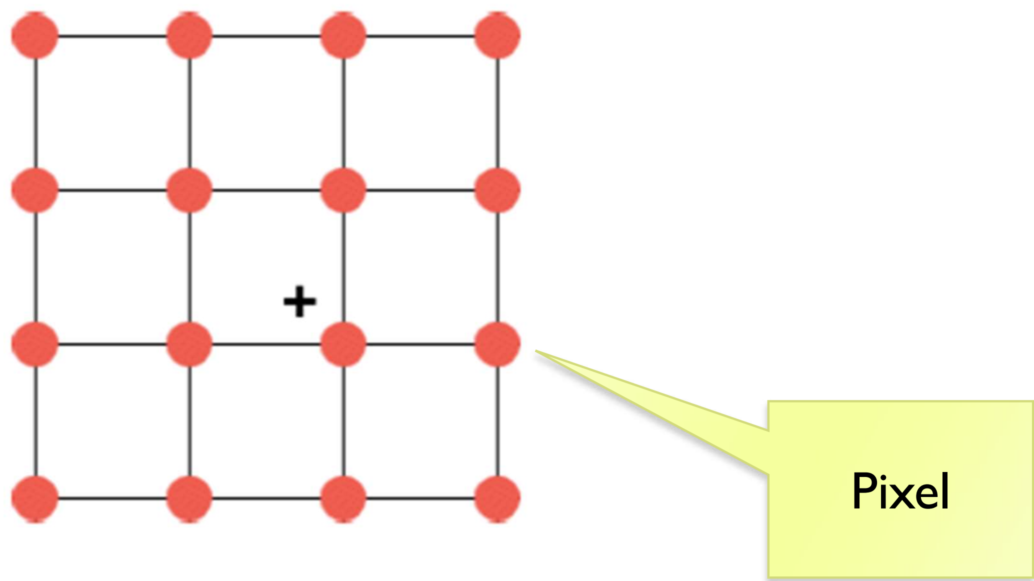
- Physical world is described in terms of continuous quantities
- But computers work only with discrete numbers
- Sampling – process of mapping continuous function to a discrete one
- Quantization – process of mapping continuous variable to a discrete one





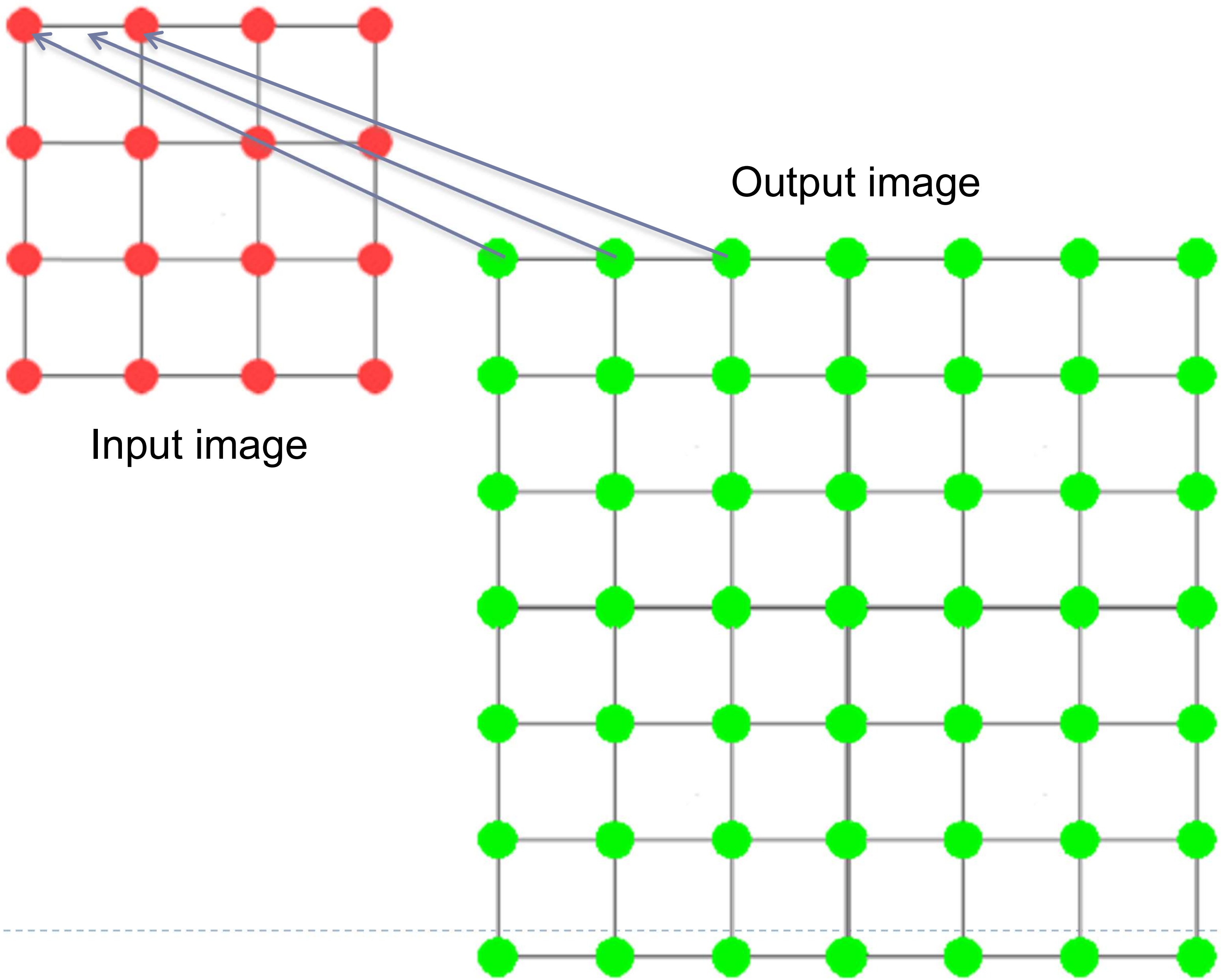
Resampling

- Some image processing operations require to know the colors that are in-between the original pixels

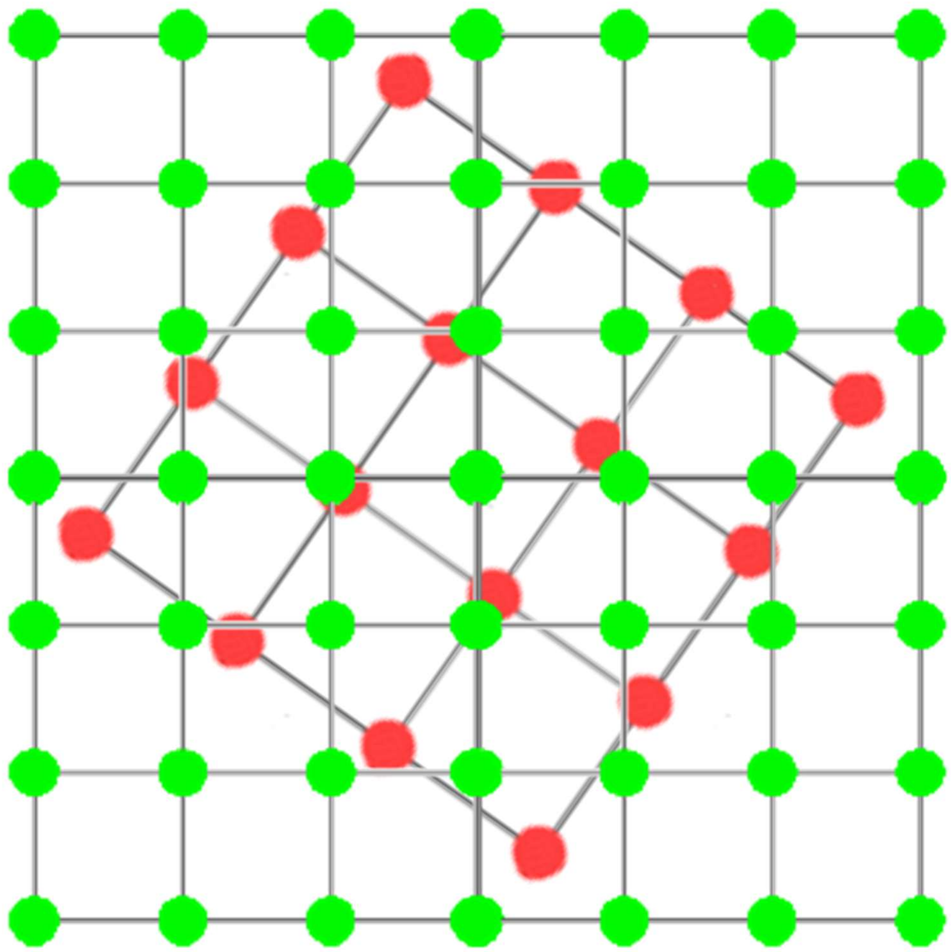


- What are those operations?
- How to find these *resampled* pixel values?

Example of resampling: magnification

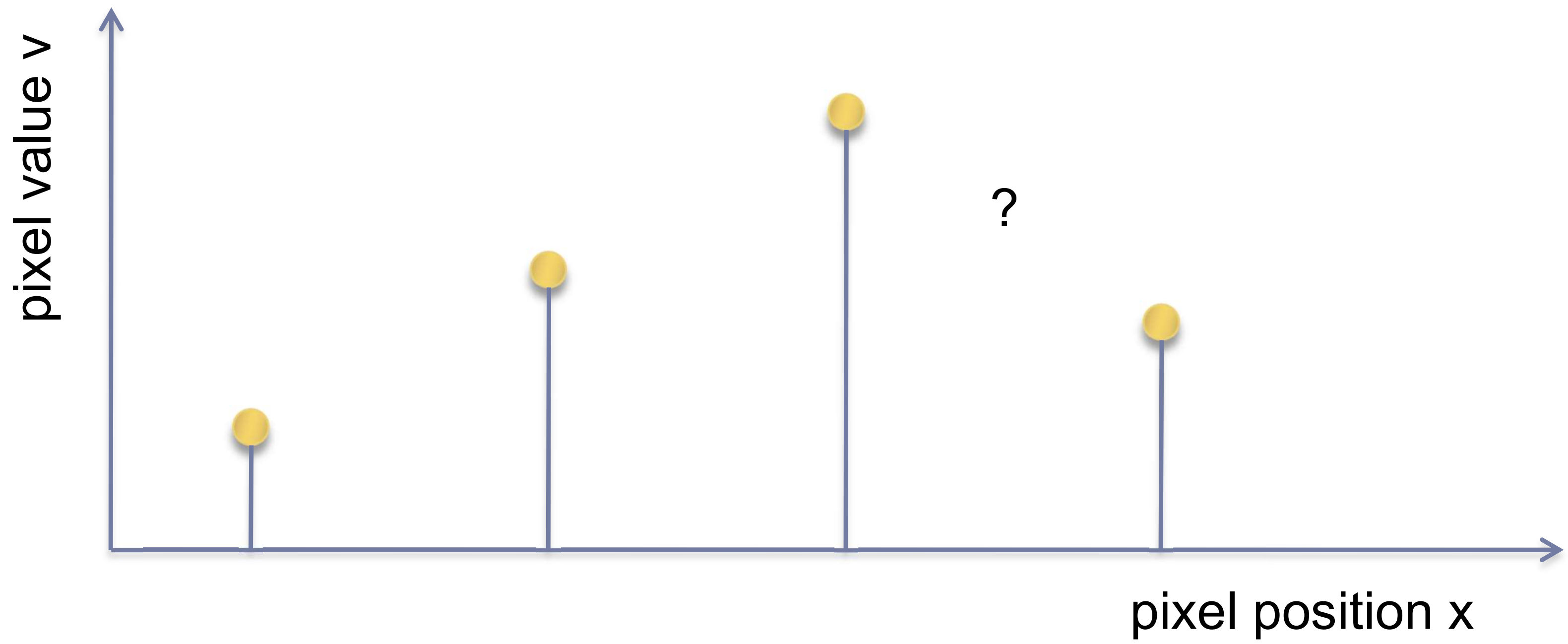


Example of resampling: scaling and rotation



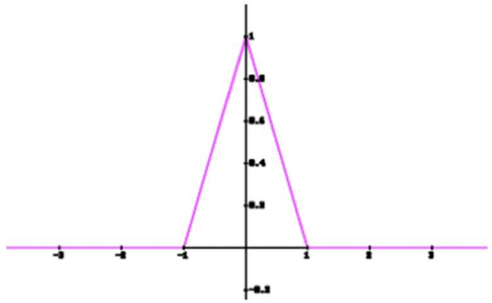
How to resample?

- In 1D: how to find the most likely resampled pixel value knowing its two neighbors?

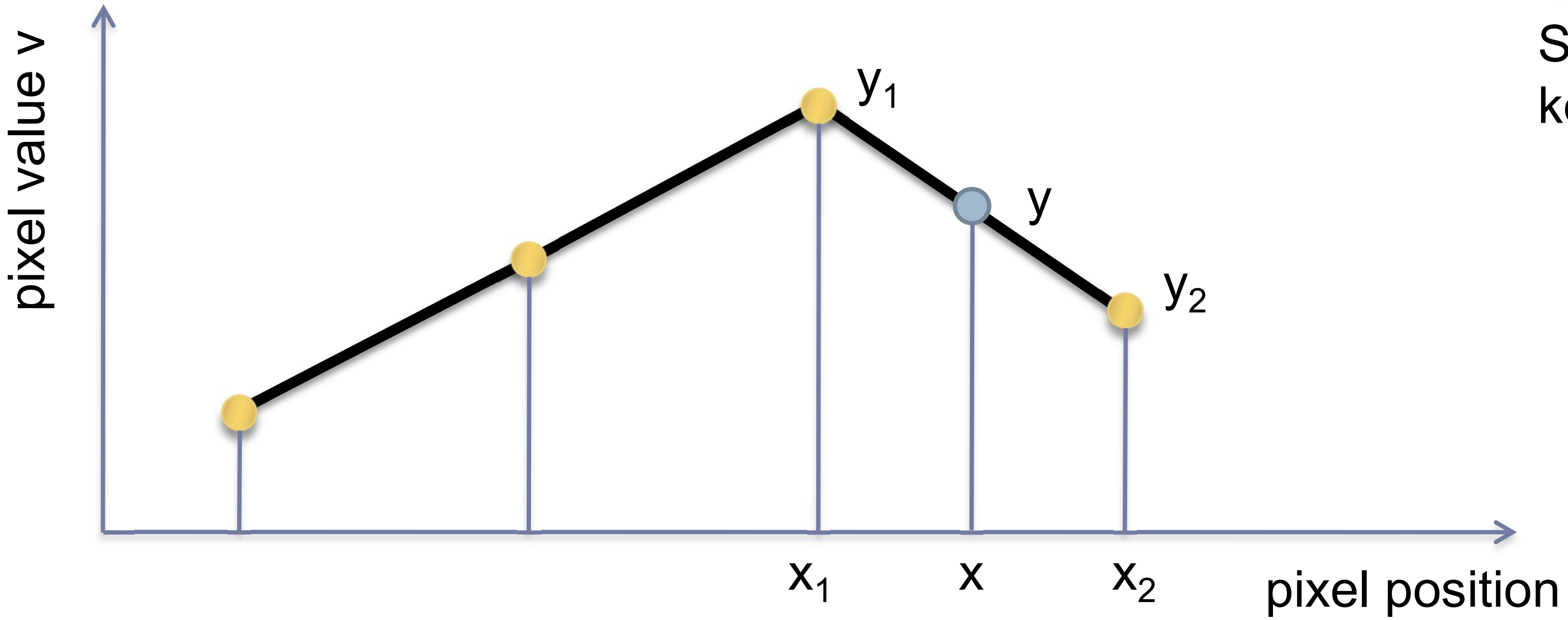


(Bi)Linear interpolation (resampling)

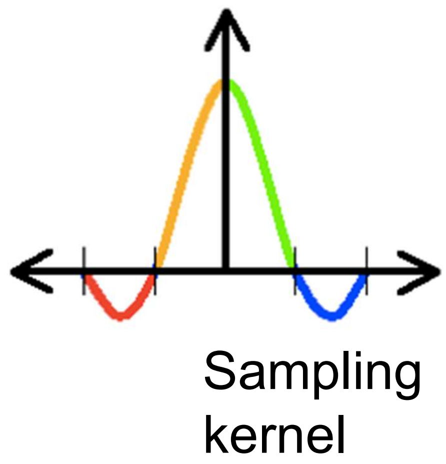
- Linear – 1D
- Bilinear – 2D

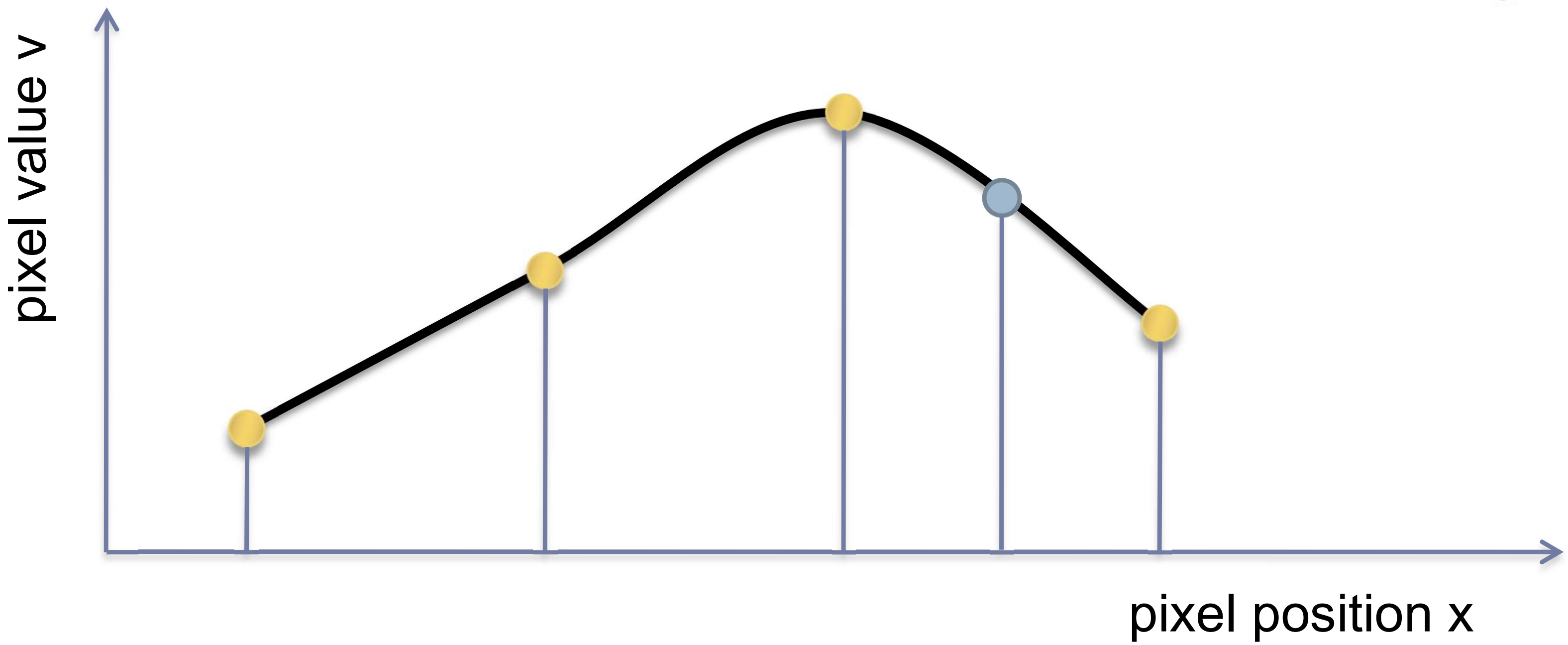


Sampling kernel

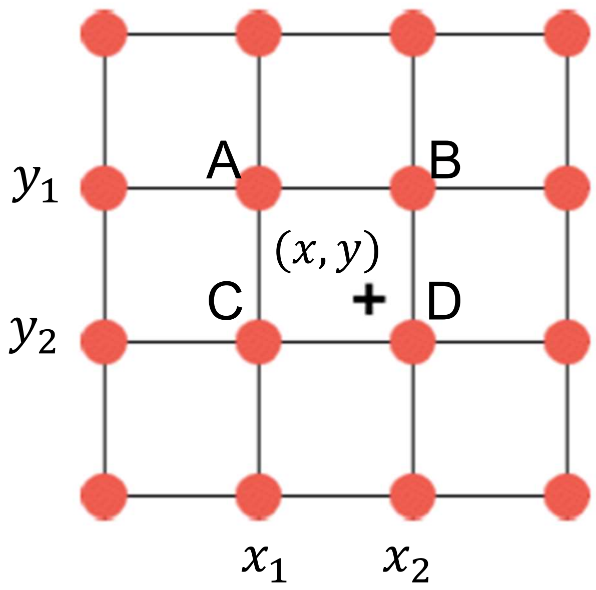


(Bi)cubic interpolation (resampling)





Bi-linear interpolation



Given the pixel values:

$$I(x_1, y_1) = A$$

$$I(x_2, y_1) = B$$

$$I(x_1, y_2) = C$$

$$I(x_2, y_2) = D$$

Calculate the value of a pixel $I(x, y) = ?$ using bi-linear interpolation.

Hint: Interpolate first between A and B, and between C and D, then interpolate between these two computed values.

Advanced Graphics & Image Processing

Introduction to Image Processing

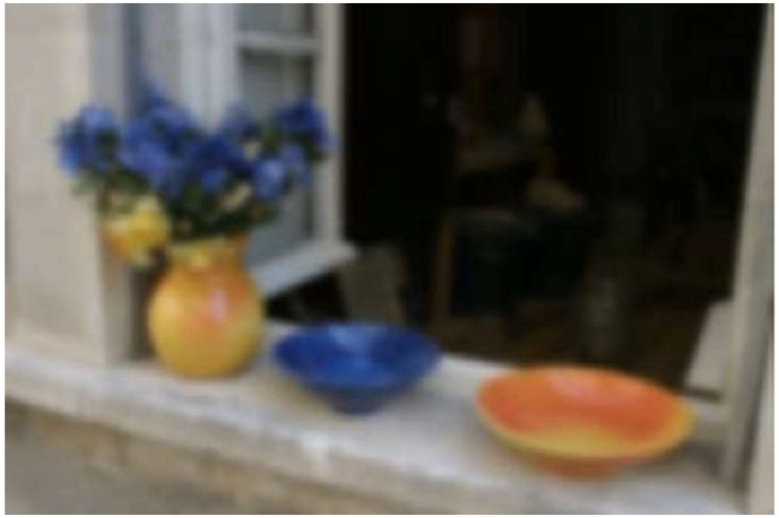
Part 2/2 – Point ops, filters and pyramids

Rafał Mantiuk

Computer Laboratory, University of Cambridge

Point operators and filters

Original



Blurred

Sharpenned

Edge-preserving filter

Point operators

- Modify each pixel independent from one another
- The simplest case: multiplication and addition

Pixel position
x=(x,y)

Input pixel
value

$$g(\boldsymbol{x}) = af(\boldsymbol{x}) + b$$

Resulting pixel
value

Gain

Bias

Pixel precision for image processing

- Given an RGB image, 8-bit per color channel (uchar)
  - What happens if the value of 10 is subtracted from the pixel value of 5 ?
  - 250 + 10 = ?
  - How to multiply pixel values by 1.5 ?
    - a) Using floating point numbers
    - b) While avoiding floating point numbers

Image blending

- Cross-dissolve between two images

Pixel from image 1

Pixel from image 2

$$g(\boldsymbol{x}) = (1-\alpha) f_0(\boldsymbol{x}) + \alpha f_1(\boldsymbol{x})$$

Resulting pixel value

Blending parameter

- where α is between 0 and 1

Image matting and compositing

- Matting – the process of extracting an object from the original image
- Compositing – the process of inserting the object into a different image
- It is convenient to represent the extracted object as an RGBA image

Transparency, alpha channel

- RGBA – red, green, blue, alpha
  - alpha = 0 – transparent pixel
  - alpha = 1 – opaque pixel
- Compositing
  - Final pixel value:

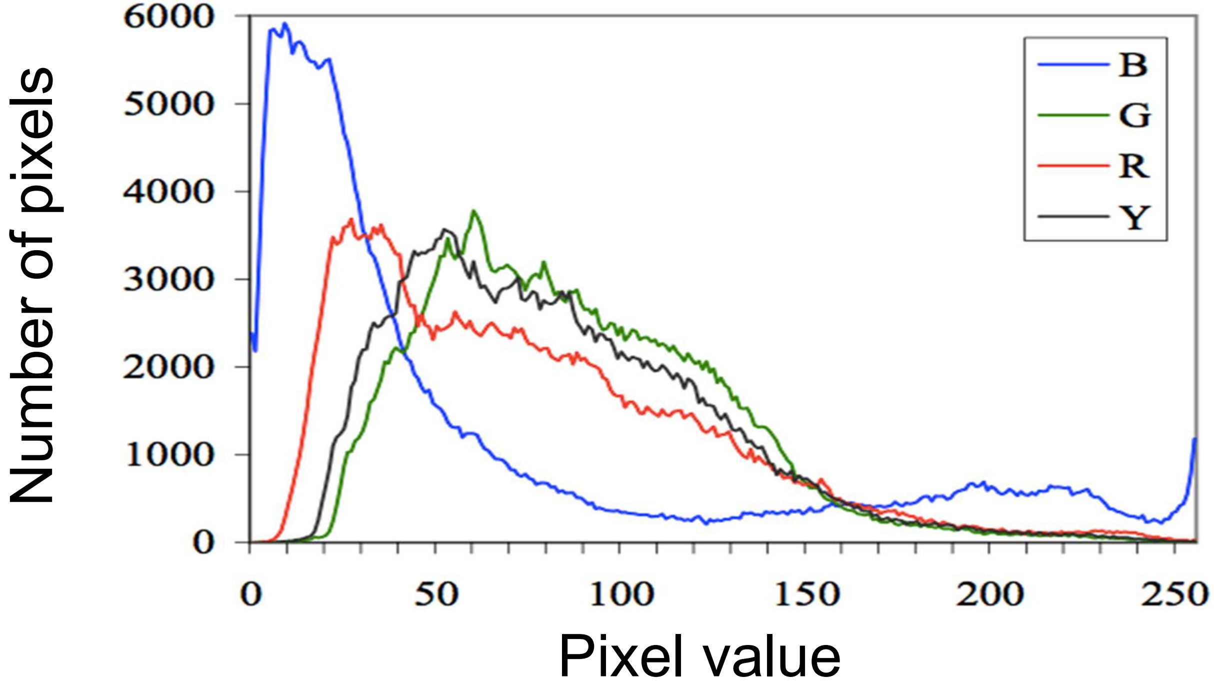
$$P = \alpha C_{pixel} + (1 - \alpha) C_{background}$$

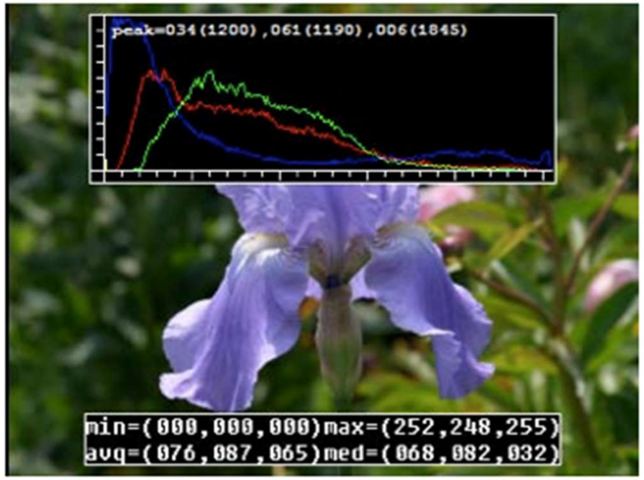
  - Multiple layers:

$$P_0 = C_{background}$$

$$P_i = \alpha_i C_i + (1 - \alpha_i) P_{i-1} \quad i = 1..N$$

Image histogram

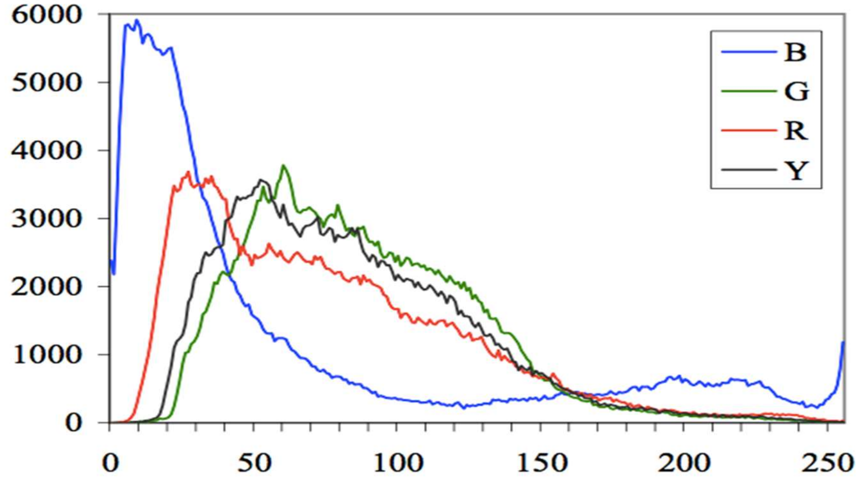




- histogram / total pixels = probability mass function
  - what probability does it represent?

Histogram equalization

- Pixels are non-uniformly distributed across the range of values



- Would the image look better if we uniformly distribute pixel values (make the histogram more uniform)?
- How can this be done?

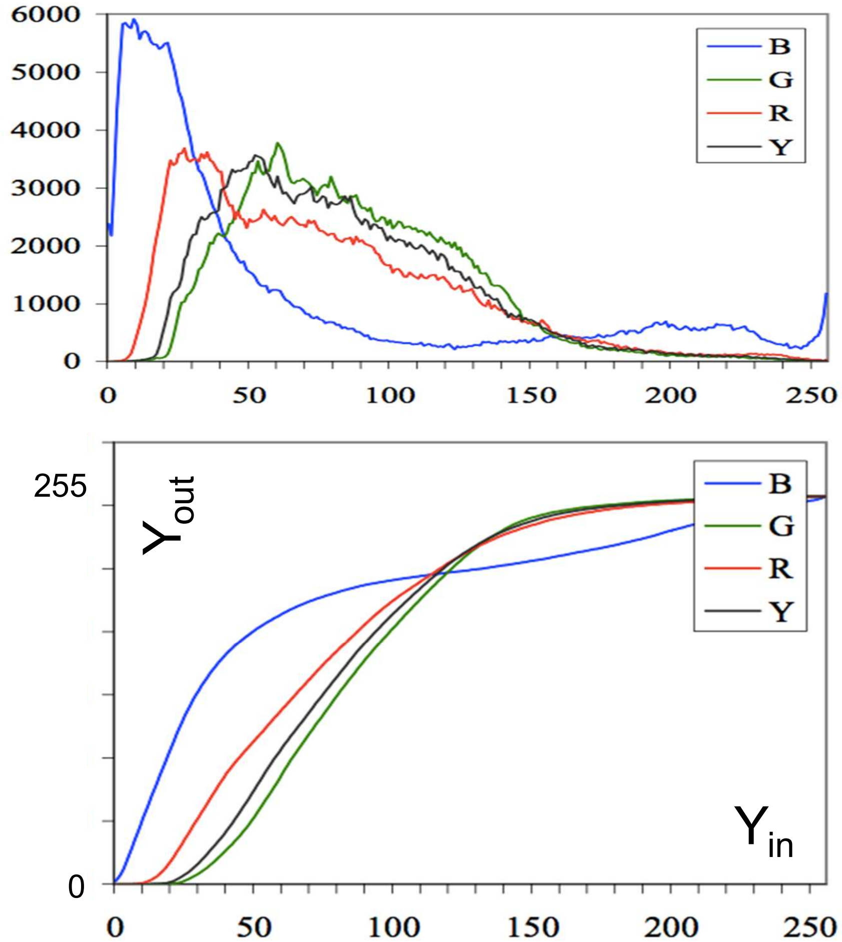
Histogram equalization

- Step 1: Compute image histogram
- Step 2: Compute a normalized cumulative histogram

$$c(I) = \frac{1}{N} \sum_{i=0}^{I} h(i)$$

- Step 3: Use the cumulative histogram to map pixels to the new values (as a look-up table)

$$Y_{out} = c(Y_{in})$$



Linear filtering (revision)

- Output pixel value is a weighted sum of neighboring pixels

Input pixel value

Kernel (filter)

$$g(i,j) = \sum_{k,l} f(i-k, j-l)h(k,l)$$

Resulting pixel value

Sum over neighboring pixels, e.g. k=-1,0,1, j=-1,0,1 for 3x3 neighborhood

compact notation $g = f * h$

Convolution operation

Linear filter: example

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

$f(x,y)$

*

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

$h(x,y)$

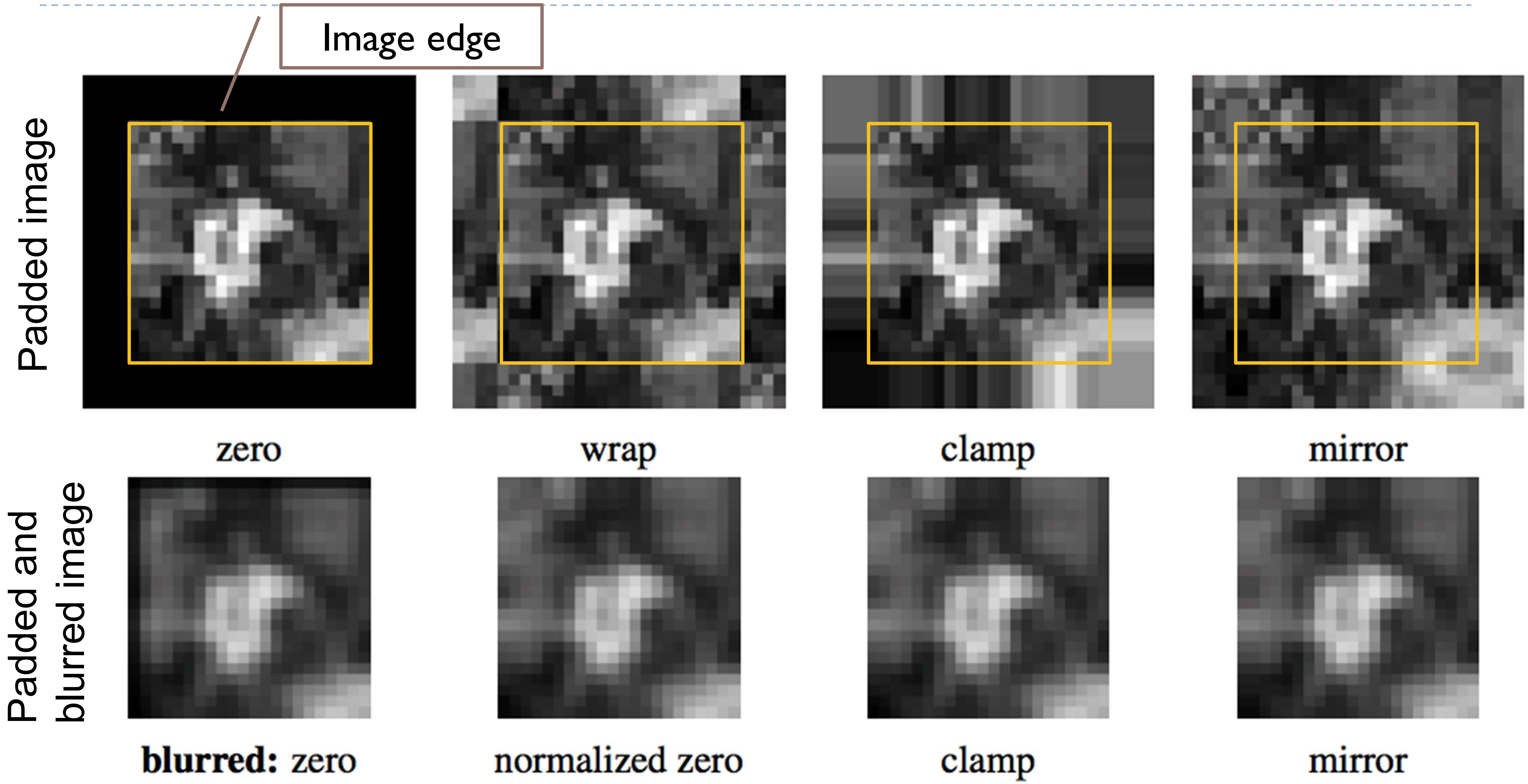
=

69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

$g(x,y)$

Why is the matrix g smaller than f ?

Padding an image



What is the computational cost of the convolution?

$$g(i,j) = \sum_{k,l} f(i-k, j-l)h(k,l)$$

- How many multiplications do we need to do to convolve 100x100 image with 9x9 kernel ?
  - The image is padded, but we do not compute the values for the padded pixels

Separable kernels

- Convolution operation can be made much faster if split into two separate steps:
  - 1) convolve all rows in the image with a 1D filter
  - 2) convolve columns in the result of 1) with another 1D filter
- But to do this, the kernel must be separable

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \cdot \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}$$

$$\vec{h} = \vec{u} \cdot \vec{v}$$

Examples of separable filters

- Box filter:

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$$

- Gaussian filter:

$$G(x,y;\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

  - What are the corresponding 1D components of this separable filter (u(x) and v(y))?

$$G(x,y) = u(x) \cdot v(y)$$

Unsharp masking

- How to use blurring to sharpen an image ?

results | original image | high-pass image | blurry image

$$g_{\mathrm{sharp}} = f + \gamma(f - h_{\mathrm{blur}} * f)$$

Why "linear" filters ?

- Linear functions have two properties:
  - Additivity: $f(x) + f(y) = f(x + y)$
  - Homogenity: $f(ax) = af(x)$ (where "f" is a linear function)
- Why is it important?
  - Linear operations can be performed in an arbitrary order
    $$blur(aF + b) = a\, blur(F) + b$$
  - Linearity of the Gaussian filter could be used to improve the performance of your image processing operation
  - This is also how the separable filters work:

Matrix multiplication

Convolution

$$(u \cdot v) * f = u * (v * f)$$

The components of a separable kernel

An image

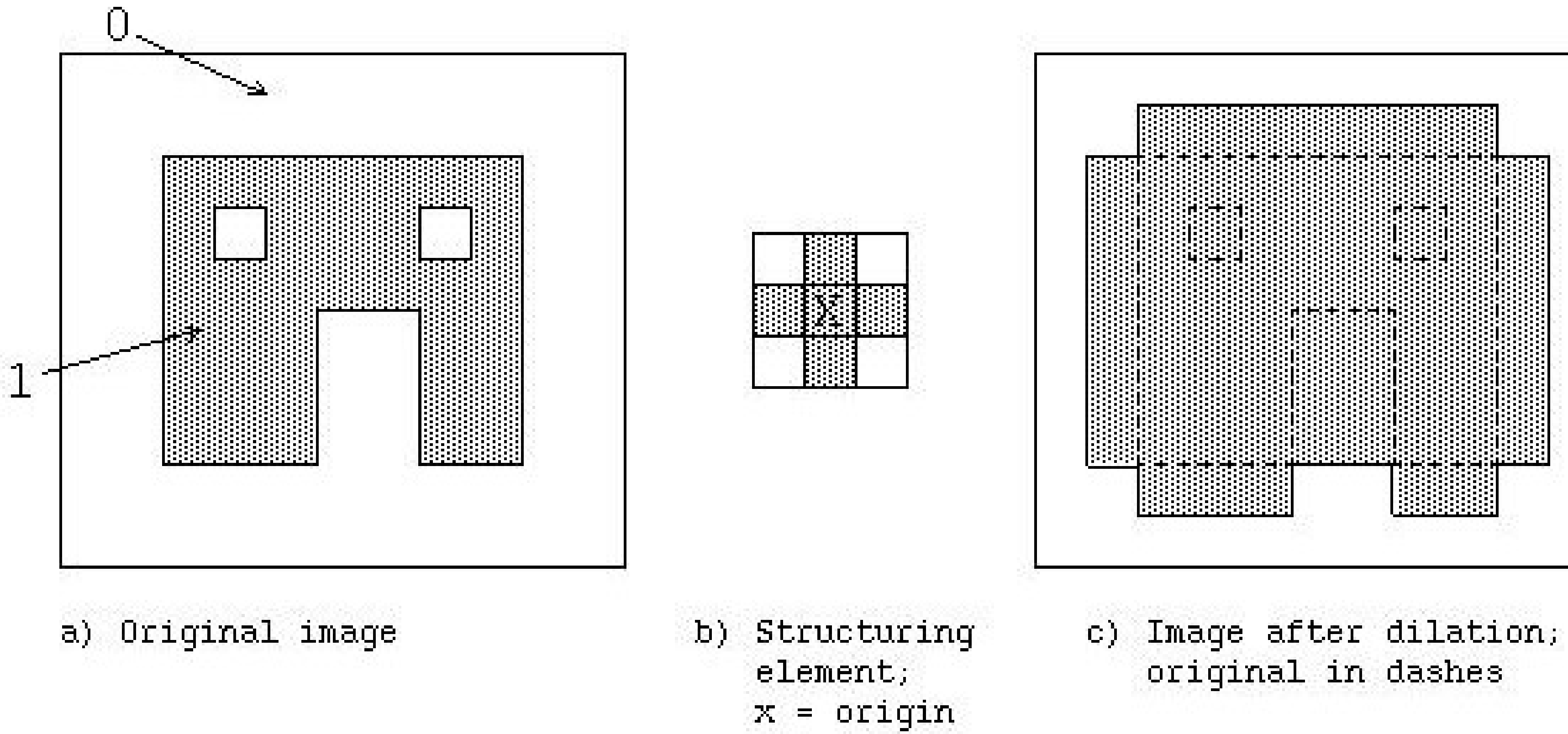
Operations on binary images

- Essential for many computer vision tasks

- Binary image can be constructed by thresholding a grayscale image

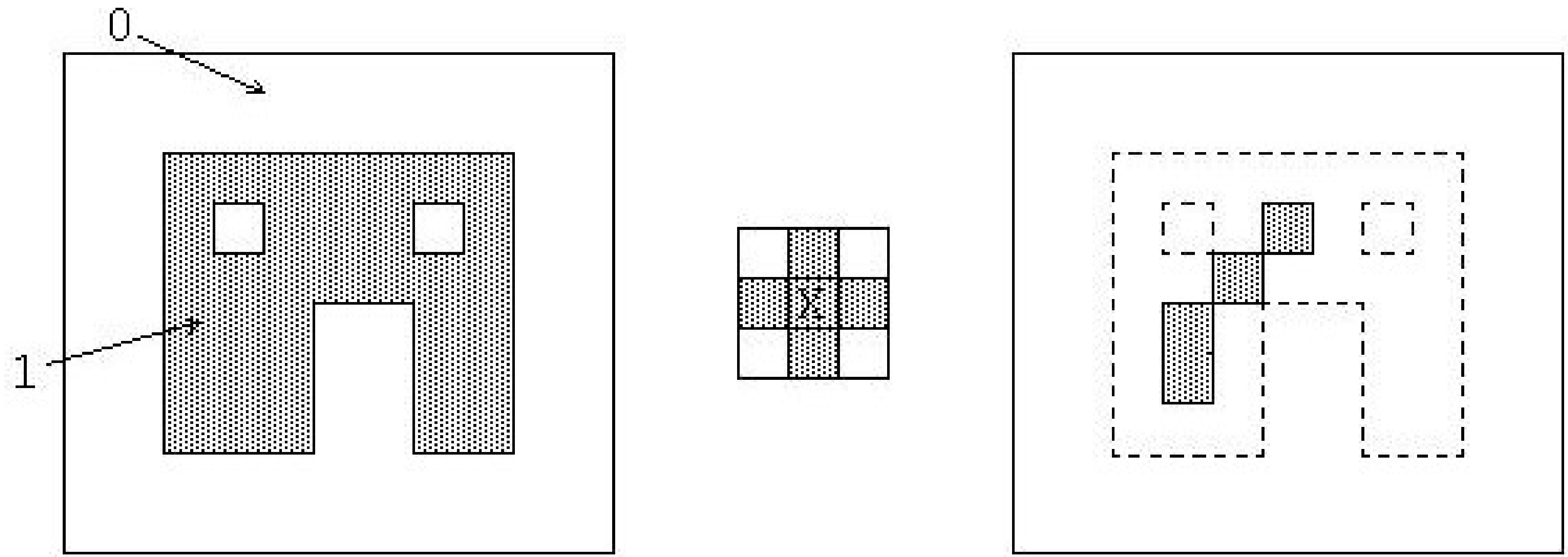
$$\theta(f, c) = \begin{cases} 1 & \text{if } f \geq c, \\ 0 & \text{else,} \end{cases}$$

Morphological filters: dilation



- Set the pixel to the maximum value of the neighboring pixels within the structuring element
- What could it be useful for ?

Morphological filters: erosion



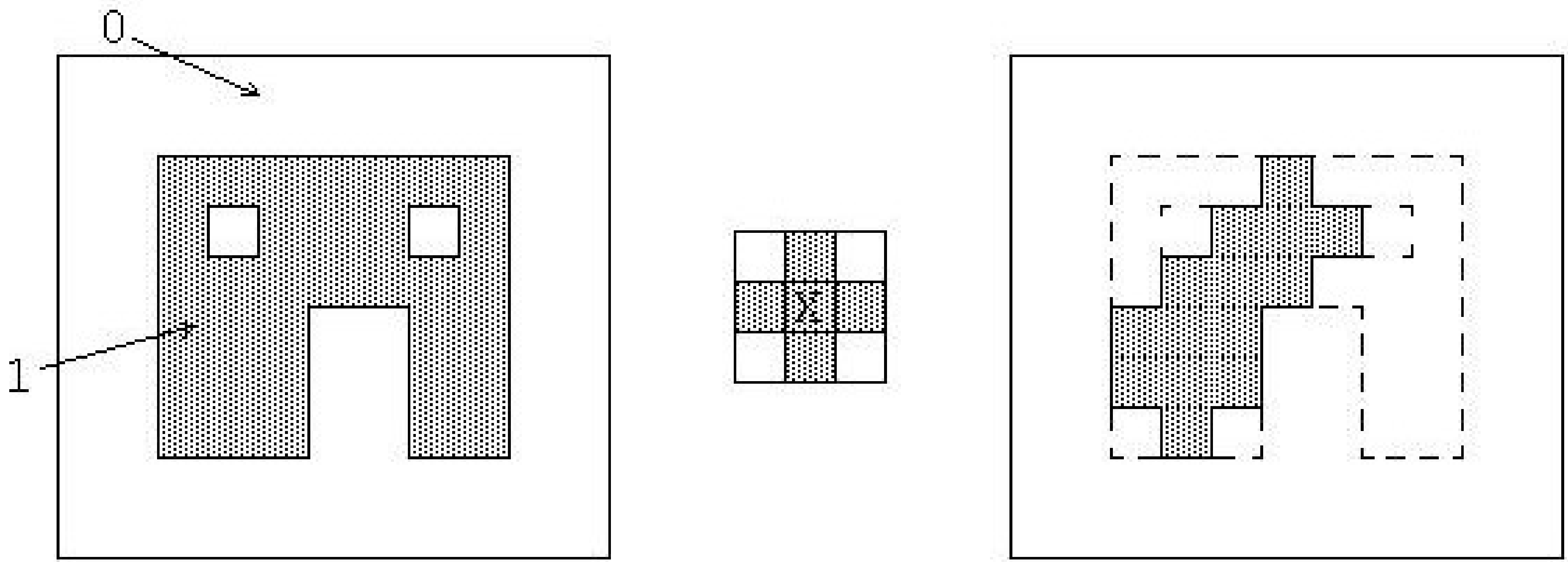
a) Original image

b) Structuring element; x = origin

c) Image after erosion; original in dashes

- Set the value to the minimum value of all the neighboring pixels within the structuring element
- What could it be useful for ?

Morphological filters: opening



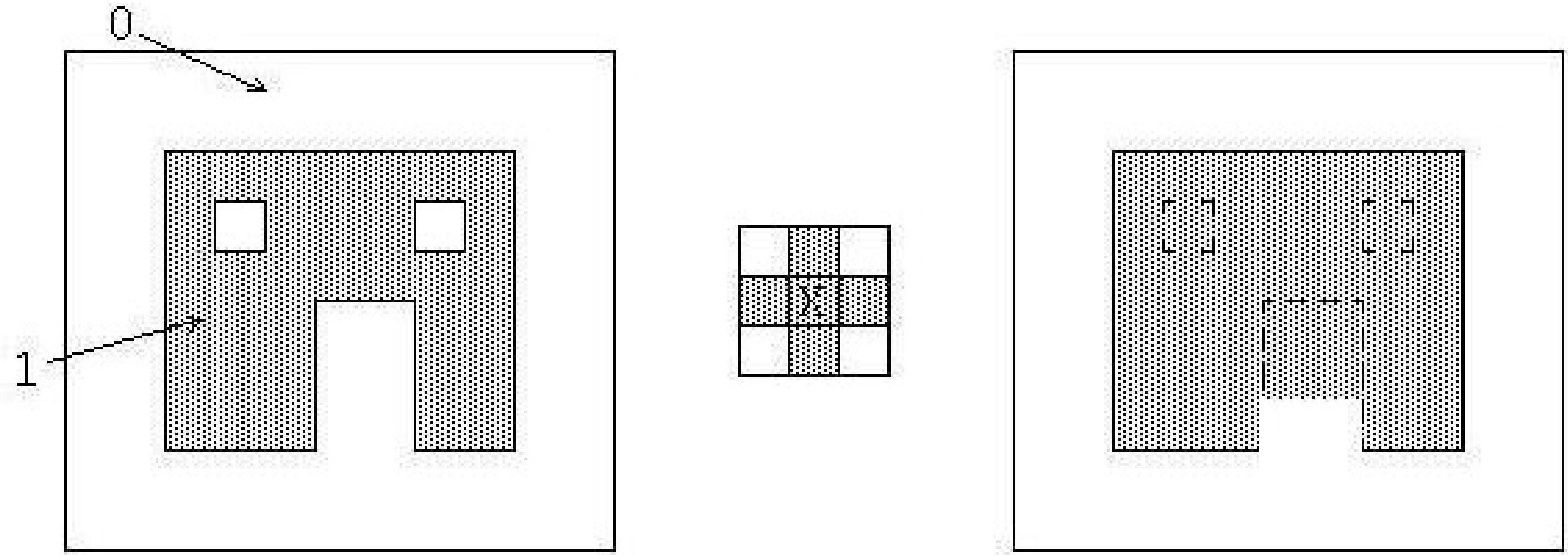
a) Original image

b) Structuring element; x = origin

c) Image after opening = erosion followed by dilation

- Erosion followed by dilation
- What could it be useful for?

Morphological filters: closing



a) Original image

b) Structuring element; x = origin

c) Image after closing = dilation followed by erosion; original in dashes.

- Dilation followed by erosion
- What could it be useful for ?

Binary morphological filters: formal definition

Number of 1s inside the region restricted by the structuring element

Binary image

Correlation (similar to convolution)

Structuring element

$$c = f \otimes s$$

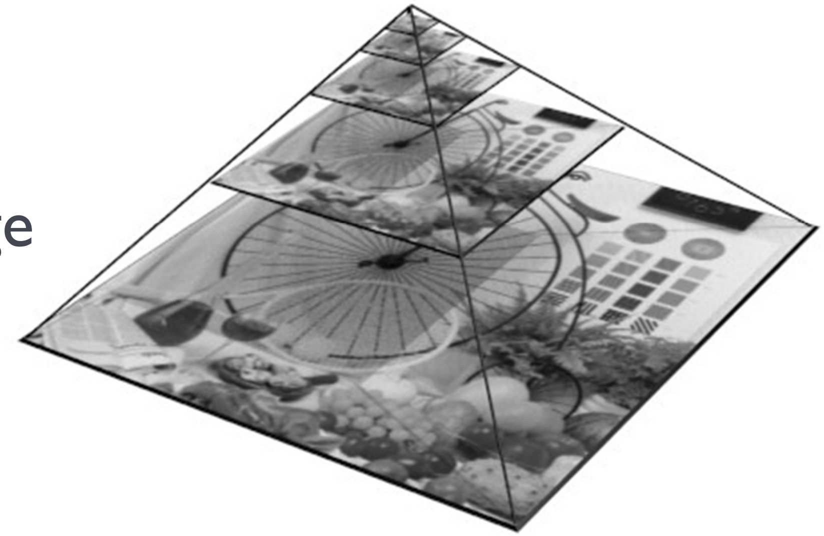
S – size of structuring element (number of 1s in the SI)

- **dilation**: $\text{dilate}(f, s) = \theta(c, 1)$;
- **erosion**: $\text{erode}(f, s) = \theta(c, S)$;
- **majority**: $\text{maj}(f, s) = \theta(c, S/2)$;
- **opening**: $\text{open}(f, s) = \text{dilate}(\text{erode}(f, s), s)$;
- **closing**: $\text{close}(f, s) = \text{erode}(\text{dilate}(f, s), s)$.

$$\theta(f, c) = \begin{cases} 1 & \text{if } f \geq c, \\ 0 & \text{else,} \end{cases}$$

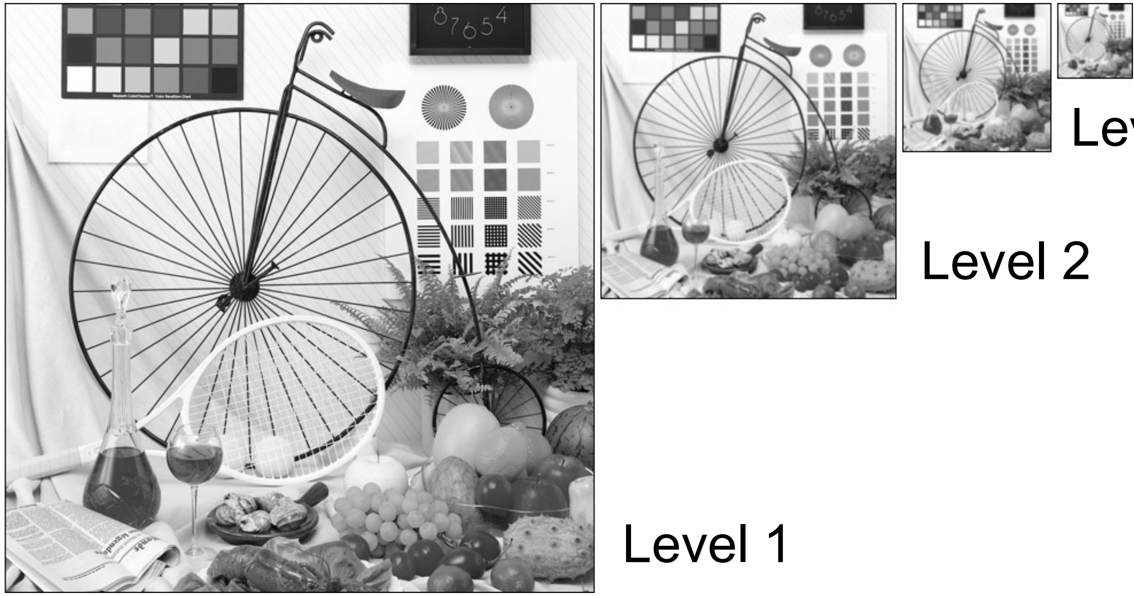
Multi-scale image processing (pyramids)

- Multi-scale processing operates on an image represented at several sizes (scales)
  - Fine level for operating on small details
  - Coarse level for operating on large features
- Example:
  - Motion estimation
    - Use fine scales for objects moving slowly
    - Use coarse scale for objects moving fast
  - Blending (to avoid sharp boundaries)



Two types of pyramids

Gaussian pyramid



Level 4

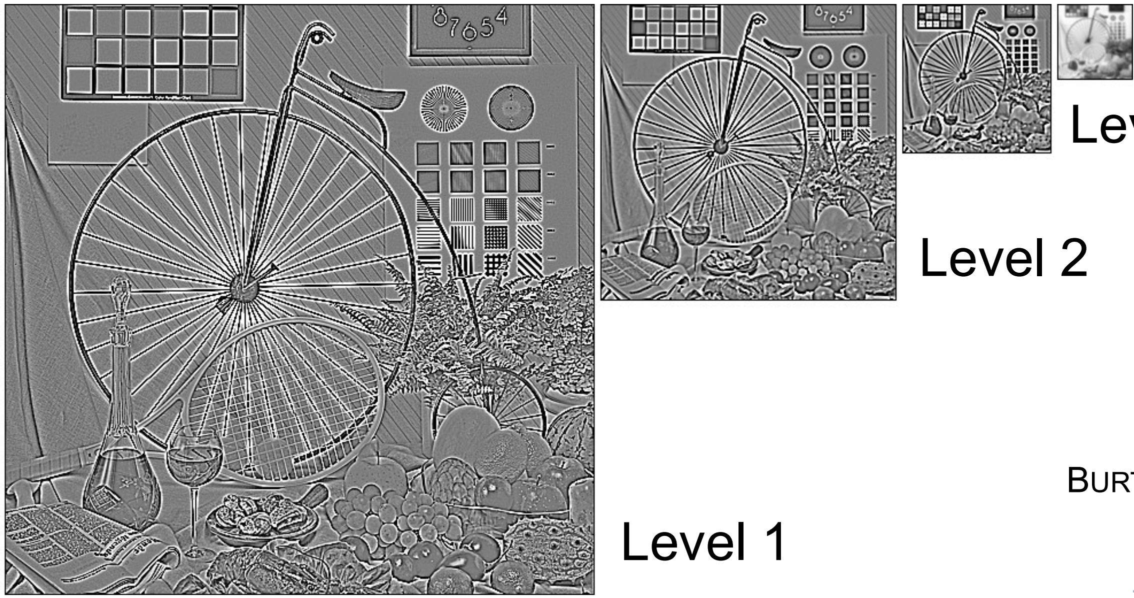
Level 3

Level 2

Level 1

Laplacian pyramid

(a.k.a DoG Diffence of Gaussians)



Level 4 (base band)

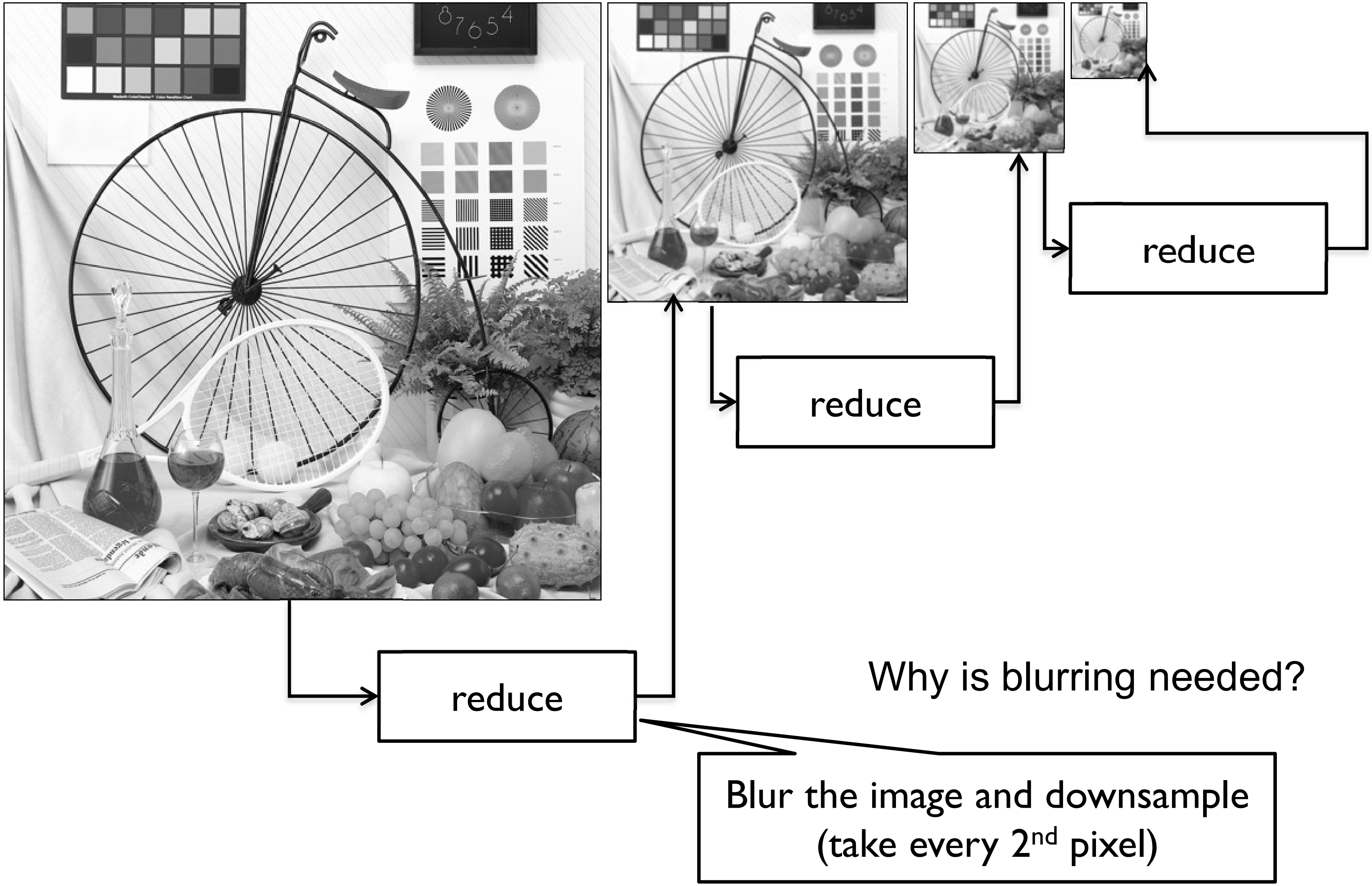
Level 3

Level 2

Level 1

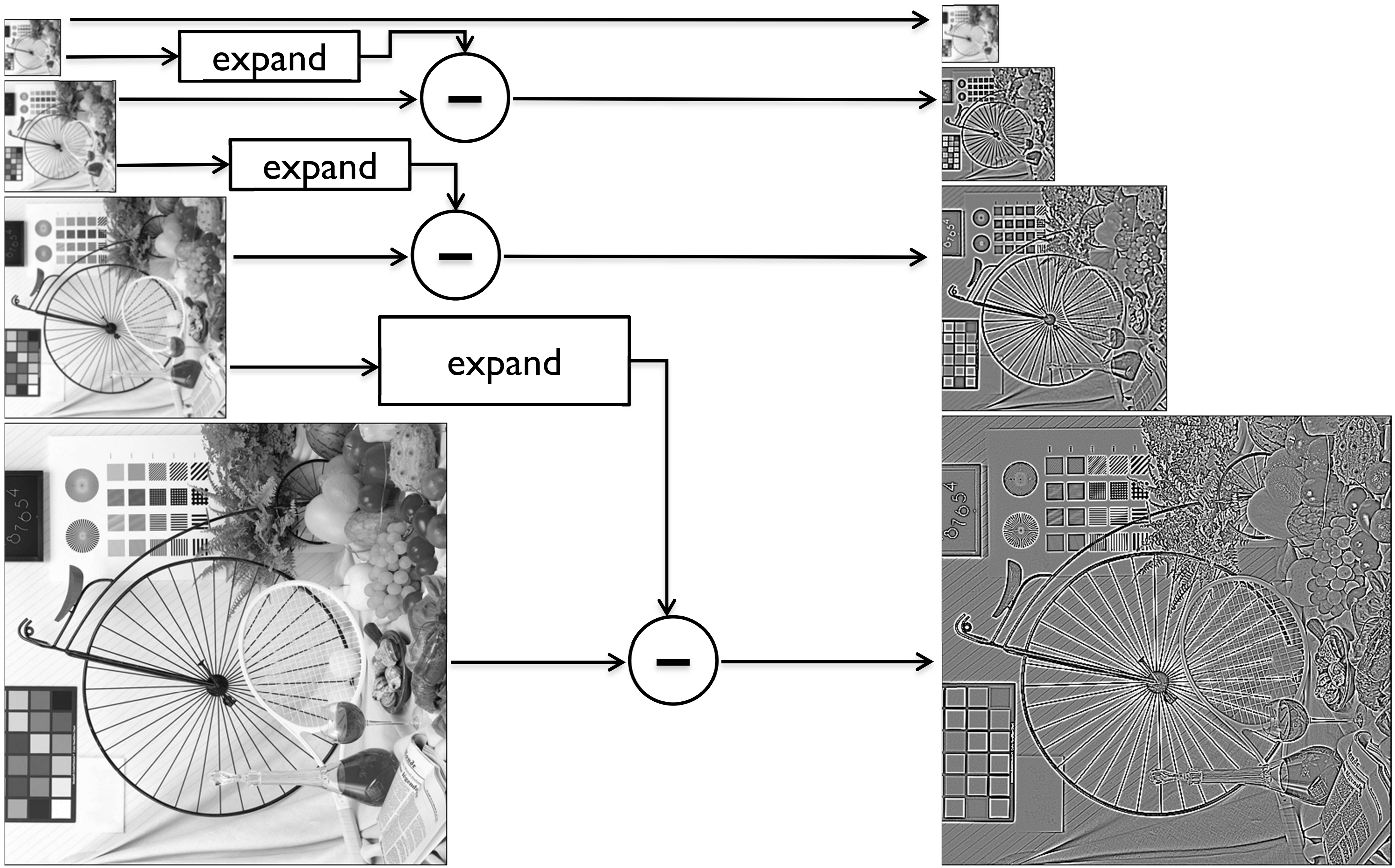
BURT, P. AND ADELSON, E. 1983. The Laplacian Pyramid as a Compact Image Code. *IEEE Transactions on Communications 31*, 4, 532–540.

Gaussian Pyramid

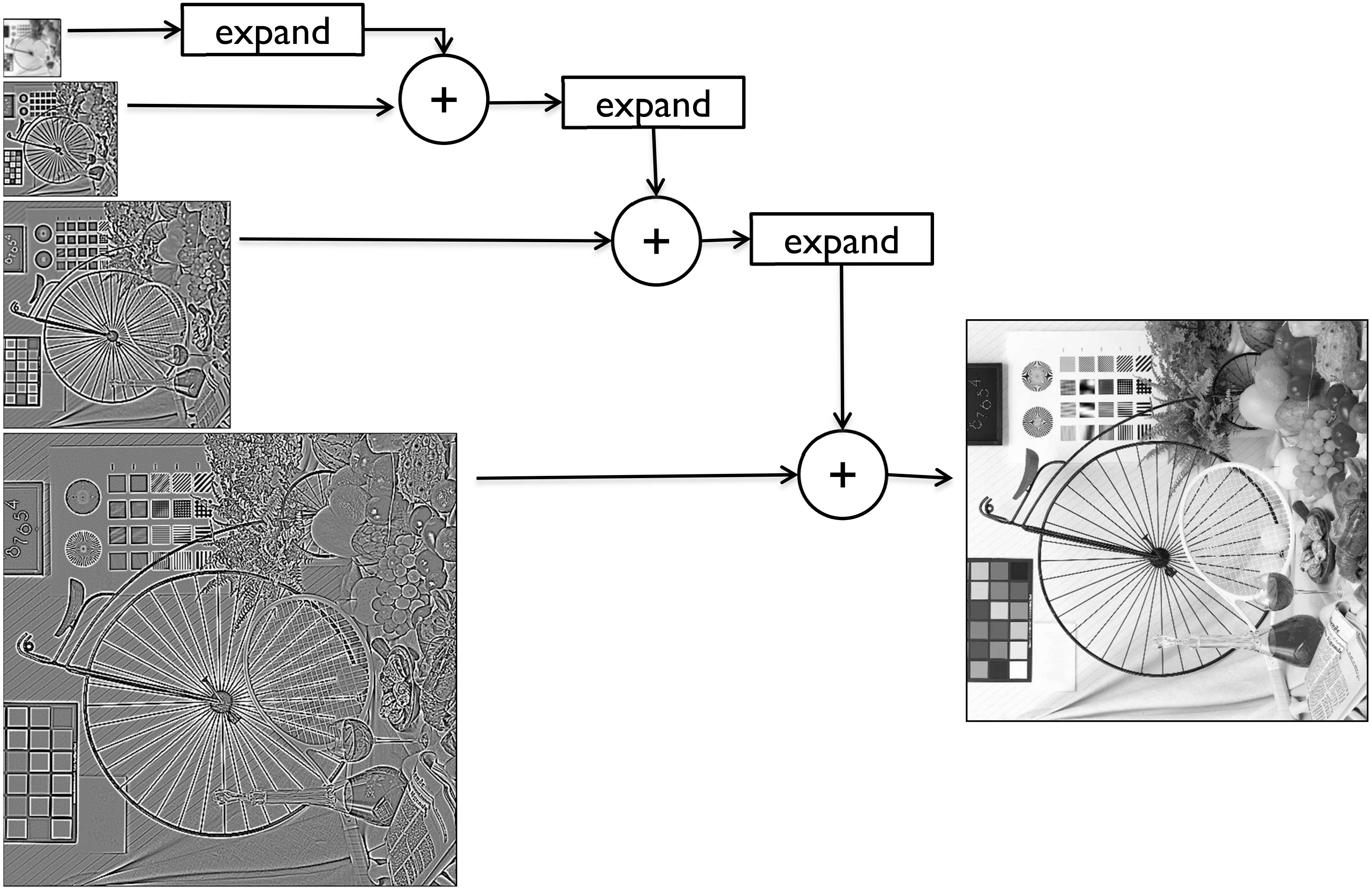


Why is blurring needed?

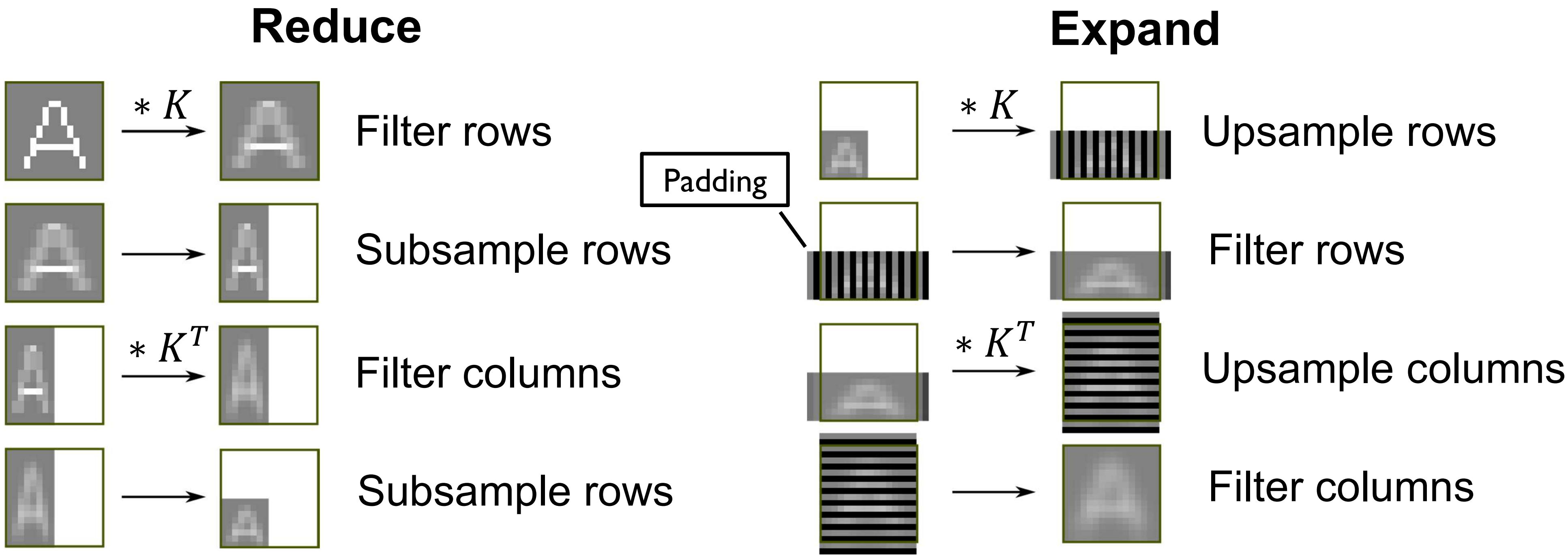
Laplacian Pyramid - decomposition



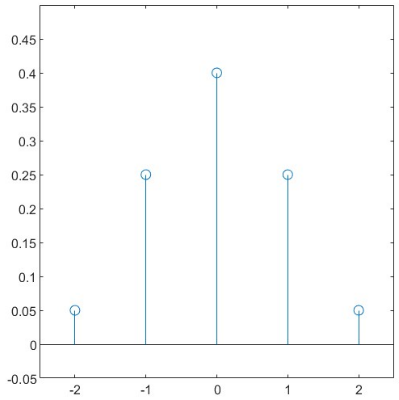
Laplacian Pyramid - synthesis

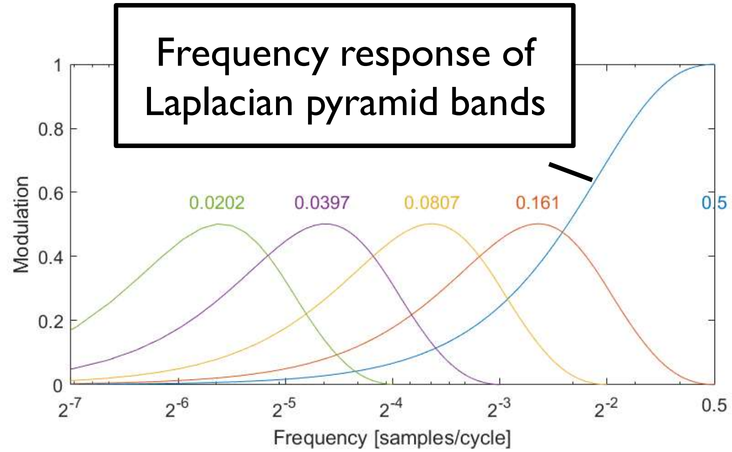


Reduce and expand



$K =$





Example: stitching and blending

Combine two images:

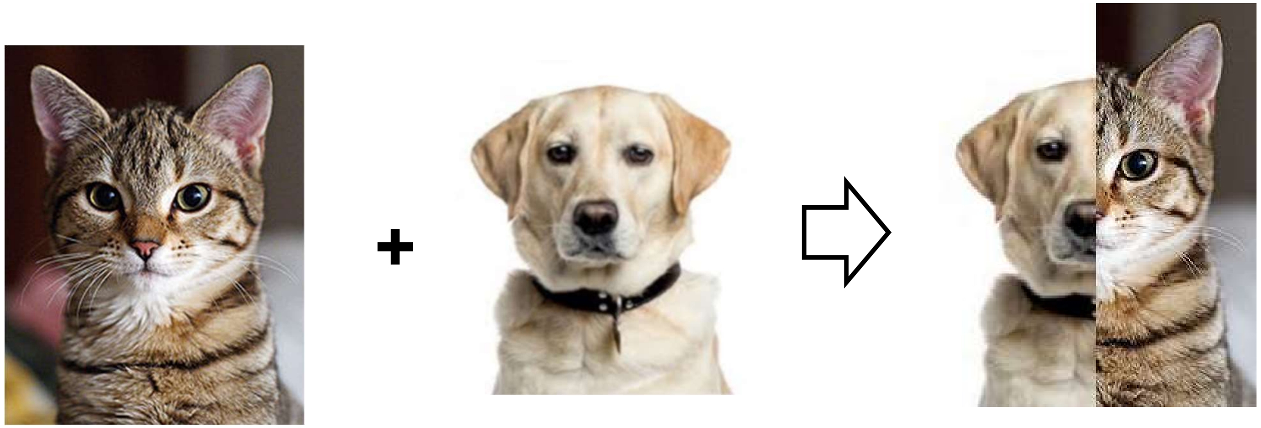


Image-space blending

Laplacian pyramid blending

References

- SZELISKI, R. 2010. *Computer Vision: Algorithms and Applications*. Springer-Verlag New York Inc.
  - Chapter 3
  - http://szeliski.org/Book

UNIVERSITY OF CAMBRIDGE
COMPUTER LABORATORY
RAINBOW GRAPHICS GROUP

Advanced Graphics & Image Processing

Advanced image processing

Part 1/2 – edge stopping filters

Rafał Mantiuk
Computer Laboratory, University of Cambridge

Edge stopping filters

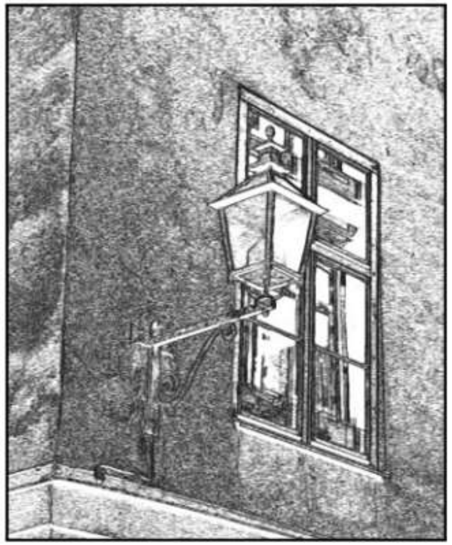
Original

Edge-aware smoothing

Detail enhancement

Stylization

Recoloring



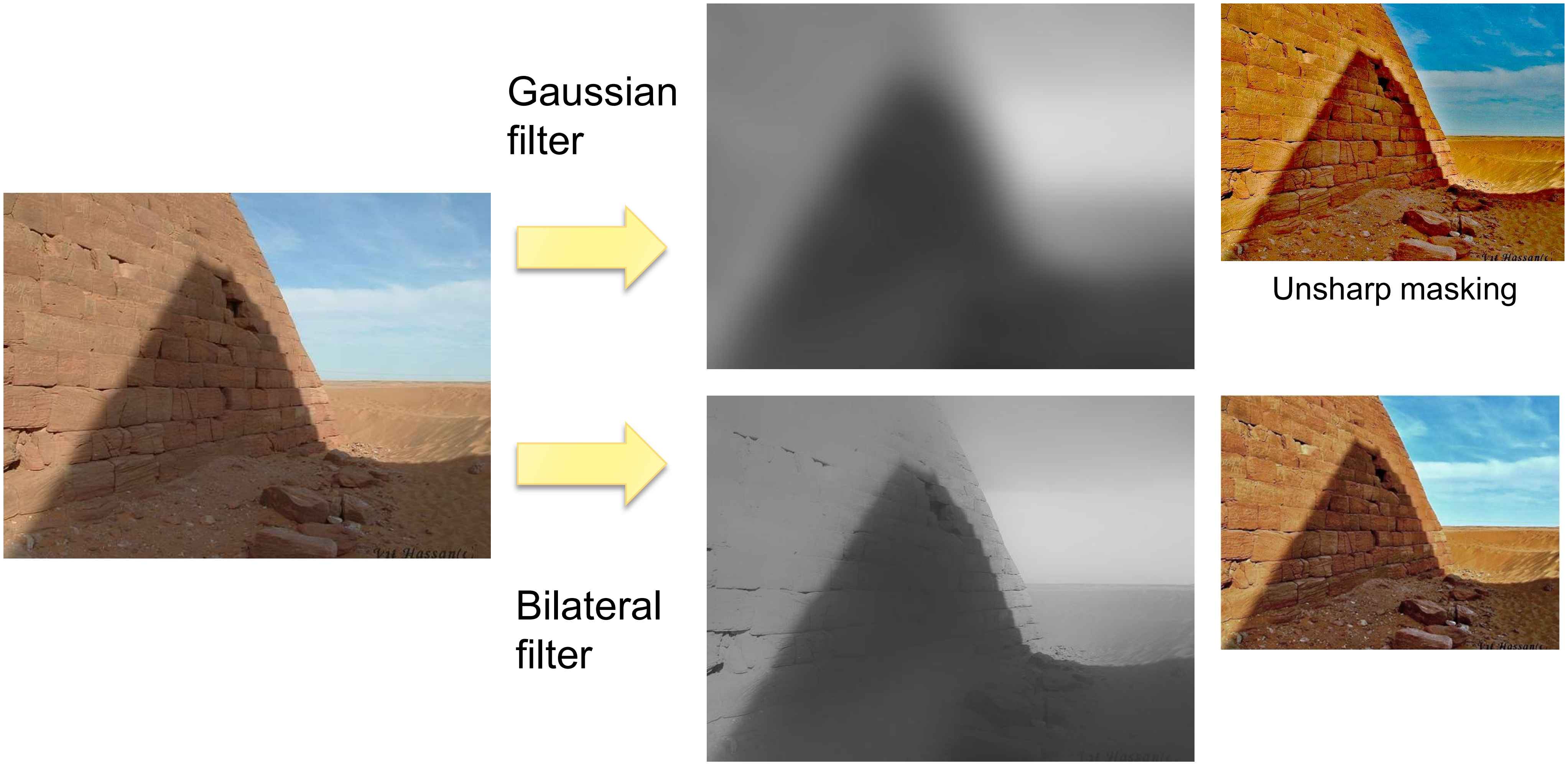
Pencil drawing

Depth-of-field

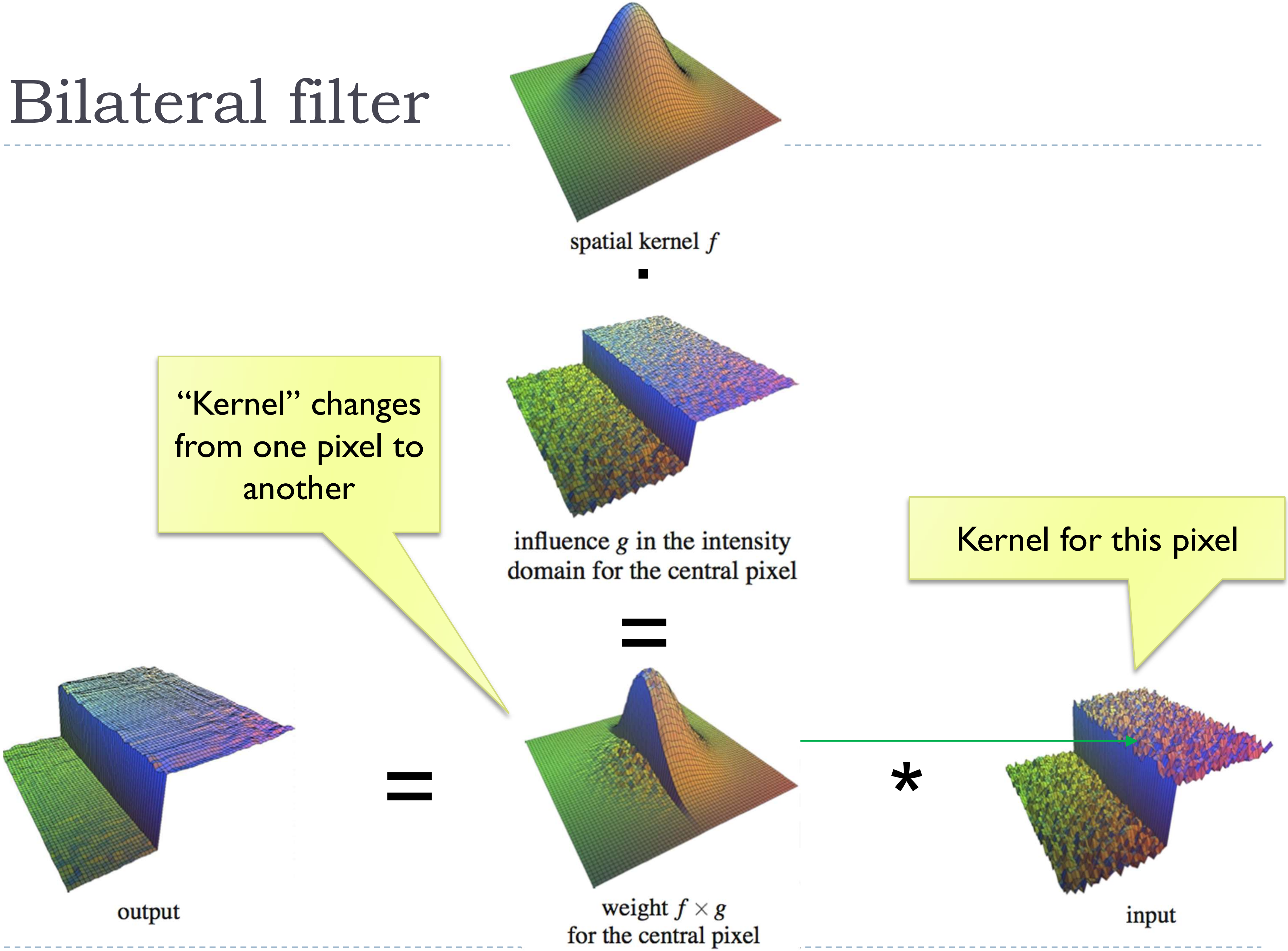
Examples from [Gastal & Oliveira 2011]

Nonlinear filters: Bilateral filter

- Goal: Smooth out the image without blurring edges



Bilateral filter



Bilateral filter

Input image

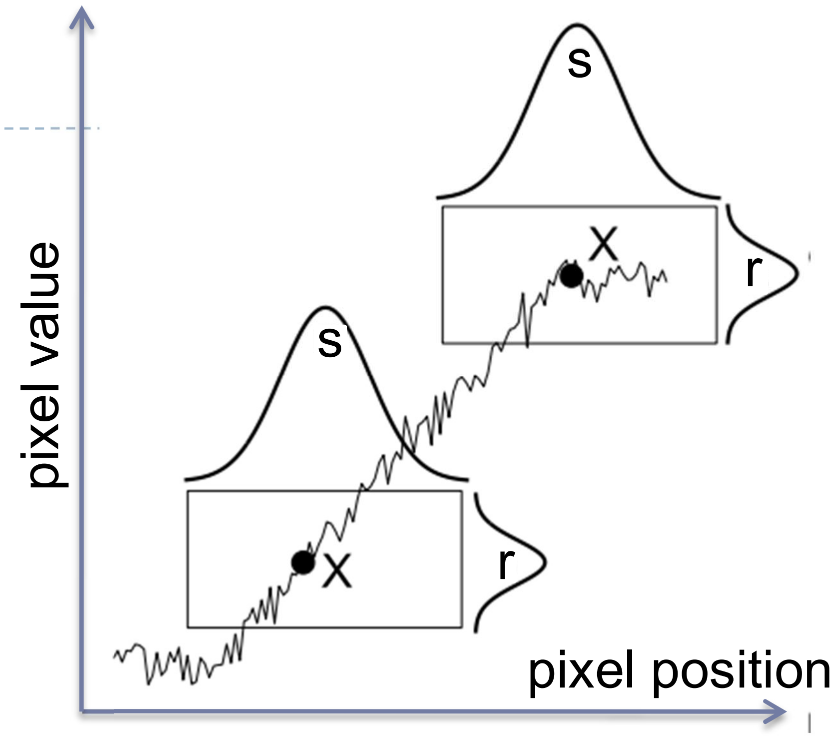
$$y(\boldsymbol{p}) = \frac{\sum_{\boldsymbol{q}\in\Omega} x(\boldsymbol{q}) w(\boldsymbol{p},\boldsymbol{q})}{\sum_{\boldsymbol{q}\in\Omega} w(\boldsymbol{p},\boldsymbol{q})}$$

Pixel coordinates $\boldsymbol{p} = (i, j)$

Neighborhood of the pixel **p**

$$w(\boldsymbol{p},\boldsymbol{q}) = g_s\underbrace{(\boldsymbol{p}-\boldsymbol{q})}_{\text{distance in the spatial position (x,y)}} g_r\underbrace{(x(\boldsymbol{p}) - x(\boldsymbol{q}))}_{\text{distance (difference) in pixel values}}$$

$$g_s(\boldsymbol{d}) = \exp\left(\frac{-\|\boldsymbol{d}\|_2}{2\sigma_s^2}\right) \qquad g_r(d) = \exp\left(\frac{-d^2}{2\sigma_s^2}\right)$$



How to make the bilateral filter fast?

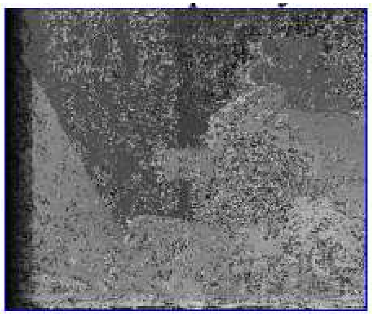
- A number of approximations have been proposed
  - Combination of linear filters [Durand & Dorsey 2002, Yang et al. 2009]
  - Bilateral grid [Chen et al. 2007]
  - Permutohedral lattice [Adams et al. 2010]
  - Domain transform [Gastal & Oliveira 2011]

Joint-bilateral filter (a.k.a guided/cross b.f.)

- The "range" term does not need to operate in the same domain as the filter output
  - Example:

A simplified algorithm from [Mueller et al. 2010]

Stereo image pair



Estimated left-to-right disparity

The "spatial" term operates on disparities

The "range" term operates on the colour image

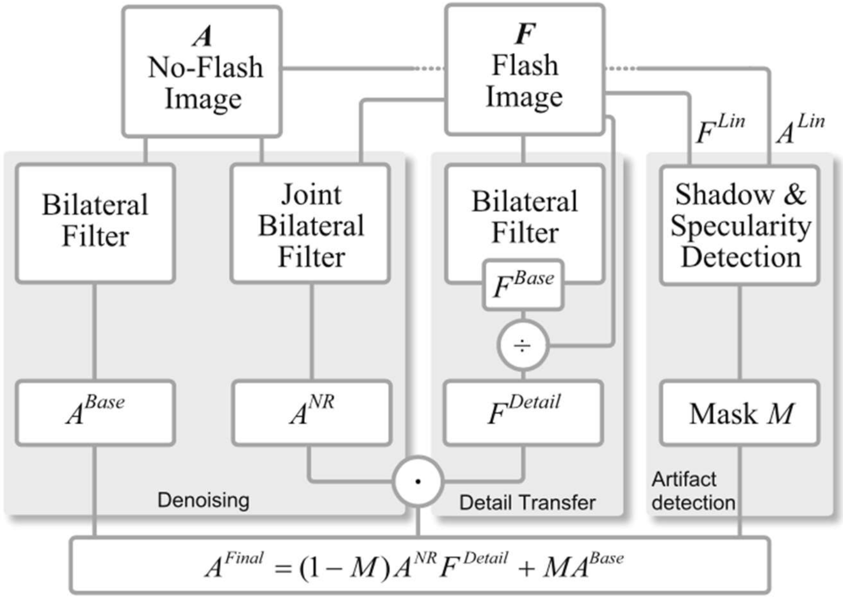
Joint bilateral filter

Filtered disparity

Joint bilateral filter: Flash / no-flash

Flash No-flash

- Preserve colour and illumination from the no-flash image
- Use flash image to remove noise and add details
- [Petshnigg et al. 2004]



Example of edge preserving filtering

- Domain Transform for Edge-Aware Image and Video Processing
- Video:
  - https://youtu.be/UI1xh1IQrTY?t=4m10s
  - From: http://inf.ufrgs.br/~eslgastal/DomainTransform/

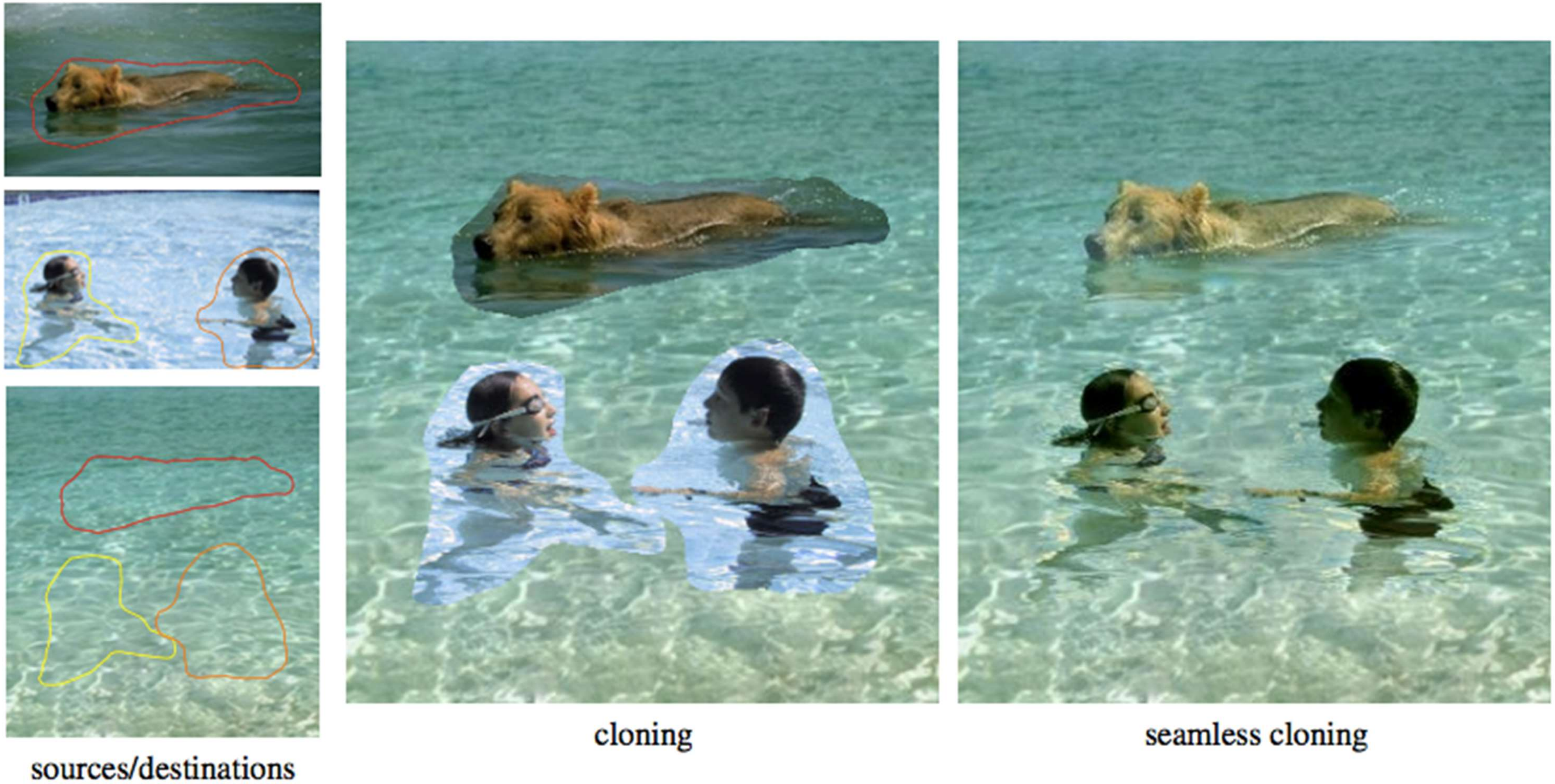
Advanced Graphics & Image Processing

Advanced image processing

Part 1/2 – processing by optimization

Rafał Mantiuk
Computer Laboratory, University of Cambridge

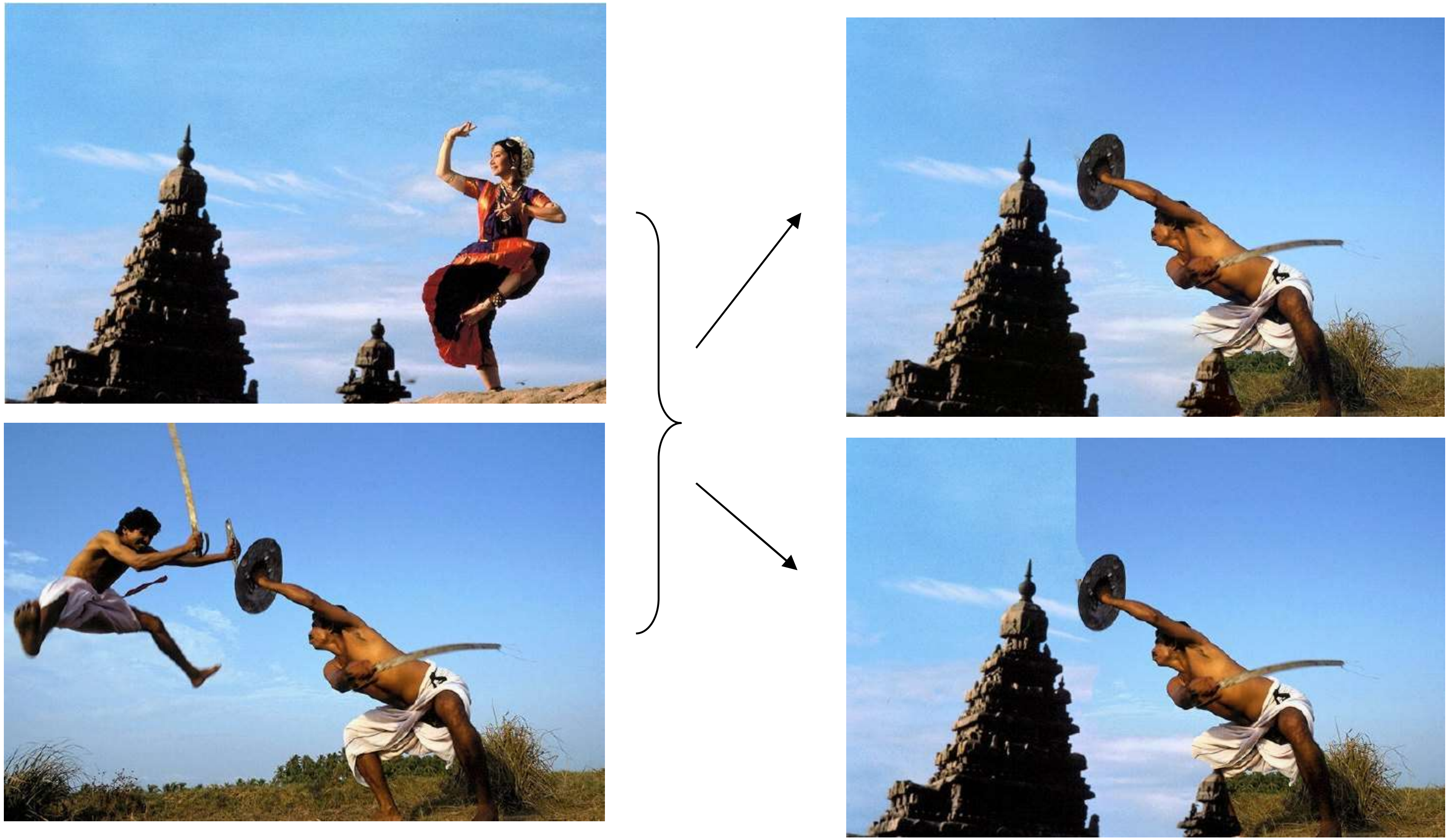
Optimization-based methods



Poisson image editing [Perez et al. 2003]

Gradient Domain compositing

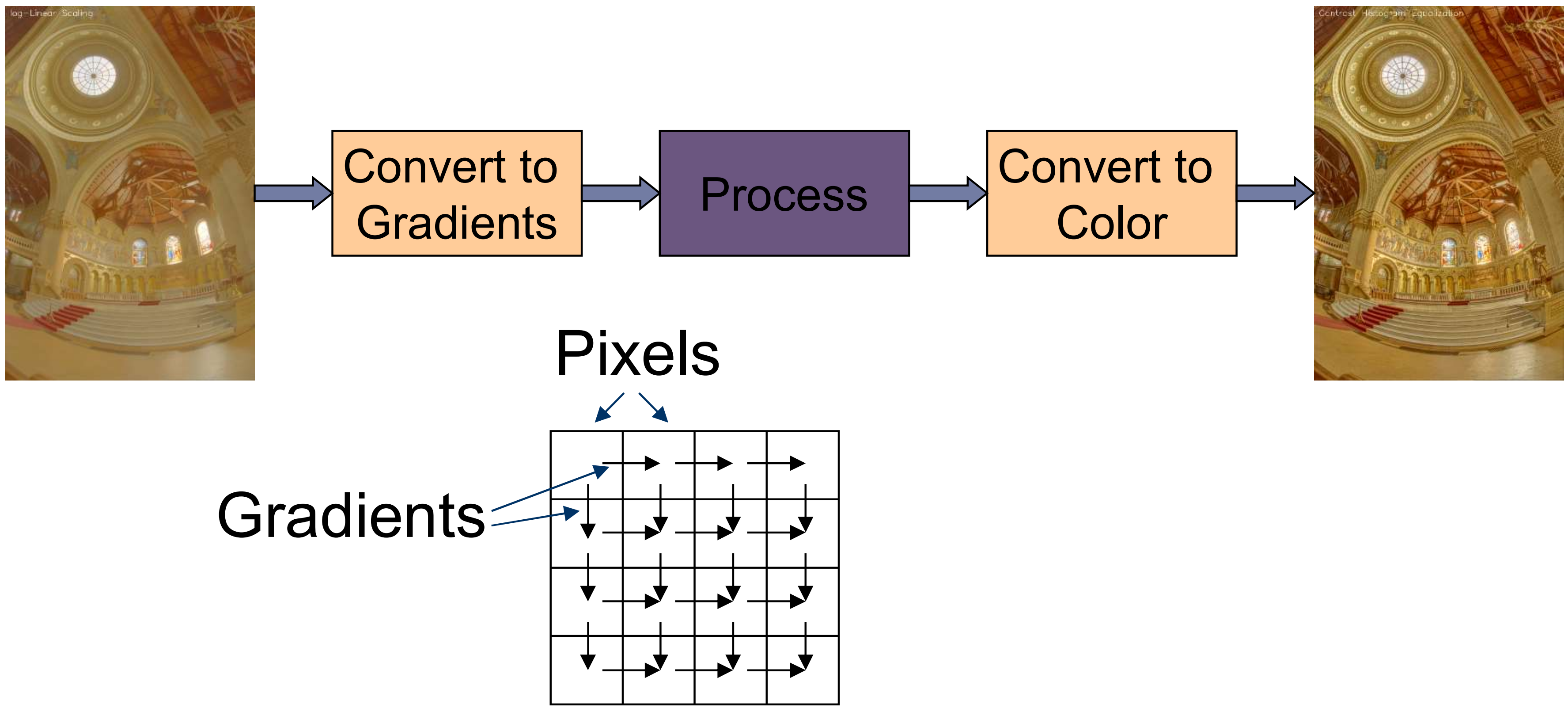
- Compositing [Wang et al. 2004]



 images from [Drori at al. 2004]

Gradient domain methods

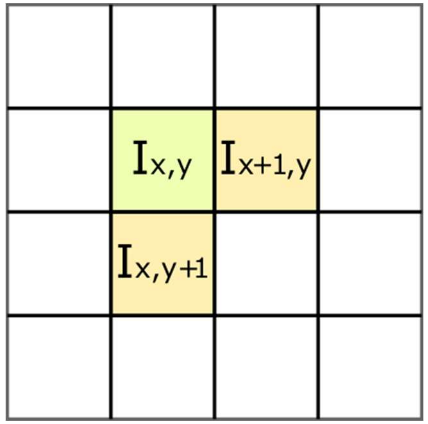
- Operate on pixel gradients instead of pixel values



Forward Transformation

- Forward Transformation
  - Compute gradients as differences between a pixel and its two neighboors

$$\nabla I_{x,y} = \begin{bmatrix} I_{x+1,y} - I_{x,y} \\ I_{x,y+1} - I_{x,y} \end{bmatrix}$$



  - Result: 2D gradient map (2 x more values then the number of pixels)

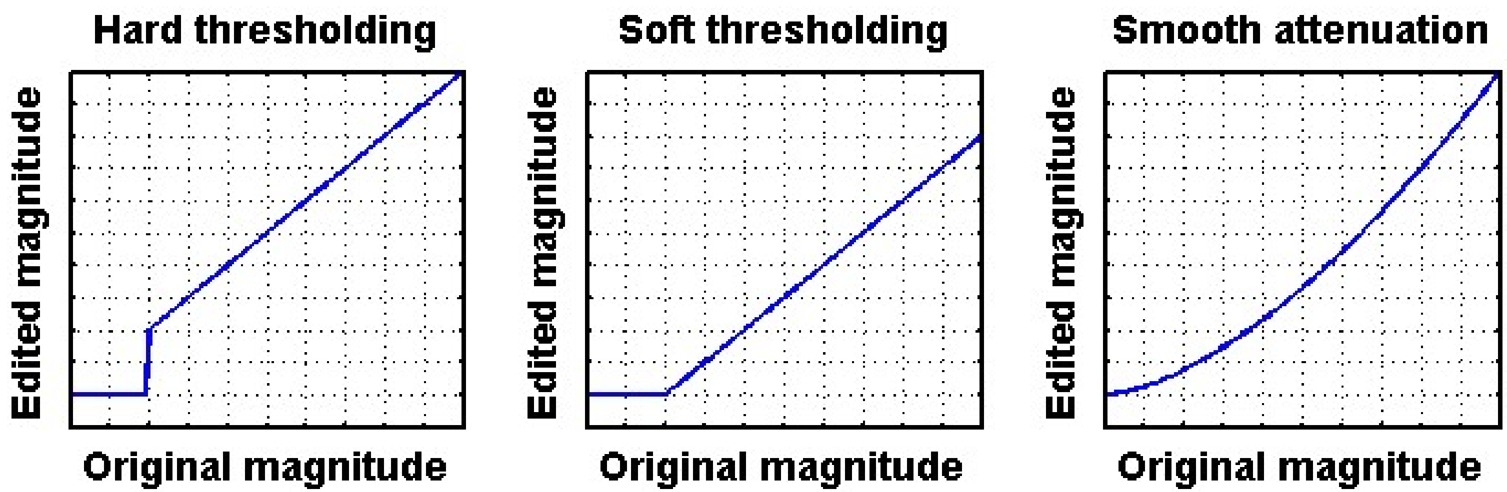
Processing gradient field

- Usually gradient magnitudes are modified while gradient direction (angle) remains the same

Gradient editing function

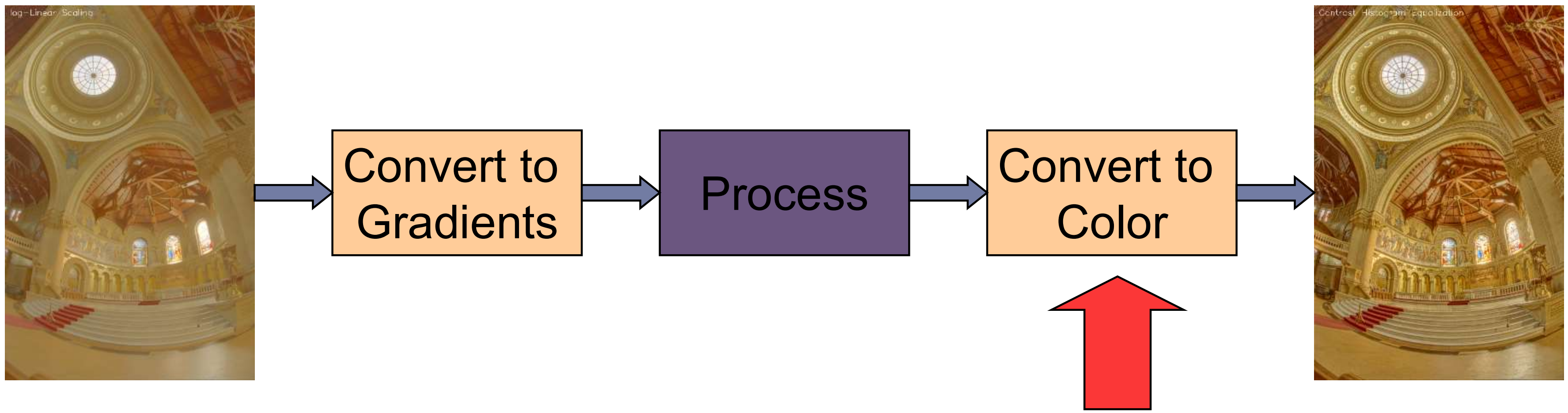
$$G_{x,y} = \nabla I_{x,y} \cdot \frac{f\left(||\nabla I_{x,y}||\right)}{||\nabla I_{x,y}||}$$

- Examples of gradient editing functions:



Inverse transform: the difficult part

- There is no strightforward transformation from gradients to luminance



- Instead, a minimization problem is solved:

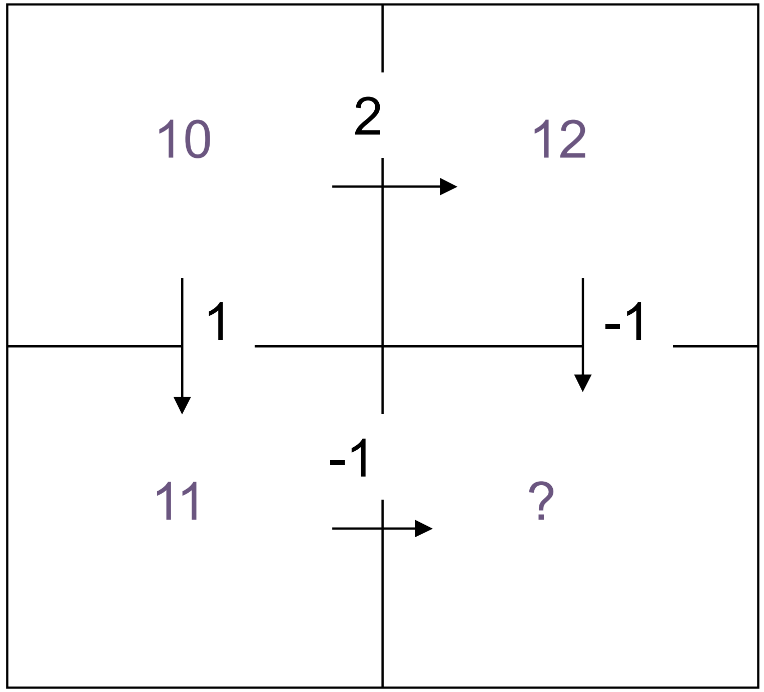
$$\arg\min_{I} \sum_{x,y} \left[\left(I_{x+1,y} - I_{x,y} - G^{(x)}_{x,y} \right)^2 + \left(I_{x,y+1} - I_{x,y} - G^{(y)}_{x,y} \right)^2 \right]$$

Image Pixels

Desired gradients

Inverse transformation

- Convert modified gradients to pixel values
  - Not trivial!
  - Most gradient fields are inconsistent - do not produce valid images
  - If no accurate solution is available, take the best possible solution
  - Analogy: system of springs



Gradient field reconstruction: derivation

- The minimization problem is given by:

$$\arg\min_I \sum_{x,y} \left[\left(I_{x+1,y} - I_{x,y} - G^{(x)}_{x,y} \right)^2 + \left(I_{x,y+1} - I_{x,y} - G^{(y)}_{x,y} \right)^2 \right]$$

- After equating derivatives over pixel values to 0 we get:
  - Derivation done in the lecture

$$I_{x-1,y} + I_{x+1,y} + I_{x,y-1} + I_{x,y+1} - 4I_{x,y} = G^{(x)}_{x,y} - G^{(x)}_{x-1,y} + G^{(y)}_{x,y} - G^{(y)}_{x,y-1}$$

  - In matrix notation:

Laplace operator (NxN matrix)

$$\nabla^2 I = \mathrm{div} G$$

Divergence of a vector field (Nx1 vector)

$$\nabla^2 I_{x,y} = I_{x-1,y} + I_{x+1,y} + I_{x,y-1} + I_{x,y+1} - 4I_{x,y}$$

Image as a column vector

$$\begin{bmatrix} I_{1,1} \\ I_{2,1} \\ \ldots \\ I_{N,M} \end{bmatrix}$$

$$\mathrm{div} G_{x,y} = G^{(x)}_{x,y} - G^{(x)}_{x-1,y} + G^{(y)}_{x,y} - G^{(y)}_{x,y-1}$$

Laplace operator for 3x3 image

$$\nabla^2 = \begin{bmatrix} -2 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -3 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -3 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -3 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -3 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -2 \end{bmatrix}$$

Solving sparse linear systems

- Just use “\” operator in Matlab / Octave:
  - x = A \ b;
- Great “cookbook”:
  - TEUKOLSKY, S.A., FLANNERY, B.P., PRESS, W.H., AND VETTERLING, W.T. 1992. *Numerical recipes in C*. Cambridge University Press, Cambridge.
- Some general methods
  - Cosine-transform – fast but cannot work with weights (next slides) and may suffer from floating point precision errors
  - Multi-grid – fast, difficult to implement, not very flexible
  - Conjugate gradient / bi-conjugate gradient – general, memory efficient, iterative but fast converging
  - Cholesky decomposition – effective when working on sparse matrices

Pinching artefacts

- A common problem of gradient-based methods is that they may result in "pinching" artefacts (left image)
- Such artefacts can be avoided by introducing weights to the optimization problem

Weighted gradients

- The new objective function is:

$$\arg\min_I \sum_{x,y} \left[w^{(x)}_{x,y} \left(I_{x+1,y} - I_{x,y} - G^{(x)}_{x,y} \right)^2 + w^{(y)}_{x,y} \left(I_{x,y+1} - I_{x,y} - G^{(y)}_{x,y} \right)^2 \right]$$

- so that higher weights are assigned to low gradient magnitudes (in the original image).

$$w^{(x)}_{x,y} = w^{(y)}_{x,y} = \frac{1}{||\nabla I^{(o)}_{x,y}|| + \epsilon}$$

- The linear system can be derived again
  - but this is a lot of work and is error-prone

Weighted gradients - matrix notation (1)

- The objective function:

$$\arg\min_I \sum_{x,y} \left[w_{x,y}^{(x)} \left(I_{x+1,y} - I_{x,y} - G_{x,y}^{(x)} \right)^2 + w_{x,y}^{(y)} \left(I_{x,y+1} - I_{x,y} - G_{x,y}^{(y)} \right)^2 \right]$$

- In the matrix notation (without weights for now):

$$\arg\min_I \left\| \begin{bmatrix} \nabla_x \\ \nabla_y \end{bmatrix} I - \begin{bmatrix} G^{(x)} \\ G^{(y)} \end{bmatrix} \right\|^2$$

- Gradient operators (for 3x3 pixel image):

$$\nabla_x = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \nabla_y = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Weighted gradients - matrix notation (2)

- The objective function again: $$\arg\min_I \left\| \begin{bmatrix} \nabla_x \\ \nabla_y \end{bmatrix} I - \begin{bmatrix} G^{(x)} \\ G^{(y)} \end{bmatrix} \right\|^2$$
- Such over-determined least-square problem can be solved using pseudo-inverse:

$$\begin{bmatrix} \nabla'_x & \nabla'_y \end{bmatrix} \begin{bmatrix} \nabla_x \\ \nabla_y \end{bmatrix} I = \begin{bmatrix} \nabla'_x & \nabla'_y \end{bmatrix} \begin{bmatrix} G^{(x)} \\ G^{(y)} \end{bmatrix}$$

- Or simply:

$$\left(\nabla'_x \nabla_x + \nabla'_y \nabla_y \right) I = \nabla'_x G^{(x)} + \nabla'_y G^{(y)}$$

- With weights:

$$\left(\nabla'_x W \nabla_x + \nabla'_y W \nabla_y \right) I = \nabla'_x W G^{(x)} + \nabla'_y W G^{(y)}$$

WLS filter: Edge stopping filter by optimization

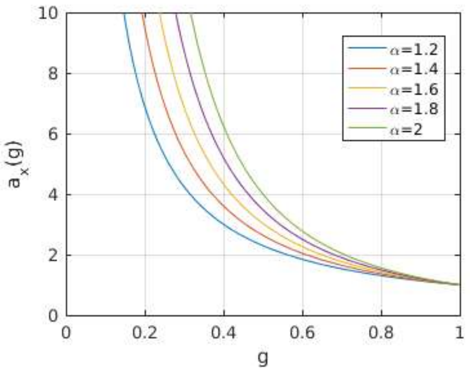
- Weighted-least-squares optimization

Make reconstructed image *u* possibly close to input *g*

Smooth out the image by making partial derivatives close to 0

$$\underset{\boldsymbol{u}}{\operatorname{argmin}} \sum_p \left((u_p - g_p)^2 + \lambda \left(a_{x,p}(g) \left(\frac{\partial u}{\partial x} \right)_p^2 + a_{y,p}(g) \left(\frac{\partial u}{\partial y} \right)_p^2 \right) \right)$$

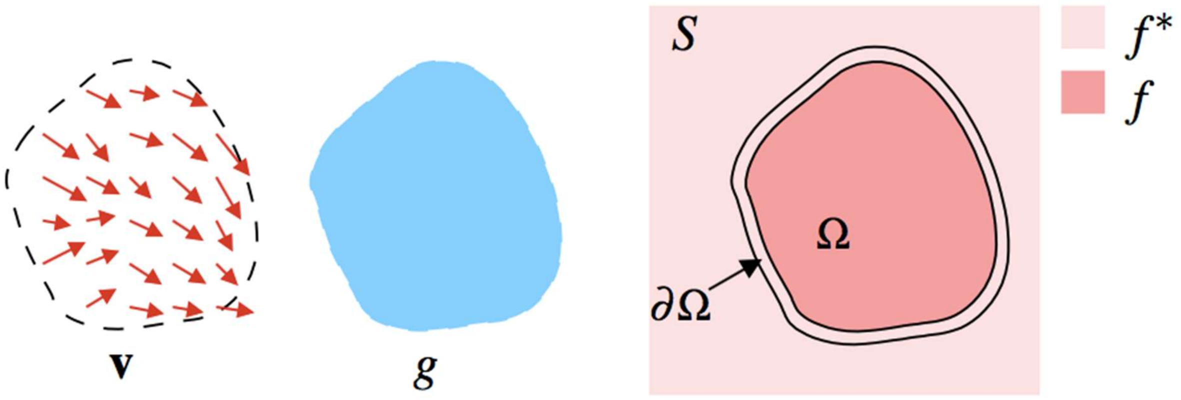
Spatially varying smoothing – less smoothing near the edges



$$a_{x,p}(g) = \frac{1}{\left| \frac{\partial u}{\partial x}(g) \right|^{\alpha} + \epsilon}$$

- [Farbman, Z., Fattal, R., Lischinski, D., & Szeliski, R. (2008). Edge-preserving decompositions for multi-scale tone and detail manipulation. *ACM SIGGRAPH 2008*, 1–10.]

Poisson image editing

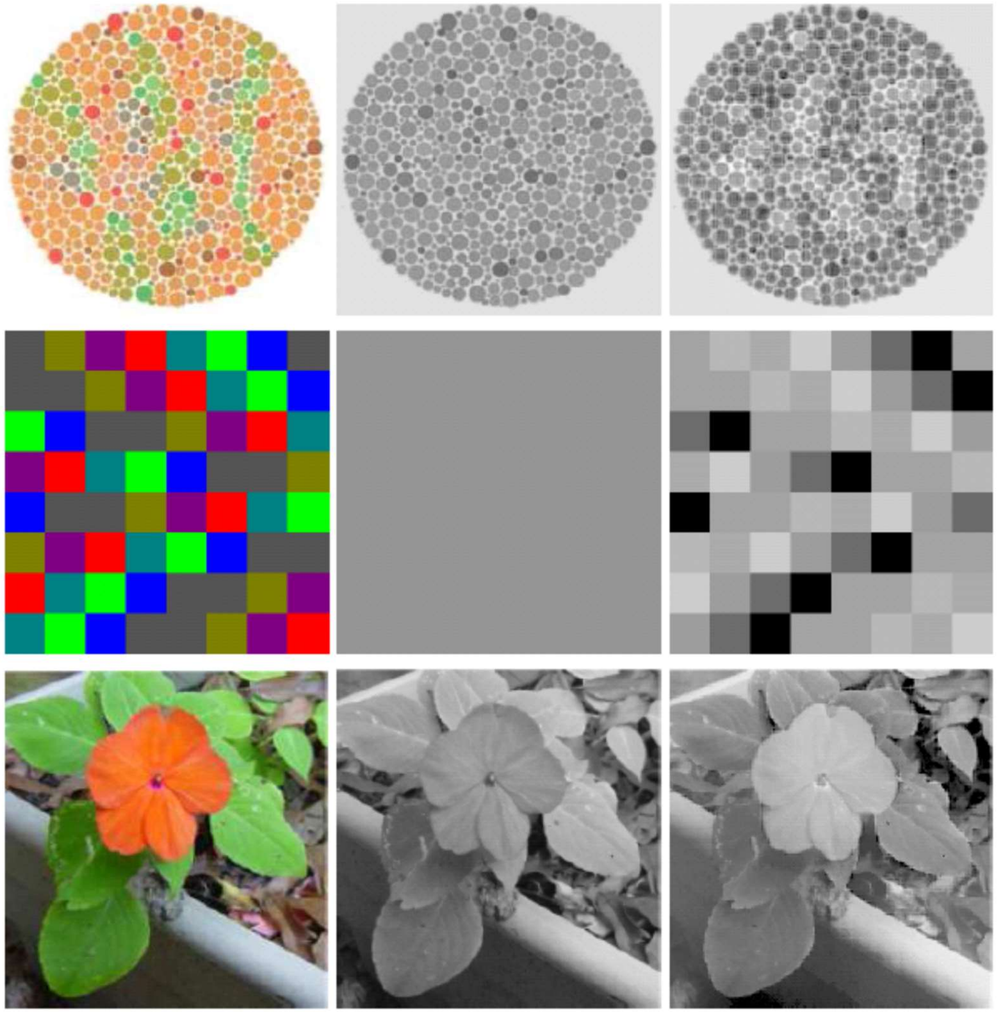


$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \quad \text{subject to:} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

- Reconstruct unknown values *f* given a source guidance gradient field *v* and the boundary conditions $f|_{\partial\Omega} = f^*|_{\partial\Omega}$
- [Pérez, P., Michel Gangnet, & Blake, A. (2003). Poisson Image Editing. *ACM Transactions on Graphics*, *3*(22), 313–318. https://doi.org/10.1145/882262.882269]
-

Color 2 Gray

- Transform color images to gray scale
- Preserve color saliency
  - When gradient in luminance close to 0
  - Replace it with gradient in chrominance
  - Reconstruct an image from gradients
- Gooch, A. A., Olsen, S. C., Tumblin, J., & Gooch, B. (2005). Color2Gray. *ACM Transactions on Graphics*, *24*(3), 634. https://doi.org/10.1145/1073204.1073241



Gradient Domain: applications

- More applications:
  - Lightness perception (Retinex) [Horn 1974]
  - Matting [Sun et al. 2004]
  - Color to gray mapping [Gooch et al. 2005]
  - Video Editing [Perez at al. 2003, Agarwala et al. 2004]
  - Photoshop's Healing Brush [Georgiev 2005]

References

- F. Durand and J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 257–266, Jul. 2002.
- E. S. L. Gastal and M. M. Oliveira, "Domain transform for edge-aware image and video processing," *ACM Trans. Graph.*, vol. 30, no. 4, p. 1, Jul. 2011.
- Patrick Pérez, Michel Gangnet, and Andrew Blake. 2003. Poisson image editing. *ACM Trans. Graph.* 22, 3 (July 2003), 313-318. DOI: http://dx.doi.org/10.1145/882262.882269
- Zeev Farbman, Raanan Fattal, Dani Lischinski, and Richard Szeliski. 2008. Edge-preserving decompositions for multi-scale tone and detail manipulation. *ACM Trans. Graph.* 27, 3, Article 67 (August 2008), 10 pages. DOI: https://doi.org/10.1145/1360612.1360666

UNIVERSITY OF CAMBRIDGE
COMPUTER LABORATORY

RAINBOW GRAPHICS GROUP

Advanced Graphics & Image Processing

Parallel programming in OpenCL

Part 1/3 – OpenCL framework

Rafał Mantiuk

Computer Laboratory, University of Cambridge

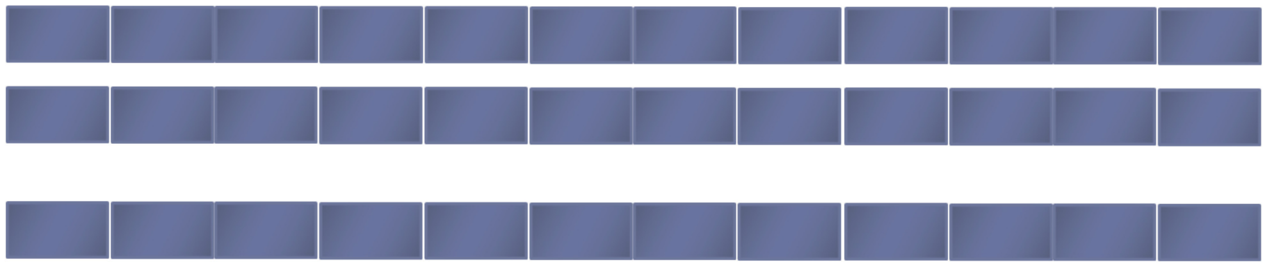
Single Program Multiple Data (SPMD)

- Consider the following vector addition example

```
for( i = 0:11 ) {
   C[ i ] = A[ i ] + B[ i ]
}
```

Serial program:
one program completes
the entire task

A
+
B
||
C



Multiple copies of the same program execute on different data in parallel

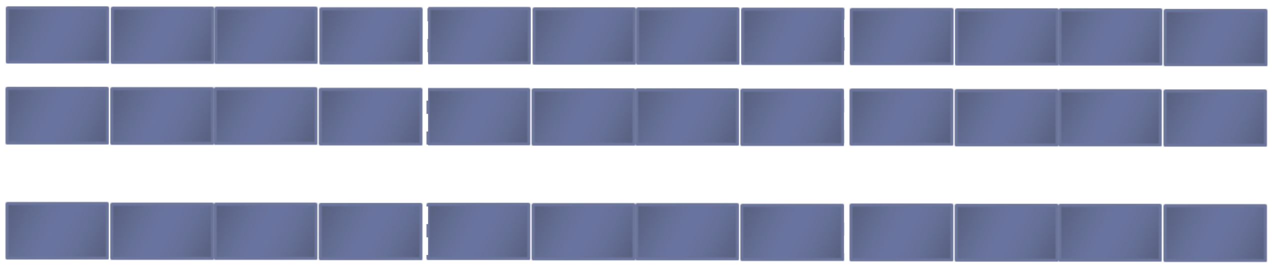
```
for( i = 0:3 ) {
   C[ i ] = A[ i ] + B[ i ]
}
```

```
for( i = 4:7 ) {
   C[ i ] = A[ i ] + B[ i ]
}
```

```
for( i = 8:11 ) {
   C[ i ] = A[ i ] + B[ i ]
}
```

SPMD program:
multiple copies of the
same program run on
different chunks of the
data

A
+
B
||
C



From: OpenCL 1.2 University Kit - http://developer.amd.com/partners/university-programs/

Parallel Software – SPMD

- In the vector addition example, each chunk of data could be executed as an independent thread
- On modern CPUs, the overhead of creating threads is so high that the chunks need to be large
  - In practice, usually a few threads (about as many as the number of CPU cores) and each is given a large amount of work to do
- For GPU programming, there is low overhead for thread creation, so we can create one thread per loop iteration

From: OpenCL 1.2 University Kit - http://developer.amd.com/partners/university-programs/

Parallel Software – SPMD

Single-threaded (CPU)

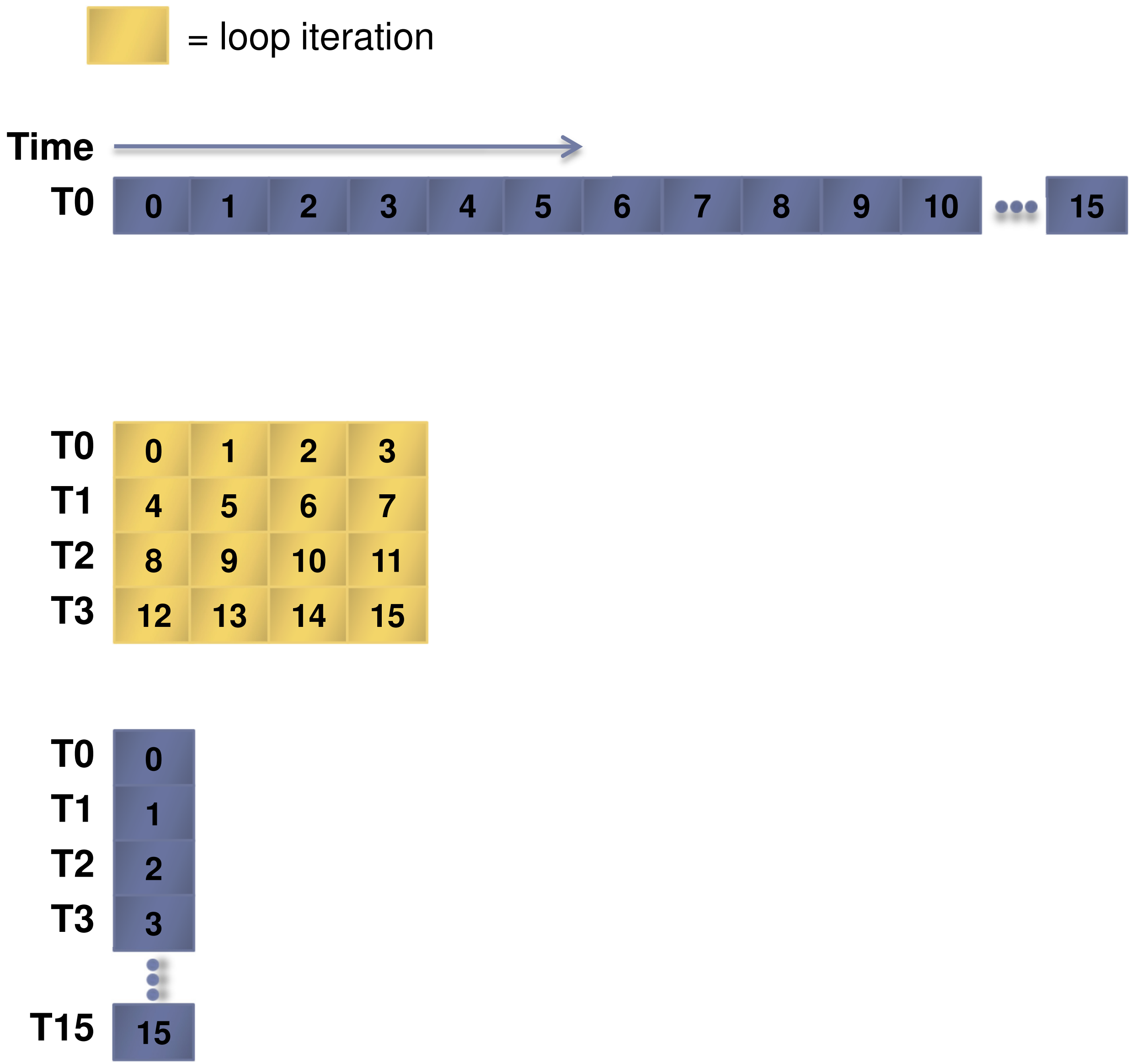
```
// there are N elements
for(i = 0; i < N; i++)
  C[i] = A[i] + B[i]
```

Multi-threaded (CPU)

```
// tid is the thread id
// P is the number of cores
for(i = 0; i < tid*N/P; i++)
  C[i] = A[i] + B[i]
```

Massively Multi-threaded (GPU)

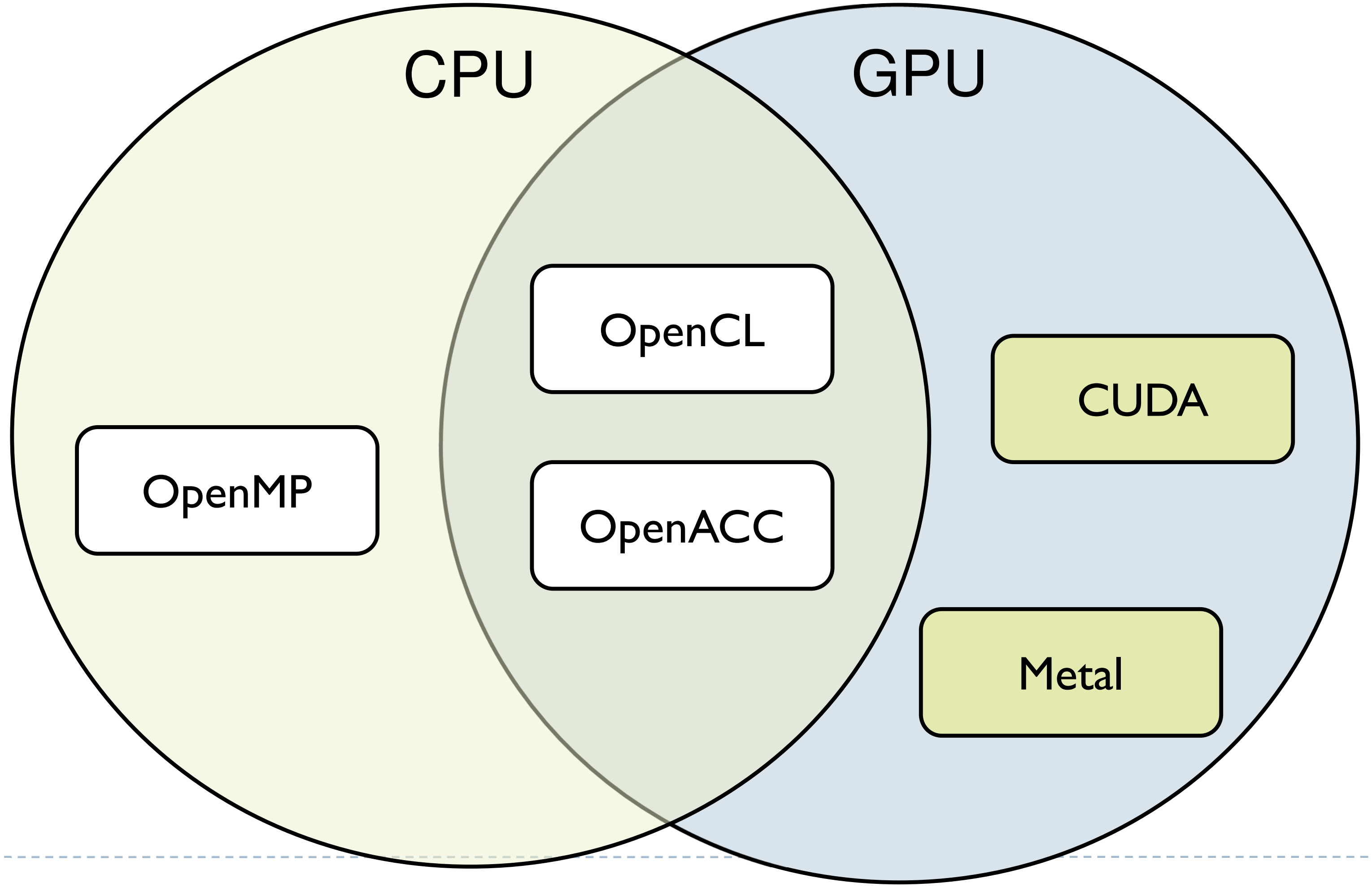
```
// tid is the thread id
C[tid] = A[tid] + B[tid]
```



From: OpenCL 1.2 University Kit - http://developer.amd.com/partners/university-programs/

Parallel programming frameworks

- These are some of more relevant frameworks for creating parallelized code



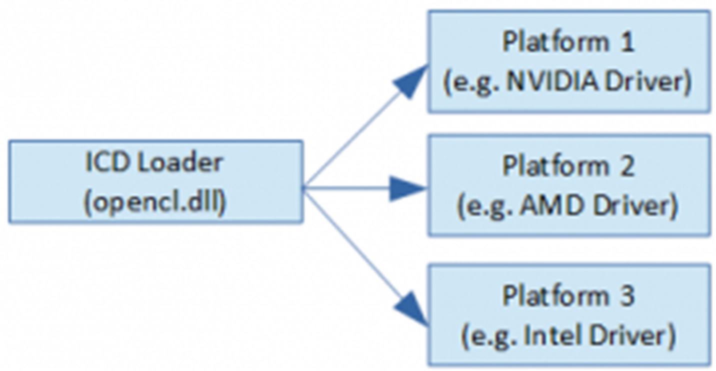
OpenCL

OpenCL

- OpenCL is a framework for writing parallelized code for CPUs, GPUs, DSPs, FPGAs and other processors
- Initially developed by Apple, now supported by AMD, IBM, Qualcomm, Intel and Nvidia (reluctantly)
- Versions
  - Latest: OpenCL 2.2
    - OpenCL C++ kernel language
    - SPIR-V as intermediate representation for kernels
      - Vulcan uses the same Standard Portable Intermediate Representation
    - AMD, Intel
  - Mostly supported: OpenCL 1.2
    - Nvidia, OSX

OpenCL platforms and drivers

- To run OpenCL code you need:
  - Generic ICD loader
    - Included in the OS
  - Installable Client Driver
    - From Nvidia, Intel, etc.
  - This applies to Windows and Linux, only one platform on Mac
- To develop OpenCL code you need:
  - OpenCL headers/libraries
    - Included in the SDKs
      - Nvidia – CUDA Toolkit
      - Intel OpenCL SDK
    - But lightweight options are also available



Programming OpenCL

- OpenCL natively offers C99 API
- But there is also a standard OpenCL C++ API wrapper
  - Strongly recommended – reduces the amount of code
- Programming OpenCL is similar to programming shaders in OpenGL
  - Host code runs on CPU and invokes **kernels**
  - Kernels are written in C-like programming language
    - In many respects similar to GLSL
  - Kernels are passed to API as strings and compiled at runtime
    - Kernels are usually stored in text files
    - Kernels can be precompiled into SPIR from OpenCL 2.1

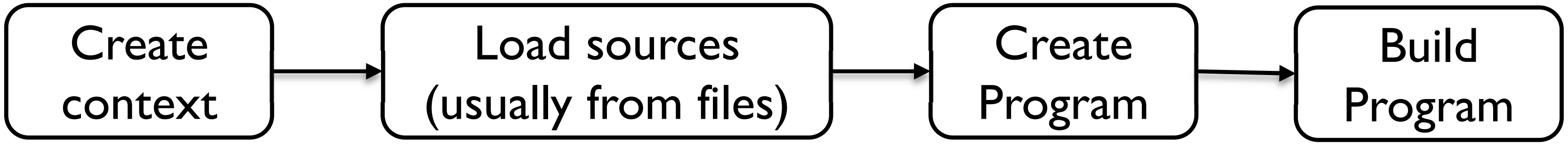
Example: Step 1 - Select device

Get all Platforms → Select Platform → Get all Devices → Select Device

```
//get all platforms (drivers)
std::vector<cl::Platform> all_platforms;
cl::Platform::get(&all_platforms);
if (all_platforms.size() == 0){
    std::cout << " No platforms found. Check OpenCL installation!\n";
    exit(1);
}
cl::Platform default_platform = all_platforms[0];
std::cout << "Using platform: " << default_platform.getInfo<CL_PLATFORM_NAME>() << "\n";

//get default device of the default platform
std::vector<cl::Device> all_devices;
default_platform.getDevices(CL_DEVICE_TYPE_ALL, &all_devices);
if (all_devices.size() == 0){
    std::cout << " No devices found. Check OpenCL installation!\n";
    exit(1);
}
cl::Device default_device = all_devices[0];
std::cout << "Using device: " << default_device.getInfo<CL_DEVICE_NAME>() << "\n";
```

Example: Step 2 - Build program

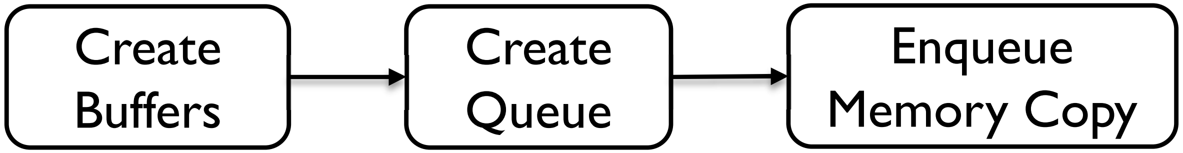



```cpp
cl::Context context({ default_device });

cl::Program::Sources sources;
// kernel calculates for each element C=A+B
std::string kernel_code =
    "__kernel void simple_add(__global const int* A, __global const int* B, __global int* C) {"
    "    int index = get_global_id(0);"
    "    C[index] = A[index] + B[index];"
    "};";
sources.push_back({ kernel_code.c_str(), kernel_code.length() });

cl::Program program(context, sources);
try {
    program.build({ default_device });
}
catch (cl::Error err) {
    std::cout << " Error building: " <<
        program.getBuildInfo<CL_PROGRAM_BUILD_LOG>(default_device) << "\n";
    exit(1);
}
```

Example: Step 3 - Create Buffers and copy memory




```
// create buffers on the device
cl::Buffer buffer_A(context, CL_MEM_READ_WRITE, sizeof(int) * 10);
cl::Buffer buffer_B(context, CL_MEM_READ_WRITE, sizeof(int) * 10);
cl::Buffer buffer_C(context, CL_MEM_READ_WRITE, sizeof(int) * 10);

int A[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
int B[] = { 0, 1, 2, 0, 1, 2, 0, 1, 2, 0 };

//create queue to which we will push commands for   the device.
cl::CommandQueue queue(context, default_device);

//write arrays A and B to the device
queue.enqueueWriteBuffer(buffer_A, CL_TRUE, 0, sizeof(int) * 10, A);
queue.enqueueWriteBuffer(buffer_B, CL_TRUE, 0, sizeof(int) * 10, B);
```

Example: Step 4 - Execute Kernel and retrieve the results

Create Kernel → Set Kernel Arguments → Enqueue Kernel → Enqueue memory copy

```
cl::Kernel kernel(program, "simple_add");

kernel.setArg(0, buffer_A);
kernel.setArg(1, buffer_B);
kernel.setArg(2, buffer_C);
queue.enqueueNDRangeKernel(kernel, cl::NullRange, cl::NDRange(10), cl::NullRange);

int C[10];
//read result C from the device to array C
queue.enqueueReadBuffer(buffer_C, CL_TRUE, 0, sizeof(int) * 10, C);
queue.finish();

std::cout << " result: \n";
for (int i = 0; i < 10; i++){
    std::cout << C[i] << " ";
}
std::cout << std::endl;
```

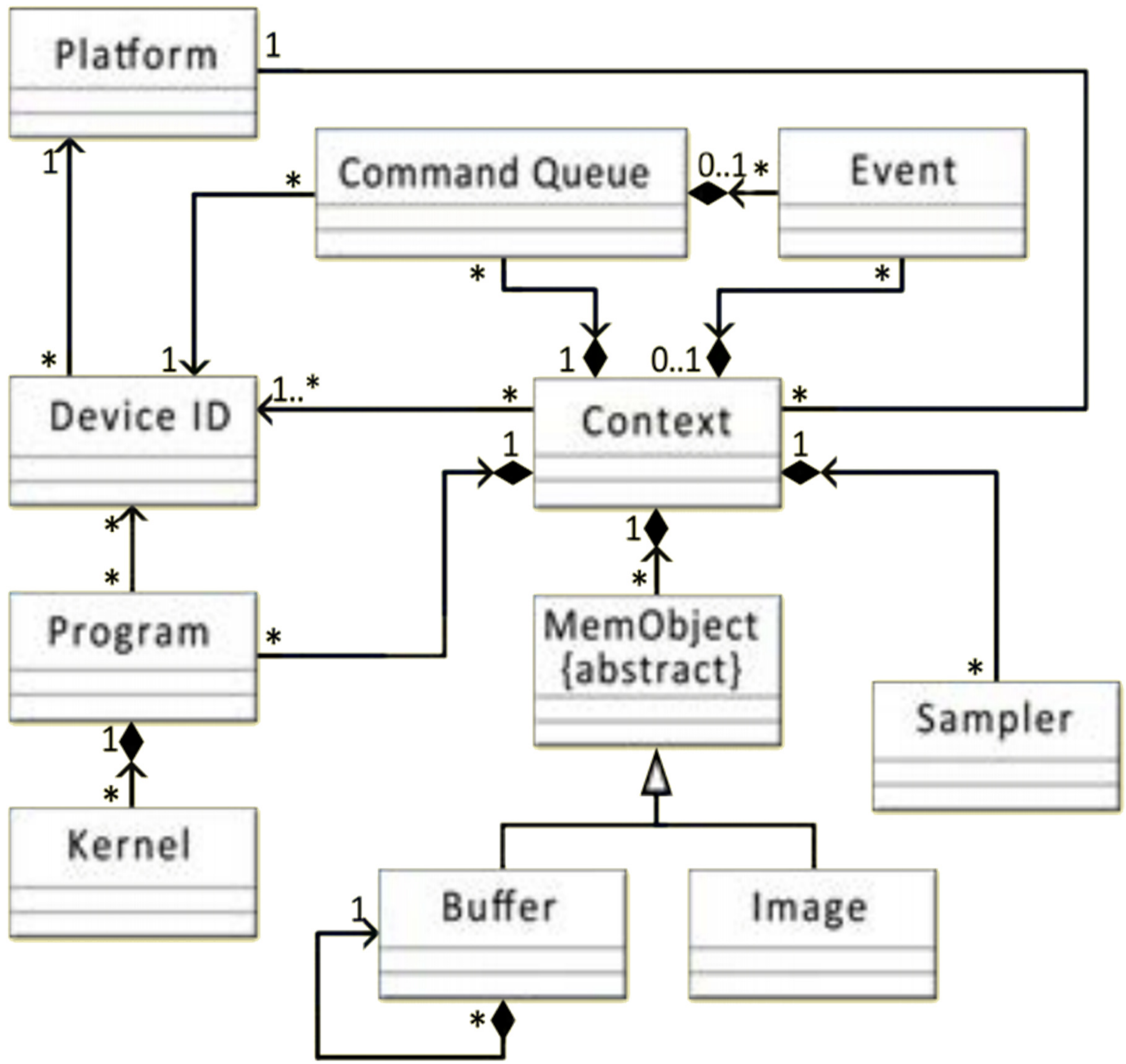
Our Kernel was

```
__kernel void simple_add(__read_only const int* A,
                         __read_only const int* B,
                         __write_only int* C) {

    int index = get_global_id(0);
    C[index]=A[index]+B[index];
};
```

OpenCL API Class Diagram

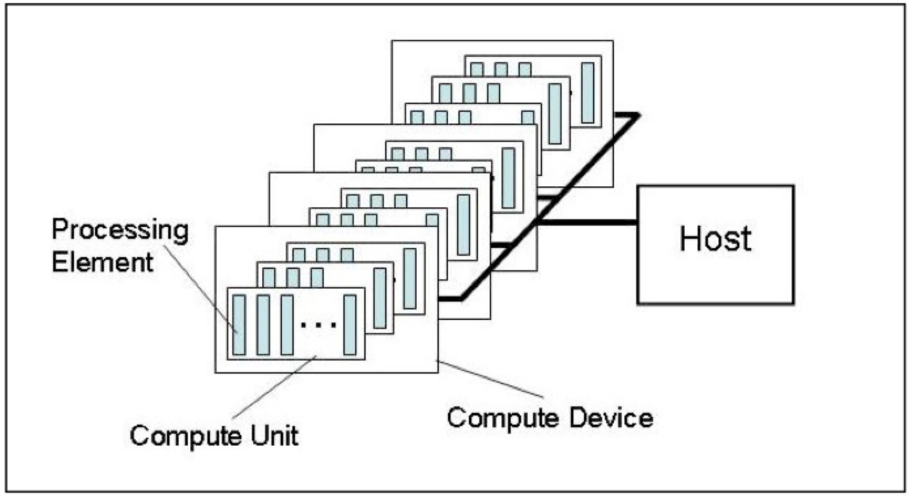
- *Platform* – Nvidia CUDA
- *Device* – GeForce 780
- *Program* – collection of *kernels*
- *Buffer / Image* – device memory
- *Sampler* – how to interpolate values for *Image*
- *Command Queue* – put a sequence of operations there
- *Event* – to notify that something has been done



From: OpenCL API 1.2 Reference Card

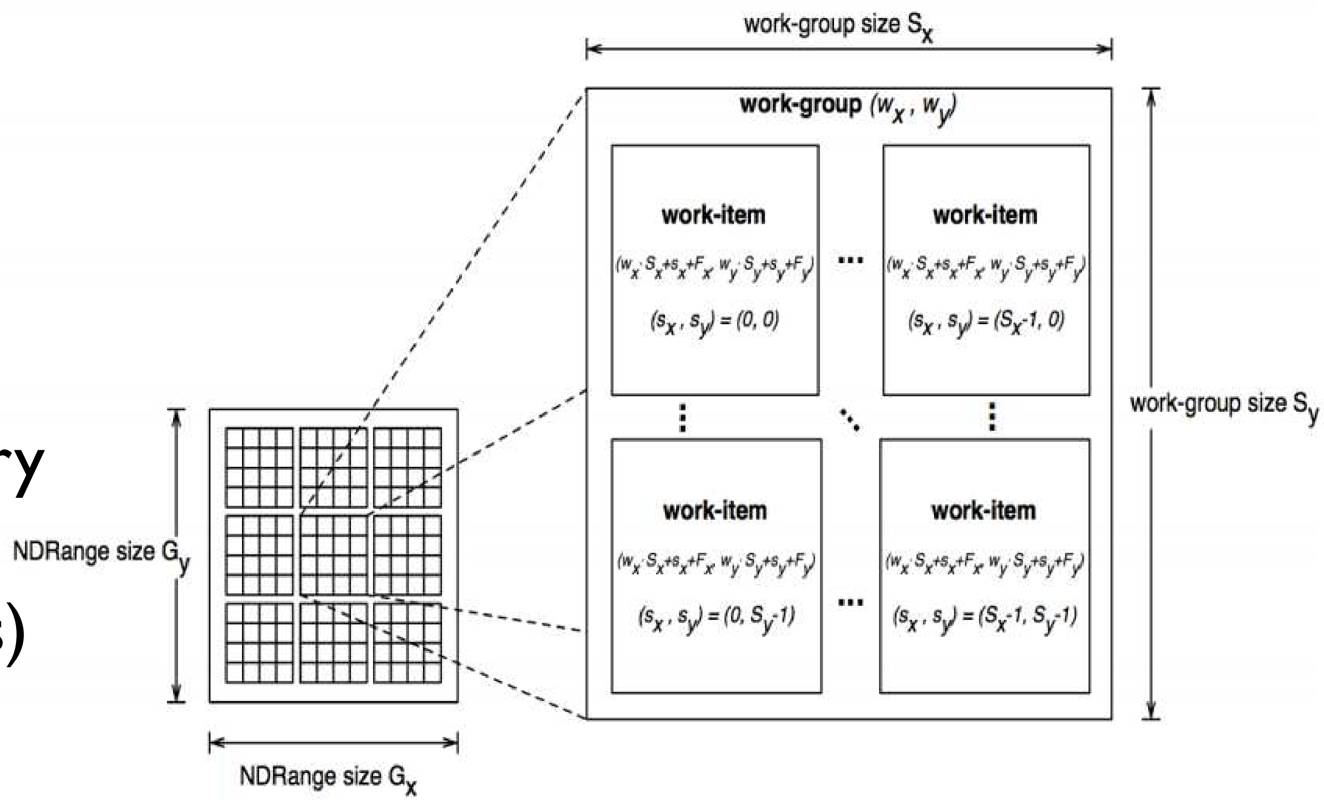
Platform model

- The host is whatever the OpenCL library runs on
  - Usually x86 CPUs for both NVIDIA and AMD
- Devices are processors that the library can talk to
  - CPUs, GPUs, DSP,s and generic accelerators
- For AMD
  - All CPUs are combined into a single device (each core is a compute unit and processing element)
  - Each GPU is a separate device



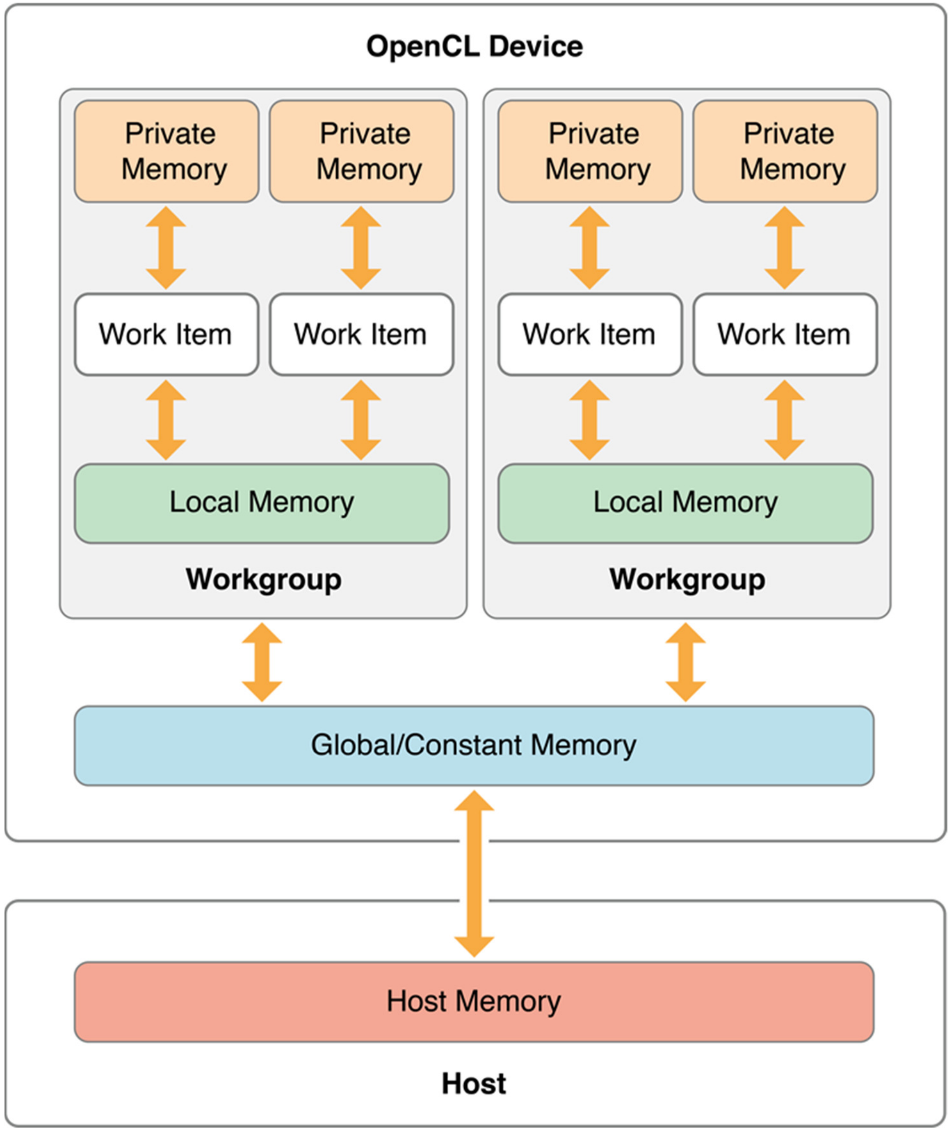
Execution model

- Each kernel executes on 1D, 2D or 3D array (NDRange)
- The array is split into work-groups
- Work items (threads) in each work-group share some local memory
- Kernel can querry
  - `get_global_id(dim)`
  - `get_group_id(dim)`
  - `get_local_id(dim)`
- Work items are not bound to any memory entity (unlike GLSL shaders)

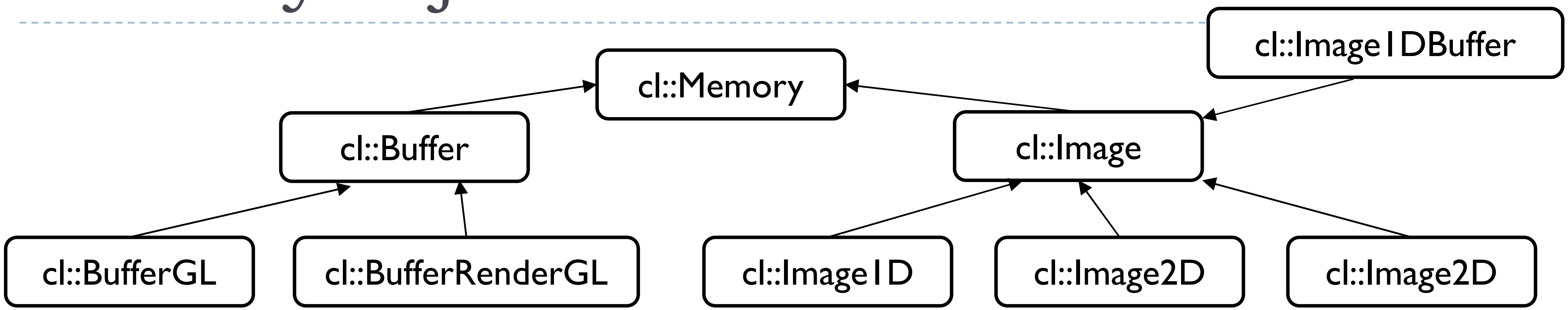


Memory model

- Host memory
  - Usually CPU memory, device does not have access to that memory
- Global memory [__*global*]
  - Device memory, for storing large data
- Constant memory [__*constant*]
- Local memory [__*local*]
  - Fast, accessible to all work-items (threads) within a workgroup
- Private memory [__*private*]
  - Accessible to a single work-item (thread)



Memory objects



This diagram is incomplete – there are more memory objects

- Buffer
  - ArrayBuffer in OpenGL
  - Accessed directly via C pointers
- Image
  - Texture in OpenGL
  - Access via texture look-up function
  - Can interpolate values, clamp, etc.

Programming model

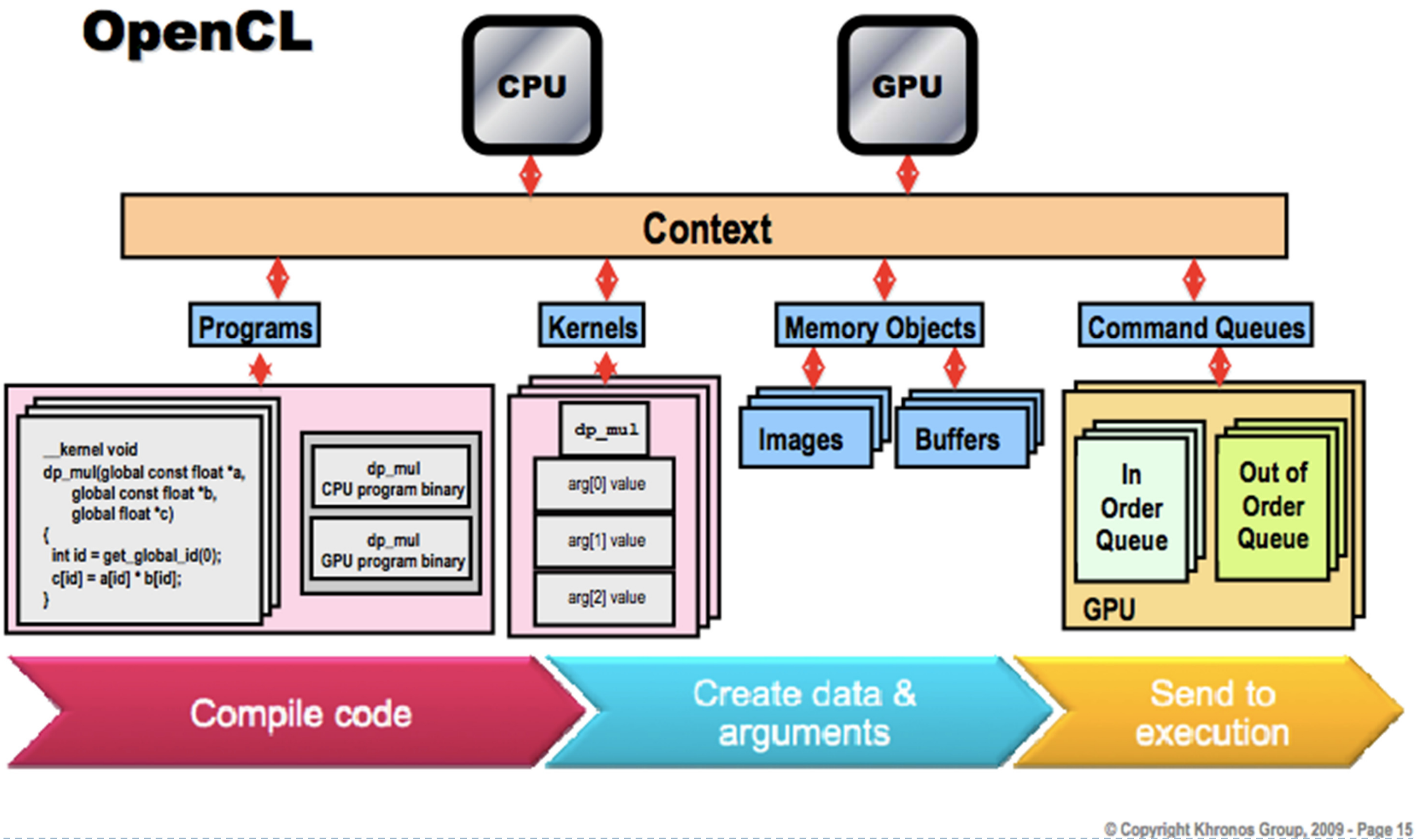
- Data parallel programming
  - Each NDRange element is assigned to a work-item (thread)
  - Each kernel can use vector-types of the device (float4, etc.)
- Task-parallel programming
  - Multiple different kernels can be executed in parallel
- Command queue

```
clCreateCommandQueue(
    cl_context context,
    cl_device_id device,
    cl_command_queue_properties properties,
    cl_int* errcode_ret)
```

> CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE
> Execute out-of-order if specified, in order otherwise

  - Provides means to both synchronize kernels and execute them in parallel

Big Picture



OpenCL
CPU
GPU
Context
Programs
Kernels
Memory Objects
Command Queues
__kernel void
dp_mul(global const float *a,
global const float *b,
global float *c)
{
int id = get_global_id(0);
c[id] = a[id] * b[id];
}
dp_mul
CPU program binary
dp_mul
GPU program binary
dp_mul
arg[0] value
arg[1] value
arg[2] value
Images
Buffers
In Order Queue
Out of Order Queue
GPU
Compile code
Create data & arguments
Send to execution
© Copyright Khronos Group, 2009 - Page 15

Advanced Graphics & Image Processing

Parallel programming in OpenCL

Part 2/3 – Thread mapping

Rafał Mantiuk

Computer Laboratory, University of Cambridge

Thread Mapping

- By using different mappings, the same thread can be assigned to access different data elements
  - The examples below show three different possible mappings of threads to data (assuming the thread id is used to access an element)

Mapping

```
int tid =
get_global_id(1) *
get_global_size(0) +
get_global_id(0);
```

```
int tid =
get_global_id(0) *
get_global_size(1) +
get_global_id(1);
```

```
int group_size =
get_local_size(0) *
get_local_size(1);

int tid =
get_group_id(1) *
get_num_groups(0) *
group_size +
get_group_id(0) *
group_size +
get_local_id(1) *
get_local_size(0) +
get_local_id(0)
```

Thread IDs

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

*assuming 2x2 groups

Thread Mapping

- Consider a serial matrix multiplication algorithm

```
for (i1=0; i1 < M; i1++)
    for (i2=0; i2 < N; i2++)
        for (i3=0; i3 < P; i3++)
            C[i1][i2] += A[i1][i3]*B[i3][i2];
```

- This algorithm is suited for output data decomposition
  - We will create *N x M* threads
    - Effectively removing the outer two loops
  - Each thread will perform *P* calculations
    - The inner loop will remain as part of the kernel
- Should the index space be MxN or NxM?

Thread Mapping

- Thread mapping 1: with an MxN index space, the kernel would be:

```
int tx = get_global_id(0);
int ty = get_global_id(1);
for(i3=0; i3<P; i3++)
    C[tx][ty] += A[tx][i3]*B[i3][ty];
```

Mapping for C

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

- Thread mapping 2: with an NxM index space, the kernel would be:

```
int tx = get_global_id (0);
int ty = get_global_id (1);
for(i3=0; i3<P; i3++)
    C[ty][tx] += A[ty][i3]*B[i3][tx];
```

Mapping for C

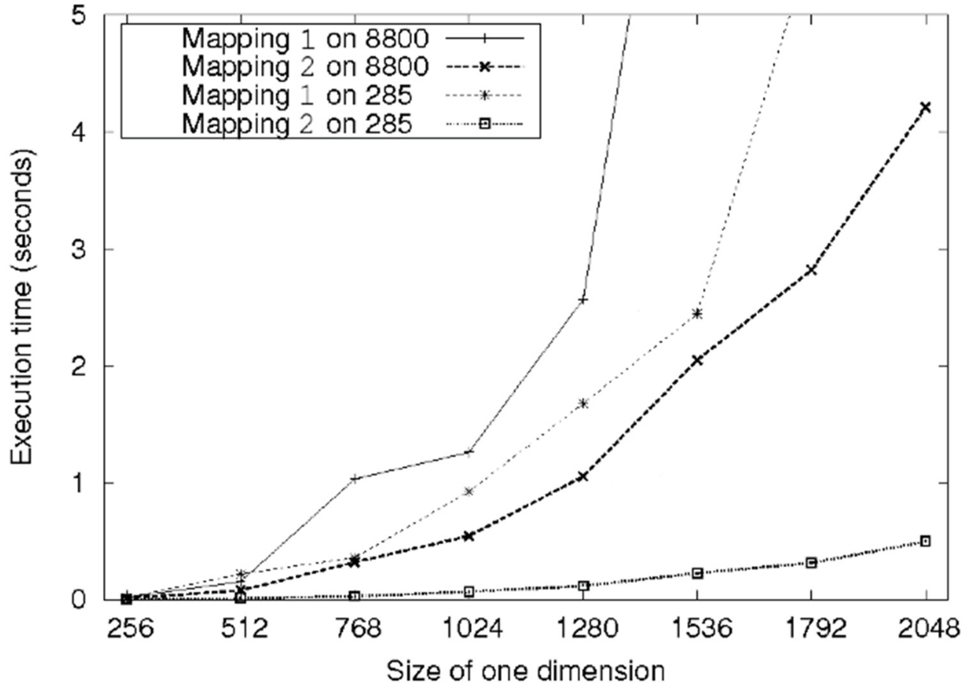
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

- Both mappings produce functionally equivalent versions of the program

From: OpenCL 1.2 University Kit - http://developer.amd.com/partners/university-programs/

Thread Mapping

- This figure shows the execution of the two thread mappings on NVIDIA GeForce 285 and 8800 GPUs



- Notice that mapping 2 is far superior in performance for both GPUs

 From: OpenCL 1.2 University Kit - http://developer.amd.com/partners/university-programs/

Thread Mapping

- The discrepancy in execution times between the mappings is due to data accesses on the global memory bus
  - Assuming row-major data, data in a row (i.e., elements in adjacent columns) are stored sequentially in memory
  - To ensure coalesced accesses, consecutive threads in the same wavefront should be mapped to columns (the second dimension) of the matrices
    - This will give coalesced accesses in Matrices B and C
    - For Matrix A, the iterator *i3* determines the access pattern for row-major data, so thread mapping does not affect it

From: OpenCL 1.2 University Kit - http://developer.amd.com/partners/university-programs/

UNIVERSITY OF CAMBRIDGE
COMPUTER LABORATORY

RAINBOW GRAPHICS GROUP

Advanced Graphics & Image Processing

Parallel programming in OpenCL

Part 3/3 – Reduction

Rafał Mantiuk

Computer Laboratory, University of Cambridge

Reduction

- GPU offers very good performance for tasks in which the results are stored independently
  - Process N data items and store in N memory location

```
float reduce_sum(float* input, int length)
{
  float accumulator = input[0];
  for(int i = 1; i < length; i++)
    accumulator += input[i];
  return accumulator;
}
```

- But many common operations require reducing N values into 1 or few values
  - sum, min, max, prod, min, histogram, …
- Those operations require an efficient implementation of reduction

- The following slides are based on AMD's OpenCL™ Optimization Case Study: Simple Reductions
  - http://developer.amd.com/resources/articles-whitepapers/opencl-optimization-case-study-simple-reductions/

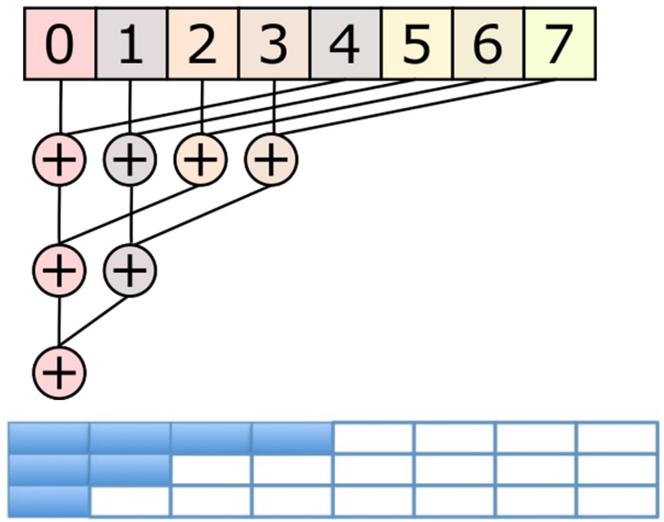
Reduction tree for the `min` operation

```
__kernel
void reduce_min(__global float* buffer,
            __local float* scratch,
            __const int length,
            __global float* result) {

  int global_index = get_global_id(0);
  int local_index = get_local_id(0);
  // Load data into local memory
  if (global_index < length) {
    scratch[local_index] = buffer[global_index];
  } else {
    scratch[local_index] = INFINITY;
  }
  barrier(CLK_LOCAL_MEM_FENCE);
  for(int offset = get_local_size(0) / 2;
      offset > 0; offset >>= 1) {
    if (local_index < offset) {
      float other = scratch[local_index + offset];
      float mine = scratch[local_index];
      scratch[local_index] = (mine < other) ? mine :
other;
    }
    barrier(CLK_LOCAL_MEM_FENCE);
  }
  if (local_index == 0) {
    result[get_group_id(0)] = scratch[0];
  }
}
```

- `barrier` ensures that all threads (work units) in the local group reach that point before execution continue
- Each iteration of the for loop computes next level of the reduction pyramid

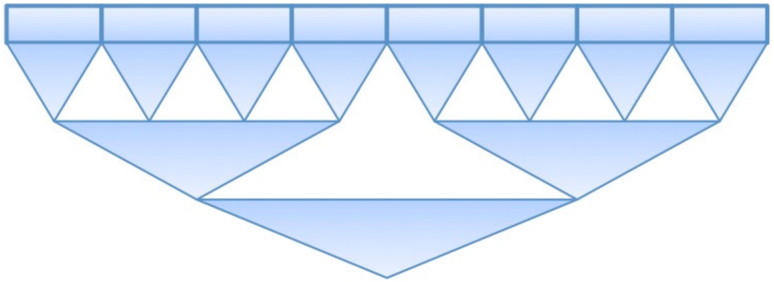
Local memory



Parallel Reduction Tree for Commutative Operator

SIMD Utilization for Reduction Tree

Multistage reduction

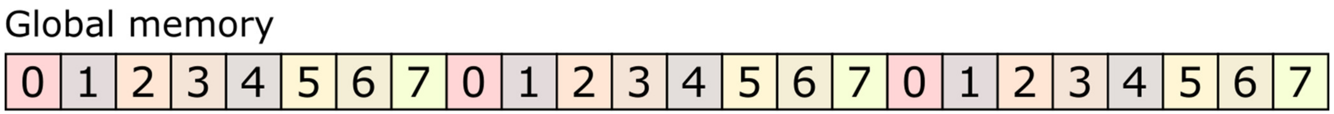


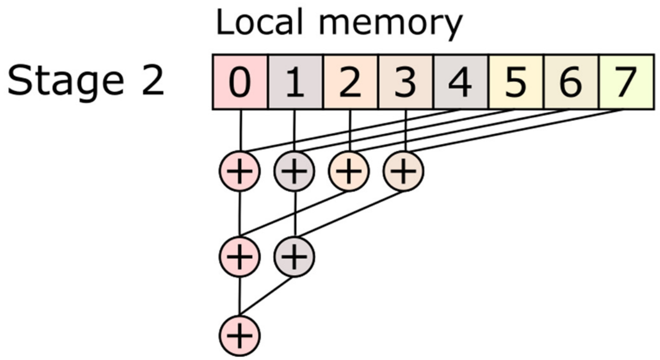
- The local memory is usually limited (e.g. 50kB), which restricts the maximum size of the array that can be processed
- Therefore, for large arrays need to be processed in multiple stages
  - The result of a local memory reduction is stored in the array and then this array is reduced

Two-stage reduction

Stage 1

Different colours denote different threads





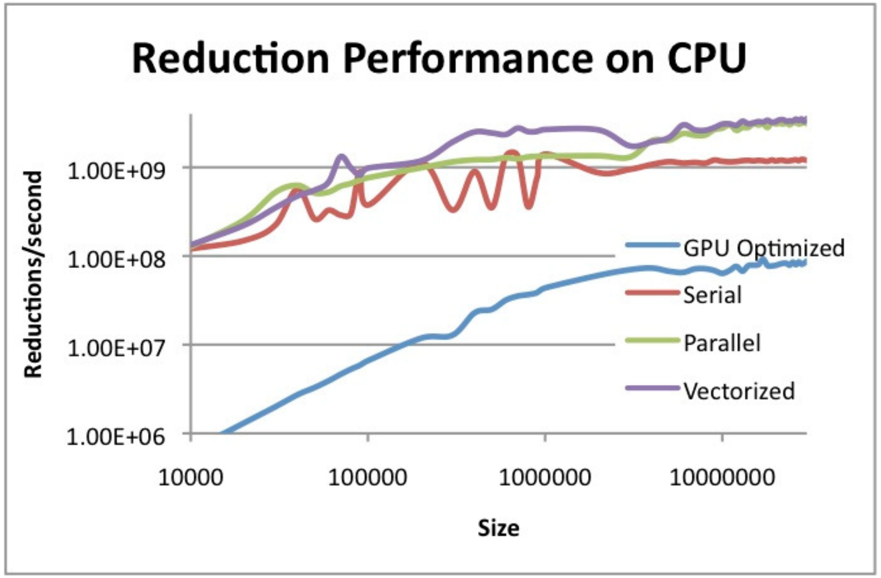
- First stage: serial reduction by N concurrent threads
  - Number of threads < data items
- Second stage: parallel reduction in local memory

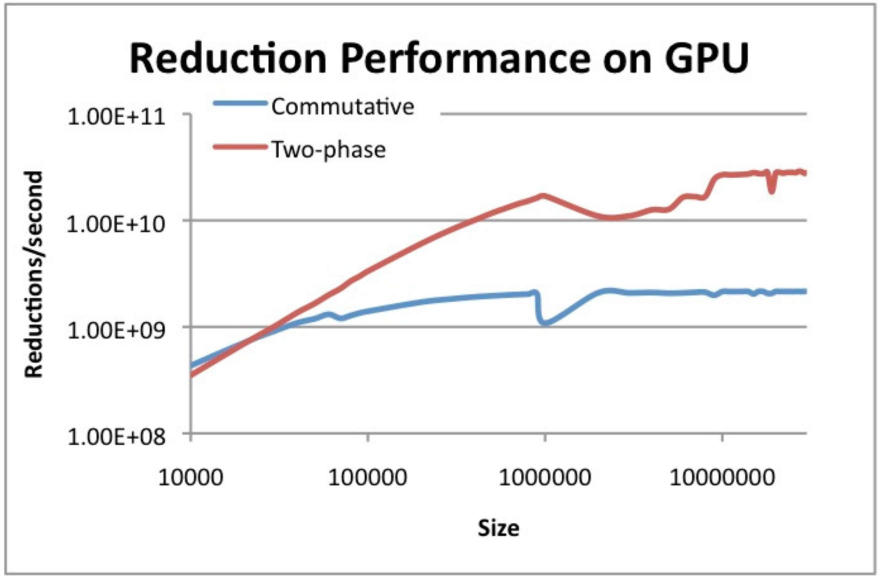
```
__kernel
void reduce(__global float* buffer,
            __local float* scratch,
            __const int length,
            __global float* result) {

  int global_index = get_global_id(0);
  float accumulator = INFINITY;
  // Loop sequentially over chunks of input
vector
  while (global_index < length) {
    float element = buffer[global_index];
    accumulator = (accumulator < element) ?
accumulator : element;
    global_index += get_global_size(0);
  }

  // Perform parallel reduction
  [The same code as in the previous example]
}
```

Reduction performance CPU/GPU



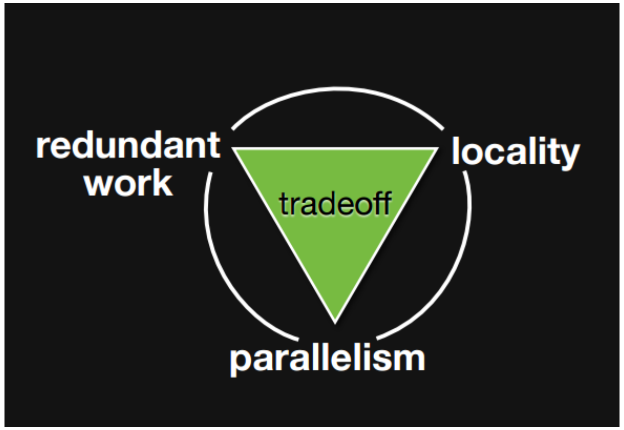


- Different reduction algorithm may be optimal for CPU and GPU
- This can also vary from one GPU to another

- The results from: http://developer.amd.com/resources/articles-whitepapers/opencl-optimization-case-study-simple-reductions/

Better way?

- **Halide - a language for image processing and computational photography**
  - http://halide-lang.org/
  - Code written in a high-level language, then translated to x86/SSE, ARM, CUDA, OpenCL
  - The optimization strategy defined separately as a *schedule*
  - Auto-tune software can test thousands of schedules and choose the one that is the best for a particular platform
  - (Semi-)automatically find the best trade-offs for a particular platform
  - Designed for image processing but similar languages created for other purposes



OpenCL resources

- https://www.khronos.org/registry/OpenCL/
- Reference cards
  - Google: "OpenCL API Reference Card"
- AMD OpenCL Programming Guide
  - http://developer.amd.com/wordpress/media/2013/07/AMD_Accelerated_Parallel_Processing_OCL_Programming_Guide-2013-06-21.pdf

Advanced Graphics & Image Processing

Image-based rendering and Light fields

Part 1/4 – context, definition and technology

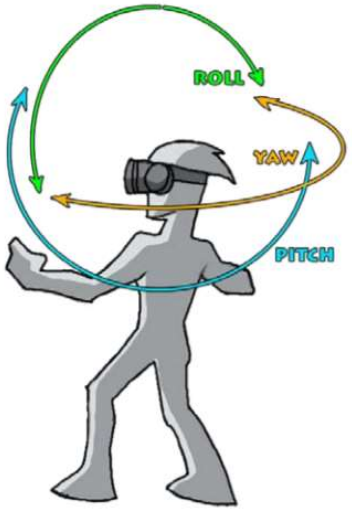
Rafał Mantiuk

Computer Laboratory, University of Cambridge

Motivation: 3DoF vs 6DoF in VR

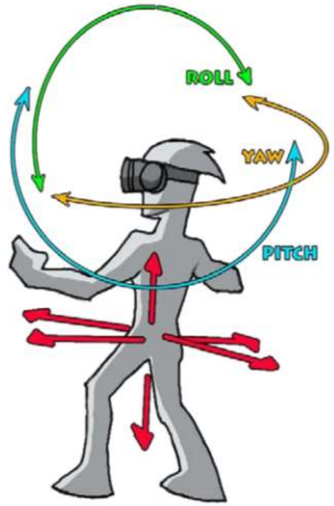
3DoF

- Tracking with inexpensive Inertial Measurements Units
- Content:
  - Geometry-based graphics
  - Omnidirectional stereo video
    - May induce cyber-sickness due due to the lack of motion depth cues



6DoF

- Requires internal (inside-out) or external tracking
- Content:
  - Geometry-based graphics
  - Point-cloud rendering
  - Image-based rendering
    - View interpolation
    - **Light fields**
    - …



Source for the images: https://www.qualcomm.com/media/documents/files/on-device-motion-tracking-for-immersive-vr.pdf

3D computer graphics

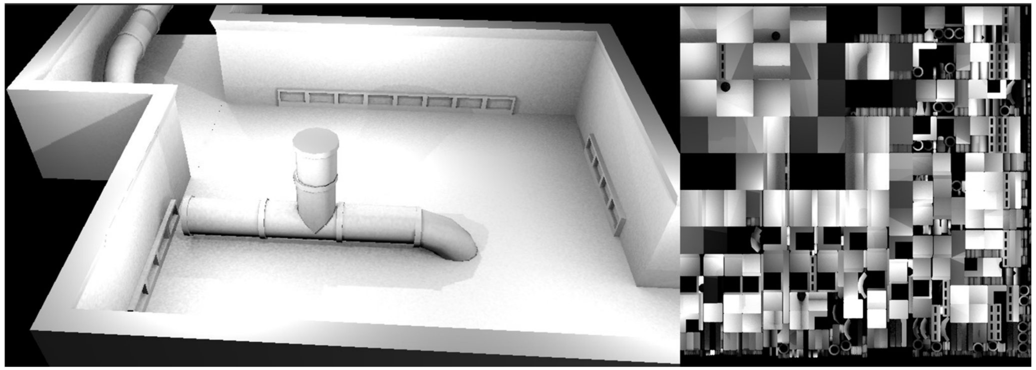
- We need:
  - Geometry + materials + textures
  - Lights
  - Camera
- Full control of illumination, realistic material appearance
- Graphics assets are expensive to create
- Rendering is expensive
  - Shading tends to takes most of the computation

Cyberpunk 2077 (C) 2020 by CD Projekt RED

Baked / precomputed illumination

- We need:
  - Geometry + textures + (light maps)
  - Camera
- No need to scan/model materials
- Much faster rendering – simplified shading

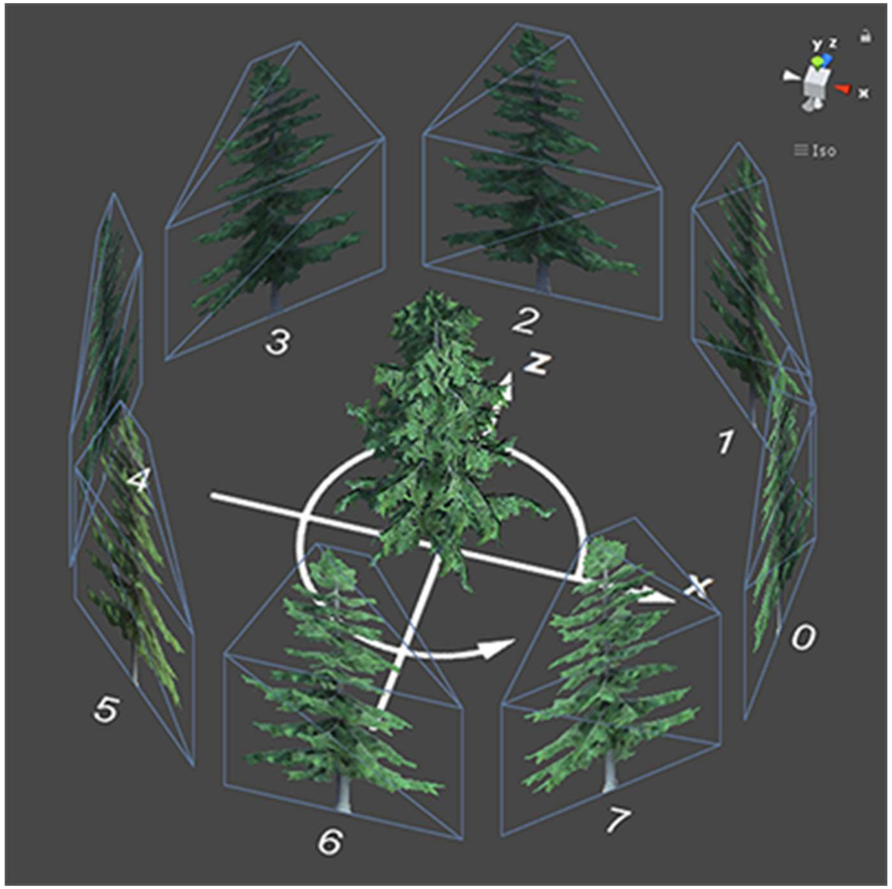
Google Earth



Precomputed light maps (from Wikipedia)

Billboards / Sprites

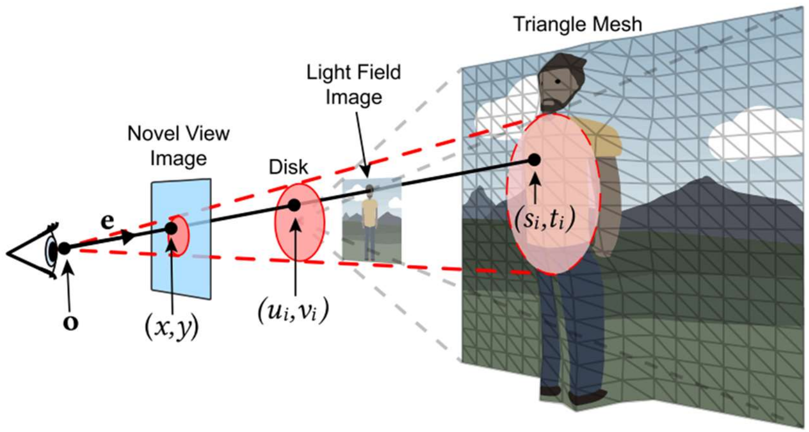
- We need:
  - Simplified geometry + textures (with alpha)
  - Lights
  - Camera
- Much faster to render than objects with 1000s of triangles
- Used for distant objects
  - or a small rendering budget
- Can be pre-computed from complex geometry



A tree rendered from a set of billboards
From:
https://docs.unity3d.com/ScriptReference/BillboardAsset.html

Light fields + depth

- We need:
  - Depth map
  - Images of the object/scene
  - Camera
- We can use camera-captured images
- View-dependent shading
- Depth-map can be computed using multi-view stereo techniques
  - CV methods can be unreliable
- No relighting



A depth map is approximated by triangle mesh and rasterized. From: Overbeck et al. TOG 2018, https://doi.org/10.1145/3272127.3275031.

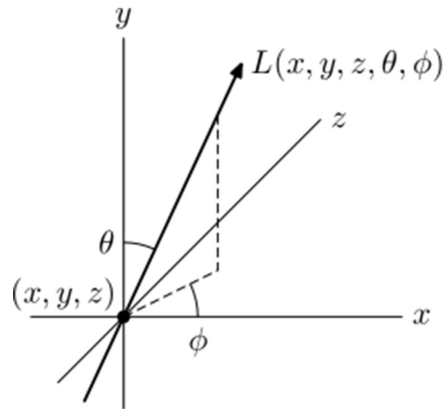
Demo: https://augmentedperception.github.io/welcome-to-lightfields/

Light fields

- We need:
  - Images of the scene
    - Or a microlens image
  - Camera
- As light fields +depth but
  - No geometry, no need for any 3D reconstruction
  - Photographs are rep-projected on the plane
  - Requires massive number of images for good quality

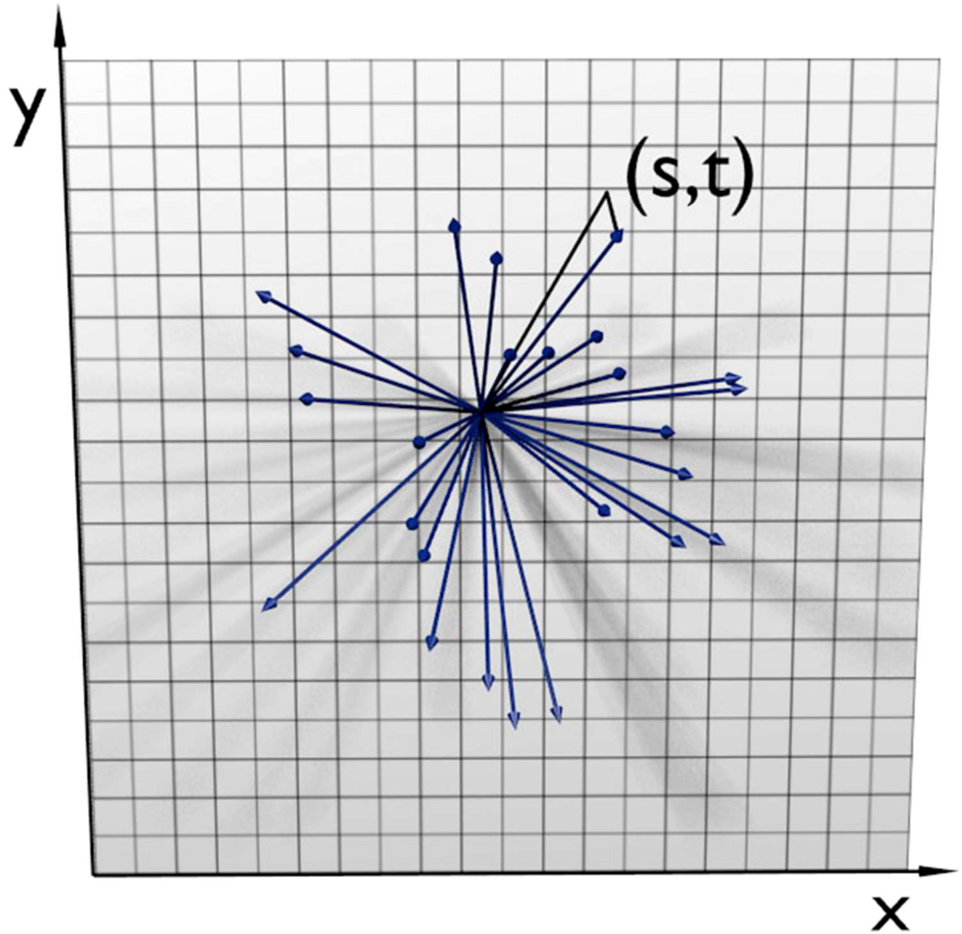
From a plenoptic function to a light field

- Plenoptic function – describes all possible rays in a 3D space
  - Function of position (x, y, z) and ray direction (θ, ϕ)
  - But also wavelength λ and time t
  - Between 5 and 7 dimensions

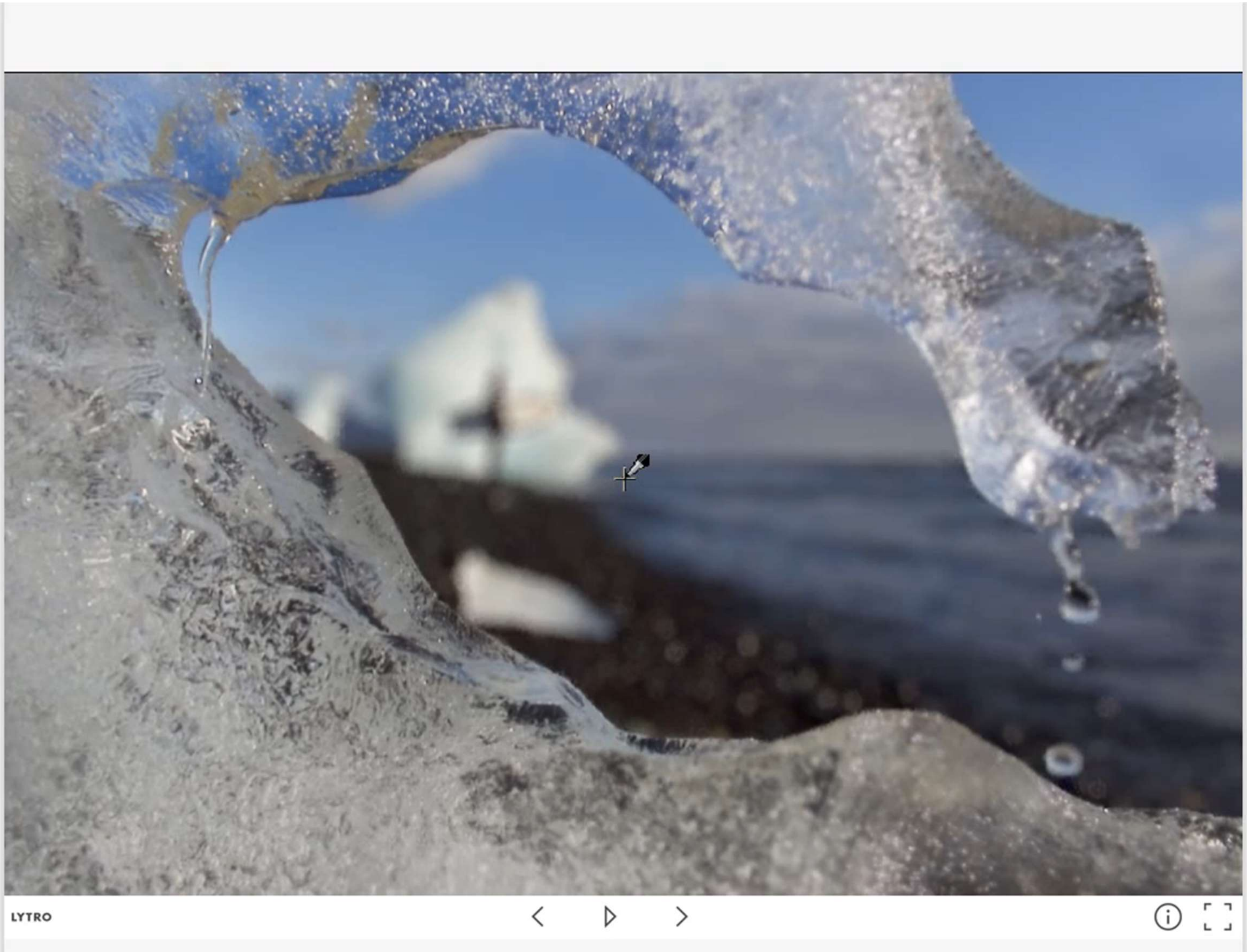


- But the number of dimensions can be reduced if
  - The camera stays outside the convex hull of the object
  - The light travels in uniform medium
  - Then, radiance L remains the same along the ray (until the ray hits an object)
  - This way we obtain a **4D light field** or **lumigraph**

Planar 4D light field



Refocusing and view point adjustment



Screen capture from http://www.lytro.com/

Depth estimation from light field

- Passive sensing of depth
- Light field captures multiple depth cues
  - Correspondance (disparity) between the views
  - Defocus
  - Occlusions

Central view | Reconstructed depth

From: ***Ting-Chun Wang, Alexei A. Efros, Ravi Ramamoorthi***; The IEEE International Conference on Computer Vision (ICCV), 2015, pp. 3487-3495

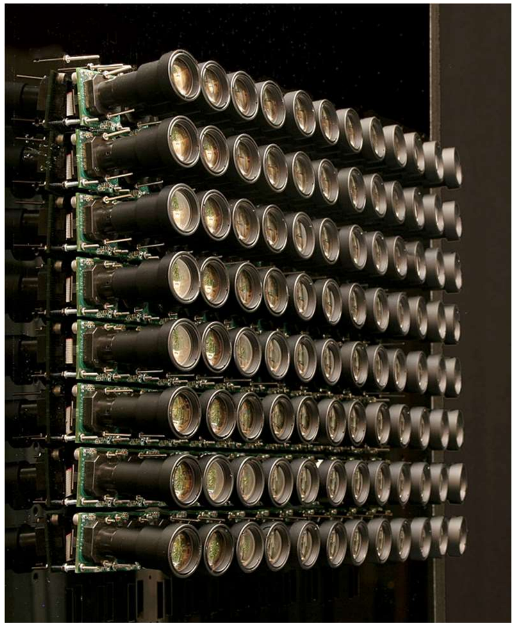
Two methods to capture light fields

Micro-lens array

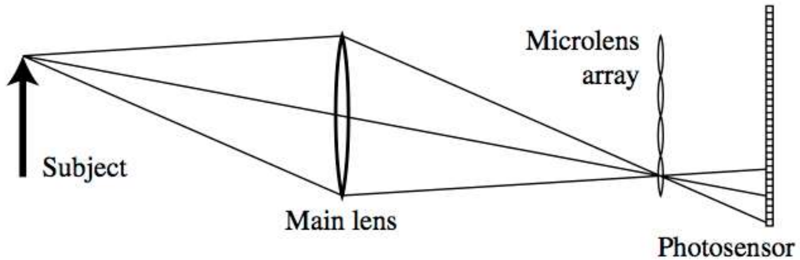
- Small baseline
- Good for digital refocusing
- Limited resolution

Camera array

- Large baseline
- High resolution
- Rendering often requires approximate depth

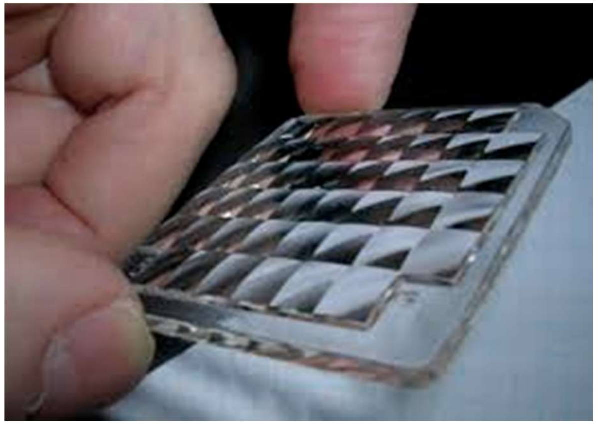


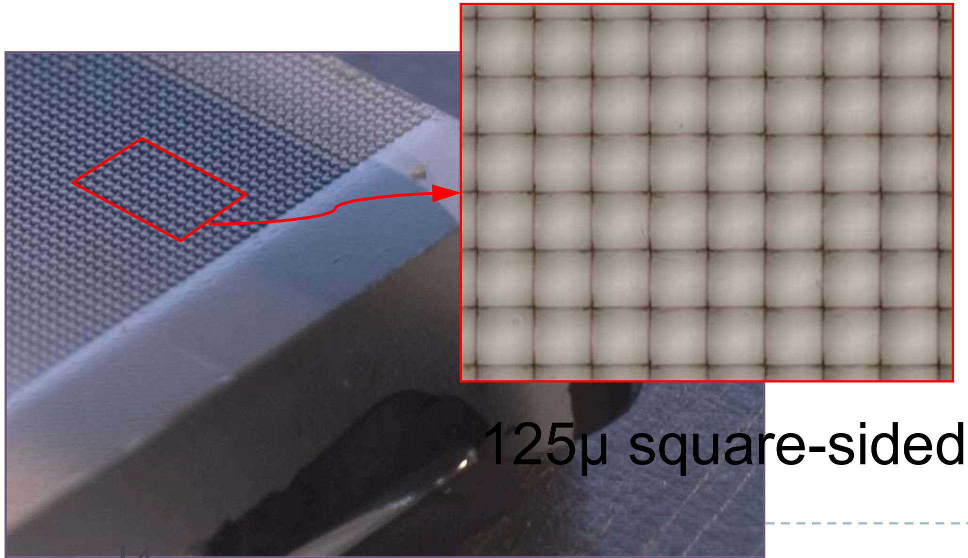
Light field image – with microlens array



Digital Refocusing using Light Field Camera

Lenslet array





125μ square-sided microlenses

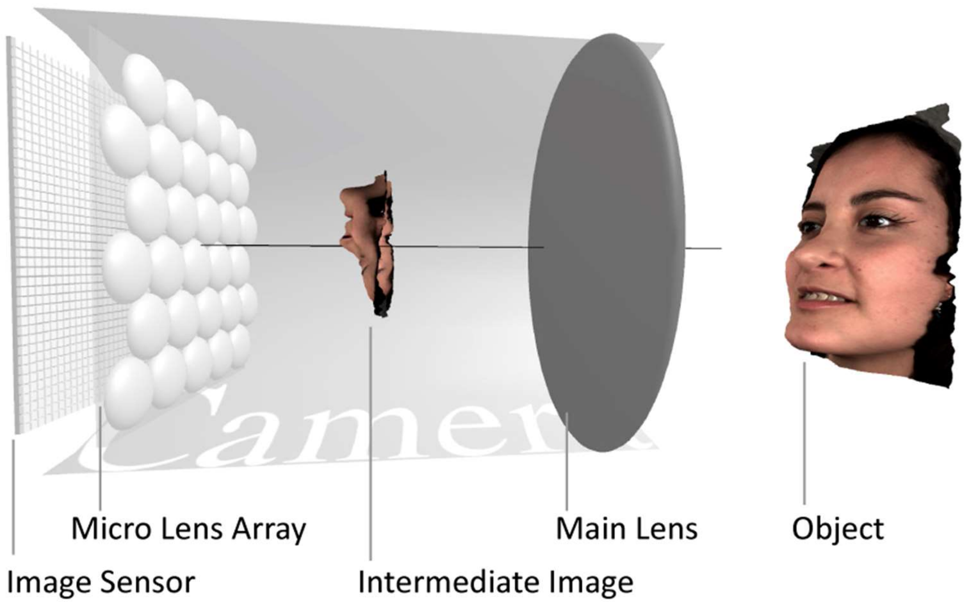
[Ng et al 2005]

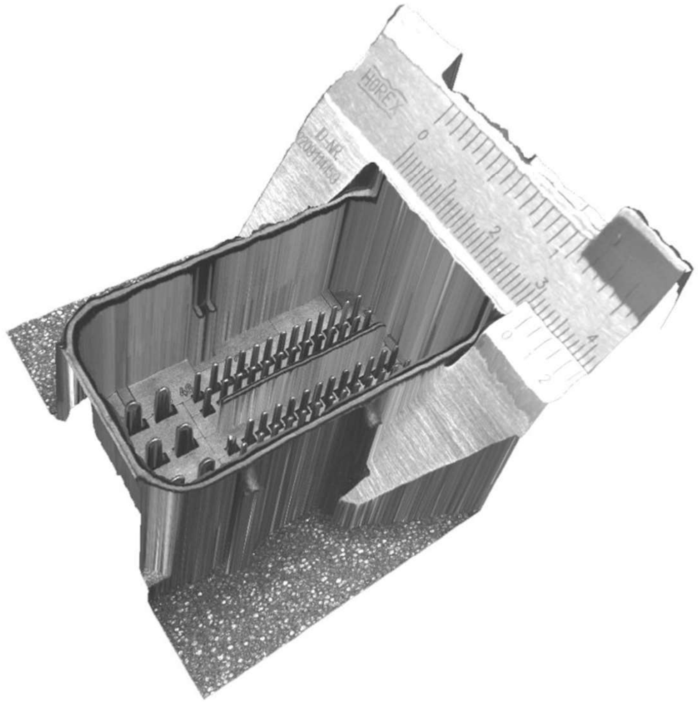
Lytro-cameras

- First commercial light-field cameras
- Lytro illum camera
  - 40 Mega-rays
  - 2D resolution: 2450 x 1634 (4 MPixels)

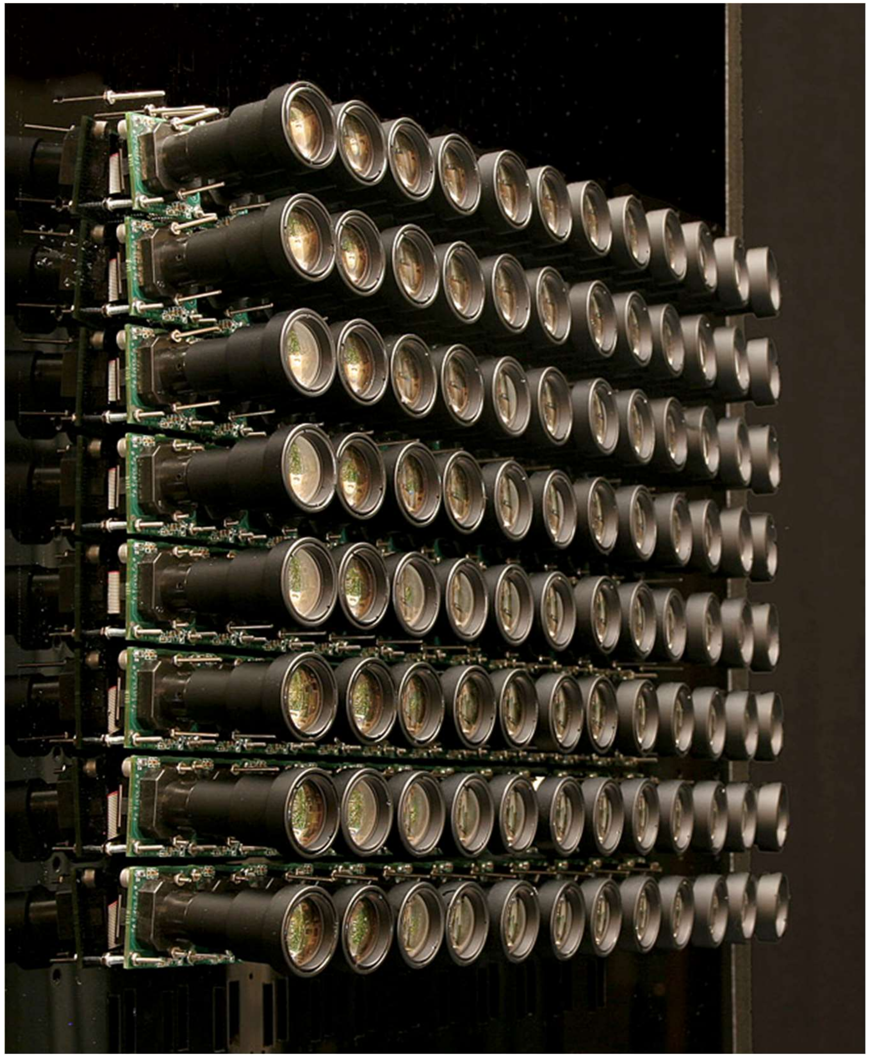
Raytrix camera

- Similar technology to Lytro
- But profiled for computer vision applications





Stanford camera array

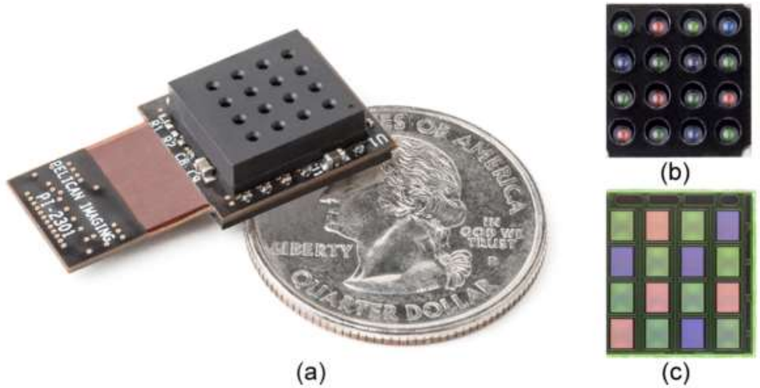


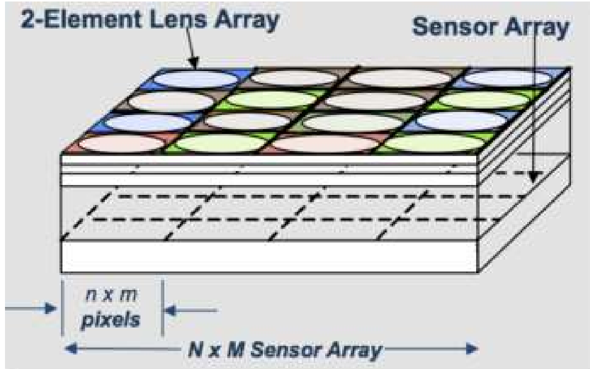
96 cameras

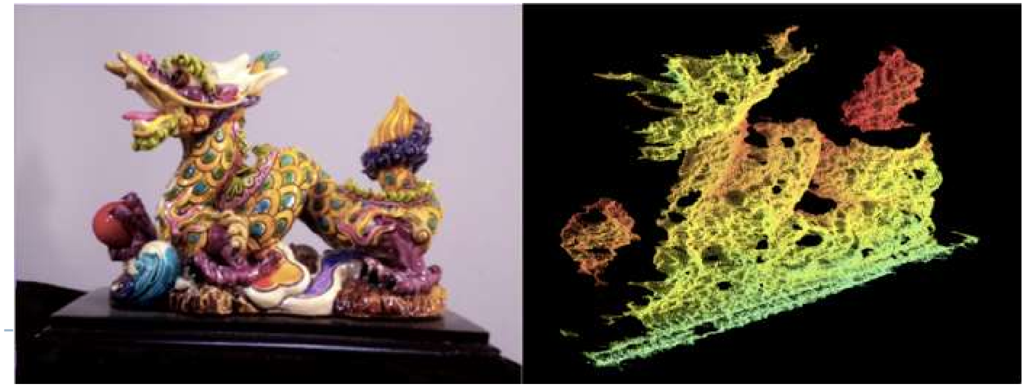
Application: Reconstruction of occluded surfaces

PiCam camera array module

- Array of 4 x 4 cameras on a single chip
- Each camera has its own lens and senses only one spectral colour band
  - Optics can be optimized for that band
- The algorithm needs to reconstruct depth







UNIVERSITY OF CAMBRIDGE
COMPUTER LABORATORY

RAINBOW GRAPHICS GROUP

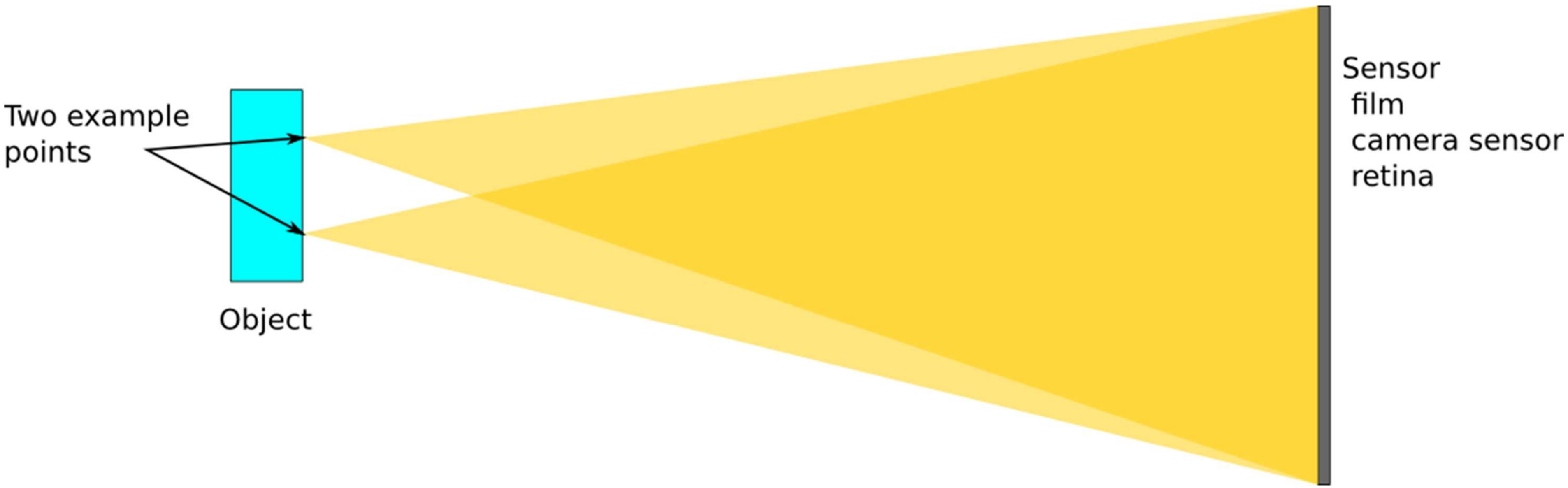
Advanced Graphics & Image Processing

Light fields

Part 2/4 – imaging and lens

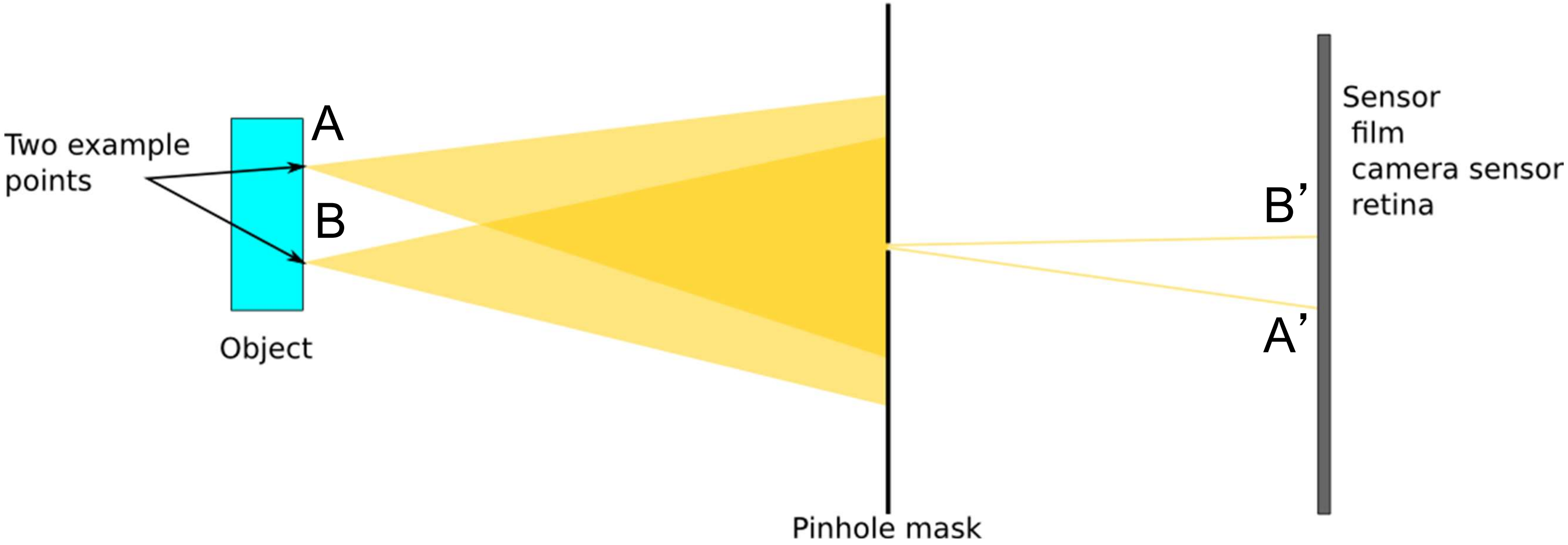
Rafał Mantiuk
Computer Laboratory, University of Cambridge

Imaging – without lens



Every point in the scene illuminates every point (pixel) on a sensor. Everything overlaps - no useful image.

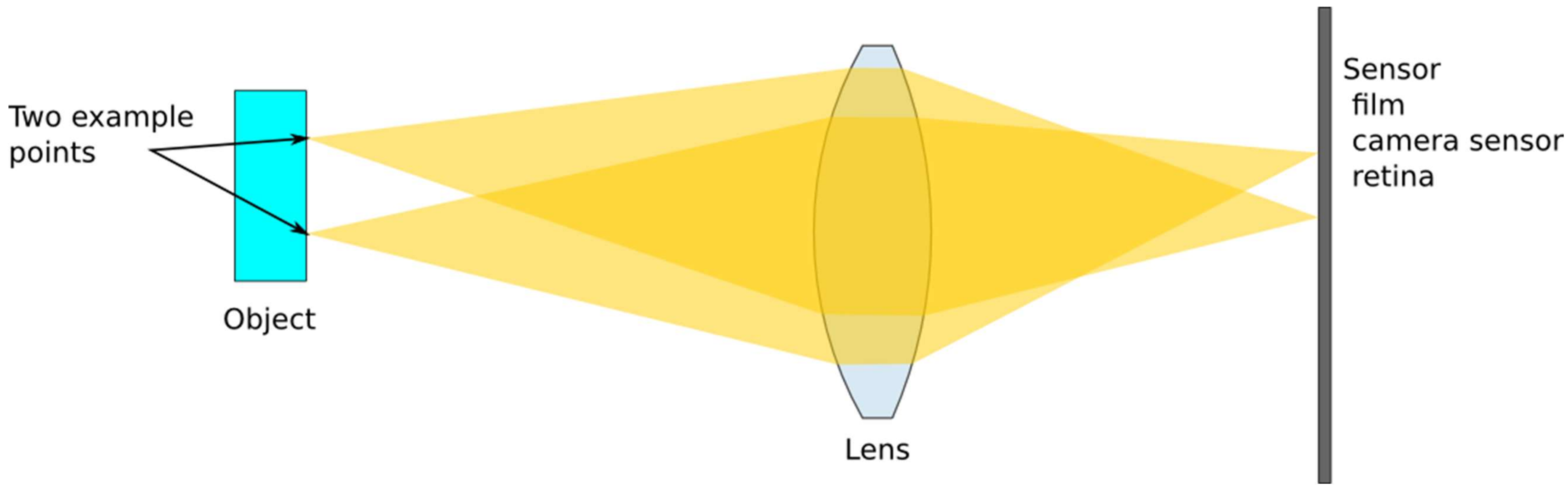
Imaging – pinhole camera



Pinhole masks all but only tiny beams of light. The light from different points is separated and the image is formed.

But very little light reaches the sensor.

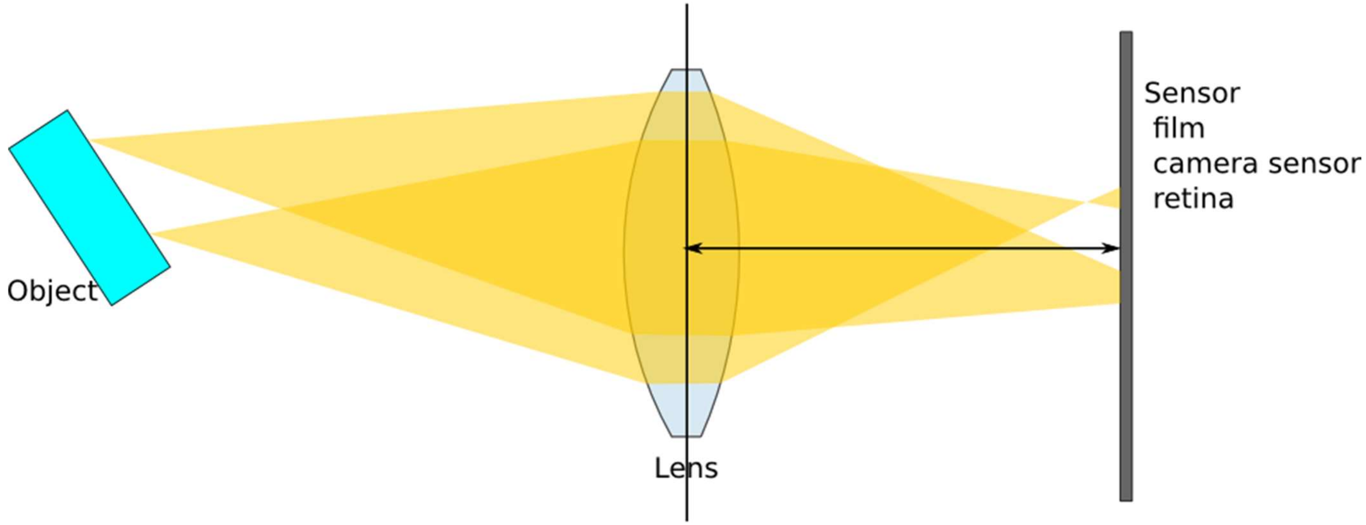
Imaging – lens



Lens can focus a beam of light on a sensor (focal plane).

Much more light-efficient than the pinhole.

Imaging – lens

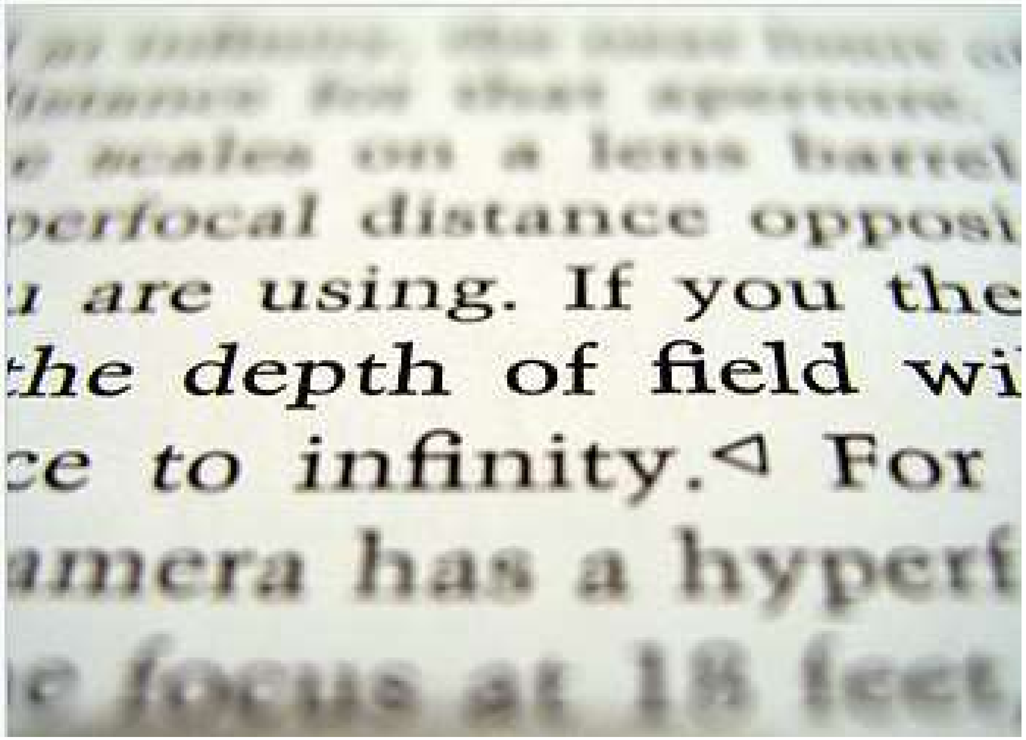


But it the light beams coming from different distances are not focused on the same plane.
These points will appear blurry in the resulting image.

Camera needs to move lens to focus an image on the sensor.

Depth of field

- Depth of field – range of depths that provides sufficient focus

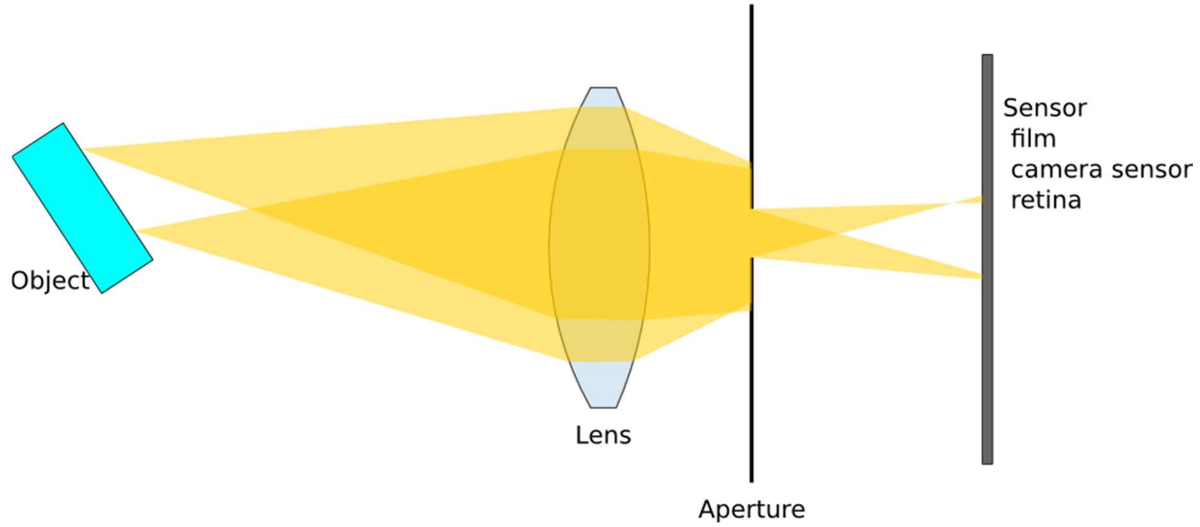


Defocus blur is often desirable

To separate the object of interest from background

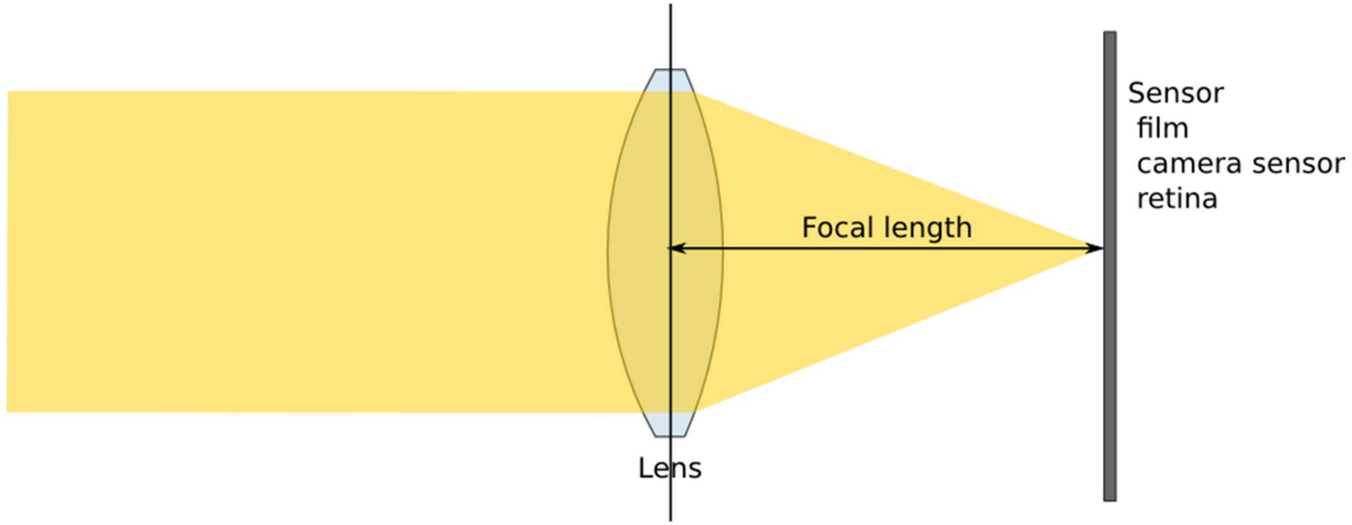
Defocus blur is a strong depth cue

Imaging – aperture



Aperture (introduced behind the lens) reduces the amount of light reaching sensor, but it also reduces blurriness from defocus (increases depth-of-field).

Imaging – lens



Focal length – length between the sensor and the lens that is needed to focus light coming from an infinite distance.

Larger focal length of a lens – more or less magnification?

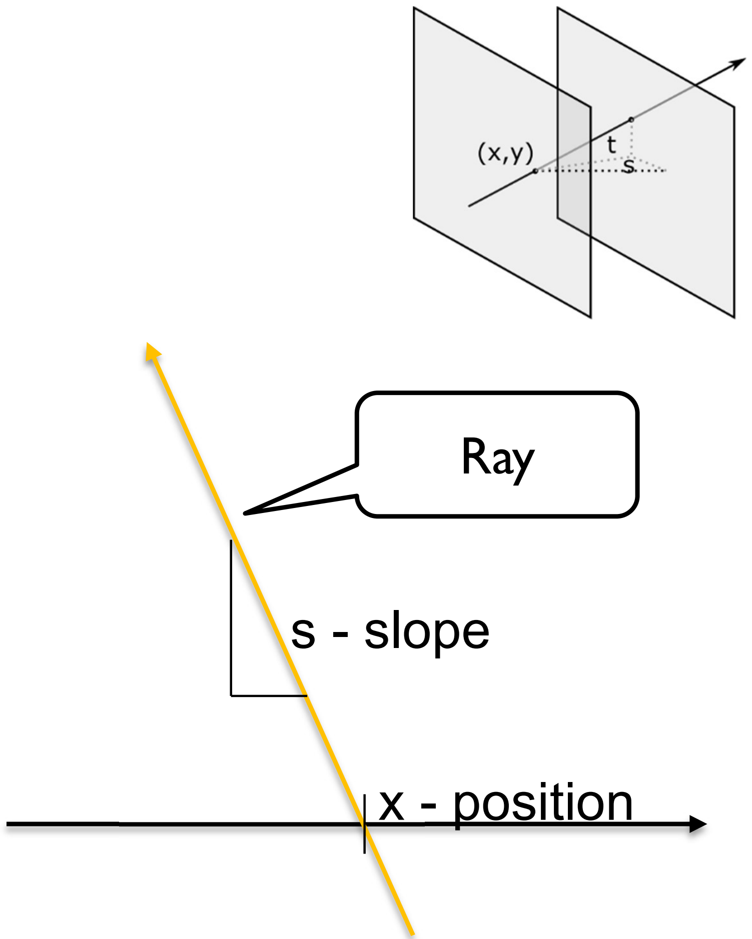
Advanced Graphics & Image Processing

Light fields

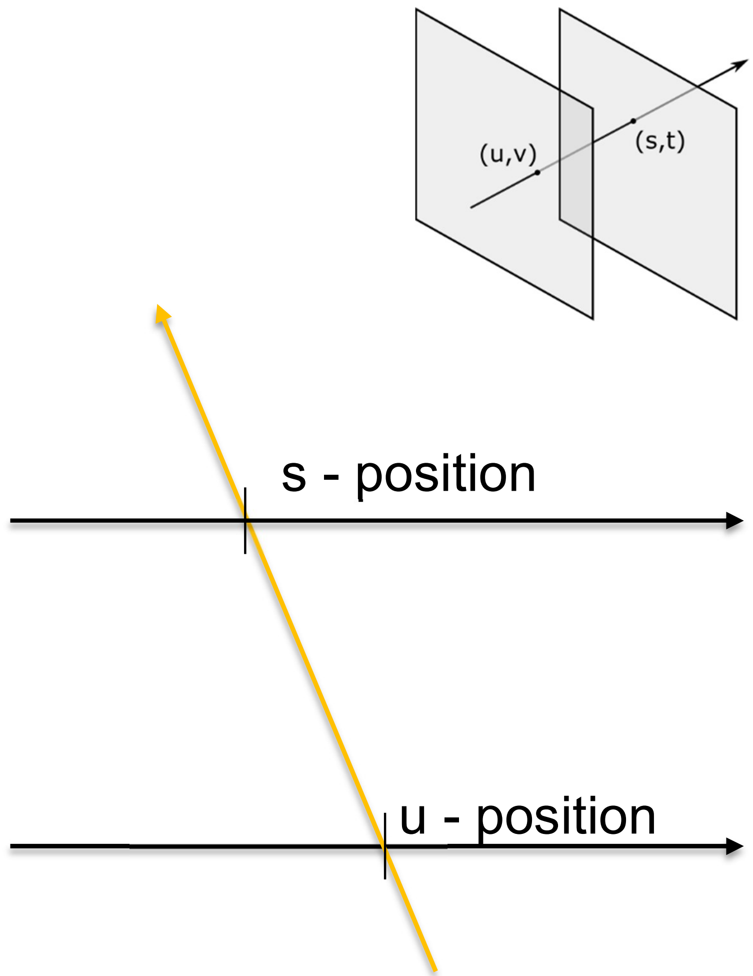
Part 3/4 – parametrization and an example

Rafał Mantiuk
Computer Laboratory, University of Cambridge

Light fields: two parametrisations (shown in 2D)

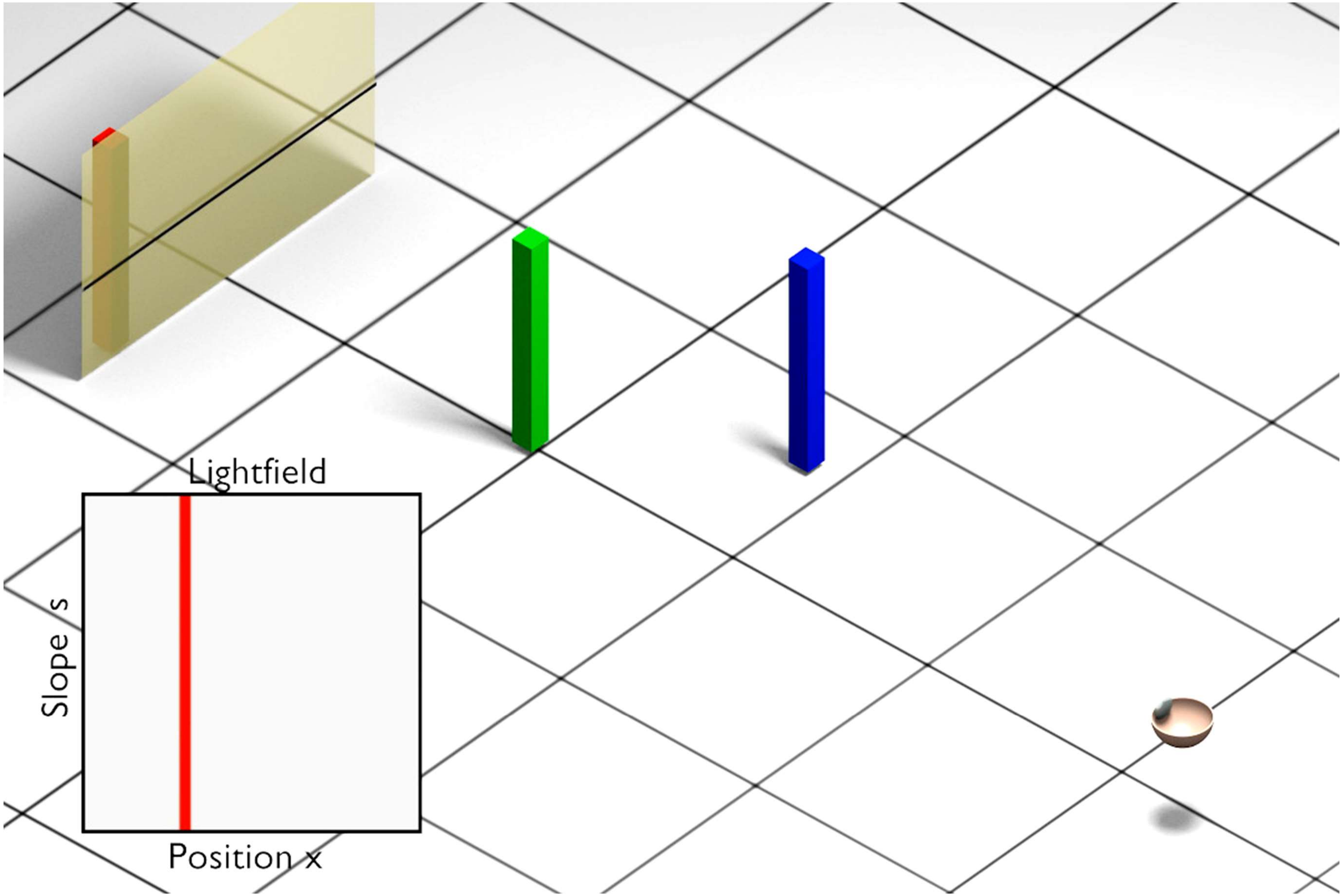


Position and slope
(slope - tangent of the angle)

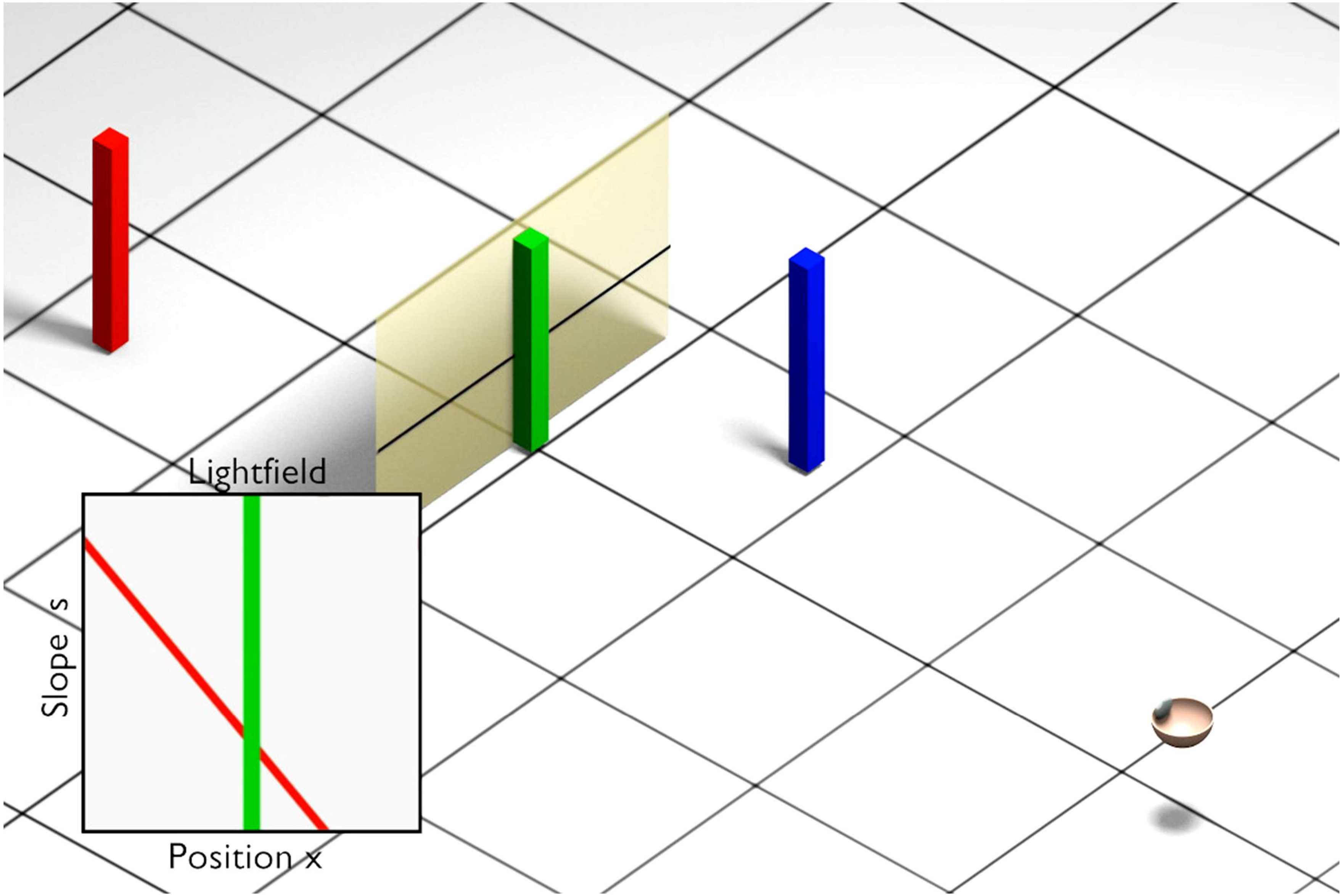


Two planes

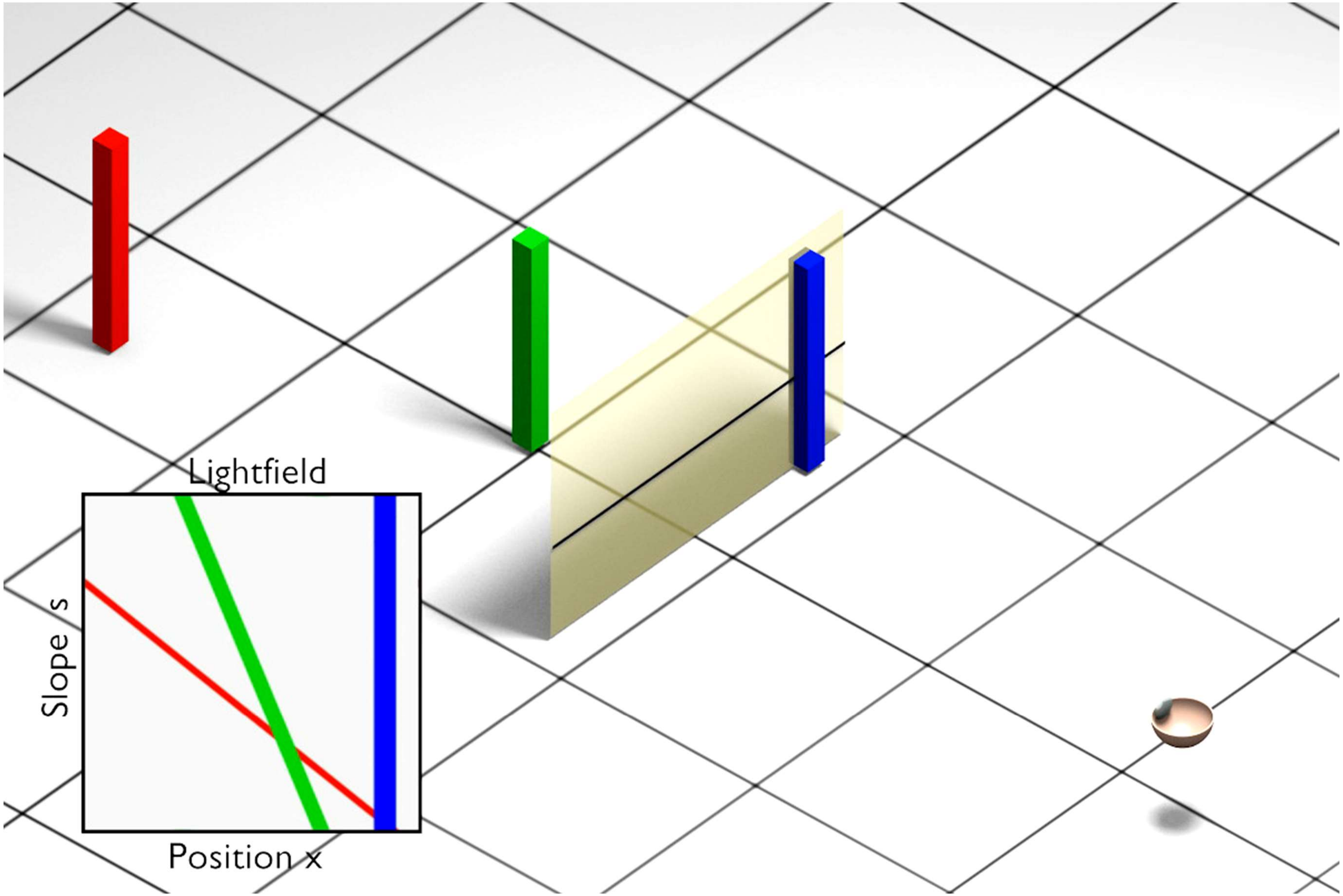
Lightfield - example



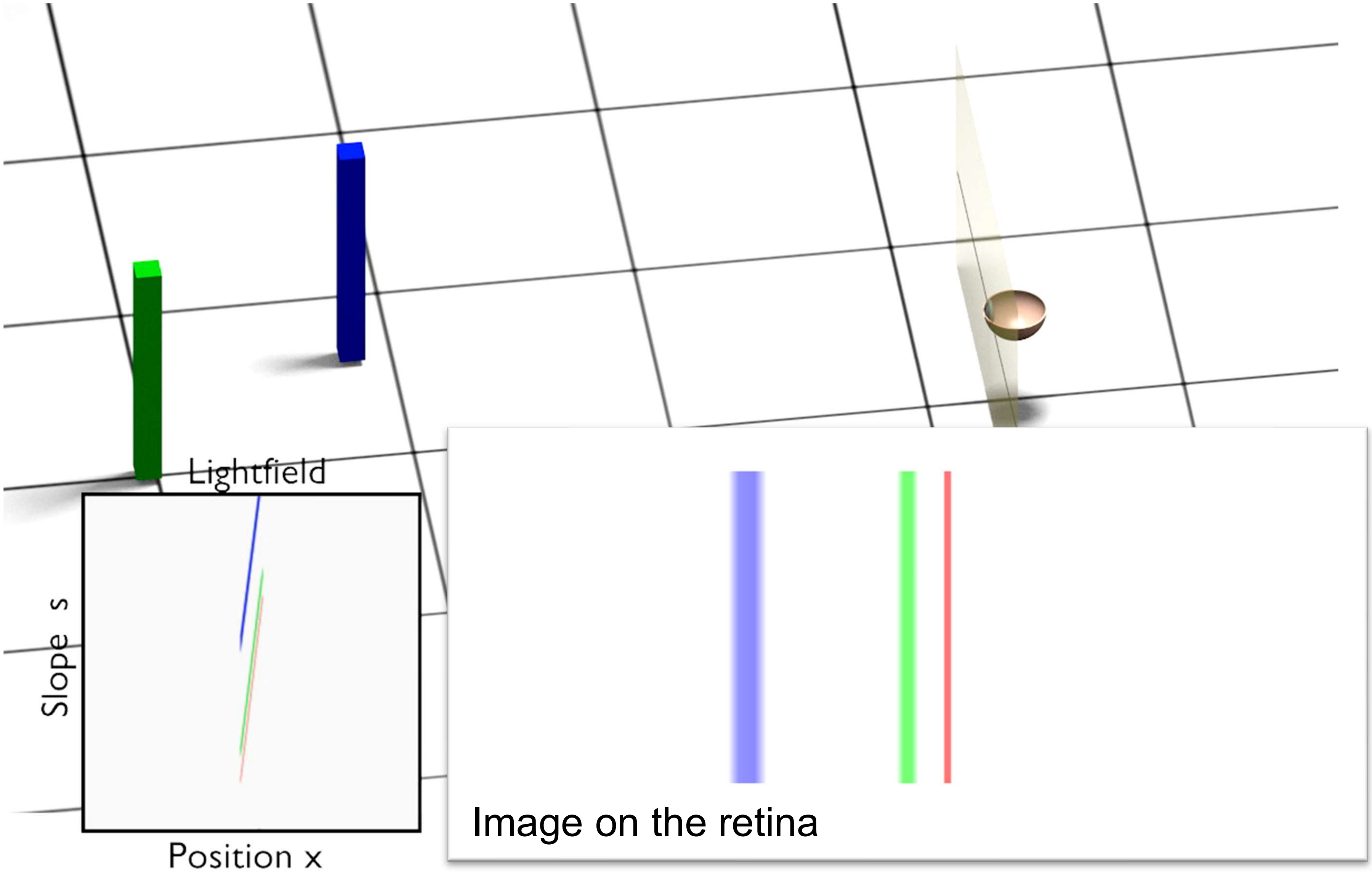
Lightfield - example



Lightfield - example



Lightfield - example



UNIVERSITY OF CAMBRIDGE
COMPUTER LABORATORY

RAINBOW GRAPHICS GROUP

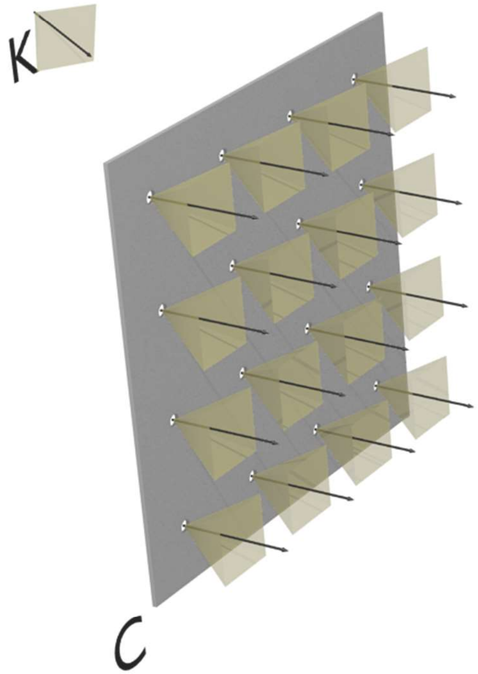
Advanced Graphics & Image Processing

Light fields

Part 4/4 – light field rendering

Rafał Mantiuk
Computer Laboratory, University of Cambridge

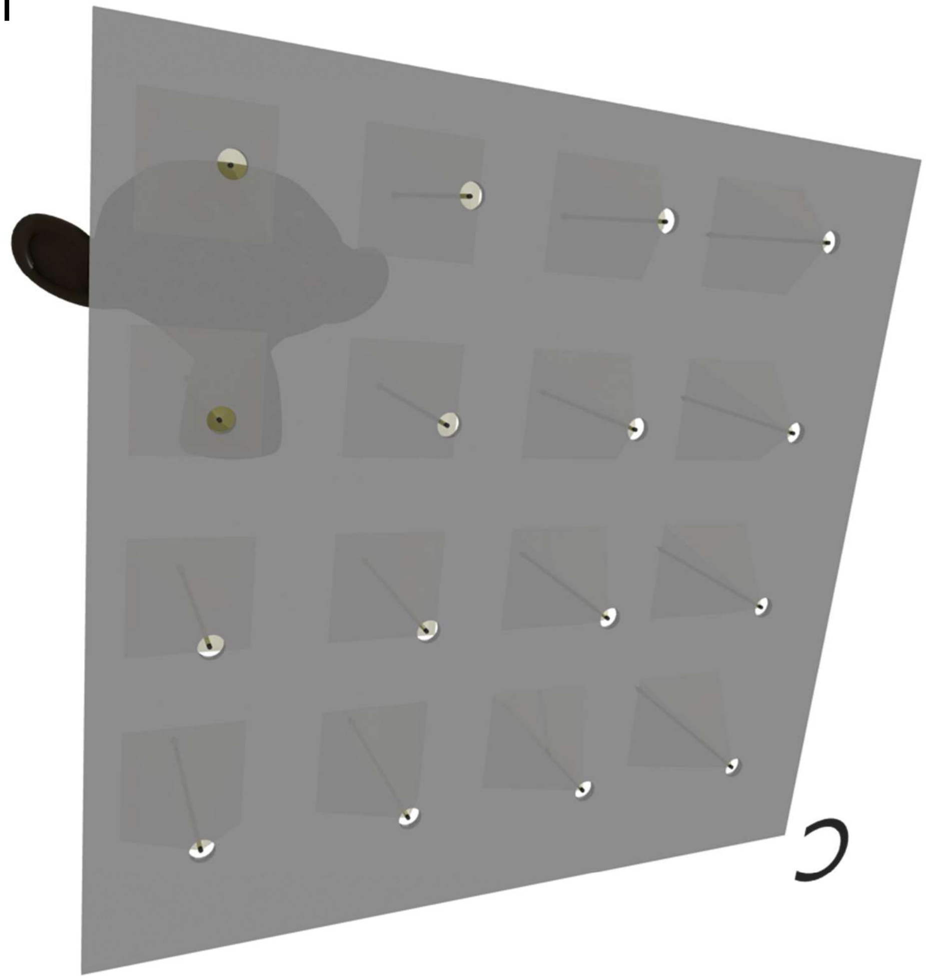
Light field rendering (1/3)



We want to render a scene (Blender monkey) as seen by camera K. We have a light field captured by a camera array. Each camera in the array has its aperture on plane C.

Light field rendering (2/3)

From the view point of camera K

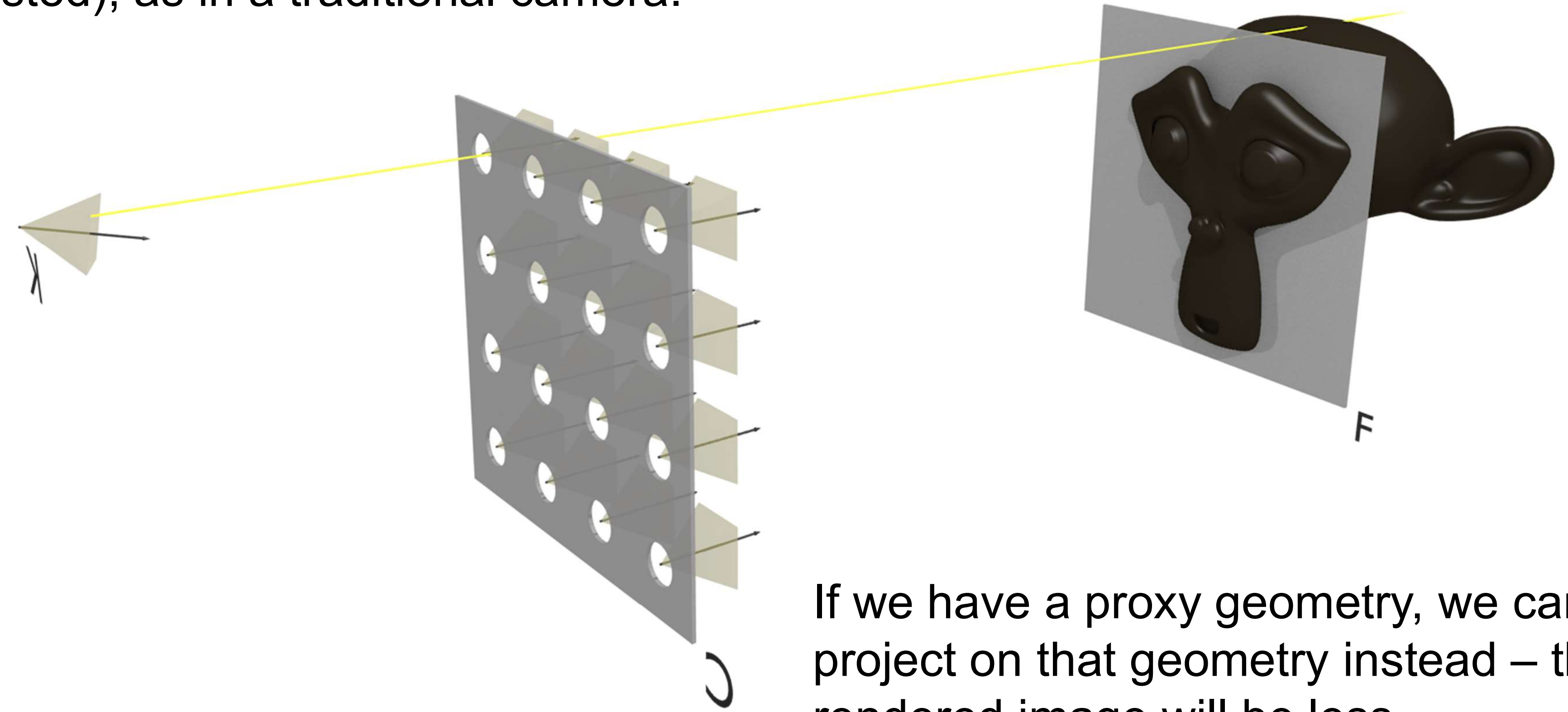


Each camera in the array provides accurate light measurements only for the rays originating from its pinhole aperture.

The missing rays can be either interpolated (reconstructed) or ignored.

Light field rendering (3/3)

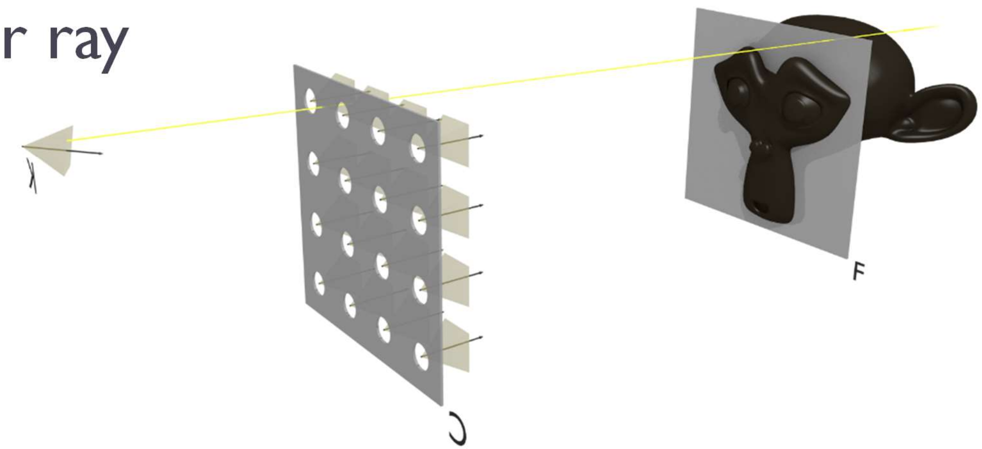
The rays from the camera need to be projected on the focal plane F. The objects on the focal plane will be sharp, and the objects in front or behind that plane will be blurry (ghosted), as in a traditional camera.



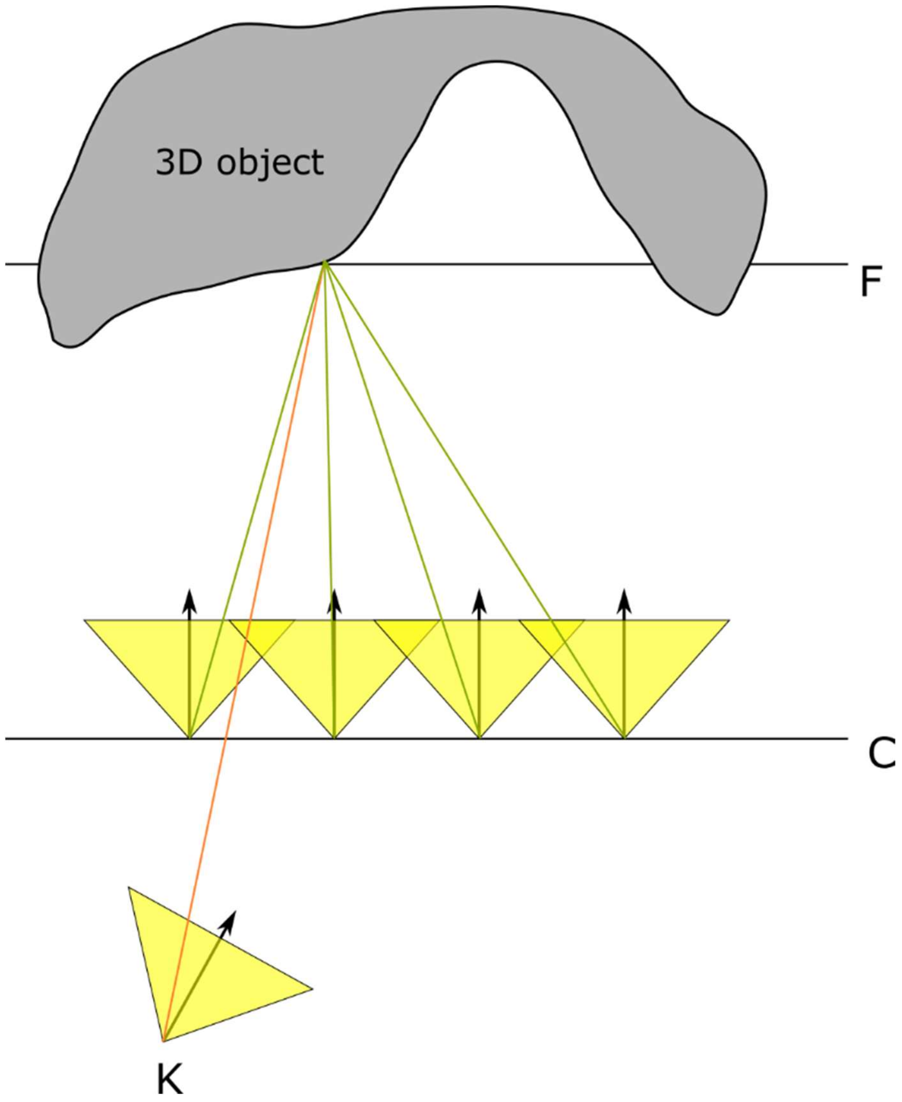
If we have a proxy geometry, we can project on that geometry instead – the rendered image will be less ghosted/blurry

Intuition behind light field rendering

- For large virtual aperture (use all cameras in the array)
  - Each camera in the array captures the scene
  - Then, each camera projects its image on the focal plane F
  - The virual camera K captures the projection
- For small virtual aperture (pinhole)
  - For each ray from the virtual camera
    - interpolate rays from 4 nearest camera images
  - Or use the nearest-neighbour ray

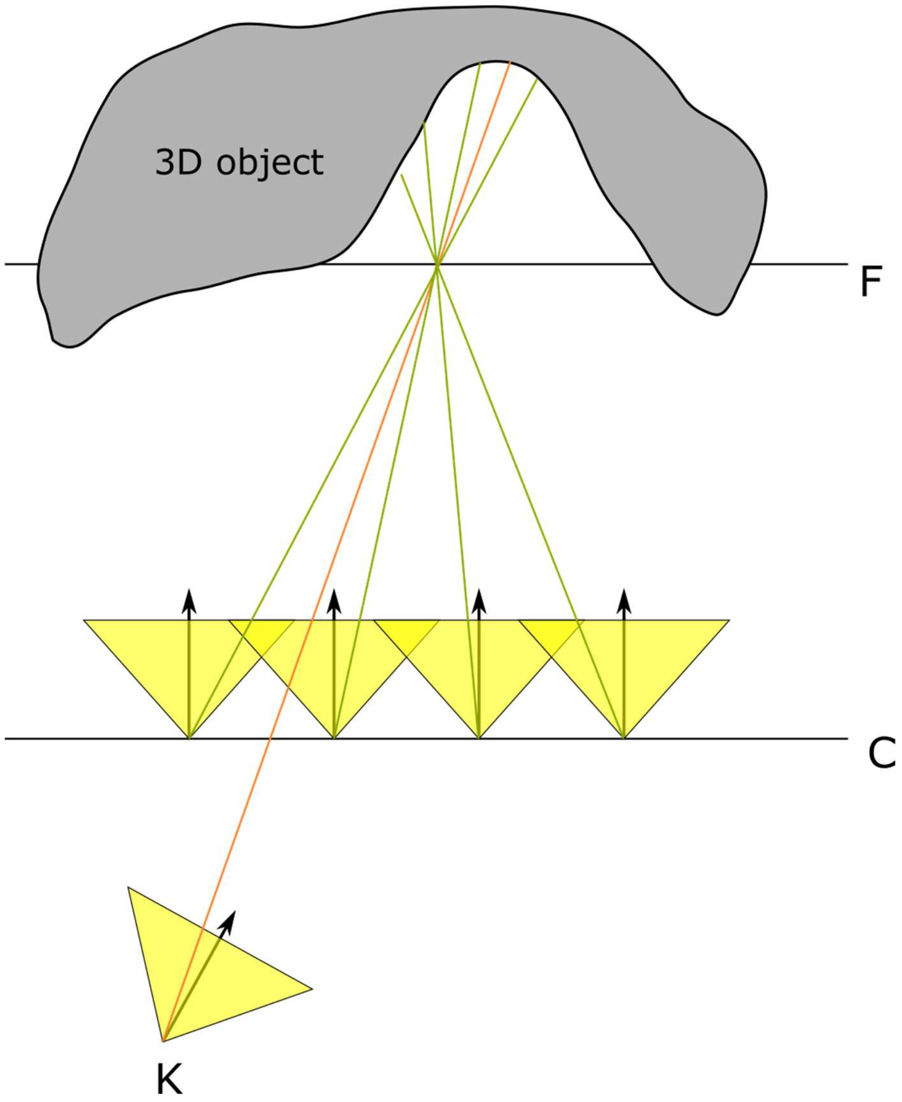


LF rendering – focal plane



- For a point on the focal plane, all cameras capture the same point on the 3D object
- They also capture approximately the same colour (for diffuse objects)
- Averaged colour will be the colour of the point on the surface

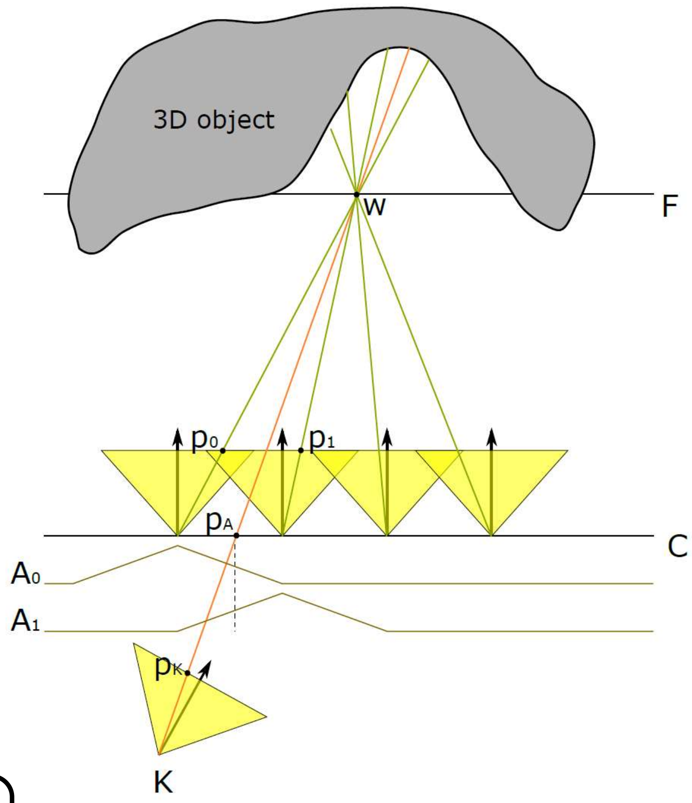
LF rendering – focal plane



- If the 3D object does not lie on the focal plane, all camaras capture different points on the object
- Averaging colour values will produce a „ghosted" image
- If we had unlimited number of cameras, this would produce a depth-of-field effect

Finding homographic transformation 1/3

- For the pixel coordinates $\boldsymbol{p}_k$ of the virtual camera K, we want to find the corresponding coordinates $\boldsymbol{p}_i$ in the camera array image
- Given the world 3D coordinates of a point $\boldsymbol{w}$:



$$\boldsymbol{p}_i = \boldsymbol{K}\,\boldsymbol{P}\,\boldsymbol{V}_i\,\boldsymbol{w}$$

Intrinsic camera matrix

Projection matrix

View matrix

$$\begin{bmatrix} x_i \\ y_i \\ w_i \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} \\ v_{21} & v_{22} & v_{23} & v_{24} \\ v_{31} & v_{32} & v_{33} & v_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Finding homographic transformation 2/3

- A homography between two views is usually found as:

$$\boldsymbol{p}_K = \boldsymbol{K}_K \boldsymbol{P} \boldsymbol{V}_K \boldsymbol{w}$$
$$\boldsymbol{p}_i = \boldsymbol{K}_i \boldsymbol{P} \boldsymbol{V}_i \boldsymbol{w}$$

hence

$$\boldsymbol{p}_i = \boldsymbol{K}_i \boldsymbol{P} \boldsymbol{V}_i \boldsymbol{V}_K^{-1} \boldsymbol{P}^{-1} \boldsymbol{K}_K^{-1} \boldsymbol{p}_K$$

- But, $\boldsymbol{K}_K \boldsymbol{P} \boldsymbol{V}_K$ is not a square matrix and cannot be inverted
  - To find the correspondence, we need to constrain 3D coordinates $\boldsymbol{w}$ to lie on the plane:

$$\boldsymbol{N} \cdot (\boldsymbol{w} - \boldsymbol{w}_F) = 0 \qquad \text{or} \qquad d = \begin{bmatrix} n_x & n_y & n_z & -\boldsymbol{N} \cdot \boldsymbol{w}_F \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Finding homographic … n 3/3

The plane in
the camera coordinates
(not world coordinates)

- Then, we add the plane equation to the projection matrix

$$\underbrace{\begin{bmatrix} x_i \\ y_i \\ d_i \\ w_i \end{bmatrix}}_{\hat{\boldsymbol{p}}_i} = \underbrace{\begin{bmatrix} f_x & 0 & 0 & c_x \\ 0 & f_y & 0 & c_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\hat{\boldsymbol{K}}_i} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ n_x^{(c)} & n_y^{(c)} & n_z^{(c)} & -\boldsymbol{N}^{(\mathbf{c})} \cdot \boldsymbol{w}_F^{(c)} \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\hat{\boldsymbol{P}}} \underbrace{\begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} \\ v_{21} & v_{22} & v_{23} & v_{24} \\ v_{31} & v_{32} & v_{33} & v_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{V}_i} \underbrace{\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}}_{\boldsymbol{w}}$$

- Where d_i is the distance to the plane (set to 0)
- Hence

$$\hat{\boldsymbol{p}_i} = \hat{\boldsymbol{K}}_i \hat{\boldsymbol{P}} \boldsymbol{V}_i \boldsymbol{V}_K^{-1} \hat{\boldsymbol{P}}^{-1} \hat{\boldsymbol{K}}_K^{-1} \hat{\boldsymbol{p}_K}$$

References

- Light fields
  - Micro-lens array

    - Ng, Ren and Levoy, Marc and Bredif, M. and D., & Gene and Horowitz, Mark and Hanrahan, P. (2005). *Light field photography with a hand-held plenoptic camera*.

  - Camera array

    - OVERBECK, R.S., ERICKSON, D., EVANGELAKOS, D., PHARR, M., AND DEBEVEC, P. 2018. A system for acquiring, processing, and rendering panoramic light field stills for virtual reality. *ACM Transactions on Graphics 37*, 6, 1–15.
    - ISAKSEN, A., MCMILLAN, L., AND GORTLER, S.J. 2000. Dynamically reparameterized light fields. *Proc of SIGGRAPH '00*, ACM Press, 297–306.

UNIVERSITY OF CAMBRIDGE
COMPUTER LABORATORY

RAINBOW GRAPHICS GROUP

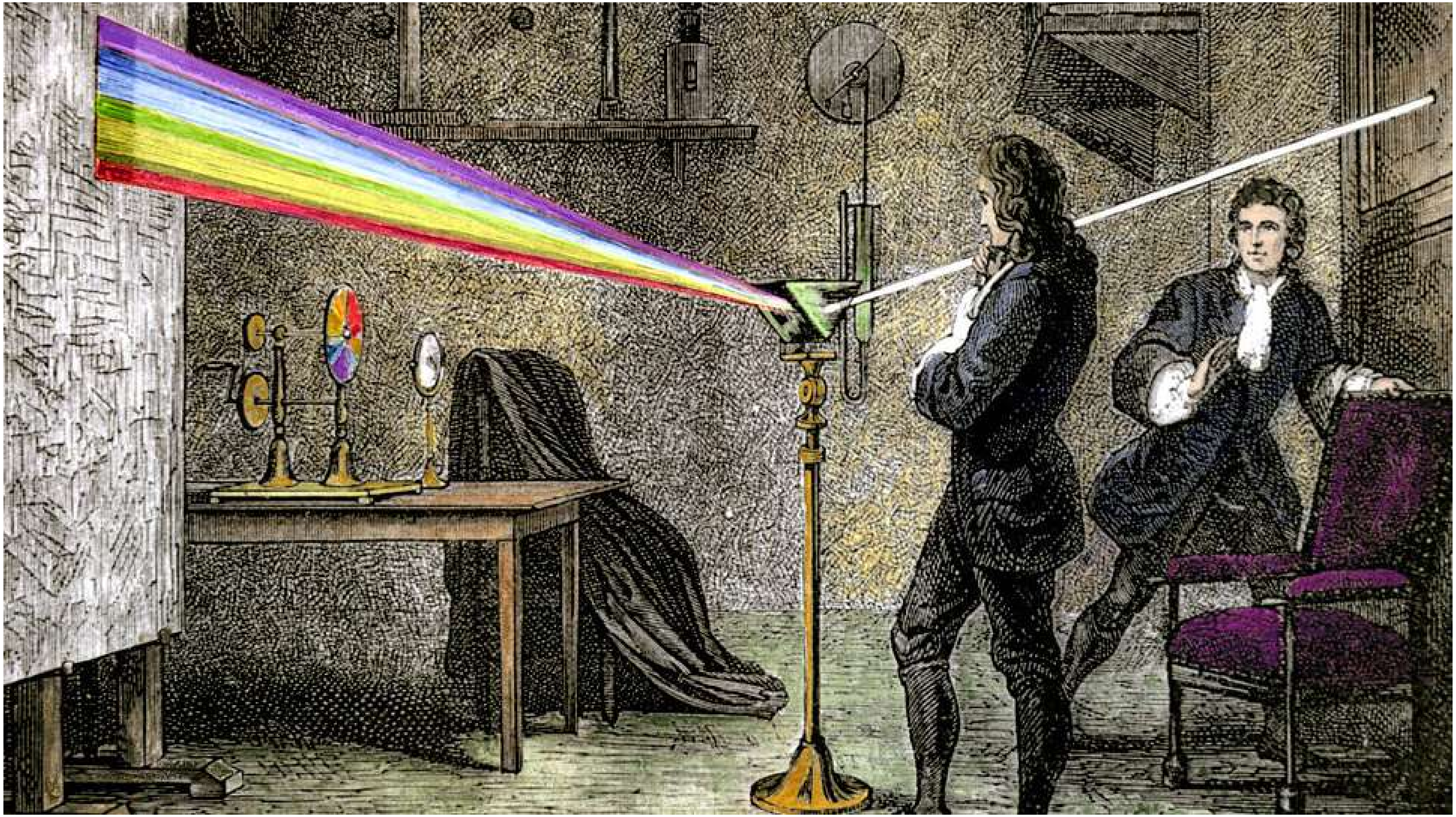
Advanced Graphics and Image Processing

Colour perception and colour spaces

Part 1/5 – physics of light

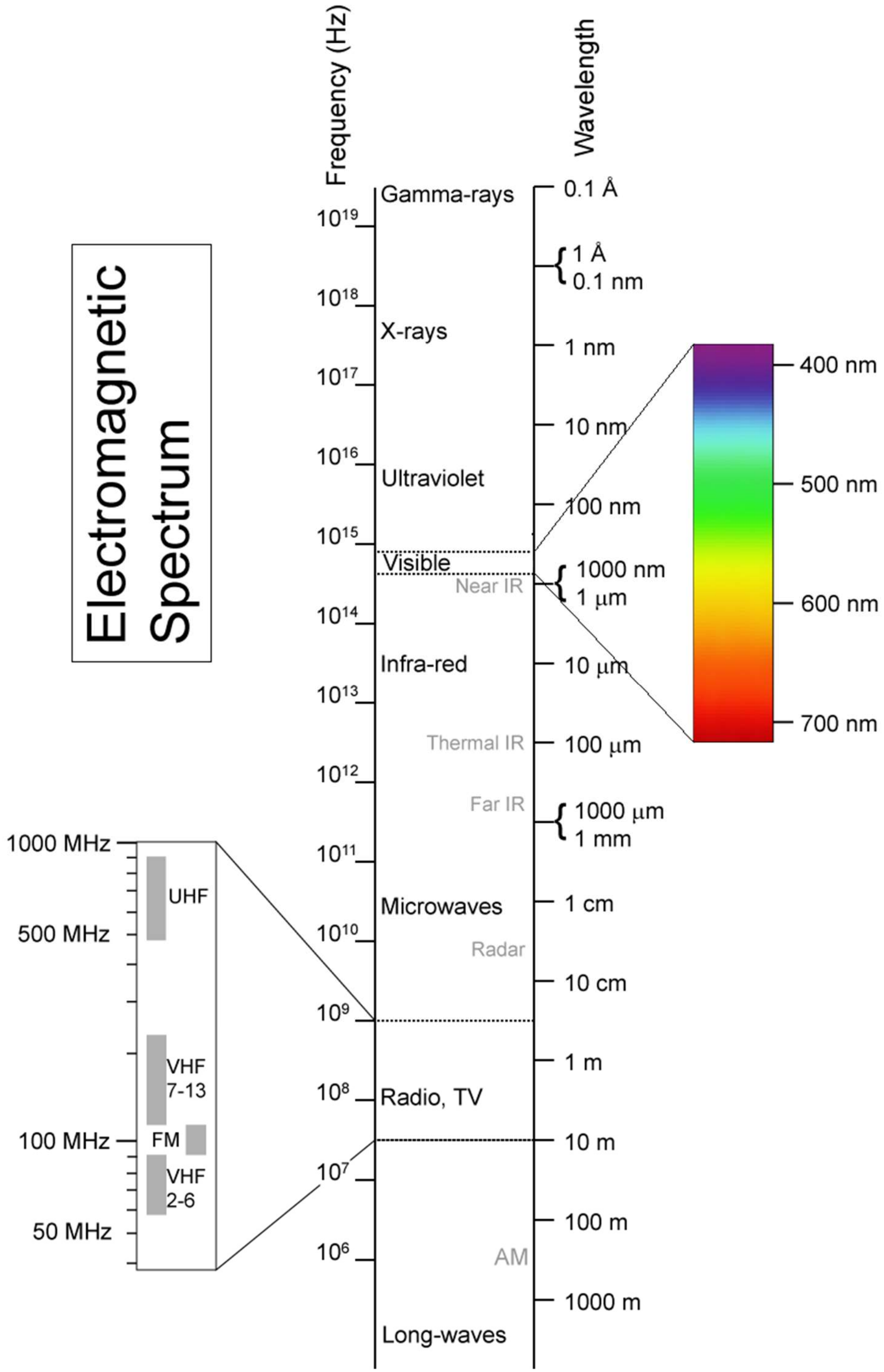
Rafał Mantiuk

Computer Laboratory, University of Cambridge



Electromagnetic spectrum

- Visible light
  - Electromagnetic waves of wavelength in the range 380nm to 730nm
  - Earth's atmosphere lets through a lot of light in this wavelength band
  - Higher in energy than thermal infrared, so heat does not interfere with vision

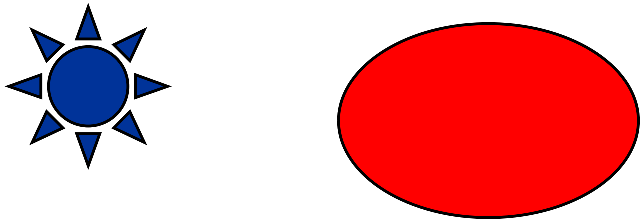


Colour

- There is no physical definition of colour – colour is the result of our perception

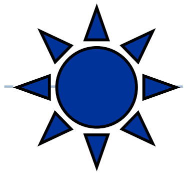
- For reflective displays / objects

$$\text{colour} = \text{perception}(\ \text{illumination} \times \text{reflectance}\ )$$



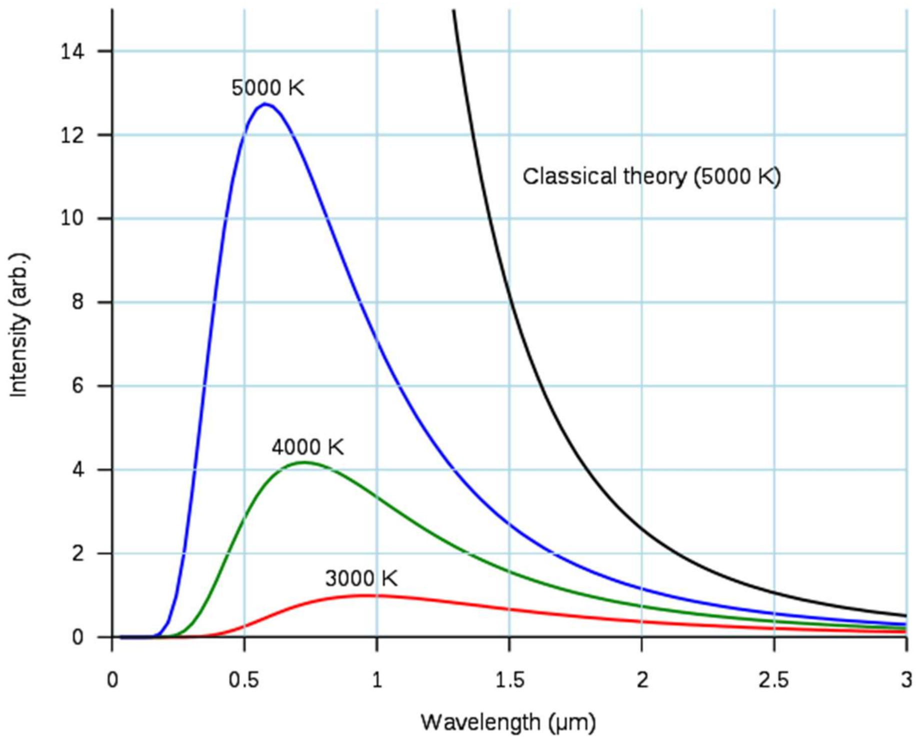
- For emissive objects or displays

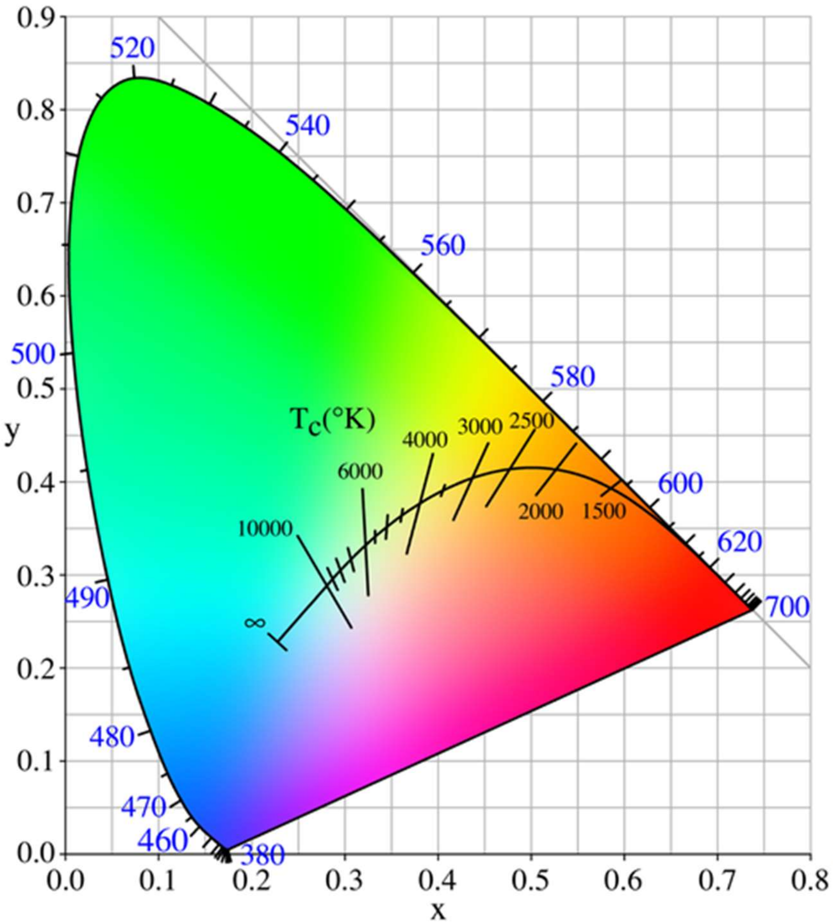
$$\text{colour} = \text{perception}(\ \text{emission}\ )$$



Black body radiation

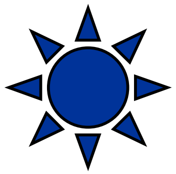
- Electromagnetic radiation emitted by a perfect absorber at a given temperature
  - Graphite is a good approximation of a black body





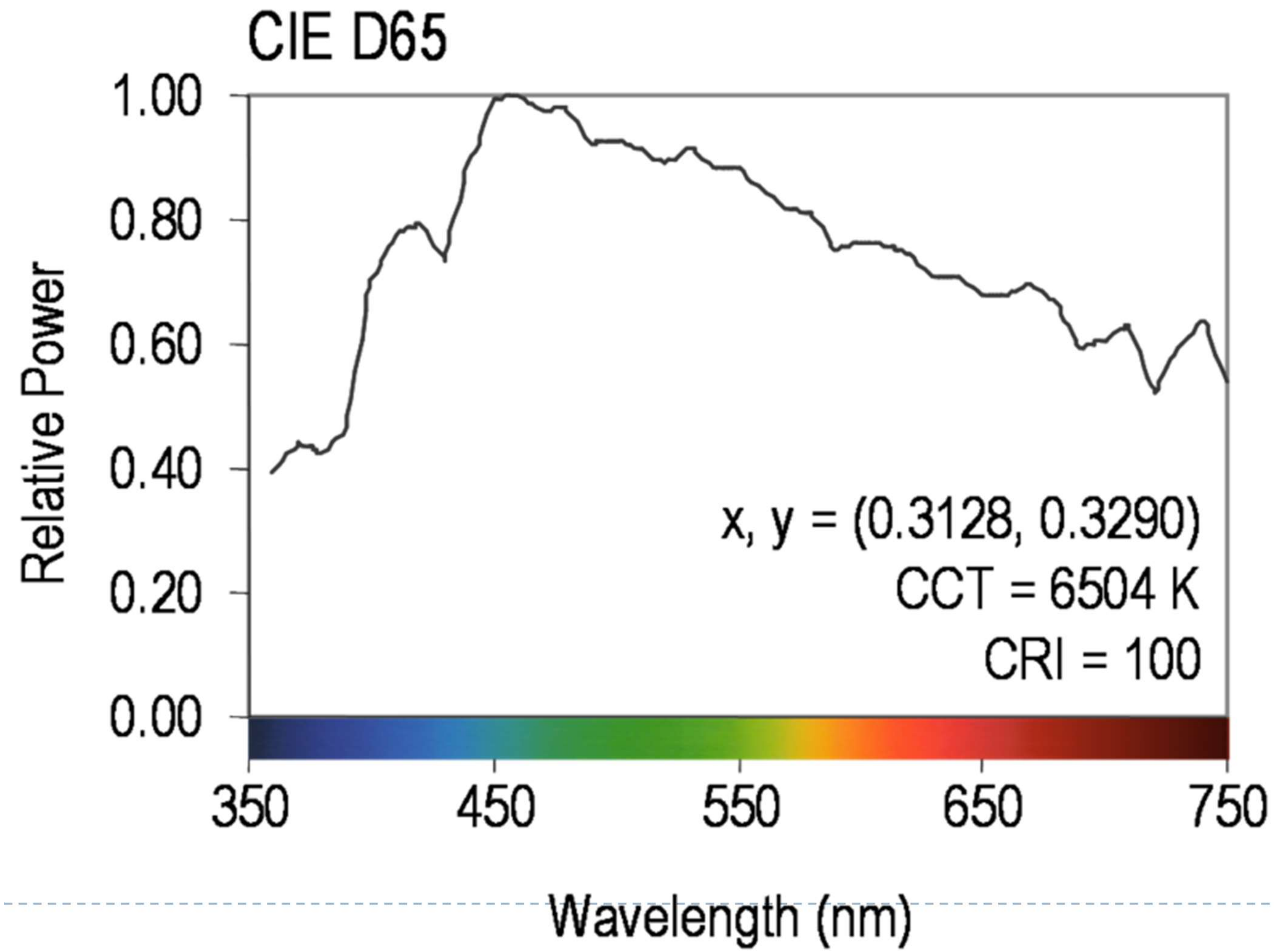
Correlated colour temperature

- The temperature of a black body radiator that produces light most closely matching the particular source
- Examples:
  - Typical north-sky light: 7500 K
  - Typical average daylight: 6500 K
  - Domestic tungsten lamp (100 to 200 W): 2800 K
  - Domestic tungsten lamp (40 to 60 W): 2700 K
  - Sunlight at sunset: 2000 K
- Useful to describe colour of the **illumination** (source of light)



Standard illuminant D65

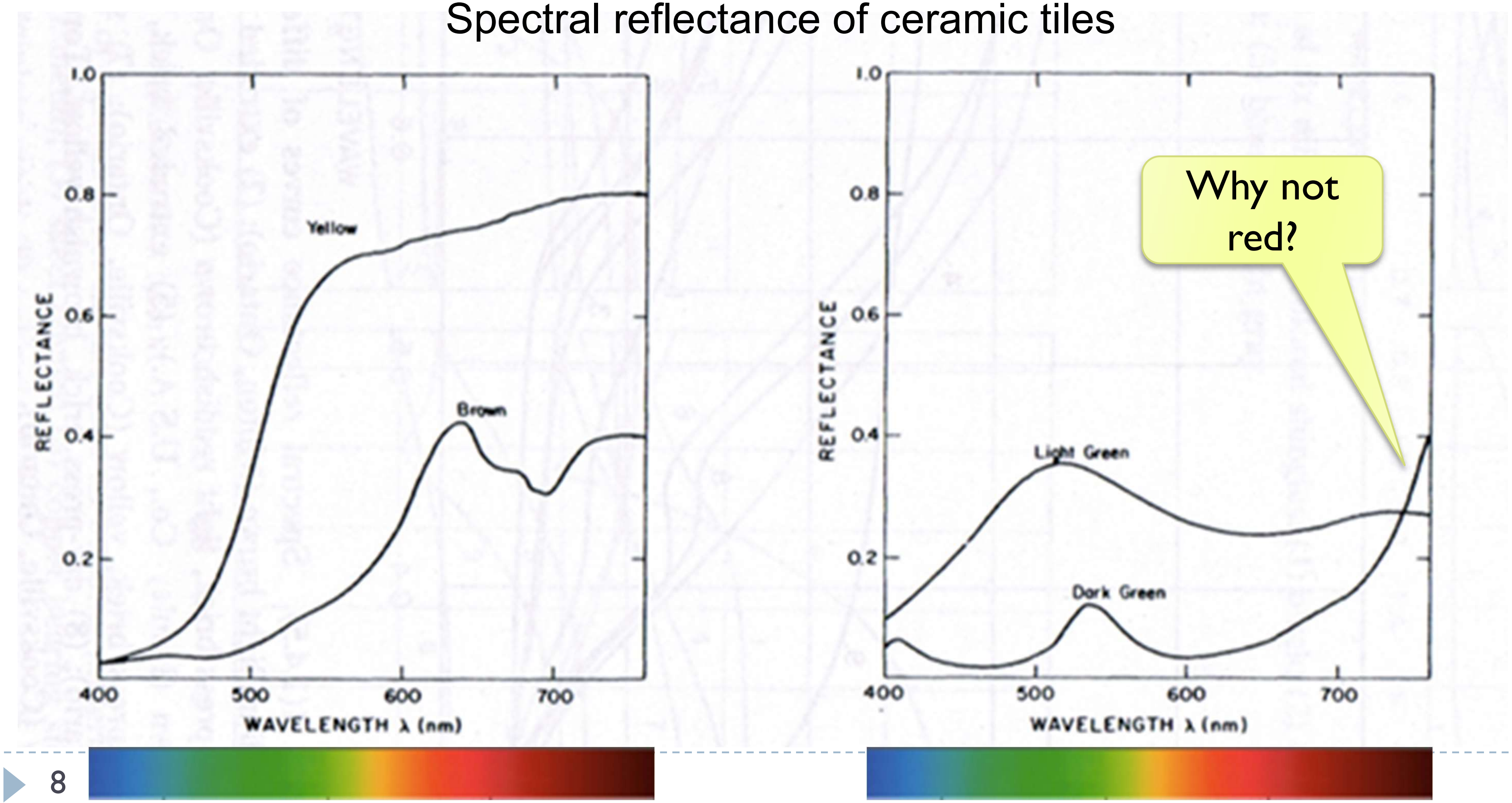
- Mid-day sun in Western Europe / Northern Europe
- Colour temperature approx. 6500 K



Reflectance

- Most of the light we see is reflected from objects
- These objects absorb a certain part of the light spectrum

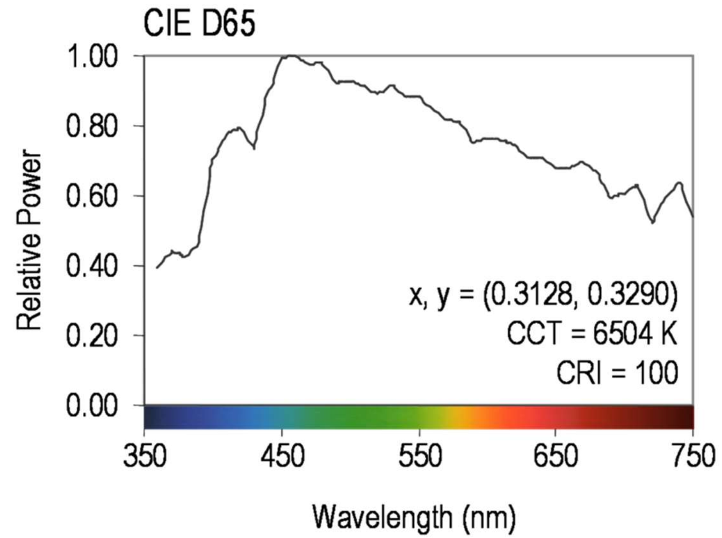
Spectral reflectance of ceramic tiles

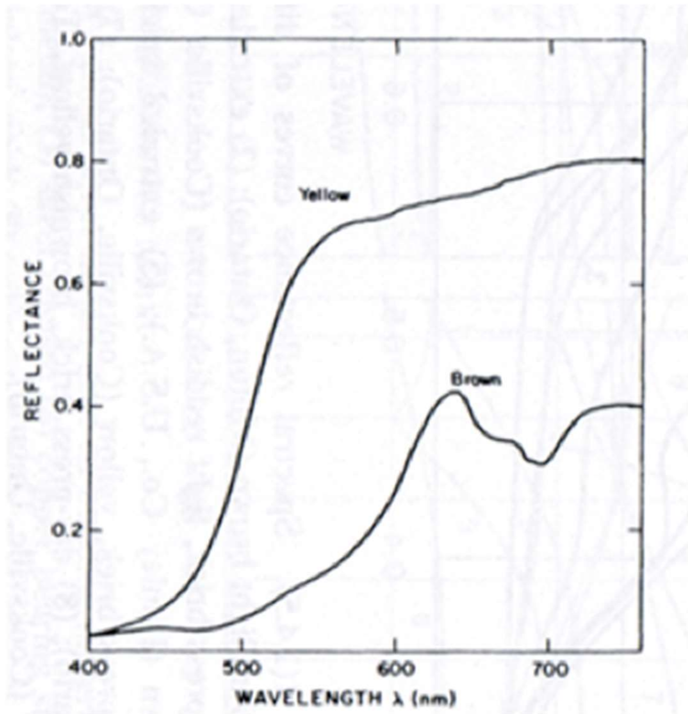


Reflected light

$$L(\lambda) = I(\lambda)R(\lambda)$$

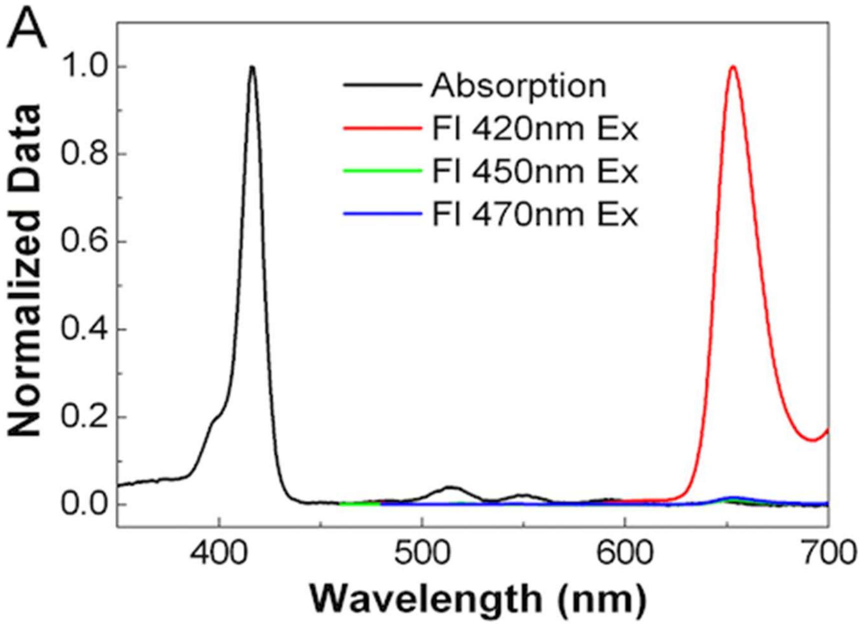
- Reflected light = illumination × reflectance

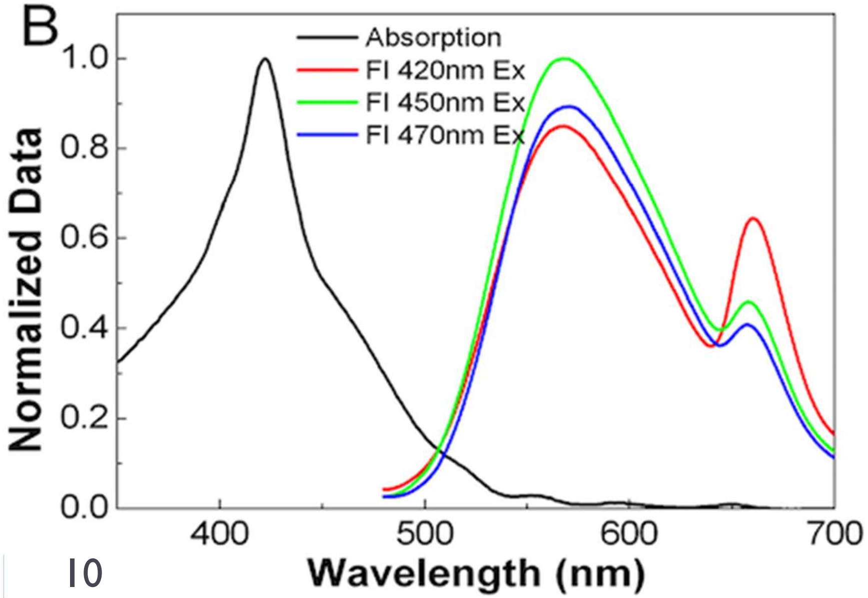


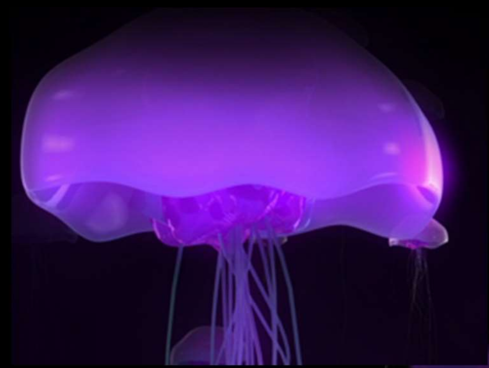


The same object may appear to have different color under different illumination.

Fluorescence







UNIVERSITY OF CAMBRIDGE
COMPUTER LABORATORY

RAINBOW GRAPHICS GROUP

Advanced Graphics and Image Processing

Colour perception and colour spaces

Part 2/5 – perception, cone fundamentals

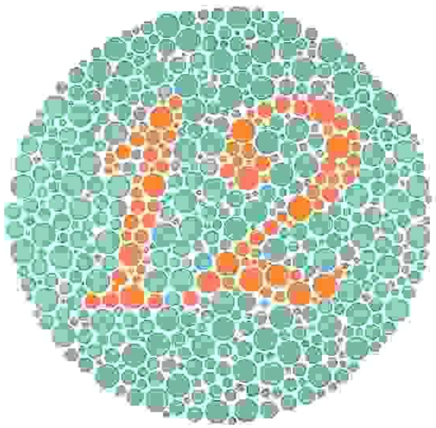
Rafał Mantiuk

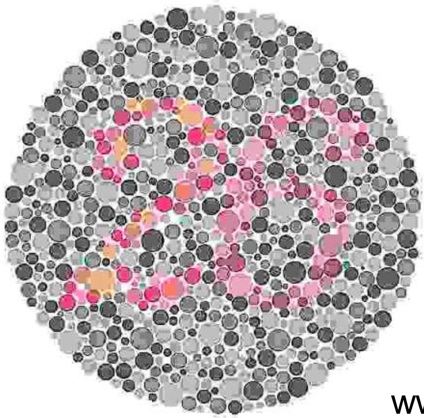
Computer Laboratory, University of Cambridge

Colour perception

- Di-chromaticity (dogs, cats)
  - Yellow & blue-violet
  - Green, orange, red indistinguishable
- Tri-chromaticity (humans, monkeys)
  - Red-ish, green-isn, blue-ish
  - Colour-deficiency
    - Most often men, green-red colour-deficiency

www.lam.mus.ca.us/cats/color/

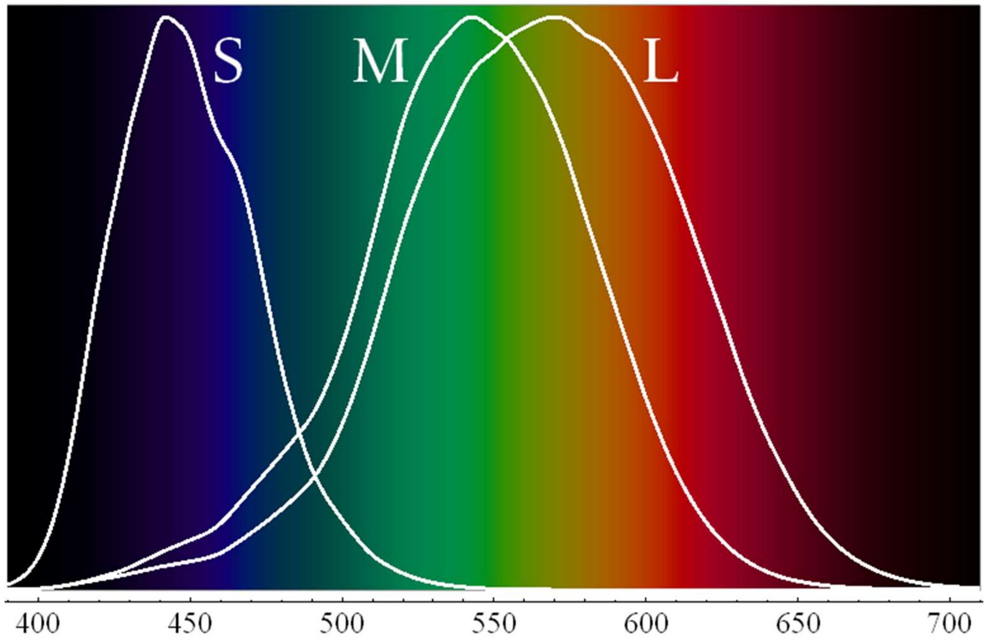




www.colorcube.com/illusions/clrblnd.html

Colour vision

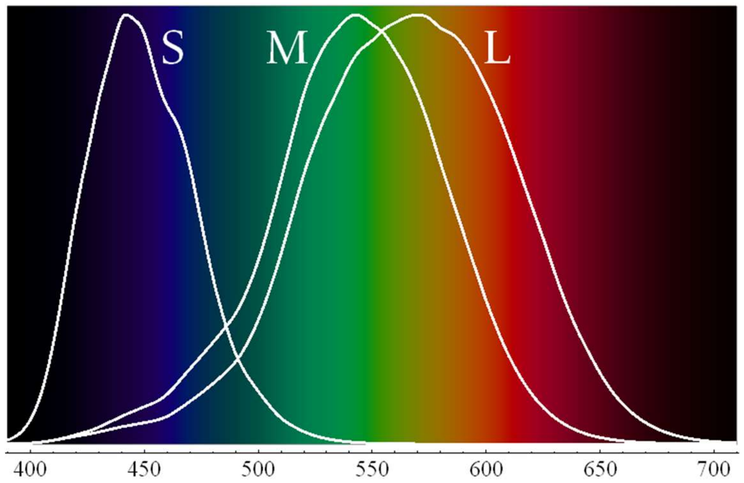
- Cones are the photreceptors responsible for colour vision
  - Only daylight, we see no colours when there is not enough light
- Three types of cones
  - S – sensitive to short wavelengths
  - M – sensitive to medium wavelengths
  - L – sensitive to long wavelengths

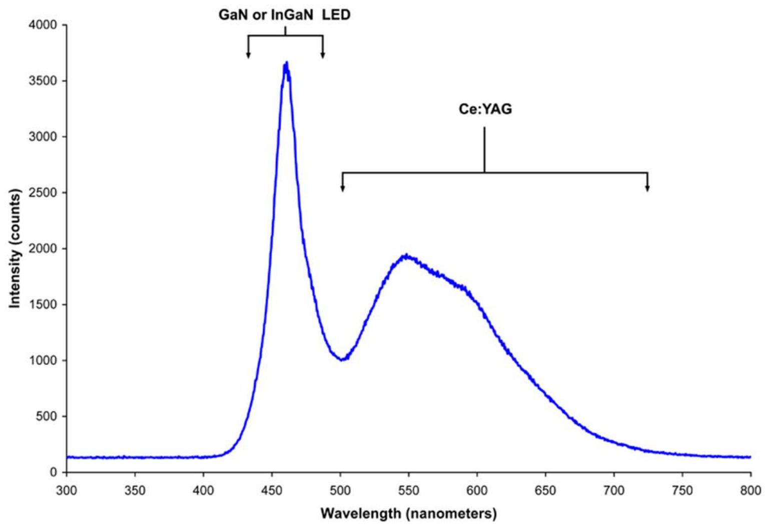


Sensitivity curves – probability that a photon of that wavelengths will be absorbed by a photoreceptor. S,M and L curves are normalized in this plot.

Perceived light

- cone response = sum(sensitivity × reflected light)





Although there is an infinite number of wavelengths, we have only three photoreceptor types to sense differences between light spectra

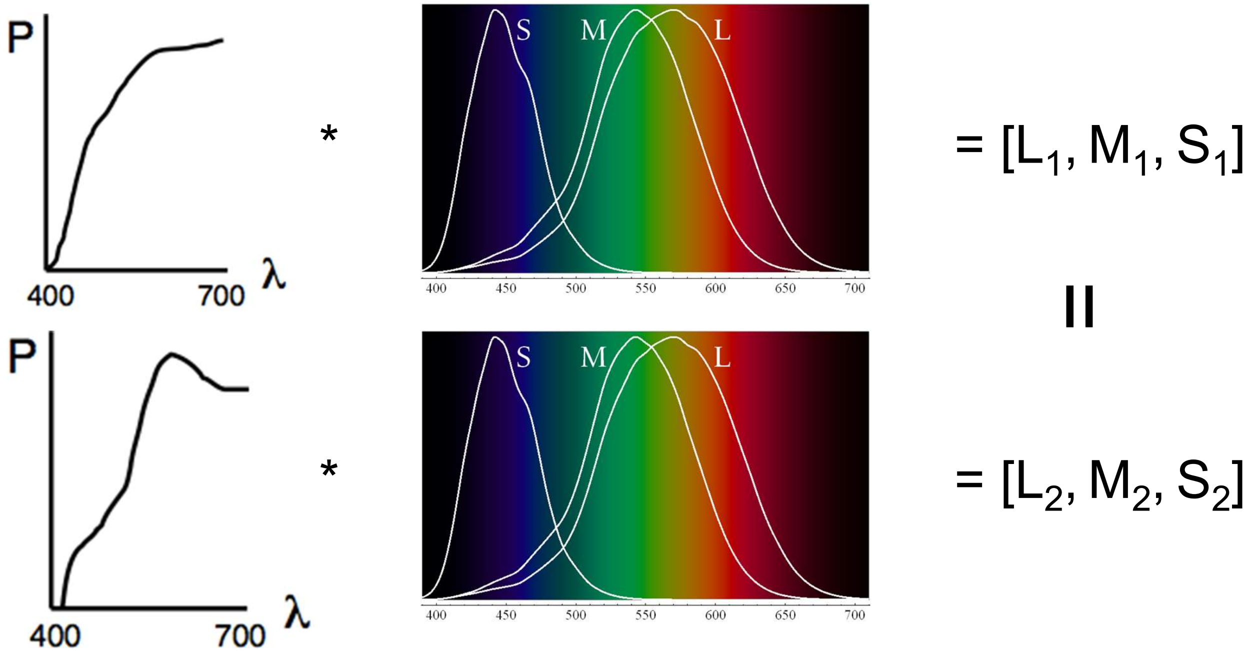
Formally

$$R_S = \int_{380}^{730} S_S(\lambda) \cdot L(\lambda) d\lambda$$

Index S for S-cones

Metamers

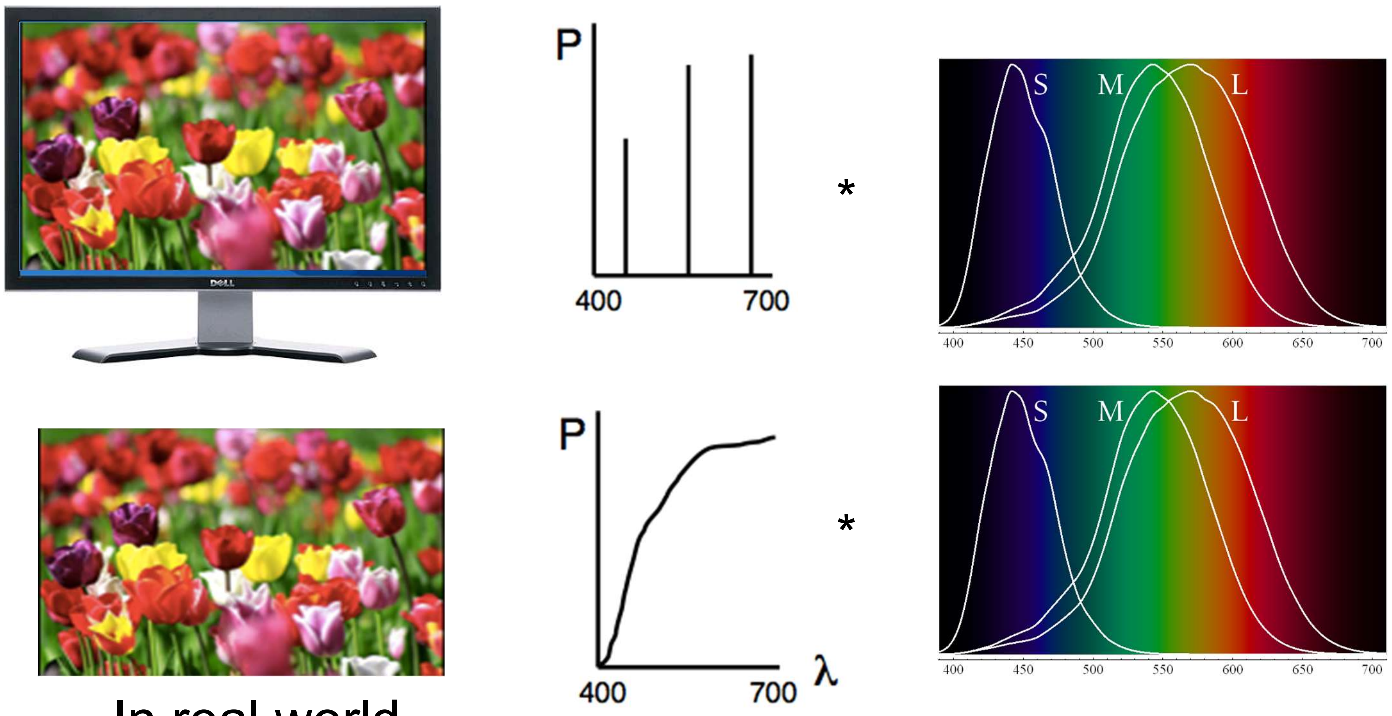
- Even if two light spectra are different, they may appear to have the same colour
- The light spectra that appear to have the same colour are called **metamers**
- Example:



Practical application of metamerism

- Displays do not emit the same light spectra as real-world objects
- Yet, the colours on a display look almost identical

On the display



$= [L_1, M_1, S_1]$

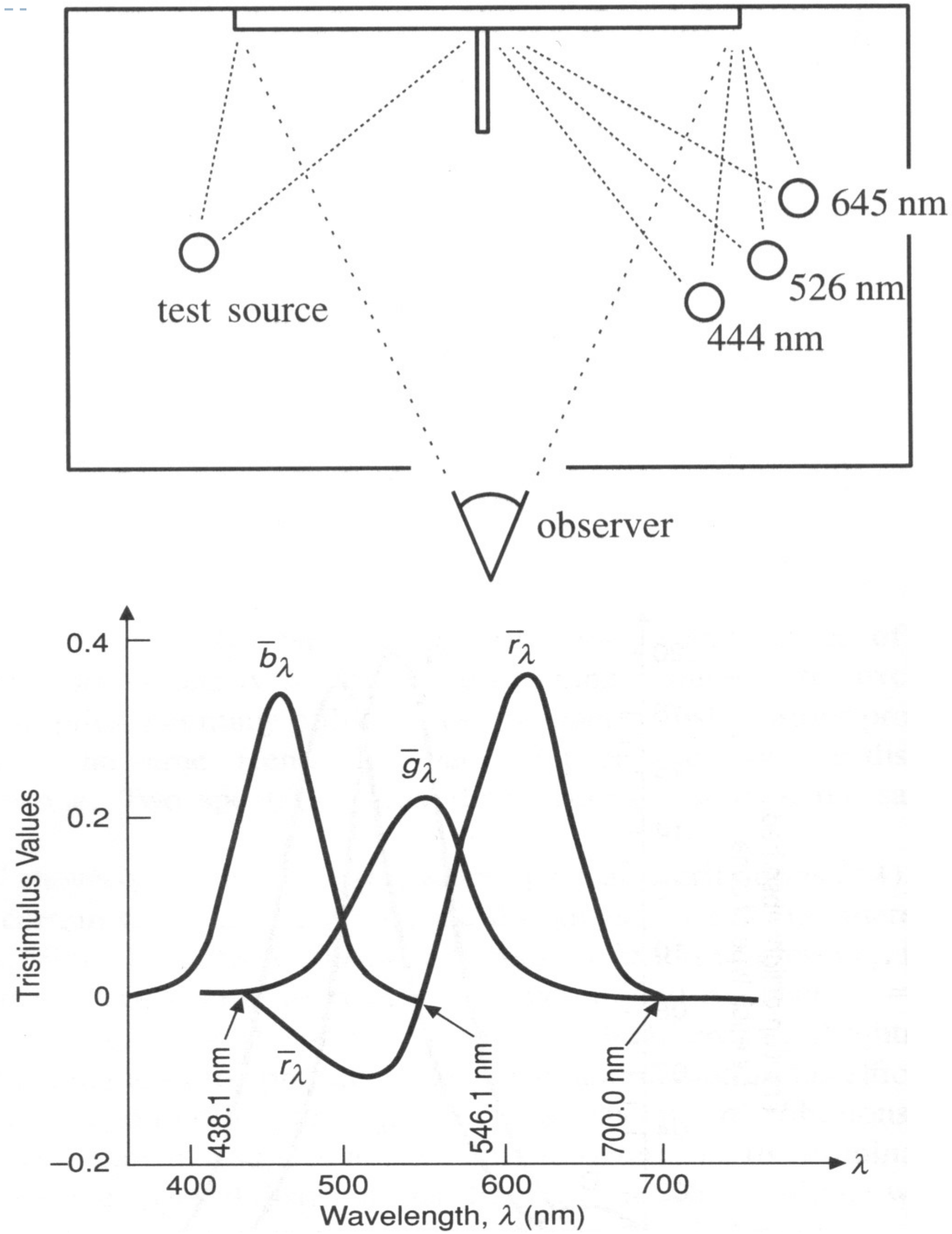
$\parallel$

$= [L_2, M_2, S_2]$

In real world

Tristimulus Colour Representation

- Observation
  - Any colour can be matched using three linear independent reference colours
  - May require "negative" contribution to test colour
  - Matching curves describe the value for matching mono-chromatic spectral colours of equal intensity
    - With respect to a certain set of primary colours

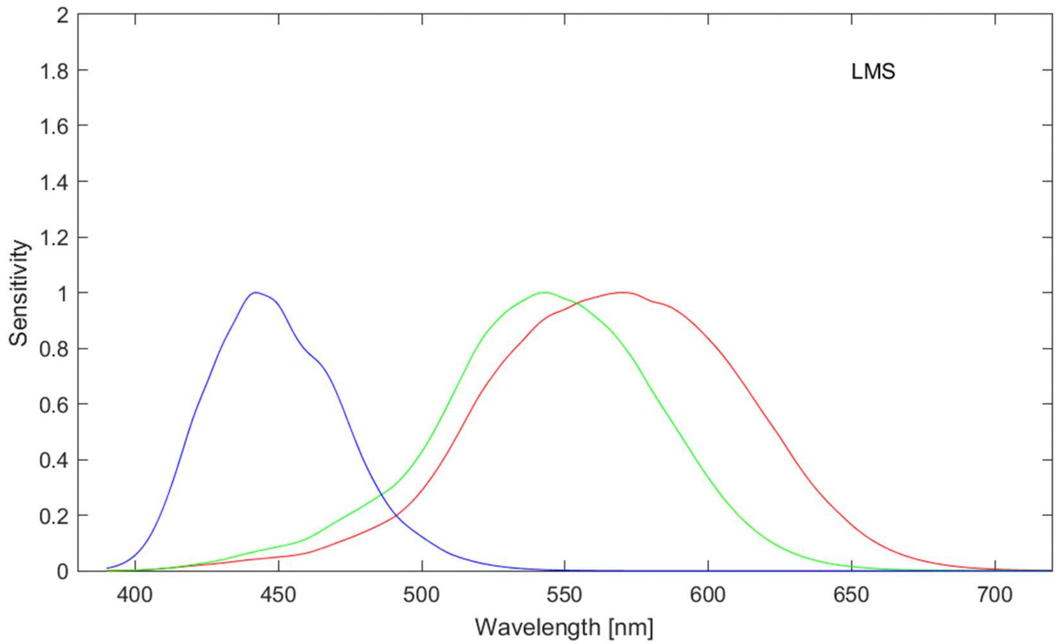


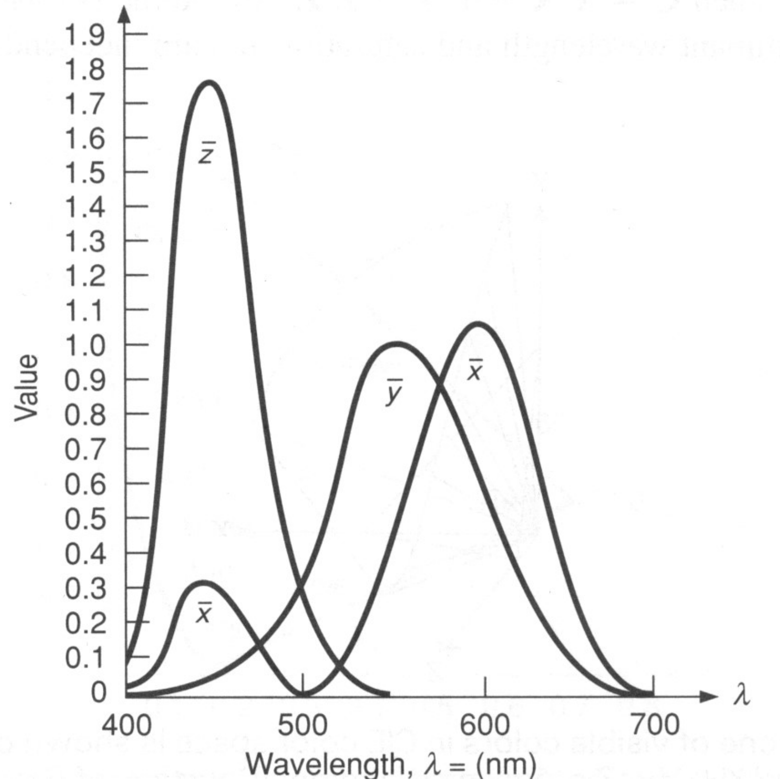
Standard Colour Space CIE-XYZ

- CIE Experiments [Guild and Wright, 1931]
  - Colour matching experiments
  - Group ~12 people with „normal“ colour vision
  - 2 degree visual field (fovea only)
- CIE 2006 XYZ
  - Derived from LMS colour matching functions by Stockman & Sharpe
  - S-cone response differs the most from CIE 1931
- CIE-XYZ Colour Space
  - Goals
    - Abstract from concrete primaries used in an experiment
    - All matching functions are positive
    - Primary „Y” is roughly proportionally to achromatic response (luminance)

Standard Colour Space CIE-XYZ

- Standardized imaginary primaries CIE XYZ (1931)
  - Could match all physically realizable colour stimuli
  - Cone sensitivity curves can be obtained by a linear transformation of CIE XYZ



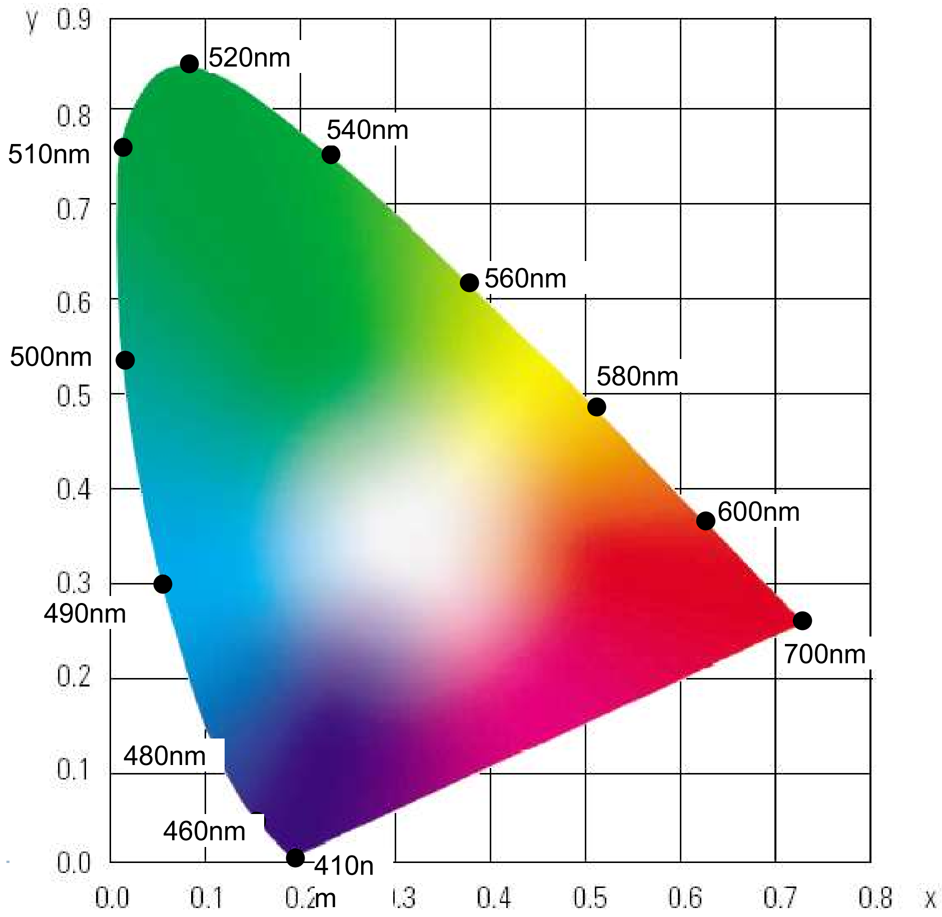


CIE chromaticity diagram

- *chromaticity* values are defined in terms of x, y, z

$$x = \frac{X}{X+Y+Z}, \quad y = \frac{Y}{X+Y+Z}, \quad z = \frac{Z}{X+Y+Z} \qquad x + y + z = 1$$

  - ignores luminance
  - can be plotted as a 2D function
- pure colours (single wavelength) lie along the outer curve
- all other colours are a mix of pure colours and hence lie inside the curve
- points outside the curve do not exist as colours



Advanced Graphics and Image Processing

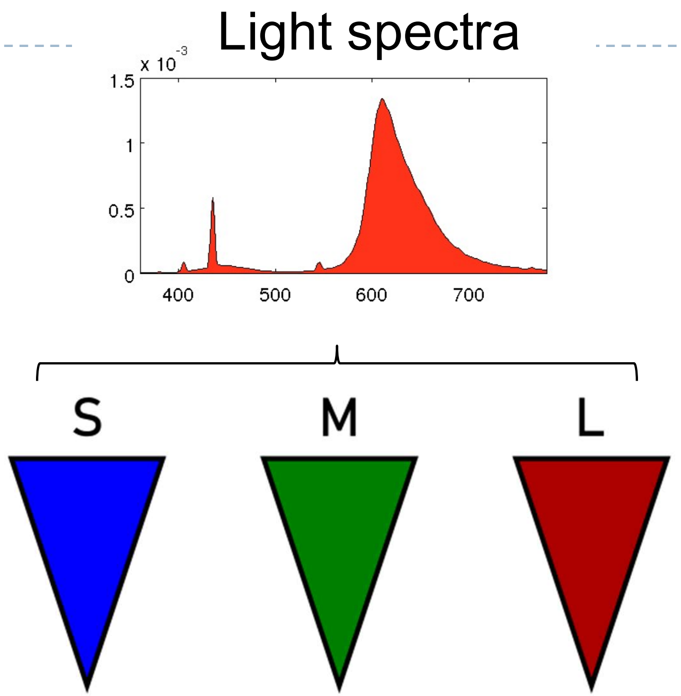
Colour perception and colour spaces

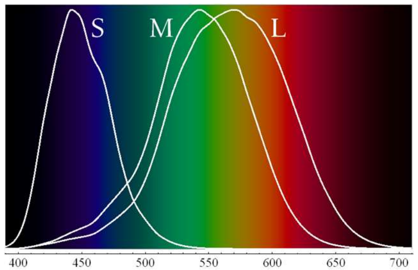
Part 3/5 – colour opponent processing

Rafał Mantiuk

Computer Laboratory, University of Cambridge

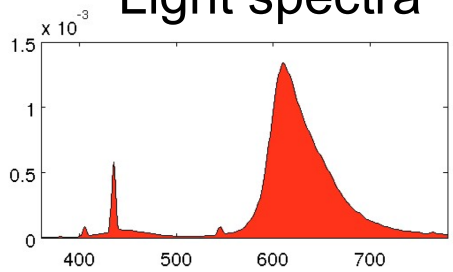
Achromatic/chromatic vision mechanisms

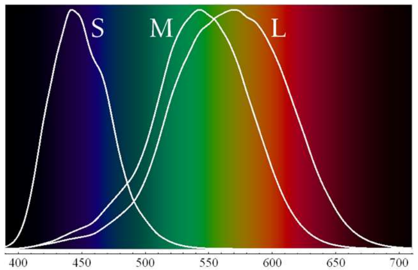


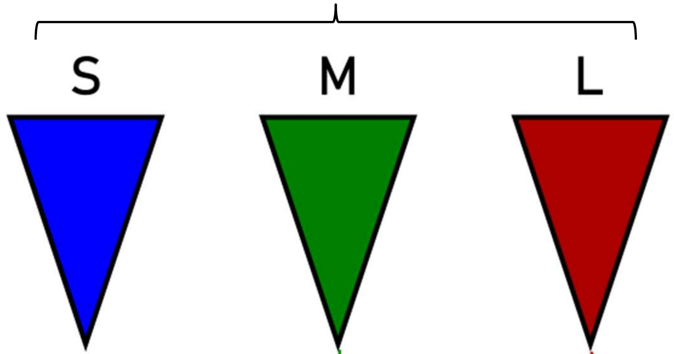


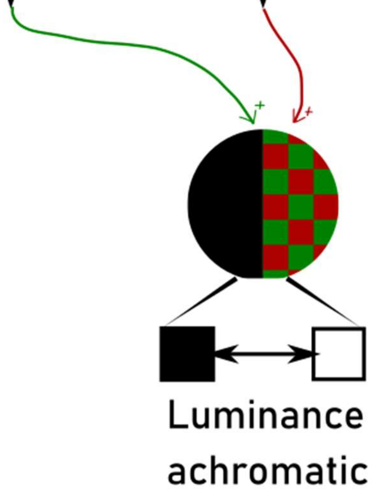
Achromatic/chromatic vision mechanisms

Light spectra



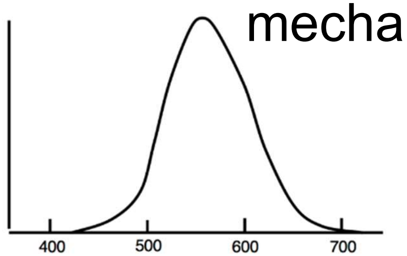




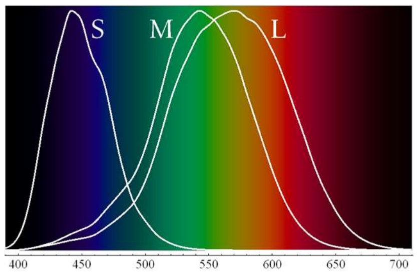


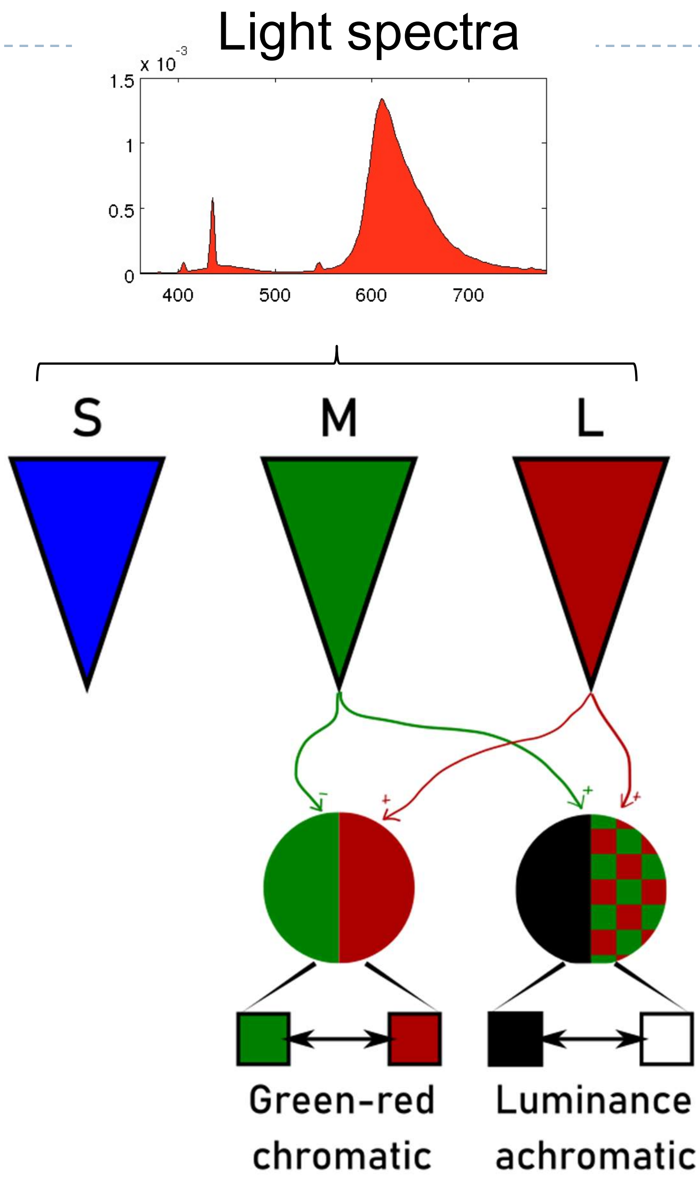
Luminance does NOT explain the brightness of light! [Koenderink et al. Vision Research 2016]

Sensitivity of the achromatic mechanism

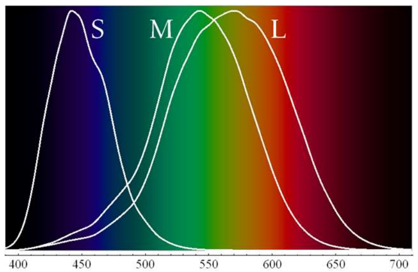


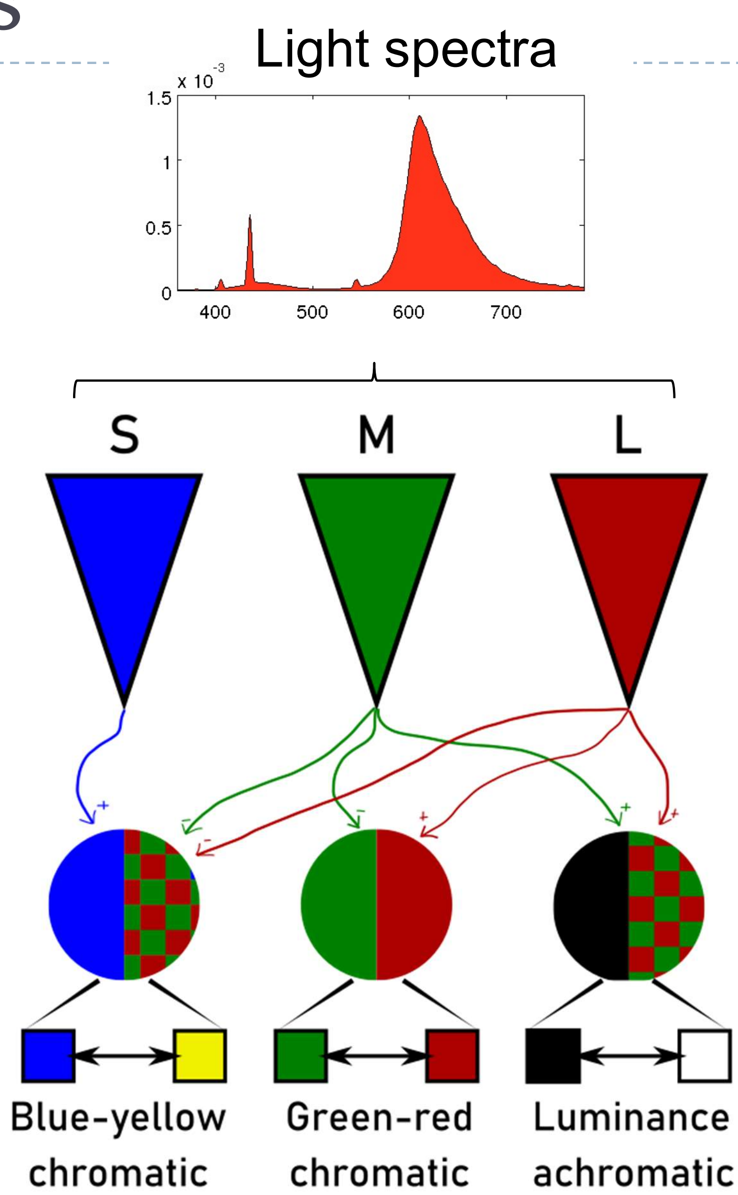
Achromatic/chromatic vision mechanisms





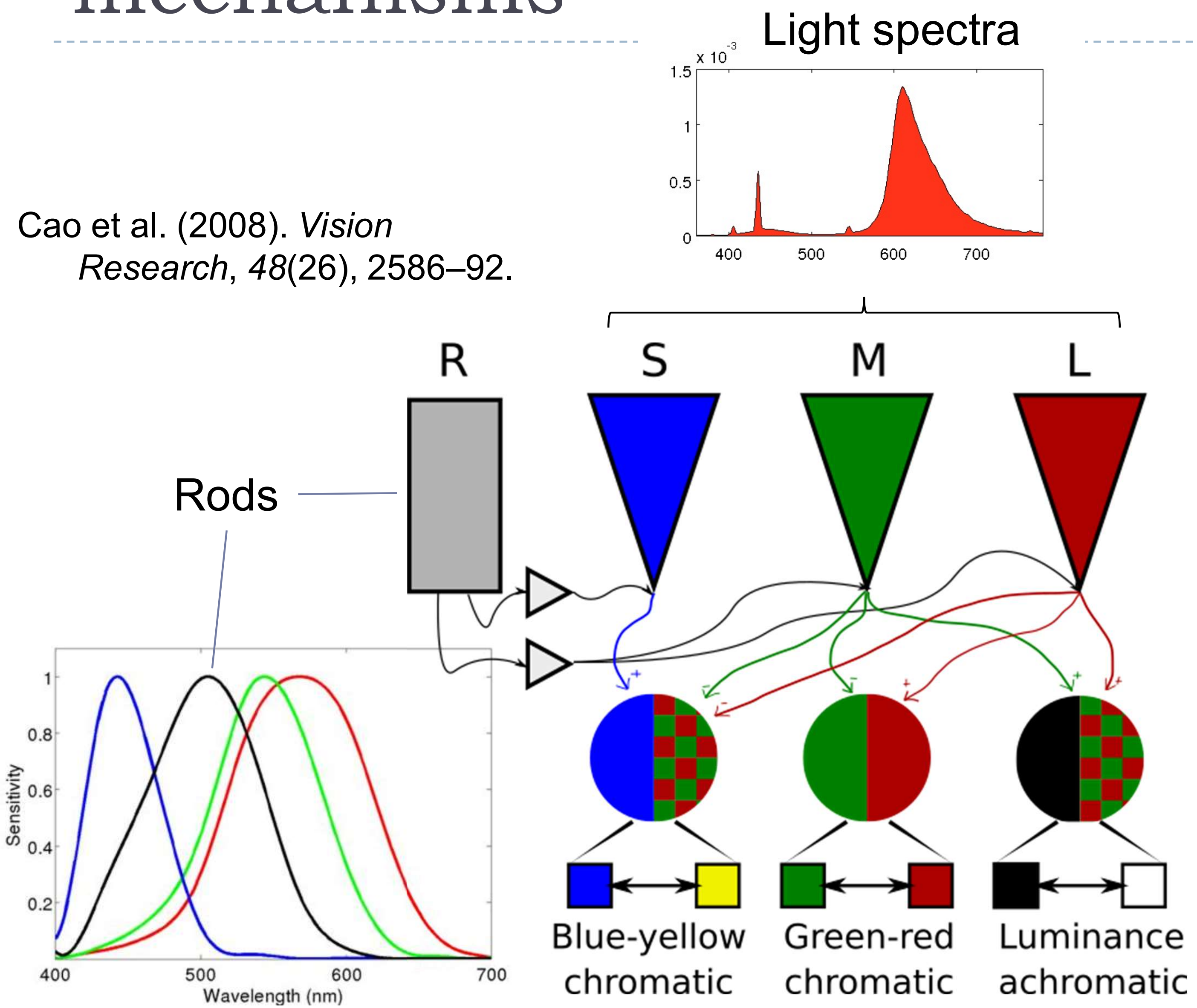
Achromatic/chromatic vision mechanisms





Achromatic/chromatic vision mechanisms

Cao et al. (2008). *Vision Research*, *48*(26), 2586–92.



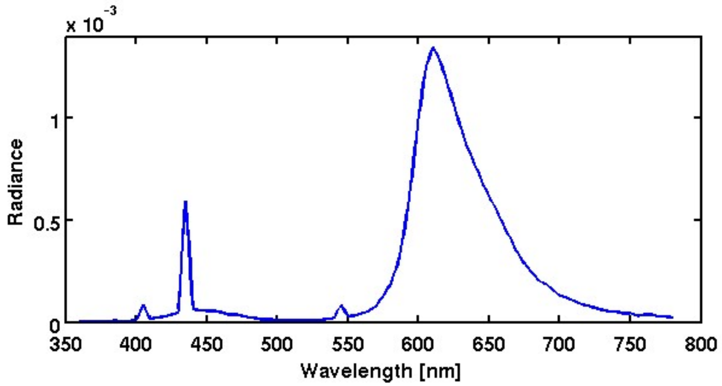
Luminance

- Luminance – measure of light weighted by the response of the achromatic mechanism. Units: cd/m^2

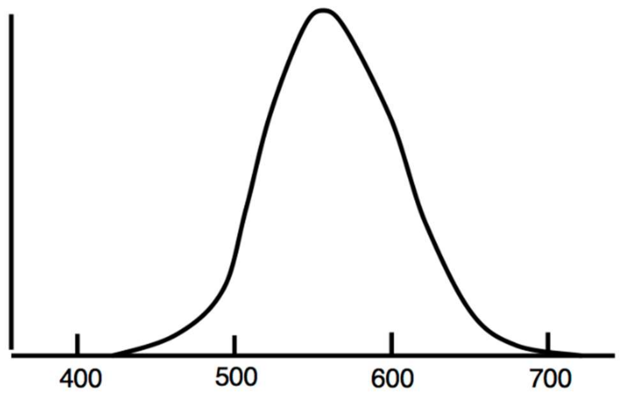
Luminance

$$L_V = \int_{350}^{700} kL(\lambda)V(\lambda)d\lambda \qquad k = \frac{1}{683.002}$$

Light spectrum (radiance)



Luminous efficiency function (weighting)



Advanced Graphics and Image Processing

Colour perception and colour spaces

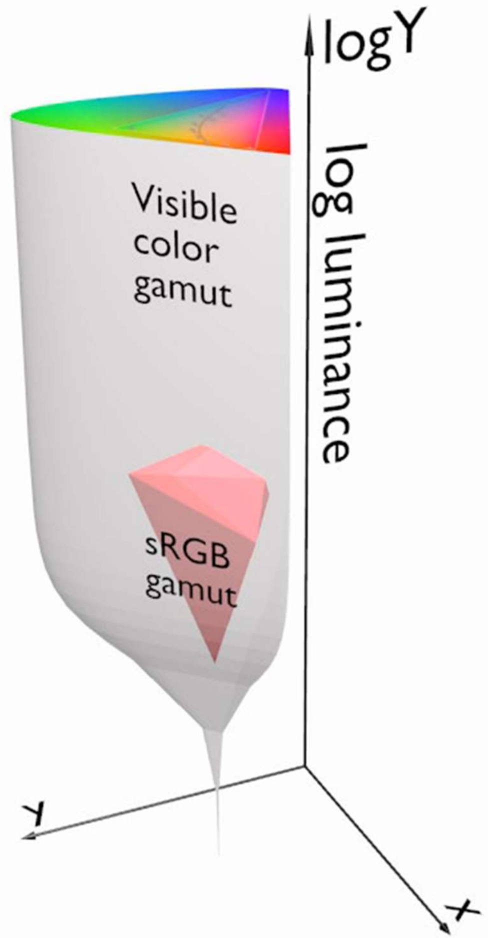
Part 4/5 – gamuts, linear and gamma-encoded colour

Rafał Mantiuk

Computer Laboratory, University of Cambridge

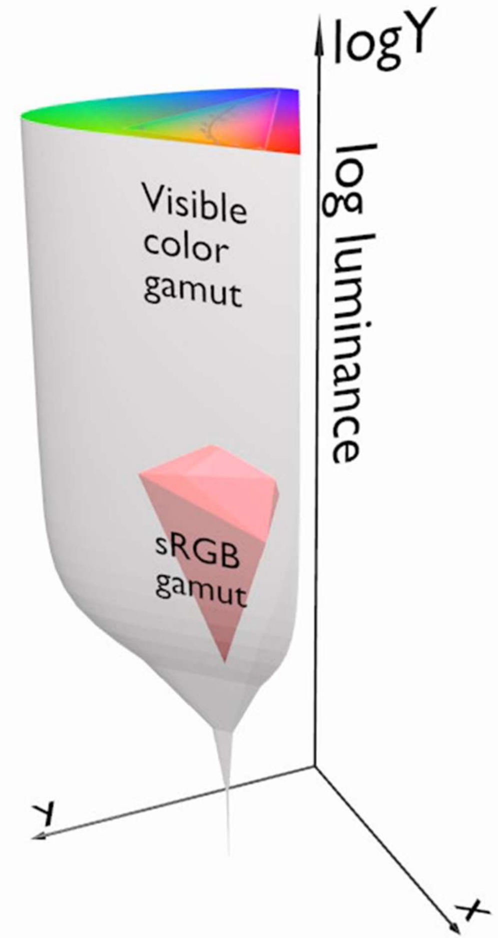
Visible vs. displayable colours

- All physically possible and visible colours form a solid in the XYZ space
- Each display device can reproduce a subspace of that space
- A chromacity diagram is a projection of a slice taken from a 3D solid in XYZ space
- Colour Gamut – the solid in a colour space
  - Usually defined in XYZ to be device-independent

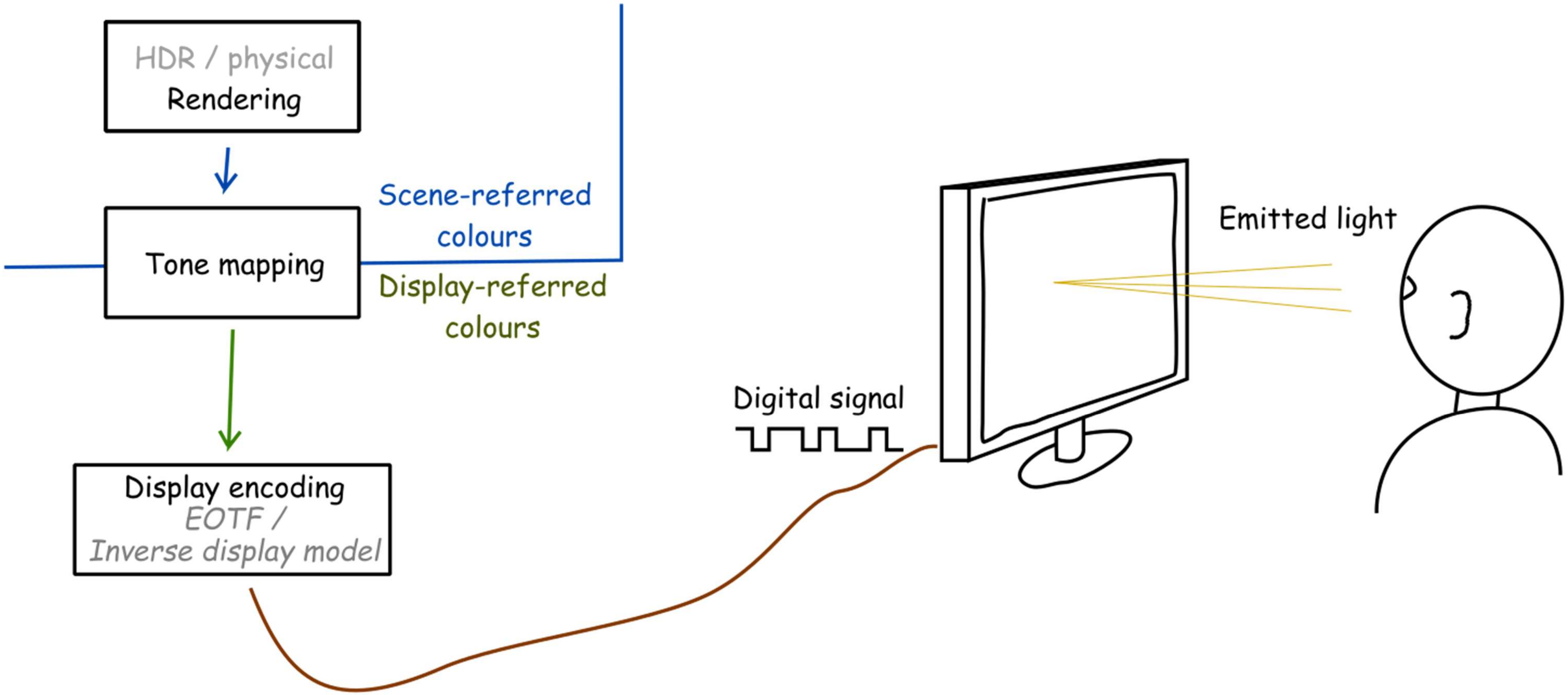


Standard vs. High Dynamic Range

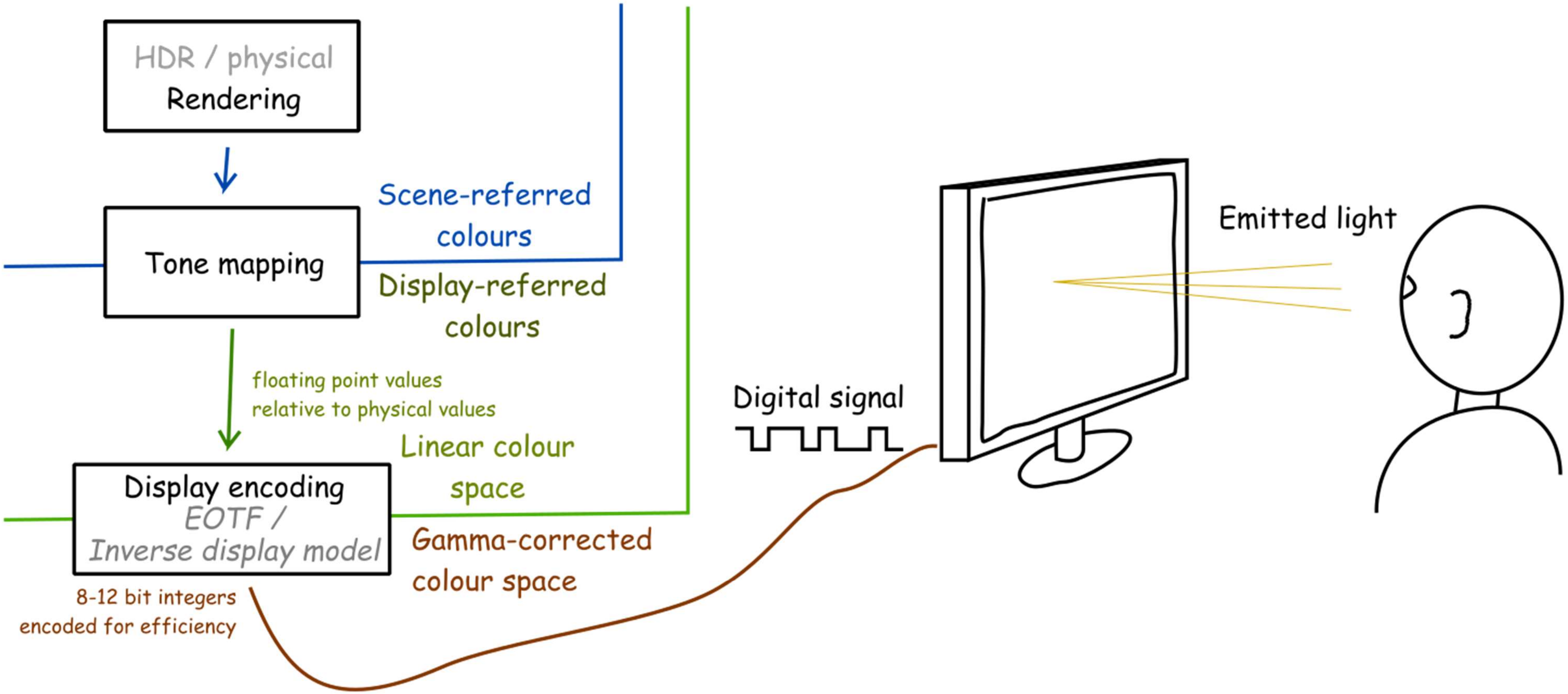
- **HDR** cameras/formats/displays attempt capture/represent/reproduce (almost) all visible colours
  - They represent scene colours and therefore we often call this representation *scene-referred*
- **SDR** cameras/formats/devices attempt to capture/represent/reproduce only colours of a standard sRGB colour gamut, mimicking the capabilities of CRTs monitors
  - They represent display colours and therefore we often call this representation *display-referred*



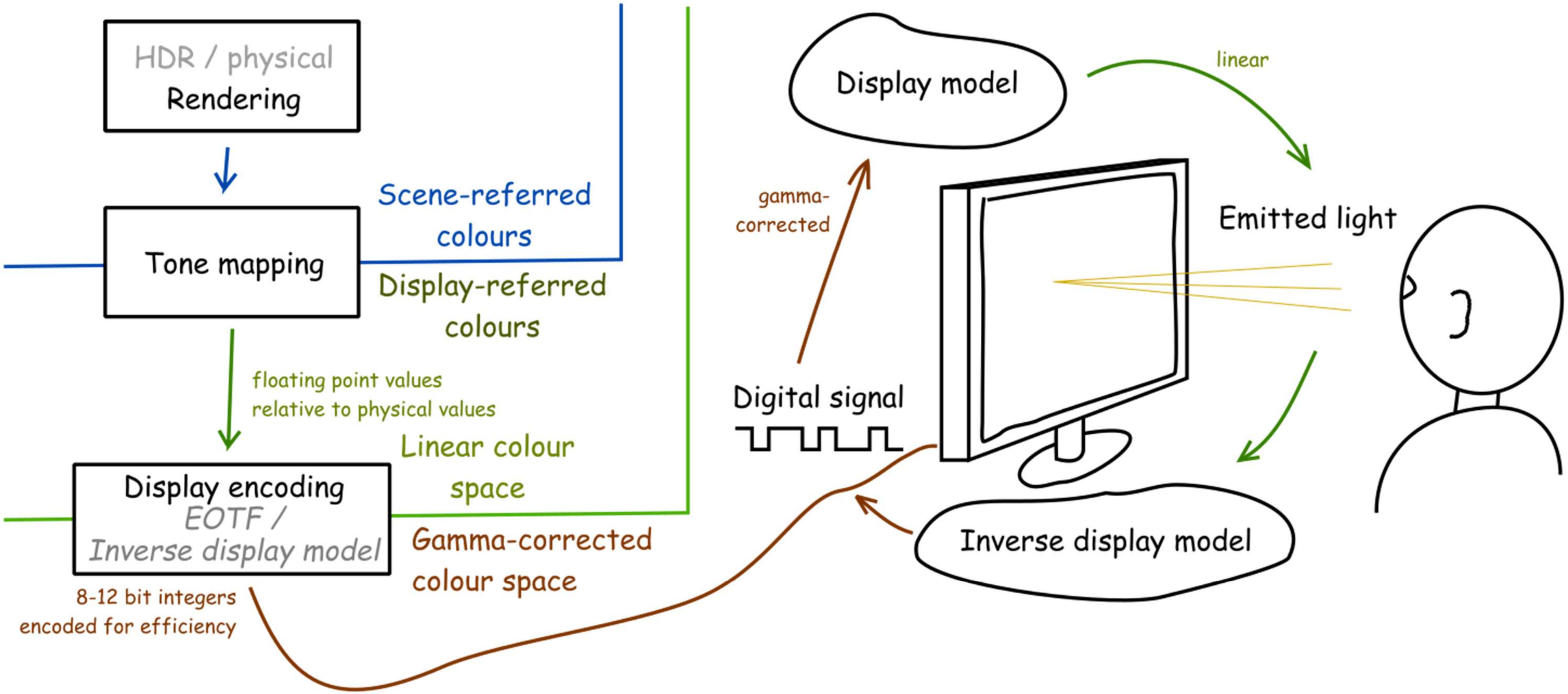
From rendering to display



From rendering to display



From rendering to display



Display encoding for SDR: gamma

- Gamma correction is often used to encode luminance or tri-stimulus color values (RGB) in imaging systems (displays, printers, cameras, etc.)

Gain

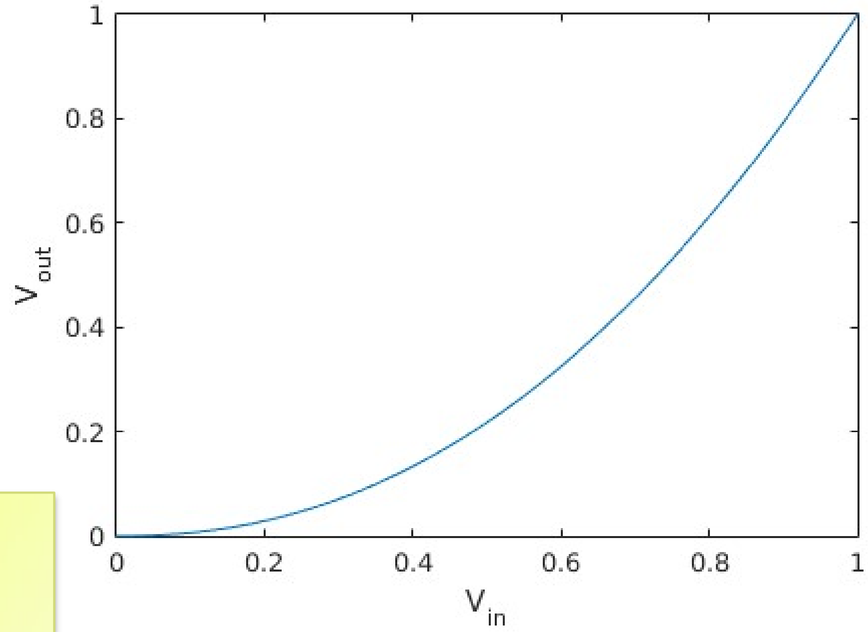
Gamma (usually =2.2)

$$\mathrm{V_{out}} = \mathrm{a} \cdot V_{in}^{\gamma}$$

(relative) Luminance
Physical signal

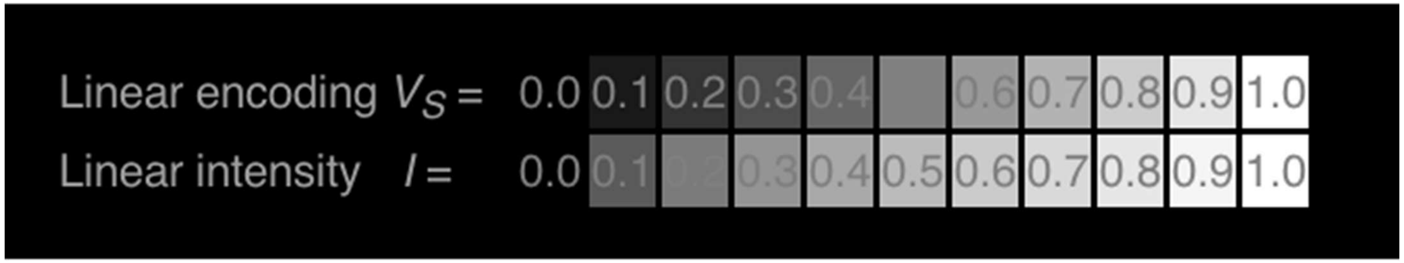
Luma
Digital signal (0-1)

Inverse: $\mathrm{V_{in}} = \left(\frac{1}{a} \cdot V_{out}\right)^{\frac{1}{\gamma}}$



Colour: the same equation applied to red, green and blue colour channels.

Why is gamma needed?



<- Pixel value (luma)

<- Luminance

- *Gamma-corrected* pixel values give a scale of brightness levels that is more perceptually uniform
- At least 12 bits (instead of 8) would be needed to encode each color channel without gamma correction
- And accidentally it was also the response of the CRT gun

Luma – gray-scale pixel value

- **Luma** - pixel "brightness" in *gamma corrected* units

$$L' = 0.2126R' + 0.7152G' + 0.0722B'$$

  - R', G' and B' are *gamma-corrected* colour values
  - Prime symbol denotes *gamma corrected*
  - Used in image/video coding

- Note that relative **luminance** if often approximated with

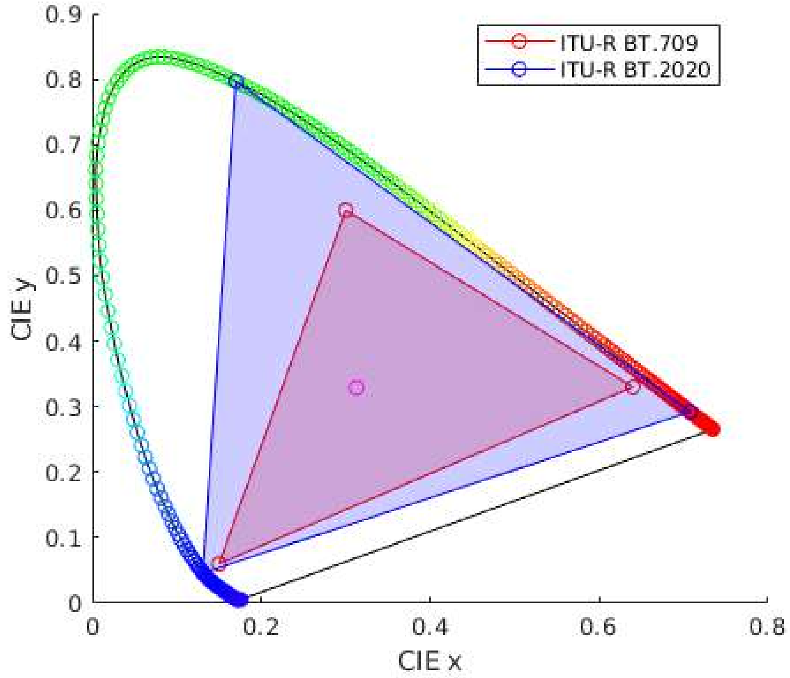
$$L = 0.2126R + 0.7152G + 0.0722B$$
$$= 0.2126(R')^{\gamma}+0.7152(G')^{\gamma}+0.0722(B')^{\gamma}$$

- $R, G,$ and B are *linear* colour values
- Luma and luminace are different quantities despite similar formulas

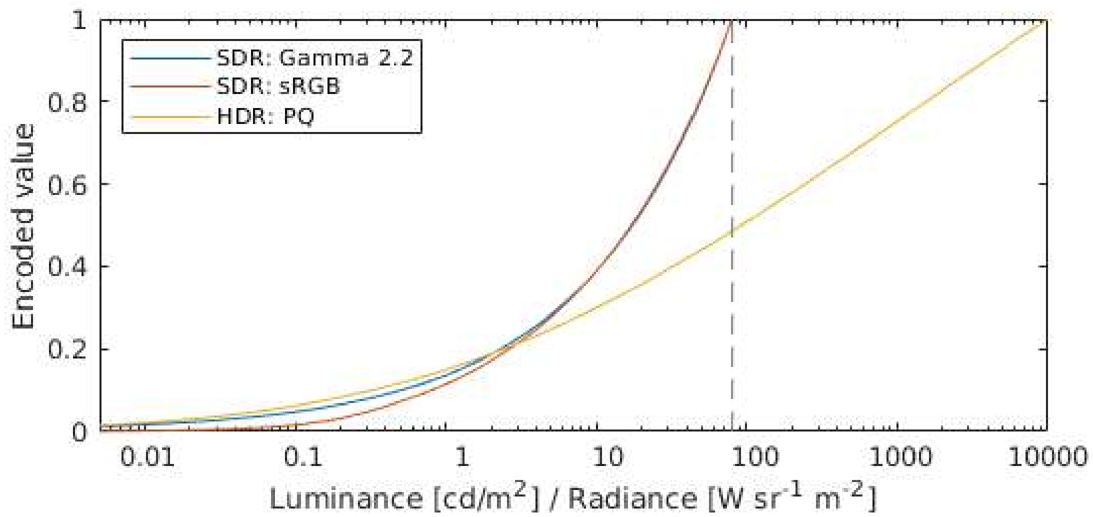
Standards for display encoding

Display type	Colour space	EOTF	Bit depth
Standard Dynamic Range	ITU-R 709	2.2 gamma / sRGB	8 to 10
High Dynamic Range	ITU-R 2020	ITU-R 2100 (PQ/HLG)	10 to 12

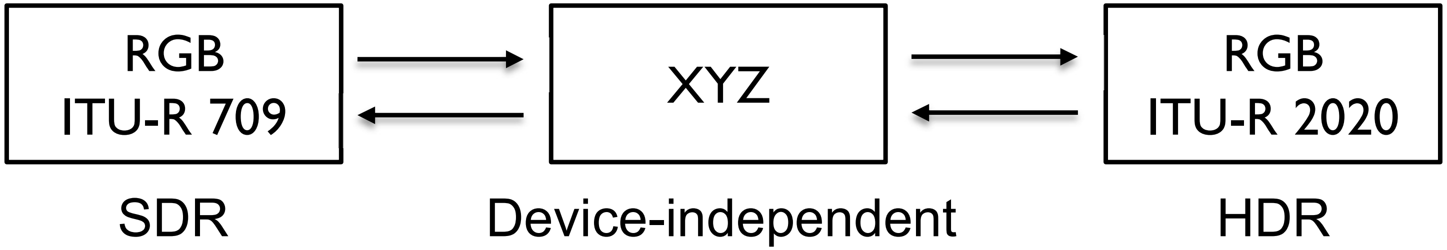
Colour space
What is the XYZ of "pure" red, green and blue



Electro-Optical Transfer Function
How to efficiently encode each primary colour



How to transform between linear RGB colour spaces?



- From ITU-R 709 RGB to XYZ:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix}_{R709toXYZ} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}_{R709}$$

Relative XYZ of the red primary

Relative XYZ of the green primary

Relative XYZ of the blue primary

Relative RGB (0-1) in the R709 space

How to transform between RGB colour spaces?

- From ITU-R **709** RGB to ITU-R **2020** RGB:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix}_{R2020} = M_{XYZtoR2020} \cdot M_{R709toXYZ} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}_{R709}$$

- From ITU-R **2020** RGB to ITU-R **709** RGB:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix}_{R709} = M_{XYZtoR709} \cdot M_{R2020toXYZ} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}_{R2020}$$

- Where:

$$M_{R709toXYZ} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix} \text{ and } M_{XYZtoR709} = M^{-1}_{R709toXYZ}$$

$$M_{R2020toXYZ} = \begin{bmatrix} 0.6370 & 0.1446 & 0.1689 \\ 0.2627 & 0.6780 & 0.0593 \\ 0.0000 & 0.0281 & 1.0610 \end{bmatrix} \text{ and } M_{XYZtoR2020} = M^{-1}_{R2020toXYZ}$$

UNIVERSITY OF CAMBRIDGE
COMPUTER LABORATORY

RAINBOW GRAPHICS GROUP

Advanced Graphics and Image Processing

Colour perception and colour spaces

Part 5/5 – colour spaces

Rafał Mantiuk

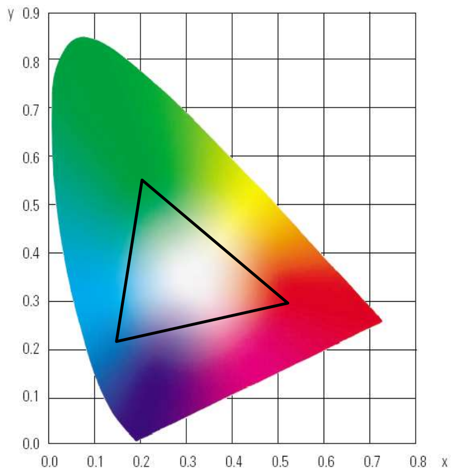
Computer Laboratory, University of Cambridge

Representing colour

- We need a way to represent colour in the computer by some set of numbers
  - A) preferably a small set of numbers which can be quantised to a fairly **small number of bits** each
    - Gamma corrected RGB, sRGB and CMYK for printers
  - B) a set of numbers that are **easy to interpret**
    - Munsell's *artists'* scheme
    - HSV, HLS
  - C) a set of numbers in a 3D space so that the (Euclidean) distance in that space corresponds to approximately **perceptually uniform** colour differences
    - CIE Lab, CIE Luv

RGB spaces

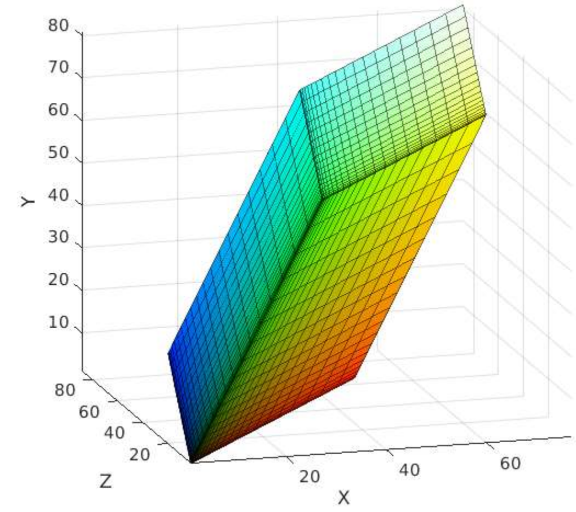
- Most display devices that output light mix red, green and blue lights to make colour
  - televisions, CRT monitors, LCD screens
- RGB colour space
  - Can be **linear** (RGB) or **display-encoded** (R'G'B')
  - Can be **scene-referred** (HDR) or **display-referred** (SDR)
- There are multiple RGB colour spaces
  - ITU-R 709 (sRGB), ITU-R 2020, Adobe RGB, DCI-P3
    - Each using different primary colours
  - And different OETFs (gamma, PQ, etc.)
- Nominally, *RGB* space is a cube



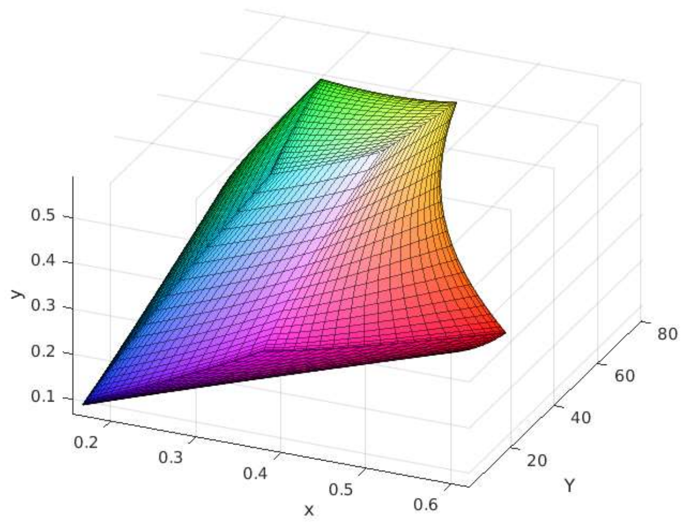
RGB in CIE *XYZ* space

- Linear RGB colour values can be transformed into CIE XYZ
  - by matrix multiplication
  - because it is a rigid transformation the colour gamut in CIE XYZ is a rotate and skewed cube
- Transformation into Yxy
  - is non-linear (non-rigid)
  - colour gamut is more complicated

RGB gamut in XYZ colour space

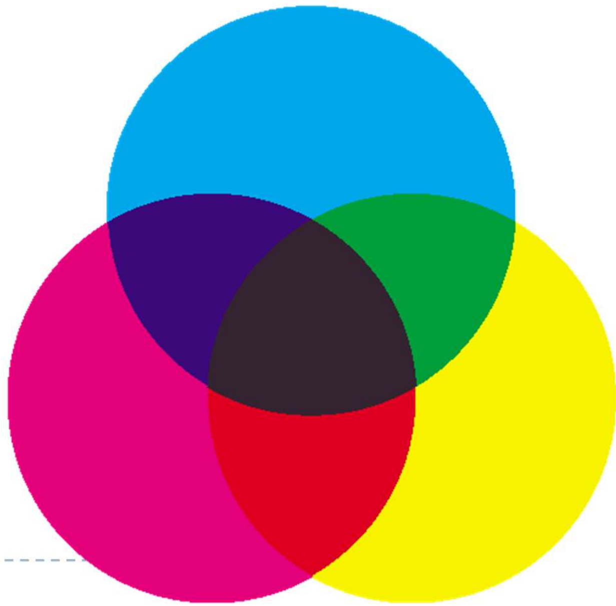


RGB gamut in Yxy colour space

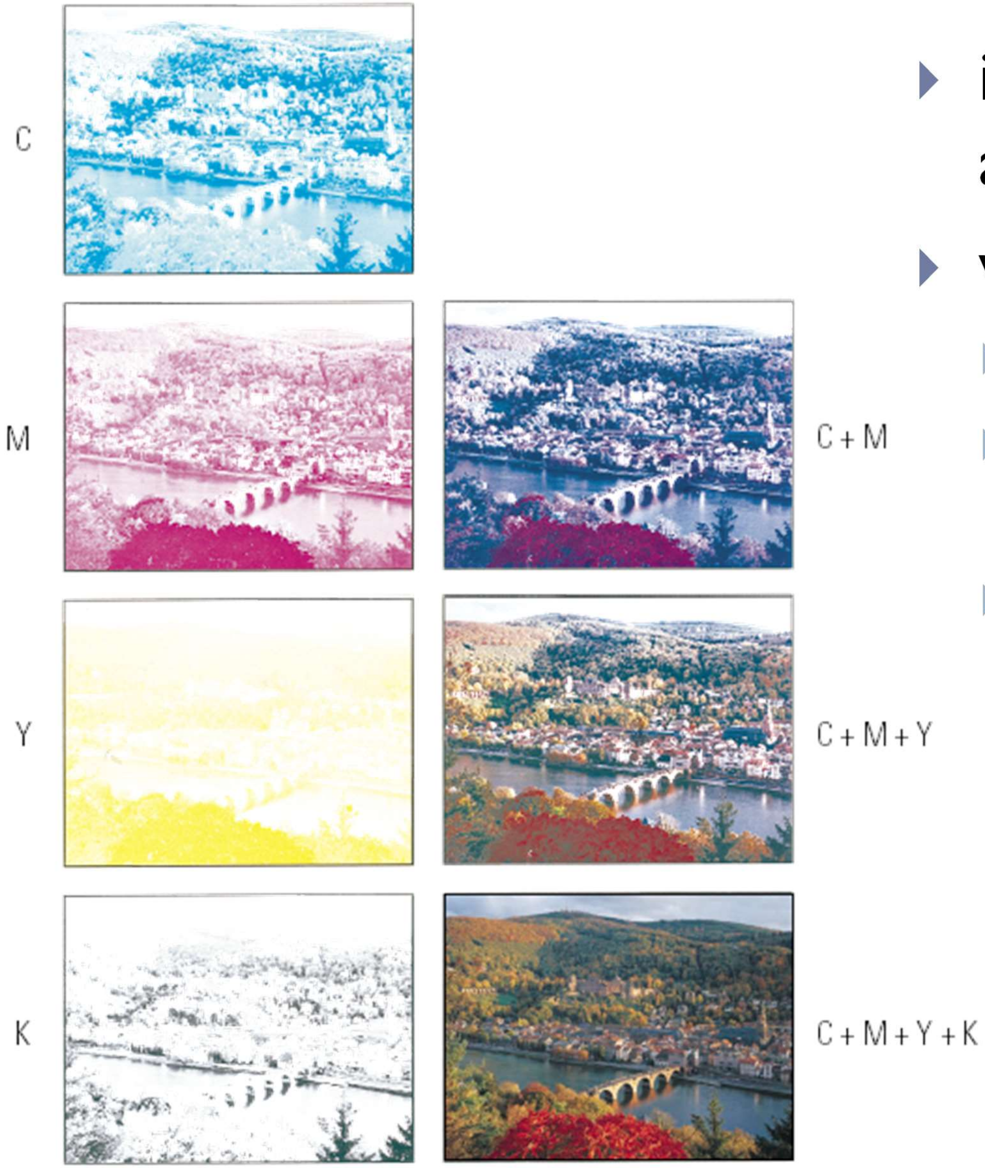


CMY space

- printers make colour by mixing coloured inks
- the important difference between inks (*CMY*) and lights (*RGB*) is that, while lights *emit* light, inks *absorb* light
  - cyan absorbs red, reflects blue and green
  - magenta absorbs green, reflects red and blue
  - yellow absorbs blue, reflects green and red
- *CMY* is, at its simplest, the inverse of *RGB*
- *CMY* space is nominally a cube



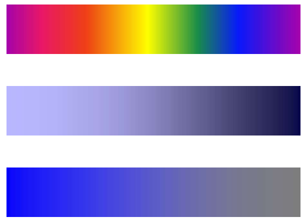
CMYK space

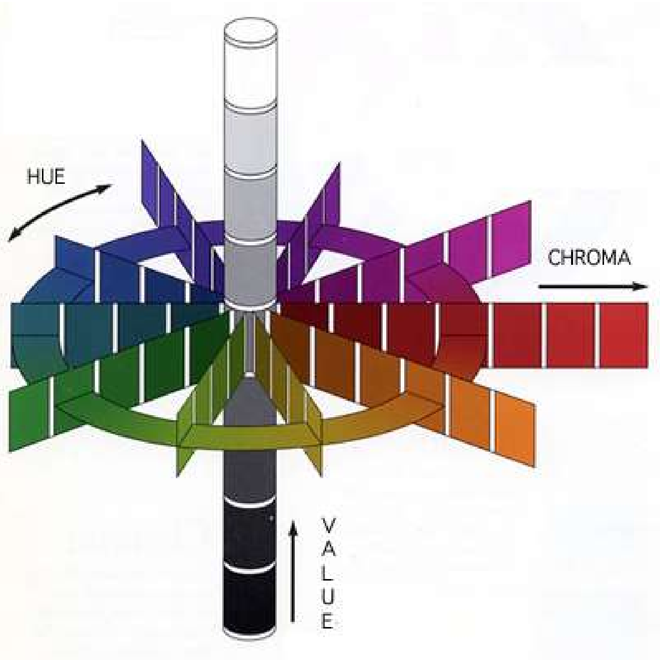


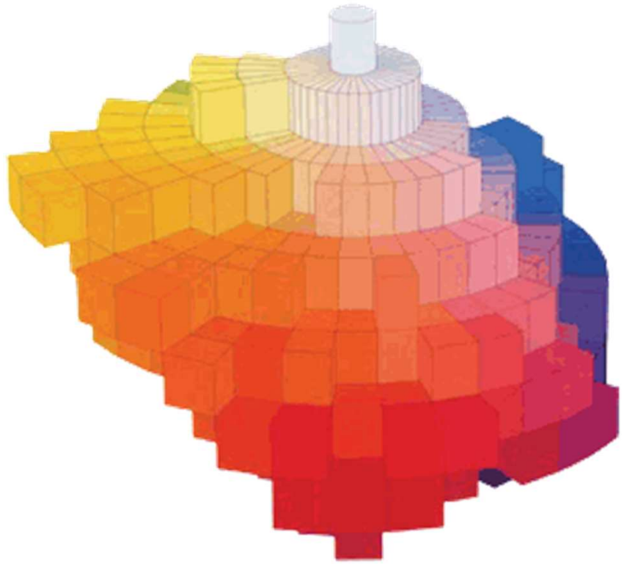
- in real printing we use black (key) as well as *CMY*
- why use black?
  - inks are not perfect absorbers
  - mixing $C + M + Y$ gives a muddy grey, not black
  - lots of text is printed in black: trying to align C, M and Y perfectly for black text would be a nightmare

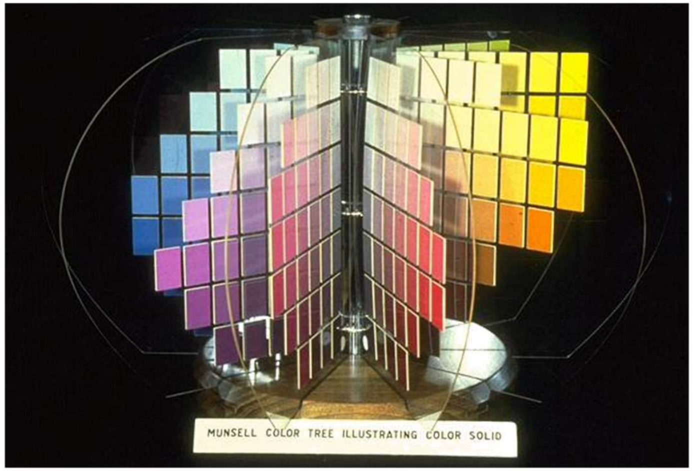
Munsell's colour classification system

- three axes
  - hue ➢ the dominant colour
  - value ➢ bright colours/dark colours
  - chroma ➢ vivid colours/dull colours
- can represent this as a 3D graph



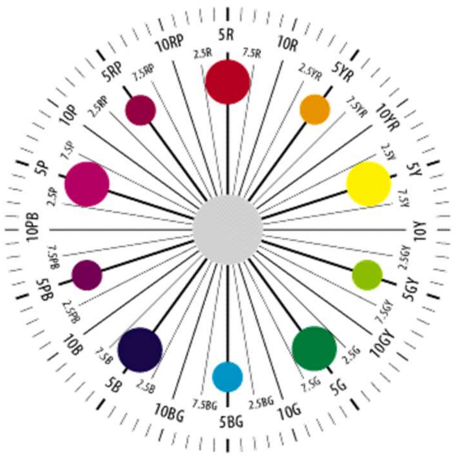


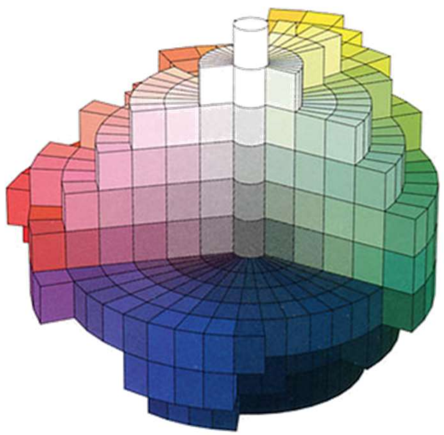


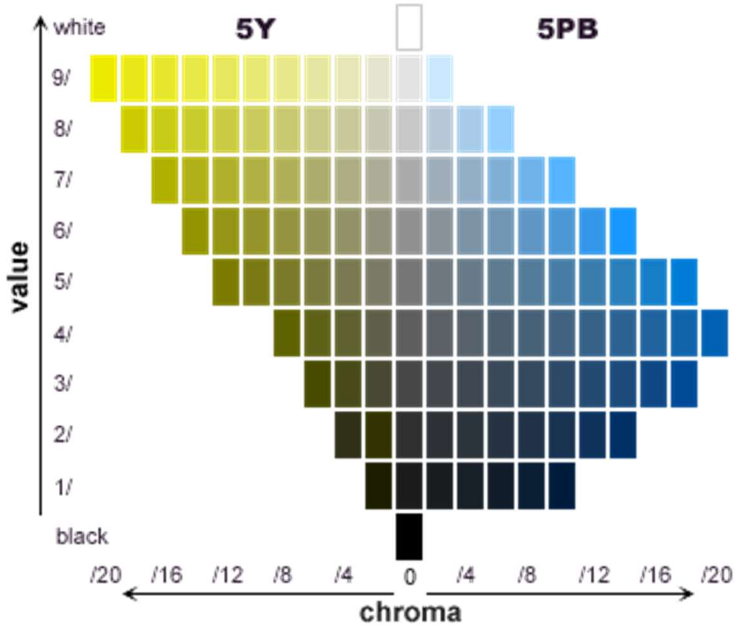


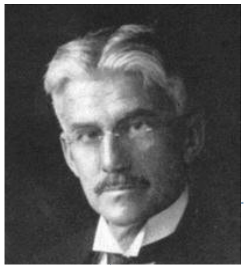
Munsell's colour classification system

- any two adjacent colours are a standard "perceptual" distance apart
  - worked out by testing it on people
  - a highly irregular space
    - e.g. vivid yellow is much brighter than vivid blue









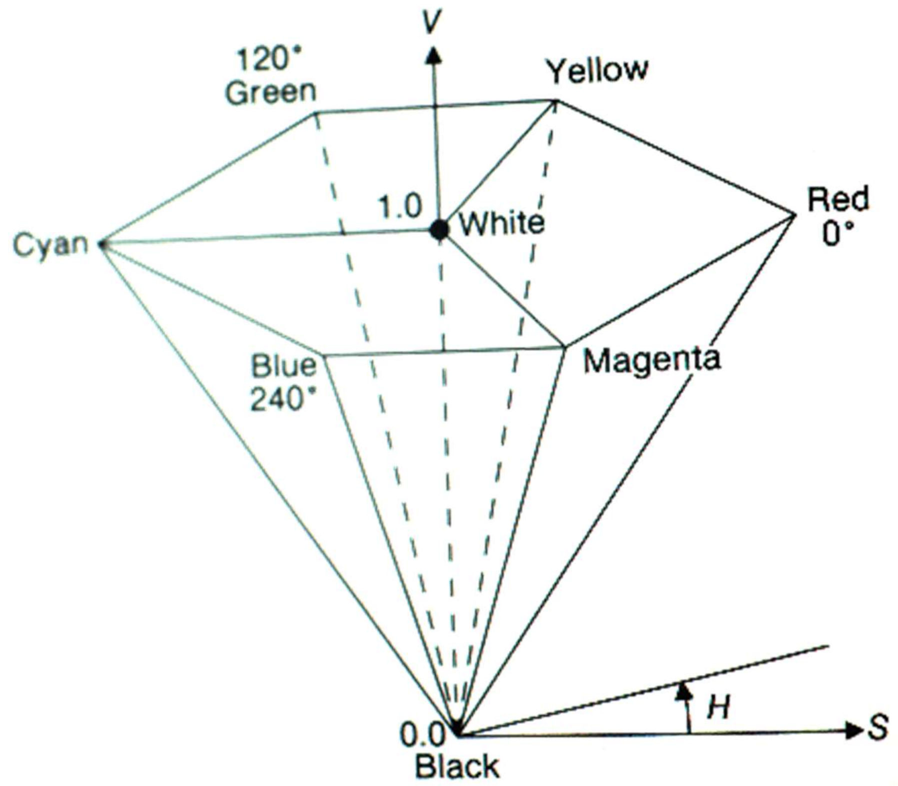
invented by Albert H. Munsell, an American artist, in 1905 in an attempt to systematically classify colours

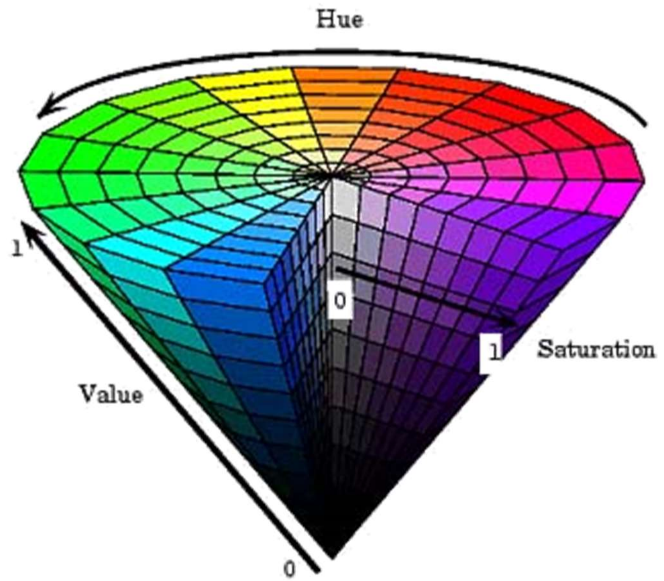
Colour spaces for user-interfaces

- *RGB* and *CMY* are based on the physical devices which produce the coloured output
- *RGB* and *CMY* are difficult for humans to use for selecting colours
- Munsell's colour system is much more intuitive:
  - hue — what is the principal colour?
  - value — how light or dark is it?
  - chroma — how vivid or dull is it?
- computer interface designers have developed basic transformations of *RGB* which resemble Munsell's human-friendly system

HSV: hue saturation value

- three axes, as with Munsell
  - hue and value have same meaning
  - the term "saturation" replaces the term "chroma"
  - simple conversion from gamma-corrected RGB to HSV

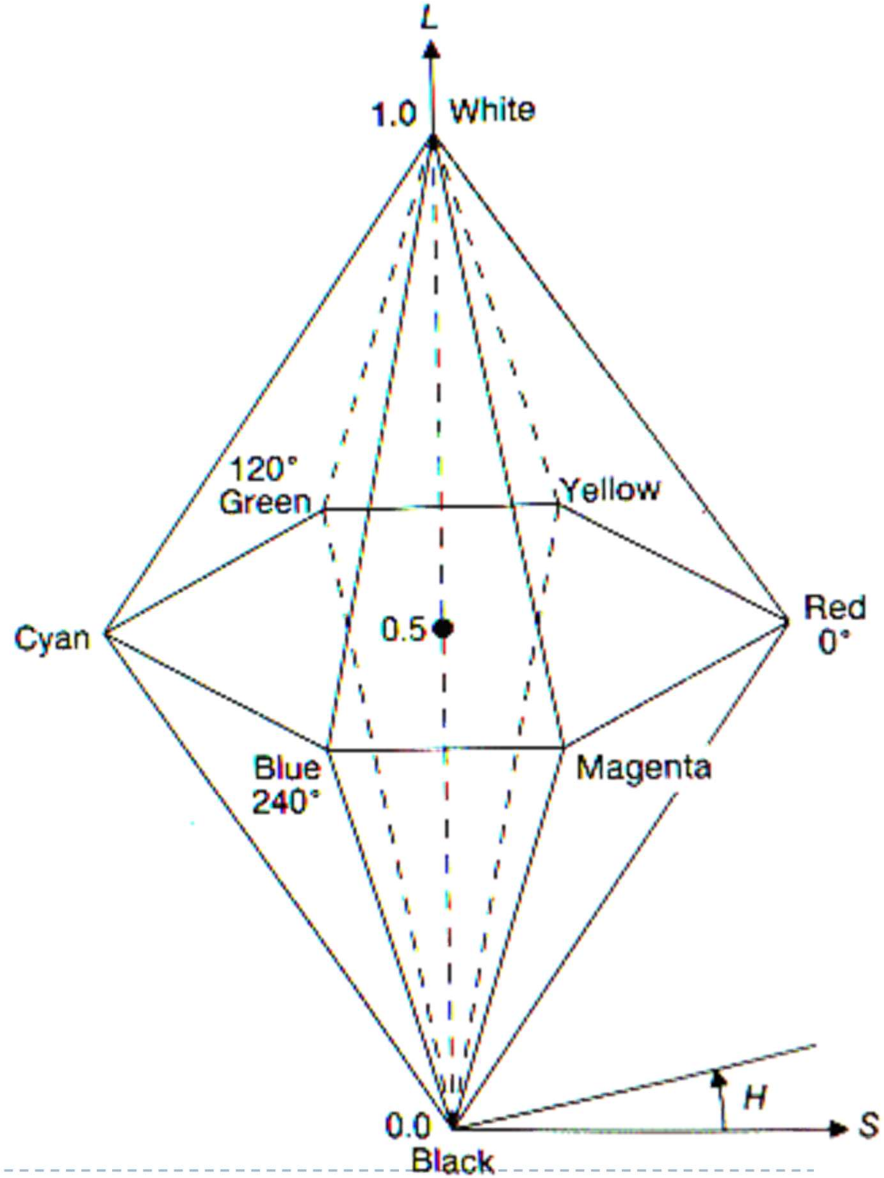




- designed by Alvy Ray Smith in 1978
- algorithm to convert *HSV* to *RGB* and back can be found in Foley et al., Figs 13.33 and 13.34

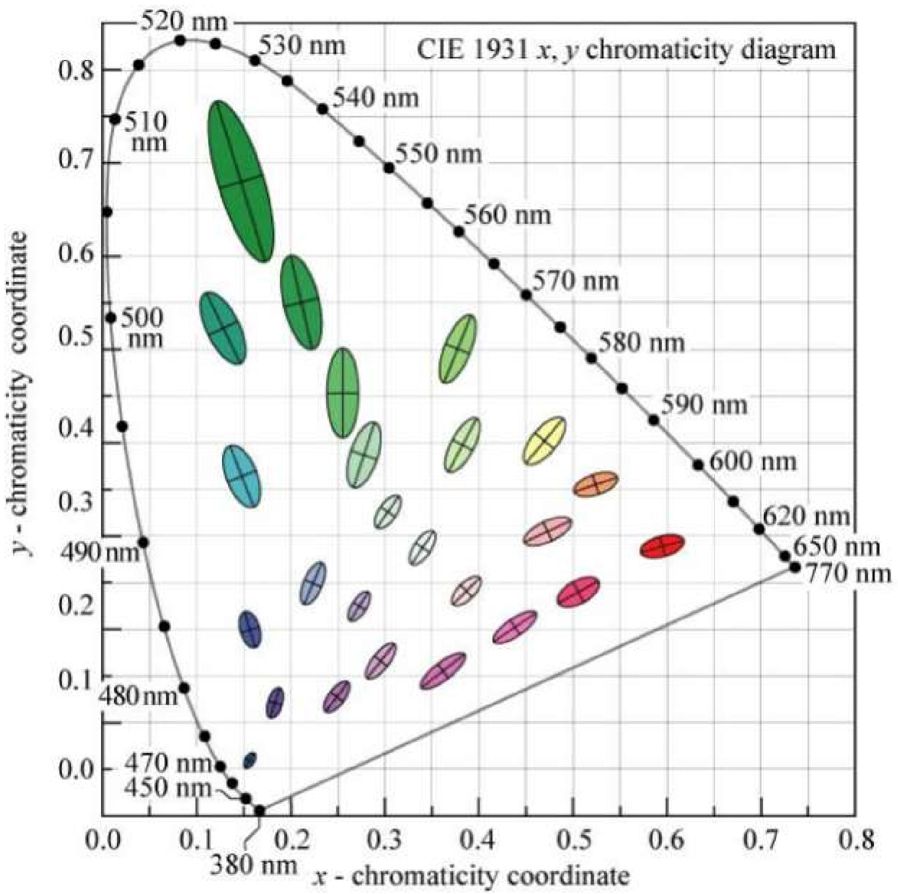
HLS: hue lightness saturation

- a simple variation of *HSV*
  - hue and saturation have same meaning
  - the term "lightness" replaces the term "value"
- designed to address the complaint that *HSV* has all pure colours having the same lightness/value as white
  - designed by Metrick in 1979
  - algorithm to convert *HLS* to *RGB* and back can be found in Foley et al., Figs 13.36 and 13.37

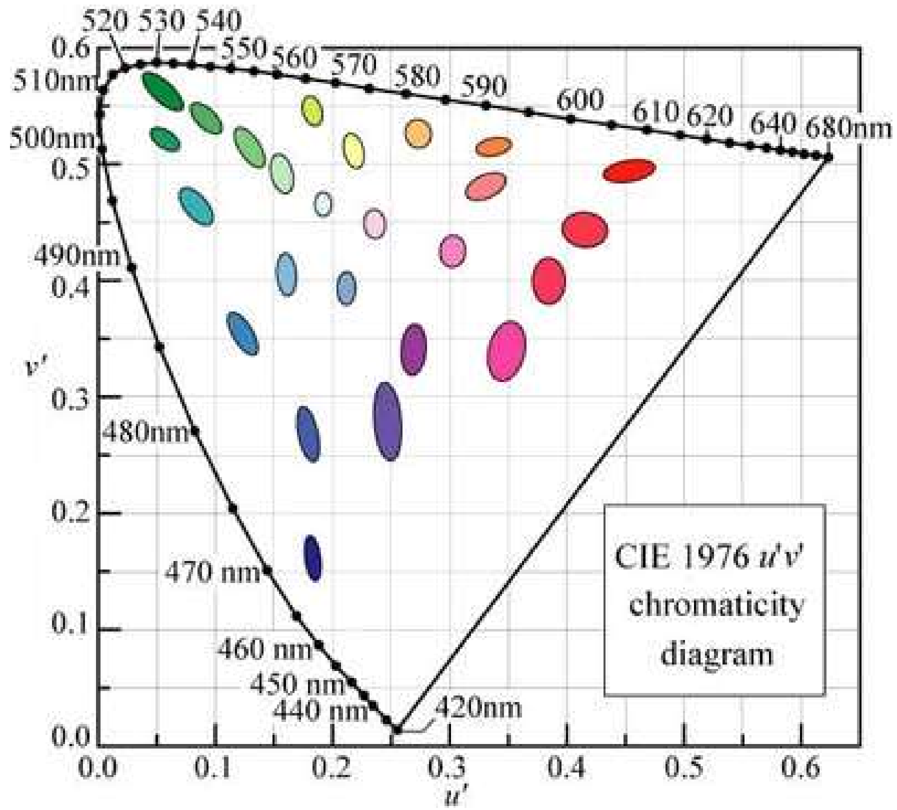


Perceptual uniformity

- MacAdam ellipses & visually indistinguishable colours



In CIE xy chromatic coordinates



In CIE u'v' chromatic coordinates

CIE L*u*v* and u'v'

- Approximately perceptually uniform
- u'v' chromacity

$$u' = \frac{4X}{X + 15Y + 3Z} = \frac{4x}{-2x + 12y + 3}$$
$$v' = \frac{9Y}{X + 15Y + 3Z} = \frac{9y}{-2x + 12y + 3}$$

- CIE LUV

Lightness

$$L^* = \begin{cases} \left(\frac{29}{3}\right)^3 Y/Y_n, & Y/Y_n \le \left(\frac{6}{29}\right)^3 \\ 116(Y/Y_n)^{1/3} - 16, & Y/Y_n > \left(\frac{6}{29}\right)^3 \end{cases}$$

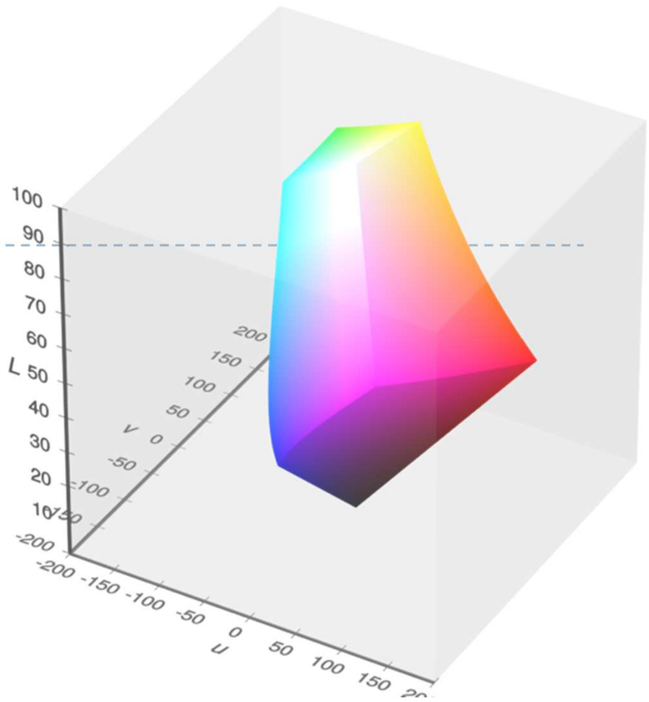
Chromacity coordinates

$$u^* = 13L^* \cdot (u' - u'_n)$$
$$v^* = 13L^* \cdot (v' - v'_n)$$

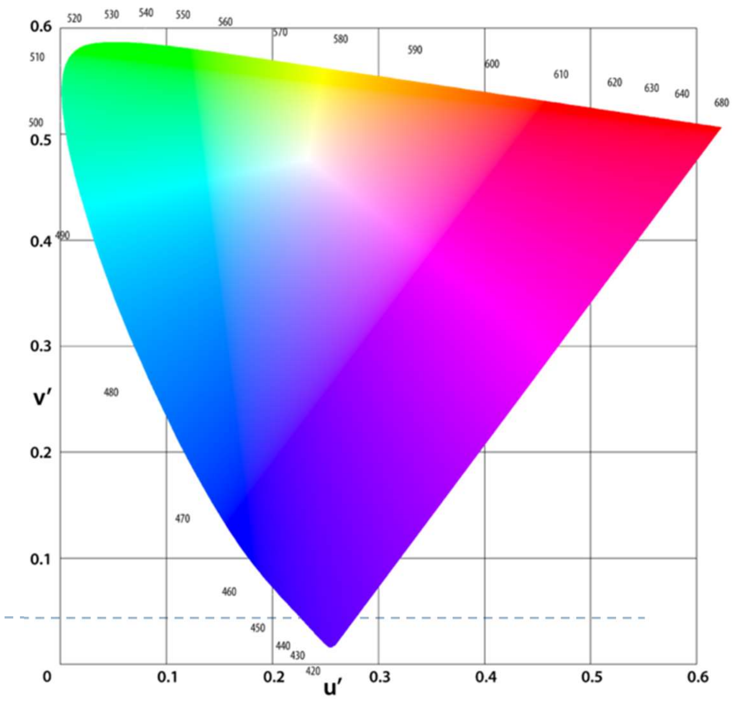
Colours less distinguishable when dark

- Hue and chroma

$$C^*_{uv} = \sqrt{(u^*)^2 + (v^*)^2}$$
$$h_{uv} = \operatorname{atan2}(v^*, u^*),$$



sRGB in CIE L*u*v*



CIE L*a*b* colour space

- Another approximately perceptually uniform colour space

$$L^\star = 116 f\left(\frac{Y}{Y_n}\right) - 16$$

$$a^\star = 500\left(f\left(\frac{X}{X_n}\right) - f\left(\frac{Y}{Y_n}\right)\right)$$

$$b^\star = 200\left(f\left(\frac{Y}{Y_n}\right) - f\left(\frac{Z}{Z_n}\right)\right)$$

Trichromatic values of the white point, e.g.

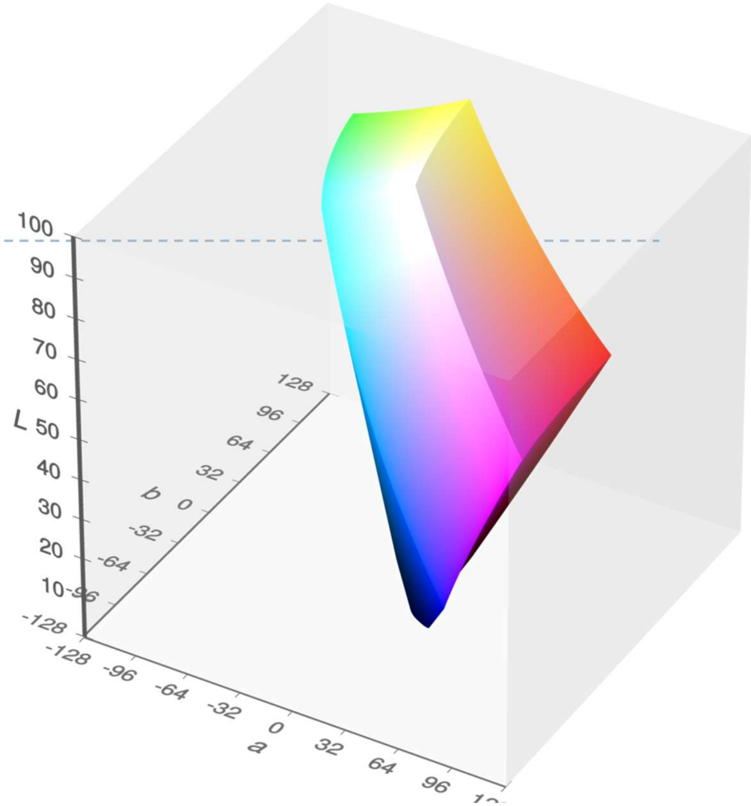
$$X_n = 95.047,$$
$$Y_n = 100.000,$$
$$Z_n = 108.883$$

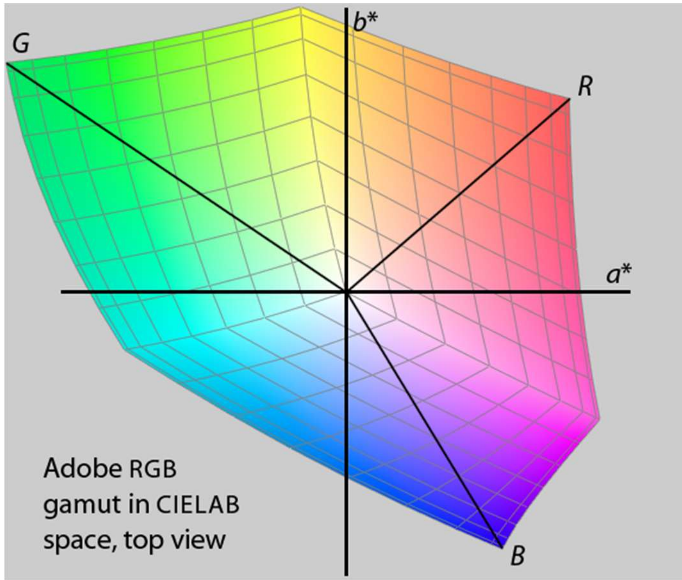
$$f(t) = \begin{cases} \sqrt[3]{t} & \text{if } t > \delta^3 \\ \frac{t}{3\delta^2} + \frac{4}{29} & \text{otherwise} \end{cases}$$

$$\delta = \frac{6}{29}$$

- Chroma and hue

$$C^\star = \sqrt{a^{\star 2} + b^{\star 2}}, \qquad h^\circ = \arctan\left(\frac{b^\star}{a^\star}\right)$$





Adobe RGB gamut in CIELAB space, top view

Lab space

- this visualization shows those colours in *Lab* space which a human can perceive
- again we see that human perception of colour is not uniform
  - perception of colour diminishes at the white and black ends of the L axis
  - the maximum perceivable chroma differs for different hues

Colour - references

- Chapters „Light” and „Colour” in
  - Shirley, P. & Marschner, S., *Fundamentals of Computer Graphics*
- Textbook on colour appearance
  - Fairchild, M. D. (2005). *Color Appearance Models* (second.). John Wiley & Sons.
- Comprehensive review of colour research
  - Wyszecki, G., & Stiles, W. S. (2000). *Color science: concepts and methods, quantitative data, and formulae* (Second ed.). John Wiley & Sons.

UNIVERSITY OF CAMBRIDGE
COMPUTER LABORATORY

RAINBOW GRAPHICS GROUP

Advanced Graphics and Image Processing

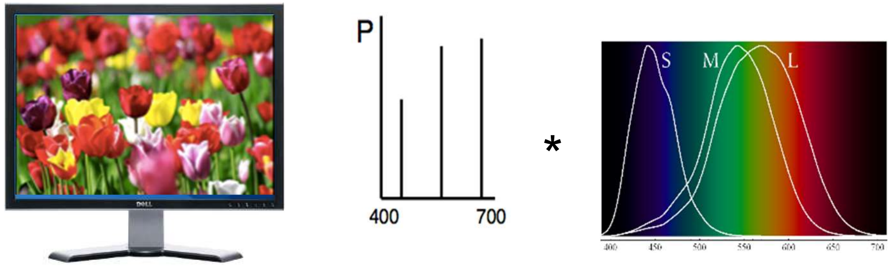
Models of early visual perception

Part 1/6 – perceived brightness of light

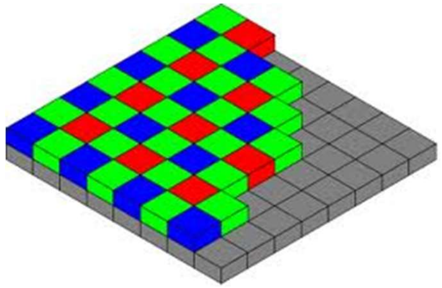
Rafal Mantiuk
Computer Laboratory, University of Cambridge

Many graphics/display solutions are motivated by visual perception

Image & video compression

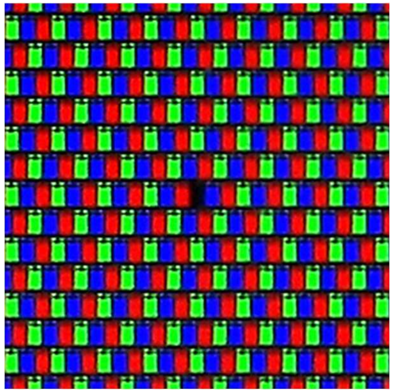


Display spectral emission - metamerism

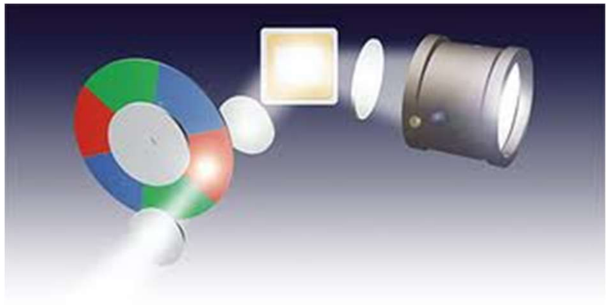


Camera's Bayer pattern

Halftonning



Display's subpixels



Color wheel in DLPs

. . .

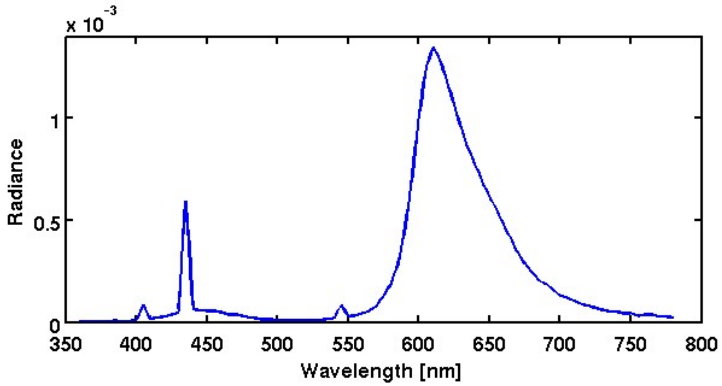
Luminance (again)

- Luminance – measure of light weighted by the response of the achromatic mechanism. Units: cd/m^2

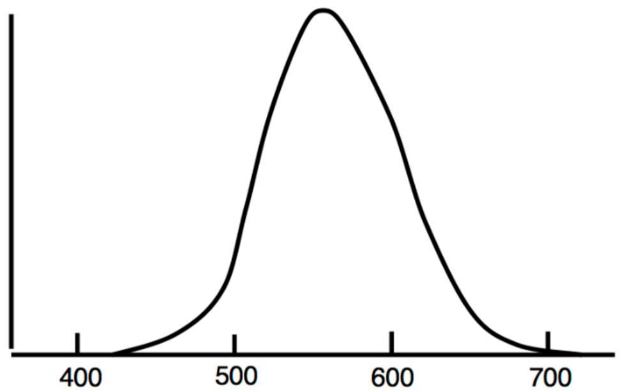
Luminance

$$L_V = \int_{350}^{700} kL(\lambda)V(\lambda)d\lambda \qquad k = \frac{1}{683.002}$$

Light spectrum (radiance)



Luminous efficiency function (weighting)



Steven's power law for brightness

- Stevens (1906-1973) measured the perceived magnitude of physical stimuli
  - Loudness of sound, tastes, smell, warmth, electric shock and brightness
  - Using the magnitude estimation methods
    - Ask to rate loudness on a scale with a known reference
- All measured stimuli followed the power law:

Perceived magnitude

$$\varphi(I) = kI^{a}$$

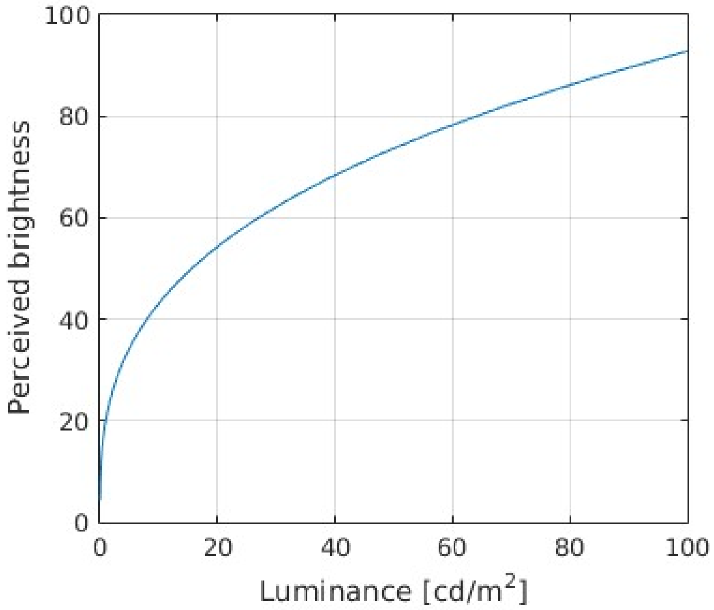
Exponent

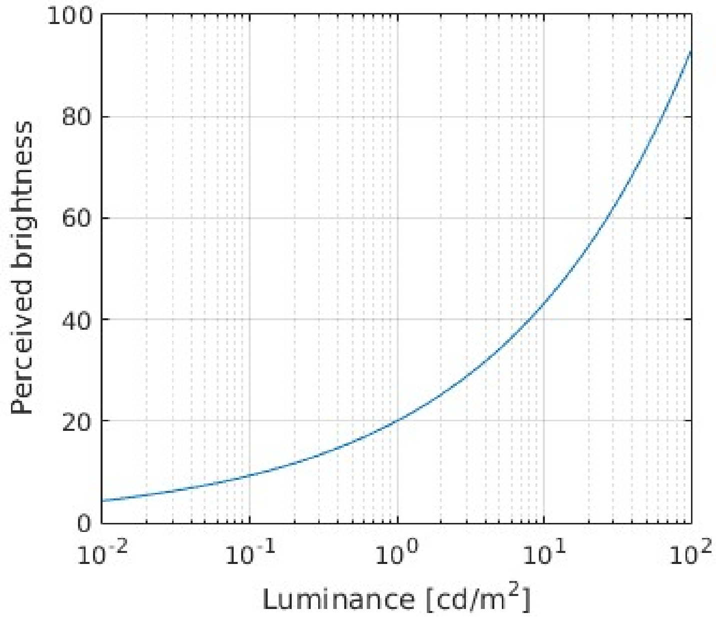
Constant

Physical stimulus

- For brightness (5 deg target in dark), a = 0.3

Steven's law for brightness

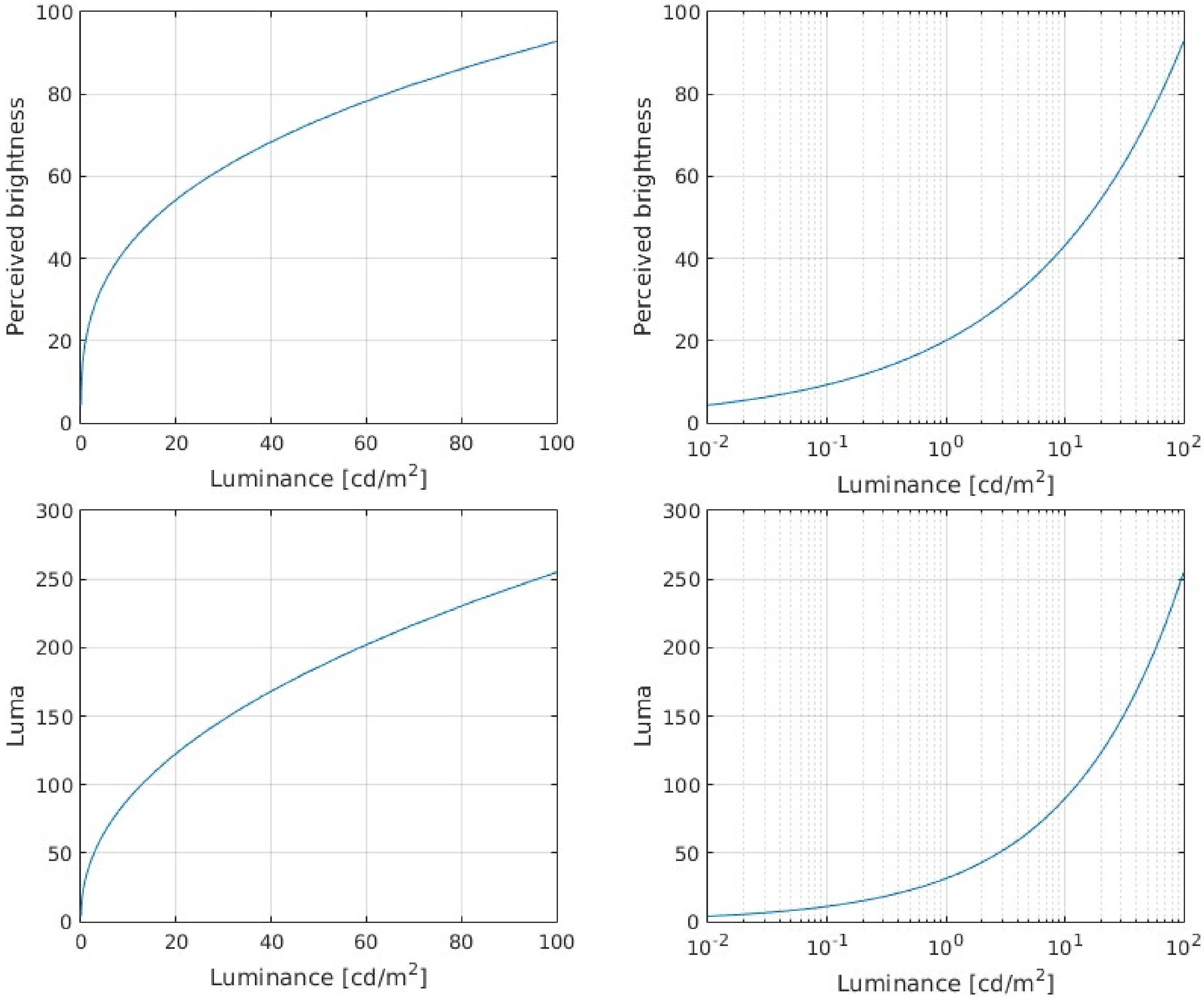




Steven's law vs. Gamma correction

Stevens' law
a=0.3

Gamma function
Gamma = 2.2



Advanced Graphics and Image Processing

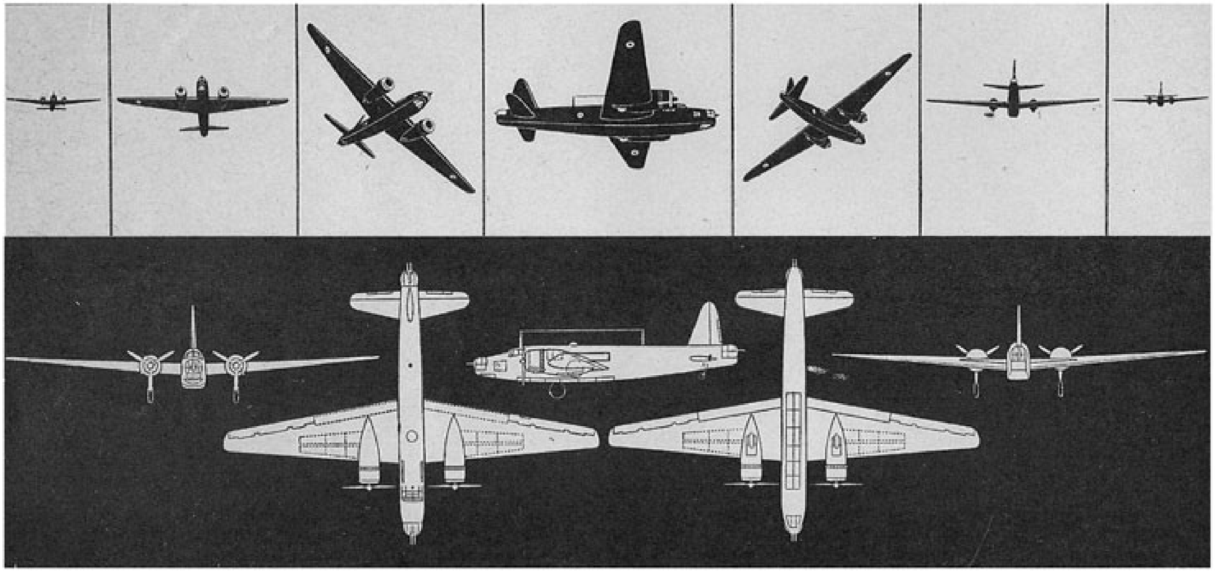
Models of early visual perception

Part 2/6 – contrast detection

Rafal Mantiuk

Computer Laboratory, University of Cambridge

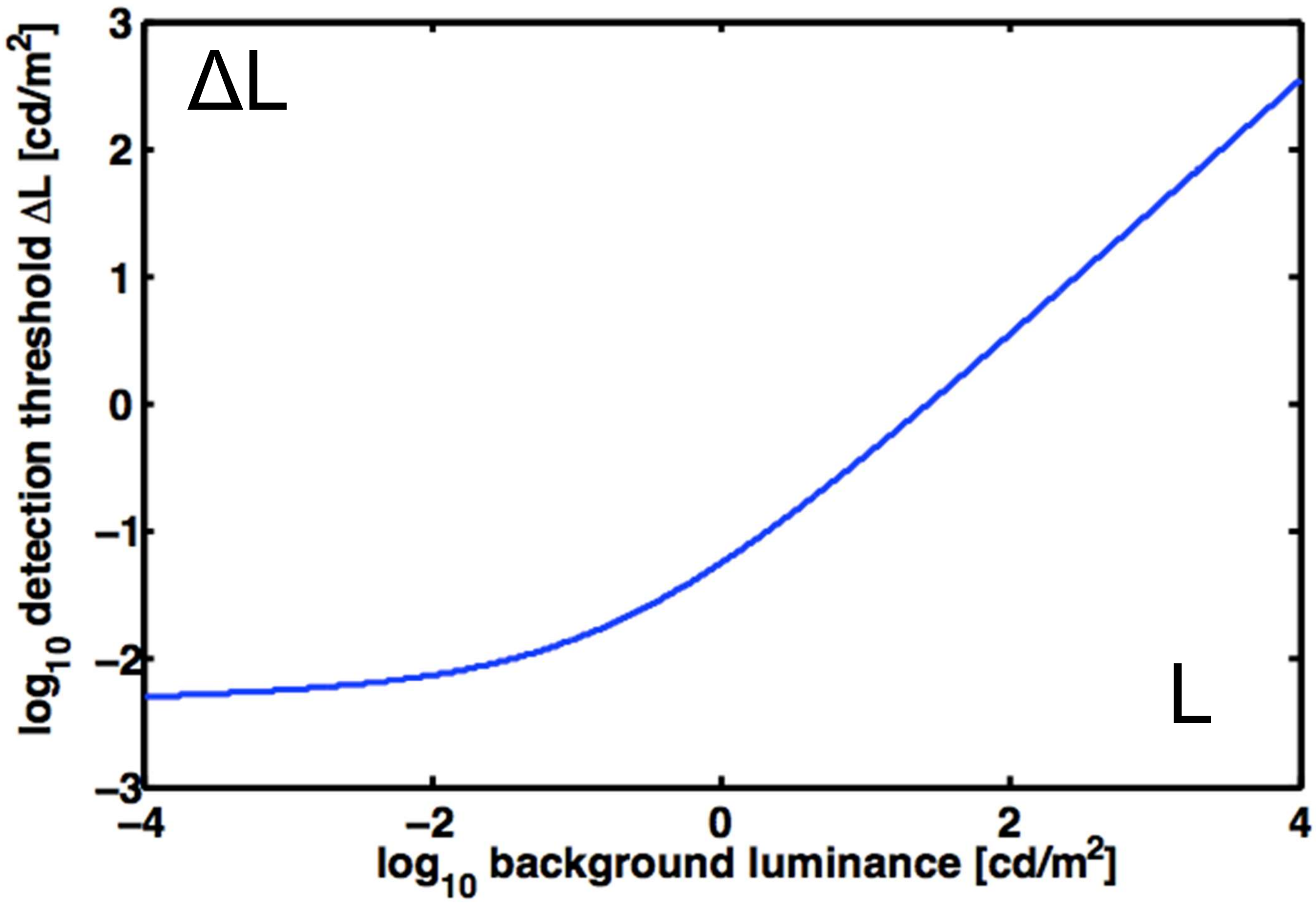
Detection thresholds

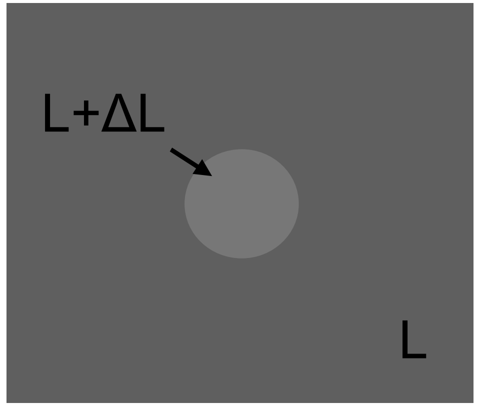


- The smallest detectable difference between
  - the luminance of the object and
  - the luminance of the background

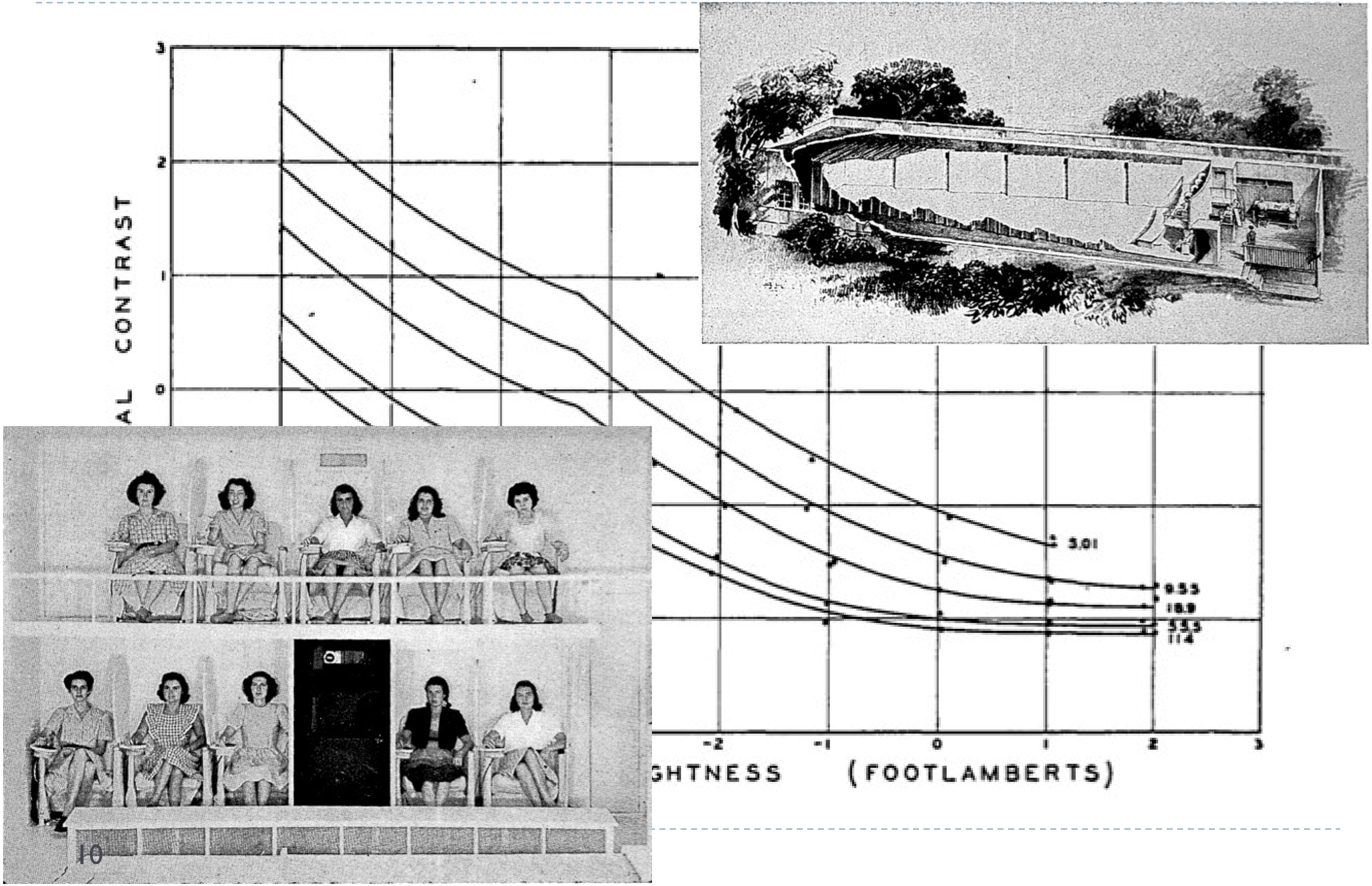
Threshold versus intensity (t.v.i.) function

- The smallest detectable difference in luminance for a given background luminance

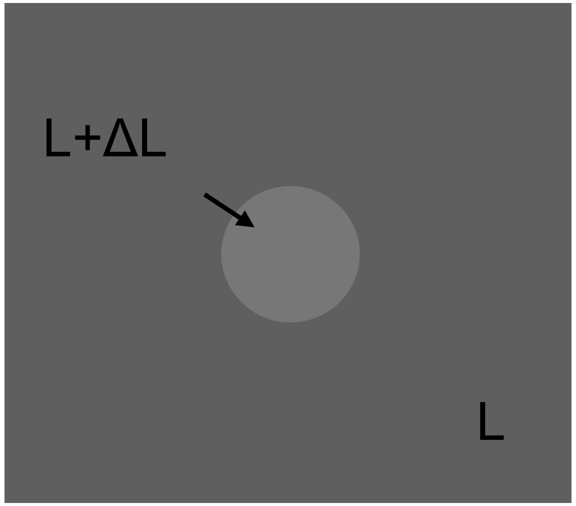


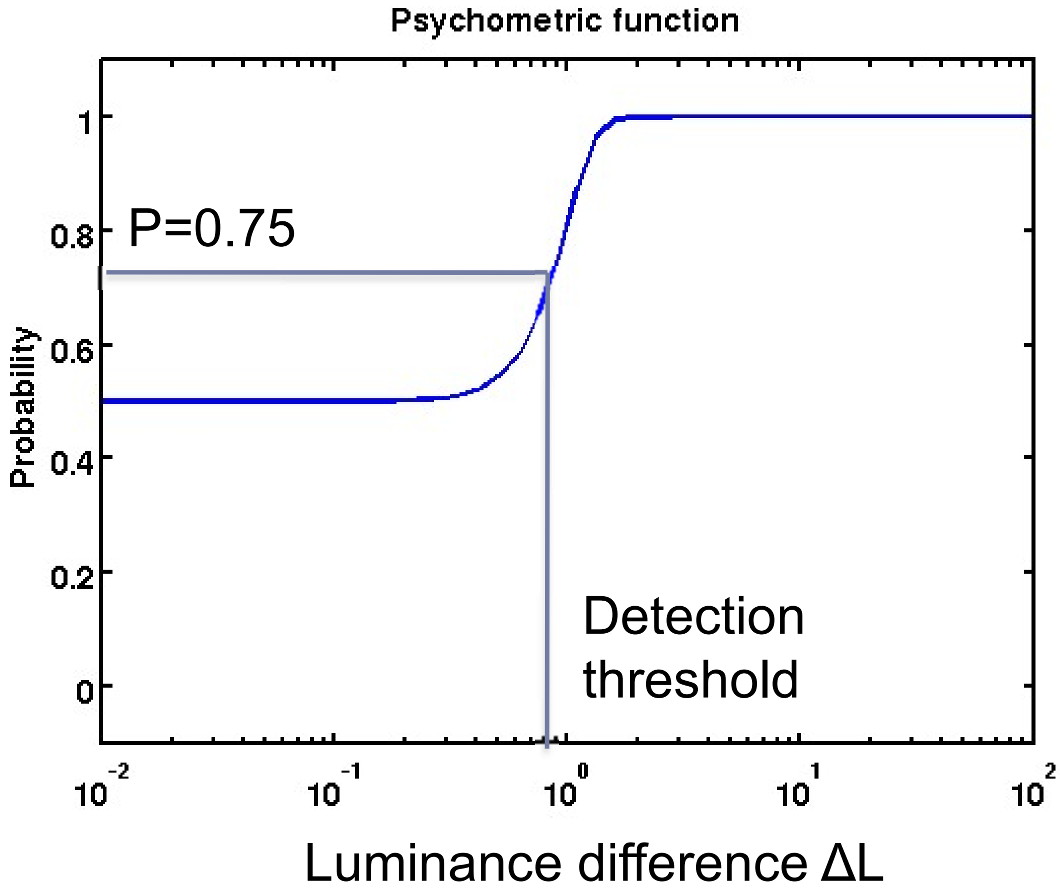


t.v.i. measurements – Blackwell 1946



Psychophysics
Threshold experiments

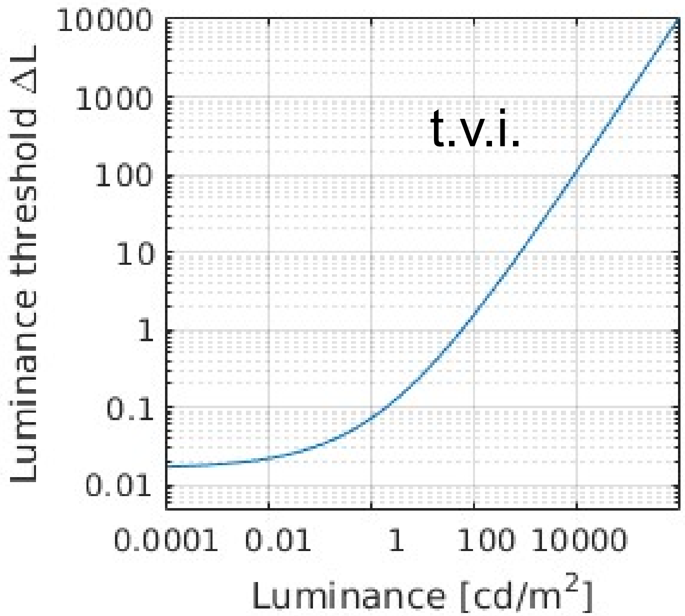




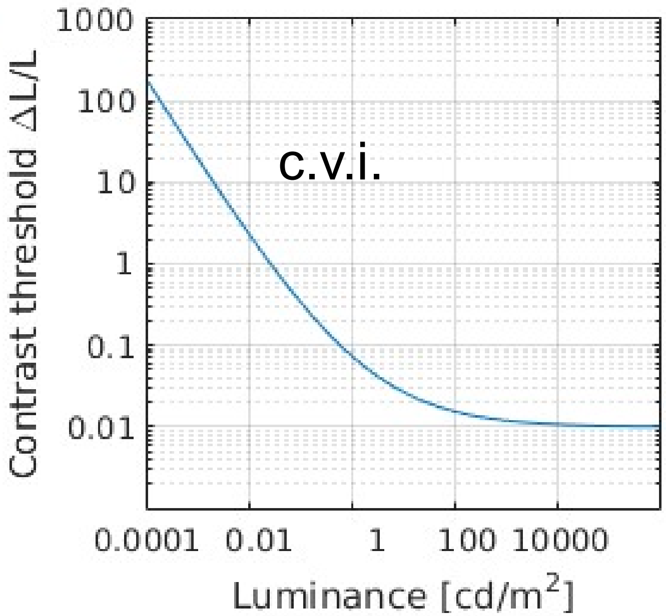
t.v.i function / c.v.i. function / Sensitivity

- The same data, different representation

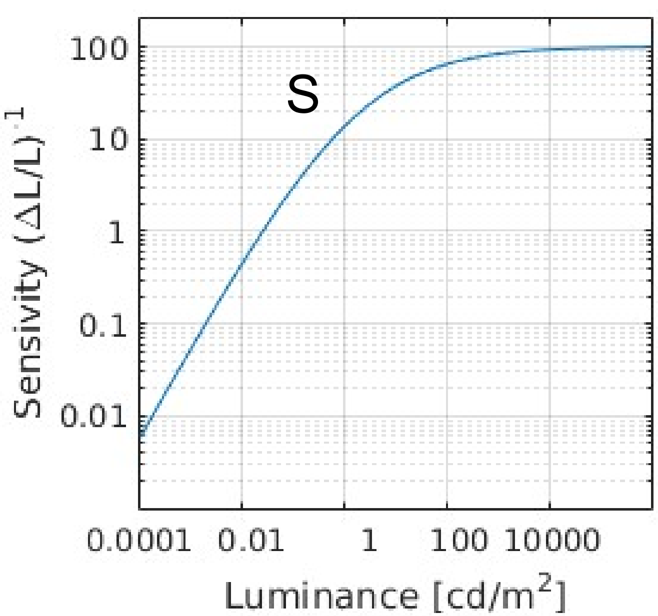
Threshold vs. intensity



Contrast vs. intensity



Sensitivity



$$\Delta L = L_{disk} - L_{background}$$

$$T = \frac{\Delta L}{L}$$

$$S = \frac{1}{T} = \frac{L}{\Delta L}$$

Sensitivity to luminance

- Weber-law – the just-noticeable difference is proportional to the magnitude of a stimulus

Ernst Heinrich Weber
[From wikipedia]

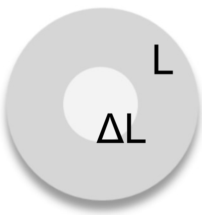
The smallest detectable luminance difference

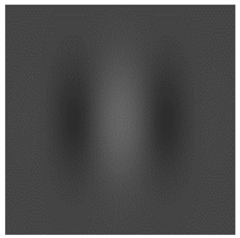
Background (adapting) luminance

$$\frac{\Delta L}{L} = k$$

Constant

Typical stimuli:





Consequence of the Weber-law

- Smallest detectable difference in luminance

$$\frac{\Delta L}{L} = k$$

For k=1%

L	ΔL
100 cd/m^2	1 cd/m^2
1 cd/m^2	0.01 cd/m^2

- Adding or subtracting luminance will have different visual impact depending on the background luminance
- Unlike LDR luma values, luminance values are **not** perceptually uniform!

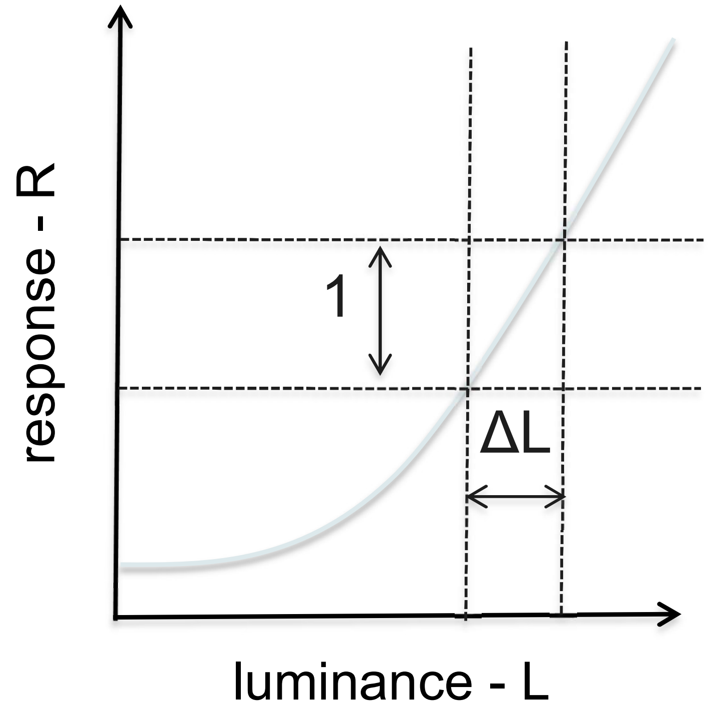
How to make luminance (more) perceptually uniform?

- Using "Fechnerian" integration

$$\frac{dR}{dl}(L) = \frac{1}{\Delta L(L)}$$

Derivative of response

Detection threshold



Luminance transducer: $$R(L) = \int_{L_{min}}^{L} \frac{1}{\Delta L(l)} dl$$

Assuming the Weber law

$$\frac{\Delta L}{L} = k$$

- and given the luminance transducer

$$R(L) = \int \frac{1}{\Delta L(l)} dl$$

- the response of the visual system to light is:

$$R(L) = \int \frac{1}{kL} dL = \frac{1}{k} \ln(L) + k_1$$

Fechner law

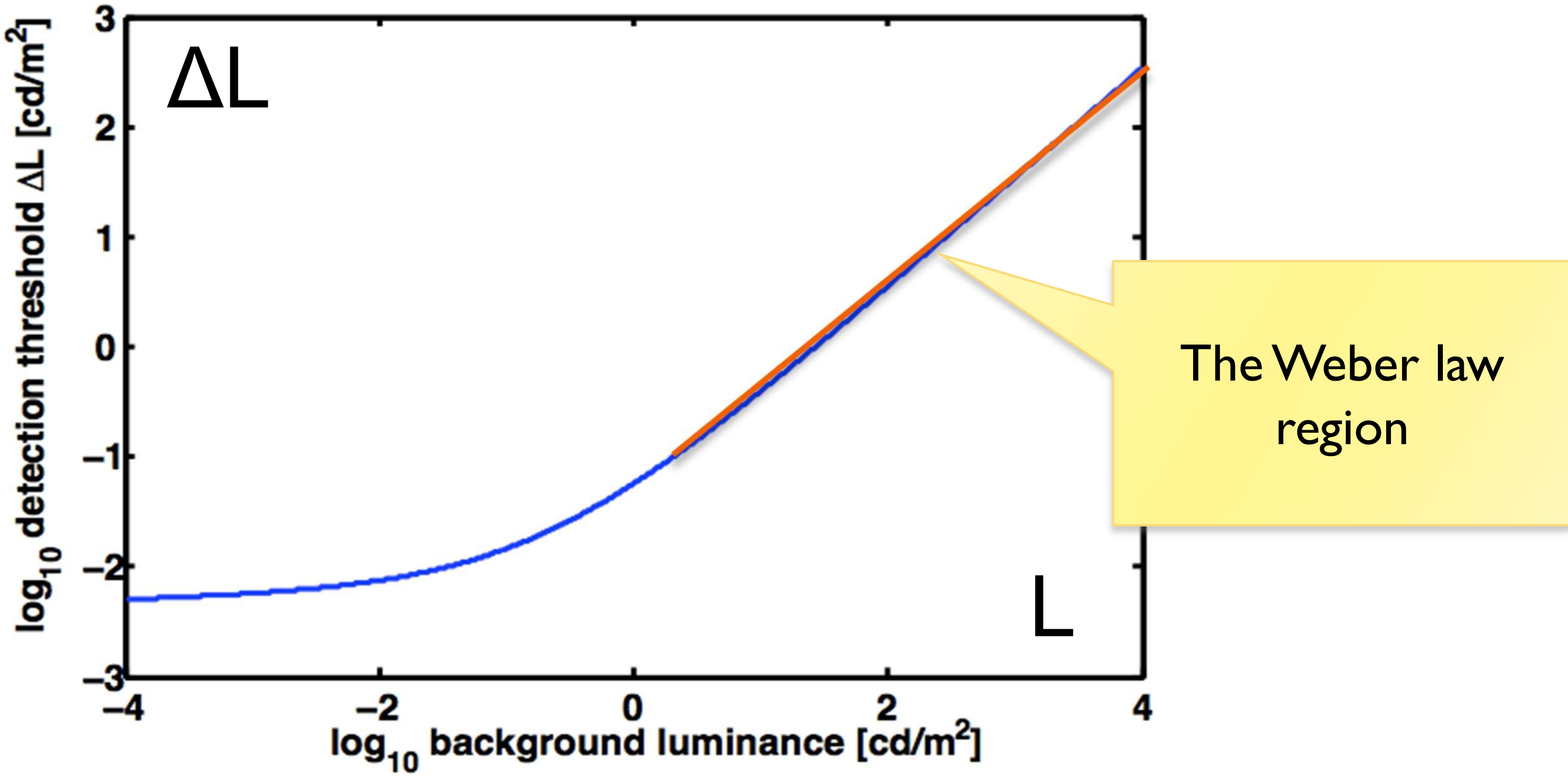
$$R(L) = a\ln(L)$$

- Response of the visual system to luminance is **approximately** logarithmic

Gustav Fechner
[From Wikipedia]

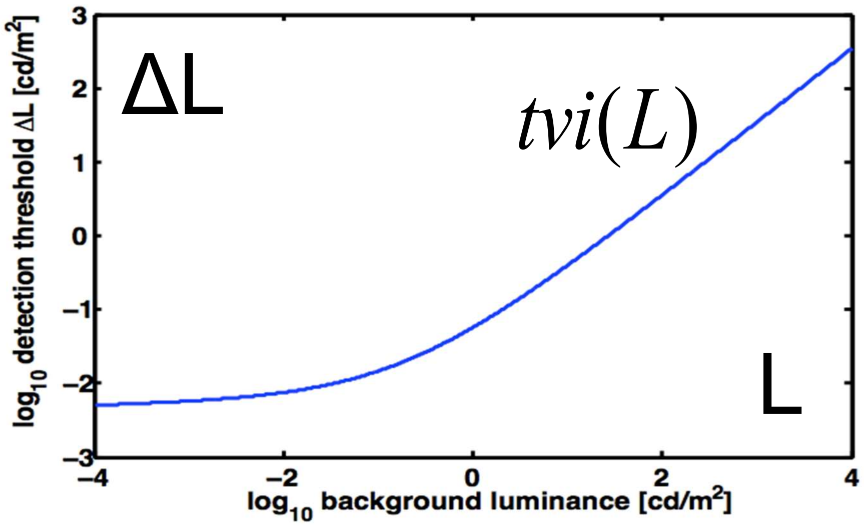
But…the Fechner law does not hold for the full luminance range

- Because the Weber law does not hold either
- Threshold vs. intensity function:



Weber-law revisited

- If we allow detection threshold to vary with luminance according to the t.v.i. function:

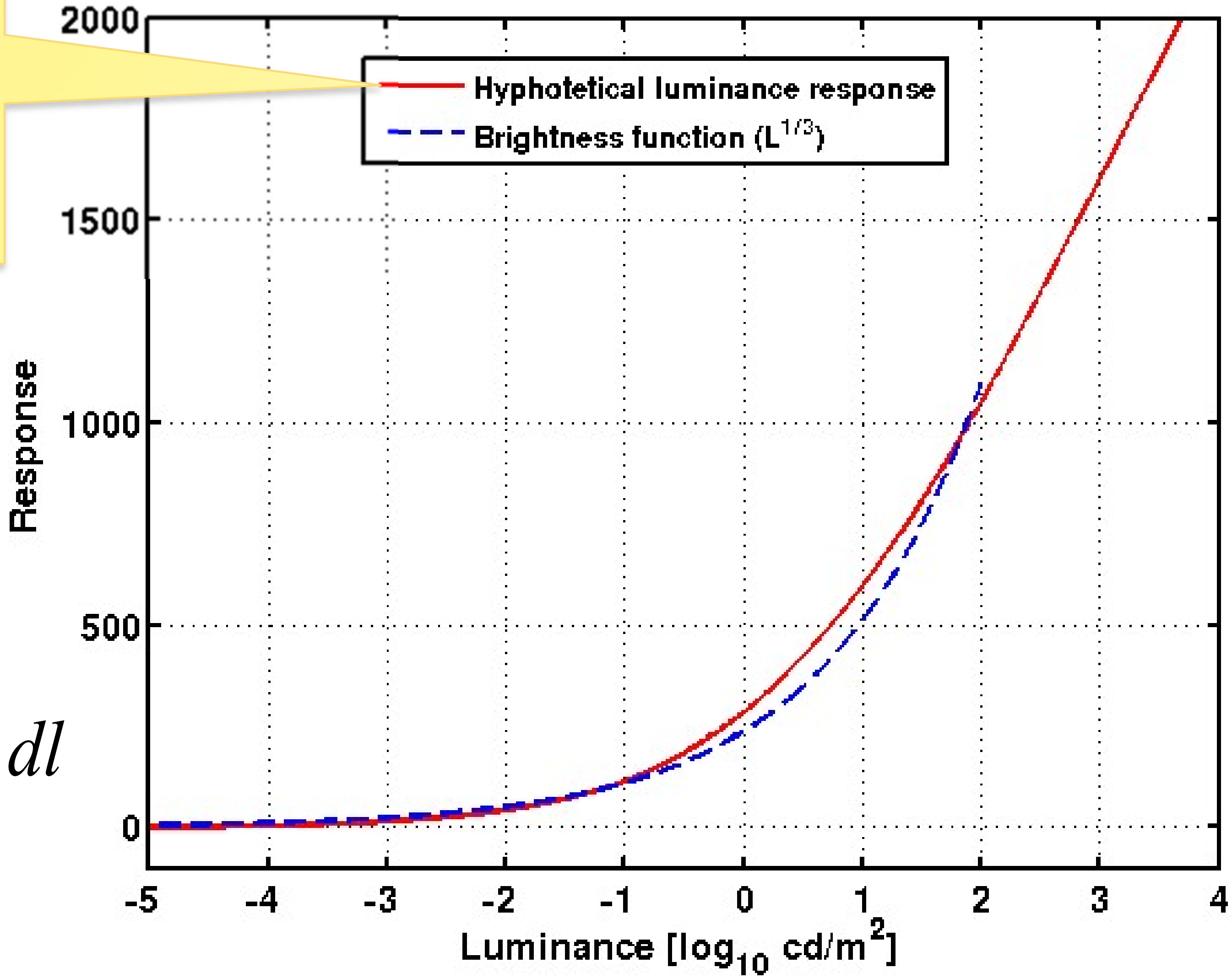


- we can get a more accurate estimate of the "response":

$$R(L) = \int_0^L \frac{1}{tvi(l)}\,dl$$

Fechnerian integration and Stevens' law

R(L) - function derived from the t.v.i. function



$$R(L) = \int_0^L \frac{1}{tvi(l)} dl$$

Applications of JND encoding – R(L)

- DICOM grayscale function
  - Function used to encode signal for medial monitors
  - 10-bit JND-scaled (just noticeable difference)
  - Equal visibility of gray levels
- HDMI 2.0a (HDR10)
  - PQ (Perceptual Quantizer) encoding
  - Dolby Vision
  - To encode pixels for high dynamic range images and video

DOLBY
VISION
The Future of Vision


© 2006 Aude Oliva & Antonio Torralba

UNIVERSITY OF CAMBRIDGE
COMPUTER LABORATORY

RAINBOW GRAPHICS GROUP

Advanced Graphics and Image Processing

Models of early visual perception

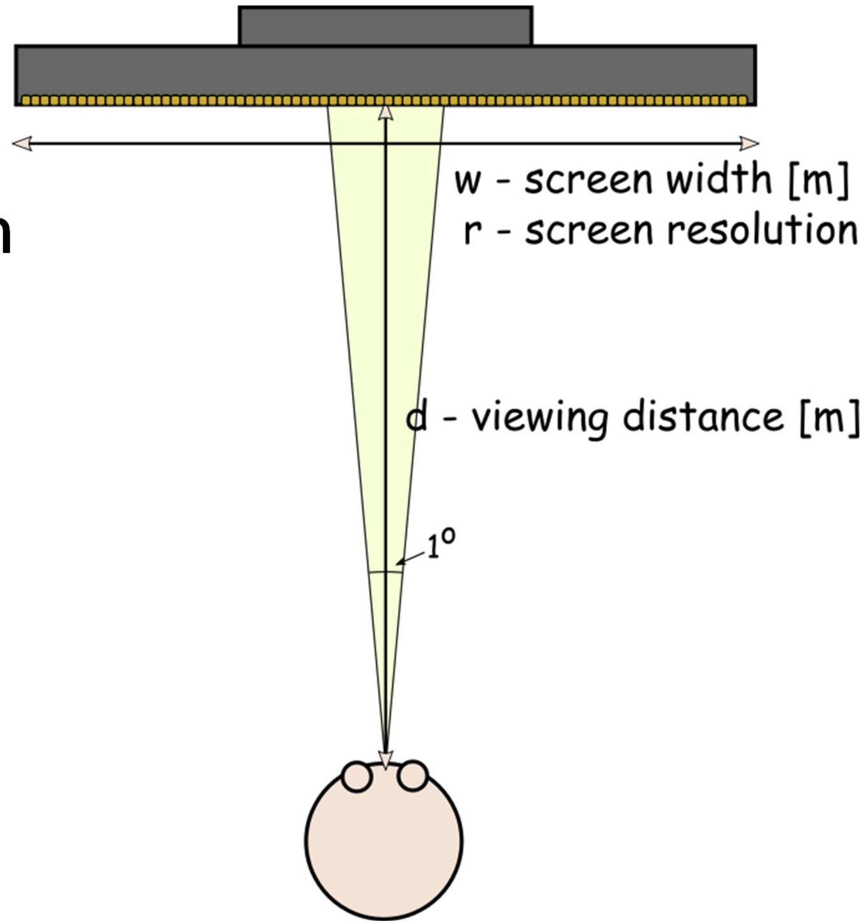
Part 3/6 – spatial contrast sensitivity and contrast constancy

Rafal Mantiuk

Computer Laboratory, University of Cambridge

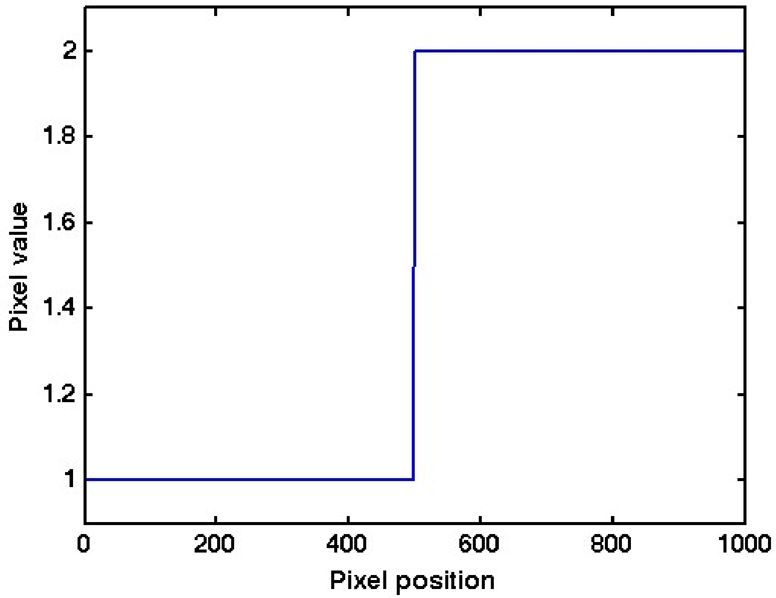
Resolution and sampling rate

- Pixels per inch [ppi]
  - Does not account for vision
- The visual resolution depends on
  - screen size
  - screen resolution
  - viewing distance
- The right measure
  - Pixels per visual degree [ppd]
  - In frequency space
    - Cycles per visual degree [cpd]

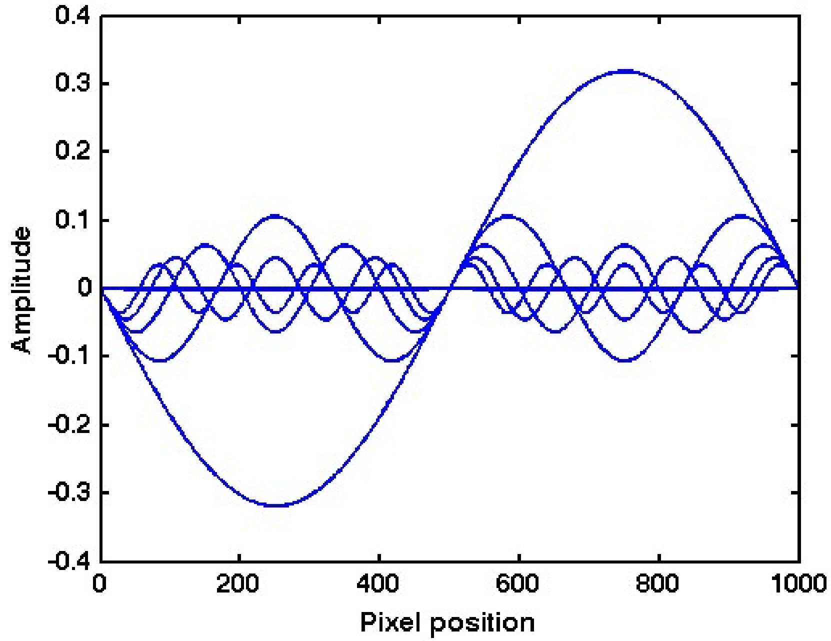


Fourier analysis

- Every N-dimensional function (including images) can be represented as a sum of sinusoidal waves of different frequency and phase



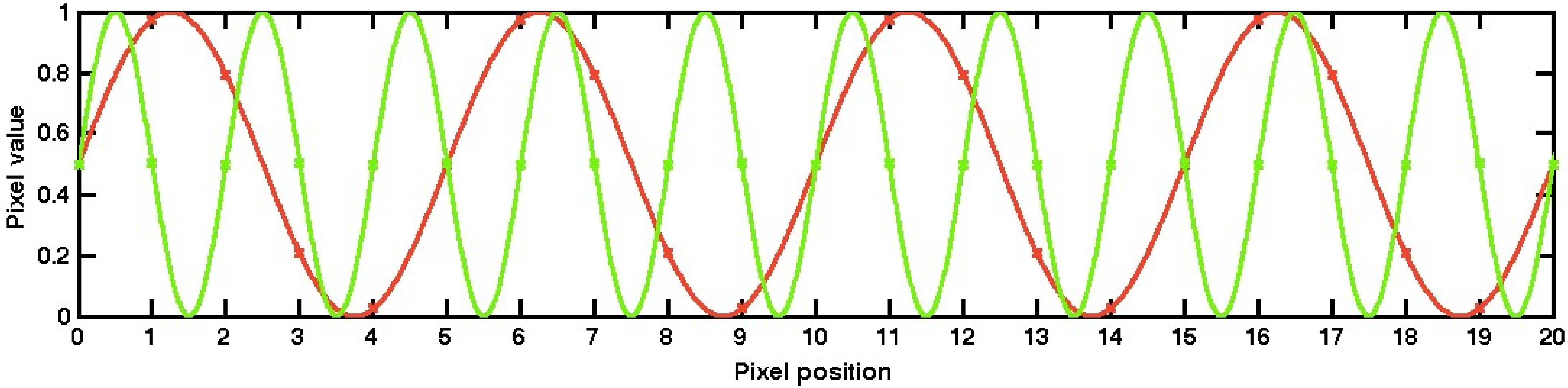
$= \sum$



- Think of “equalizer” in audio software, which manipulates each frequency

Spatial frequency in images

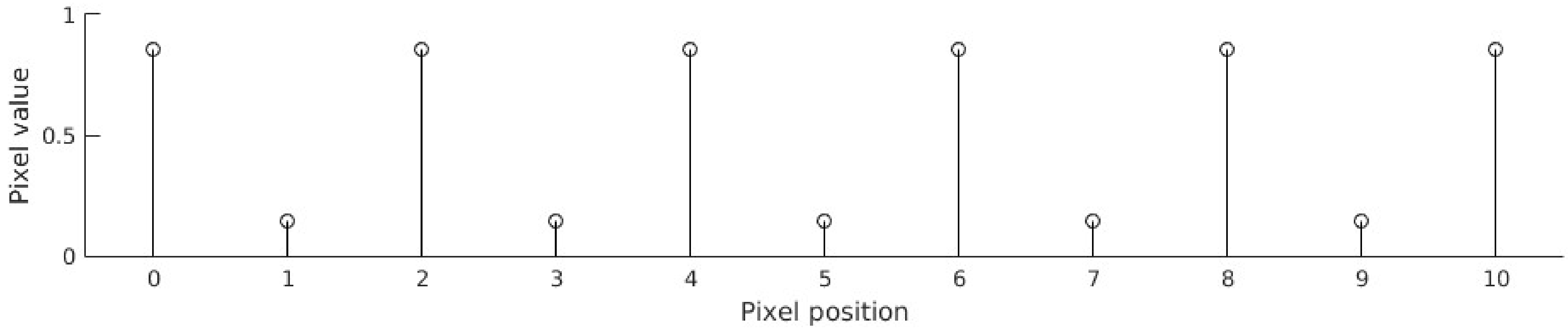
- Image space units: cycles per sample (or cycles per pixel)



  - What are the screen-space frequencies of the red and green sinusoid?
- The visual system units: cycles per degree
  - If the angular resolution of the viewed image is 55 pixels per degree, what is the frequency of the sinusoids in cycles per degree?

Nyquist frequency

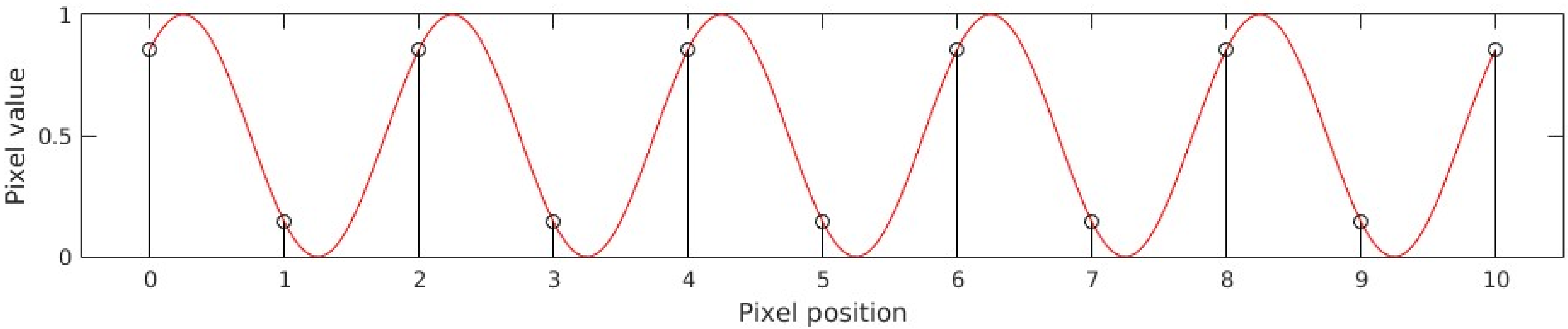
- Sampling density restricts the highest spatial frequency signal that can be (uniquely) reconstructed
  - Sampling density – how many pixels per image/visual angle/…



  - Any number of sinusoids can be fitted to this set of samples
  - It is possible to fit an infinite number of sinusoids if we allow infinitely high frequency

Nyquist frequency

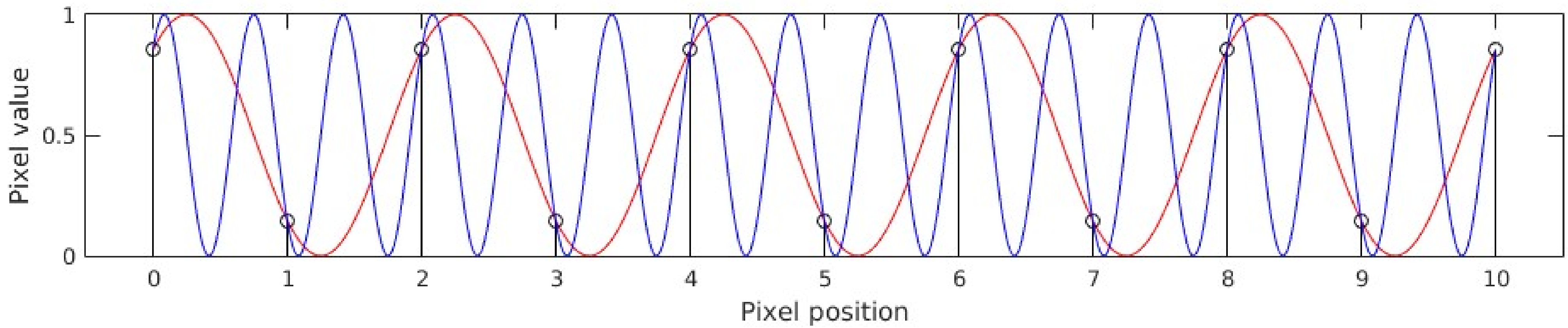
- Sampling density restricts the highest spatial frequency signal that can be (uniquely) reconstructed
  - Sampling density – how many pixels per image/visual angle/…



  - Any number of sinusoids can be fitted to this set of samples
  - It is possible to fit an infinite number of sinusoids if we allow infinitely high frequency

Nyquist frequency

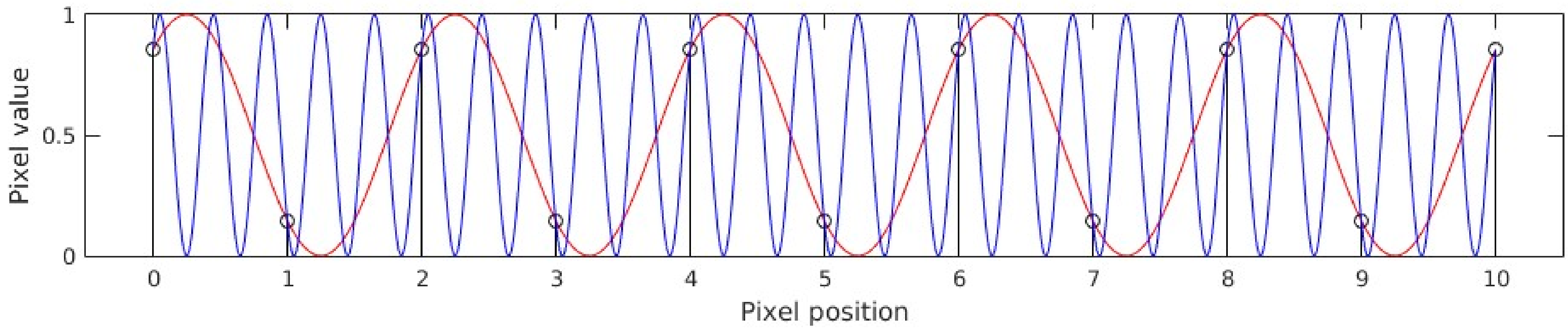
- Sampling density restricts the highest spatial frequency signal that can be (uniquely) reconstructed
  - Sampling density – how many pixels per image/visual angle/…



  - Any number of sinusoids can be fitted to this set of samples
  - It is possible to fit an infinite number of sinusoids if we allow infinitely high frequency

Nyquist frequency

- Sampling density restricts the highest spatial frequency signal that can be (uniquely) reconstructed
  - Sampling density – how many pixels per image/visual angle/…

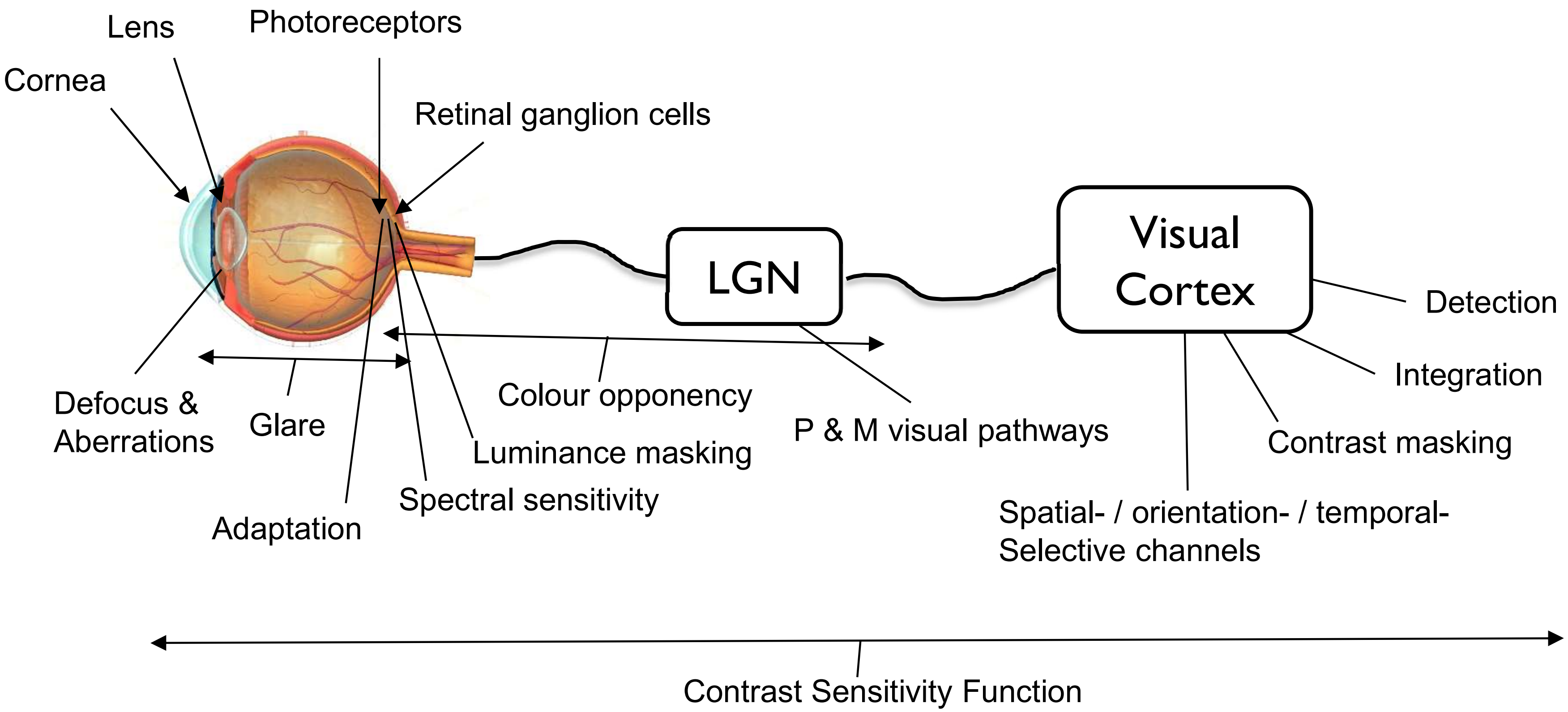


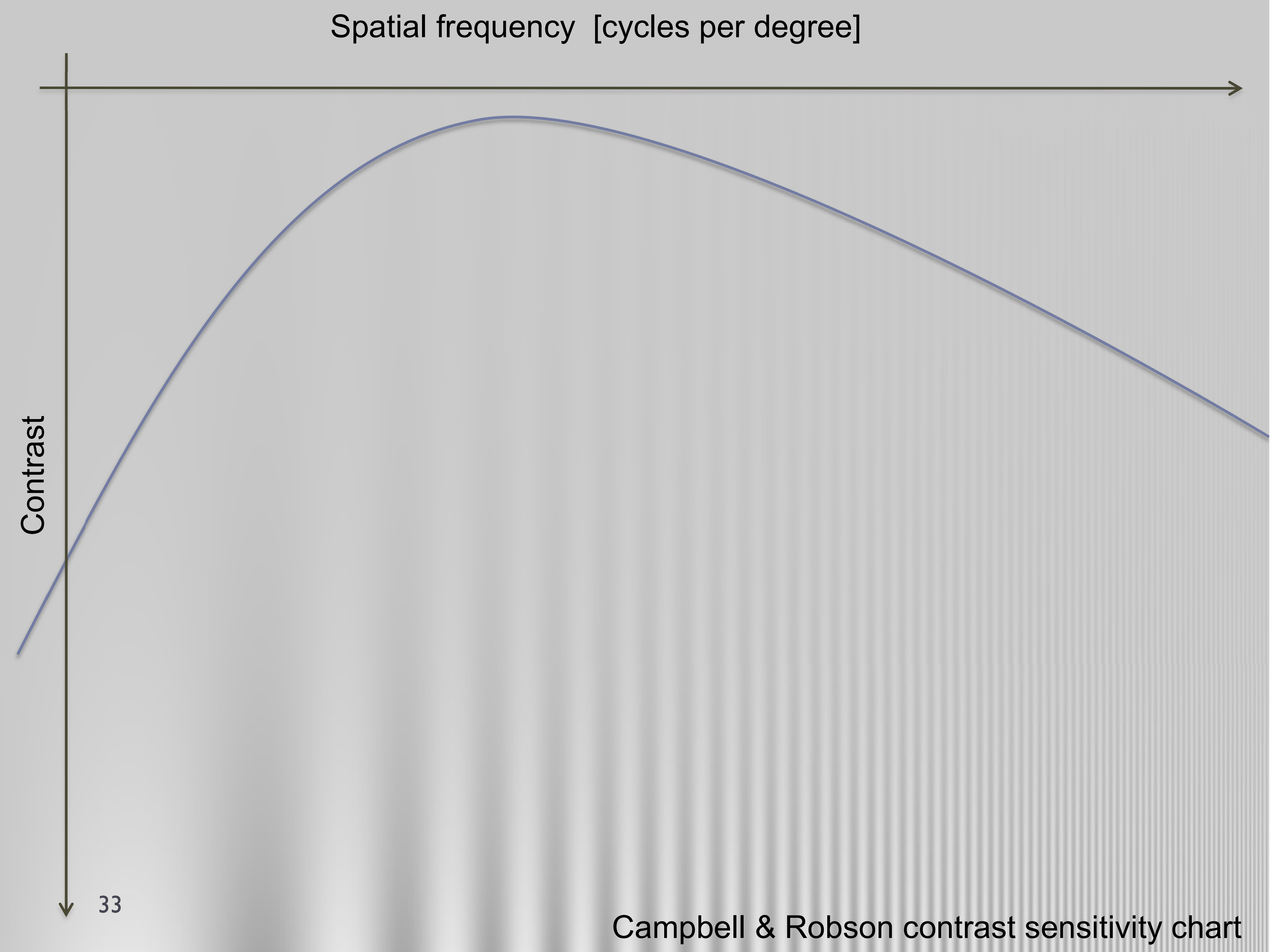
  - Any number of sinusoids can be fitted to this set of samples
  - It is possible to fit an infinite number of sinusoids if we allow infinitely high frequency

Nyquist frequency / aliasing

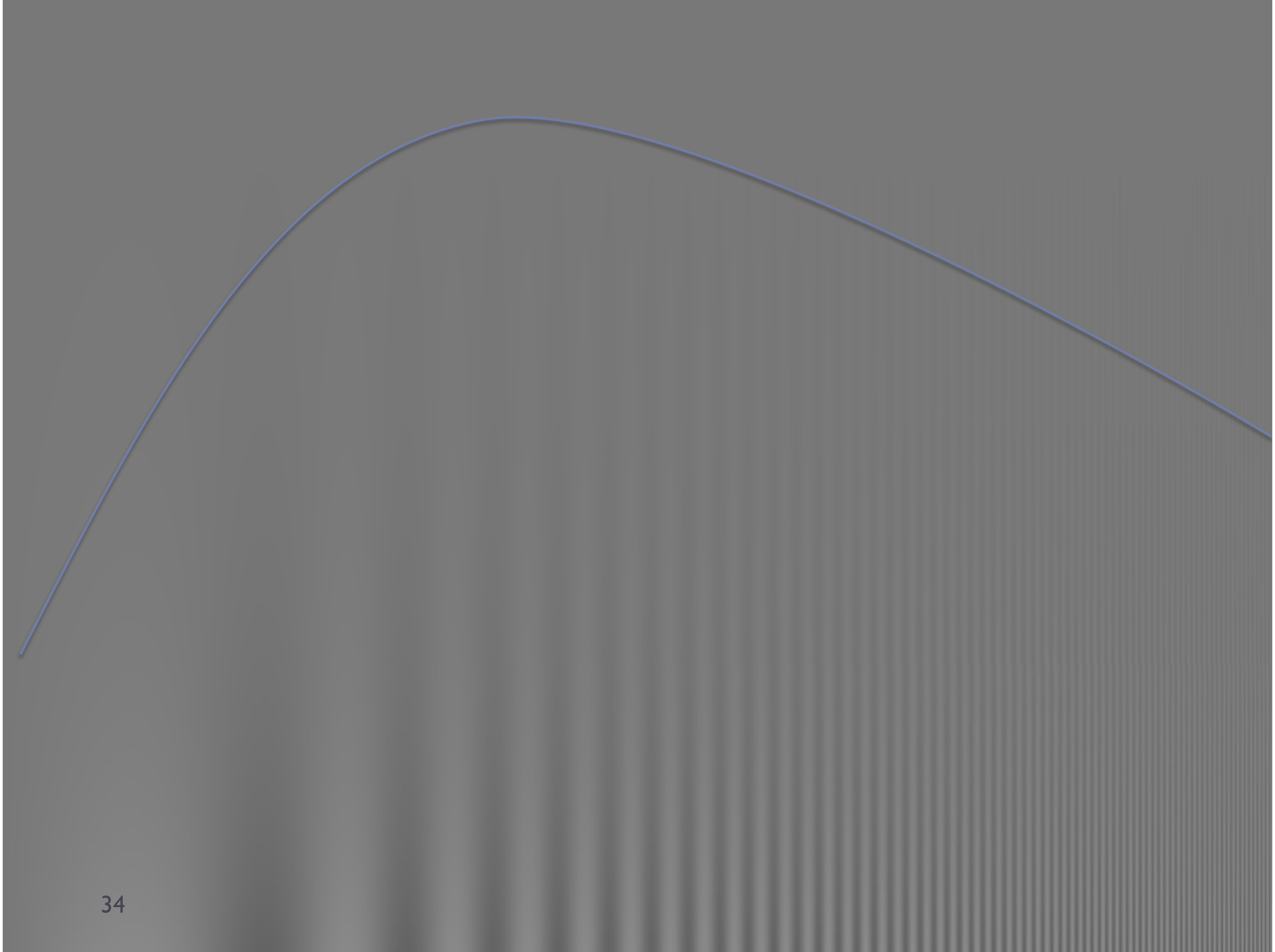
- Nuquist frequency is the highest frequency that can be represented by a discrete set of uniform samples (pixels)
- Nuquist frequency = 0.5 sampling rate
  - For audio
    - If the sampling rate is 44100 samples per second (audio CD), then the Nyquist frequency is 22050 Hz
  - For images (visual degrees)
    - If the sampling rate is 60 pixels per degree, then the Nyquist frequency is 30 cycles per degree
- When resampling an image to lower resolution, the frequency content above the Nyquist frequency needs to be removed (reduced in practice)
  - Otherwise **aliasing** is visible

Modeling contrast detection





Campbell & Robson contrast sensitivity chart



Contrast sensitivity function

Temporal frequency

Stimulus size

Spatial frequency

Eccentricity

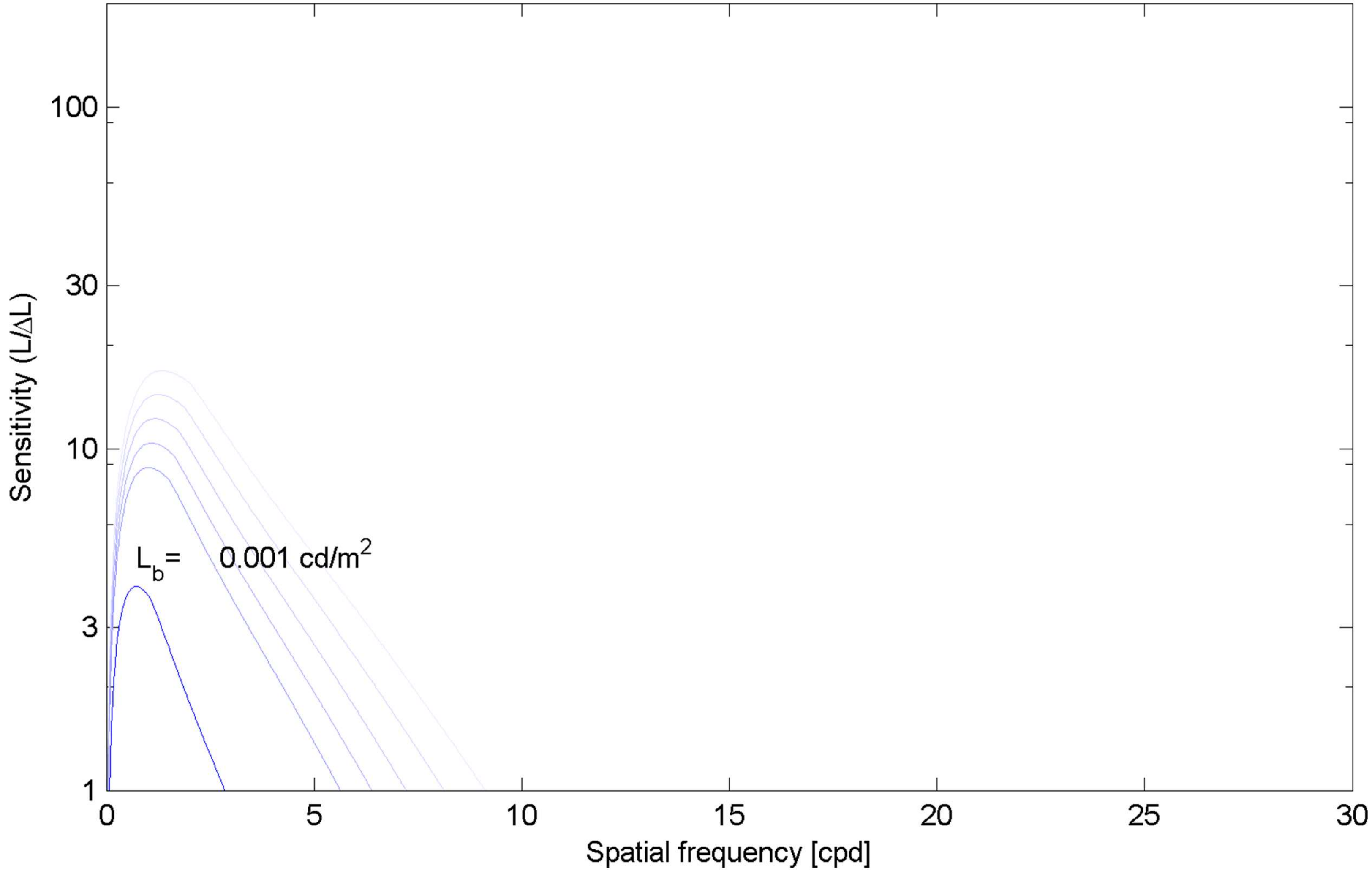
$$CSF = S(\rho, \theta, \omega, l, i^2, d, e)$$

Orientation

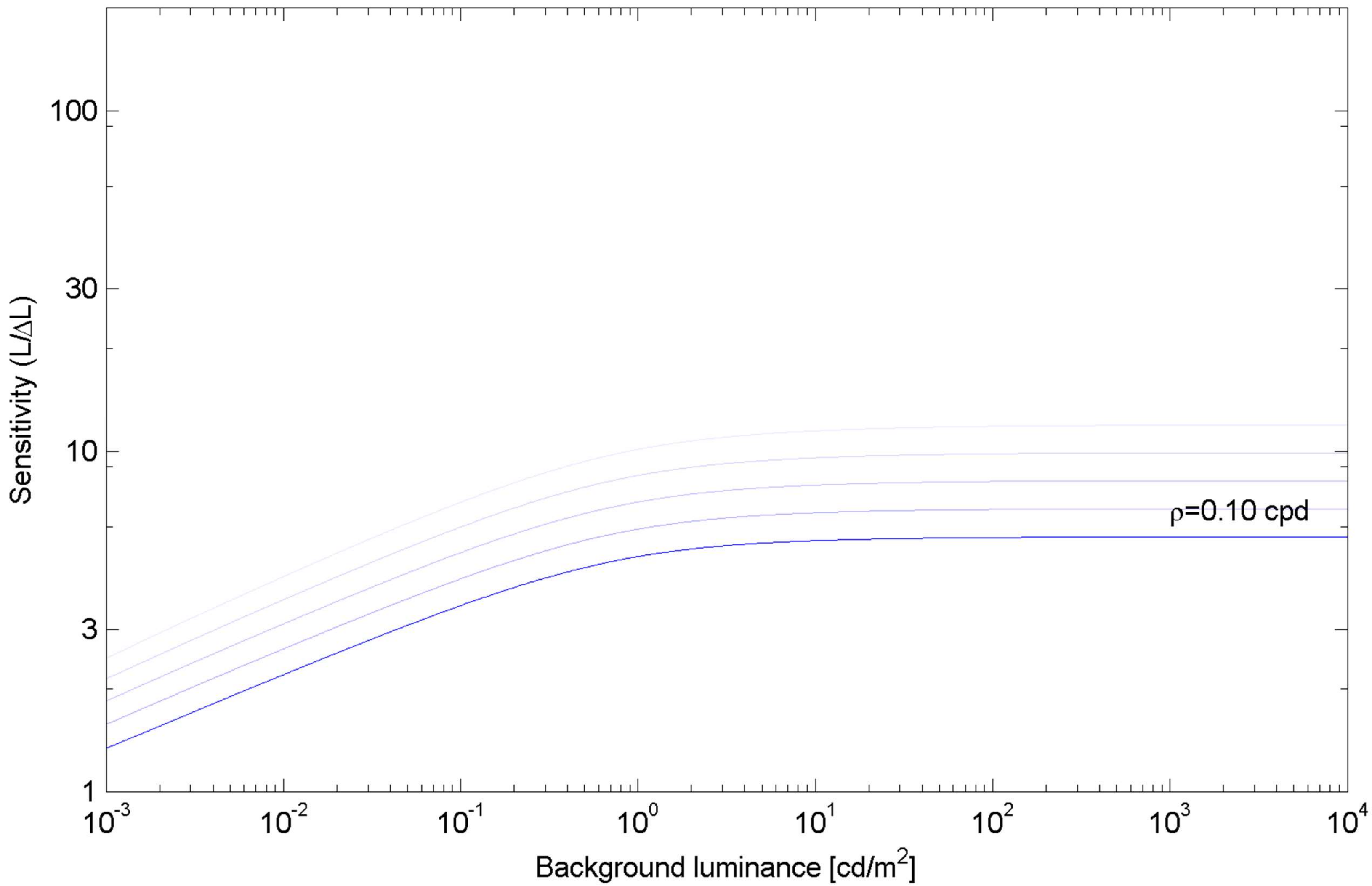
Viewing distance

Adapting luminance

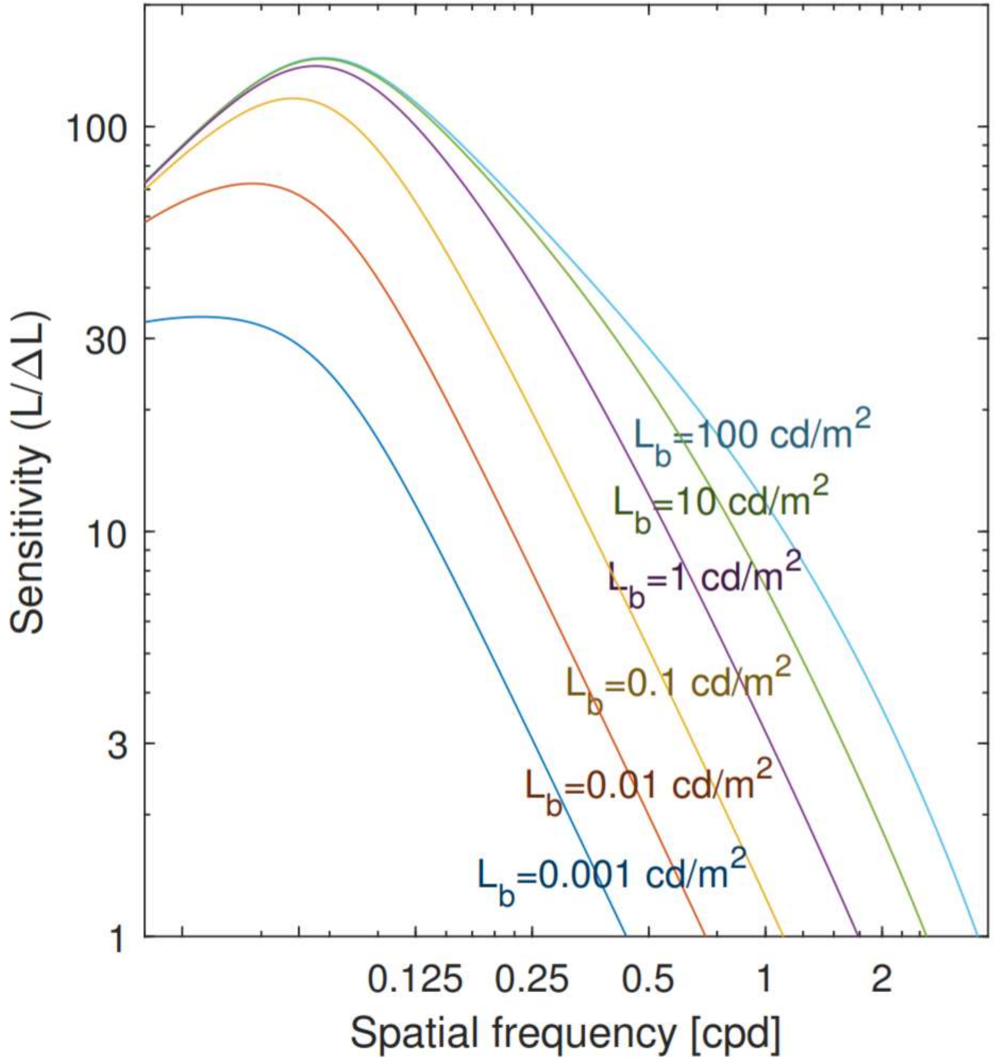
CSF as a function of spatial frequency

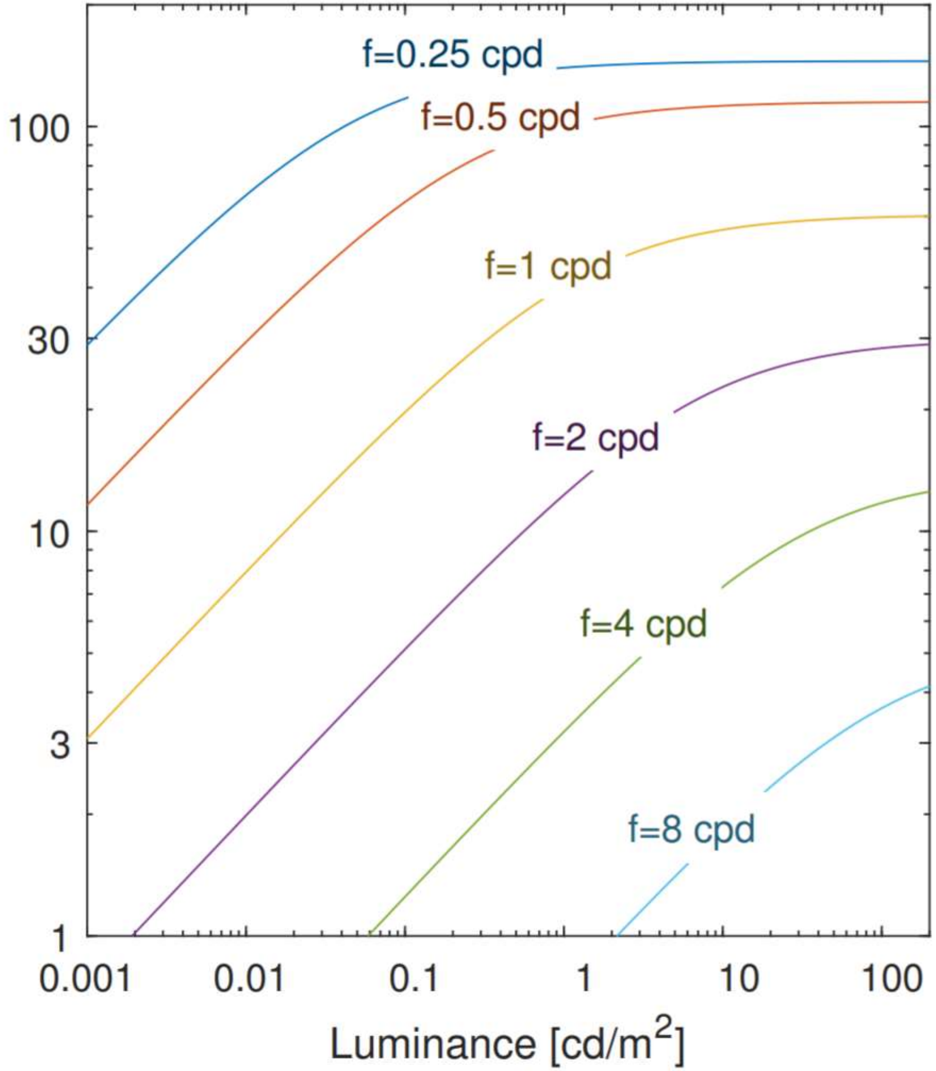


CSF as a function of background luminance



CSF as a function of spatial frequency and background luminance





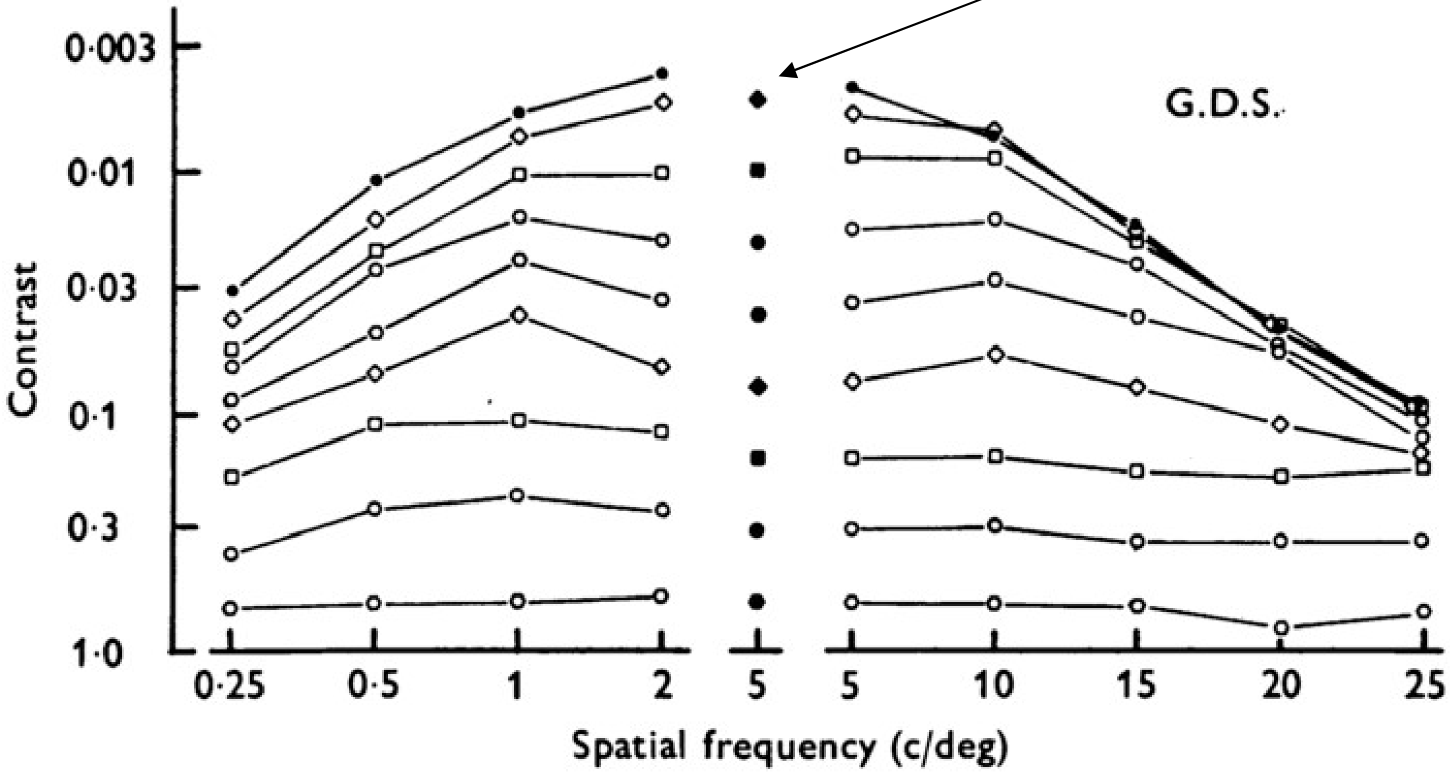
Contrast constancy

Experiment: Adjust the amplitude of one sinusoidal grating until it matches the perceived magnitude of another sinusoidal grating.

Test

Match?

Reference



From: Georgeson and Sullivan. 1975. J. Phsysio.

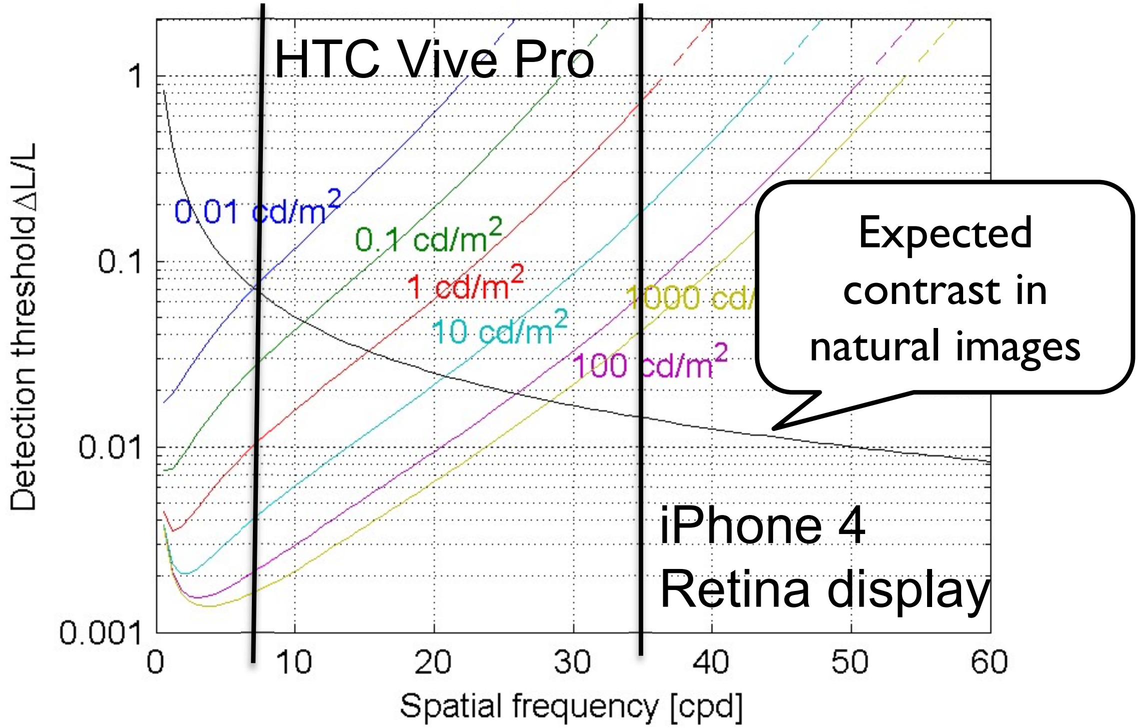

Contrast constancy
No CSF above the detection threshold

CSF and the resolution

- CSF plotted as the detection contrast

$$\frac{\Delta L}{L_b} = S^{-1}$$

- The contrast below each line is invisible
- Maximum perceivable resolution depends on luminance

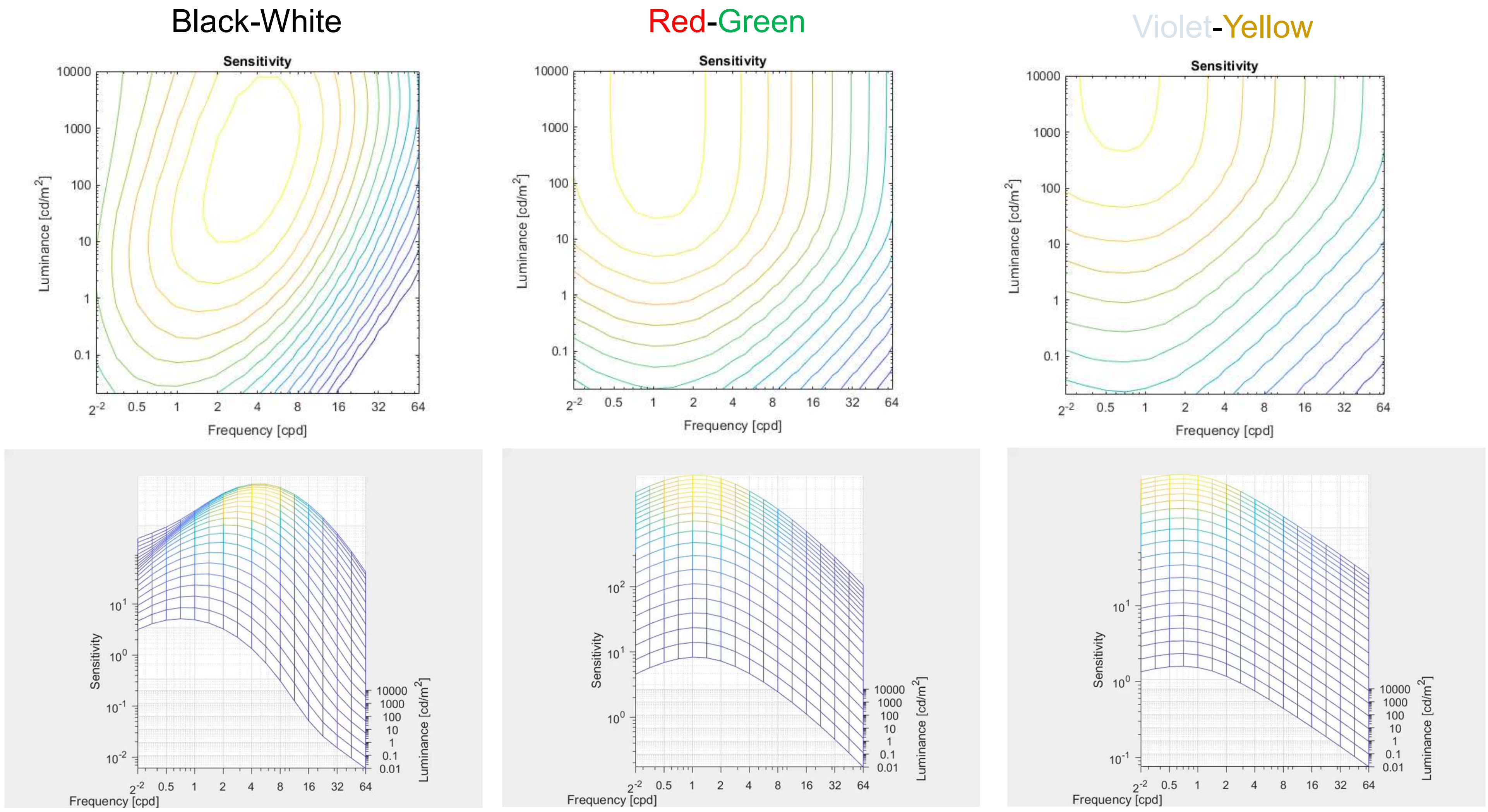


CSF models:
Barten, P. G. J. (2004).
https://doi.org/10.1117/12.537476

Spatio-chromatic CSF

Spatio-chromatic contrast sensitivity

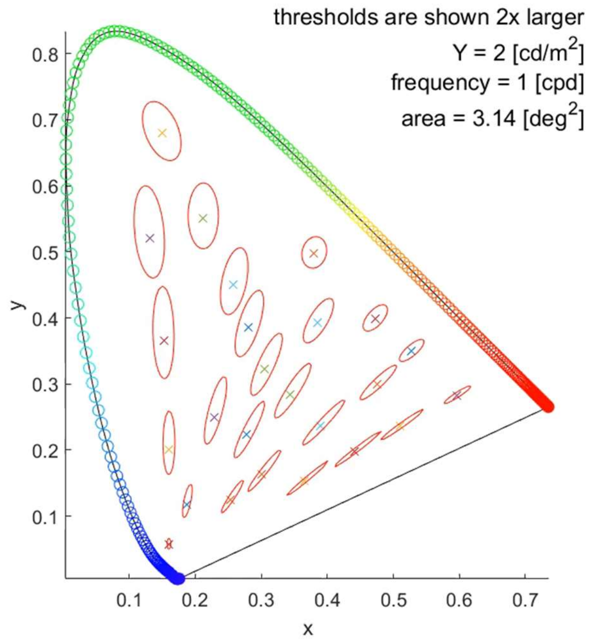
- CSF as a function of **luminance** and **frequency**

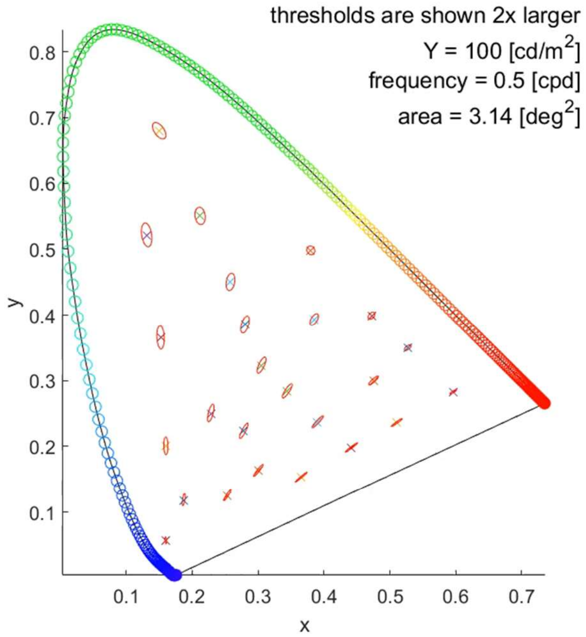


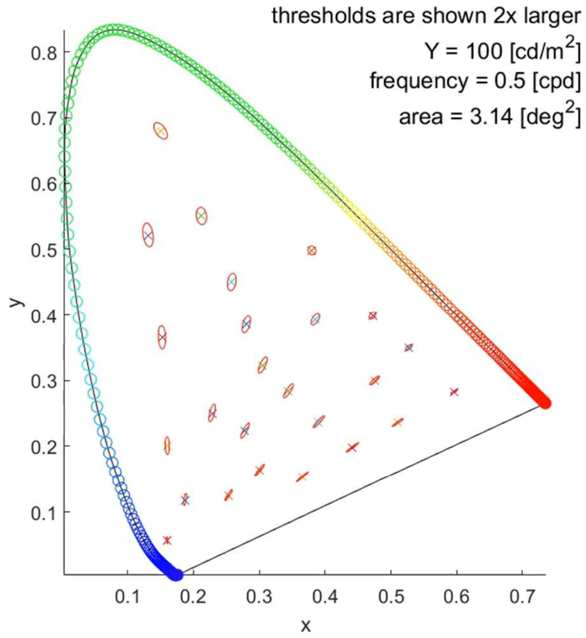
http://dx.doi.org/10.2352/issn.2169-2629.2020.28.1

CSF and colour ellipses

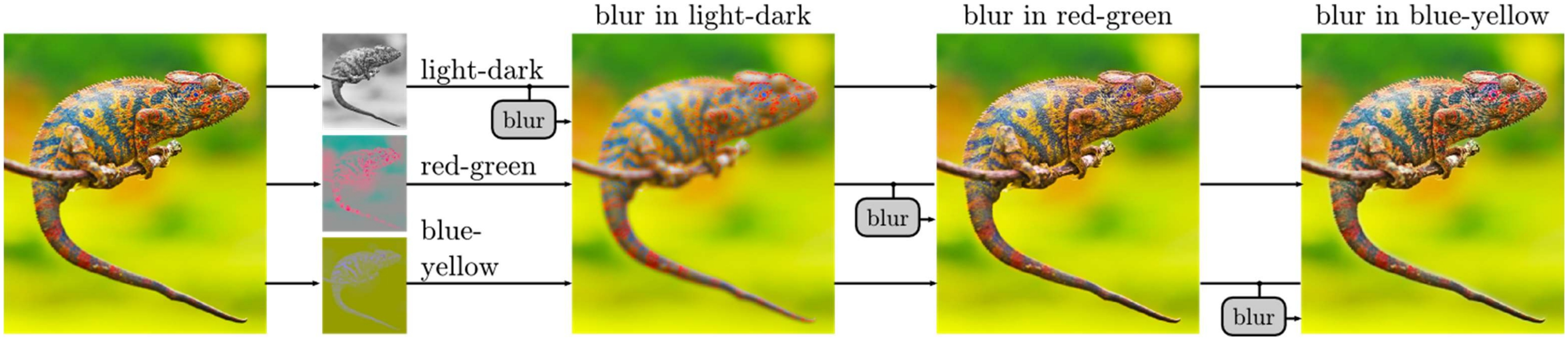
- Colour discrimination as a function of
  - Background colour and luminance [LMS]
  - Spatial frequency [cpd]
  - Size [deg]







Visibility of blur



- The same amount of blur was introduced into light-dark, red-green and blue-yellow colour opponent channels
- The blur is only visible in light-dark channel
- This property is used in image and video compression
  - Sub-sampling of colour channels (4:2:1)

UNIVERSITY OF CAMBRIDGE
COMPUTER LABORATORY
RAINBOW GRAPHICS GROUP

Advanced Graphics and Image Processing

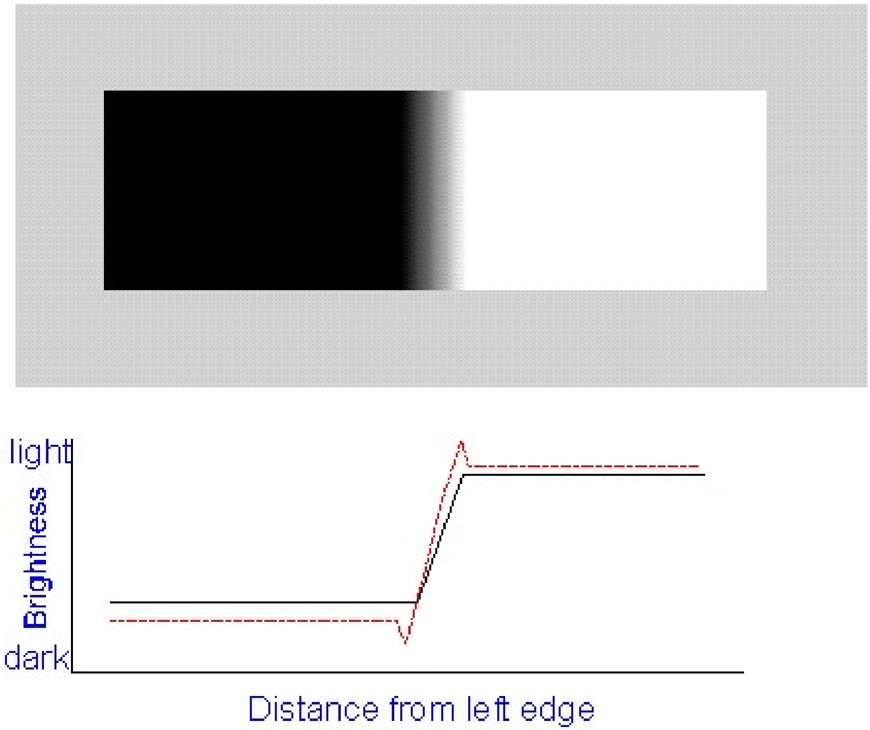
Models of early visual perception

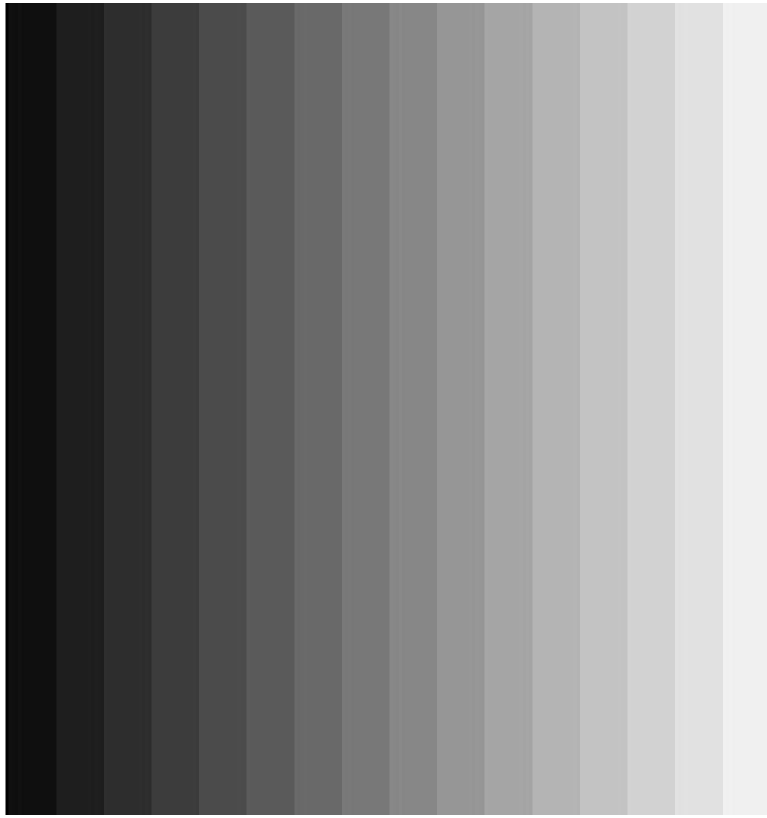
Part 4/6 – lateral inhibition and multi-resolution models

Rafal Mantiuk
Computer Laboratory, University of Cambridge

Mach Bands – evidence for band-pass visual processing

- "Overshooting" along edges
  - Extra-bright rims on bright sides
  - Extra-dark rims on dark sides
- Due to "Lateral Inhibition"

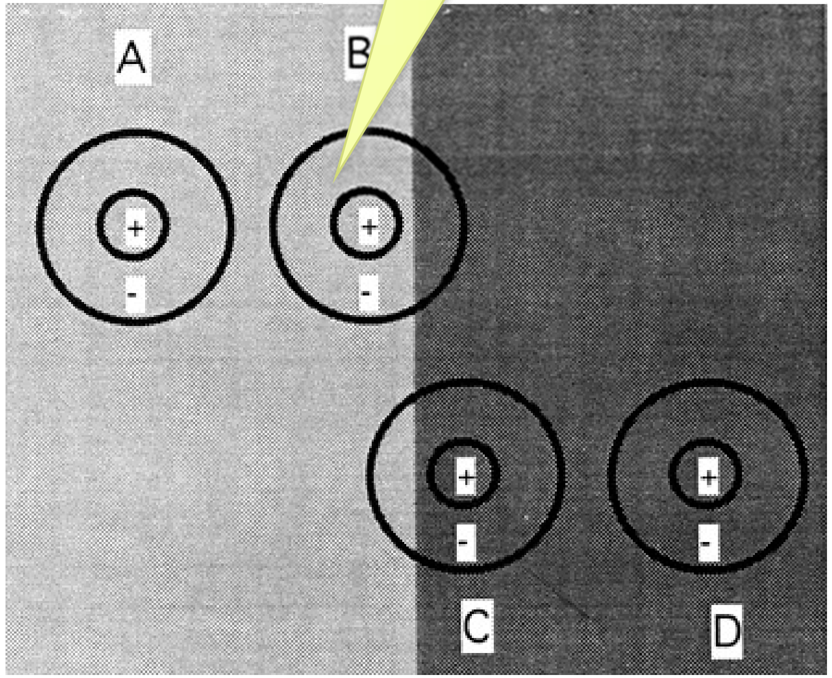


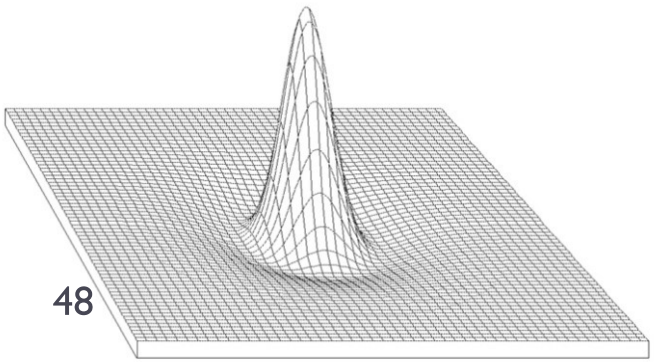


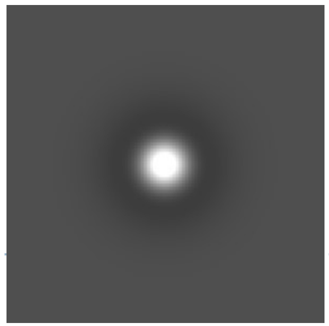
Centre-surround (Lateral Inhibition)

- "Pre-processing" step within the retina
  - Surrounding brightness level weighted negatively
    - A: high stimulus, maximal bright inhibition
    - B: high stimulus, reduced inhibition & stronger response
    - D: low stimulus, maximal inhibition
    - C: low stimulus, increased inhibition & weaker response

Center-surround receptive fields (groups of photoreceptors)







Centre-surround: Hermann Grid

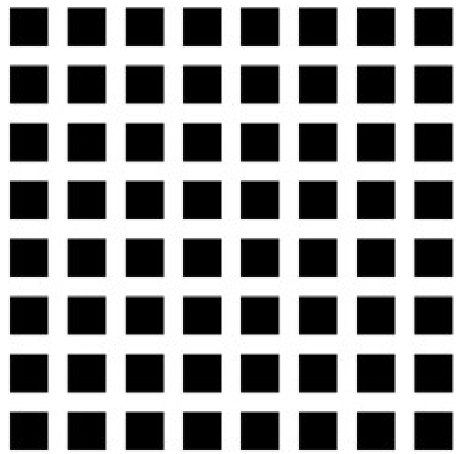
- Dark dots at crossings
- Explanation
  - Crossings (A)
    - More surround stimulation (more bright area)

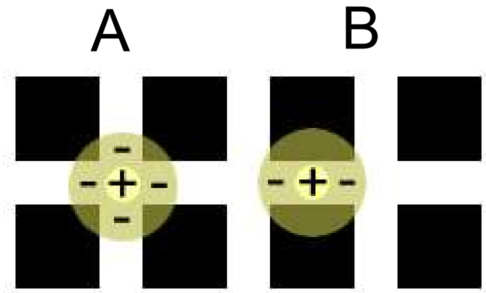
    ⇒ Less inhibition

    ⇒ Weaker response
  - Streets (B)
    - Less surround stimulation

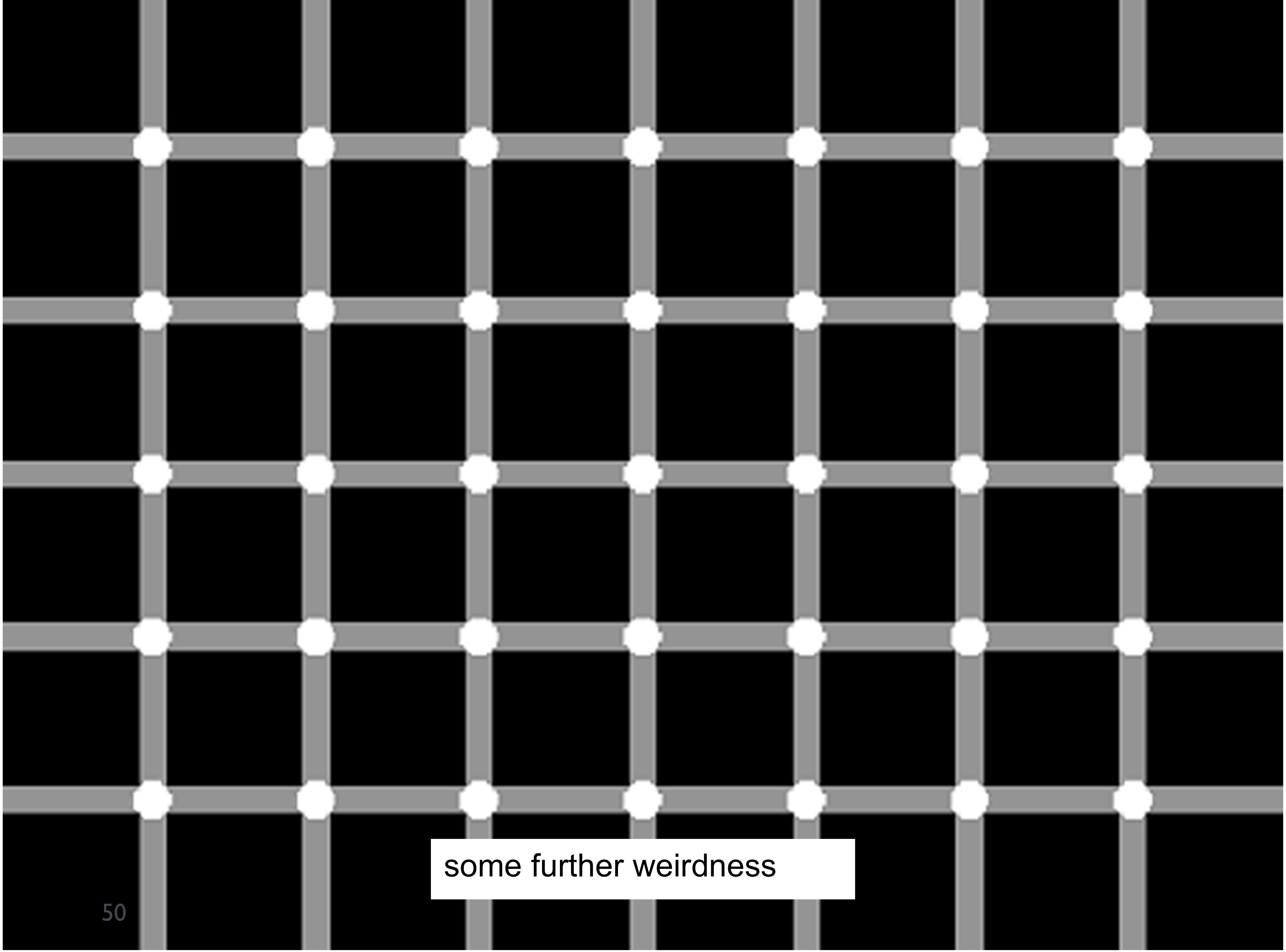
    ⇒ More inhibition

    ⇒ Greater response
- Simulation
  - Darker at crossings, brighter in streets
  - Appears more steady
  - What if reversed ?





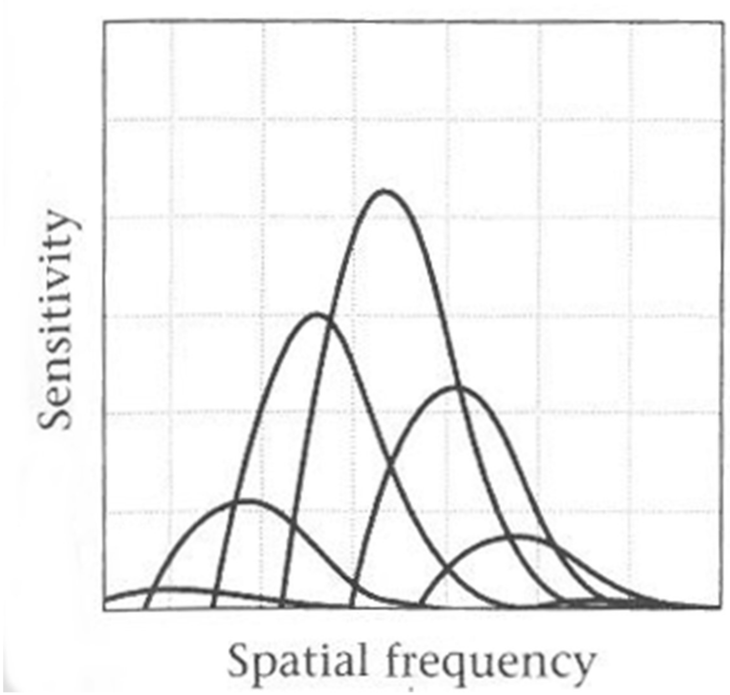
Simulation



some further weirdness

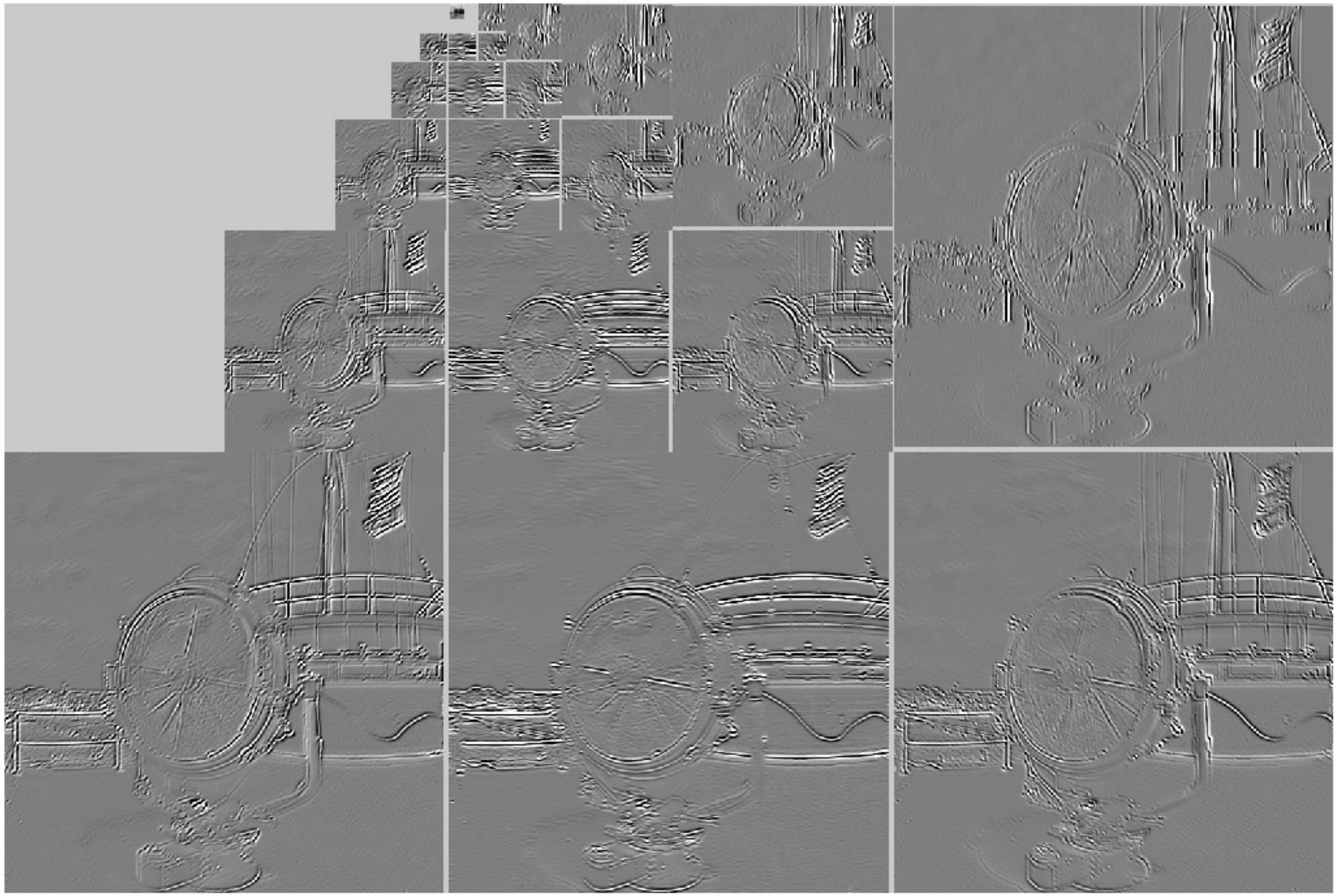
Spatial-frequency selective channels

- The visual information is decomposed in the visual cortex into multiple channels
  - The channels are selective to spatial frequency, temporal frequency and orientation
  - Each channel is affected by different „noise" level
  - The CSF is the net result of information being passed in noise-affected visual channels



From: Wandell, 1995

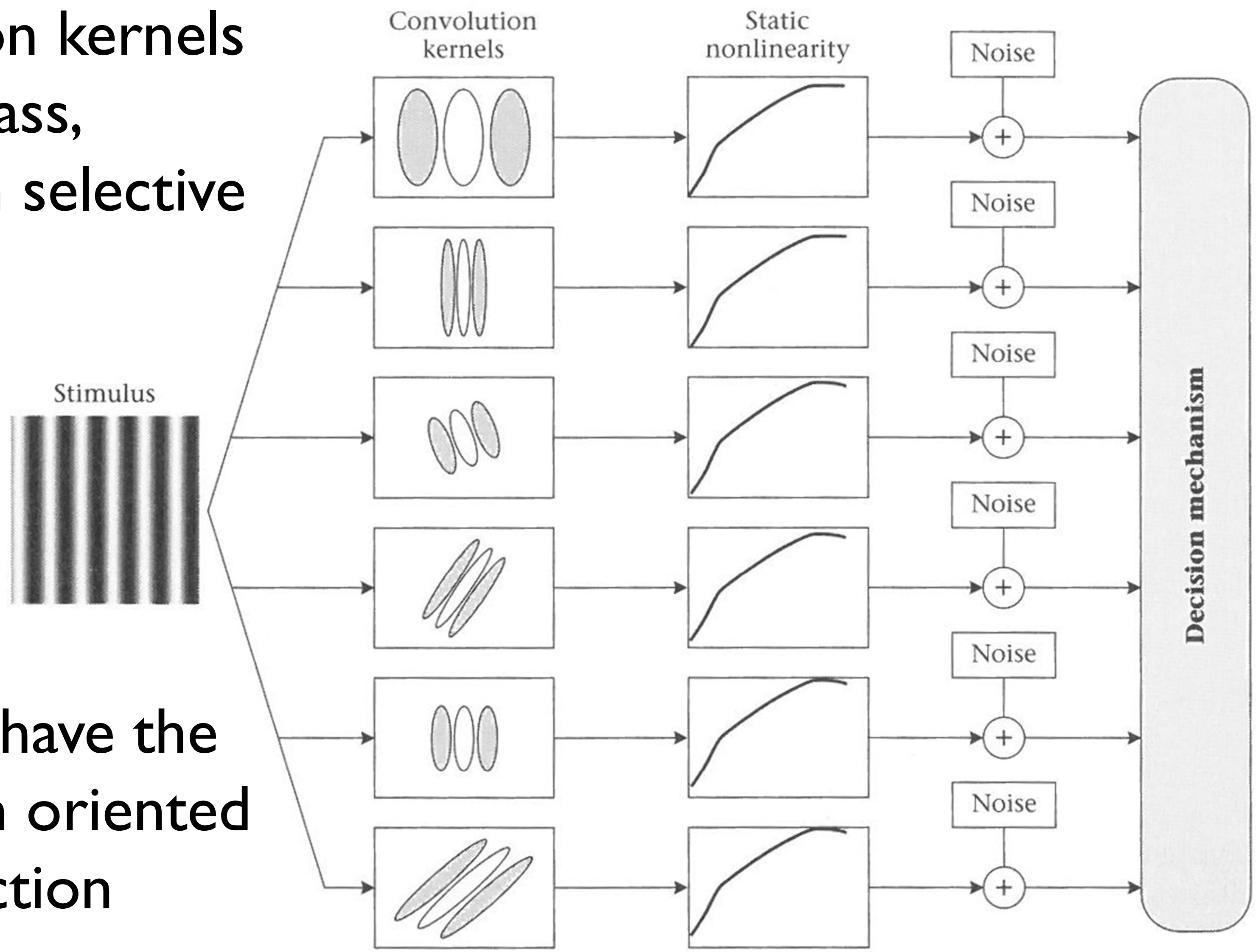
Multi-scale decomposition



Steerable pyramid decomposition

Multi-resolution visual model

- Convolution kernels are band-pass, orientation selective filters
- The filters have the shape of an oriented Gabor function



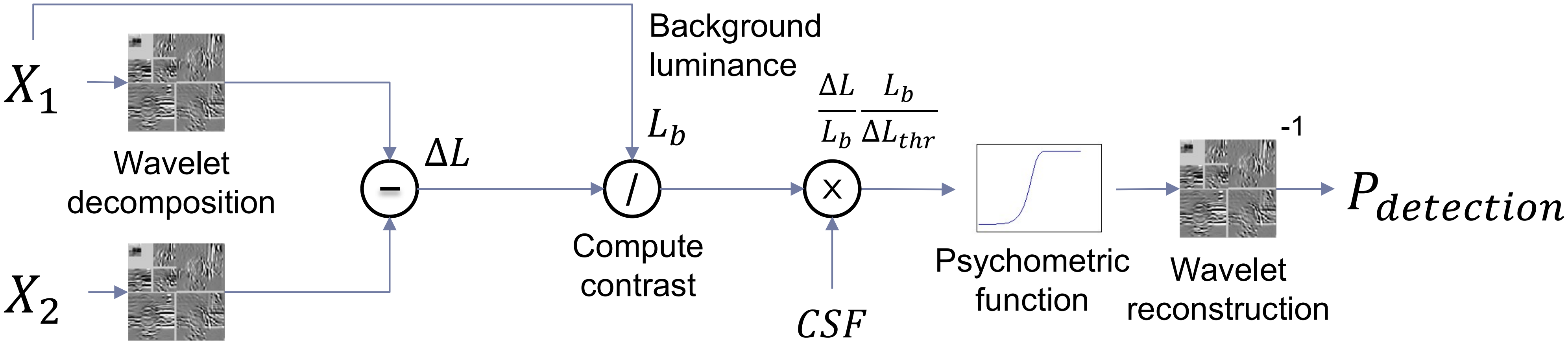
From: Wandell, 1995

Predicting visible differences with CSF

- We can use CSF to find the probability of spotting a difference beween a pair of images X_1 and X_2:

$$p(f[X_1] = f[X_2] \mid X_1, X_2, CSF) \qquad f[X]$$

The percept of image X

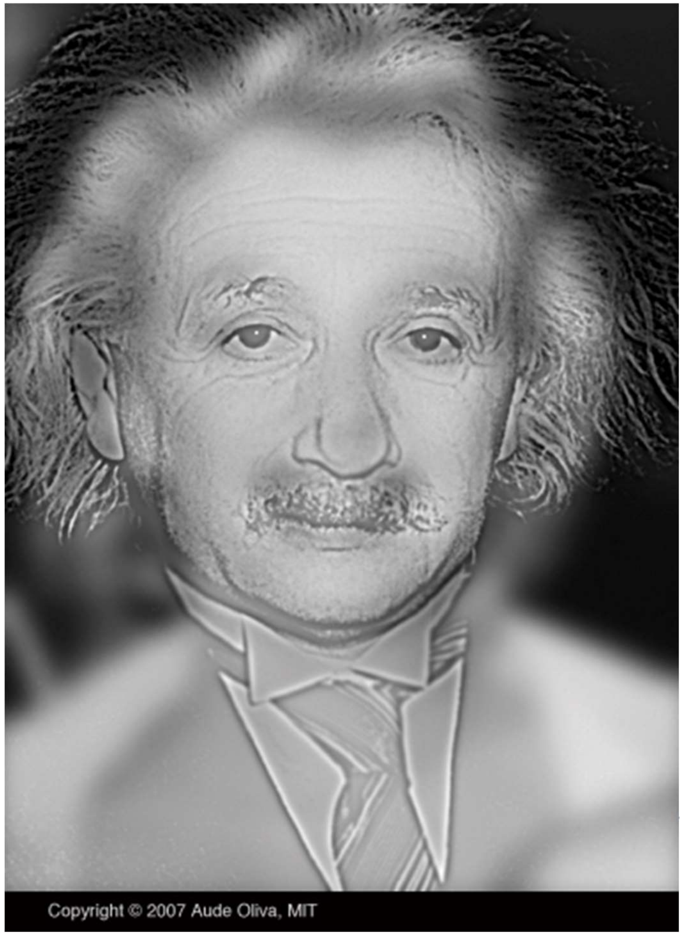


(simplified) Visual Difference Predictor

Daly, S. (1993).

Applications of multi-scale models

- JPEG2000
  - Wavelet decomposition
- JPEG / MPEG
  - Frequency transforms
- Image pyramids
  - Blending & stitching
  - Hybrid images



Hybrid Images by Aude Oliva
http://cvcl.mit.edu/hybrid_gallery

UNIVERSITY OF CAMBRIDGE
COMPUTER LABORATORY

RAINBOW GRAPHICS GROUP

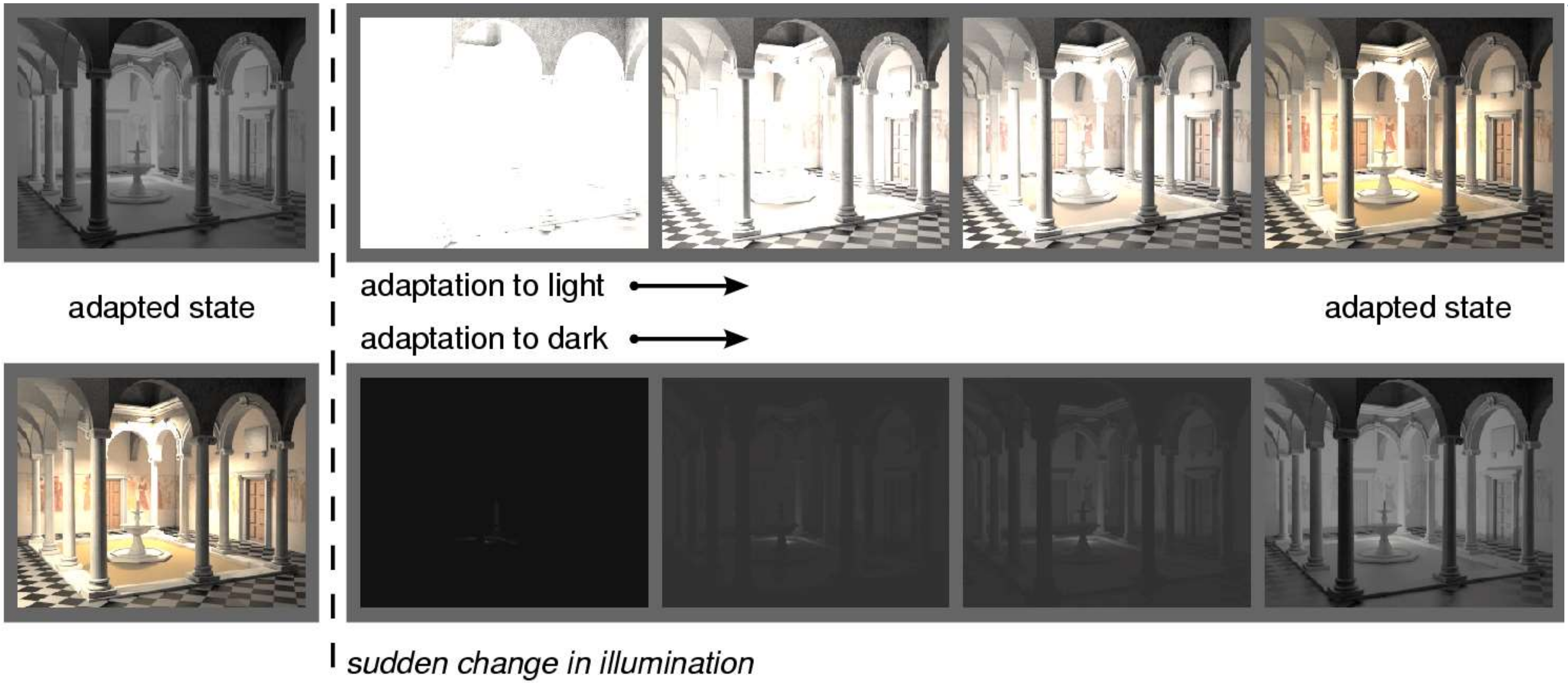
Advanced Graphics and Image Processing

Models of early visual perception

Part 5/6 – light and dark adaptation

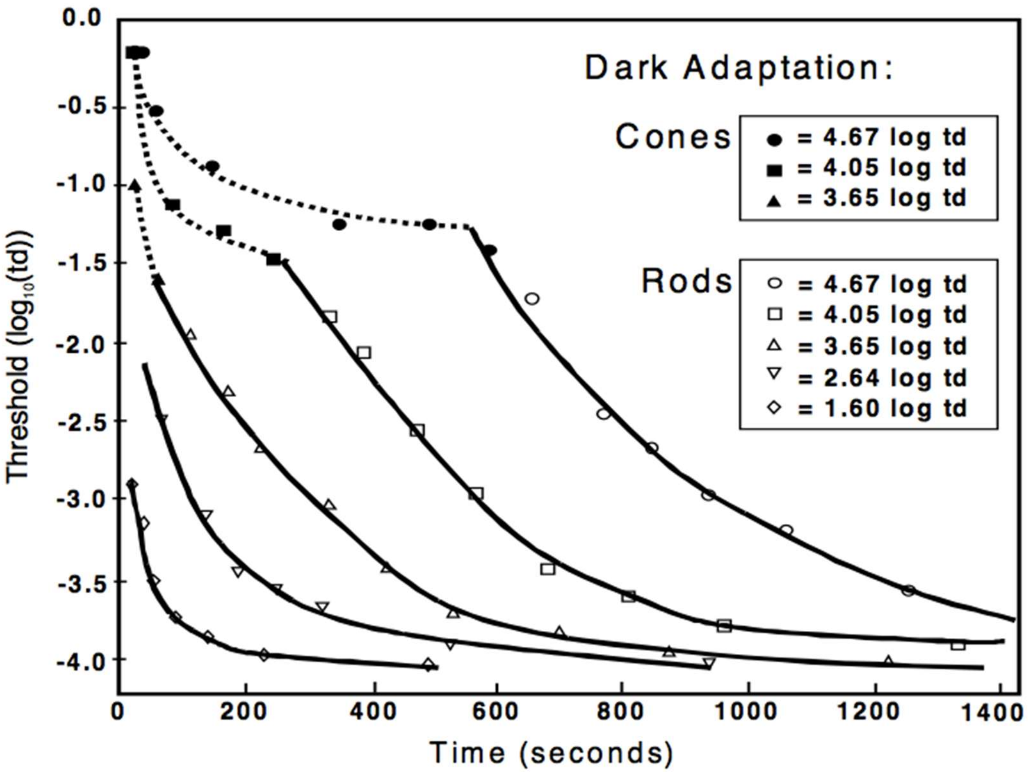
Rafal Mantiuk
Computer Laboratory, University of Cambridge

Light and dark adaptation

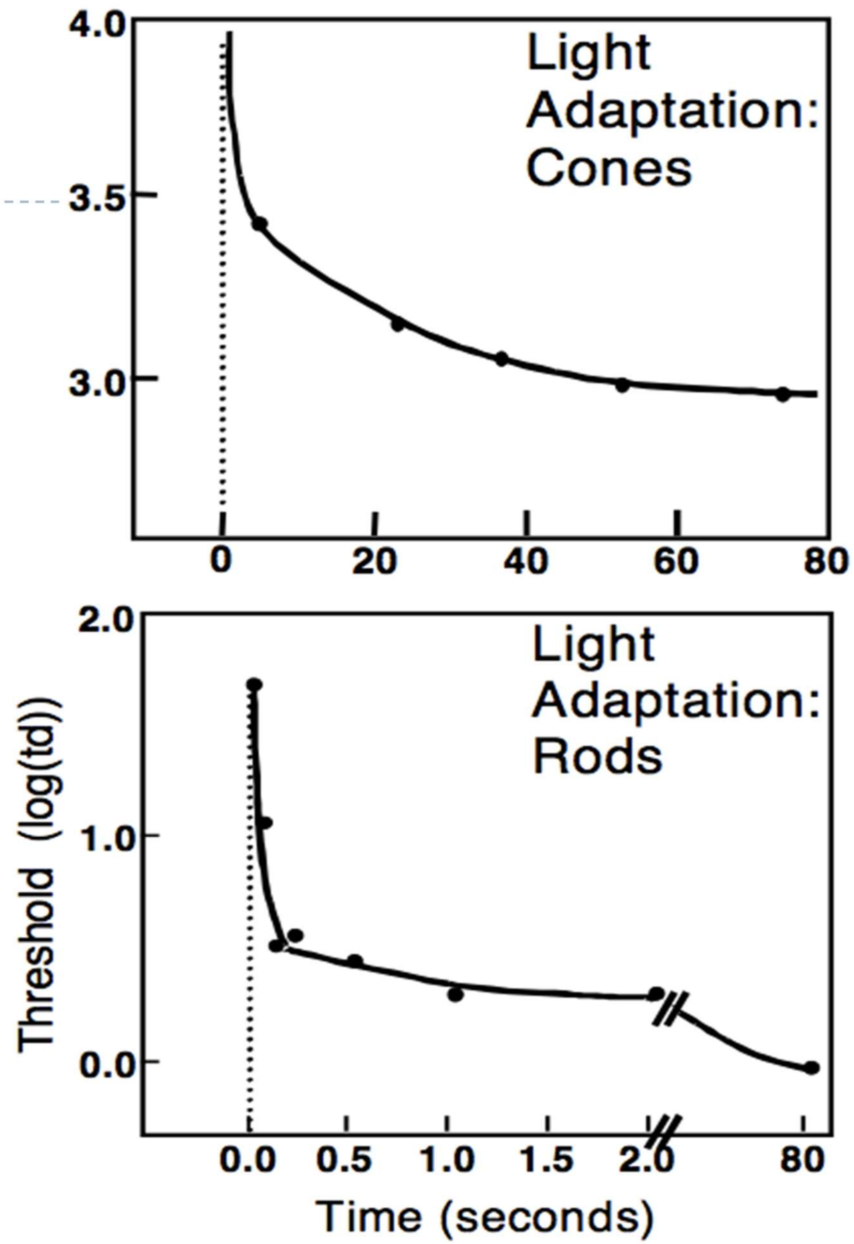


- Light adaptation: from dark to bright
- Dark adaptation: from bright to dark (much slower)

Time-course of adaptation



Bright -> Dark

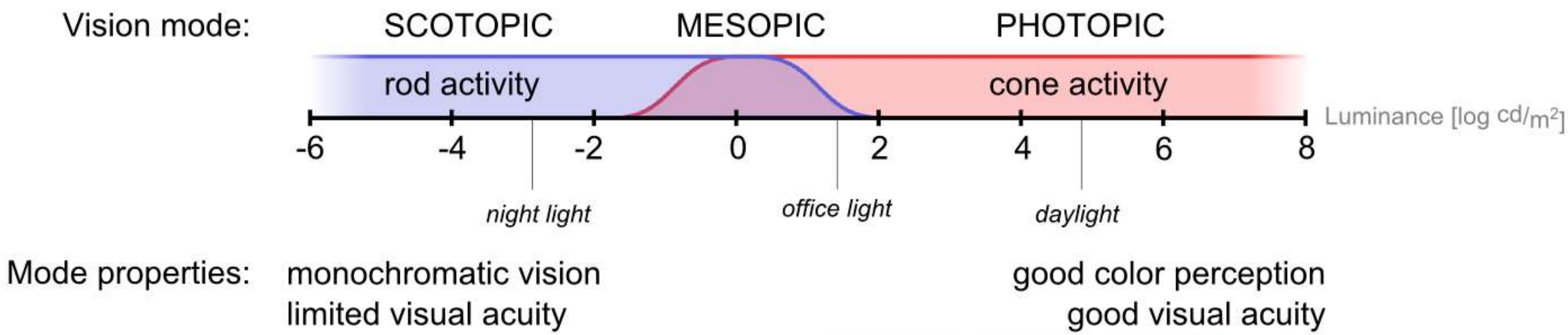


Dark -> Bright

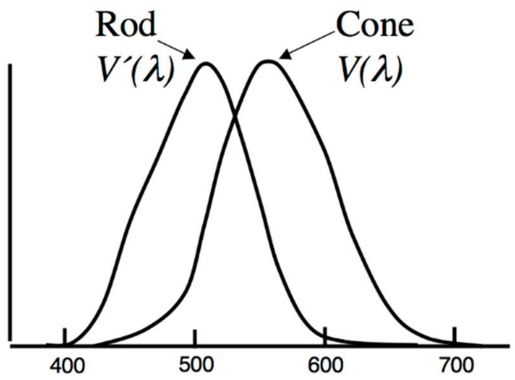
Temporal adaptation mechanisms

- Bleaching & recovery of photopigment
  - Slow assymetric (light -> dark, dark -> light)
  - Reaction times (1-1000 sec)
  - Separate time-course for rods and cones
- Neural adaptation
  - Fast
  - Approx. symmetric reaction times (10-3000 ms)
- Pupil
  - Diameter varies between 3 and 8 mm
  - About 1:7 variation in retinal illumunation

Night and daylight vision



Luminous efficiency



UNIVERSITY OF CAMBRIDGE
COMPUTER LABORATORY

RAINBOW GRAPHICS GROUP

Advanced Graphics and Image Processing

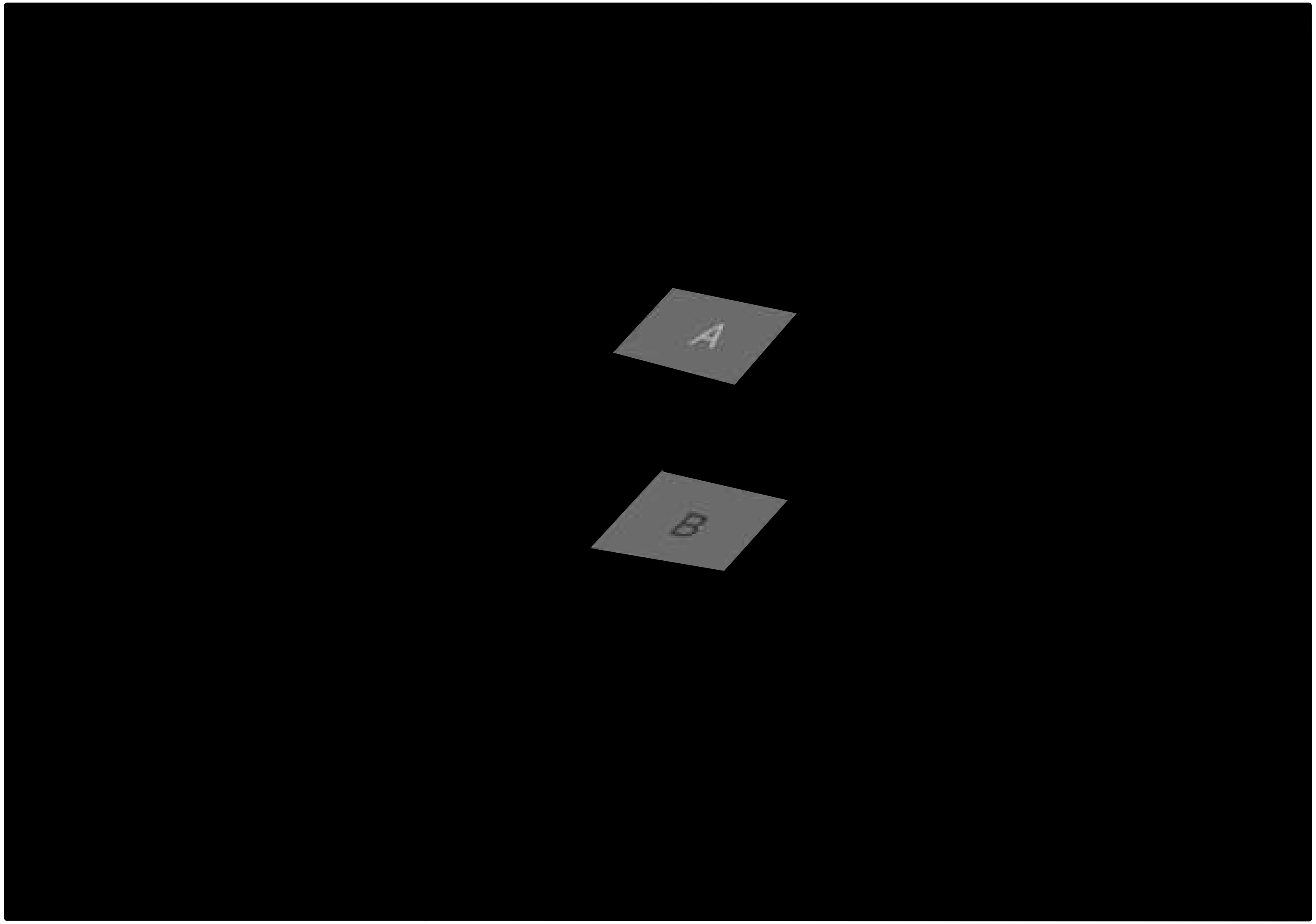
Models of early visual perception

Part 6/6 – high(er) level vision

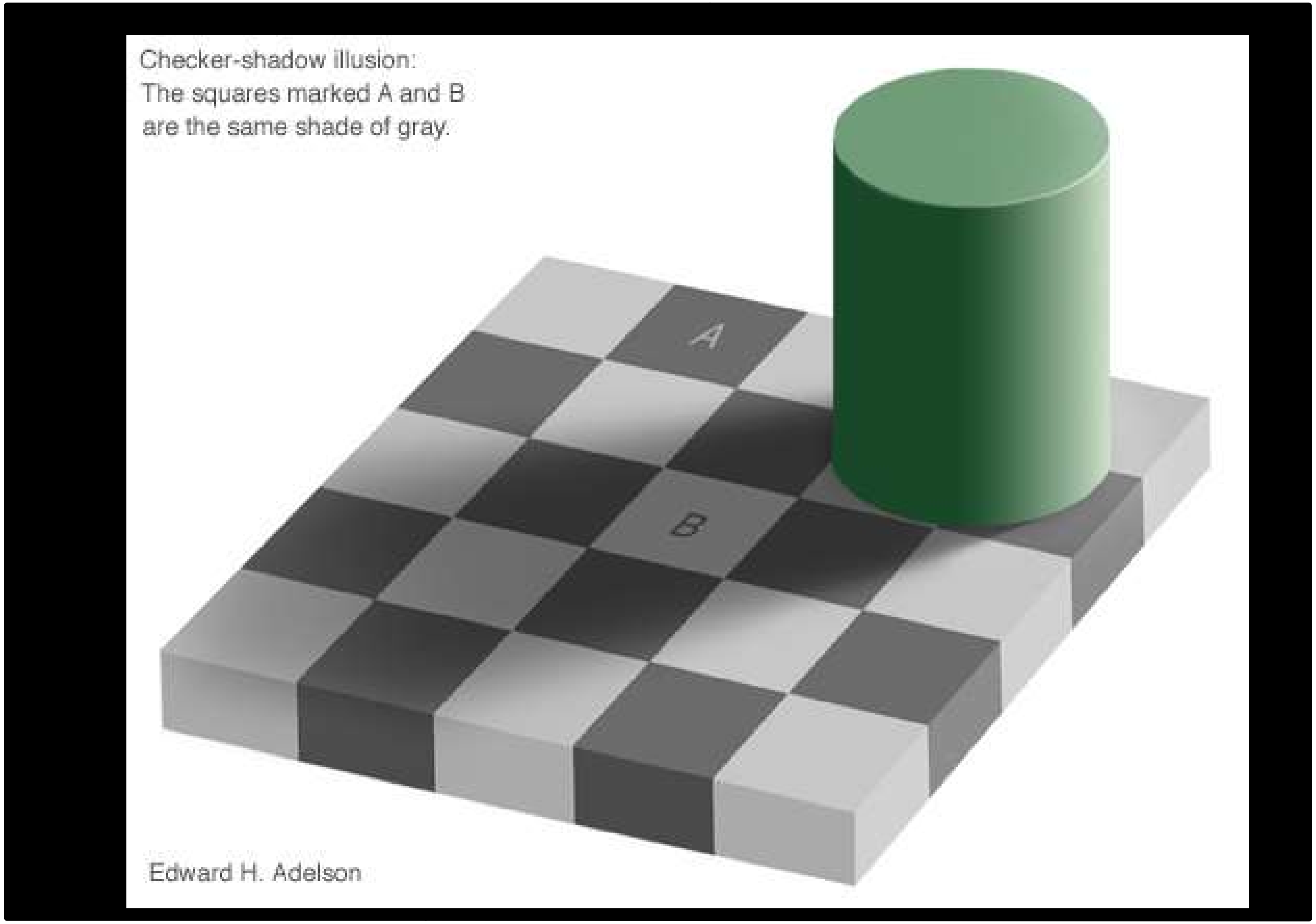
Rafal Mantiuk
Computer Laboratory, University of Cambridge

Simultaneous contrast

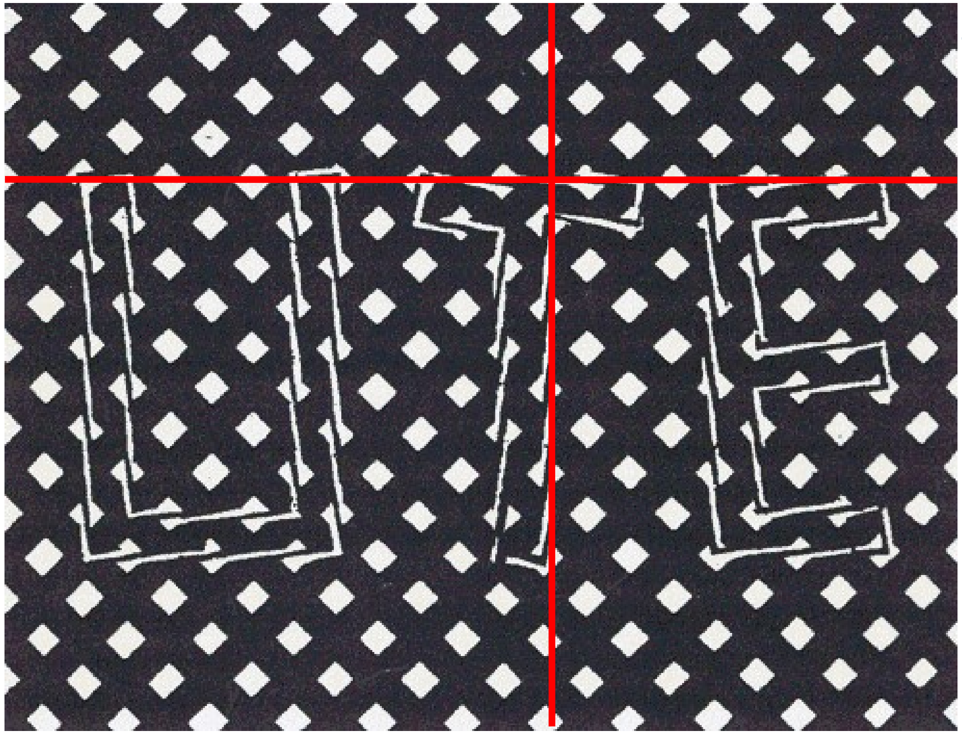
High-Level Contrast Processing

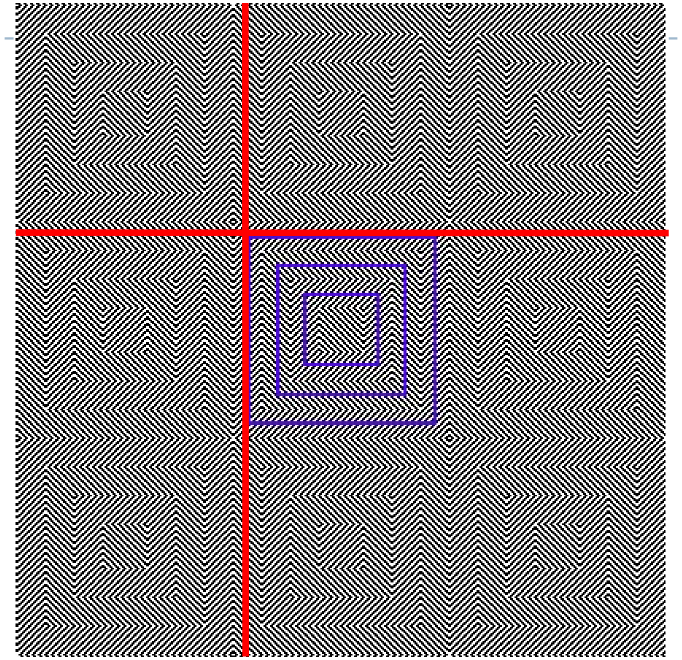


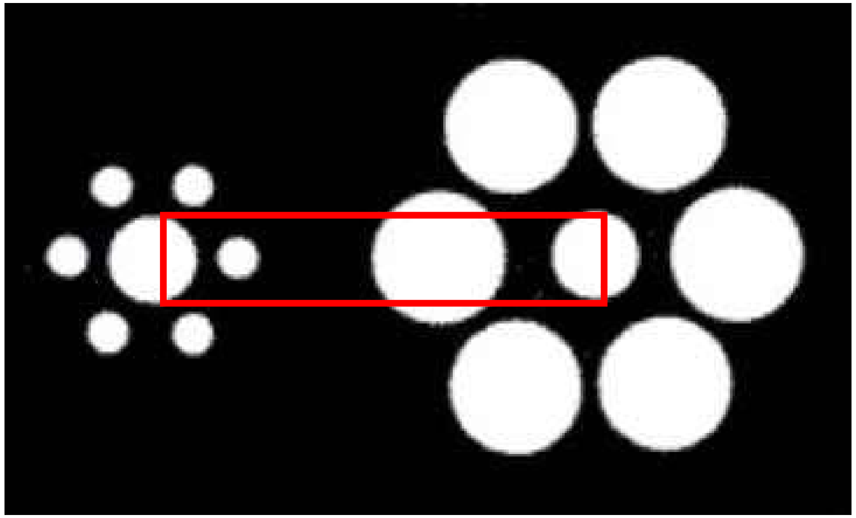
High-Level Contrast Processing



Shape Perception



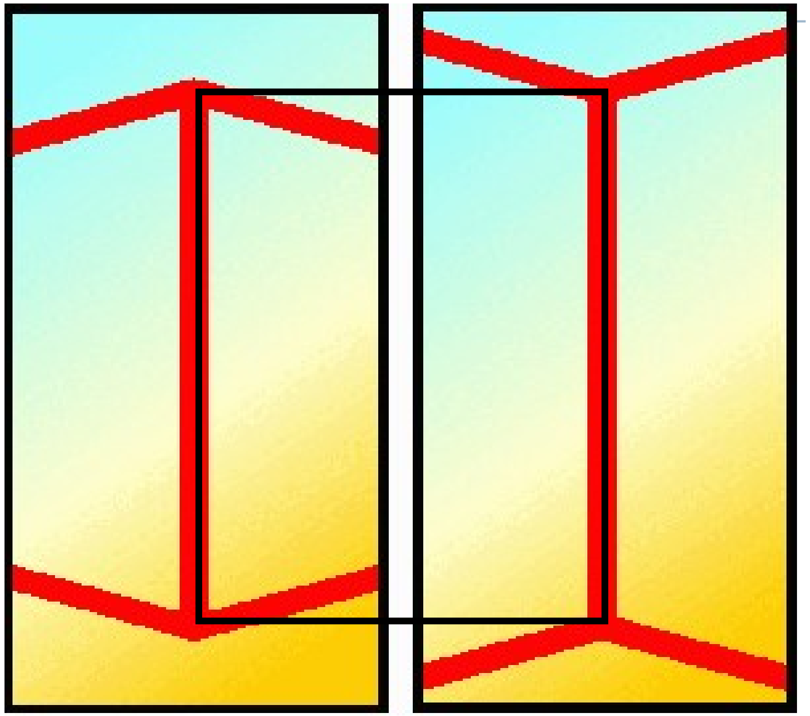


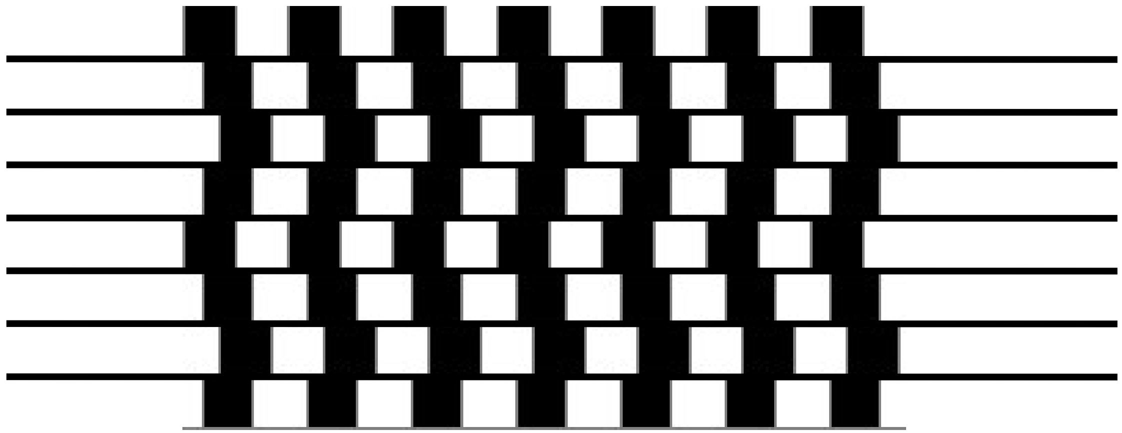


- Depends on surrounding primitives
  - Directional emphasis
  - Size emphasis

http://www.panoptikum.net/optischetaeuschungen/index.html

Shape Processing: Geometrical Clues



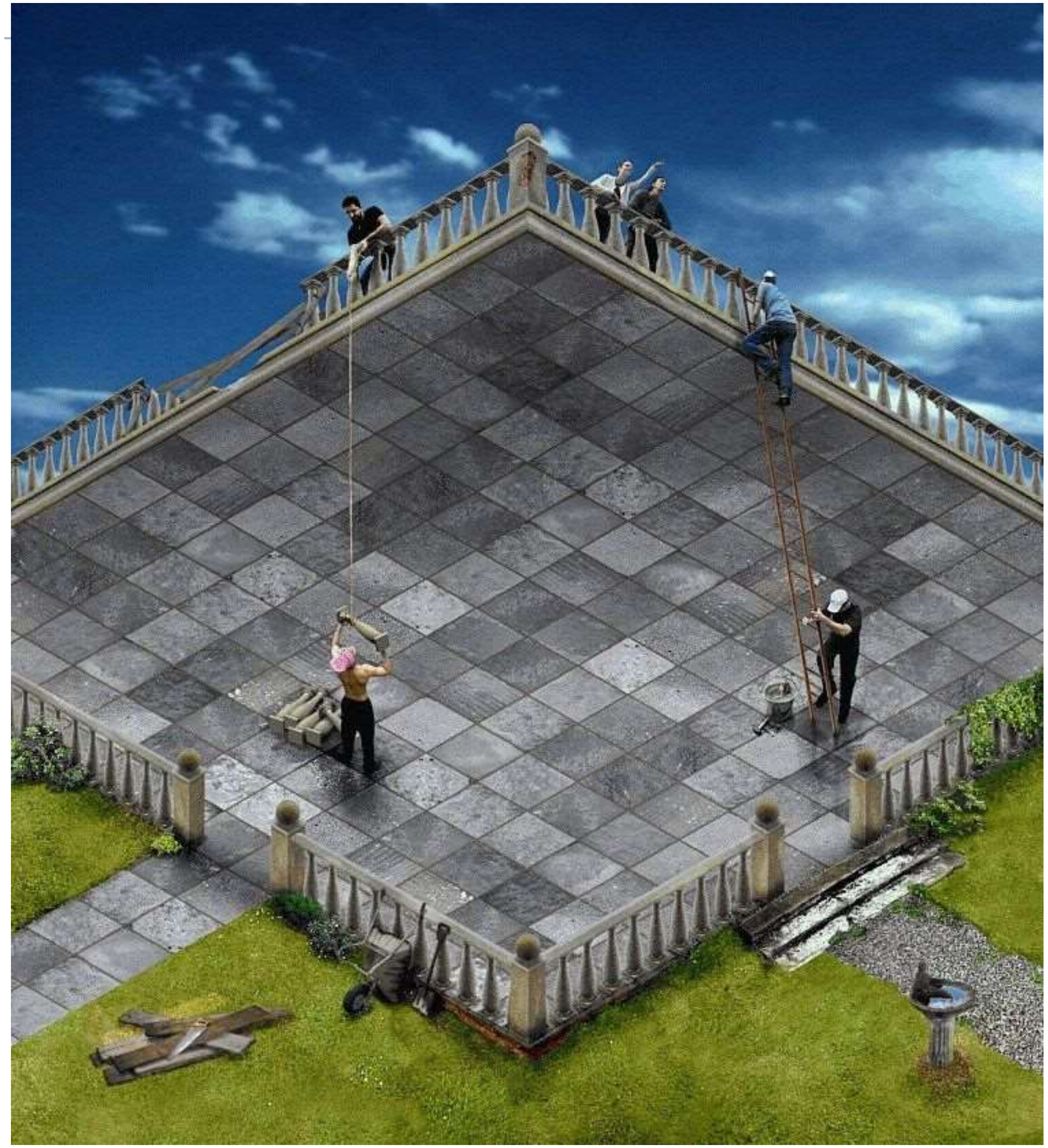


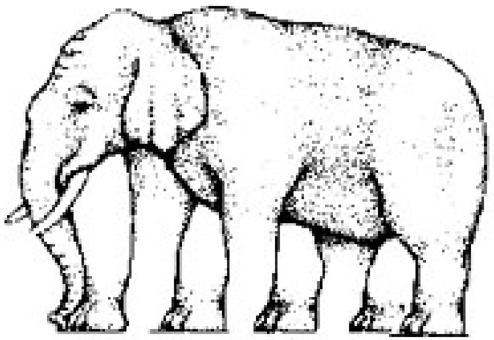
http://www.panoptikum.net/optischetaeuschungen/index.html

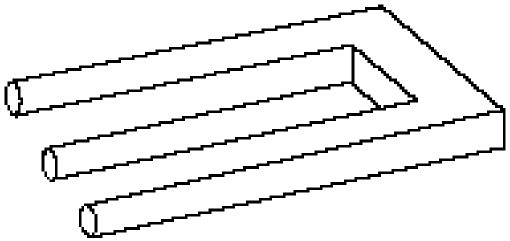
- Automatic geometrical interpretation
  - 3D perspective
  - Implicit scene depth

Impossible Scenes

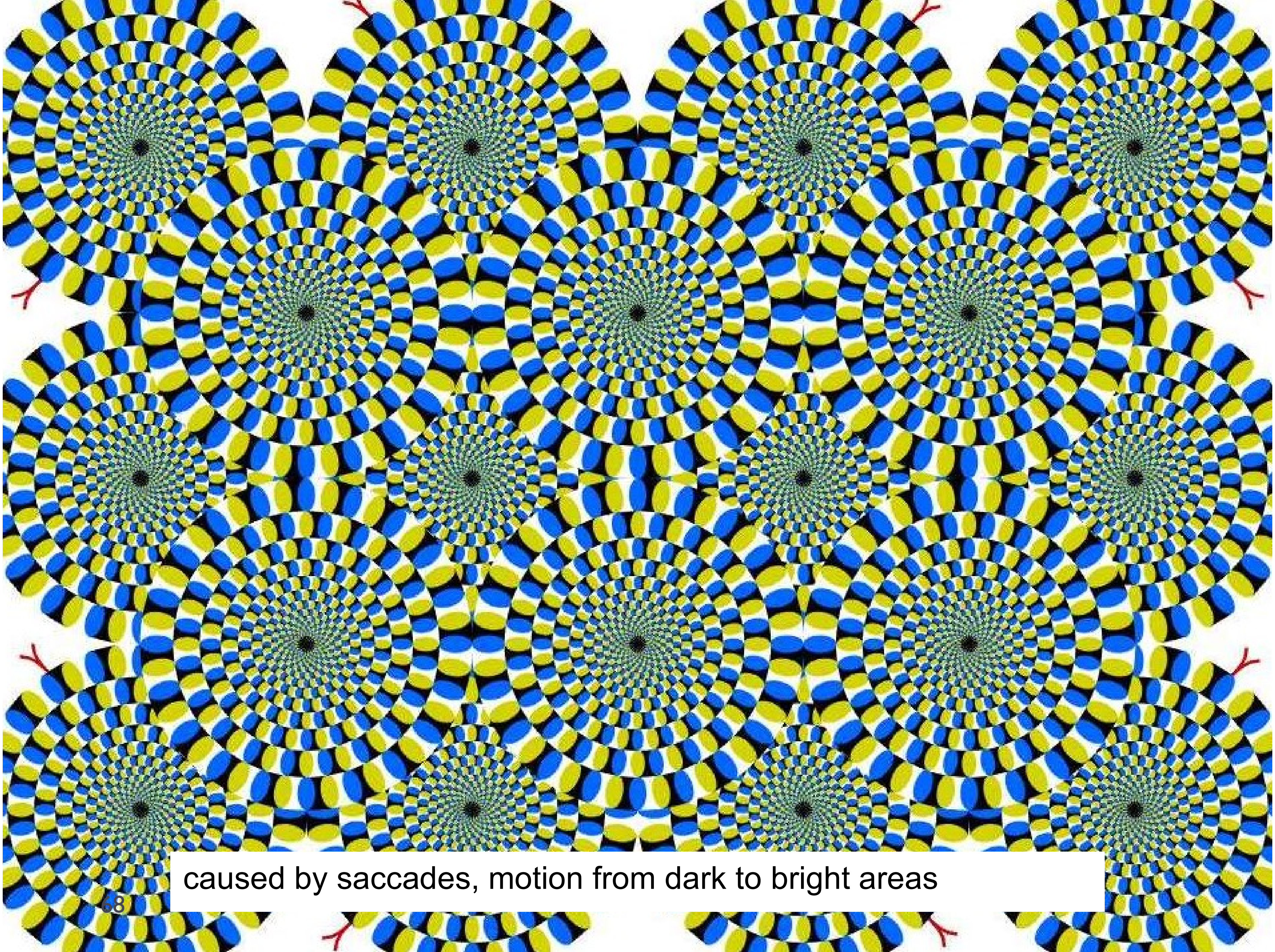
- Escher et.al.
  - Confuse HVS by presenting contradicting visual clues
  - Local vs. global processing







http://www.panoptikum.net/optischetaeuschungen/index.html



caused by saccades, motion from dark to bright areas

Law of closure

References

- Wandell, B. A. (1995). *Foundations of vision*. Sinauer Associates.
- Mantiuk, R. K., Myszkowski, K., & Seidel, H. (2015). High Dynamic Range Imaging. In *Wiley Encyclopedia of Electrical and Electronics Engineering*. Wiley.
  - Section 2.4
  - Available online: http://www.cl.cam.ac.uk/~rkm38/hdri_book.html

Advanced Graphics and Image Processing

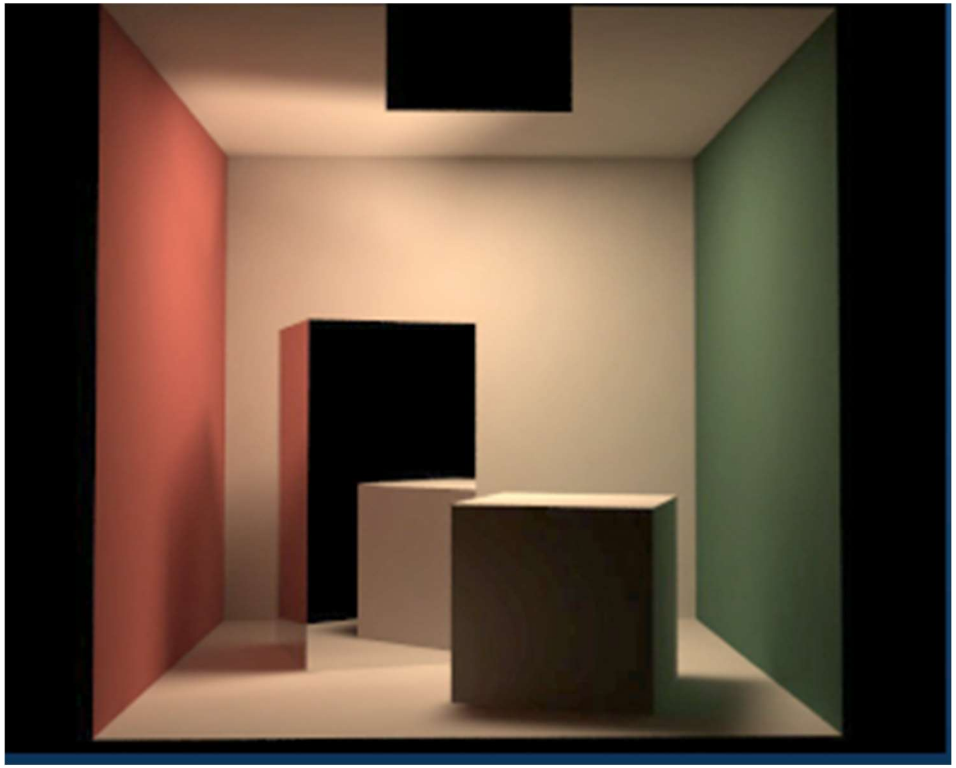
High dynamic range and tone mapping

Part 1/2 – context, the need for tone-mapping

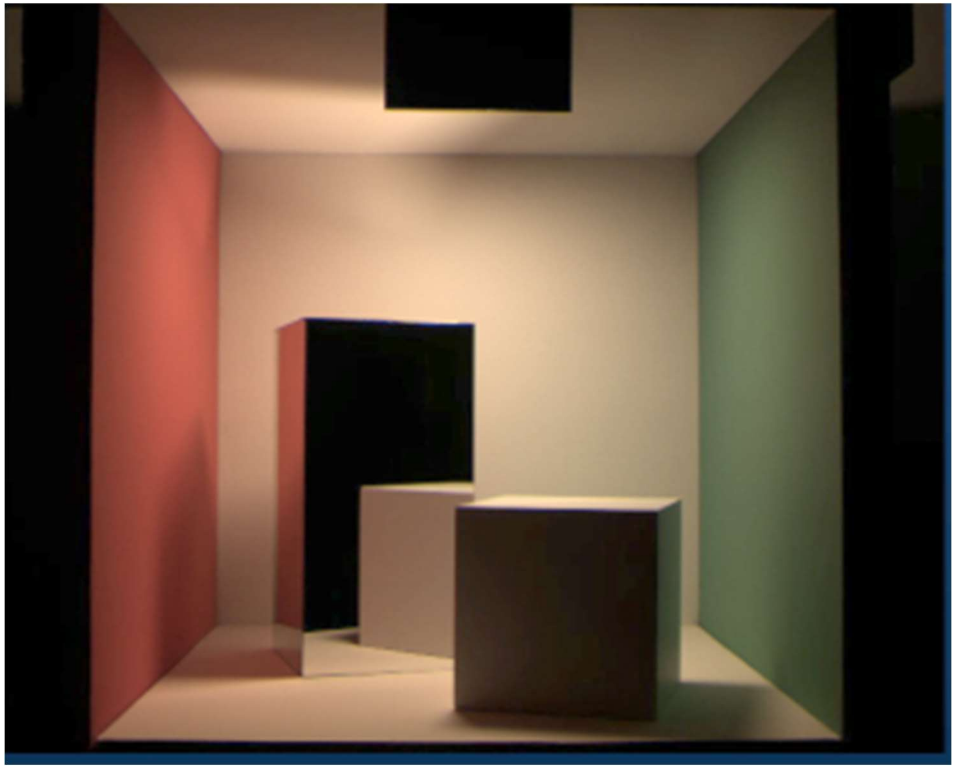
Rafał Mantiuk

Computer Laboratory, University of Cambridge

Cornell Box: need for tone-mapping in graphics

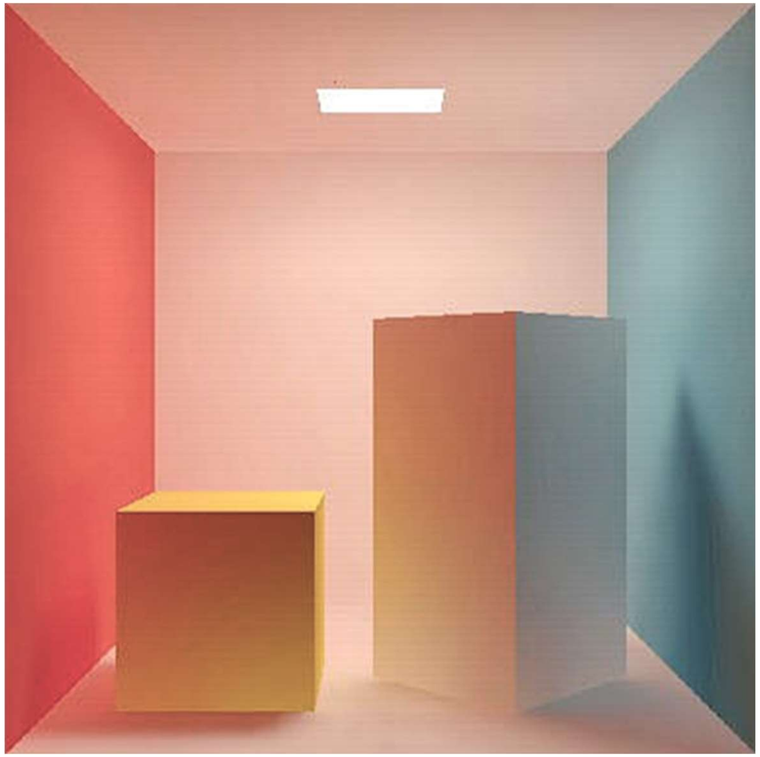


Rendering



Photograph

Real-world scenes are more challenging



- The match could not be achieved if the light source in the top of the box was visible
- The display could not reproduce the right level of brightness

Dynamic range

Luminance

$$\frac{\max L}{\min L}$$

(for SNR>3)

Dynamic range (contrast)

- As ratio:

$$C = \frac{L_{\max}}{L_{\min}}$$

  - Usually written as C:1, for example 1000:1.

- As "orders of magnitude" or log10 units:

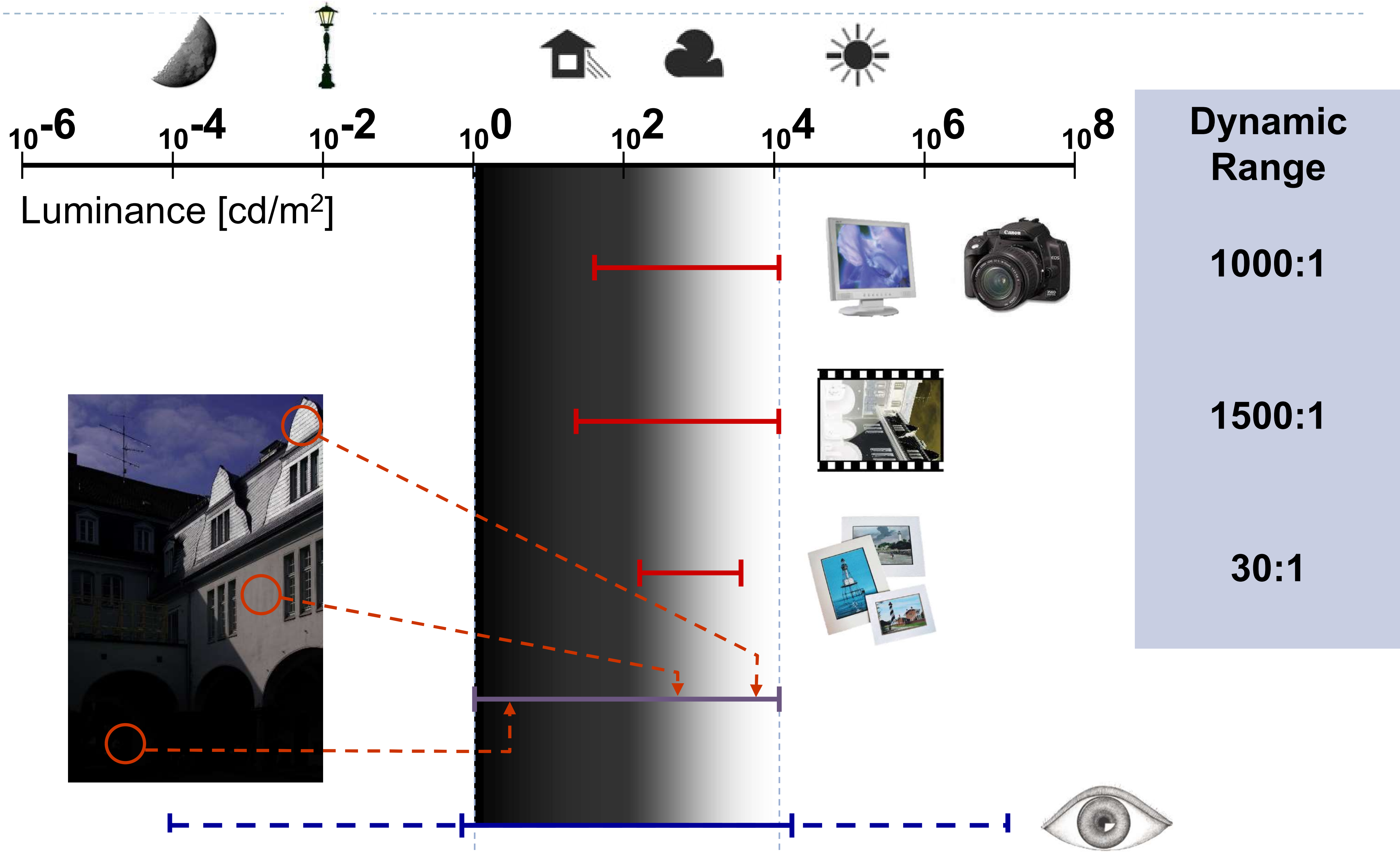
$$C_{10} = \log_{10} \frac{L_{\max}}{L_{\min}}$$

- As stops:

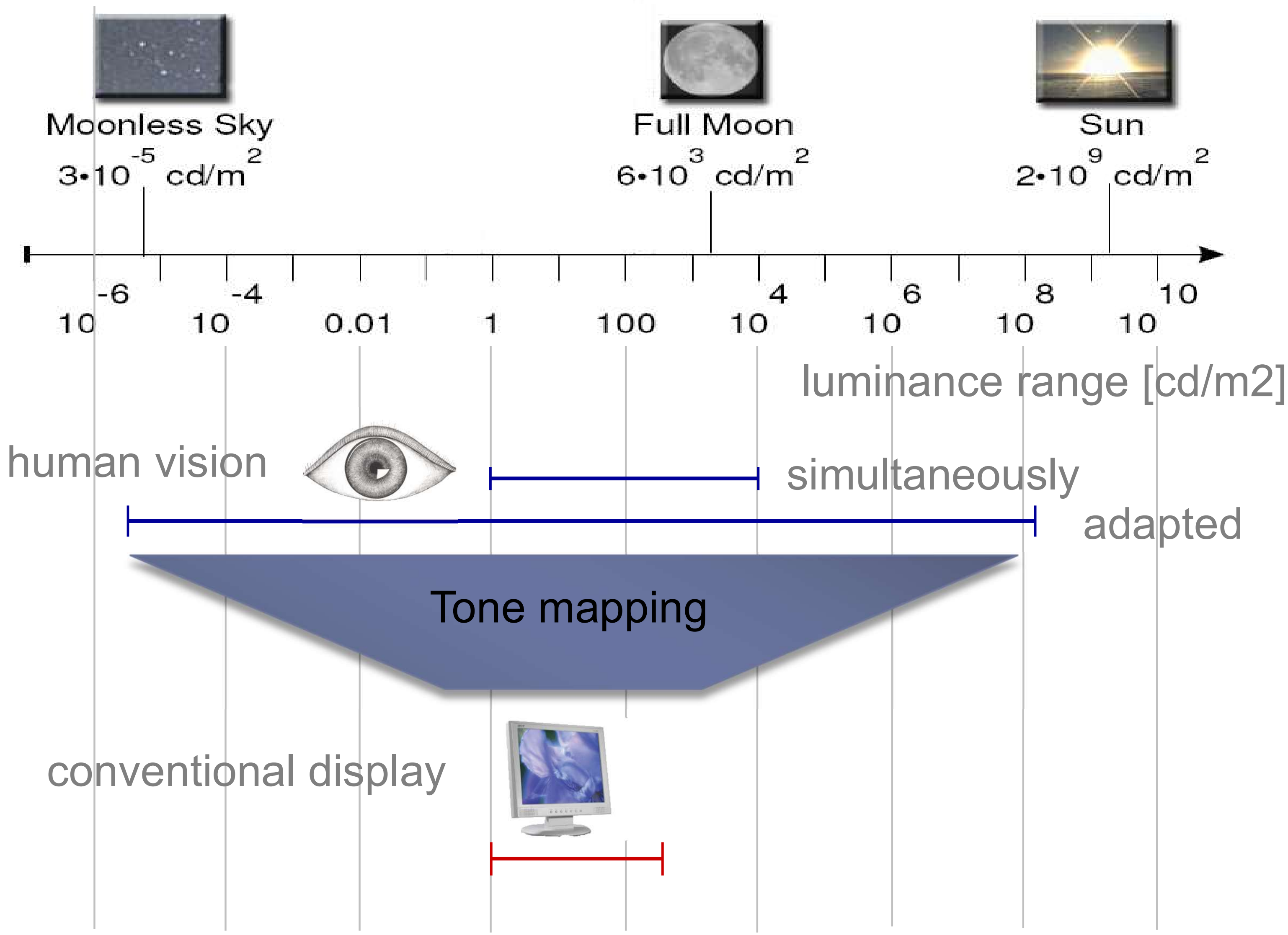
$$C_2 = \log_2 \frac{L_{\max}}{L_{\min}}$$

One stop is doubling of halving the amount of light

High dynamic range (HDR)



Tone-mapping problem



Why do we need tone mapping?

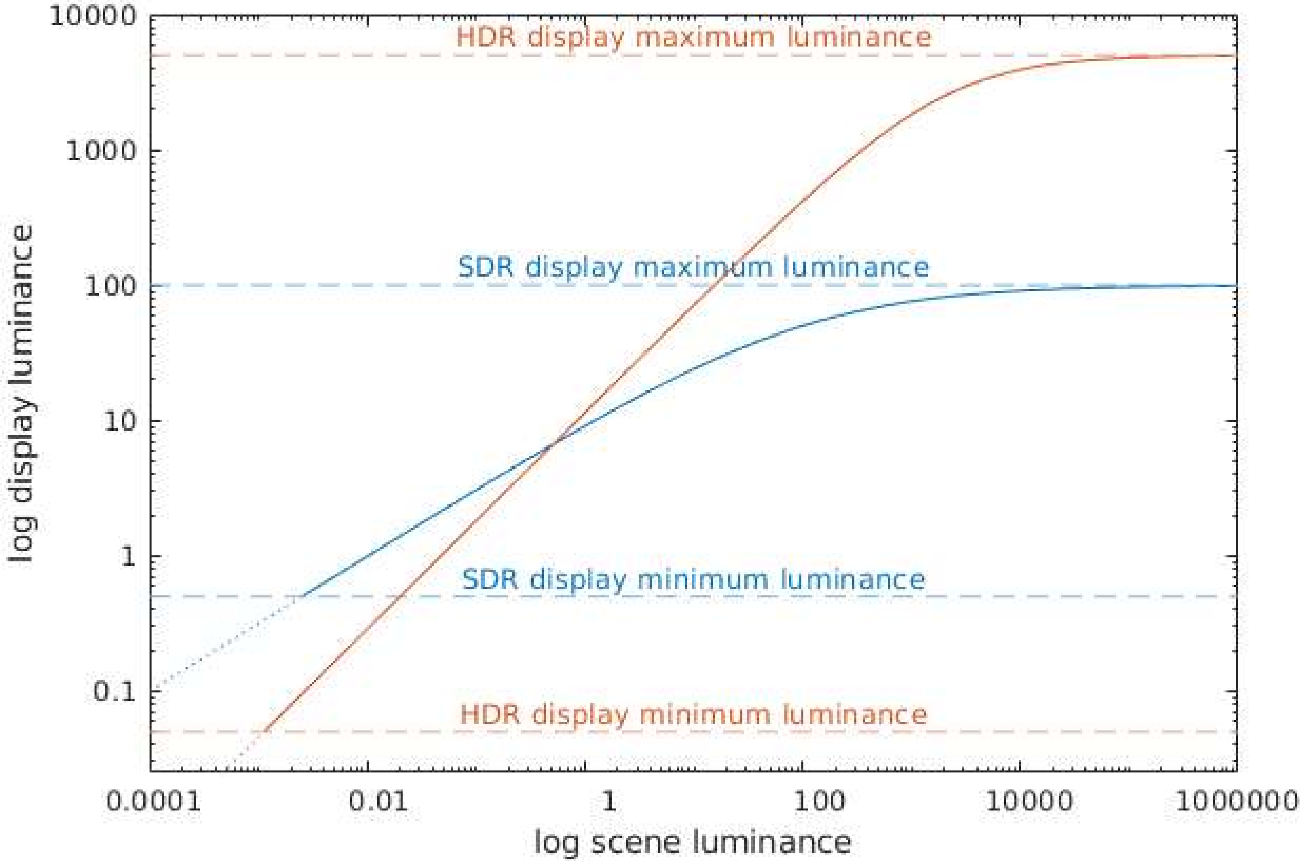
- To **reduce dynamic range**
- To **customize the look**
  - colour grading
- To **simulate human vision**
  - for example night vision

- To adapt displayed images to a **display and viewing conditions**
- To make rendered images look **more realistic**
- To map from **scene- to display-referred** colours

- Different tone mapping operators achieve different goals

From scene- to display-referred colours

- The primary purpose of tone mapping is to transform an image from *scene-referred* to *display-referred* colours



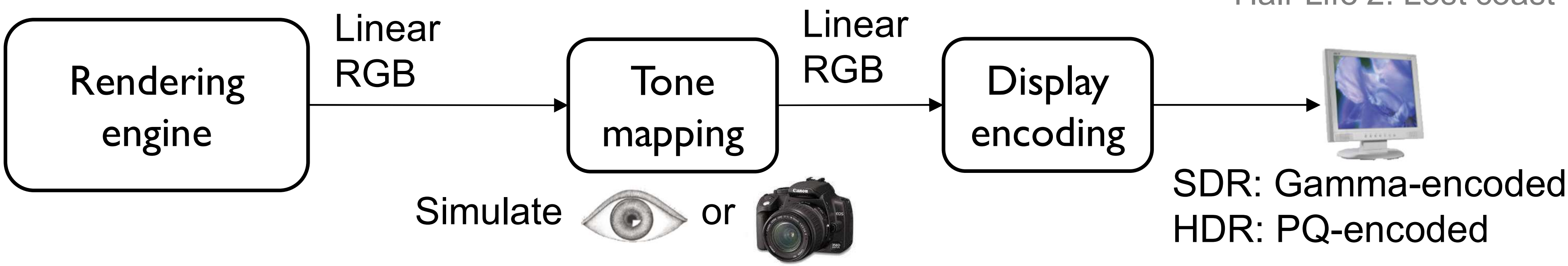
Tone-mapping in rendering

- Any physically-based rendering requires tone-mapping
- “HDR rendering” in games is pseudo-physically-based rendering
- Goal: to simulate a camera or the eye
- Greatly enhances realism

LDR illumination
No tone-mapping

HDR illumination
Tone-mapping

Half-Life 2: Lost coast



Basic tone-mapping and display coding

- The simplest form of tone-mapping is the exposure/brightness adjustment:

Display-referred red value

$$R_d = \frac{R_s}{L_{white}}$$

Scene-referred

Scene-referred luminance of white

  - R for red, the same for green and blue
  - No contrast compression, only for a moderate dynamic range
- The simplest form of display coding is the "gamma"

Prime (') denotes a gamma-corrected value

$$R' = (R_d)^{\frac{1}{\gamma}}$$

Typically γ=2.2

  - For SDR displays only

UNIVERSITY OF CAMBRIDGE
COMPUTER LABORATORY

RAINBOW GRAPHICS GROUP

Advanced Graphics and Image Processing

High dynamic range and tone mapping

Part 2/2 – tone mapping techniques

Rafał Mantiuk

Computer Laboratory, University of Cambridge

Techniques

- Arithmetic of HDR images
- Display model
- Tone-curve
- Color transfer
- Base-detail separation
- Glare

Arithmetic of HDR images

- How do the basic arithmetic operations
  - Addition
  - Multiplication
  - Power function

affect the appearance of an HDR image?

- We work in the luminance space (NOT luma)
- The same operations can be applied to linear RGB
  - Or only to luminance and the colour can be transferred

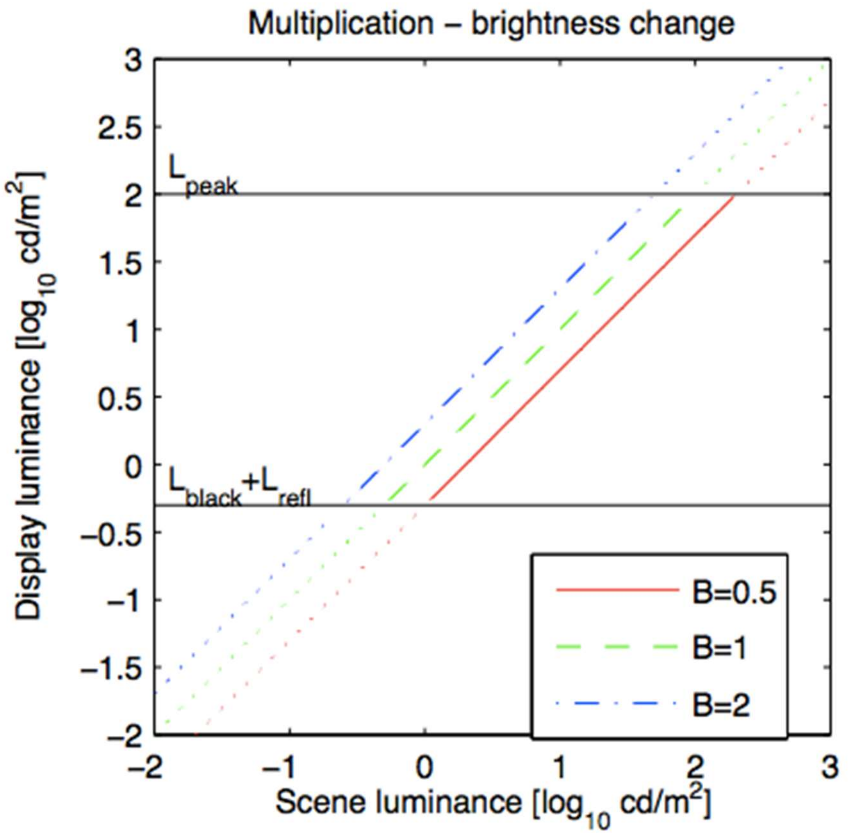
Multiplication – brightness change

Resulting luminance

Input luminance

$$T(L_p) = B \cdot L_p$$

Brightness change parameter



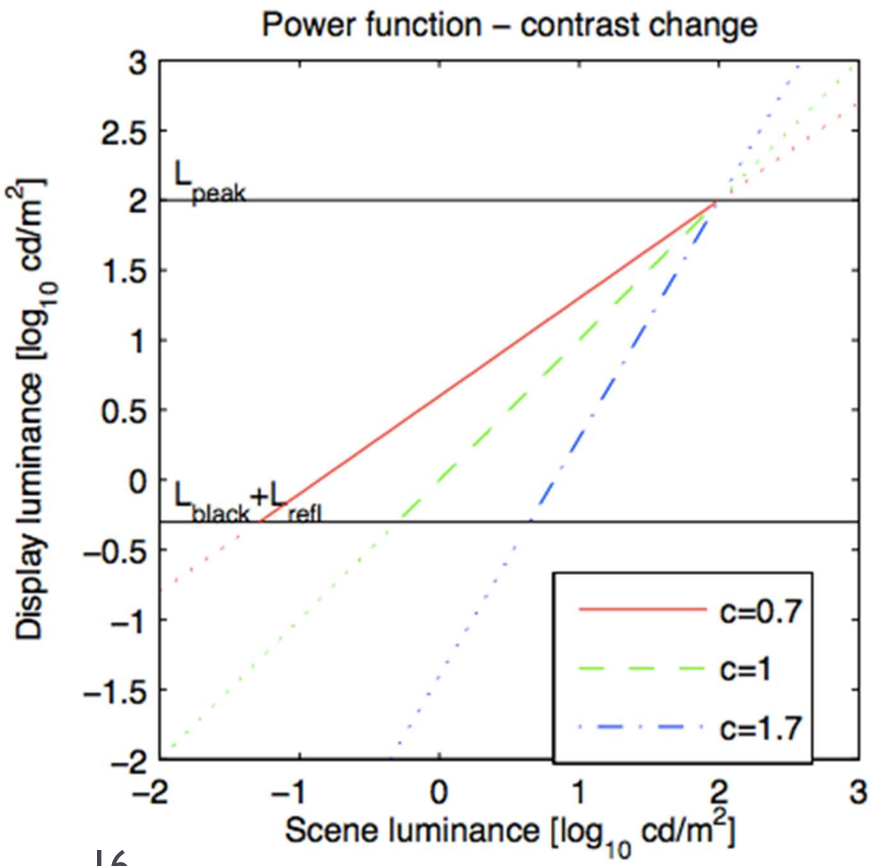
- Multiplication makes the image brighter or darker
- It does not change the dynamic range!

Power function – contrast change

$$T(L_p) = \left(\frac{L_p}{L_{white}}\right)^c$$

Contrast change (gamma)

Luminance of white

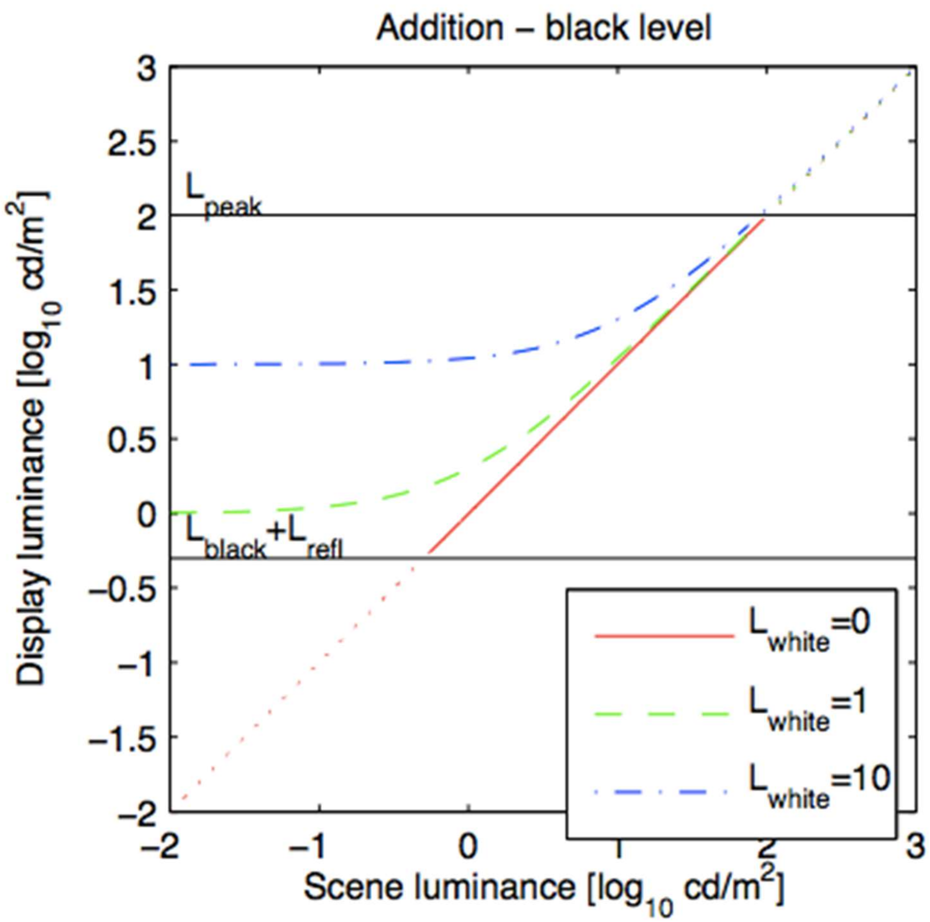


- Power function stretches or shrinks image dynamic range
- It is usually performed relative to a reference white colour/luminance
- Apparent brightness changes is the side effect of pushing tones towards or away from the white point
- Slope on a log-log plot explains contrast change

Addition – black level

$$T(L_p) = L_p + F$$

Black level (flare, fog)



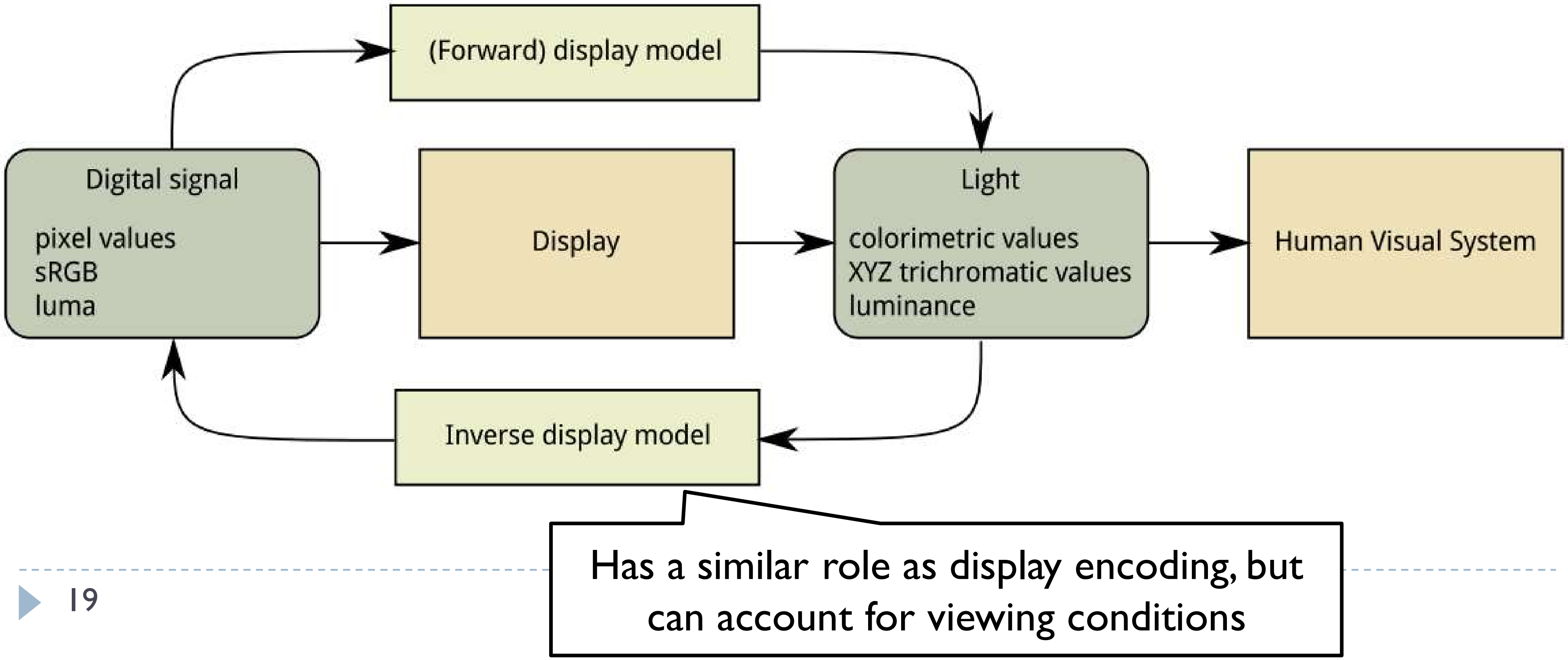
- Addition elevates black level, adds „fog" to an image
- It affects mostly darker tones
- It reduces image dynamic range

Techniques

- Arithmetic of HDR images
- Display model
- Tone-curve
- Color transfer
- Base-detail separation
- Glare

Display-adaptive tone mapping

- Tone-mapping can account for the physical model of a display
  - How a display transforms pixel values into emitted light
  - Useful for ambient light compensation



(Forward) Display model

- GOG: Gain-Gamma-Offset

Display black level

Luminance

Peak luminance

Gamma

Screen reflections

$$L = \underbrace{(L_{peak} - L_{black})}_{\text{Gain}} V^{\gamma} + \underbrace{L_{black} + L_{refl}}_{\text{Offset}}$$

Pixel value 0-1

Reflectance factor (0.01)

$$L_{refl} = \frac{k}{\pi} E_{amb}$$

Ambient illumination (in lux)

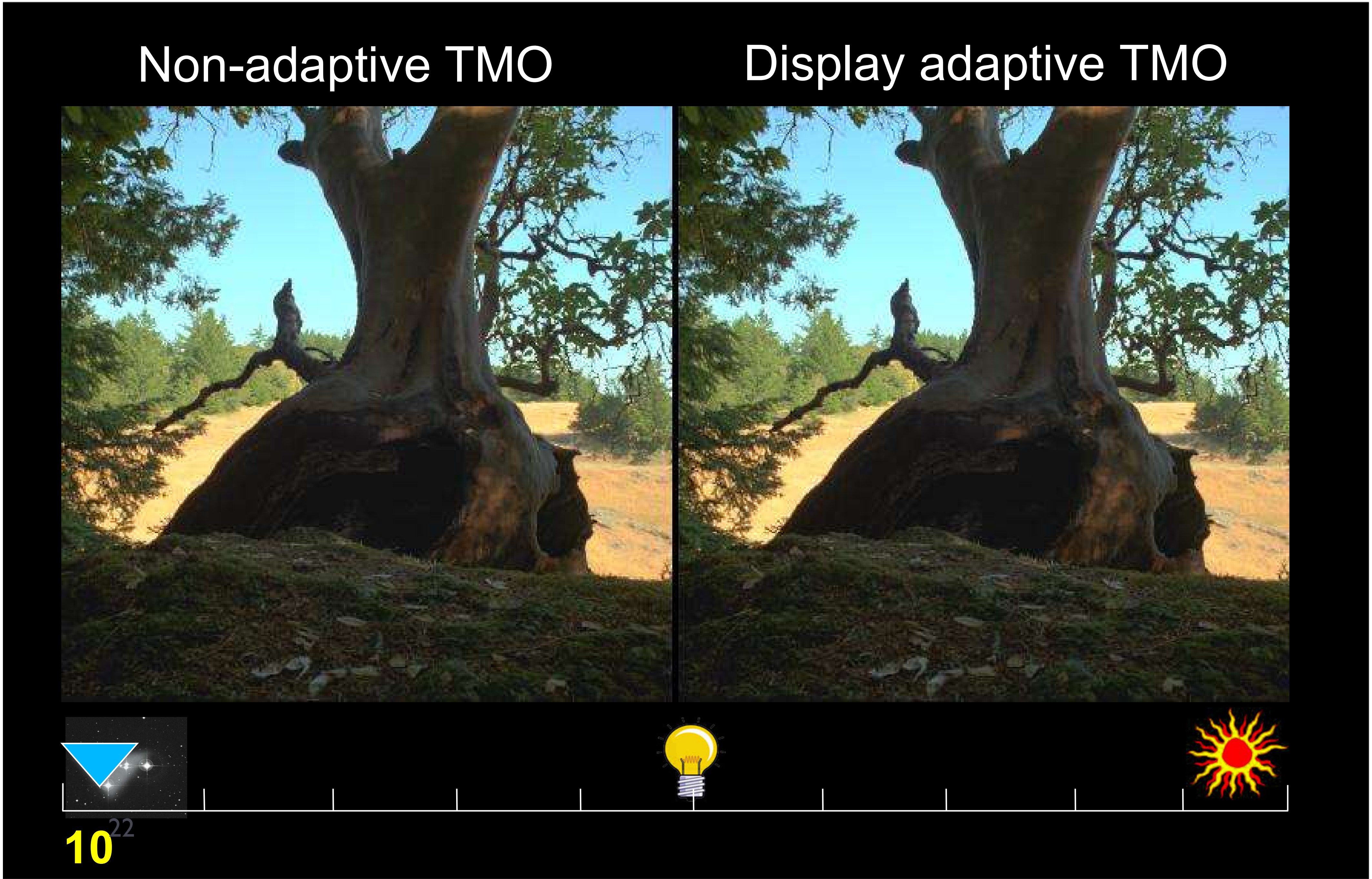
Inverse display model

Symbols are the same as for the forward display model

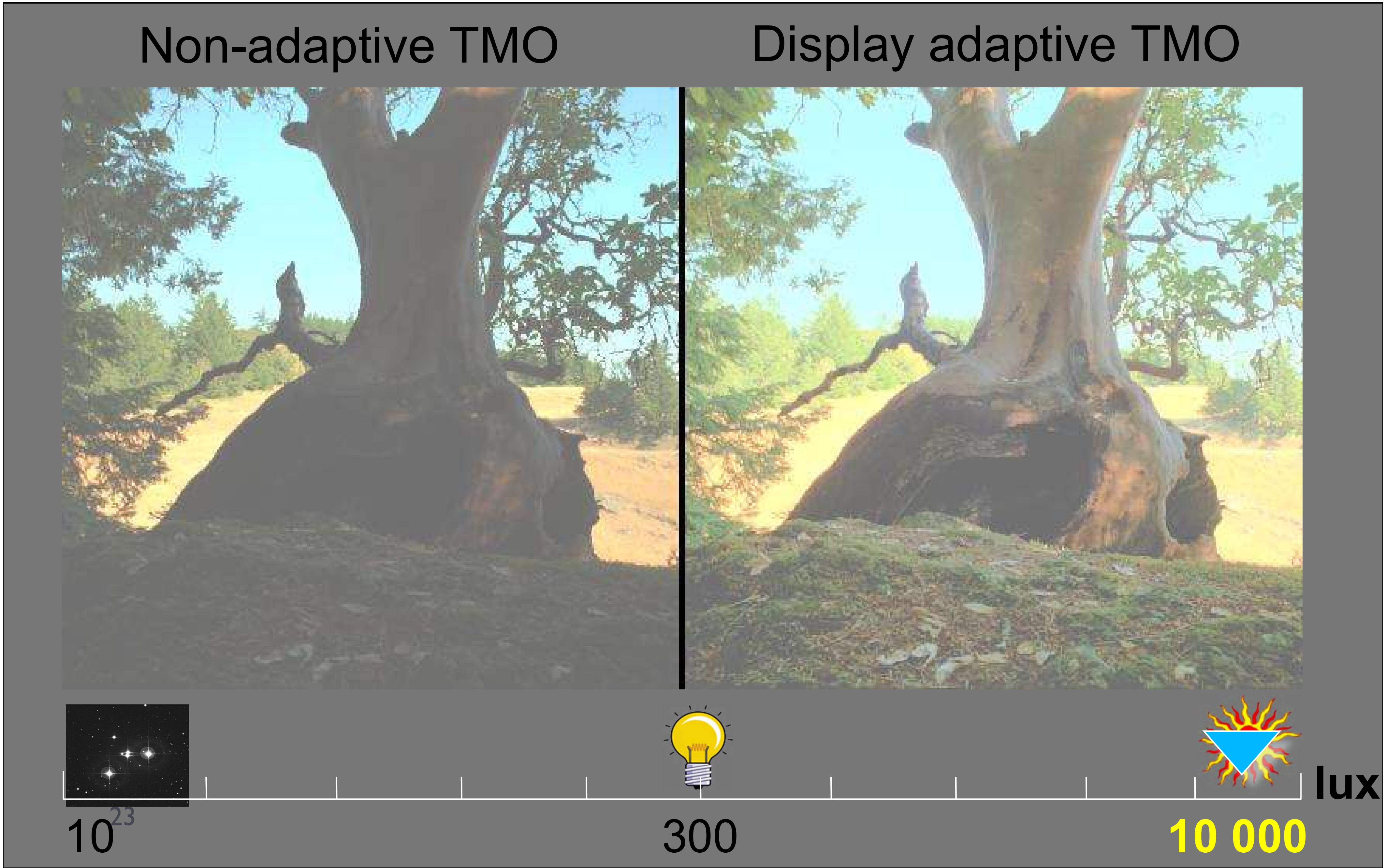
$$V = \left(\frac{L - L_{black} - L_{refl}}{L_{peak} - L_{black}}\right)^{(1/\gamma)}$$

Note: This display model does not address any colour issues. The same equation is applied to red, green and blue color channels. The assumption is that the display primaries are the same as for the sRGB color space.

Ambient illumination compensation



Ambient illumination compensation



Example: Ambient light compensation

- We are looking at the screen in bright light

$$L_{peak} = 100\ [cd \cdot m^{-2}] \qquad k = 0.005$$

Modern screens have reflectivity of around 0.5%

$$L_{black} = 0.1\ [cd \cdot m^{-2}]$$

$$E_{amb} = 2000\ [lux] \qquad L_{refl} = \frac{0.005}{\pi} 2000 = 3.183\ [cd \cdot m^{-2}]$$

- We assume that the dynamic of the input is 2.6 (≈400:1)

$$r_{in} = 2.6 \qquad r_{out} = \log_{10} \frac{L_{peak}}{L_{black} + L_{refl}} = 1.77$$

- First, we need to compress contrast to fit the available dynamic range, then compensate for ambient light

$$L_{out} = \left(\frac{L_{in}}{L_{wp}}\right)^{\frac{r_{out}}{r_{in}}} - L_{refl}$$

The resulting value is in luminance, must be mapped to display luma / gamma corrected values (display encoded)

Simplest, but not the best tone mapping

Techniques

- Arithmetic of HDR images
- Display model
- Tone-curve
- Color transfer
- Base-detail separation
- Glare

Tone-curve

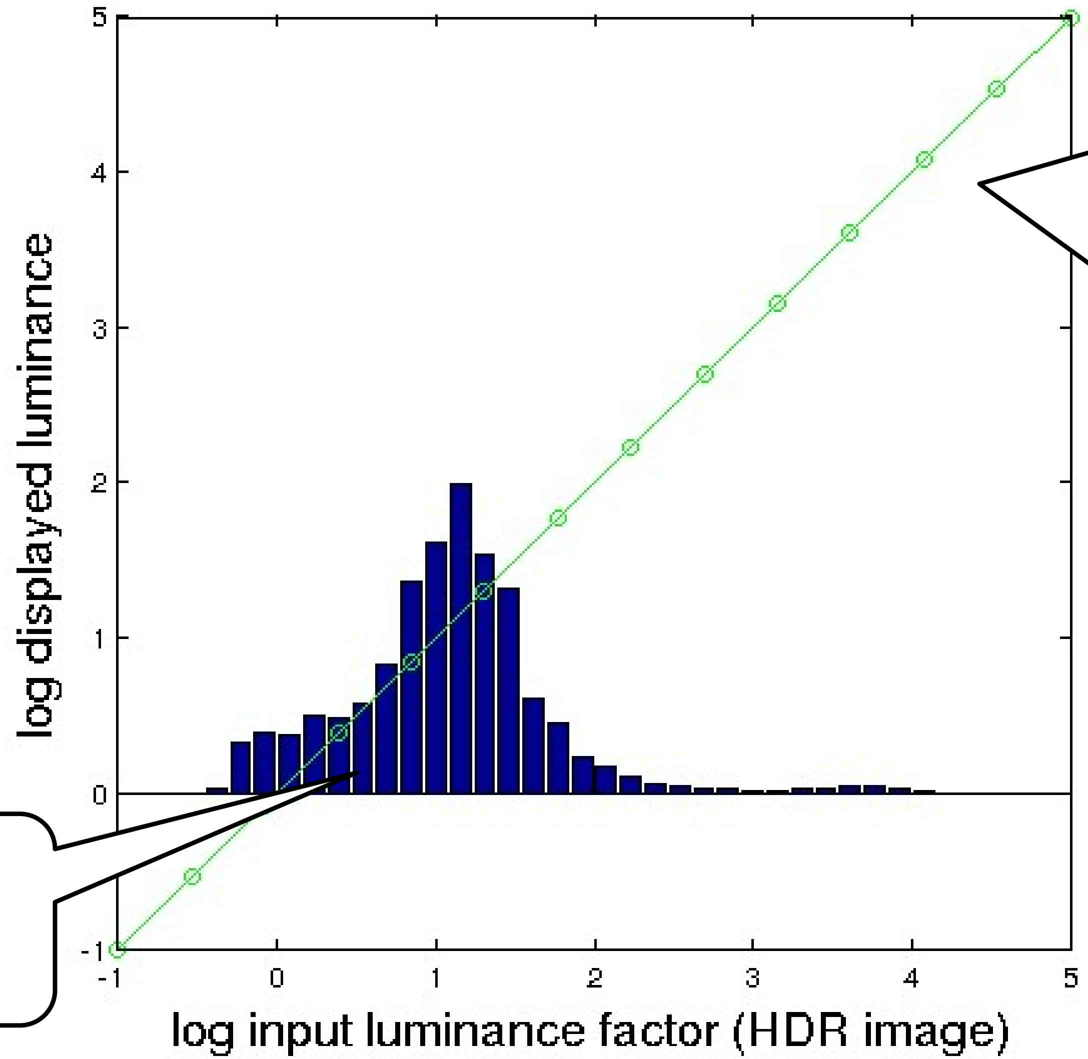
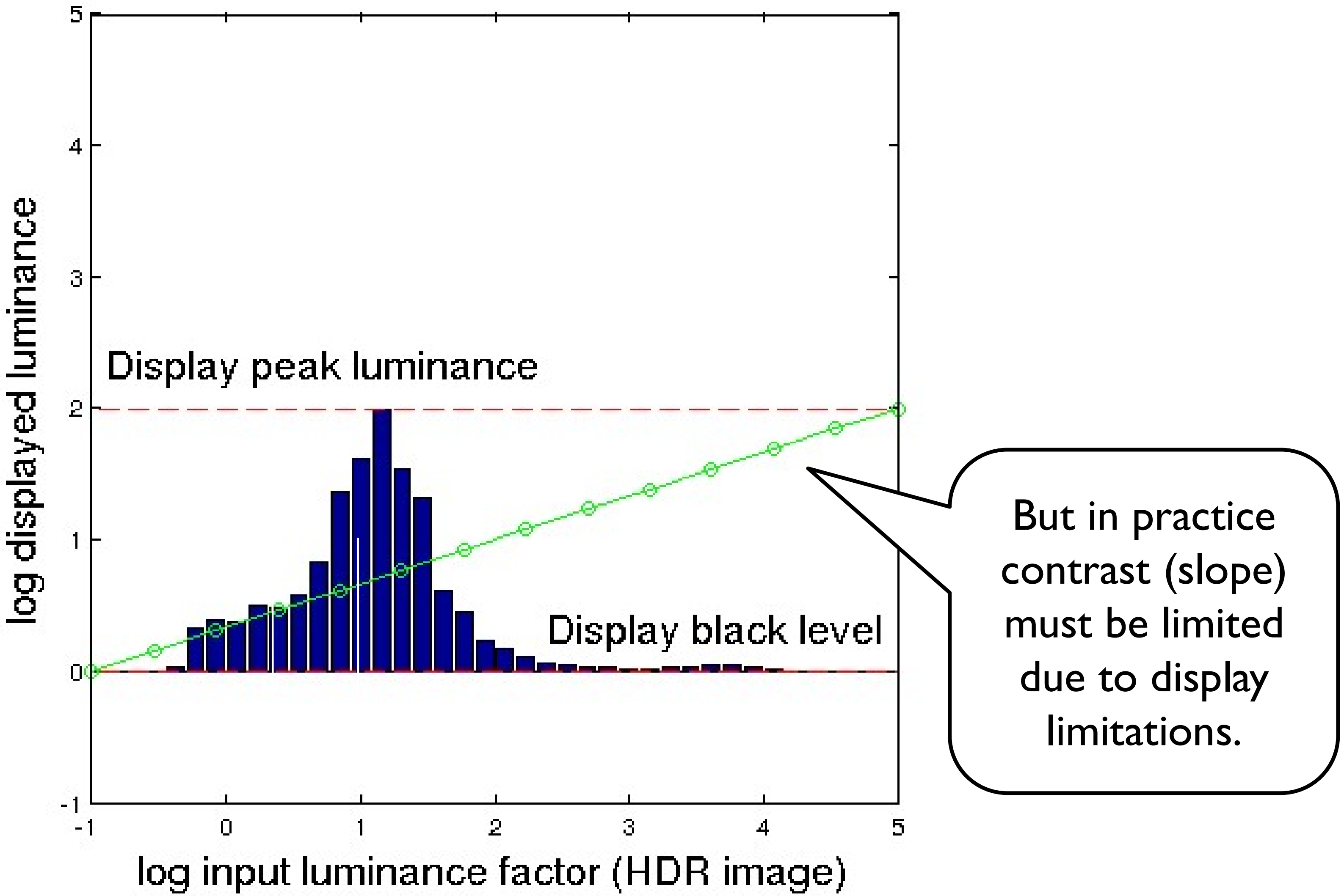


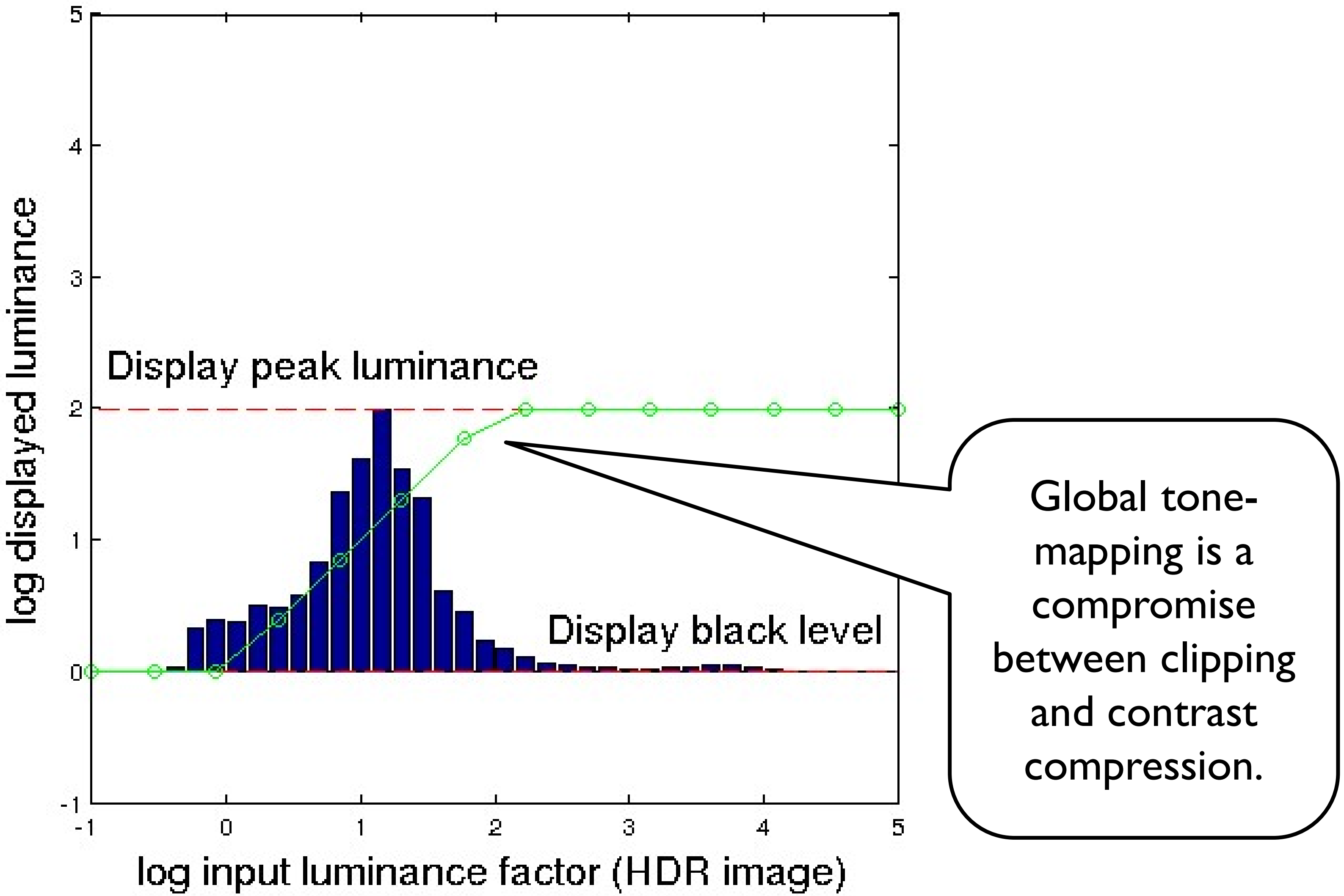
Image histogram

Best tone-mapping is the one which does not do anything, i.e. slope of the tone-mapping curves is equal to 1.

Tone-curve

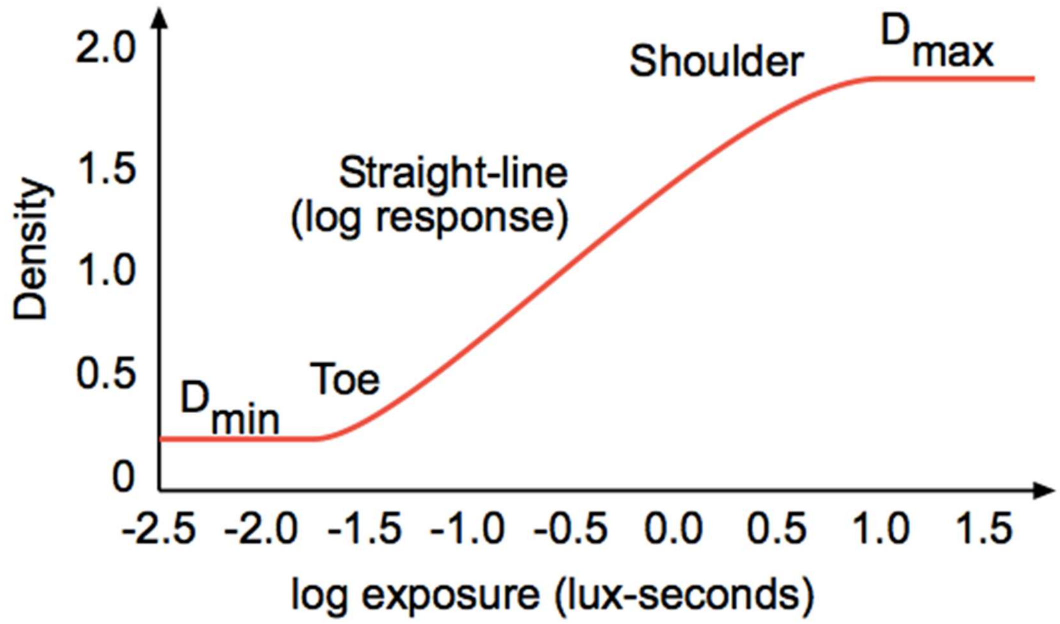

Display peak luminance
Display black level
log displayed luminance
log input luminance factor (HDR image)
5
4
3
2
1
0
-1
-1
0
1
2
3
4
5
But in practice contrast (slope) must be limited due to display limitations.

Tone-curve


Display peak luminance
Display black level
log displayed luminance
log input luminance factor (HDR image)
5
4
3
2
1
0
-1
-1
0
1
2
3
4
5
Global tone-mapping is a compromise between clipping and contrast compression.

Sigmoidal tone-curves

- Very common in digital cameras
  - Mimic the response of analog film
  - Analog film has been engineered over many years to produce good tone-reproduction
- Fast to compute



Sigmoidal tone mapping

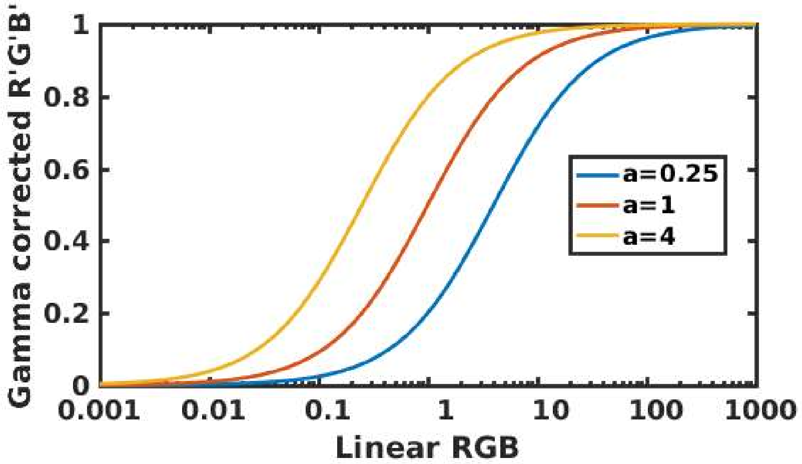
- Simple formula for a sigmoidal tone-curve:

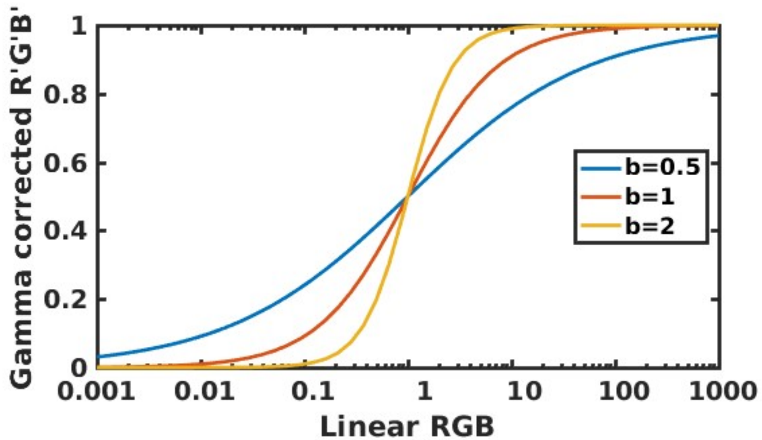
$$R'(x,y) = \frac{R(x,y)^b}{\left(\frac{L_m}{a}\right)^b + R(x,y)^b}$$

where L_m is the geometric mean (or mean of logarithms):

$$L_m = exp\left(\frac{1}{N}\sum_{(x,y)} \ln(L(x,y))\right)$$

and $L(x,y)$ is the luminance of the pixel (x,y).





Sigmoidal tone mapping example

a=0.25

a=1

a=4

b=0.5 b=1 b=2

Histogram equalization

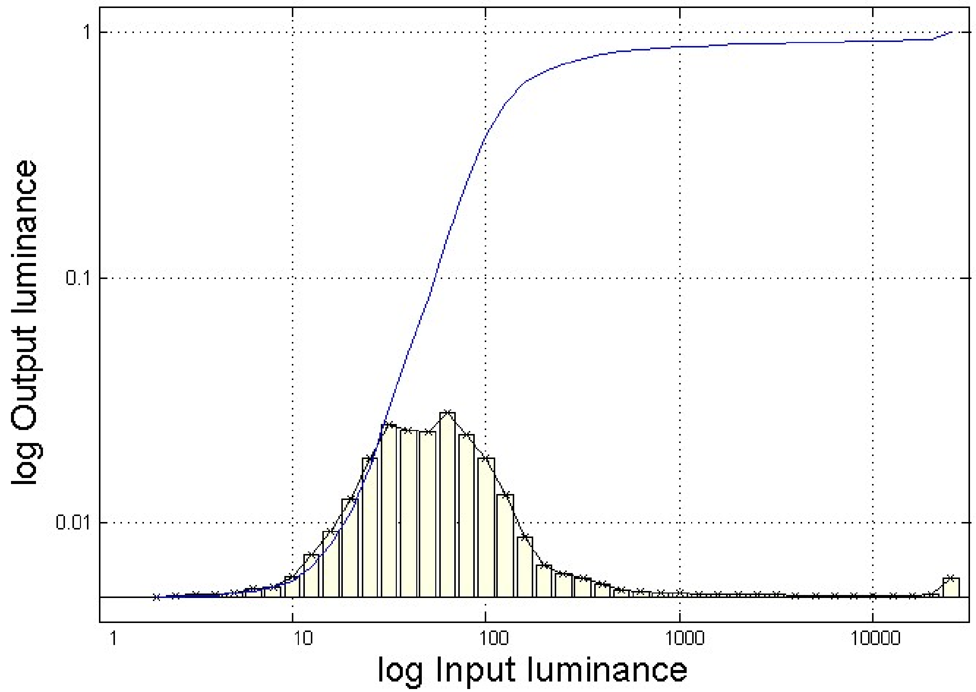
- 1. Compute normalized cummulative image histogram

$$c(I) = \frac{1}{N}\sum_{i=0}^{I} h(i) = c(I-1) + \frac{1}{N}h(I)$$

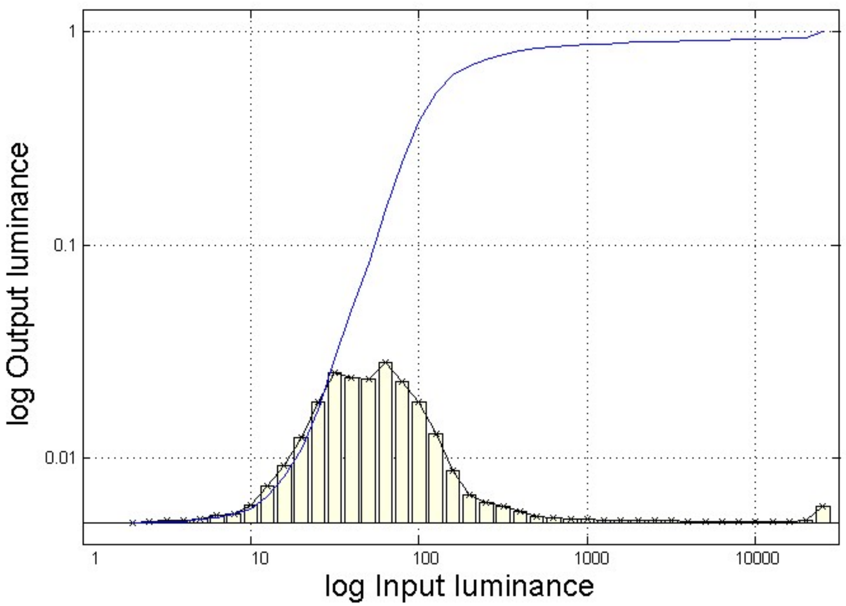
  - For HDR, operate in the log domain
- 2. Use the cummulative histogram as a tone-mapping function

$$Y_{out} = c(Y_{in})$$

  - For HDR, map the log-10 values to the [-dr_{out} ; 0] range
    - where dr_{out} is the target dynamic range (of a display)



Histogram equalization



- Steepest slope for strongly represented bins
  - If many pixels have the same value - enhance contrast
  - Reduce contrast, if few pixels
- Histogram Equalization distributes contrast distortions relative to the “importance” of a brightness level

Histogram adjustment with a linear ceiling

- [Larson et al. 1997, IEEE TVCG]

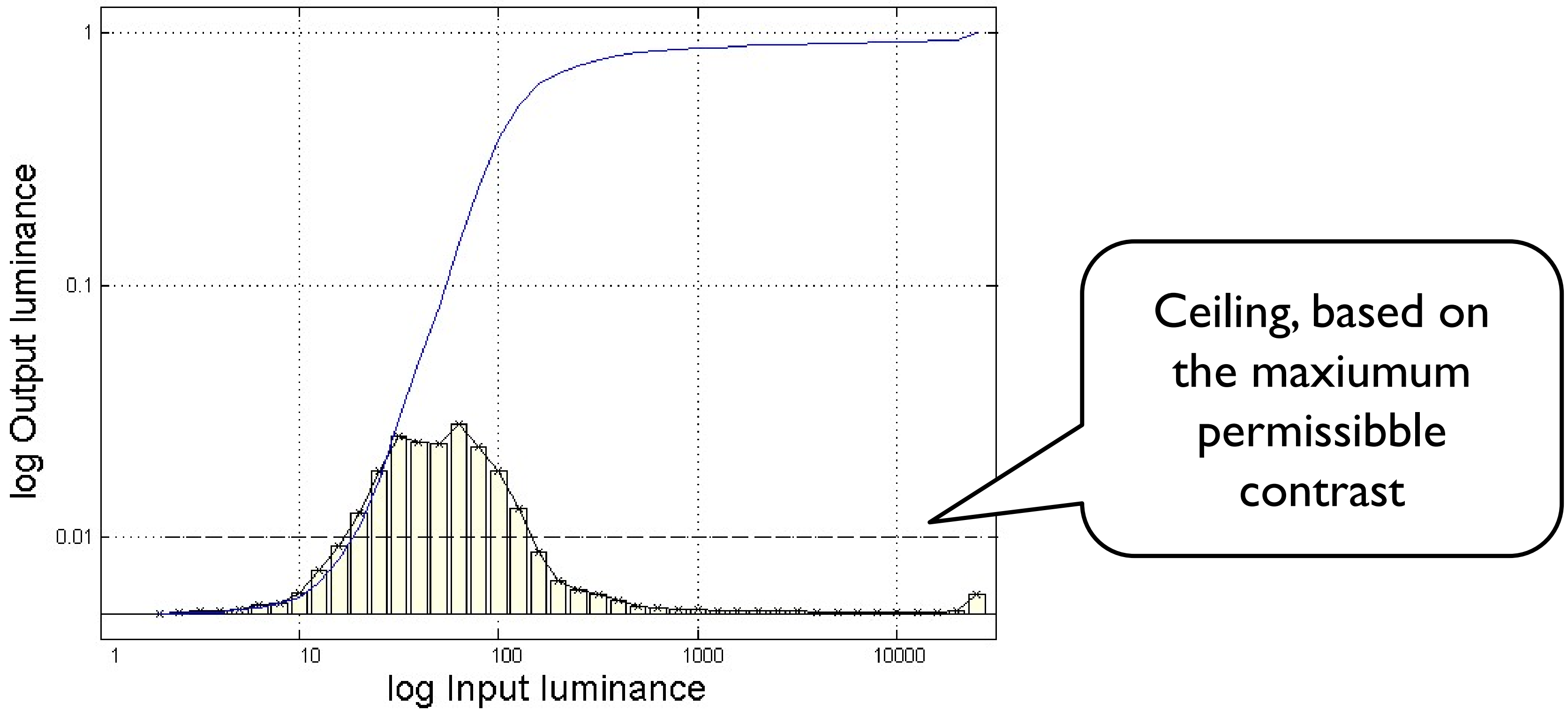
Linear mapping

Histogram equalization

Histogram equalization with a ceiling

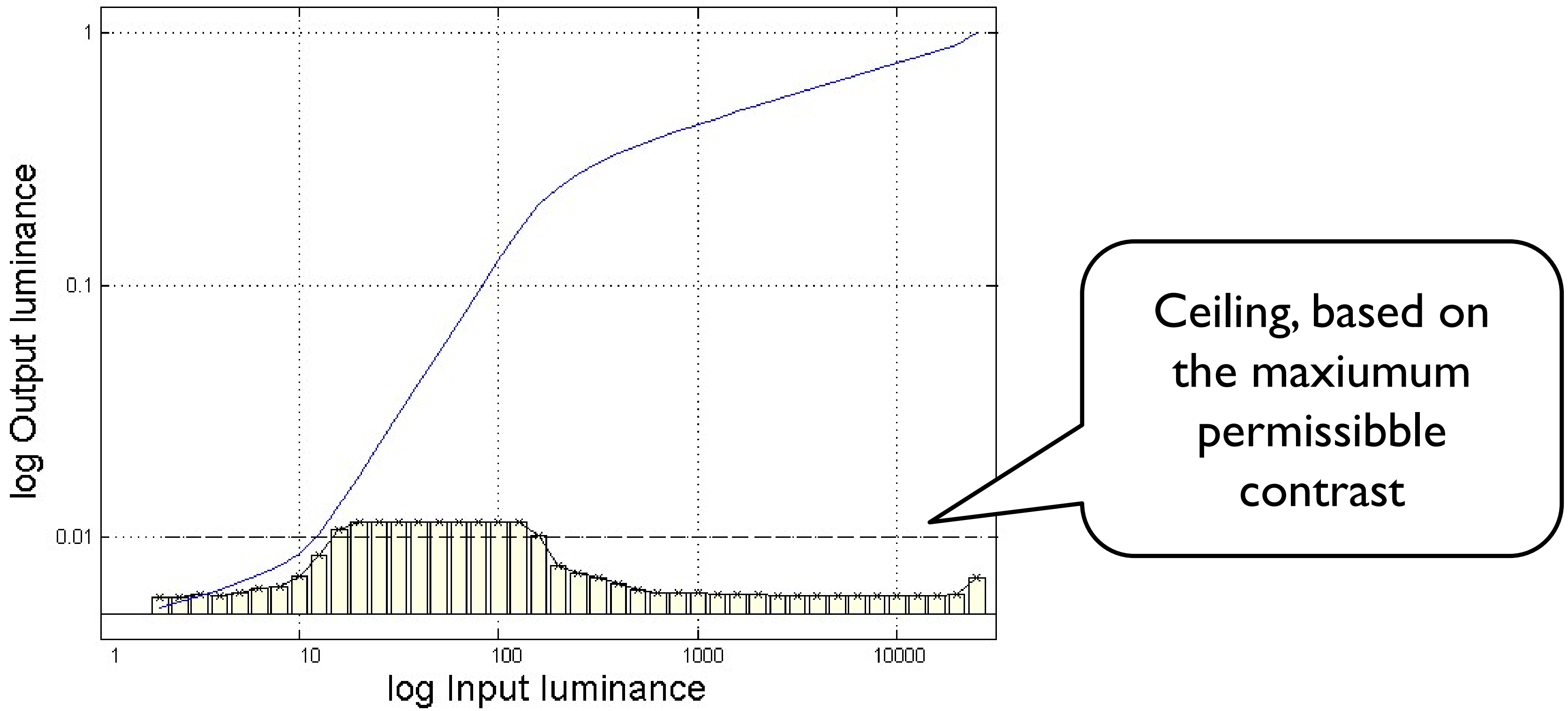
Histogram adjustment with a linear ceiling

- Truncate the bins that exceed the ceiling;
- Distribute the removed counts to all bins;
- Repeat until converges



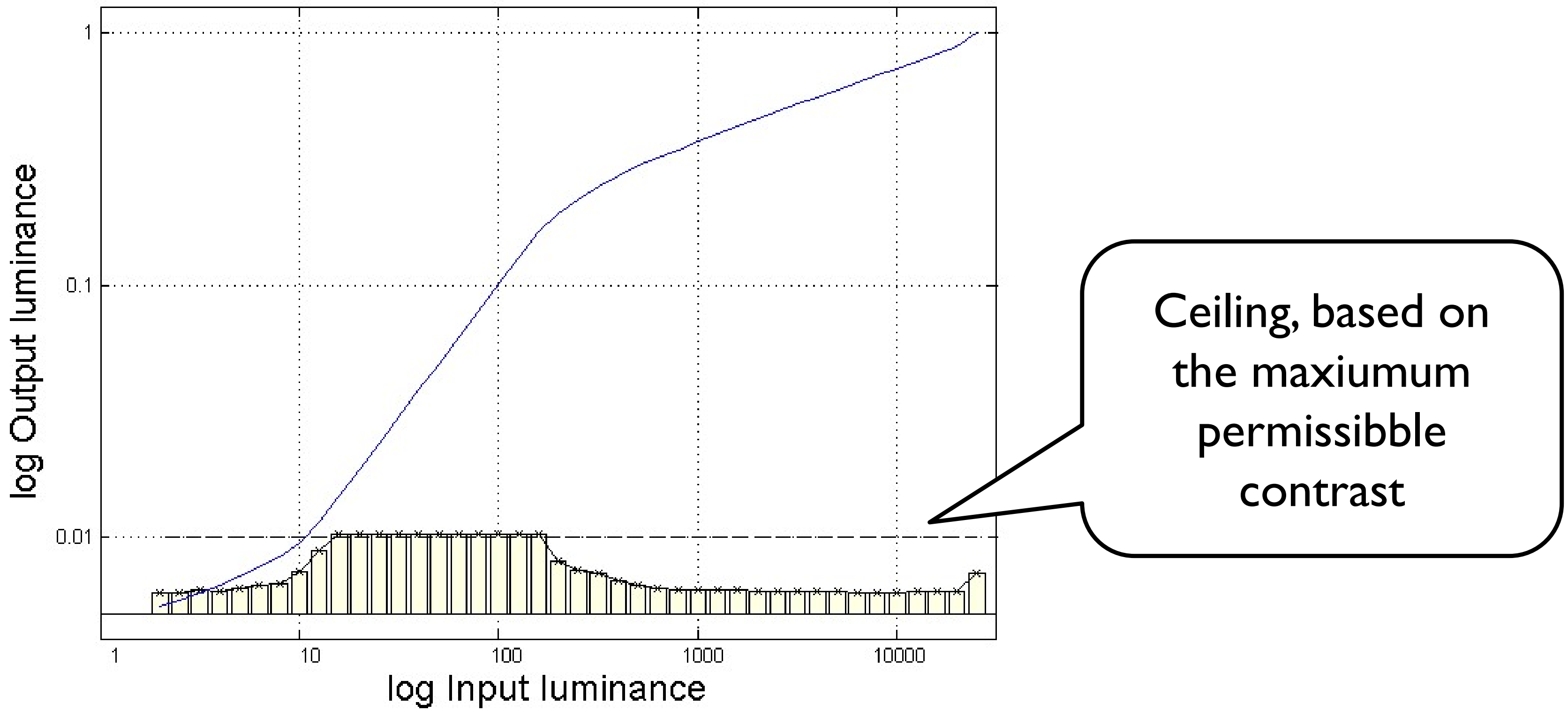
Histogram adjustment with a linear ceiling

- Truncate the bins that exceed the ceiling;
- Distribute the removed counts to all bins;
- Repeat until converges



Histogram adjustment with a linear ceiling

- Truncate the bins that exceed the ceiling;
- Distribute the removed counts to all bins;
- Repeat until converges



Techniques

- Arithmetic of HDR images
- Display model
- Tone-curve
- Color transfer
- Base-detail separation
- Glare

Colour transfer in tone-mapping

- Many tone-mapping operators work on luminance, mean or maximum colour channel value
  - For speed
  - To avoid colour artefacts
- Colours must be transferred later form the original image
- Colour transfer in the linear RGB colour space:

Output color channel (red)

$$R_{out} = \left(\frac{R_{in}}{L_{in}} \right)^{s} \cdot L_{out}$$

Saturation parameter

Resulting luminance

- The same formula applies to green (G) and blue (B) linear colour values

Colour transfer: out-of-gamut problem

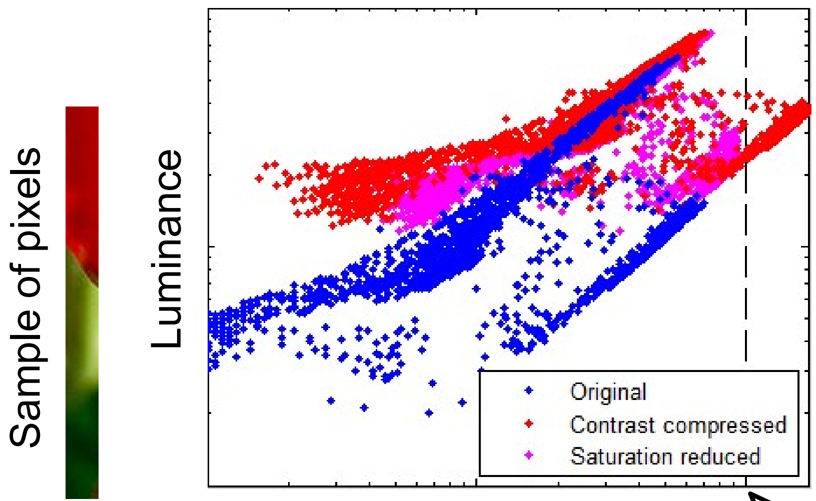
- Colours often fall outside the colour gamut when contrast is compressed

Original image

Contrast reduced (s=1)

Saturation reduced (s=0.6)

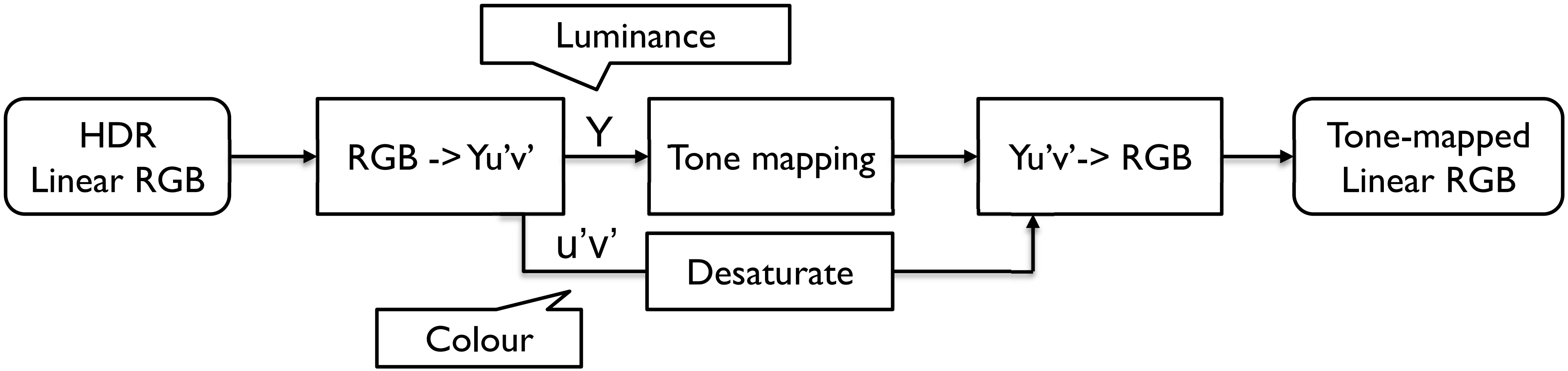
Colours before/after processing



- Reduction in saturation is needed to bring the colors into gamut

Colour transfer: alternative method

- Colour transfer in linear RGB will alter resulting luminance
- Colours can be also transferred and saturation adjusted using CIE u'v' chromatic coordinates



Chroma of the white

- To correct saturation:

$$u'_{out} = (u'_{in} - u'_w) \cdot s + u'_w \qquad u'_w = 0.1978$$

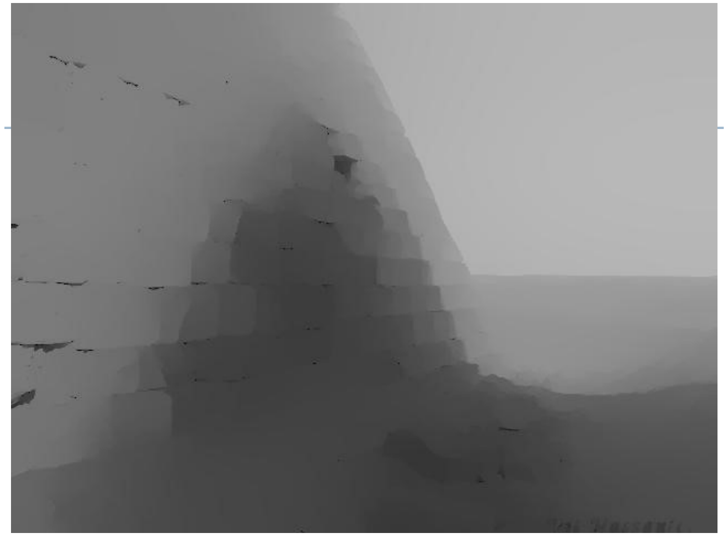
$$v'_{out} = (v'_{in} - v'_w) \cdot s + v'_w \qquad v'_w = 0.4683$$

Techniques

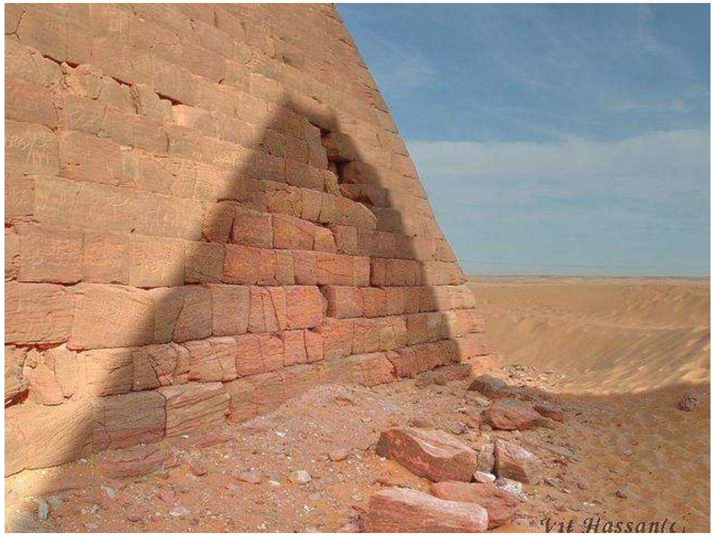
- Arithmetic of HDR images
- Display model
- Tone-curve
- Color transfer
- Base-detail separation
- Glare

Illumination & reflectance separation

Input



Illumination



Reflectance

$$Y = I \cdot R$$

Image

Illumination

Reflectance

Illumination and reflectance

Reflectance

- White ≈ 90%
- Black ≈ 3%
- Dynamic range < 100:1
- Reflectance critical for object & shape detection

Illumination

- Sun ≈ 10^9 cd/m^2
- Lowest perceivable luminance ≈ 10^{-6} cd/m^2
- Dynamic range 10,000:1 or more
- Visual system partially discounts illumination

Reflectance & Illumination TMO

- Hypothesis: *Distortions in reflectance are more apparent than the distortions in illumination*
- Tone mapping could preserve reflectance but compress illumination

Illumination

Tone-mapped image

$$L_d = R \cdot T(I)$$

Reflectance

Tone-mapping

- for example:

$$L_d = R \cdot (I / L_{white})^c \cdot L_{white}$$

How to separate the two?

- (Incoming) illumination – slowly changing
  - except very abrupt transitions on shadow boundaries
- Reflectance – low contrast and high frequency variations

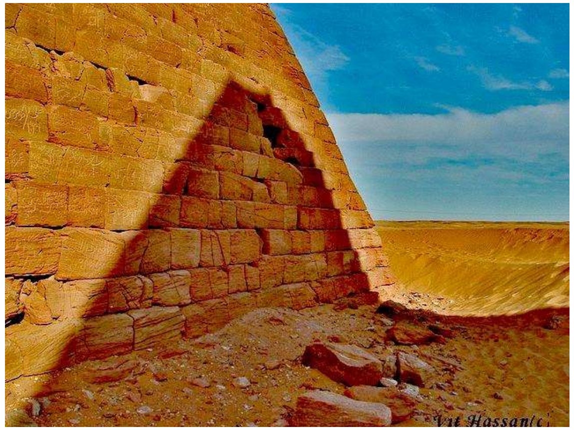
Gaussian filter

$$f(x) = \frac{1}{2\pi\sigma_s} e^{\frac{-x^2}{2\sigma_s^2}}$$

- First order approximation

- Blurs sharp boundaries
- Causes halos

Tone mapping result



Bilateral filter

$$I_p \approx \frac{1}{k_s} \sum_{t \in \Omega} f(p-t)\, g(L_p - L_t)\, L_p$$

- Better preserves sharp edges

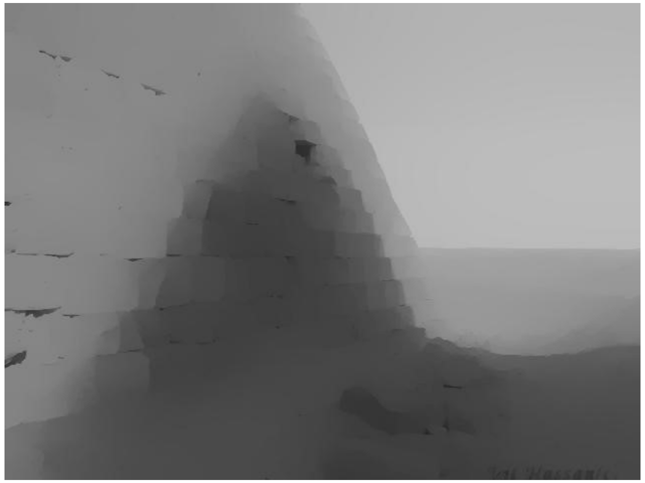
Tone mapping result

- Still some blurring on the edges
- Reflectance is not perfectly separated from illumination near edges

[Durand & Dorsey, SIGGRAPH 2002]

Weighted-least-squares (WLS) filter

- Stronger smoothing and still distinct edges



Tone mapping result

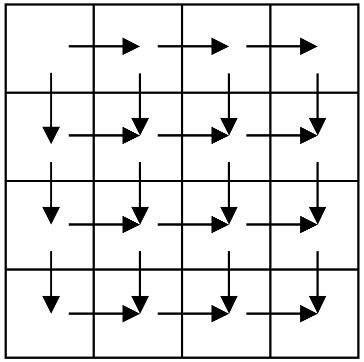
- Can produce stronger effects with fewer artifacts
- See „Advanced image processing” lecture

[Farbman et al., SIGGRAPH 2008]

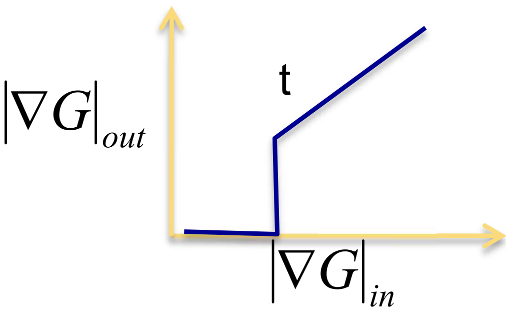
Retinex

- Retinex algorithm was initially intended to separate reflectance from illumination [Land 1964]
  - There are many variations of Retinex, but the general principle is to eliminate from an image small gradients, which are attributed to the illumination

1 step: compute gradients in log domain



2nd step: set to 0 gradients less than the threshold



3rd step: reconstruct an image from the vector field

$$\nabla^2 I = \operatorname{div} G$$

For example by solving the Poisson equation

Retinex examples

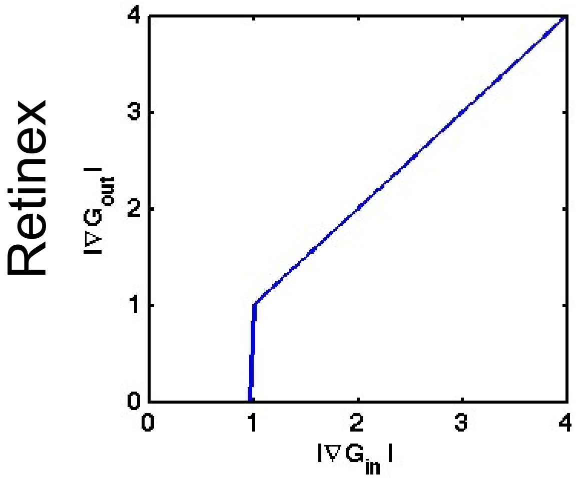
From: http://dragon.larc.nasa.gov/retinex/757/

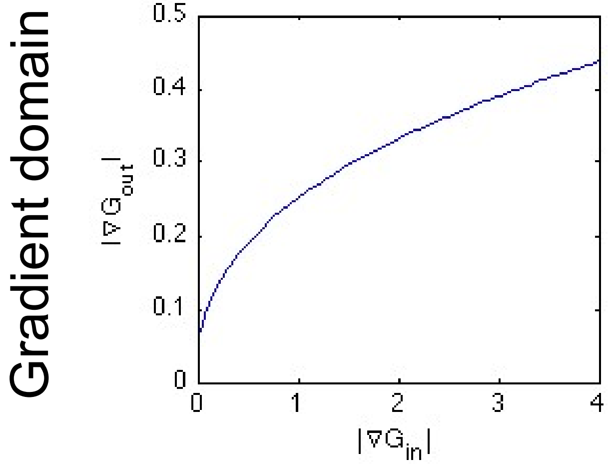
From:http://www.ipol.im/pub/algo/lmps_retinex_poisson_equation/#ref_1

Gradient domain HDR compression

[Fattal et al., SIGGRAPH 2002]

- Similarly to Retinex, it operates on log-gradients
- But the function amplifies small contrast instead of removing it



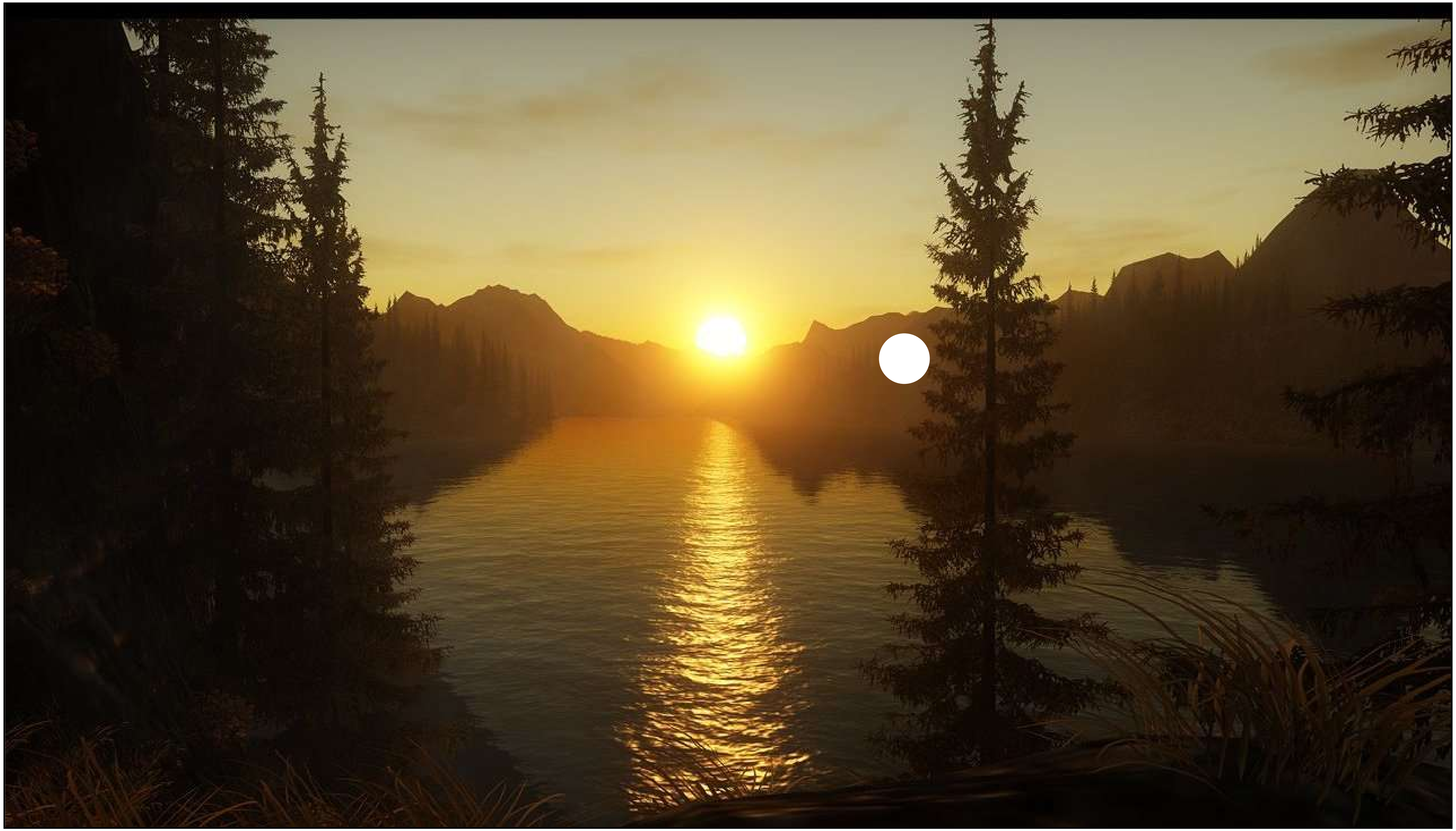


- Contrast compression achieved by global contrast reduction
  - Enhance reflectance, then compress everything

Techniques

- Arithmetic of HDR images
- Display model
- Tone-curve
- Color transfer
- Base-detail separation
- Glare

Glare



“Alan Wake” © Remedy Entertainment

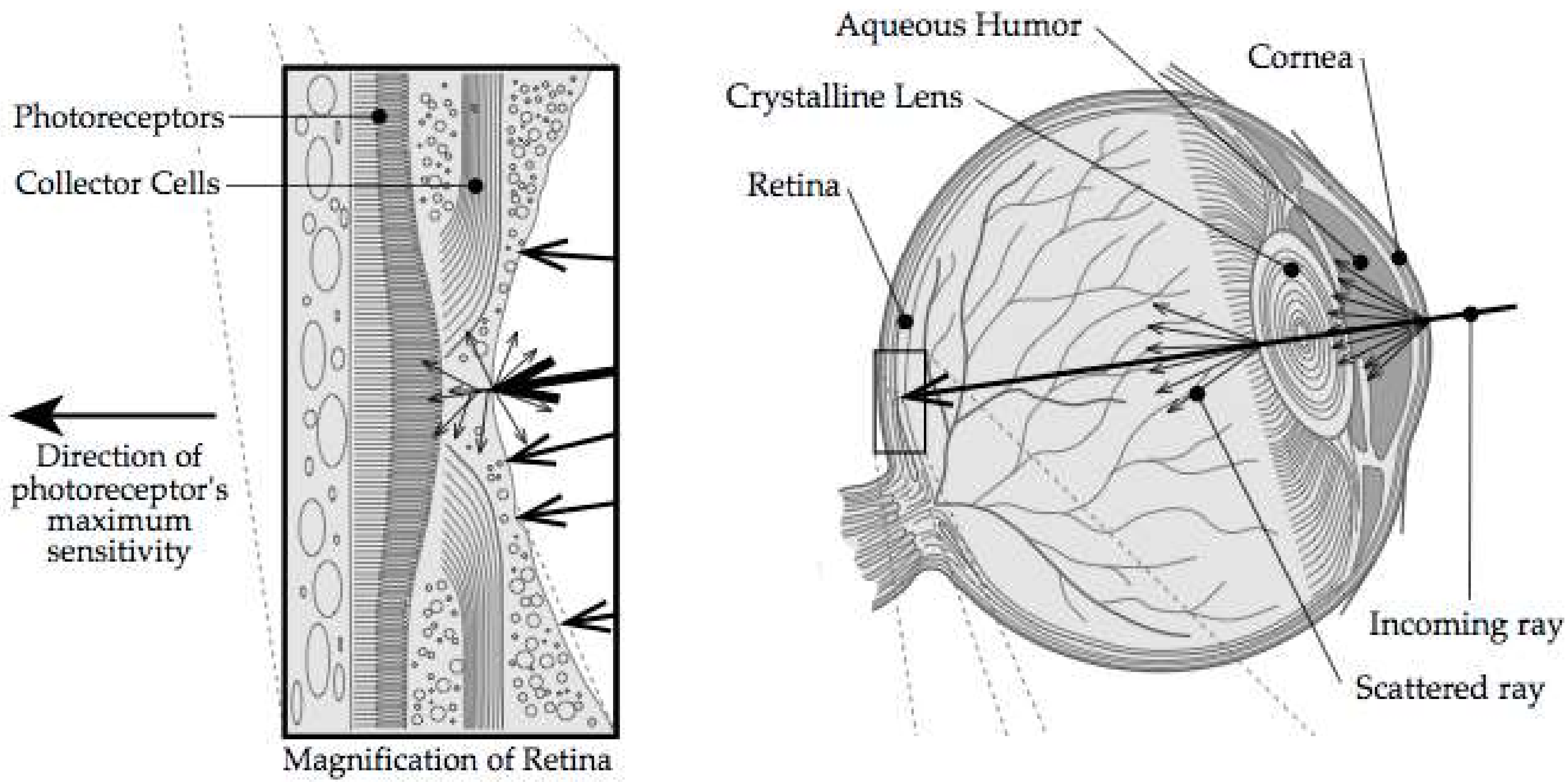
Glare Illusion

Photography

Painting

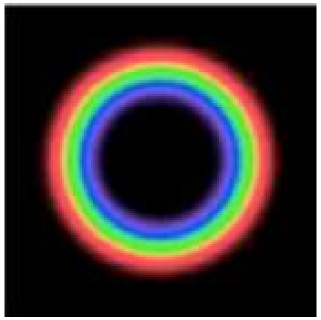
Computer Graphics
HDR rendering in games

Scattering of the light in the eye

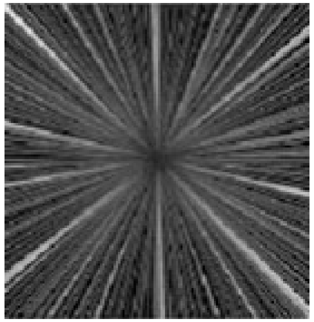


From: Sekuler, R., and Blake, R. Perception, second ed. McGraw- Hill, New York, 1990

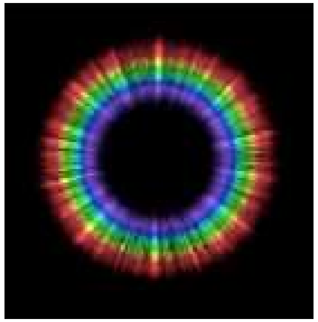
Ciliary corona and lenticular halo



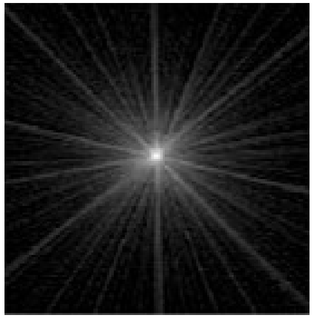
*



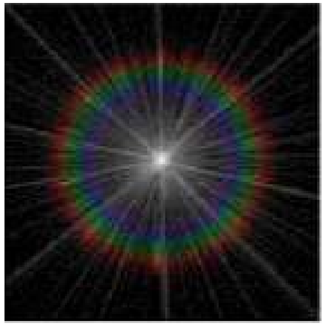
=

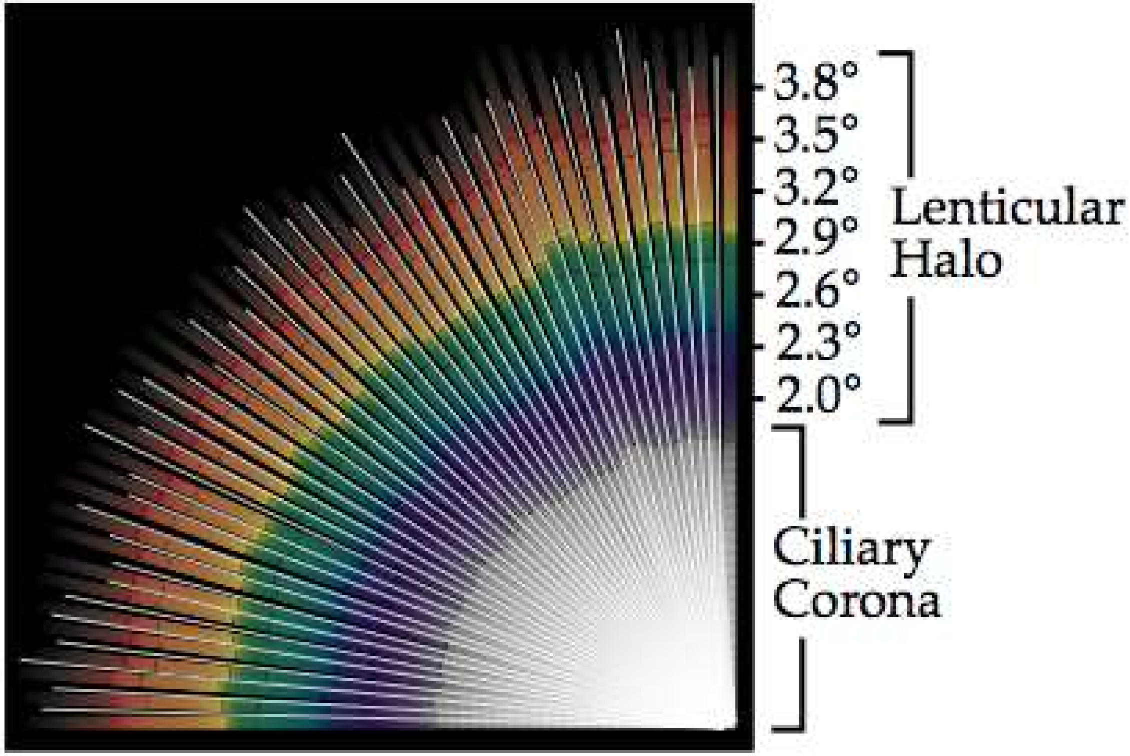


+



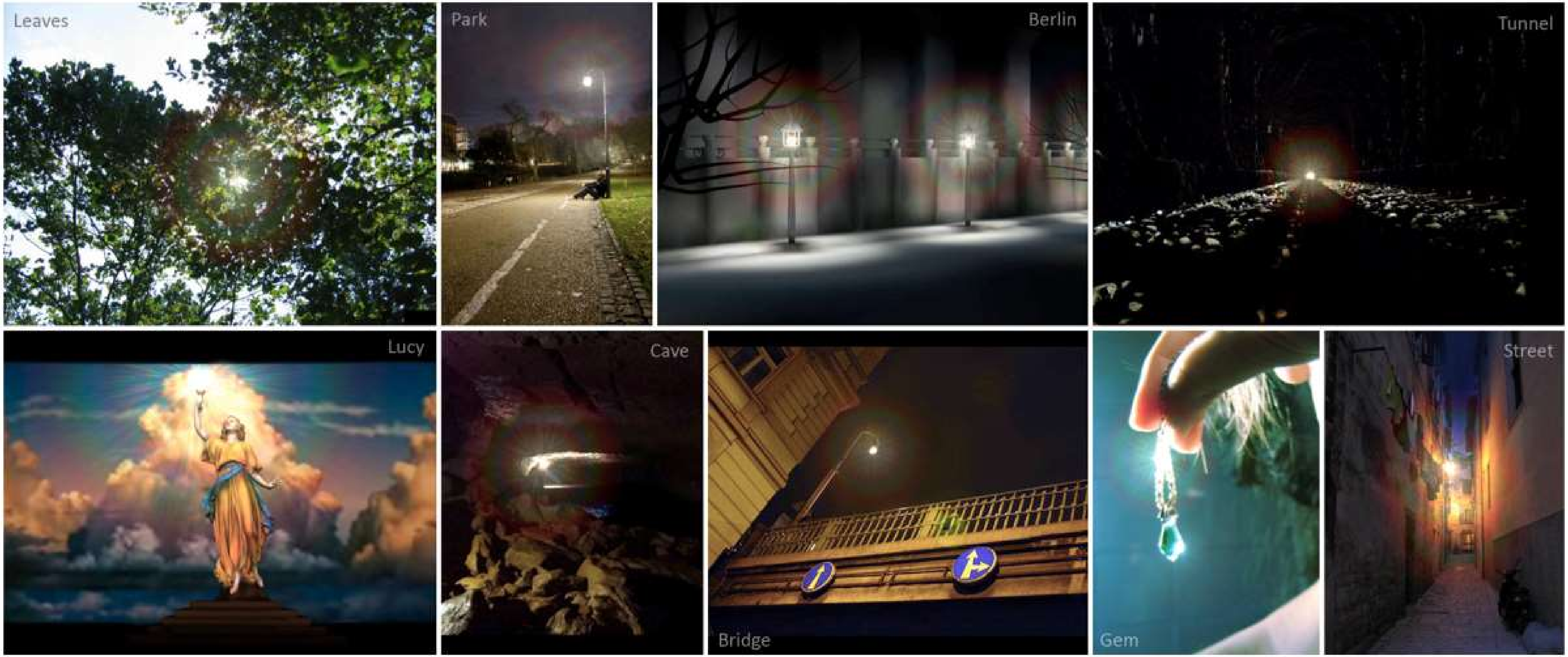
=





From: Spencer, G. et al. 1995. Proc. of SIGGRAPH. (1995)

Examples of simulated glare

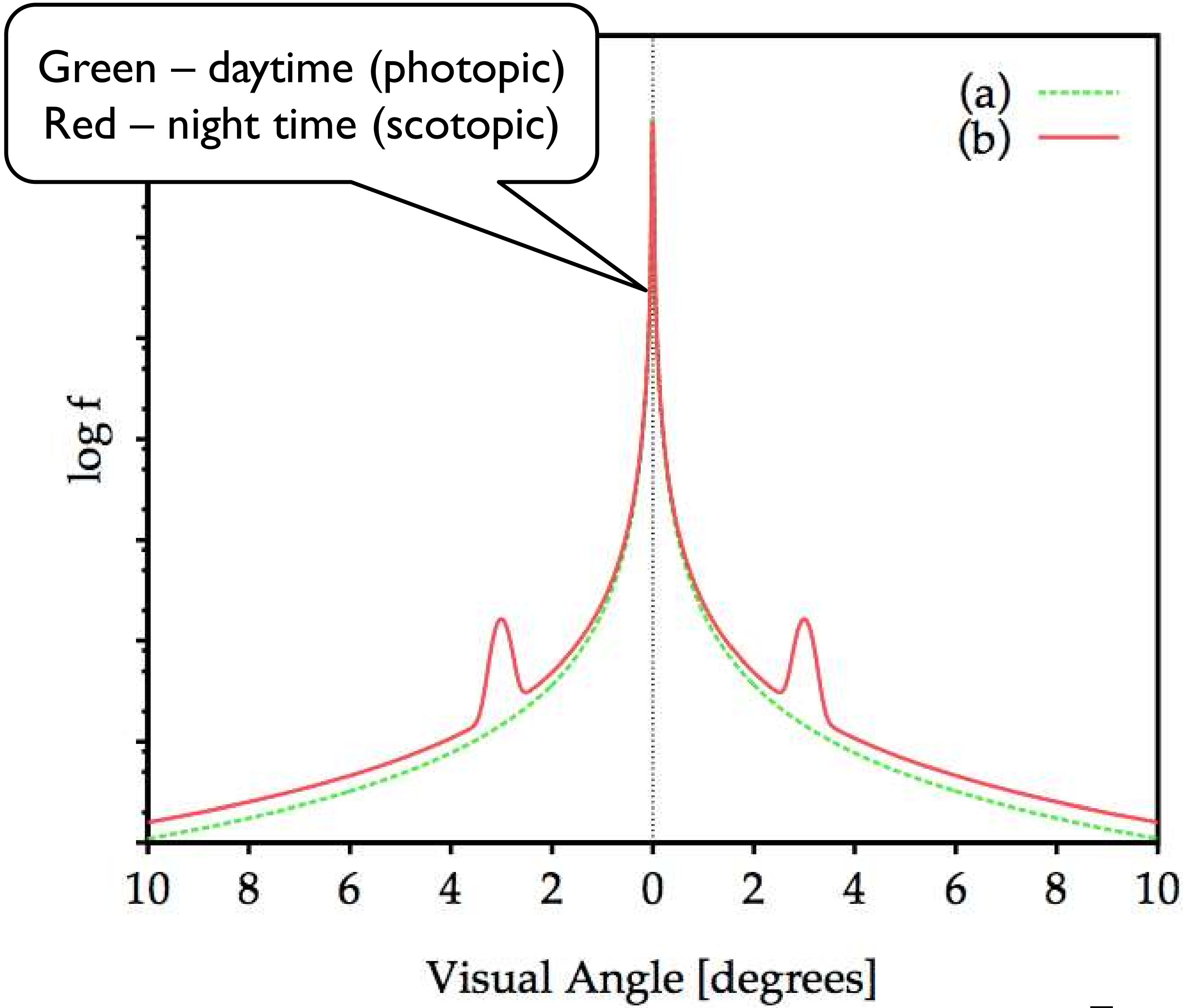


[From Ritschel et al, Eurographics 2009]

Temporal glare

[From Ritschel et al, Eurographics 2009]

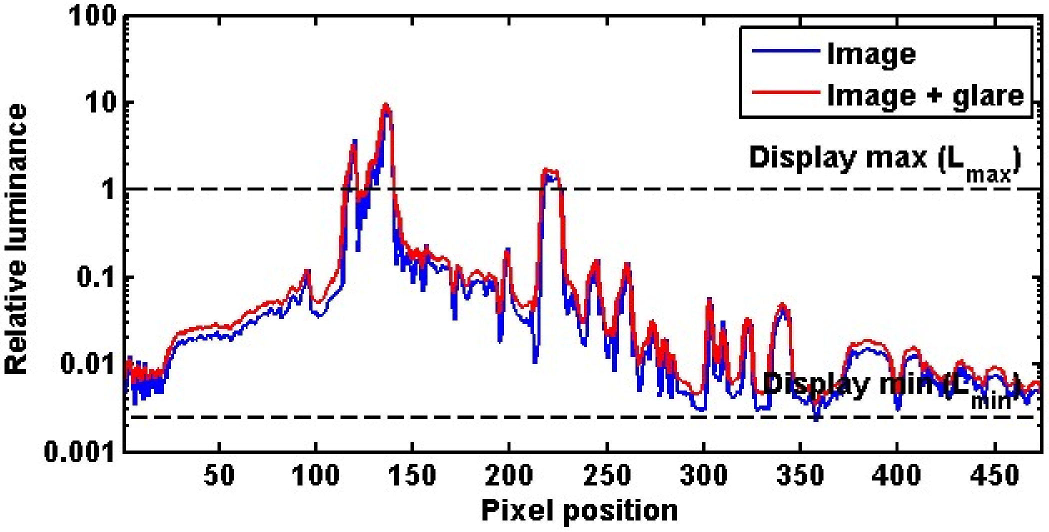
Point Spread Function of the eye



- What portion of the light is scattered towards a certain visual angle
- To simulate:
  - construct a digital filter
  - convolve the image with that filter

From: Spencer, G. et al. 1995. Proc. of SIGGRAPH. (1995)

Selective application of glare

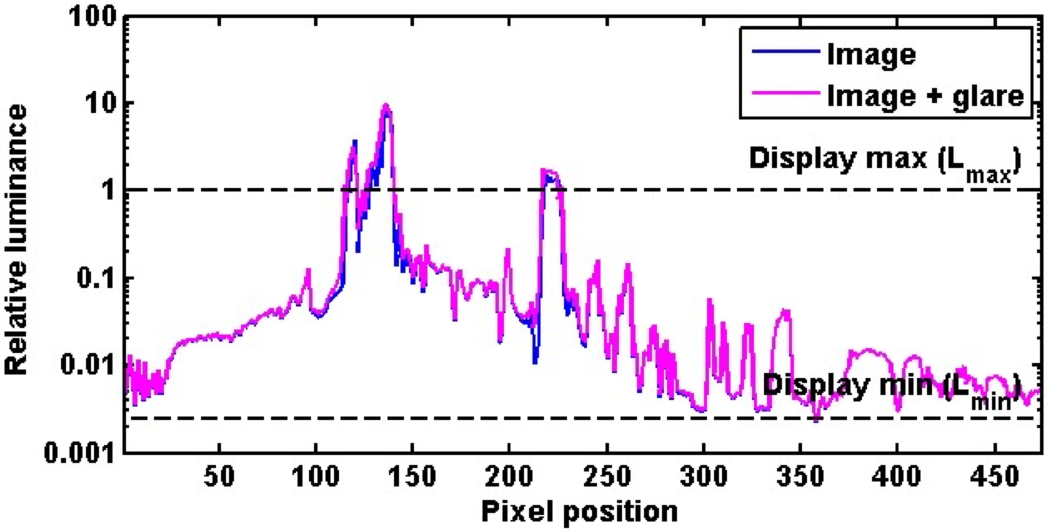


- A) Glare applied to the entire image

$$I_g = I * G$$

Glare kernel (PSF)

- Reduces image contrast and sharpness



B) Glare applied only to the clipped pixels

$$I_g = I + I_{cliped} * G - I_{cliped}$$

where $I_{cliped} = \begin{cases} I & for\ I > 1 \\ 0 & otherwise \end{cases}$

Better image quality

Selective application of glare

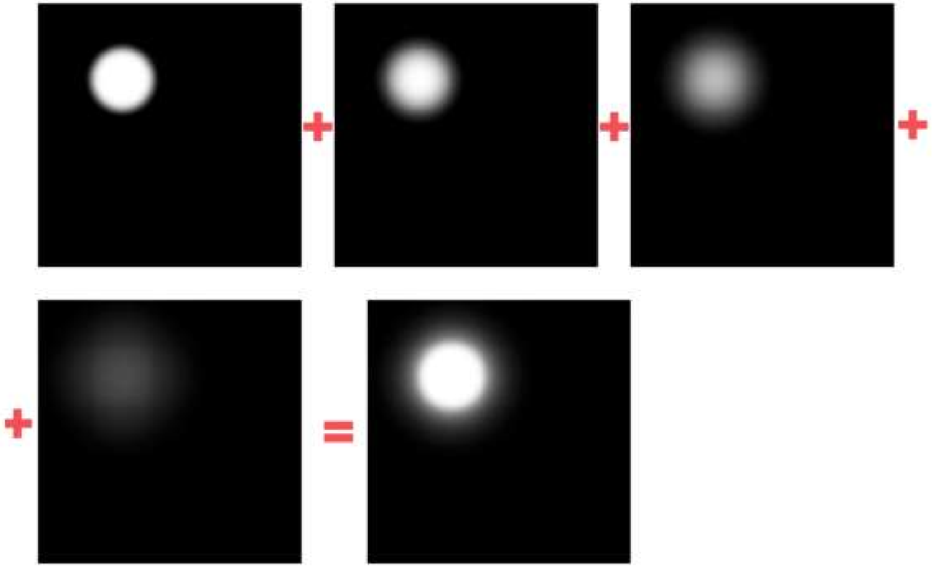
A) Glare applied to
the entire image

Original image

B) Glare applied to
clipped pixels only

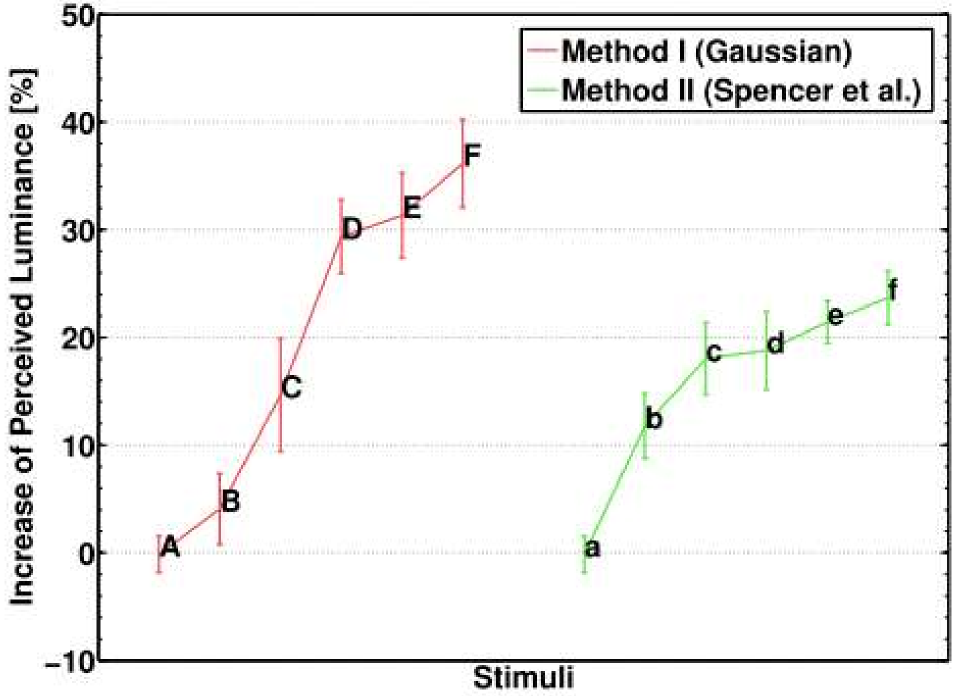
Glare (or bloom) in games

- Convolution with large, non-separable filters is too slow
- The effect is approximated by a combination of Gaussian filters
  - Each filter with different "sigma"
- The effect is meant to look good, not be be accurate model of light scattering
- Some games simulate camera rather than the eye



Does the exact shape of the PSF matter?

- The illusion of increased brightness works even if the PSF is very different from the PSF of the eye



red - Gaussian

green - accurate

[Yoshida et al., APGV 2008]

HDR rendering – motion blur

From LDR pixels From HDR pixels

References

- Comprehensive book on HDR Imaging
  - E. Reinhard, W. Heidrich, P. Debevec, S. Pattanaik, G. Ward, and K. Myszkowski, High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting, 2nd editio. Morgan Kaufmann, 2010.
- Overview of HDR imaging & tone-mapping
  - http://www.cl.cam.ac.uk/~rkm38/hdri_book.html
- Review of recent video tone-mapping
  - A comparative review of tone-mapping algorithms for high dynamic range video *Gabriel Eilertsen, Rafal K. Mantiuk, Jonas Unger*, Eurographics State-of-The-Art Report 2017.
- Selected papers on tone-mapping:
  - G. W. Larson, H. Rushmeier, and C. Piatko, "A visibility matching tone reproduction operator for high dynamic range scenes," *IEEE Trans. Vis. Comput. Graph.*, vol. 3, no. 4, pp. 291–306, 1997.
  - R. Wanat and R. K. Mantiuk, "Simulating and compensating changes in appearance between day and night vision," *ACM Trans. Graph. (Proc. SIGGRAPH)*, vol. 33, no. 4, p. 147, 2014.
  - Spencer, G. et al. 1995. Physically-Based Glare Effects for Digital Images. Proceedings of SIGGRAPH. (1995), 325–334
  - Ritschel, T. et al. 2009. Temporal Glare: Real-Time Dynamic Simulation of the Scattering in the Human Eye. Computer Graphics Forum. 28, 2 (Apr. 2009), 183–192
  - ...

UNIVERSITY OF CAMBRIDGE
COMPUTER LABORATORY
RAINBOW GRAPHICS GROUP

Advanced Graphics & Image Processing

Virtual and Augmented Reality

Part 1/2 – virtual reality

Rafał Mantiuk
Dept. of Computer Science and Technology, University of Cambridge

The slides used in this lecture are the courtesy of Gordon Wetzstein.
From Virtual Reality course: http://stanford.edu/class/ee267/

vir·tu·al re·al·i·ty

vərCH(əw)əl rēˈalədē

the computer-generated simulation of a three-dimensional image or environment that can be interacted with in a seemingly real or physical way by a person using special electronic equipment, such as a helmet with a screen inside or gloves fitted with sensors.

vpl research

simulation & training

visualization & entertainment

remote control of vehicles, e.g. drones

gaming

robotic surgery

architecture walkthroughs

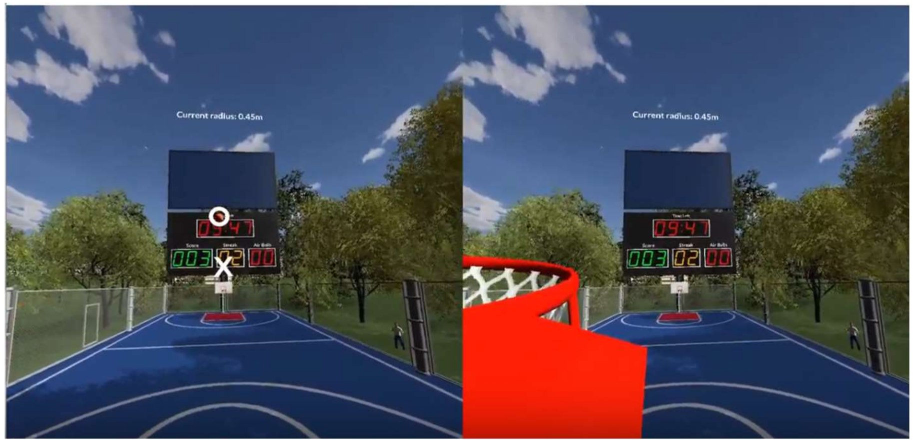
education

virtual travel

a trip down the rabbit hole

Vision treatment in VR

- **Treatment of amblyopia**
  - Training the brain to use the “lazy” eye



Images courtesy of VIVID VISION

Exciting Engineering Aspects of VR/AR

- cloud computing
- shared experiences

- CPU, GPU
- IPU, DPU?

- compression, streaming

- sensors & imaging
- computer vision
- scene understanding

images by microsoft, facebook

- VR cameras

- photonics / waveguides
- human perception
- displays: visual, auditory, vestibular, haptic, …

- HCI
- applications

Where We Want It To Be

image by ray ban

Personal Computer
e.g. Commodore PET 1983

Laptop
e.g. Apple MacBook

Smartphone
e.g. Google Pixel

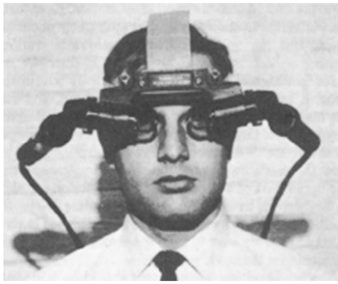
AR/VR
e.g. Microsoft Hololens

???

A Brief History of Virtual Reality

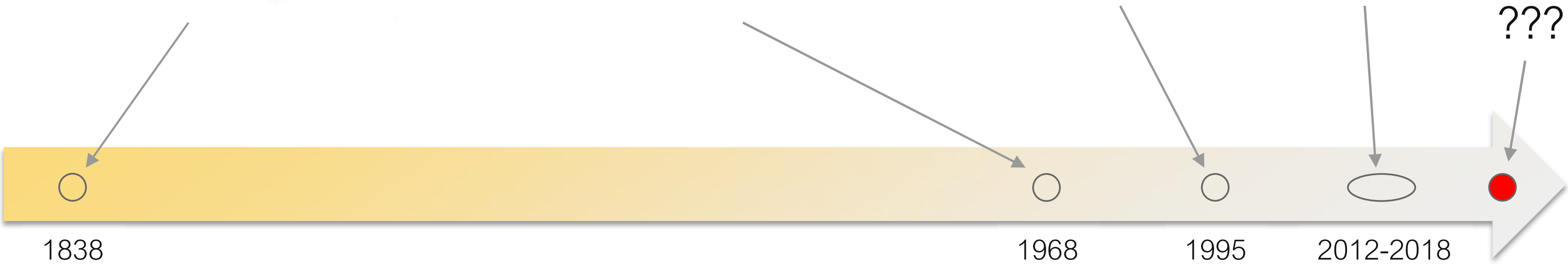
Stereoscopes
Wheatstone, Brewster, …

VR & AR
Ivan Sutherland



Nintendo
Virtual Boy

VR explosion
Oculus, Sony, HTC, MS, …



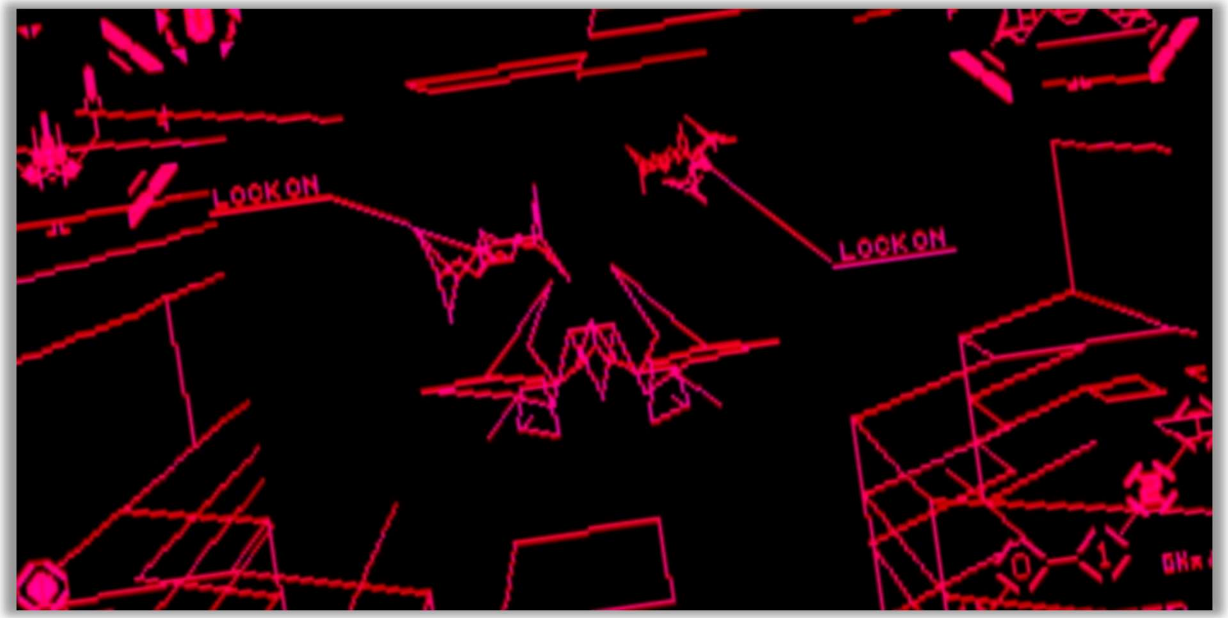
Ivan Sutherland’s HMD

- optical see-through AR, including:
  - displays (2x 1” CRTs)
  - rendering
  - head tracking
  - interaction
  - model generation
- computer graphics
- human-computer interaction

I. Sutherland “A head-mounted three-dimensional display”, Fall Joint Computer Conference 1968

Nintendo Virtual Boy

- computer graphics & GPUs were not ready yet!

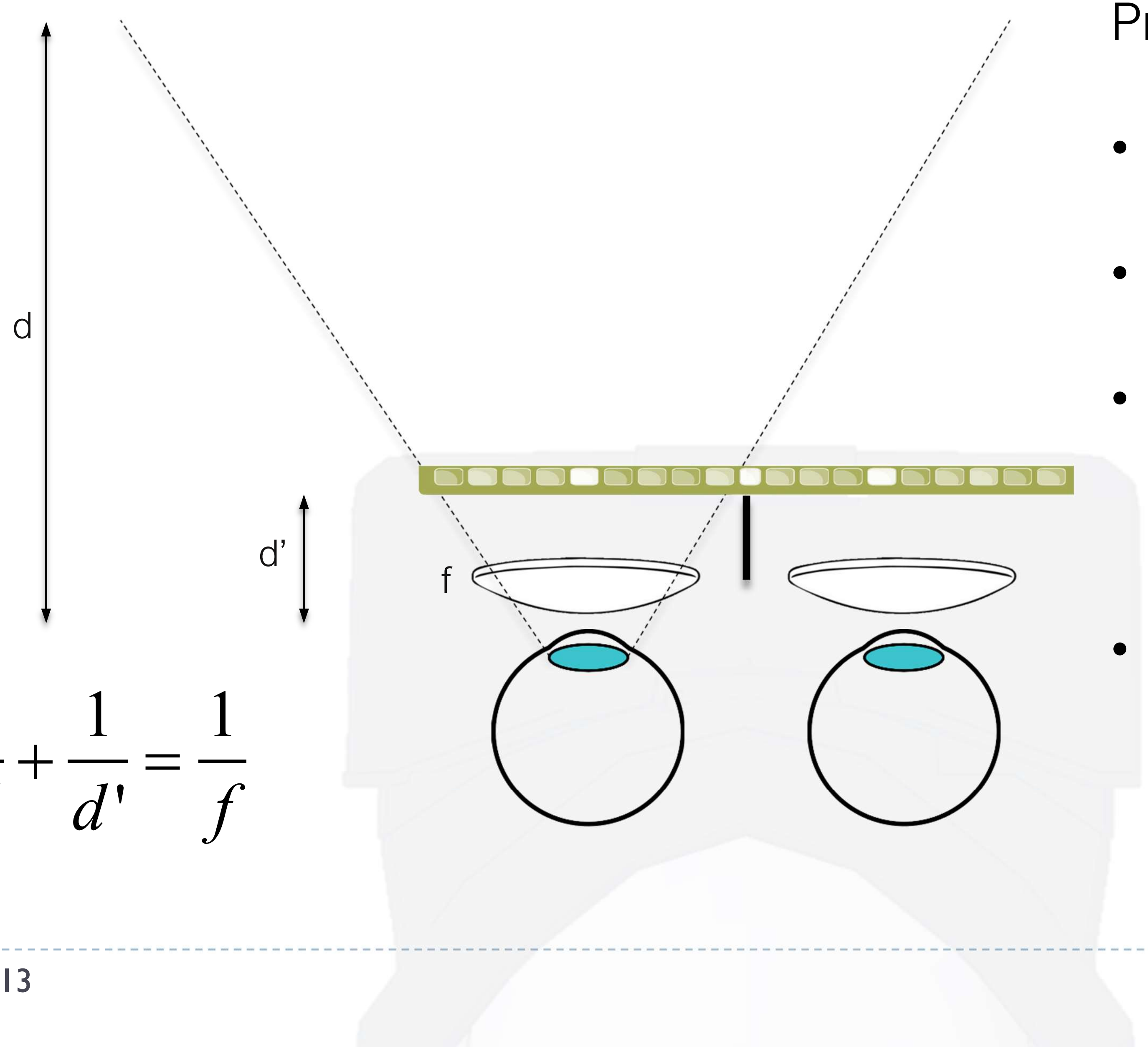


Game: Red Alarm

Where we are now

IFIXIT teardown

Virtual Image

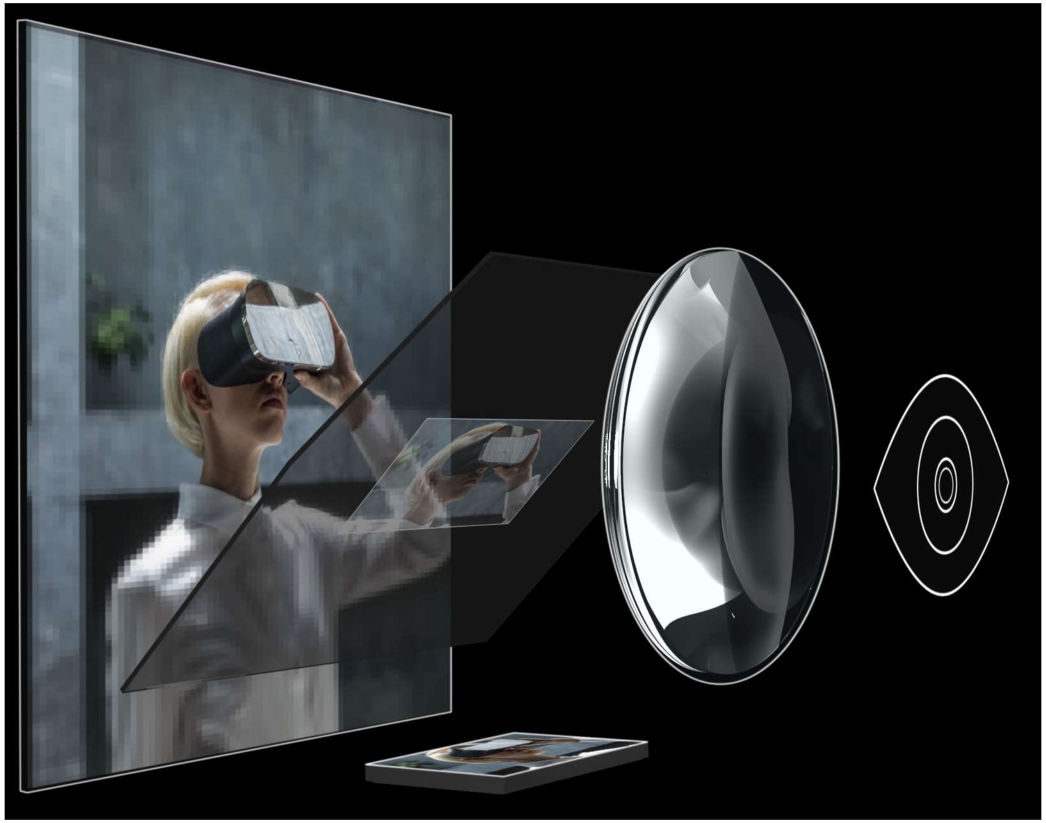


$$\frac{1}{d} + \frac{1}{d'} = \frac{1}{f}$$

Problems:

- fixed focal plane
- no focus cues ☹
- cannot drive accommodation with rendering!
- limited resolution

A dual-resolution display



- High resolution image in the centre, low resolution fills wide field-of-view
- Two displays combined using a beam-splitter
- Image from: https://varjo.com/bionic-display/

UNIVERSITY OF CAMBRIDGE
COMPUTER LABORATORY

RAINBOW GRAPHICS GROUP

Advanced Graphics & Image Processing

Virtual and Augmented Reality

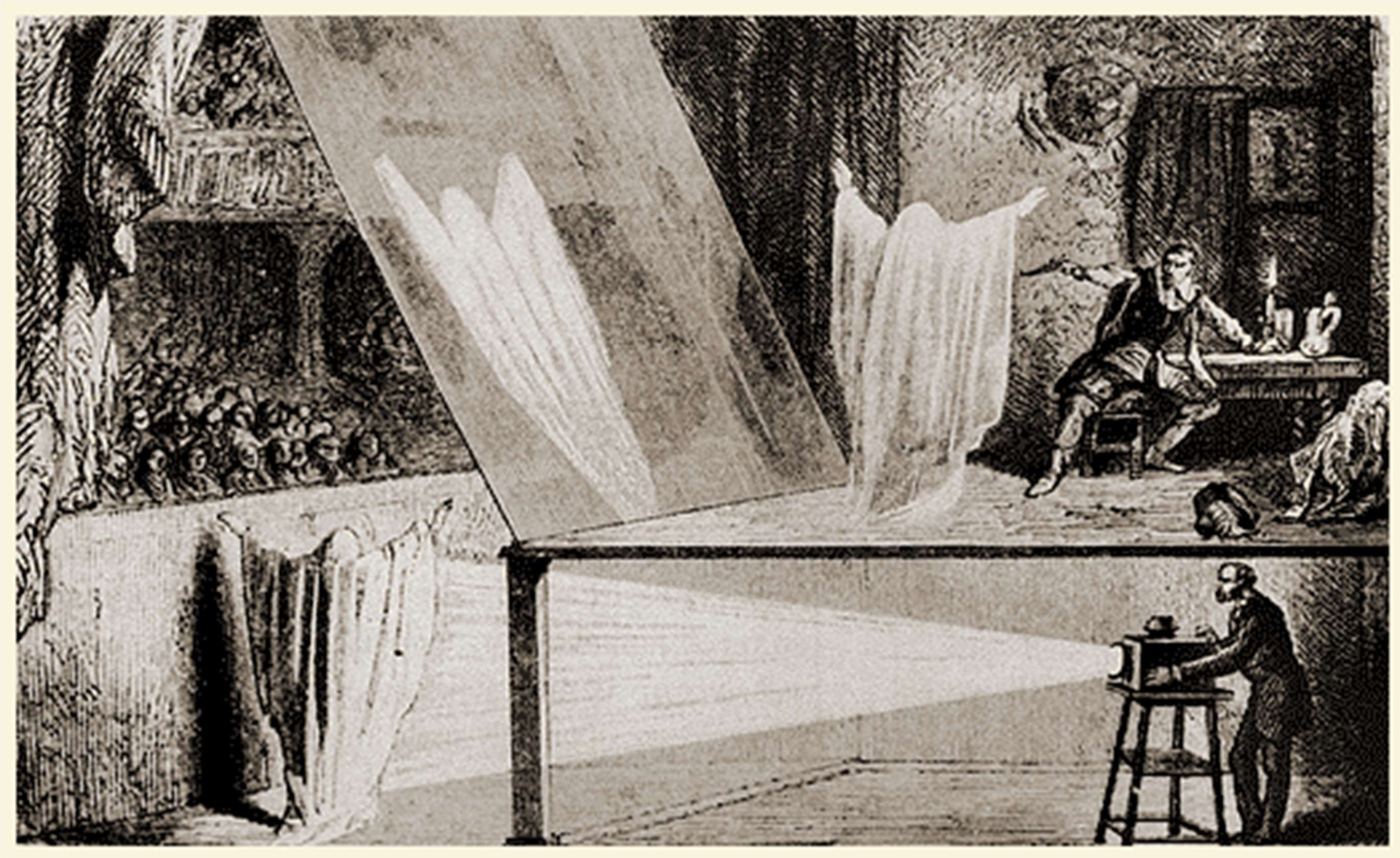
Part 1/2 – augmented reality

Rafał Mantiuk

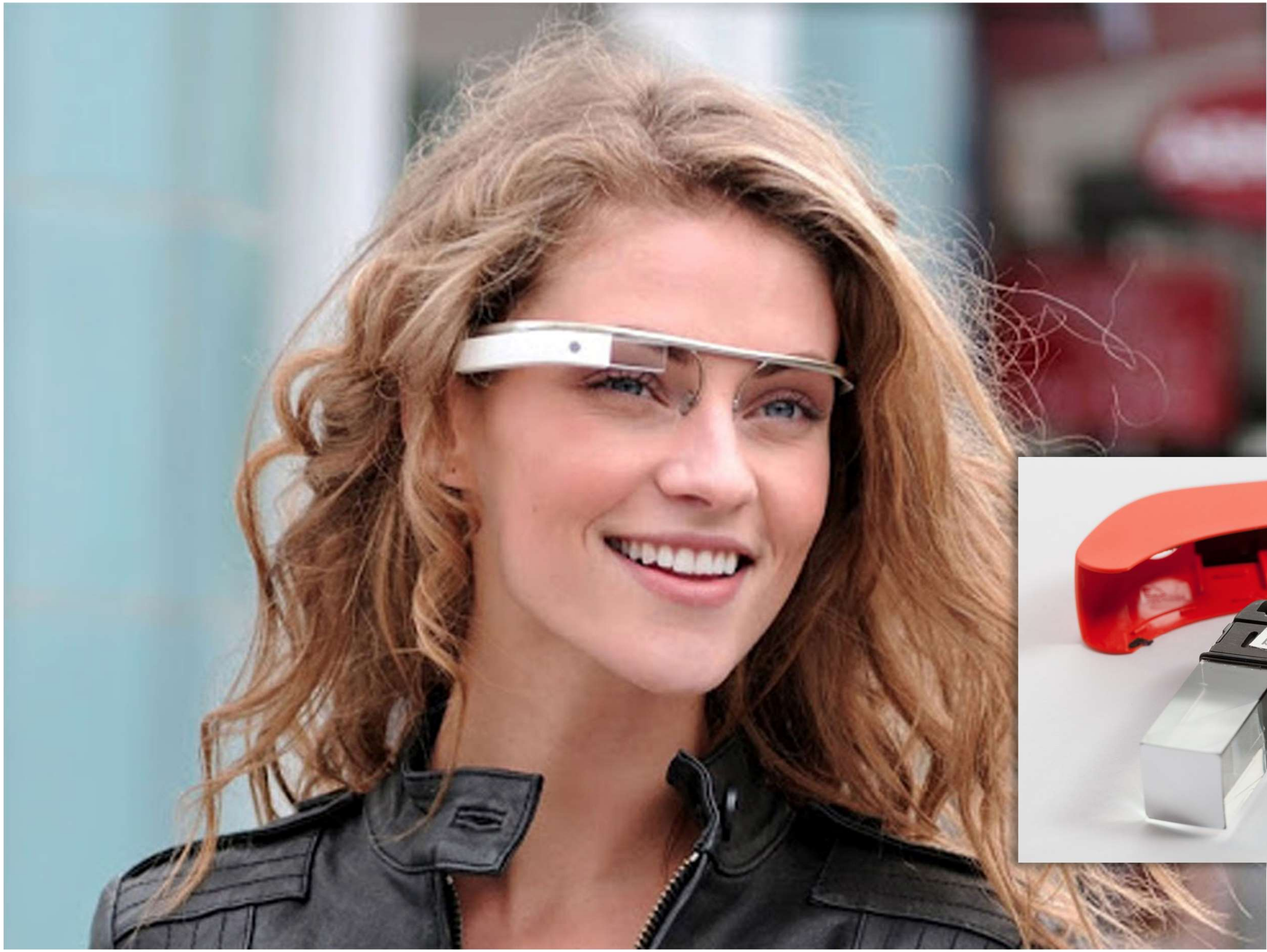
Dept. of Computer Science and Technology, University of Cambridge

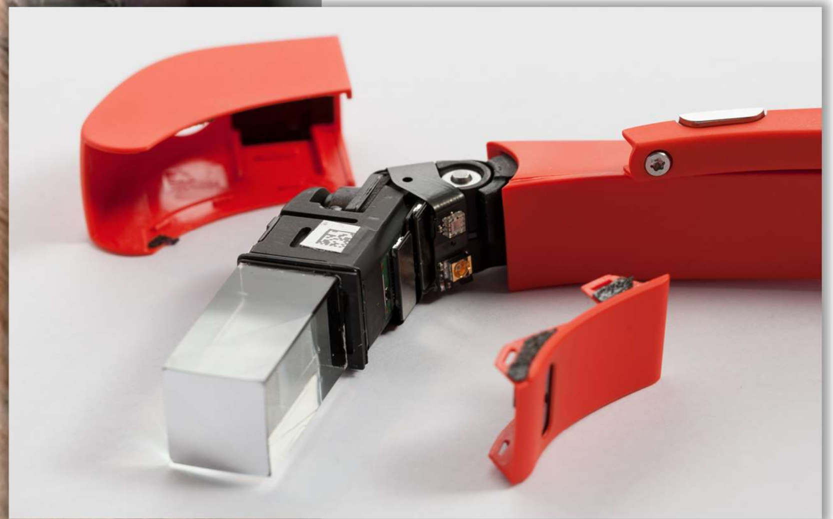
The slides used in this lecture are the courtesy of Gordon Wetzstein.
From Virtual Reality course: http://stanford.edu/class/ee267/

Pepper's Ghost 1862

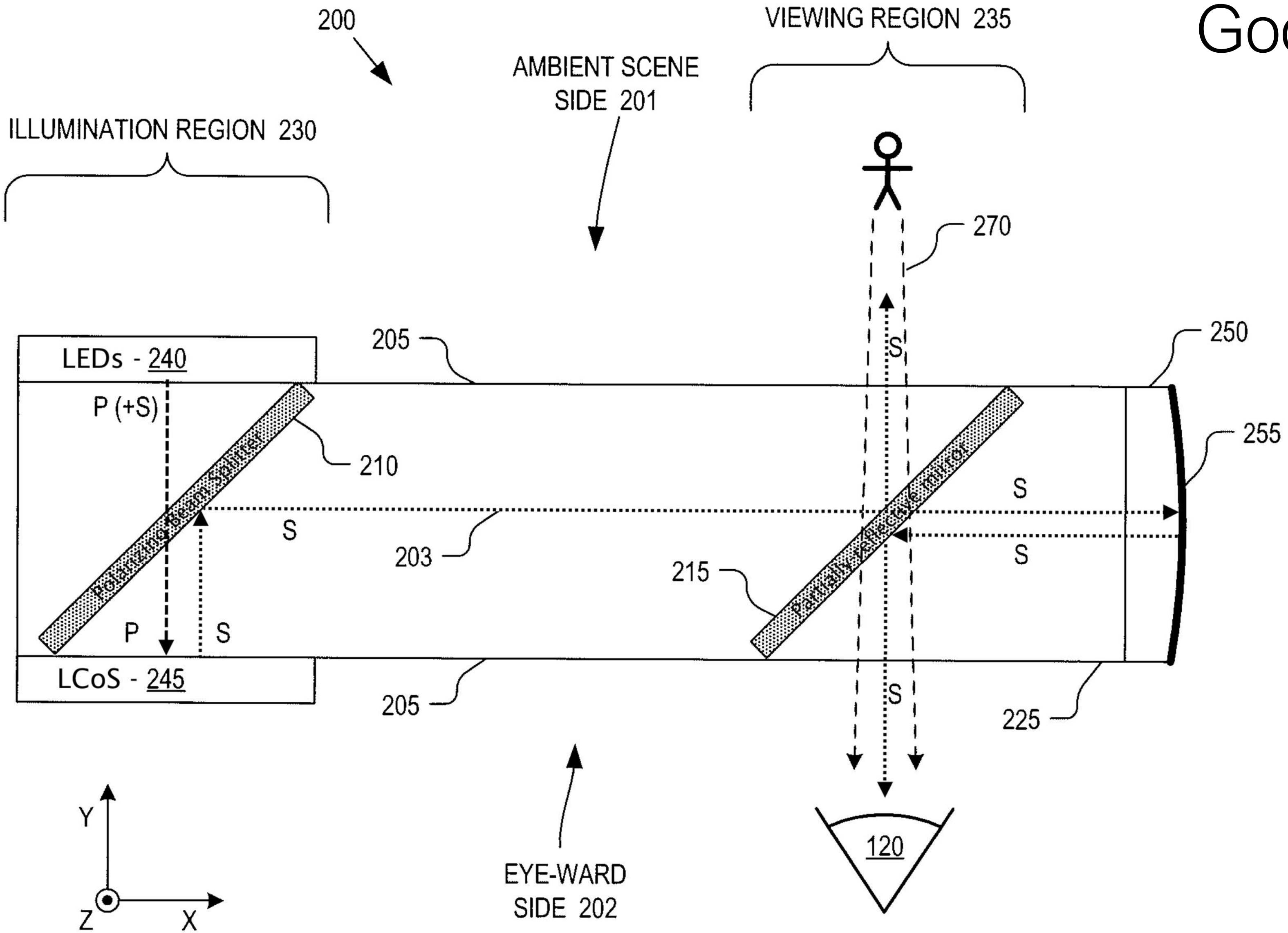


Google Glass





Google Glass



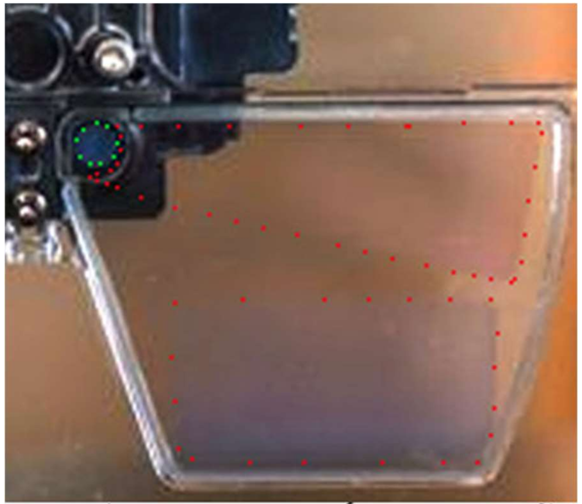
Meta 2

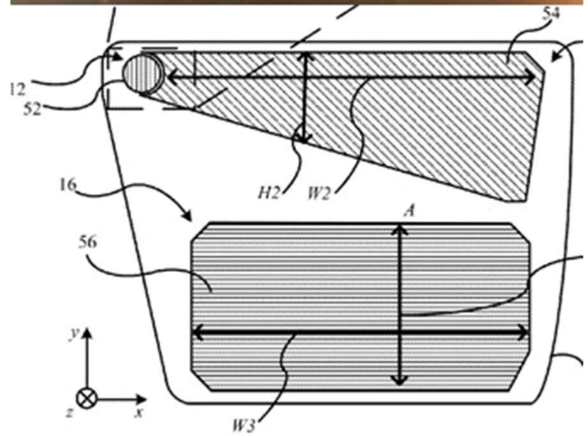
- Larger field of view (90 deg) than Glass
- Also larger device form factor

Microsoft HoloLens

Microsoft HoloLens

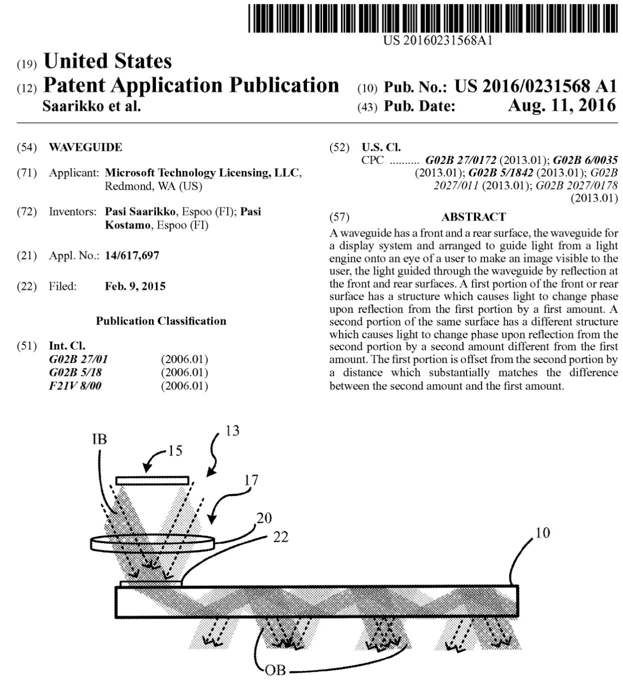
- diffraction grating
- small FOV (30x17), but good image quality





US 2016/0231568

Fig. 3B



US 20160231568A1

(19) **United States**
(12) **Patent Application Publication** (10) **Pub. No.: US 2016/0231568 A1**
Saarikko et al. (43) **Pub. Date: Aug. 11, 2016**

(54) **WAVEGUIDE**

(71) Applicant: **Microsoft Technology Licensing, LLC**, Redmond, WA (US)

(72) Inventors: **Pasi Saarikko**, Espoo (FI); **Pasi Kostamo**, Espoo (FI)

(21) Appl. No.: **14/617,697**

(22) Filed: **Feb. 9, 2015**

Publication Classification

(51) **Int. Cl.**
G02B 27/01 (2006.01)
G02B 5/18 (2006.01)
F21V 8/00 (2006.01)

(52) **U.S. Cl.**
CPC *G02B 27/0172* (2013.01); *G02B 6/0035* (2013.01); *G02B 5/1842* (2013.01); *G02B 2027/011* (2013.01); *G02B 2027/0178* (2013.01)

(57) **ABSTRACT**

A waveguide has a front and a rear surface, the waveguide for a display system and arranged to guide light from a light engine onto an eye of a user to make an image visible to the user, the light guided through the waveguide by reflection at the front and rear surfaces. A first portion of the front or rear surface has a structure which causes light to change phase upon reflection from the first portion by a first amount. A second portion of the same surface has a different structure which causes light to change phase upon reflection from the second portion by a second amount different from the first amount. The first portion is offset from the second portion by a distance which substantially matches the difference between the second amount and the first amount.

Microsoft HoloLens 2

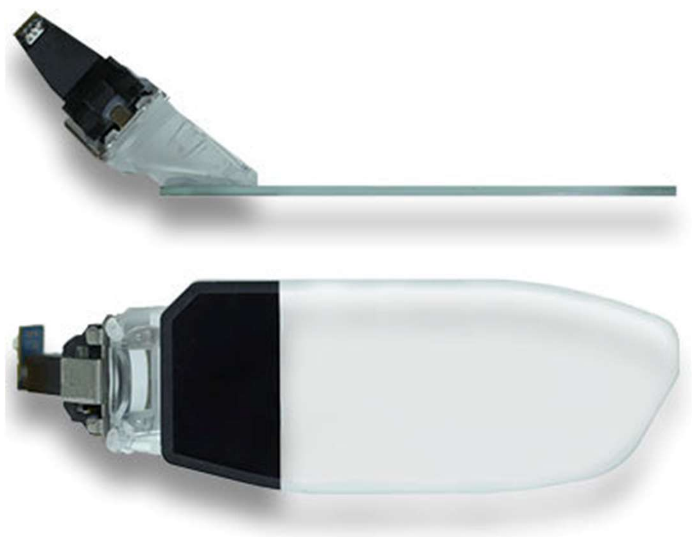
- Wider field of view (52 deg)
- High resolution (47 pix per deg)
- Improved ergonomics
- Better hand tracking

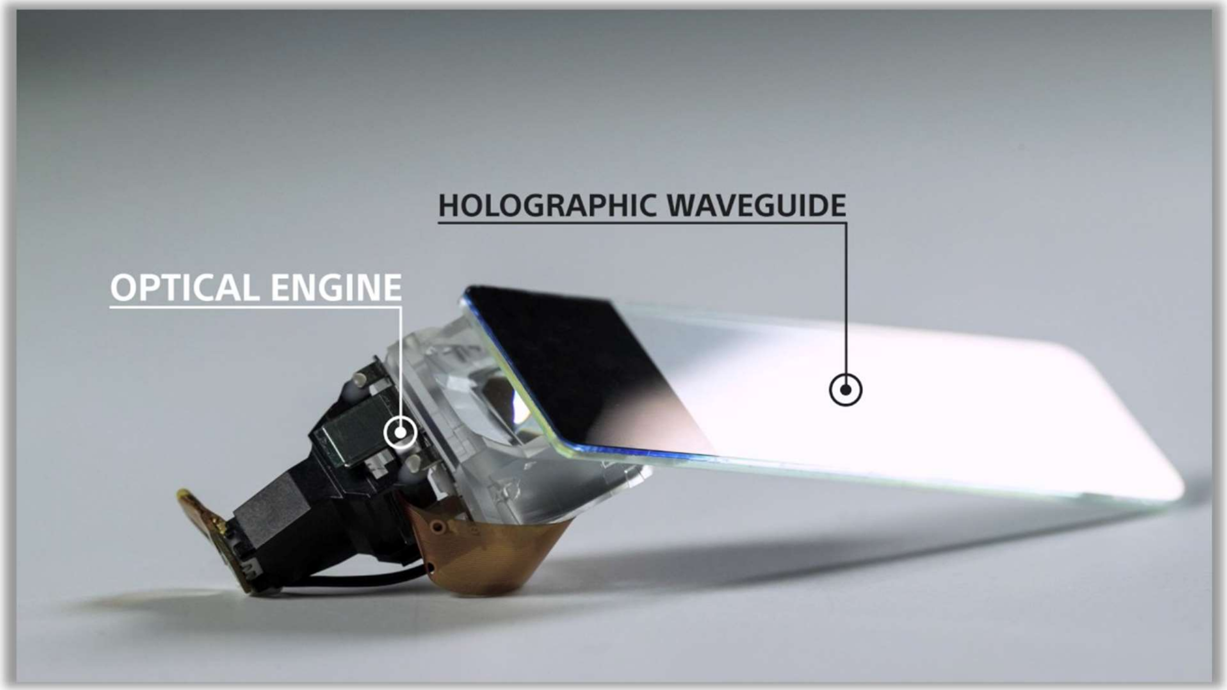
Zeiss Smart Optics

- great device form factor
- polycarbonate light guide – easy to manufacture and robust
- smaller field of view (17 deg)

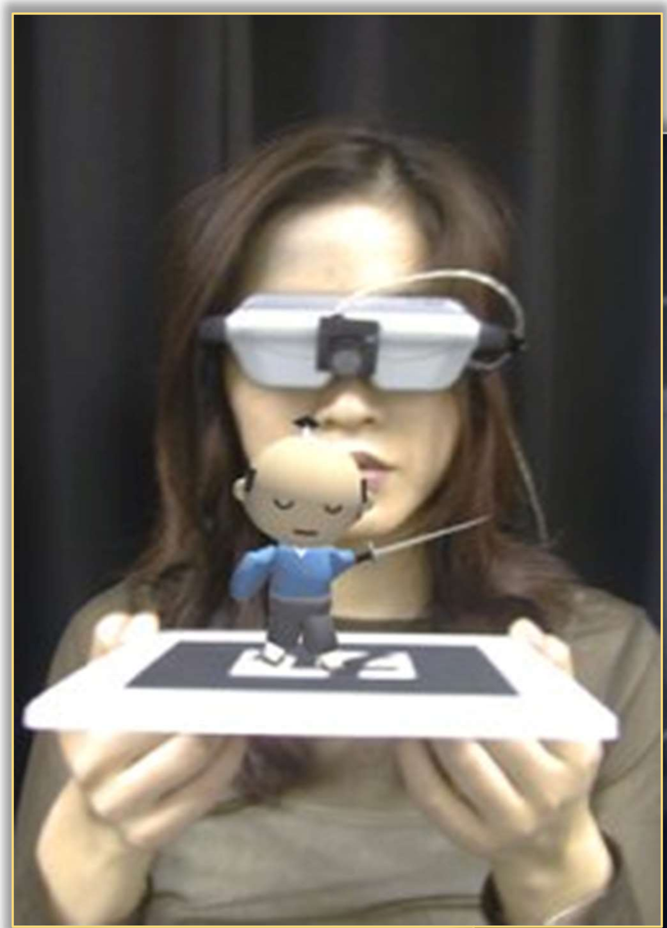
Sony IMX-001

- also great form factor
- small FOV (9x6 deg)
- monochrome





Video AR: ARCore, ARKit, ARToolKit, …



VR/AR challenges

- Latency (next lecture)
- Tracking
- 3D Image quality and resolution
- Reproduction of depth cues (last lecture)
- Rendering & bandwidth
- Simulation/cyber sickness
- Content creation
  - Game engines
  - Image-Based-Rendering

Simulation sickness

- Conflict between vestibular and visual systems
  - When camera motion inconsistent with head motion
  - Frame of reference (e.g. cockpit) helps
  - Worse with larger FOV
  - Worse with high luminance and flicker

References

- LaValle "Virtual Reality", Cambridge University Press, 2016
  - http://vr.cs.uiuc.edu/
- Virtual Reality course from the Stanford Computational Imaging group
  - http://stanford.edu/class/ee267/

Display Technologies

Advanced Graphics and Image Processing

Rafał Mantiuk
Computer Laboratory, University of Cambridge

Overview

- Temporal aspects
  - Latency in VR
  - Eye-movement
  - Hold-type blur
- 2D displays
  - 2D spatial light modulators
  - High dynamic range displays

Latency in VR

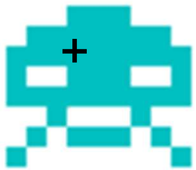
- Sources of latency in VR
  - IMU ~1 ms
  - sensor fusion, data transfer
  - rendering: depends on complexity of scene & GPU – a few ms
  - data transfer again
  - Display
    - 60 Hz = 16.6 ms;
    - 70 Hz = 11.1 ms;
    - 120 Hz = 8.3 ms.
- Target latency
  - Maximum acceptable: 20ms
  - Much smaller (5ms) desired for interactive applications
- Example
  - 16 ms (display) + 16 ms (rendering) + 4 ms (orientation tracking) = 36 ms latency total
  - At 60 deg/s head motion, 1Kx1K, 100deg fov display:
    - 19 pixels error
    - Too much

Post-rendering image warp (time warp)

- To minimize end-to-end latency
- The method:
  - get current camera pose
  - render into a larger raster than the screen buffer
  - get new camera pose
  - warp rendered image using the latest pose, send to the display
    - 2D image translation
    - 2D image warp
    - 3D image warp
- Original paper from Mark et al. 1997, also Darsa et al. 1997

Eye movement - basics

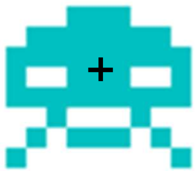
Fixation



Drift: 0.15-0.8 deg/s

Eye movement - basics

Saccade



160-300 deg/s

Eye movement - basics

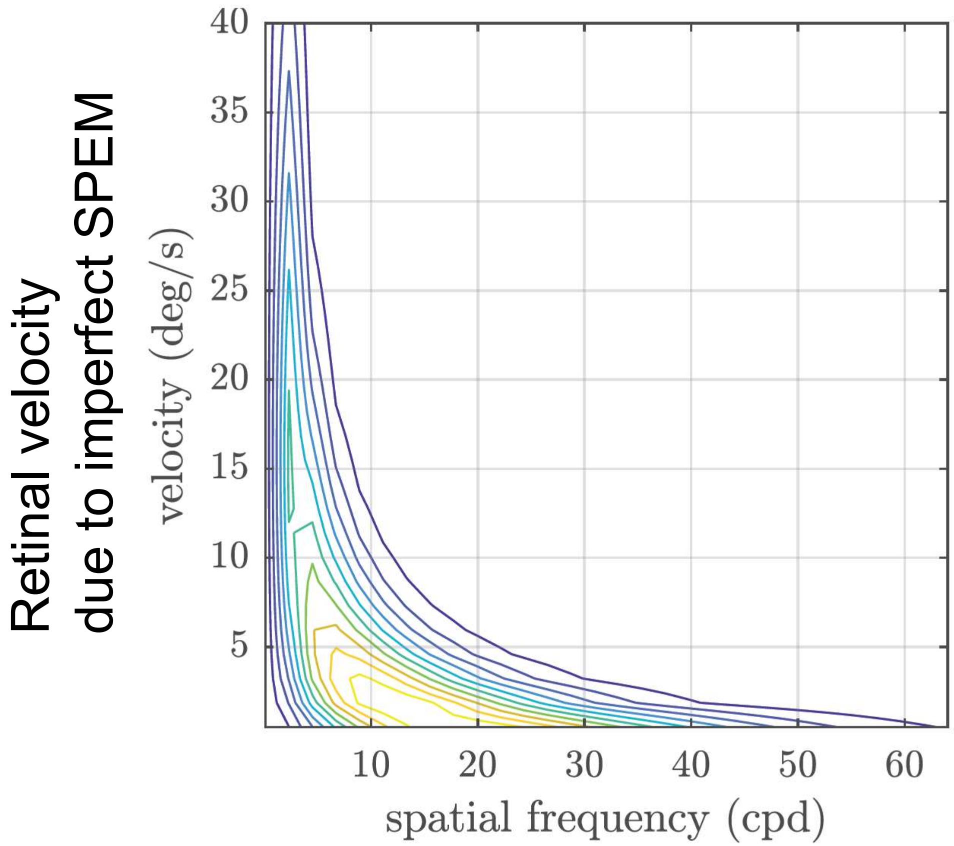
Smooth Pursuit Eye Motion (SPEM)

Up to 80 deg/s
The gaze tends to be 5-20% slower than the object

Retinal velocity

- The eye tracks moving objects
  - Smooth Pursuit Eye Motion (SPEM) stabilizes images on the retina
  - But SPEM is imperfect
- Loss of sensitivity mostly caused by imperfect SPEM
  - SPEM worse at high velocities

Spatio-velocity contrast sensitivity



Kelly's model [1979]

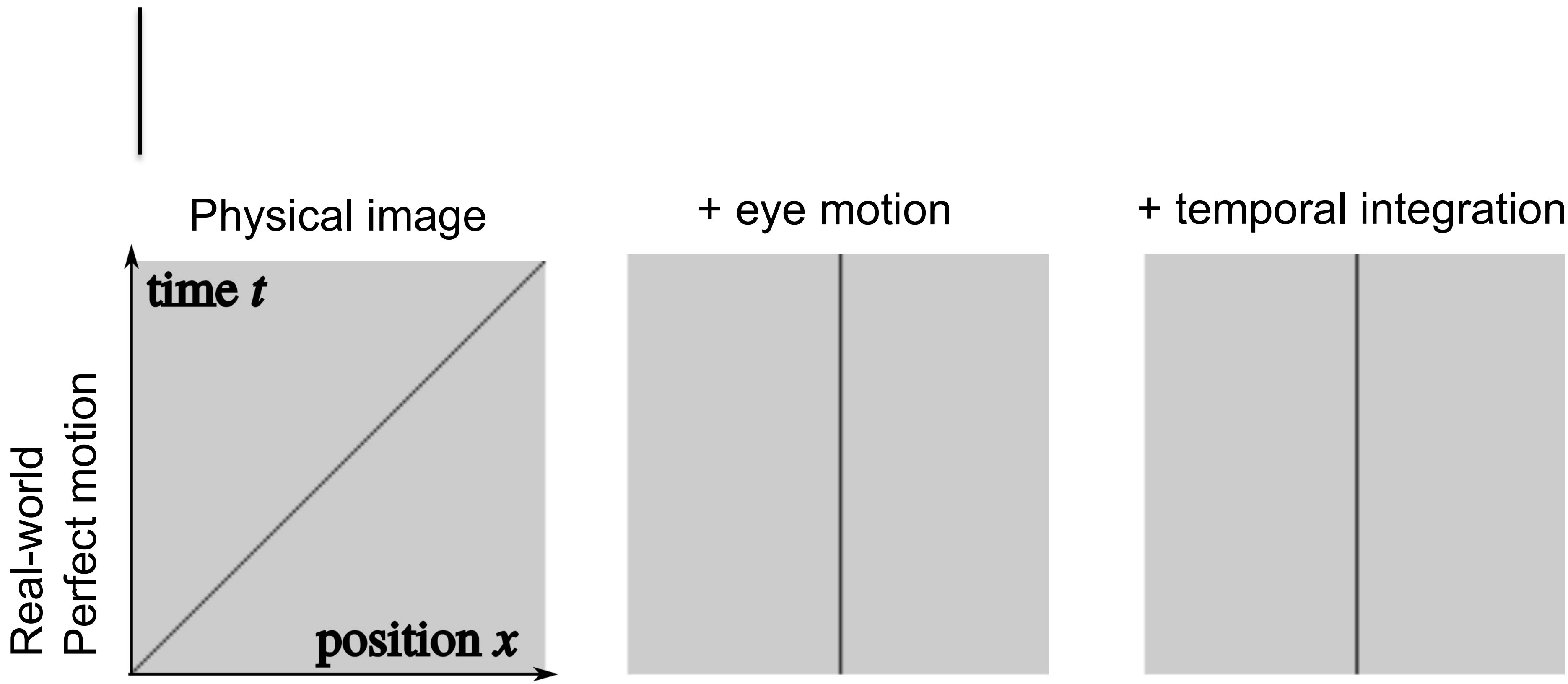
Motion sharpening

- The visual system "sharpens" objects moving at speeds of 6 deg/s or more

  - Potentially a reason why VR appears sharper than it actually is

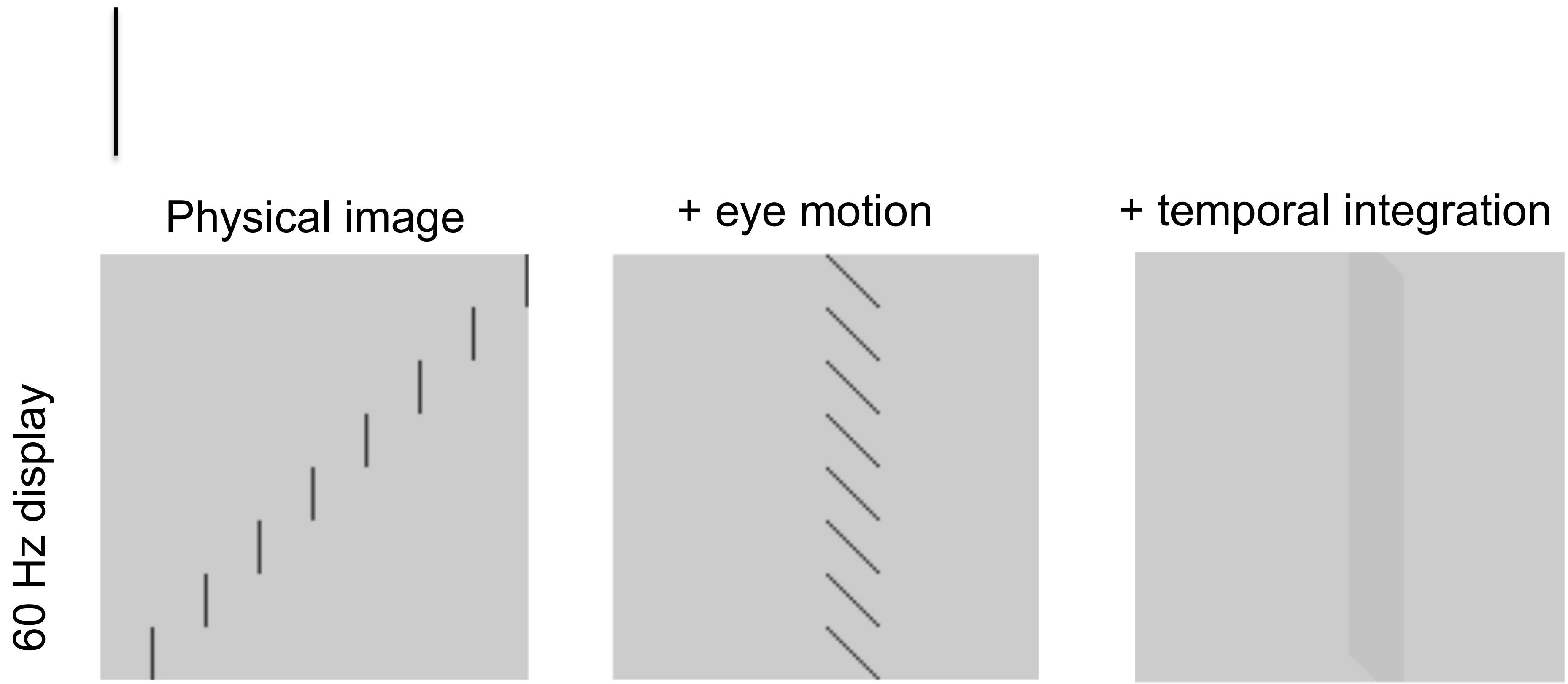
Hold-type blur

- The eye smoothly follows a moving object
- But the image on the display is "frozen" for 1/60th of a second



Hold-type blur

- The eye smoothly follows a moving object
- But the image on the display is "frozen" for 1/60th of a second

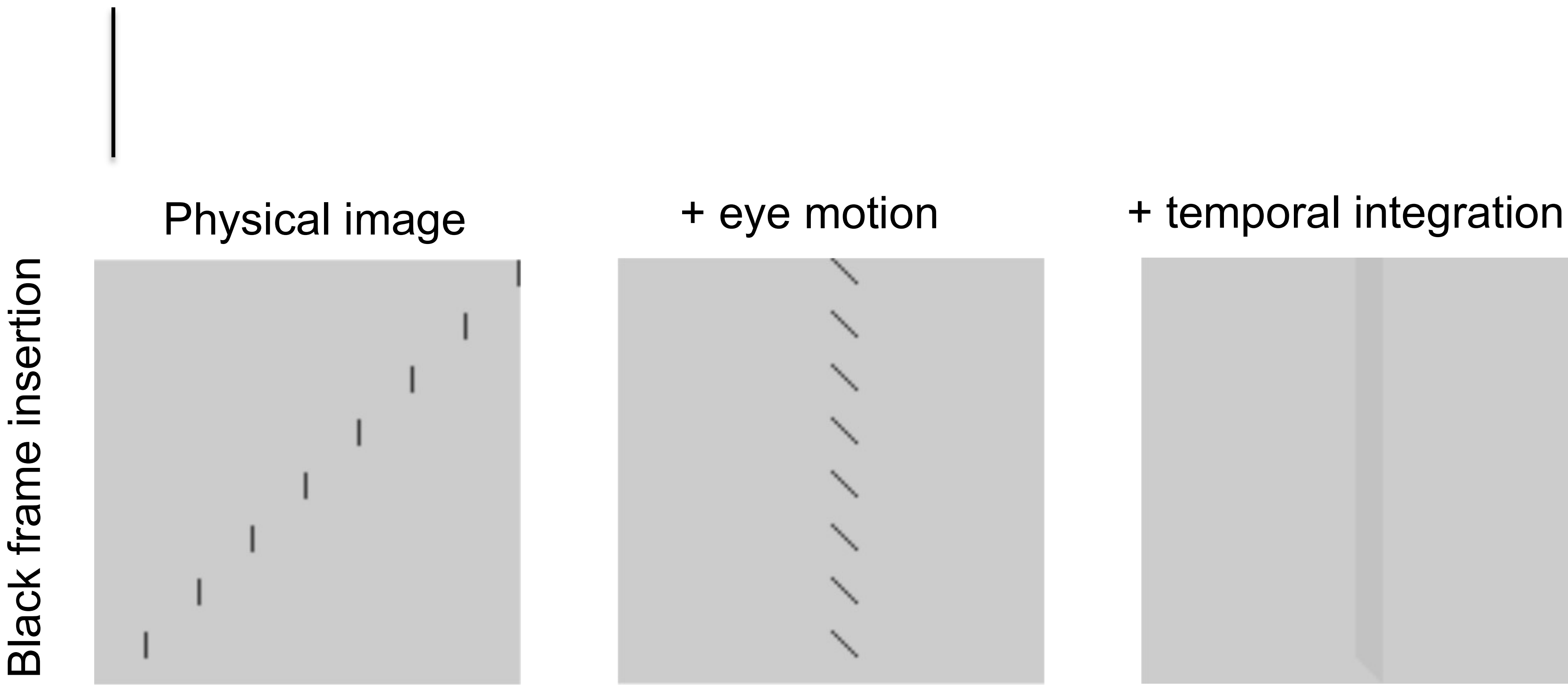


Original scene

With hold-type blur

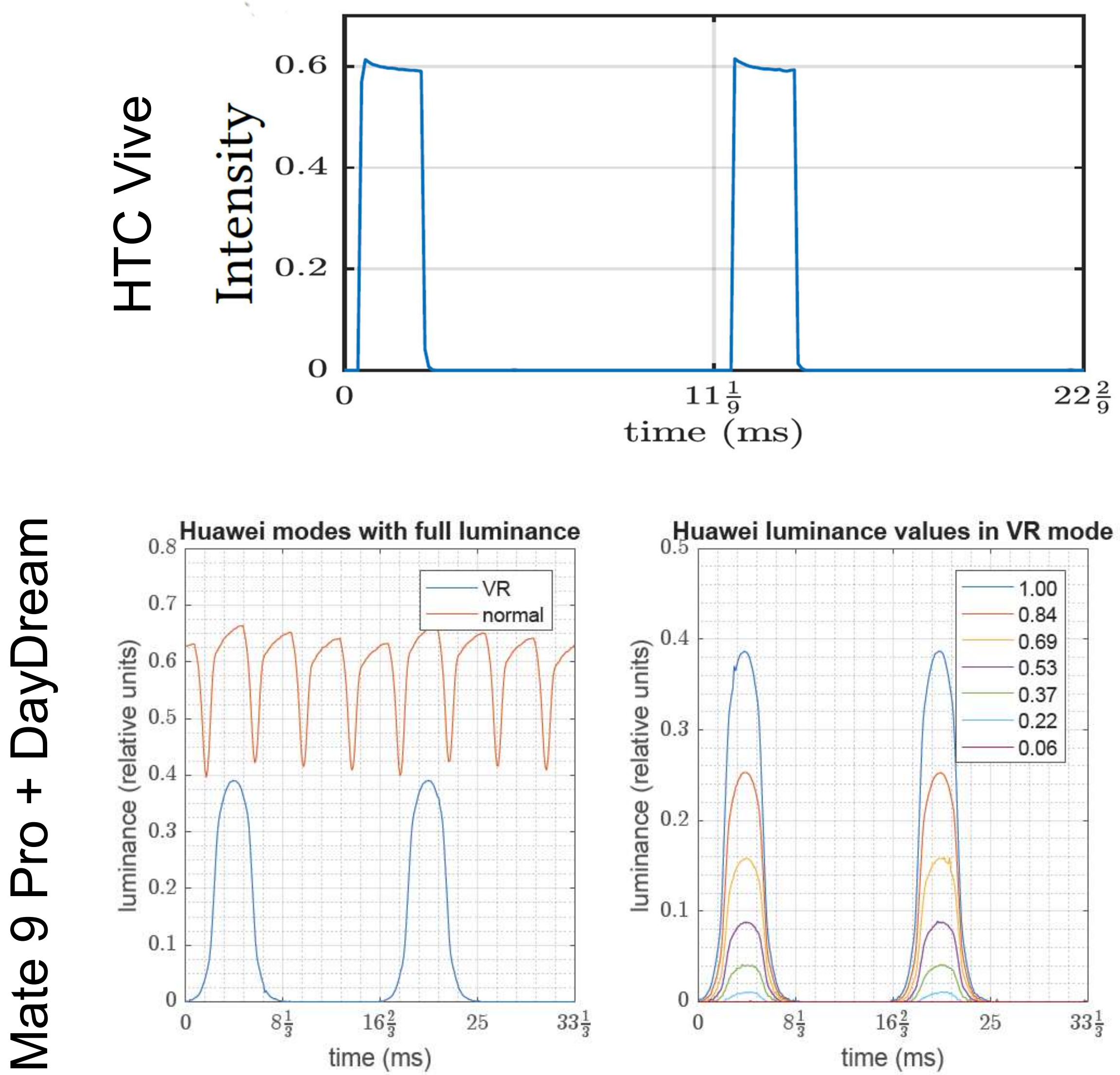
Hold-type blur

- The eye smoothly follows a moving object
- But the image on the display is “frozen” for 1/60th of a second



Low persistence displays

- Most VR displays flash an image for a fraction of frame duration
- This reduces hold-type blur
- And also reduces the perceived lag of the rendering



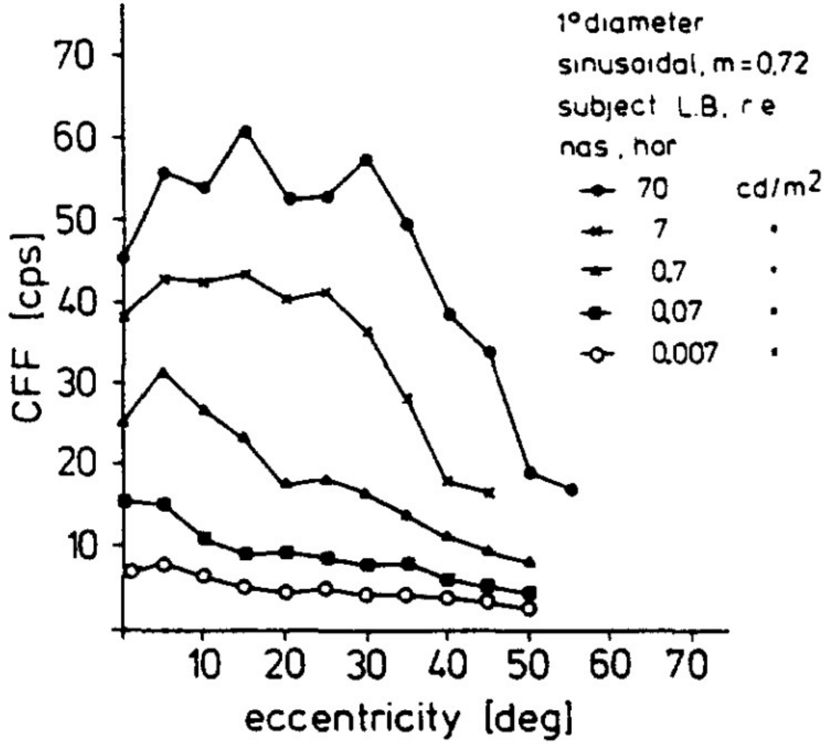
Black frame insertion

- Which invader appears sharper?

- A similar idea to low-persistence displays in VR
- Reduces hold-type blur

Flicker

- Critical Flicker Frequency
  - The lowest frequency at which flickering stimulus appears as a steady field
  - Measured for full-on / off presentation
  - Strongly depends on luminance – big issue for HDR VR headsets
  - Increases with eccentricity
  - and stimulus size
  - It is possible to detect flicker even at 2kHz
    - For saccadic eye motion

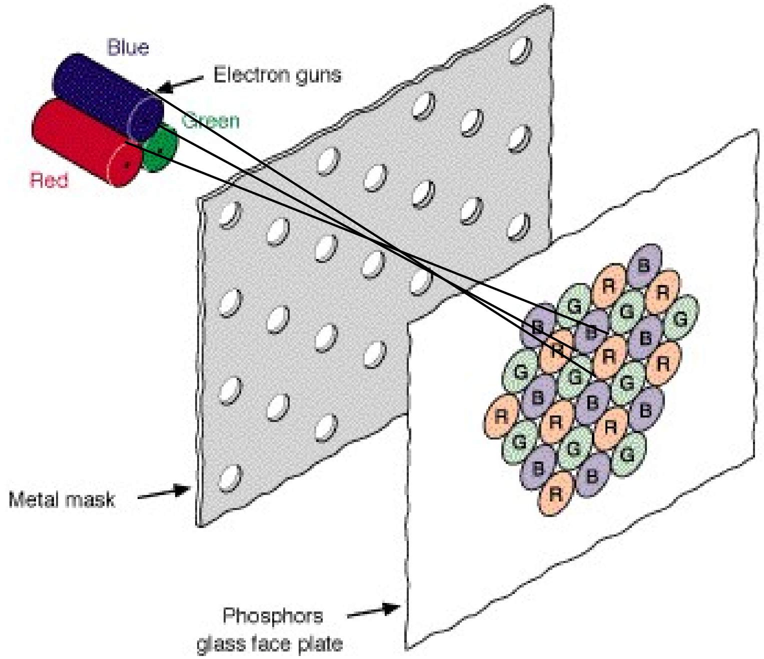


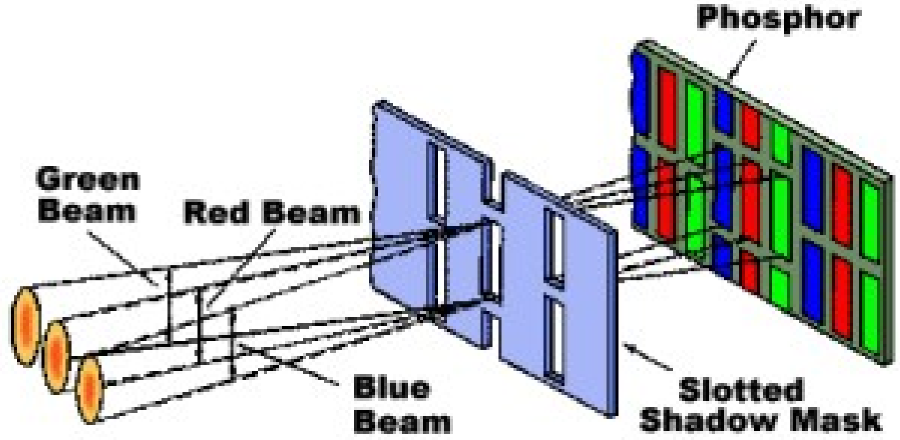
[Hartmann et al. 1979]

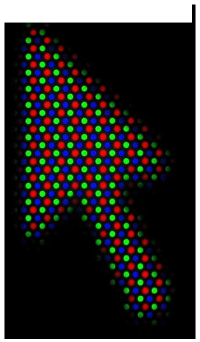
Overview

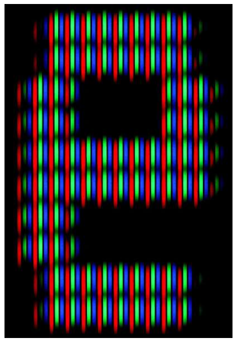
- Temporal aspects
  - Latency in VR
  - Eye-movement
  - Hold-type blur
- 2D displays
  - 2D spatial light modulators
  - High dynamic range displays

Cathode Ray Tube





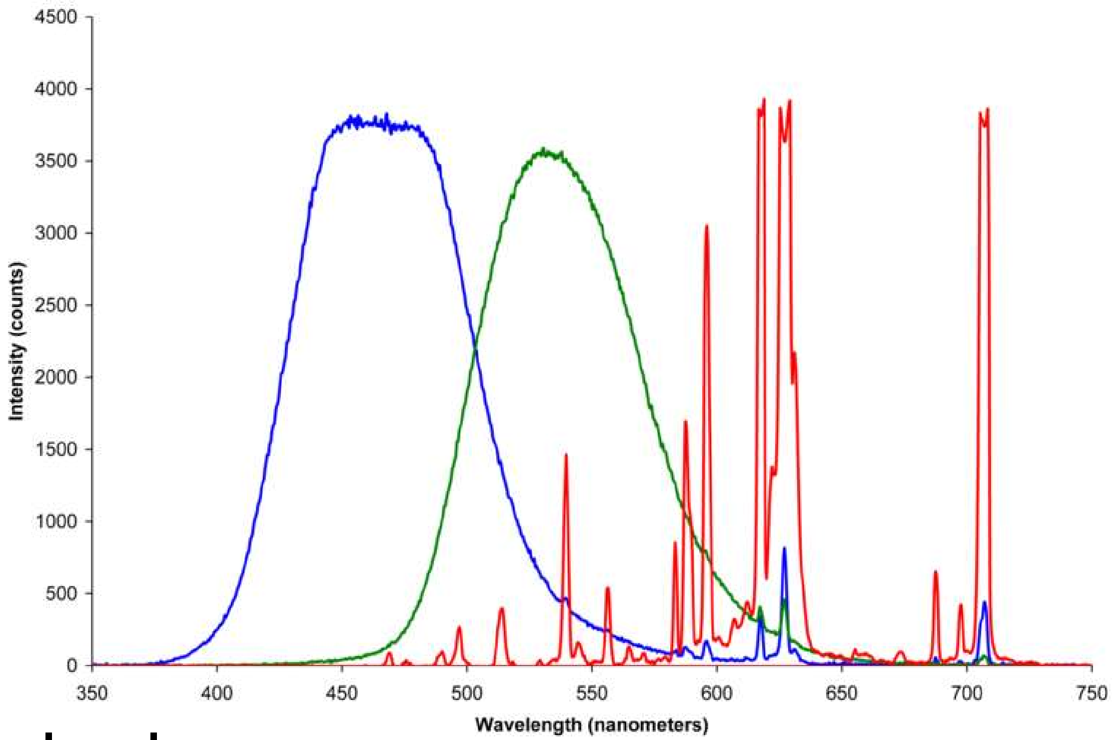




[from wikipedia]

Spectral Composition

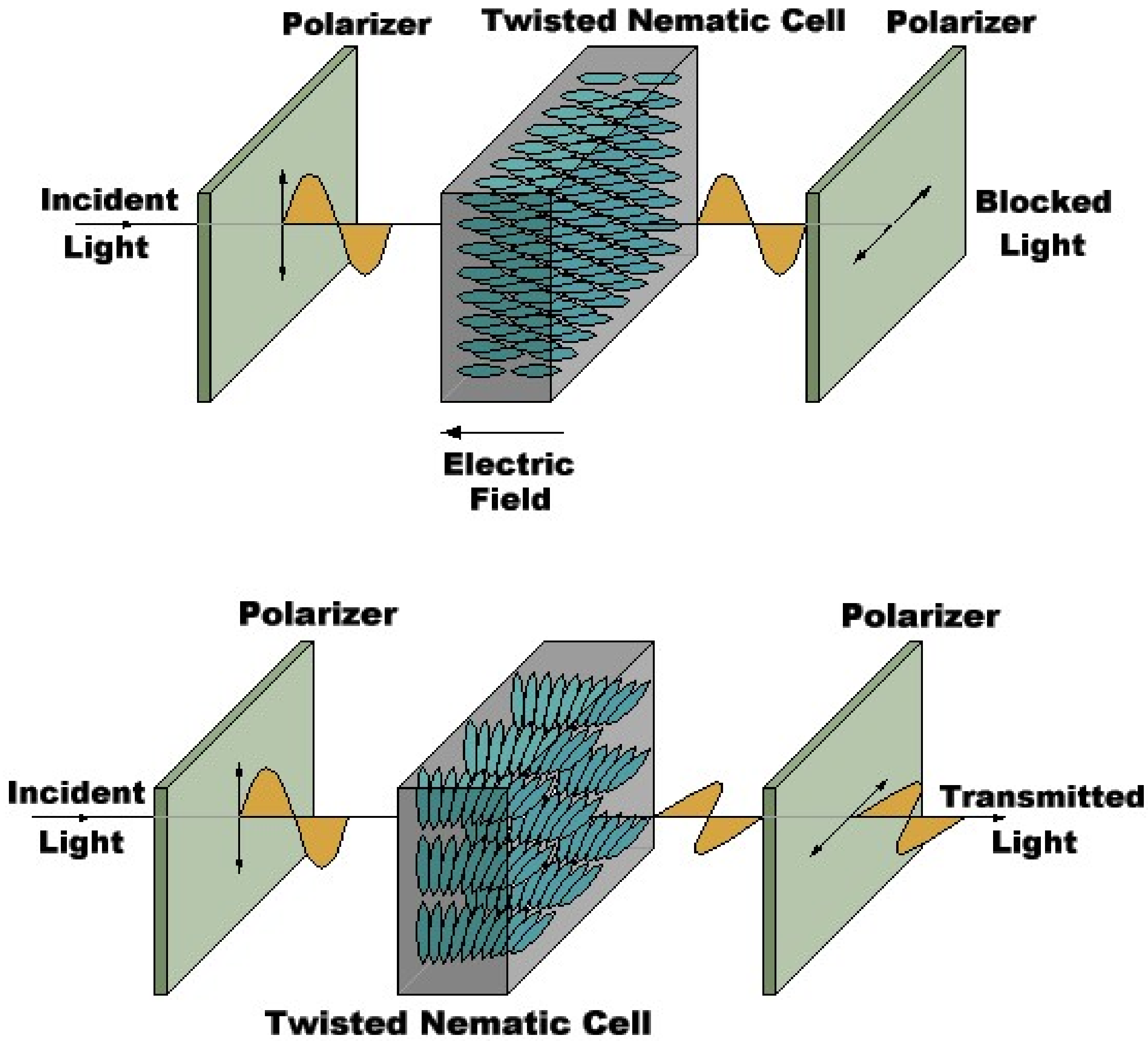
- three different phosphors



[from wikipedia]

- saturated and natural colors
- inexpensive
- high contrast and brightness

Liquid Chrystal Displays (LCD)



From: http://computer.howstuffworks.com/monitor5.htm

Twisted neumatic LC cell

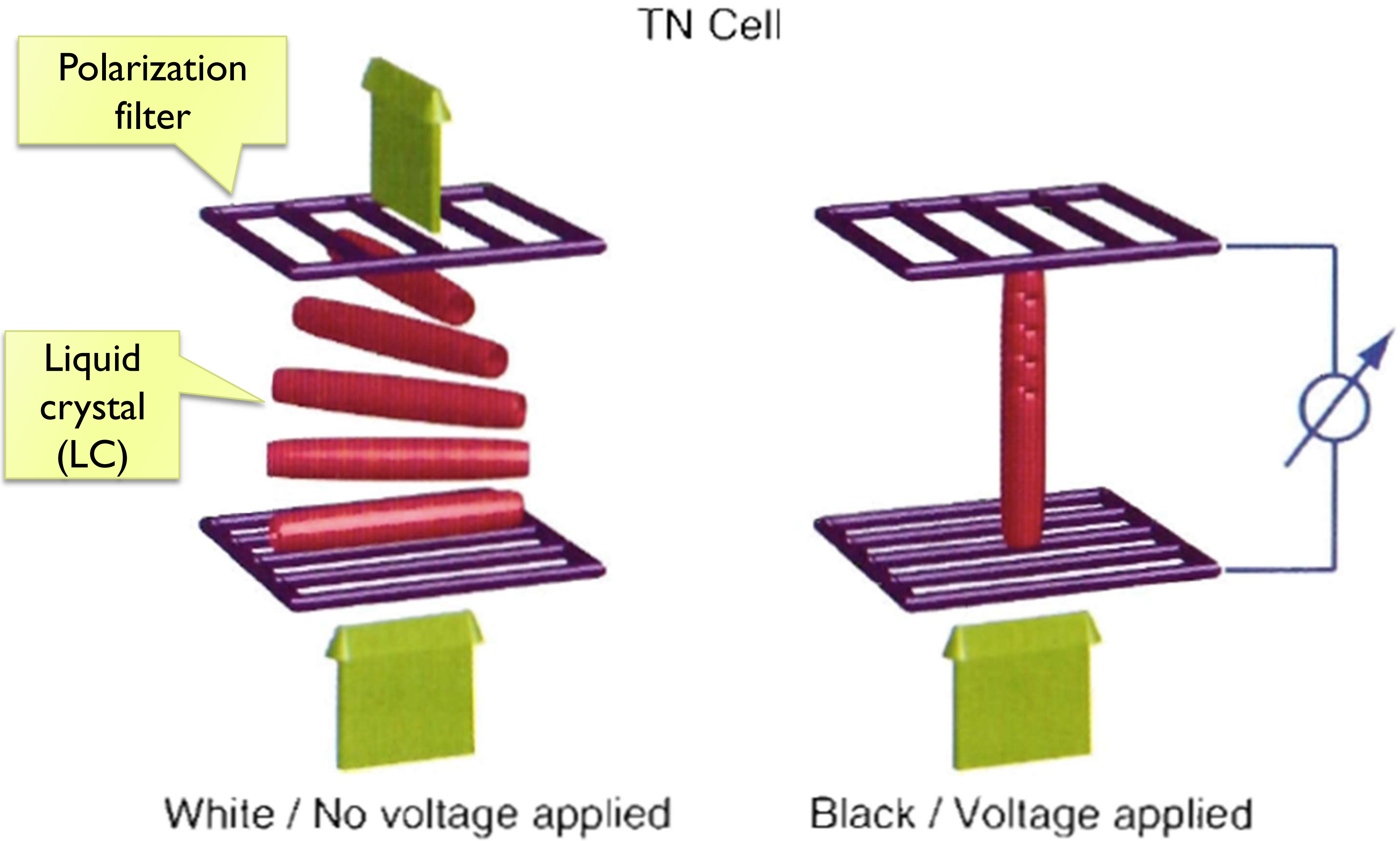


Figure from: High Dynamic Range Imaging by E. Reinhard et al.

In-plane switching cell (IPS)

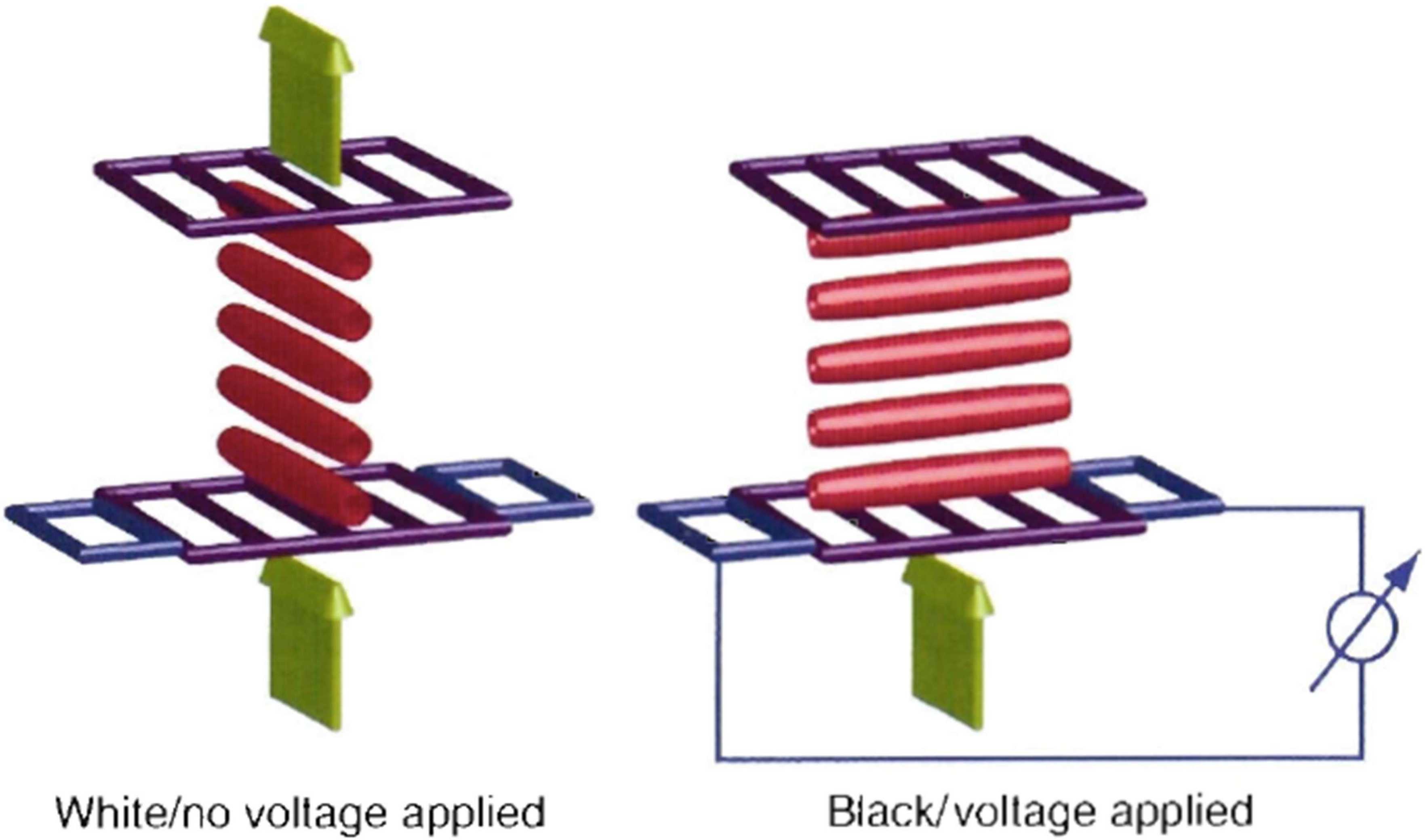
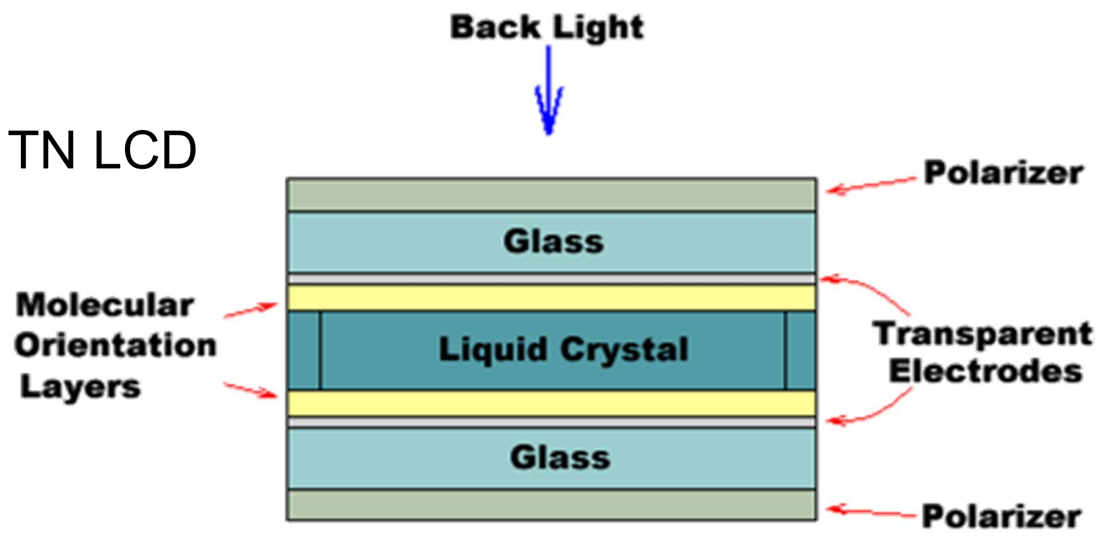


Figure from: High Dynamic Range Imaging by E. Reinhard et al.

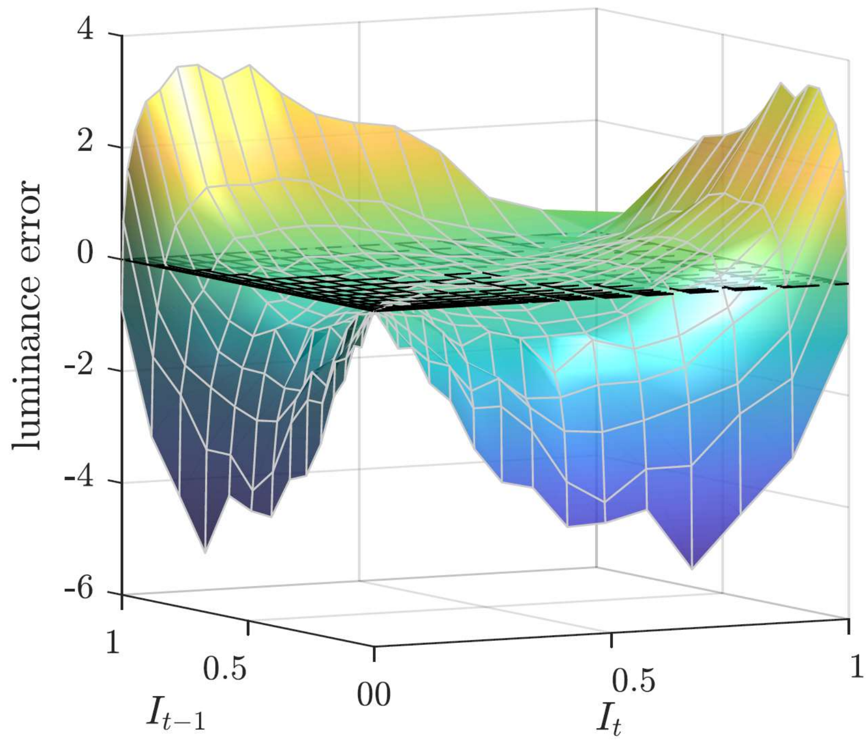
LCD



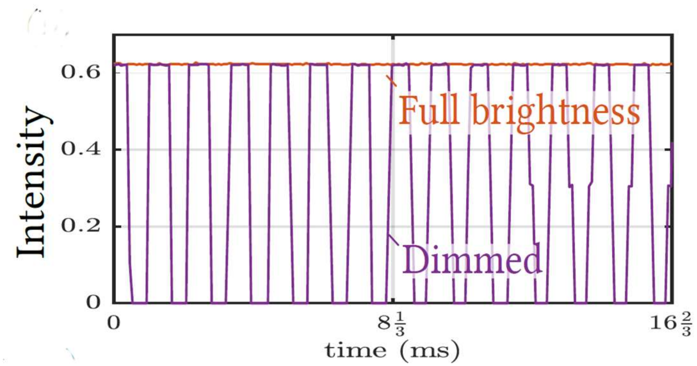
- color may change with the viewing angle
- contrast up to 3000:1
- higher resolution results in smaller fill-factor
- color LCD transmits only up to 8% (more often close to 4-5%) light when set to full white

LCD temporal response

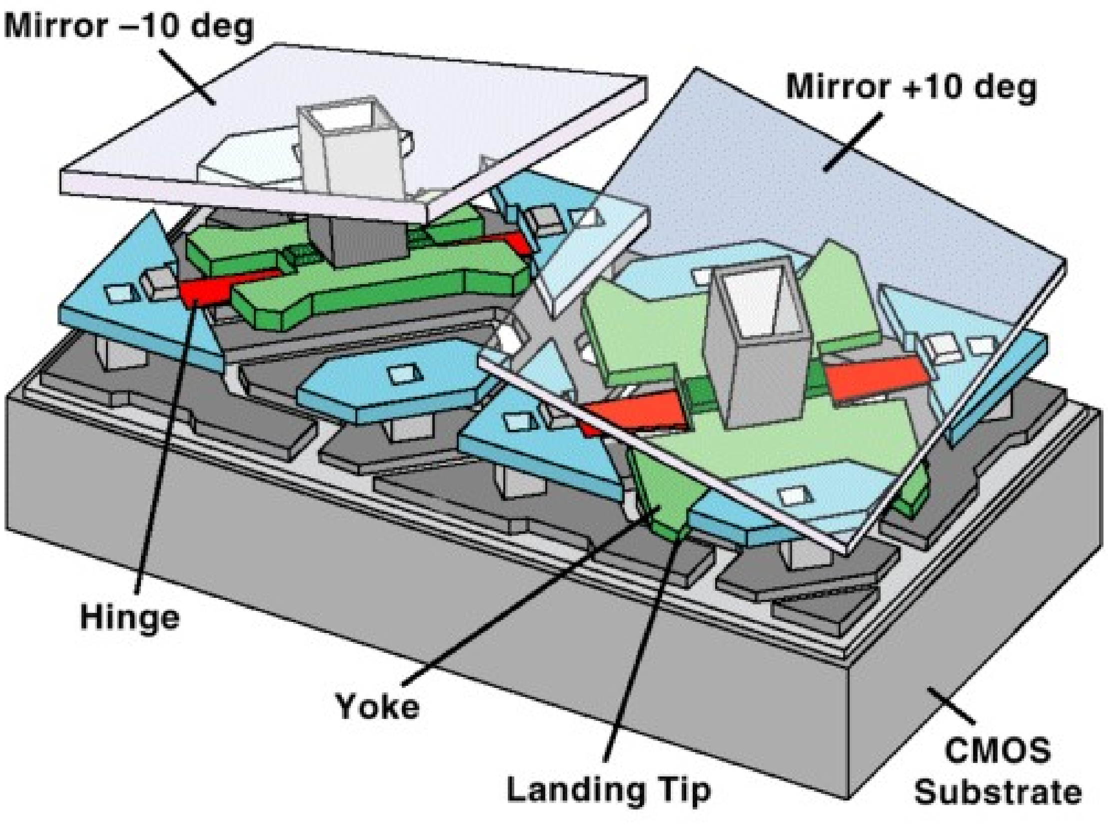
- Experiment on an IPS LCD screen
- We rapidly switched between two intensity levels at 120Hz
- Measured luminance integrated over 1s
- The top plot shows the difference between expected ($\frac{I_{t-1}+I_t}{2}$) and measured luminance

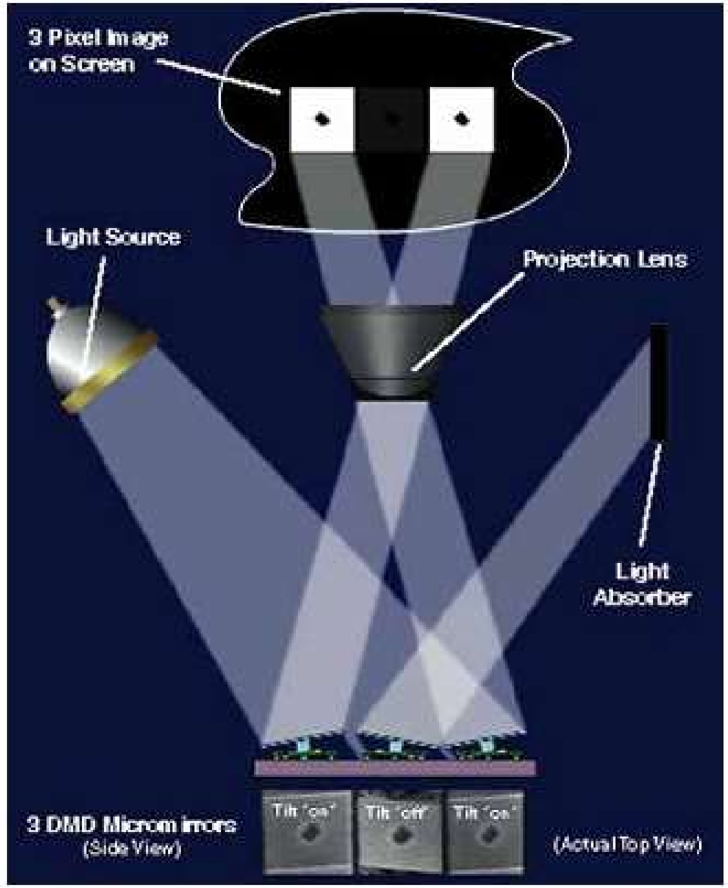


- The bottom plot: intensity measurement for the full brightness and half-brightness display settings



Digital Micromirror Devices (DMDs/DLP)

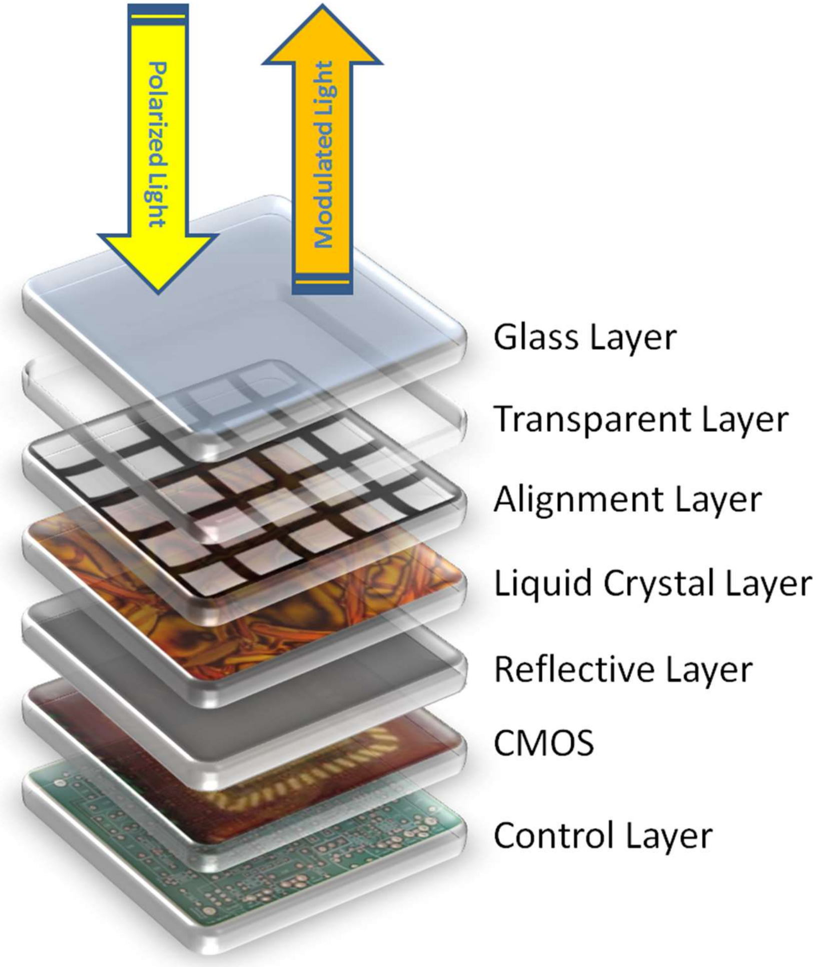




Texas Instruments

- 2-D array of mirrors
- Truly digital pixels
- Grey levels via Pulse-Width Modulation

Liquid Crystal on Silicon (LCoS)

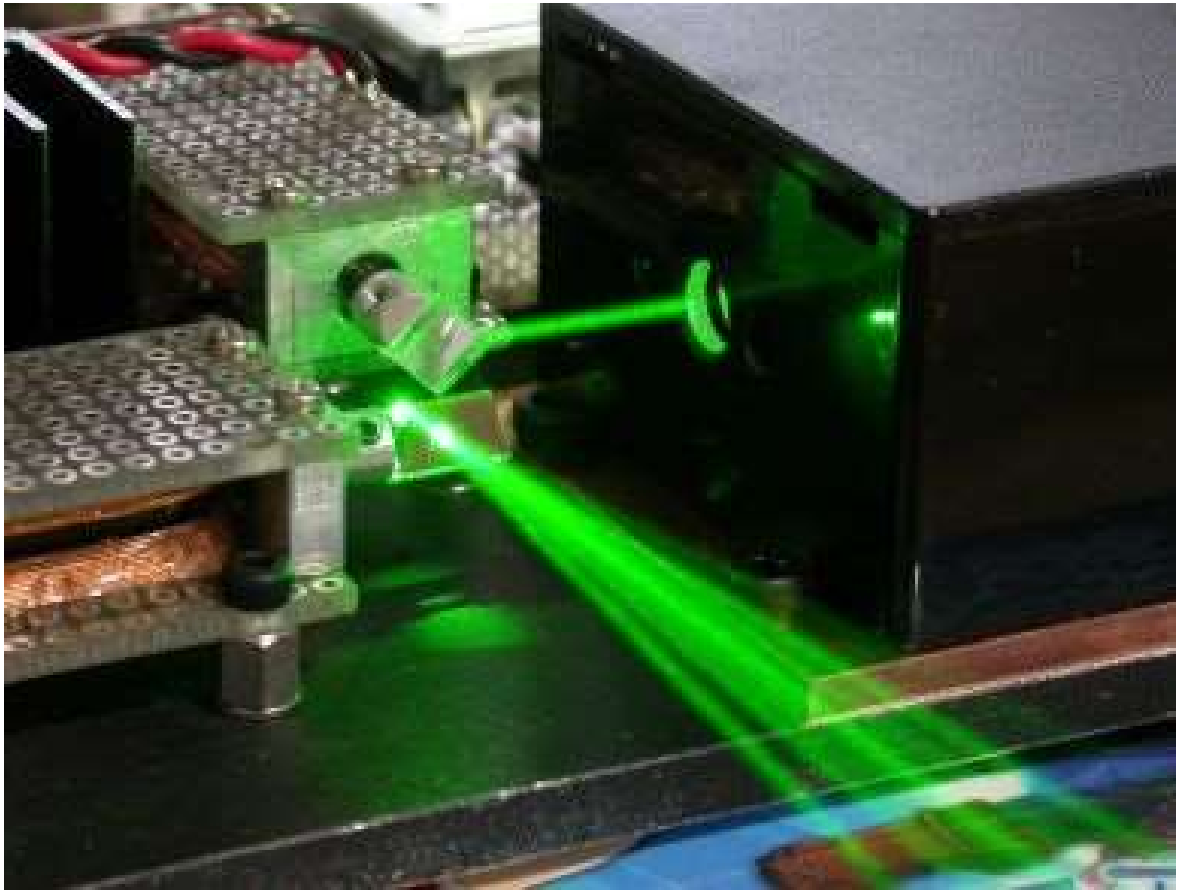


- basically a reflective LCD
- standard component in projectors and head mounted displays
- used e.g. in google glass

Scanning Laser Projector

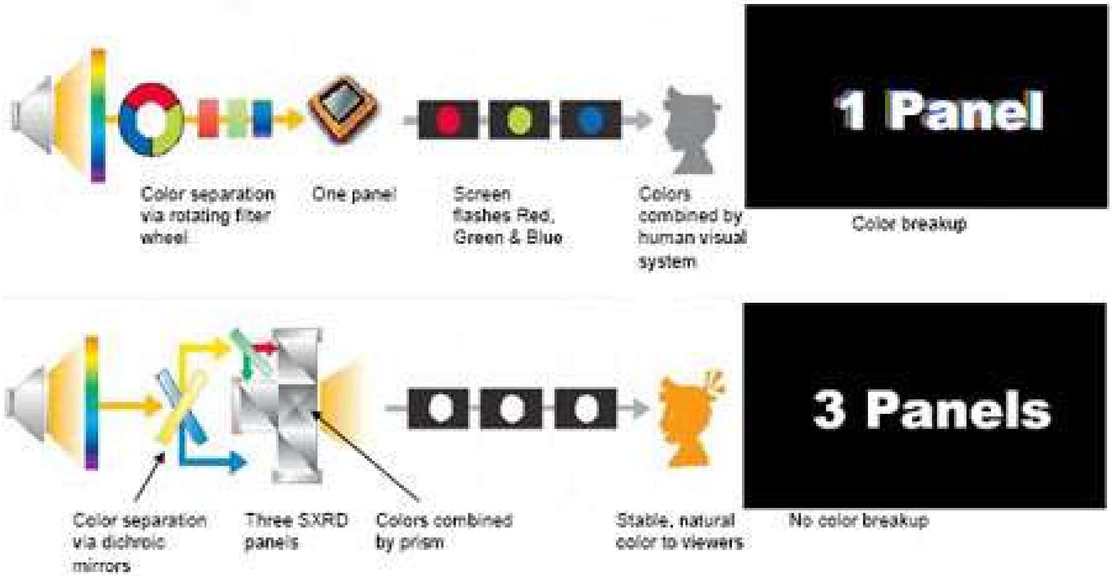
- maximum contrast
- scanning rays

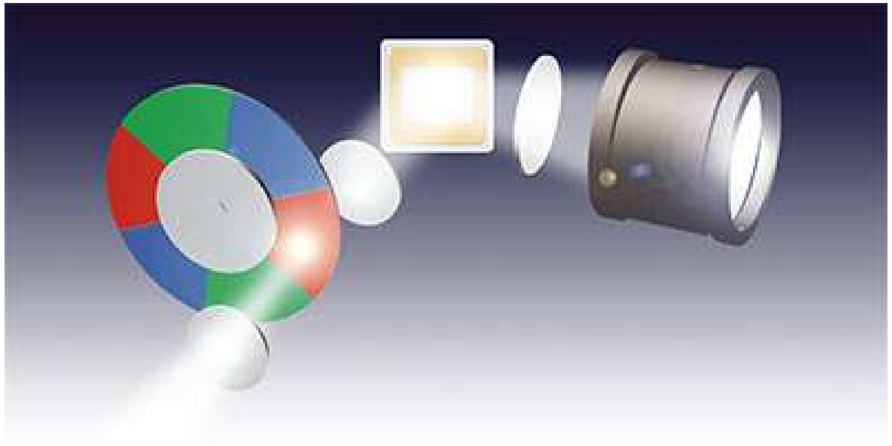
- very high power lasers needed for high brightness



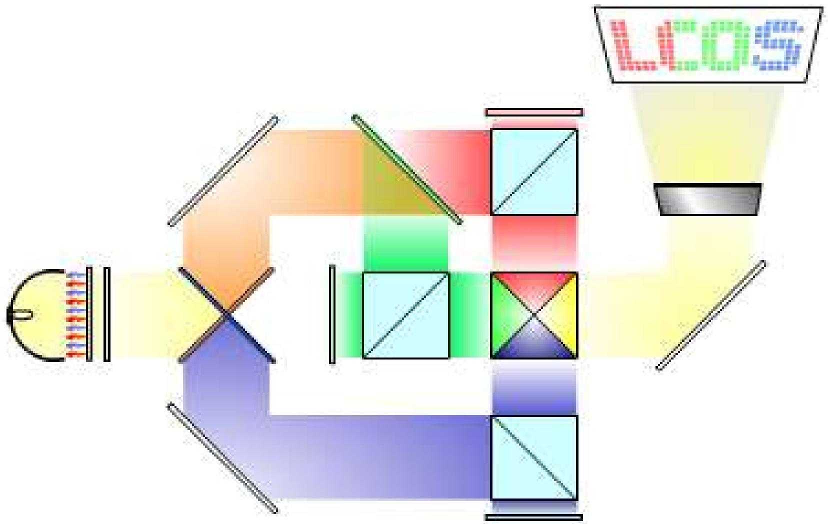
http://elm-chan.org/works/vlp/report_e.html

3-chip vs. Color Wheel Display





- color wheel
  - cheap
  - time sequenced colors
  - color fringes with motion/video
- 3-chip
  - complicated setup
  - no color fringes

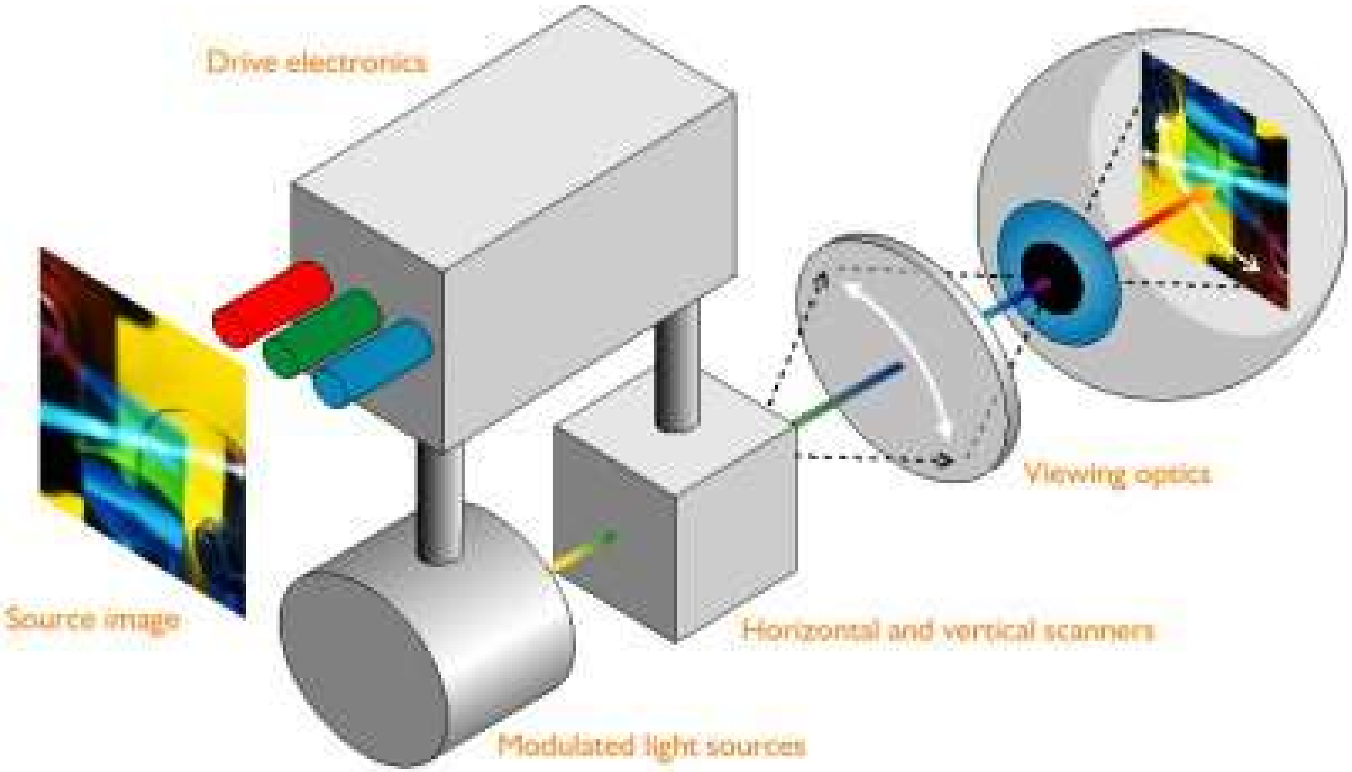


AiRScouter

Virtual Retinal Display

- projection onto the retina
- challenge – small viewing box

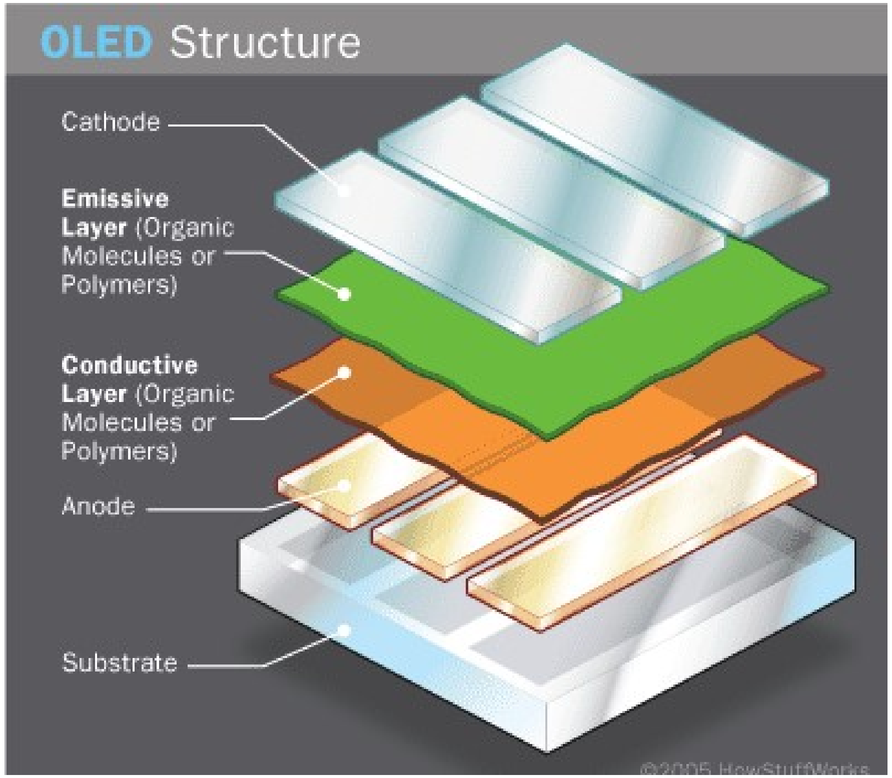
From:
http://www.engadget.com/2010/09/17/brothers-airscouter-floats-a-16-inch-display-onto-your-eye-bisc/



Google – project Glass

OLED

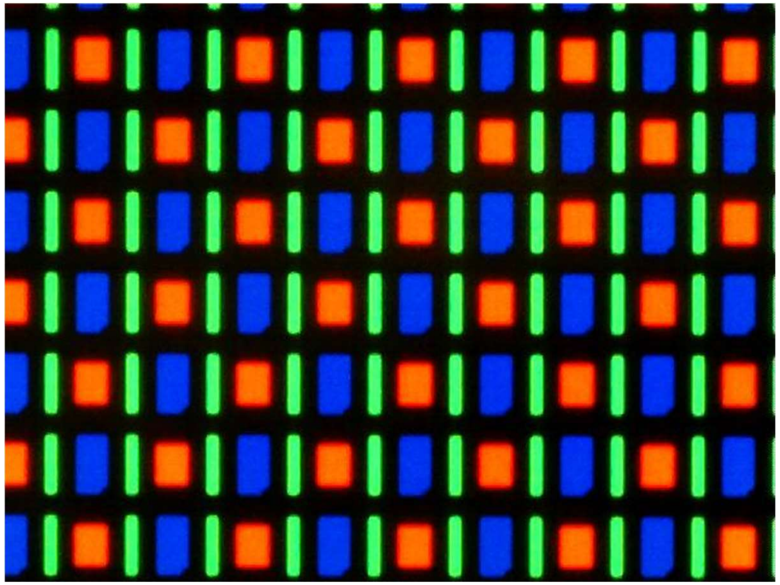
- based on electrophosphorescence
- large viewing angle
- the power consumption varies with the brightness of the image
- fast (< 1 microsec)
- arbitrary sizes



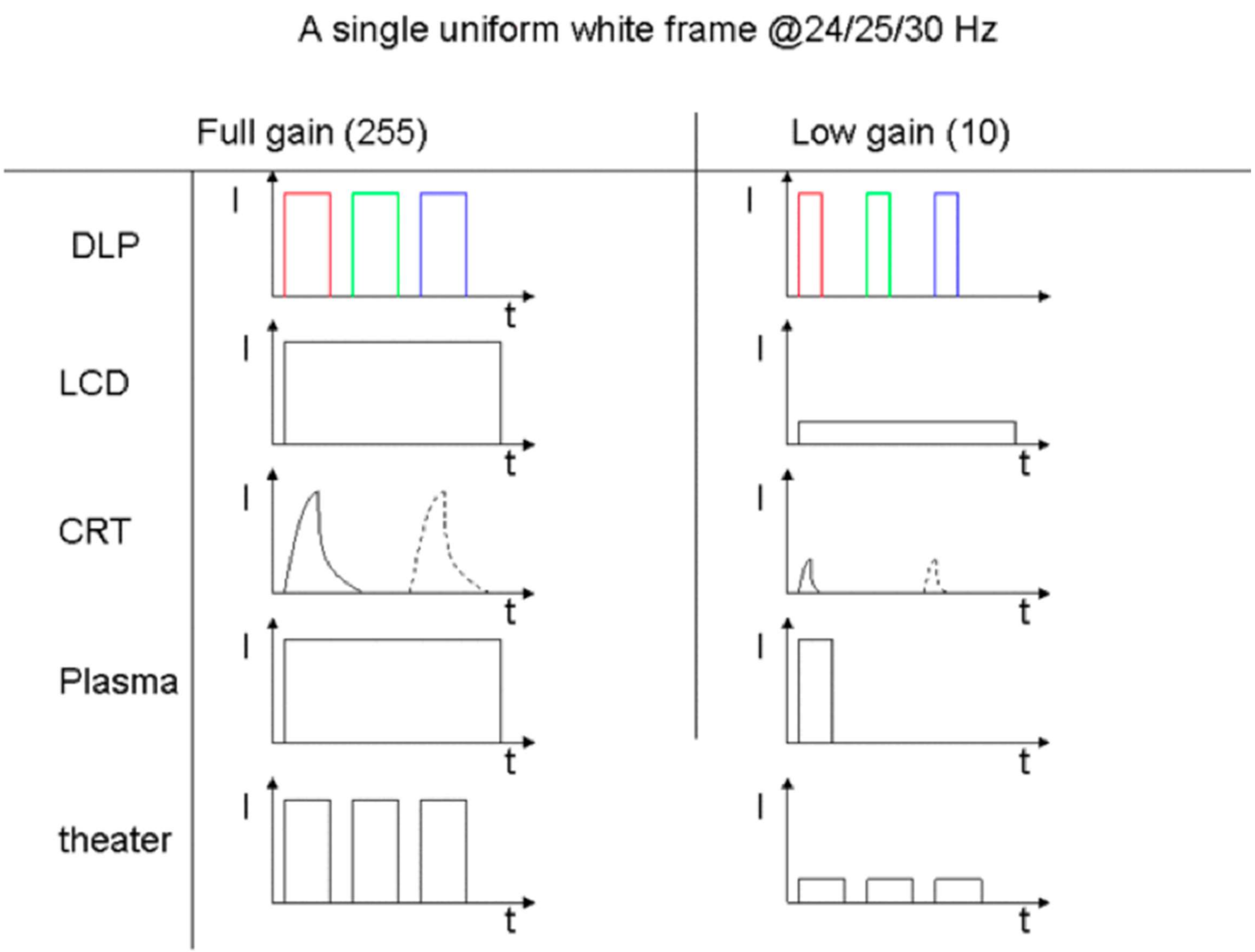
- life-span can be short
  - Worst for blue OLEDs

Active matrix OLED

- Commonly used in mobile phones (AMOLED)
- Very good contrast
  - But the screen more affected by glare than LCD
- But limited brightness
  - The brighter is OLED, the shorter is its live-span



Temporal characteristic

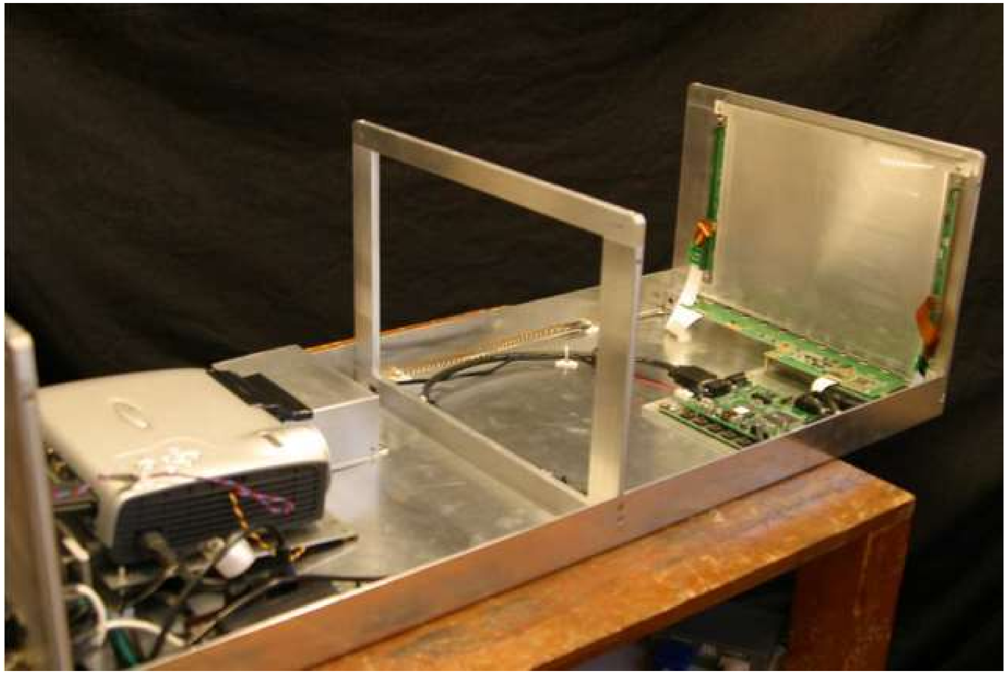


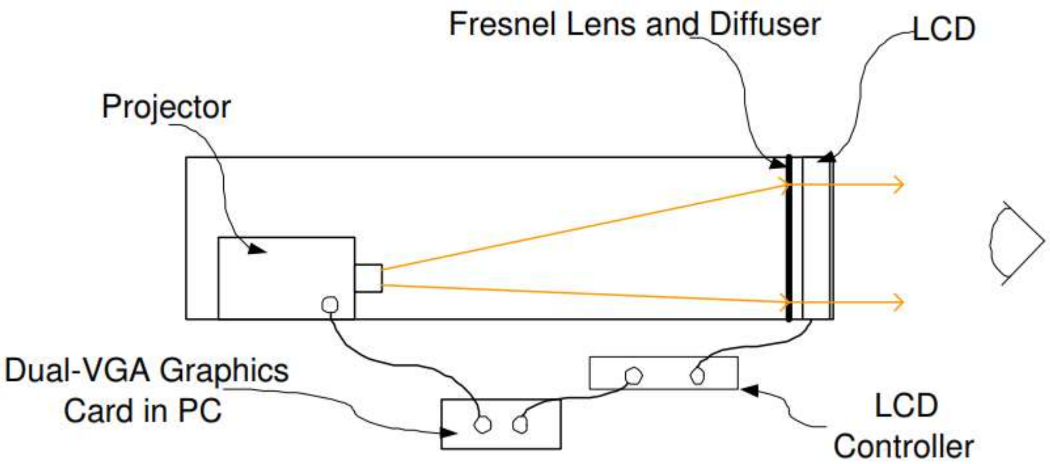
From: http://en.wikipedia.org/wiki/Comparison_of_display_technology

Electronic Paper

www.eink.com

Prototype HDR display (2004)

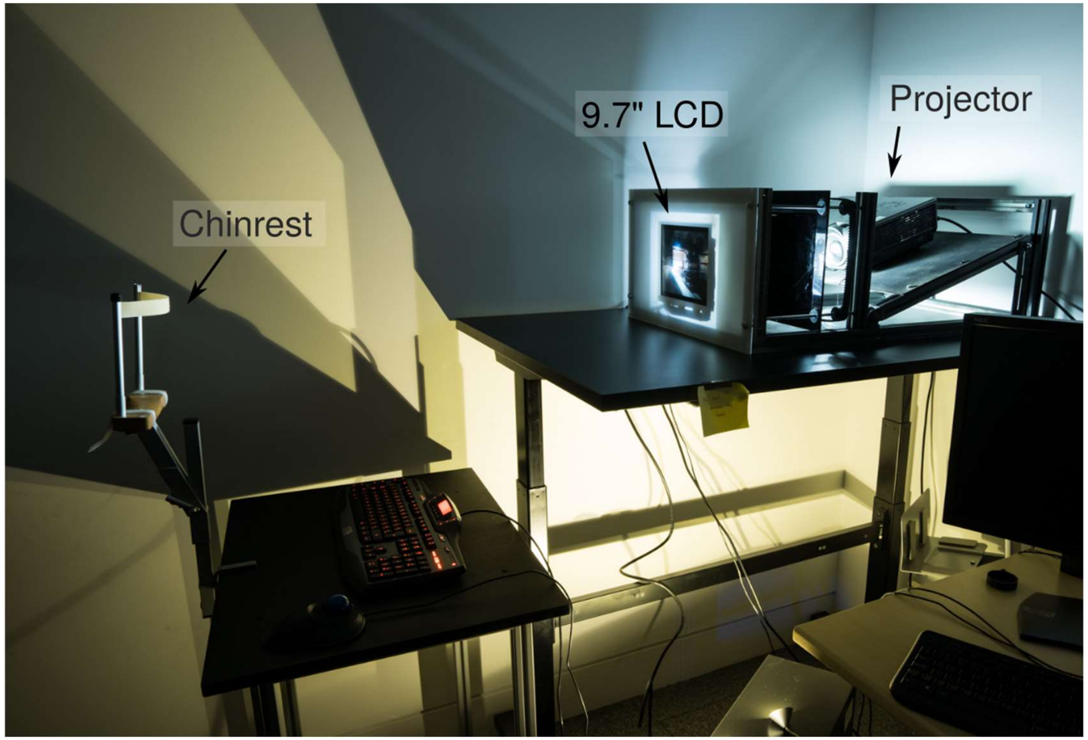


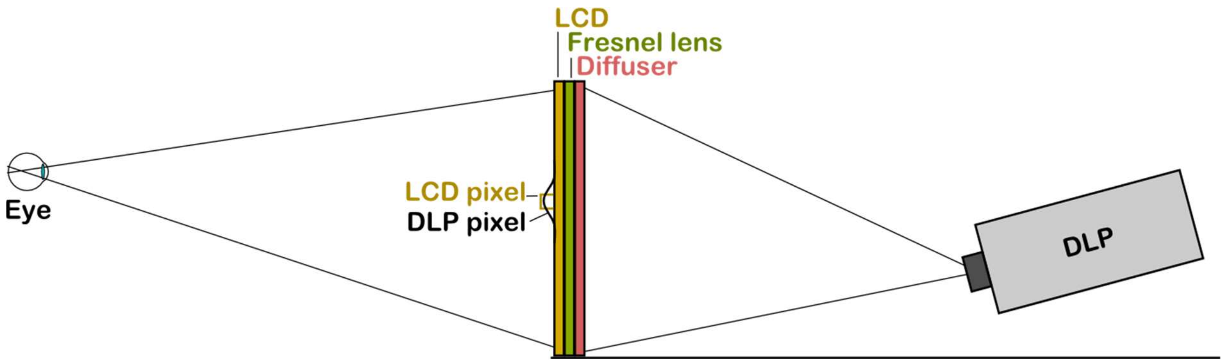


From [Seetzen et al. SIGGRAPH 2004]

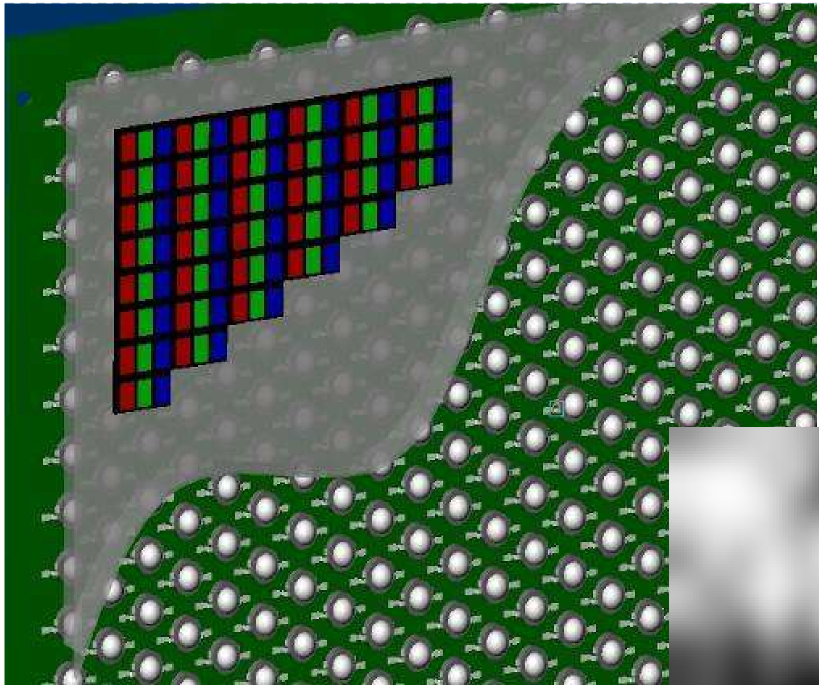
Cambridge experimental HDR display

- 35,000 cd/m^2 peak luminance
- 0.01 cd/m^2 black level
- LCD resolution: 2048x1536
- Backlight (DLP) resolution: 1024x768
- Geometric-calibration with a DSLR camera
- Display uniformity compensation
- Bit-depth of DLP and LCD extended to 10 bits using spatio-temporal dithering





Modern HDR displays



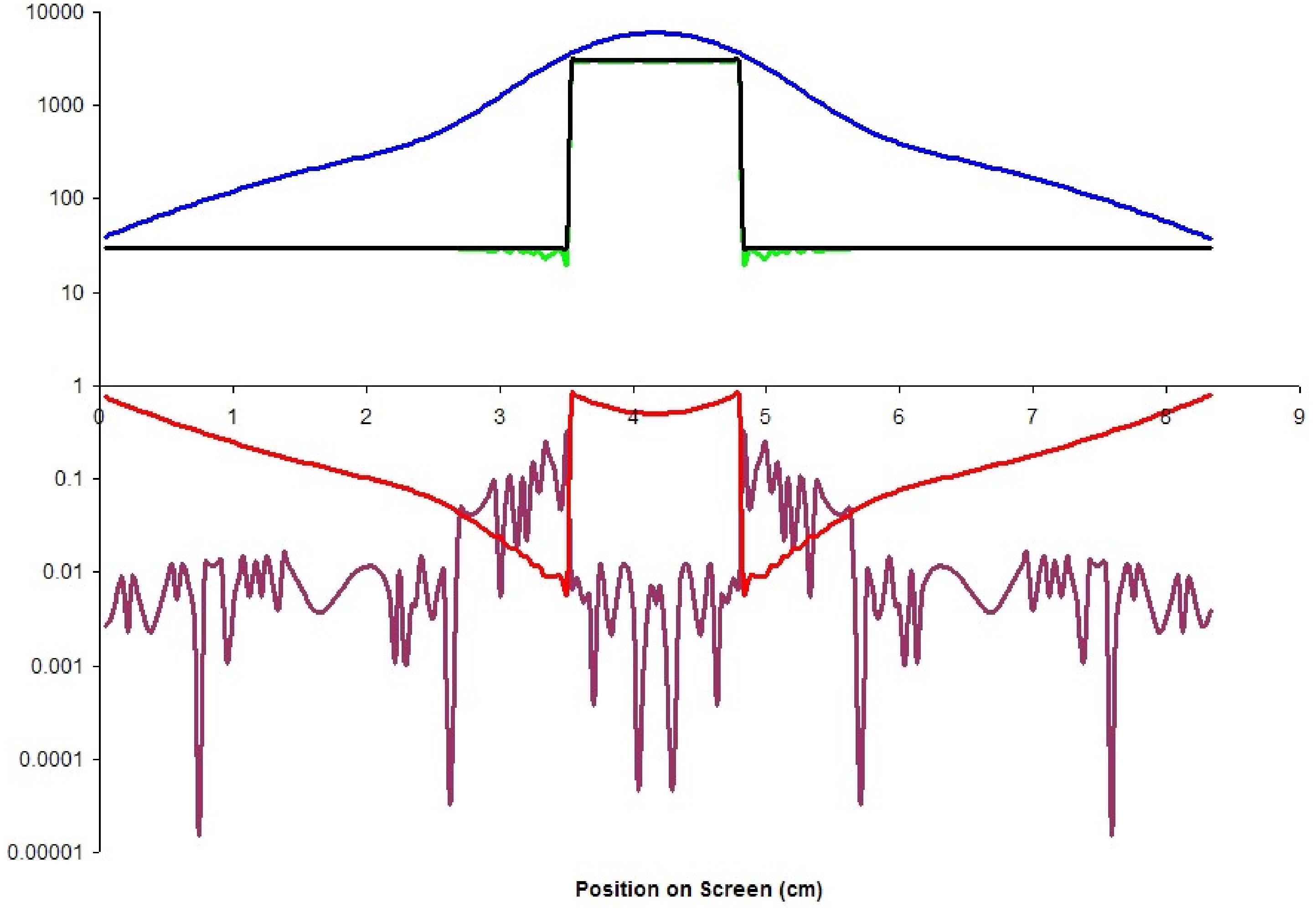
- Modulated LED array
- Conventional LCD
- Image compensation

Low resolution LED Array x High resolution Colour Image = High Dynamic Range Display

HDR Display

- Two spatial modulators
  - 1st modulator contrast 1000:1
  - 2nd modulator contrast 1000:1
  - Combined contrast 1000,000:1
- Idea: Replace constant backlight of LCD panels with an array of LEDs
  - Very few (about 1000) LEDs sufficient
  - Every LED intensity can be set individually
  - Very flat form factor (fits in standard LCD housing)
- Issue:
  - LEDs larger than LCD pixels
  - This limits maximum local contrast

Veiling Luminance



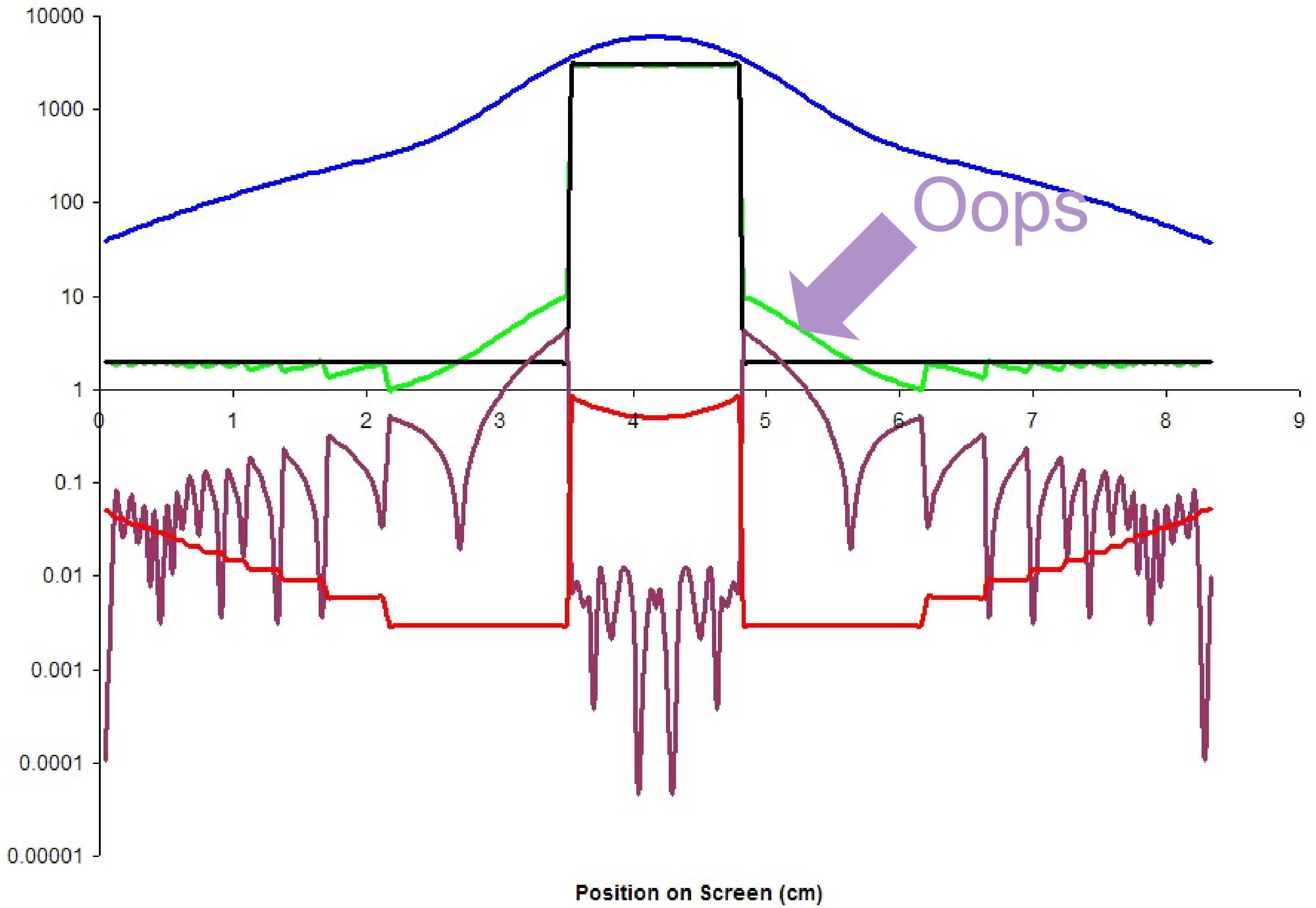
Receive Image

Drive LED

Divide Image by LED light field to obtain LCD values

Output Luminance is the product of LED light field and LCD transmission (modest error)

Veiling Luminance



Receive Image

Drive LED

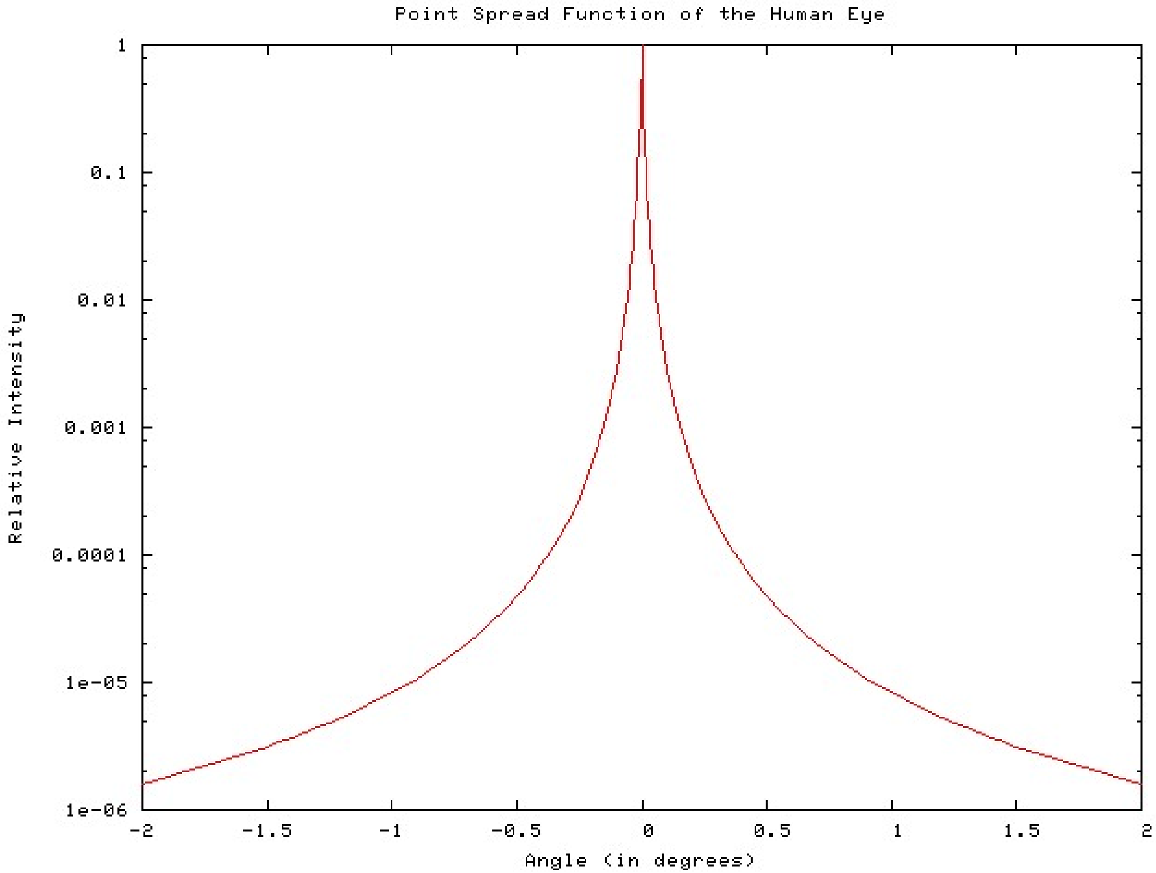
Divide Image by LED light field to obtain LCD values

Output Luminance is the product of LED light field and LCD transmission

(Problematic error)

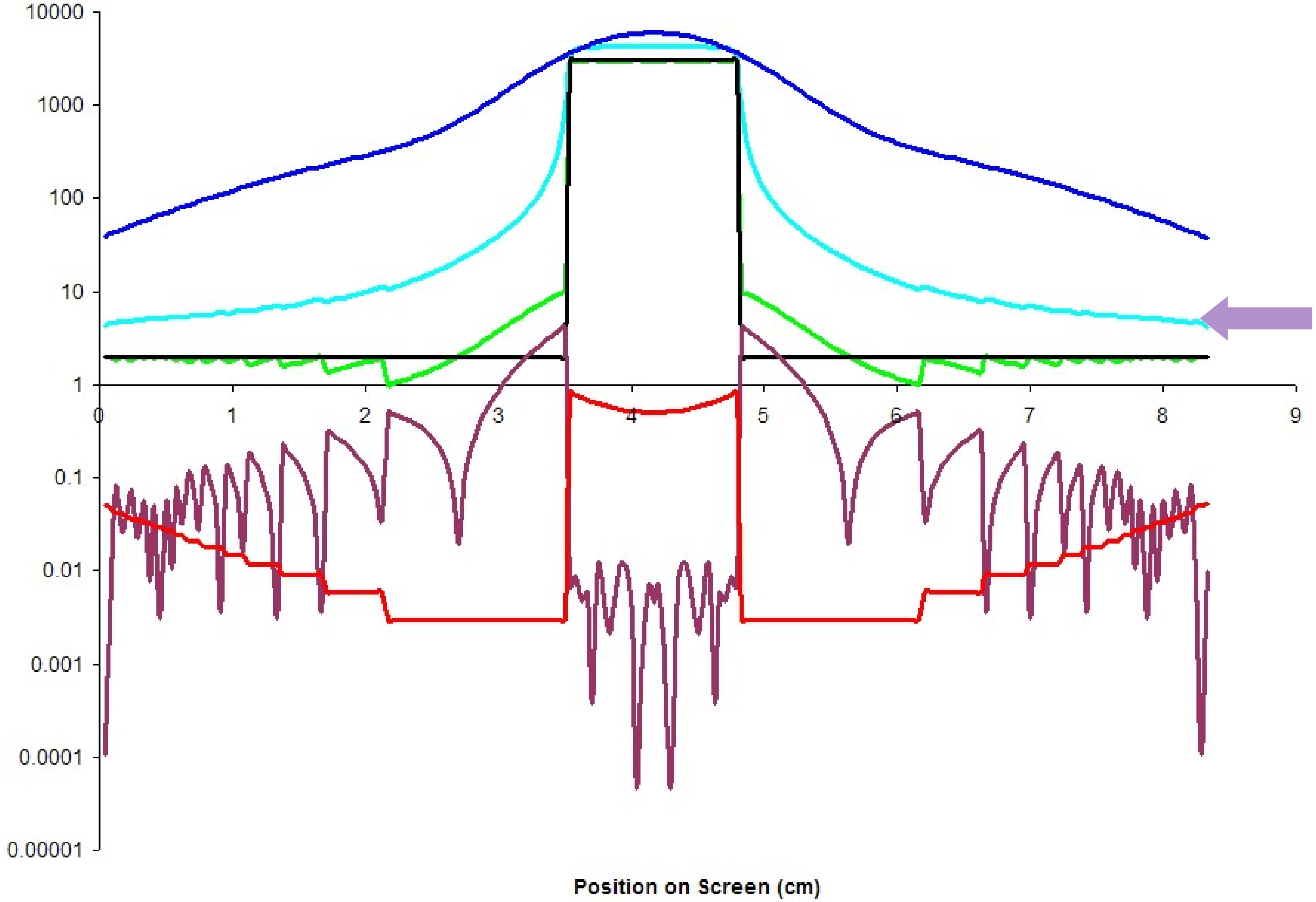
Veiling Luminance

- Maximum perceivable contrast
  - Globally very high (5-6 orders of magnitude)
    - That is why we create these displays!
  - Locally can be low: 150:1
- Point-spread function of human eye
  - Refer to „HDR and tone mapping" lecture
  - Consequence: high contrast edges cannot be perceived at full contrast



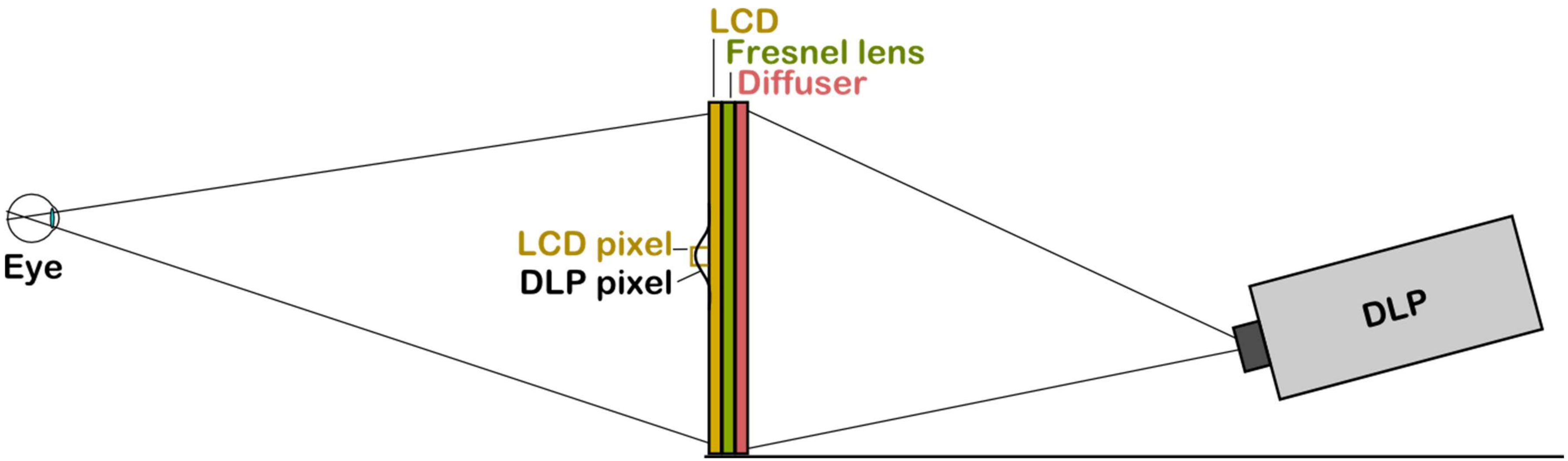
Veiling Glare (Camera)

Veiling Luminance



Veiling Luminance masks imperfection

HDR rendering algorithm - high level



Desired image

DLP blur (PSF)

$$\underset{L,D}{\operatorname{argmin}} \left|\left| I(x,y) - g * D(x,y)L(x,y) \right|\right|_2$$

DLP image

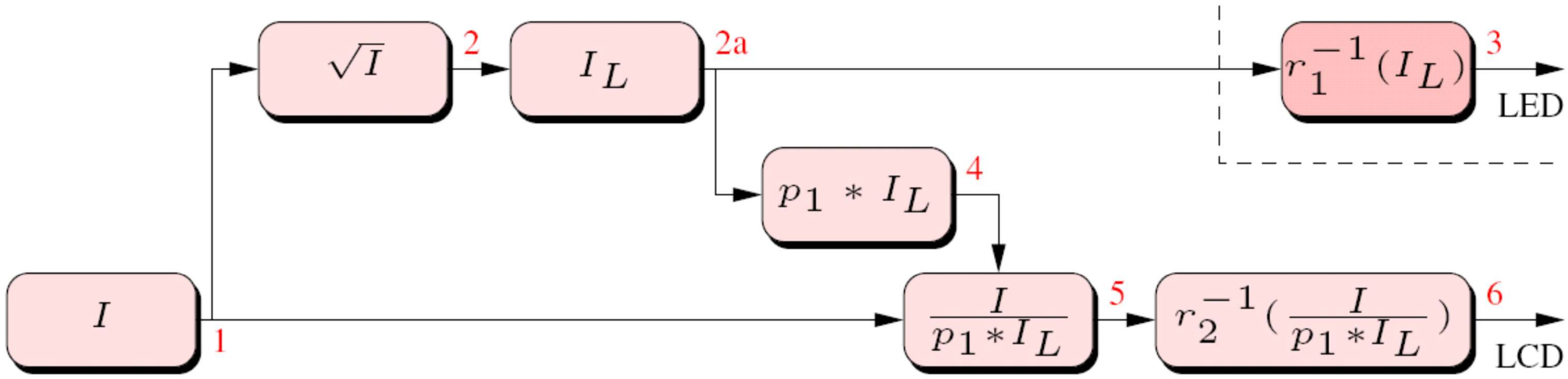
LCD image

Subject to:

$$\forall(x,y) \quad L_{min} \leq L(x,y) \leq L_{max}$$

$$\forall(x,y) \quad D_{min} \leq D(x,y) \leq D_{max}$$

Simplified HDR rendering algorithm



Rendering Algorithm

References

- HAINICH, R.R. AND BIMBER, O. 2011. *Displays: Fundamentals and Applications*. CRC Press.
- SEETZEN, H., HEIDRICH, W., STUERZLINGER, W., ET AL. 2004. High dynamic range display systems. *ACM Transactions on Graphics 23*, 3, 760.
- Visual motion test for high-frame-rate monitors:
  - https://www.testufo.com/

UNIVERSITY OF CAMBRIDGE
COMPUTER LABORATORY

RAINBOW GRAPHICS GROUP

Advanced Graphics & Image Processing

Stereo Rendering

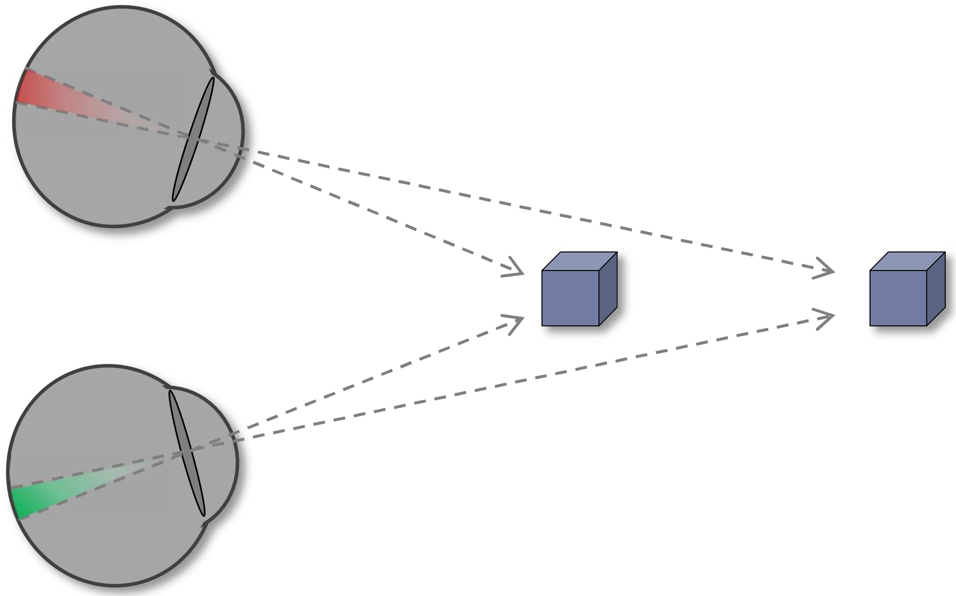
Part 1/3 – depth perception

Rafał Mantiuk
Dept. of Computer Science and Technology, University of Cambridge

Depth perception

We see depth due to depth cues.

Stereoscopic depth cues:
binocular disparity



The slides in this section are the courtesy of Piotr Didyk (http://people.mpi-inf.mpg.de/~pdidyk/)

Depth perception

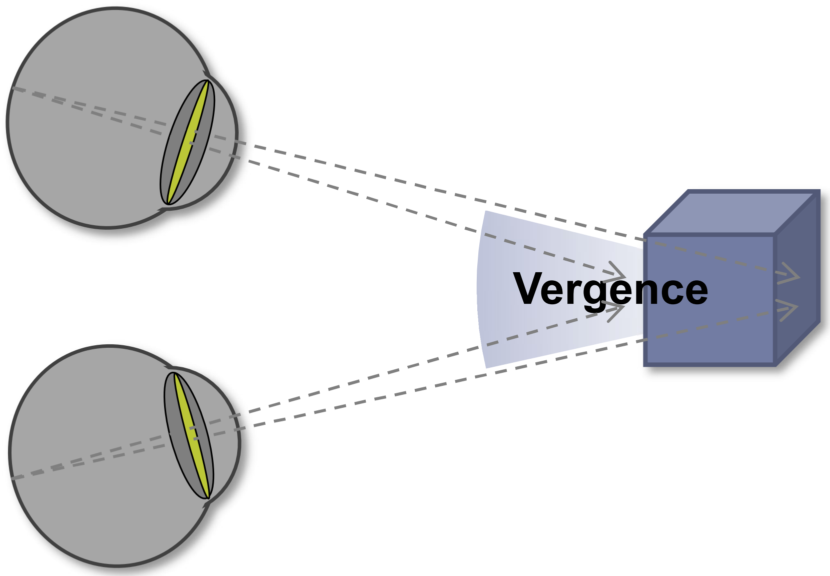
We see depth due to depth cues.

Stereoscopic depth cues:

binocular disparity

Ocular depth cues:

accommodation, vergence



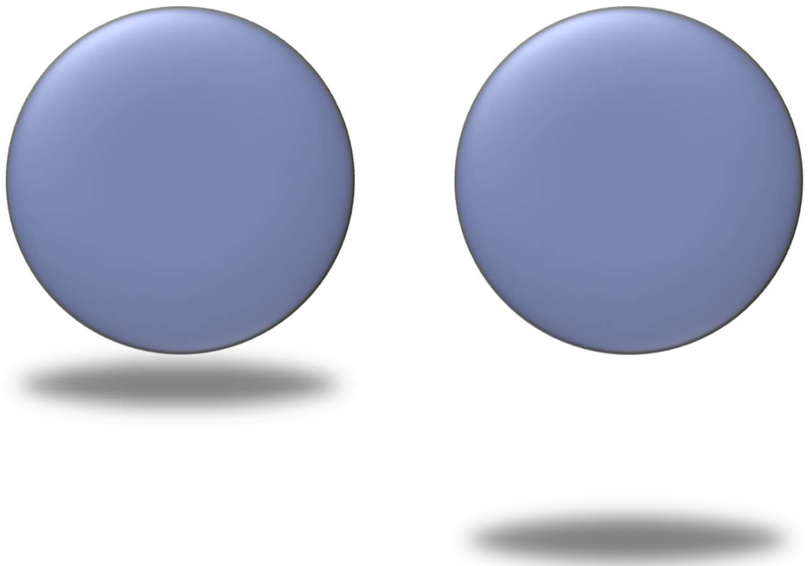
Depth perception

We see depth due to depth cues.

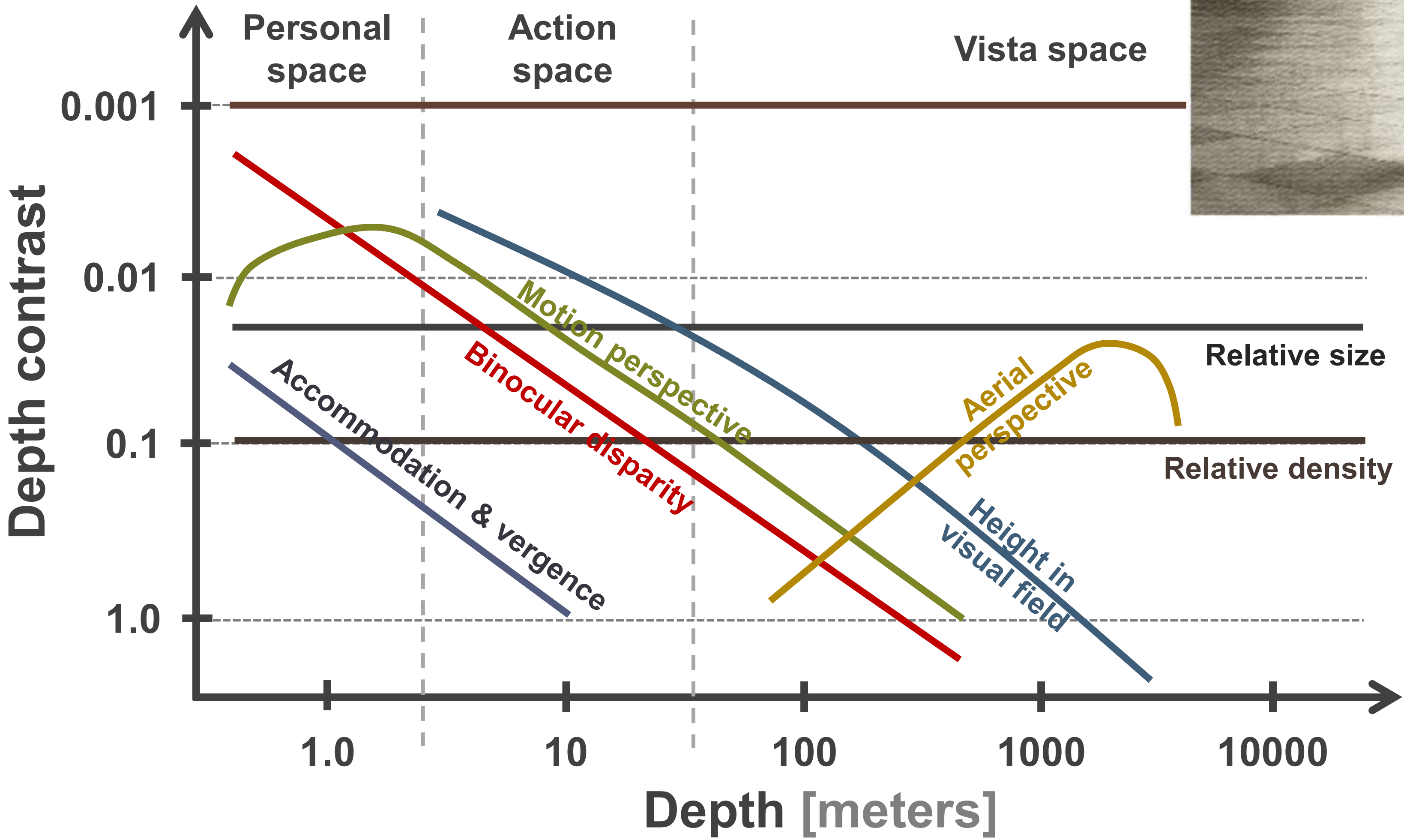
Stereoscopic depth cues:
binocular disparity

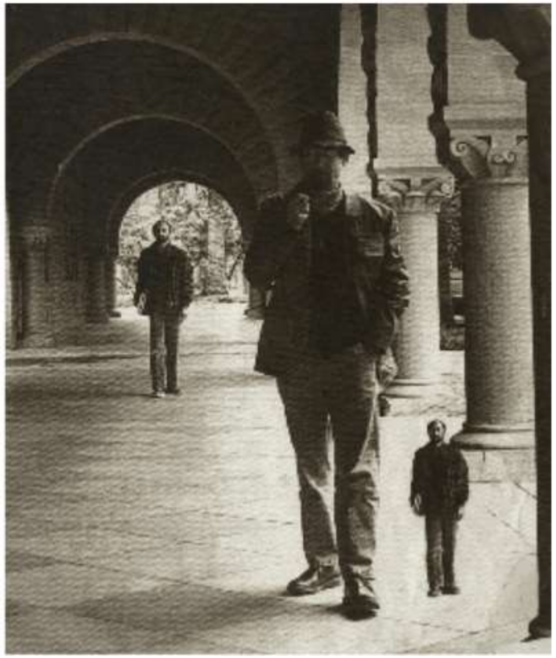
Ocular depth cues:
accommodation, vergence

Pictorial depth cues:
occlusion, size, shadows…



Cues sensitivity





"Perceiving layout and knowing distances: The integration, relative potency, and contextual use of different information about depth"
by Cutting and Vishton [1995]

Depth perception

We see depth due to depth cues.

Stereoscopic depth cues:
binocular disparity

Ocular depth cues:
accommodation, vergence

Pictorial depth cues:
occlusion, size, shadows…

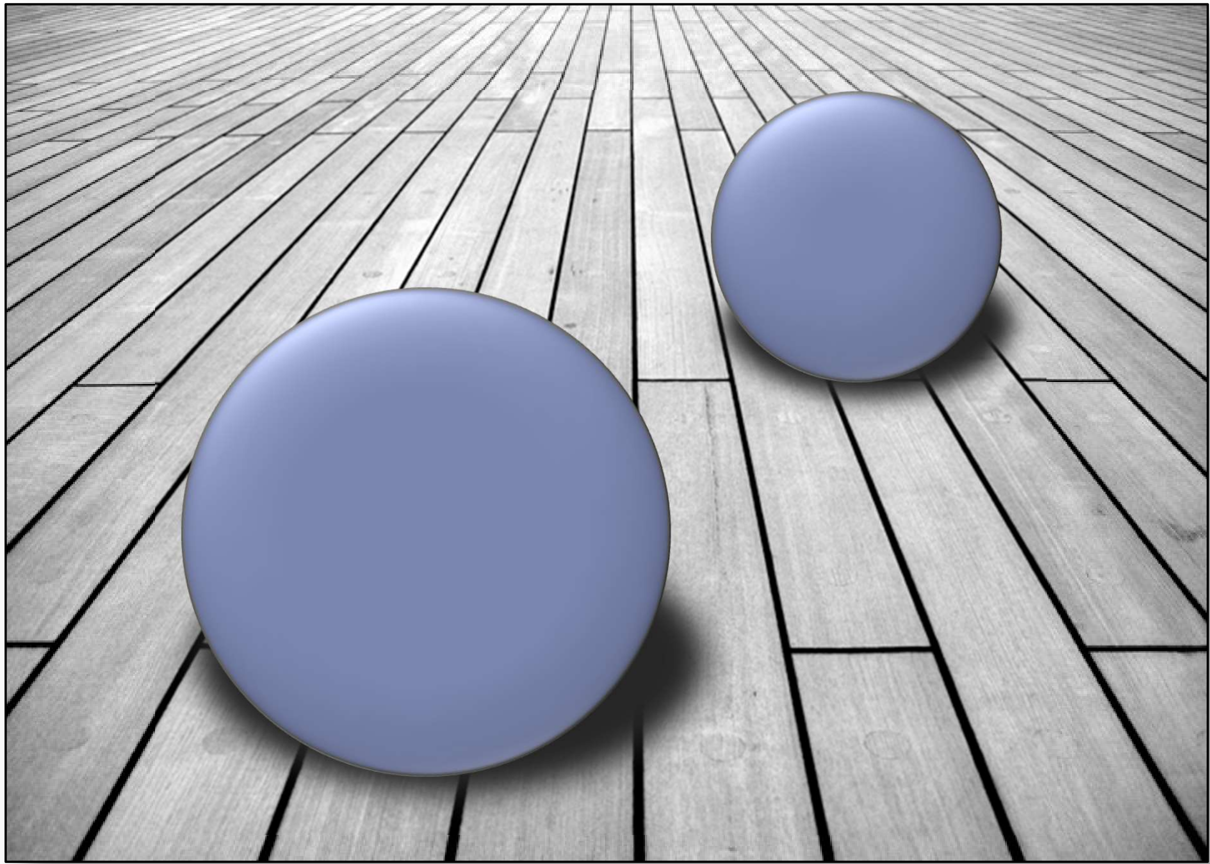
Challenge:
Consistency is required!

Simple conflict example

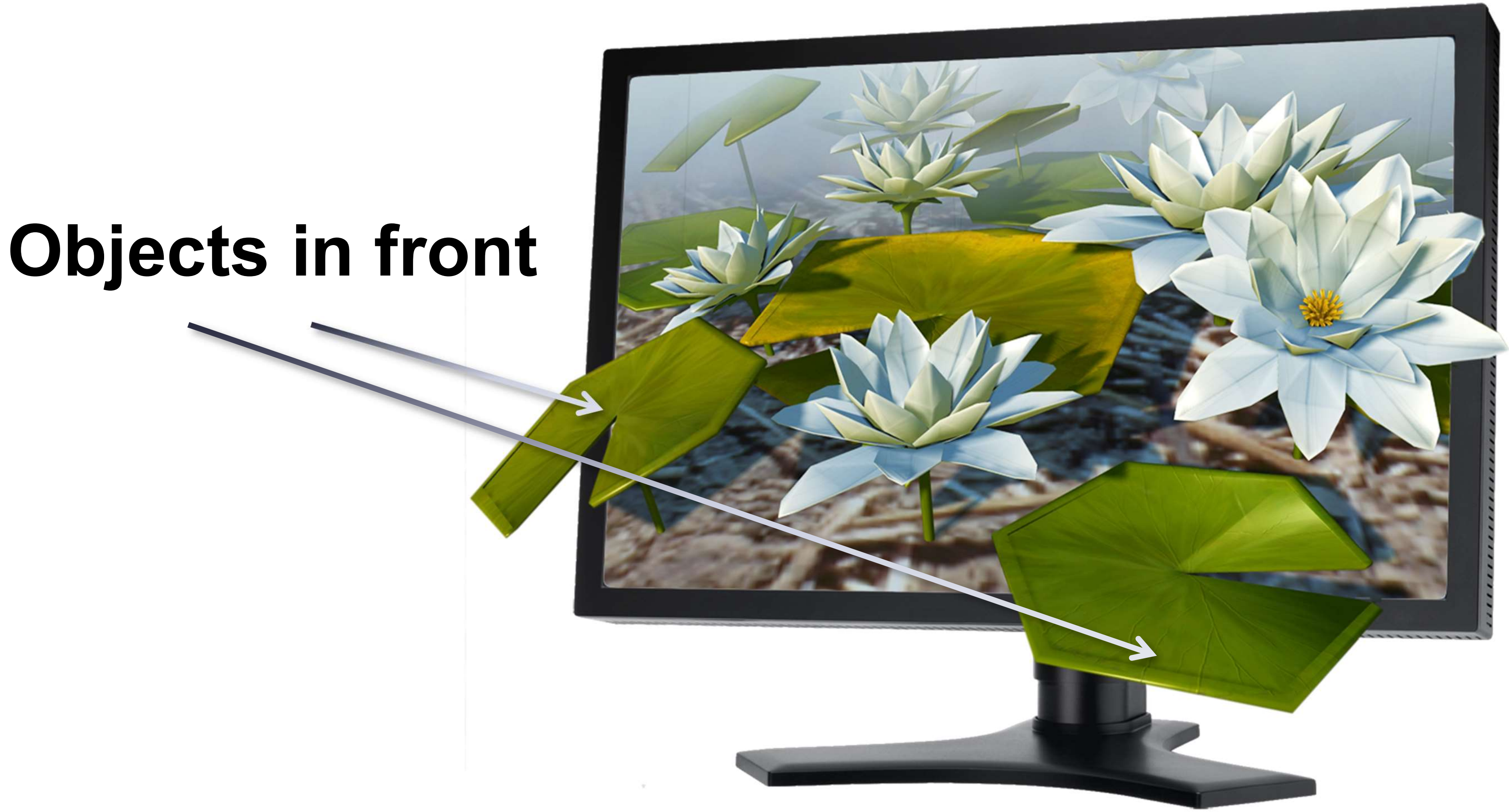
Present cues:

- Size
- Shadows
- Perspective

- **Occlusion**



Disparity & occlusion conflict



Disparity & occlusion conflict

Disparity & occlusion conflict

Depth perception

We see depth due to depth cues.

Stereoscopic depth cues:
binocular disparity

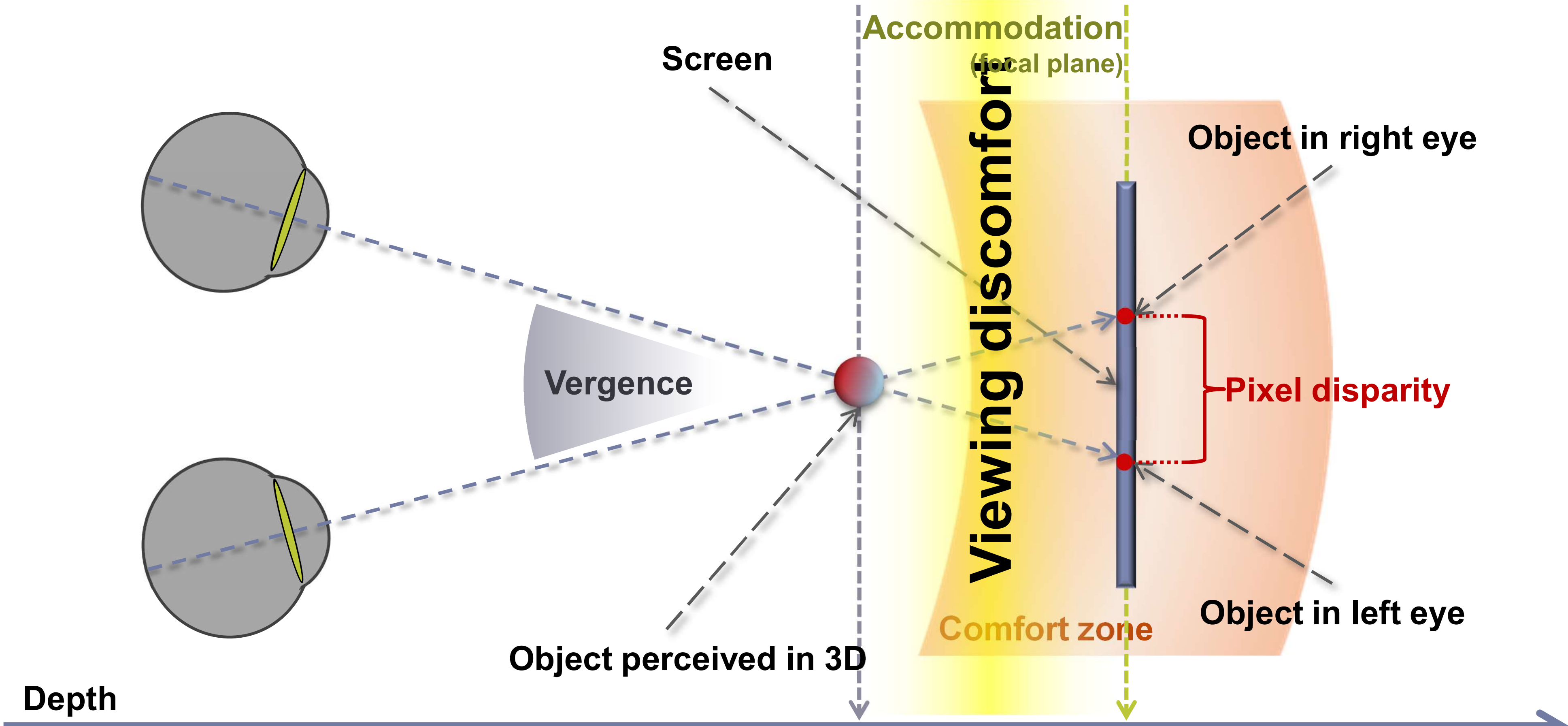
Ocular depth cues:
accommodation, vergence

→ **Require 3D space**
We cheat our Visual System!

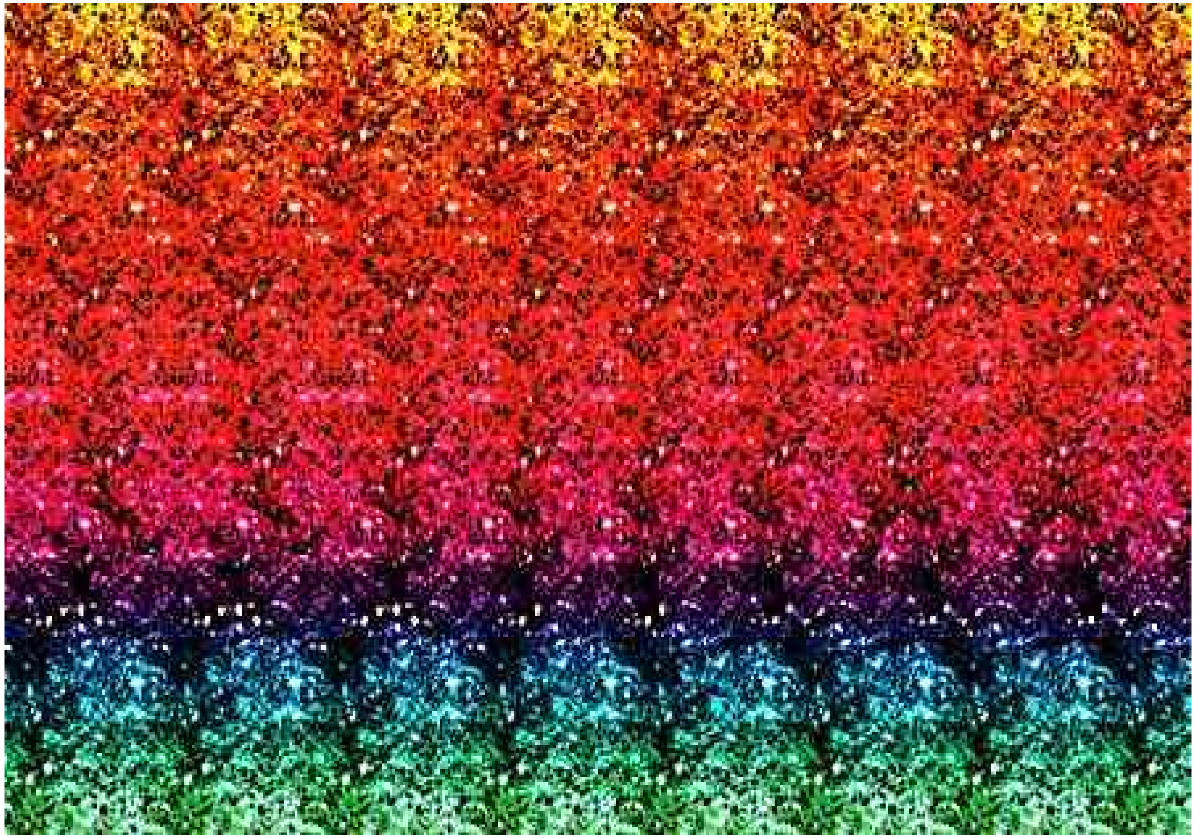
Pictorial depth cues:
occlusion, size, shadows…

→ **Reproducible on a flat displays**

Cheating our HVS

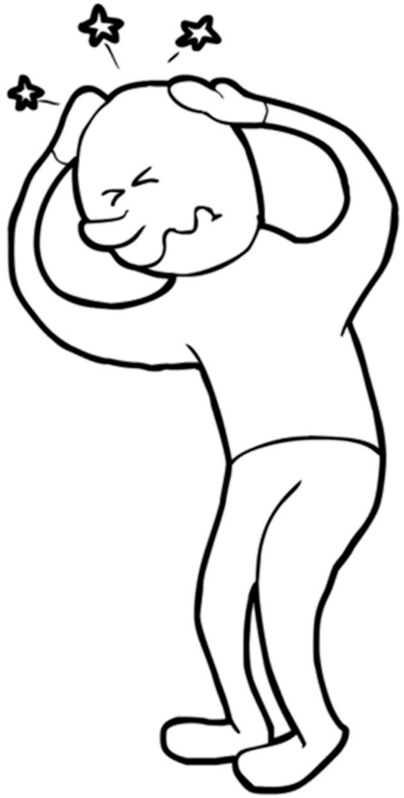


Single Image Random Dot Stereograms



- Fight the vergence vs. accommodation conflict to see the hidden image

Viewing discomfort

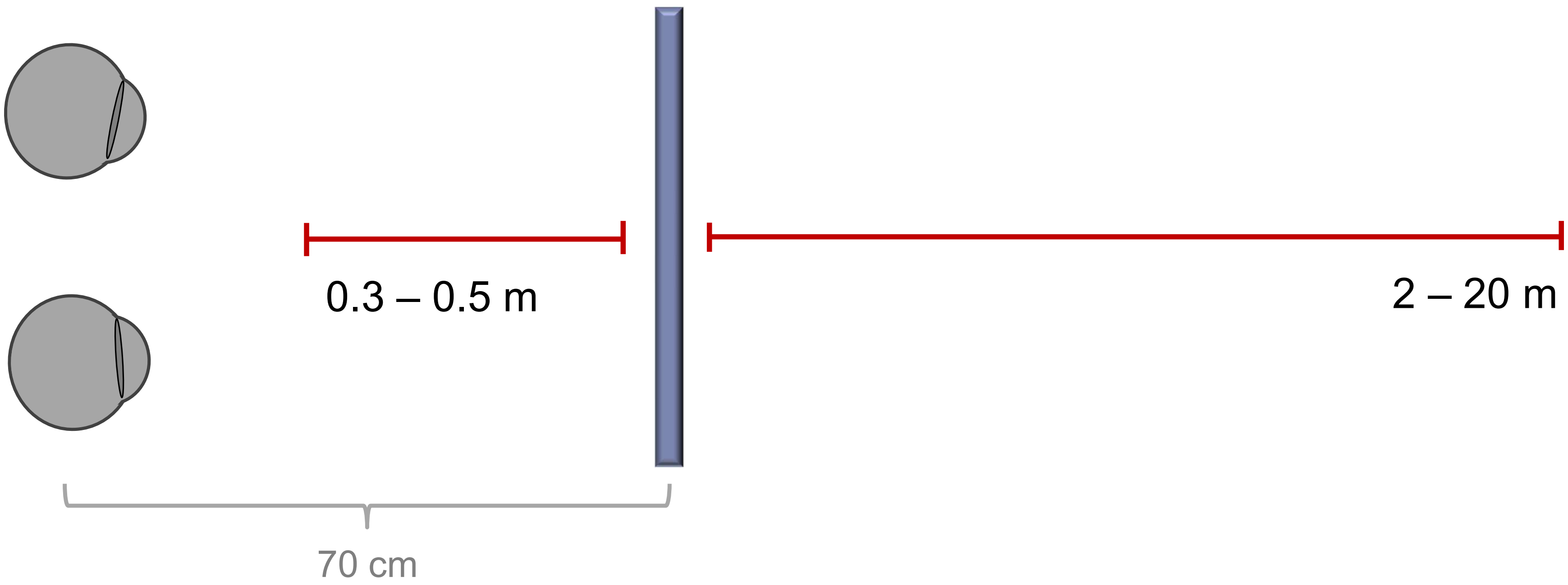


Comfort zones

Comfort zone size depends on:

- Presented content
- Viewing condition

Simple scene



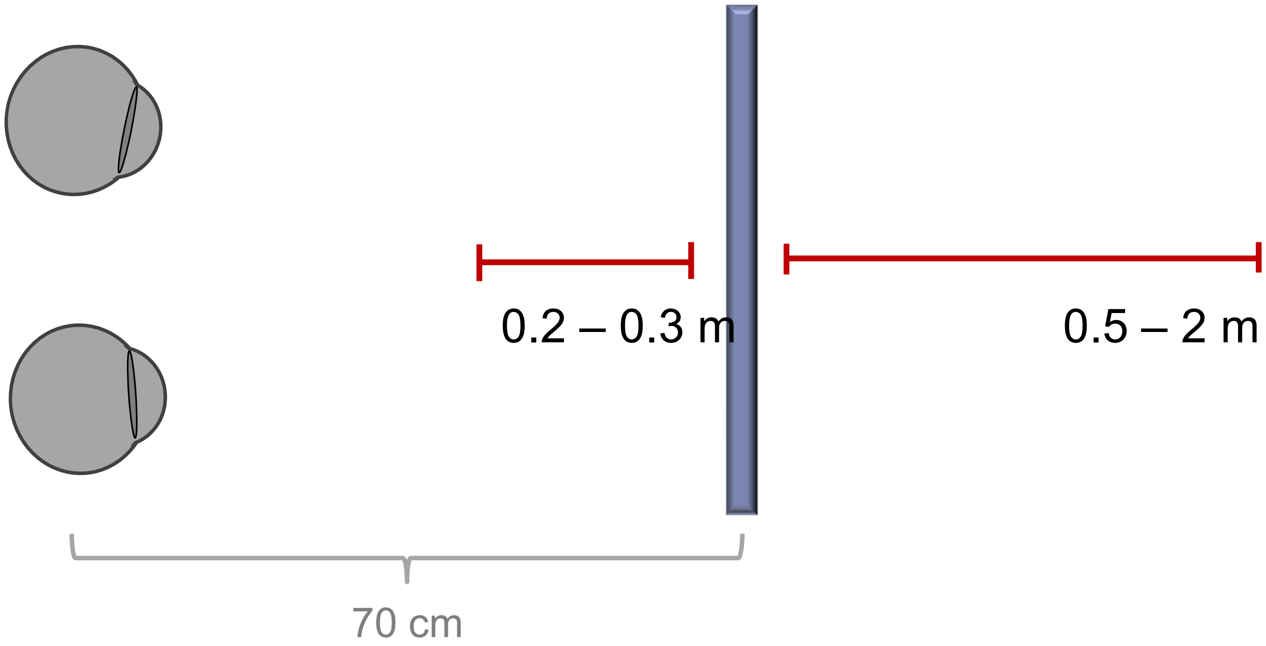
"Controlling Perceived Depth in Stereoscopic Images" by Jones et al. 2001

Comfort zones

Comfort zone size depends on:

- Presented content
- Viewing condition

Simple scene, user allowed to look away from screen



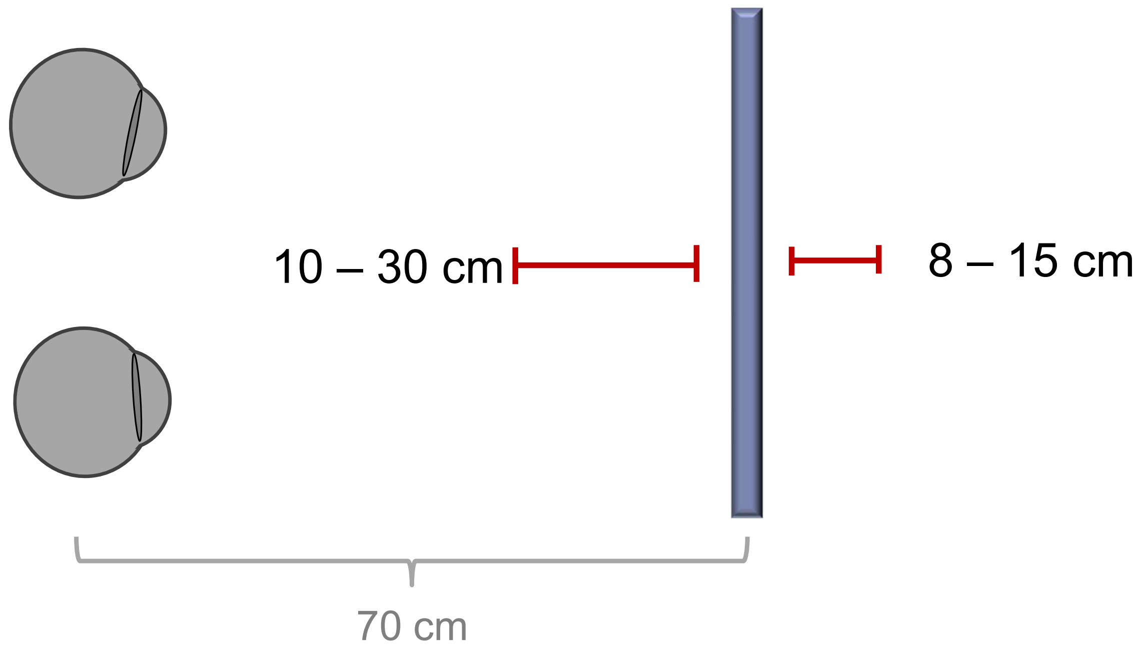
"Controlling Perceived Depth in Stereoscopic Images" by Jones et al. 2001

Comfort zones

Comfort zone size depends on:

- Presented content
- Viewing condition

Difficult scene



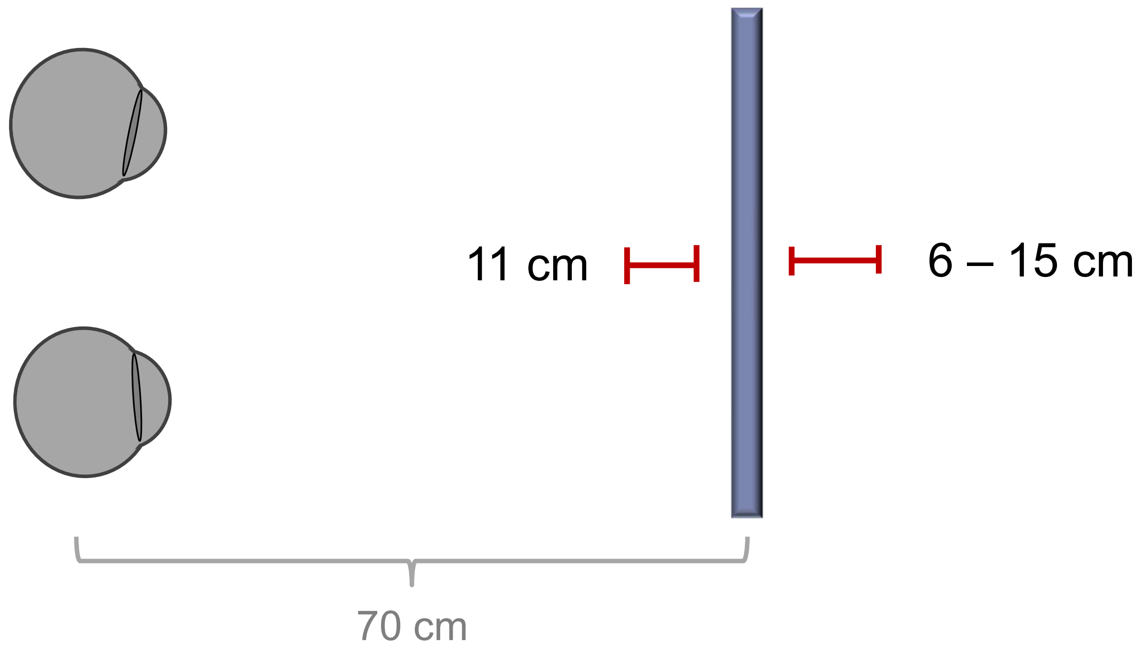
"Controlling Perceived Depth in Stereoscopic Images" by Jones et al. 2001

Comfort zones

Comfort zone size depends on:

- Presented content
- Viewing condition

Difficult scene, user allowed to look away from screen



"Controlling Perceived Depth in Stereoscopic Images" by Jones et al. 2001

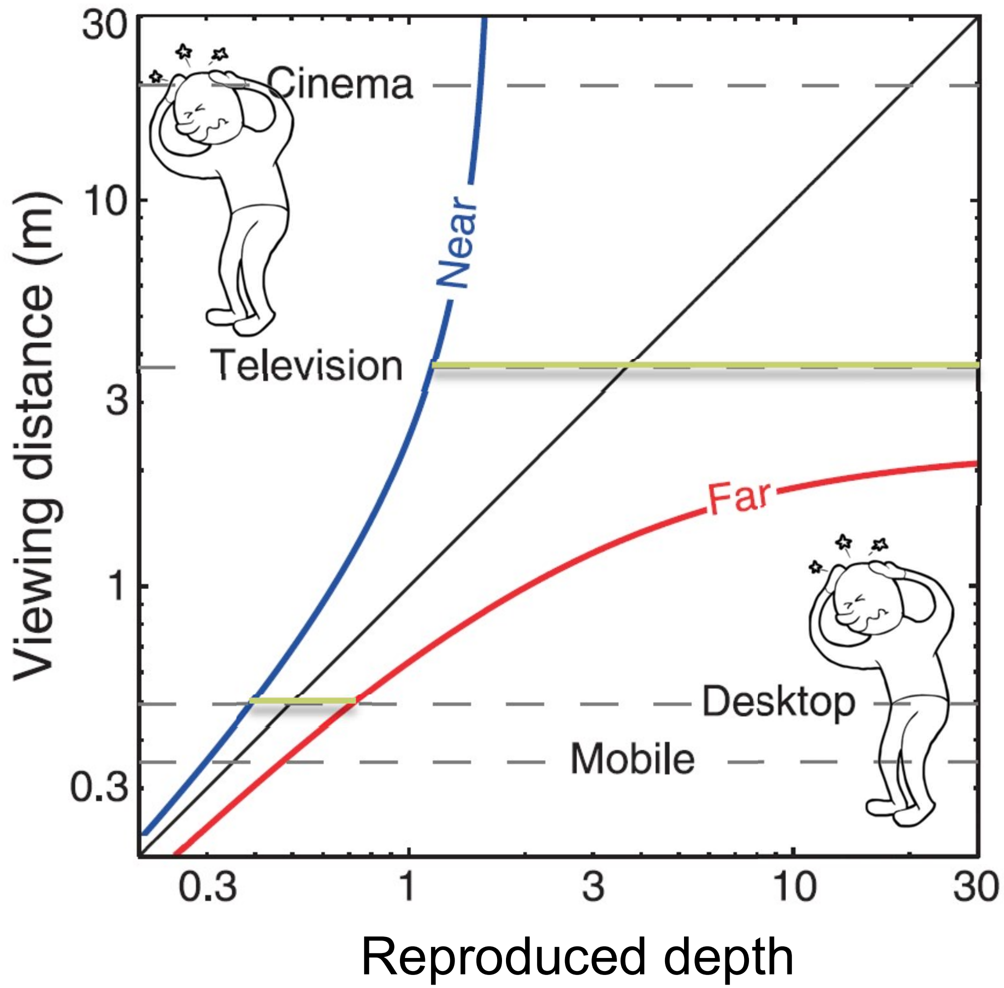
Comfort zones

Comfort zone size depends on:

- Presented content
- Viewing condition
- Screen distance

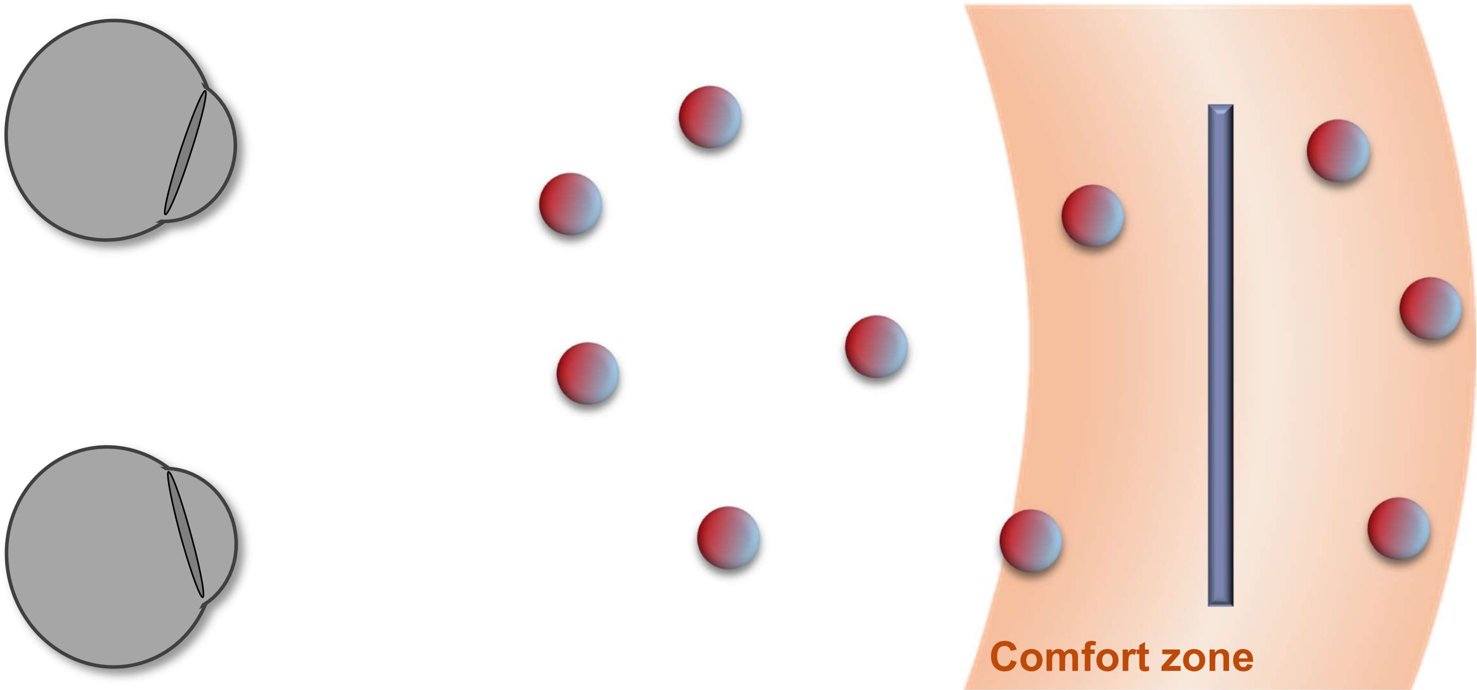
Other factors:

- Distance between eyes
- Depth of field
- Temporal coherence



"The zone of comfort: Predicting visual discomfort with stereo displays" by Shibata et al. 2011

Depth manipulation



Viewing discomfort $\xrightarrow{\text{Scene manipulation}}$ **Viewing comfort**

Advanced Graphics & Image Processing

Stereo Rendering

Part 2/3 – 3D display technologies

Rafał Mantiuk

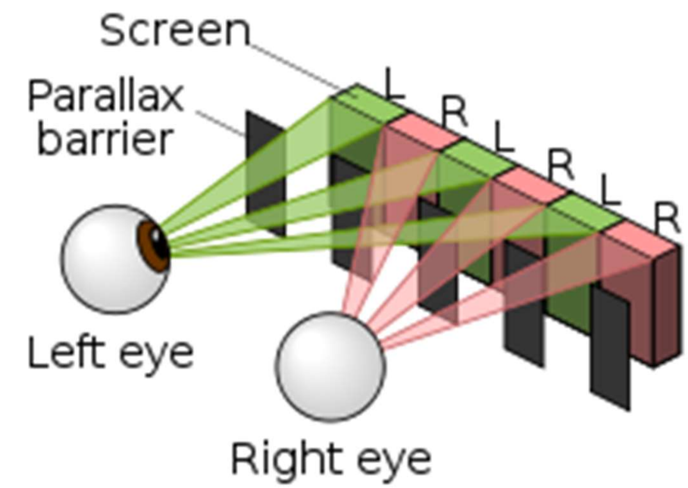
Dept. of Computer Science and Technology, University of Cambridge

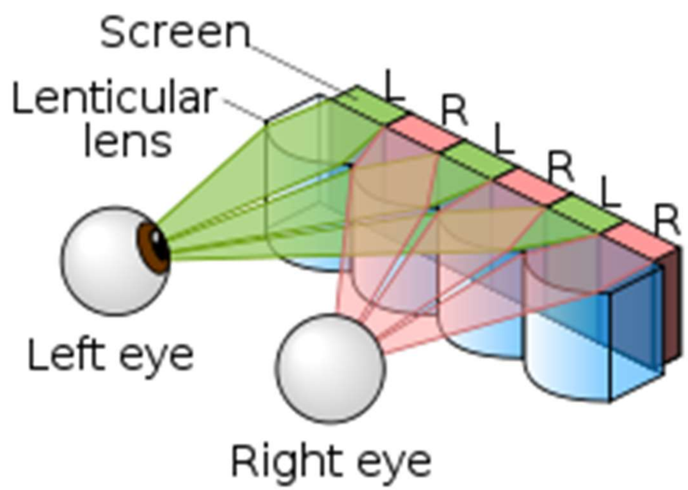
Stereoscopic displays

- Stereoscopic (with glasses)
  - Anaglyps (red & cyan glasses)
  - Shutter glasses: most TV sets
  - Circular polarization: RealD 3D cinema, 3D displays from LG
  - Interference filters: Dolby 3D cinema
- How do they work?
- Which method suffers from:
  - reduced brightness;
  - distorted colours;
  - cross-talk between the eyes;
  - cost (to manufacture)?

Stereoscopic displays

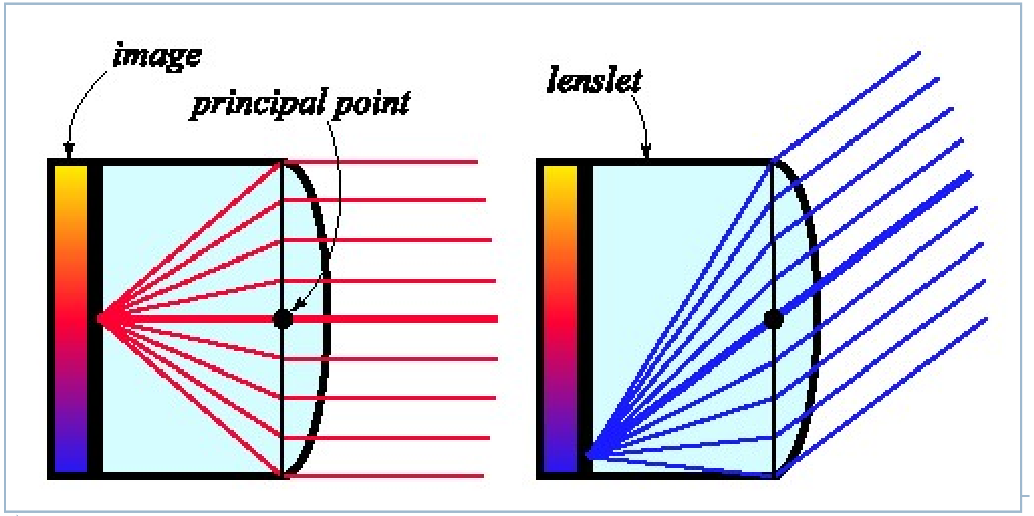
- Auto-stereoscopic (without glasses)
  - Parallax barrier
    - Example: Nintendo 3DS, some laptops and mobile phones
    - Switchable 2D/3D
  - Lenticular lens
    - Better efficiency
    - Non-switchable

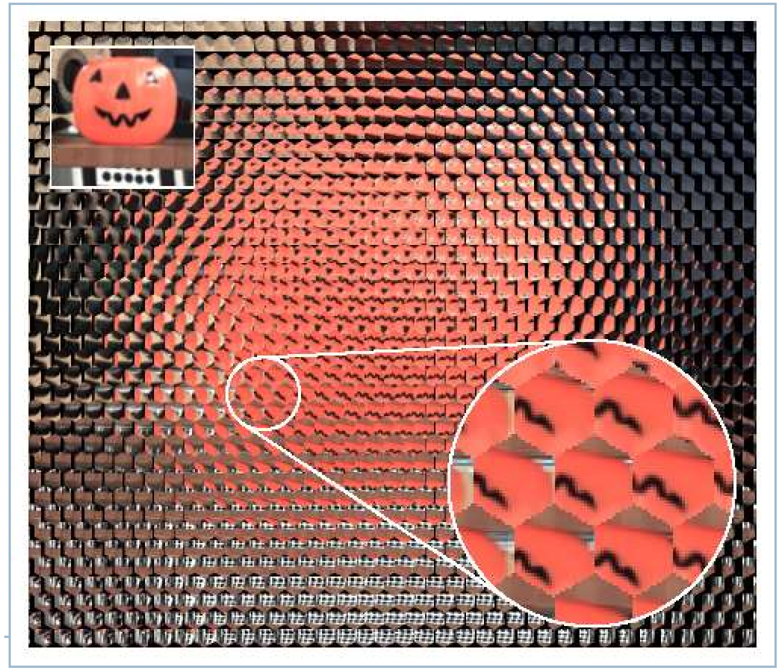




Light field Displays

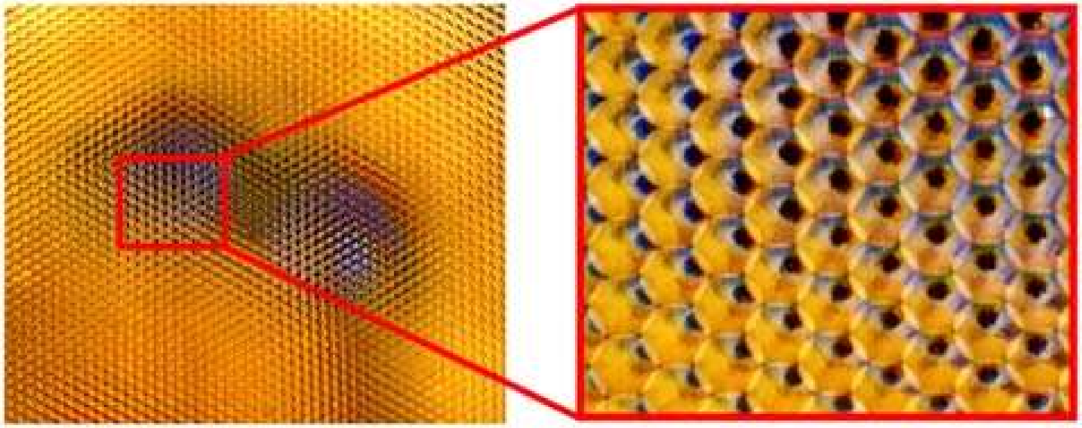
- integral photography, e. g. [Okano98]
- micro lens-array in front of screen
- screen at focal distance of micro lenses
  - → Parallel rays for each pixel
  - → Each eye sees a different pixel





Light field Displays

integral photograph close-up



one particular view

- need high resolution images
- taken with micro lens array
- screen is auto-stereoscopic

  → no glasses, multiple users

Advanced Graphics & Image Processing

Stereo Rendering

Part 3/3 – stereo rendering

Rafał Mantiuk
Dept. of Computer Science and Technology, University of Cambridge

Put on Your 3D Glasses Now!

pintrest.com

Anaglyph Stereo - Monochrome

- render L & R images, convert to grayscale
- merge into red-cyan anaglyph by assigning I(r)=L, I(g,b)=R (I is anaglyph)

from movie “Bick Buck Bunny”

Anaglyph Stereo – Full Color

- render L & R images, do not convert to grayscale
- merge into red-cyan anaglyph by assigning I(r)=L(r), I(g,b)=R(g,b) (I is anaglyph)


from movie “Bick Buck Bunny”

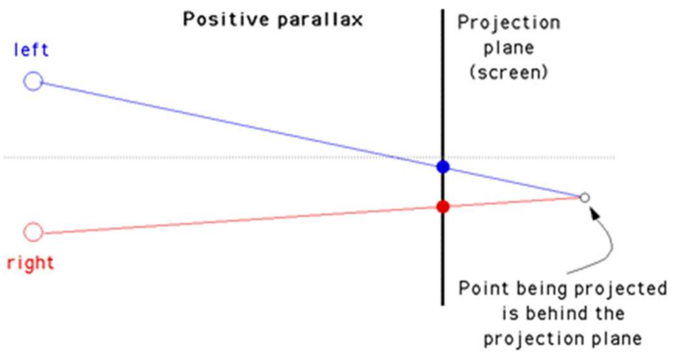
Open Source Movie: Big Buck Bunny

Rendered with Blender (Open Source 3D Modeling Program)

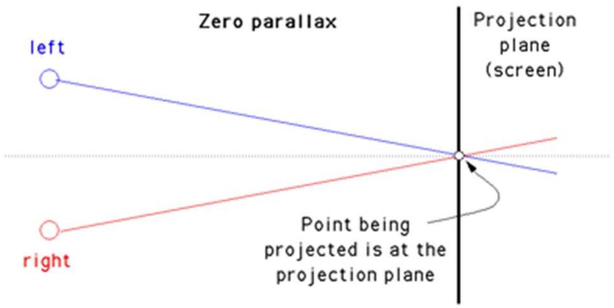
http://bbb3d.renderfarming.net/download.html

Parallax

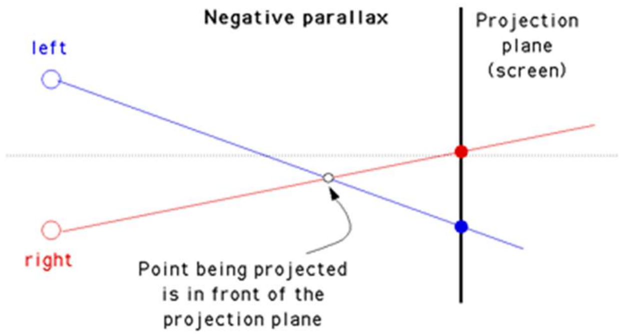
- Parallax is the relative distance of a 3D point projected into the 2 stereo images



case 1



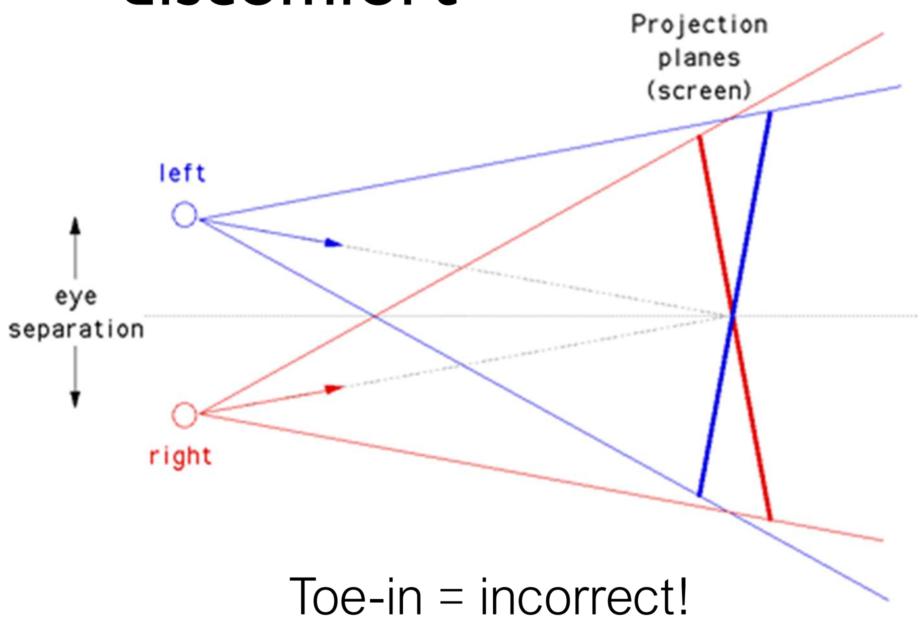
case 2



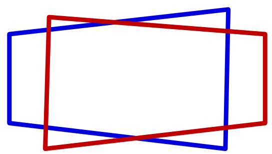
case 3

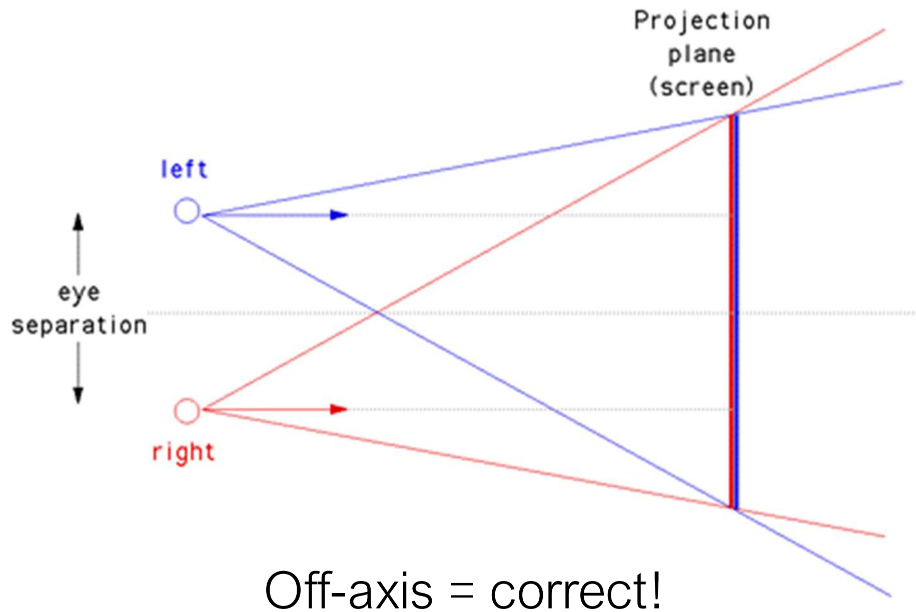
Parallax

- visual system only uses horizontal parallax, no vertical parallax!
- naïve toe-in method creates vertical parallax and visual discomfort



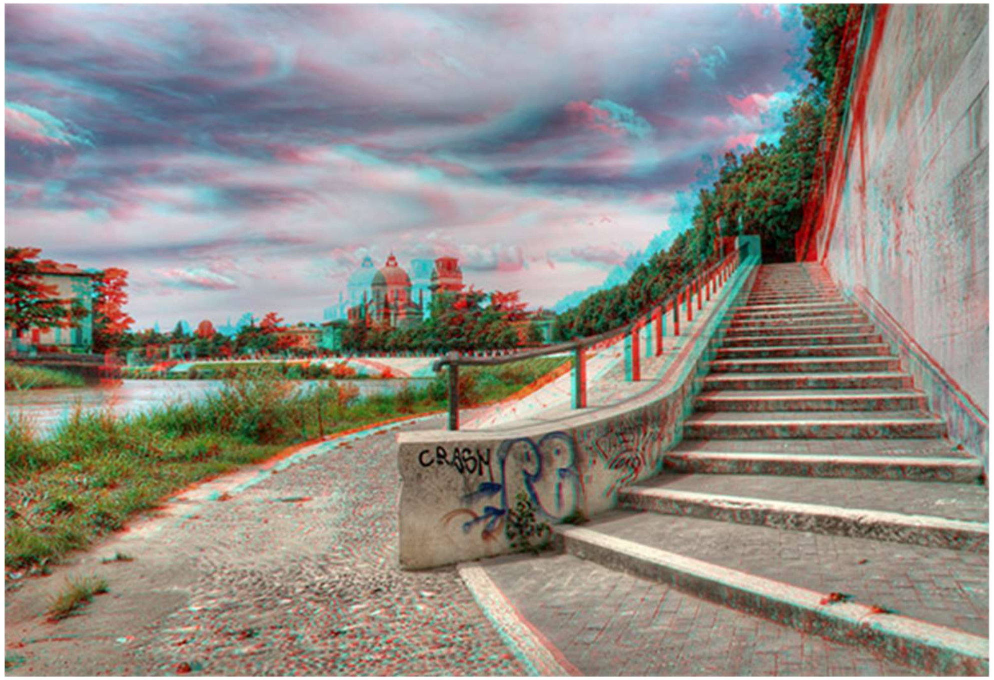
Toe-in = incorrect!



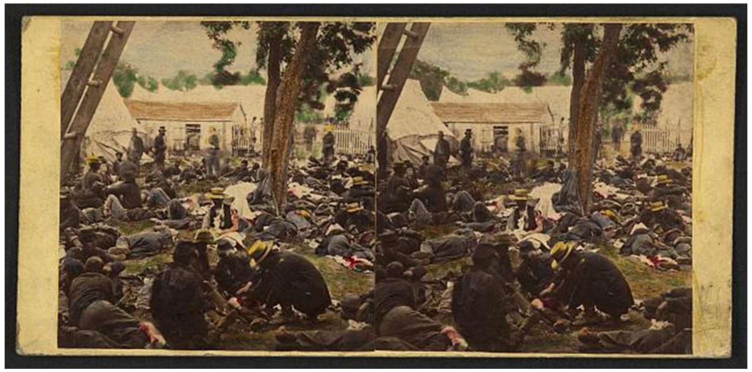


Off-axis = correct!

Parallax – well done



Parallax – well done



1862
“Tending wounded Union soldiers at Savage's Station, Virginia, during the Peninsular Campaign”,
Library of Congress Prints and Photographs Division

Parallax – not well done (vertical parallax = unnatural)

References

- LaValle "Virtual Reality", Cambridge University Press, 2016
  - Chapter 6
  - http://vr.cs.uiuc.edu/

Advanced Graphics and Image Processing - Lecture notes

Rafał Mantiuk

Lent term 2018/19

1 Contrast- and gradient-based methods

Many problems in image processing are easier to solve or produce better results if operations are not peformed directly on image pixel values but on differences between pixels. Instead of altering pixels, we can transform an image into gradient field and then edit the values in the gradient field. Once we are done with editing, we need to reconstruct an image from the modified gradient field.

A few examples of gradient-based methods are shown in Figures 1 and 2.

In one common case such differences between pixels represent gradients: for image I, a gradient at a pixel location (x, y) is computed as:

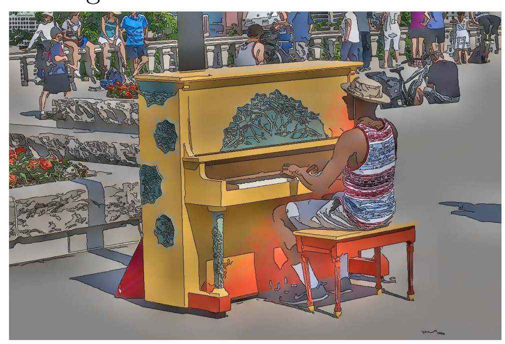
$$\nabla I_{x,y} = \begin{bmatrix} I_{x+1,y} - I_{x,y} \\ I_{x,y+1} - I_{x,y} \end{bmatrix}. \tag{1}$$

The equation above is obviously a discrete approximation of a gradient, as we are dealing with discrete pixel values rather than a continous function. This particular approximation is called forward difference, as we take the difference between the next and current pixel. Other choices include backward differences (current minus previous pixel) or central differences (next minus previous pixel).

Once a gradient field is computed, we can start modifying it. Usually better effects are achieved if the magnitude of gradients is modified and the orientation of each gradient remains unchanged. This can be achieved by

(a) Original image

(b) Details enhanced



(c) Cartoonized image

Figure 1: Two examples of gradient-based processing. Texture details in the original image were enhanced to produce the result shown in (b). Contrast was removed everywhere except at the edges to produced a cartoonized image in (c).

multiplying gradients by the gradient editing function $f()$:

$$G_{x,y} = \nabla I_{x,y} \cdot \frac{f\left(||\nabla I_{x,y}||\right)}{||\nabla I_{x,y}||} \tag{2}$$

where $||\cdot||$ operator computes the magnitude (norm) of the gradient.

We try to reconstruct pixel values, which would result in a gradient field that is the closest to our modifed gradient field $G = [G^{(x)} \quad G^{(y)}]'$. In particular, we can try to minimize the squared differences between gradients in actual image and modified gradients:

$$\arg\min_{I} \sum_{x,y} \left[\left(I_{x+1,y} - I_{x,y} - G^{(x)}_{x,y}\right)^2 + \left(I_{x,y+1} - I_{x,y} - G^{(y)}_{x,y}\right)^2 \right] , \tag{3}$$

(a) Naive image copy & paste (b) Gradient-domain copy & paste

Figure 2: Comparison of naive and gradient domain image copy & paste.

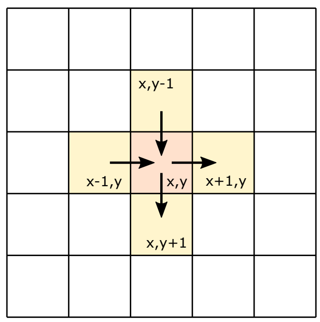


Figure 3: When using forward-differences, a pixel with the coordinates (x, y) is referred to in at moost four partial derivates, two along x-axis and two along y-axis.

where the summation is over the entire image. To minimize the function above, we need to equate its partial derivatives to 0. As we optimze for pixel values, we need to compute partial derivates with respect to $I_{x,y}$. Fortunately, most terms in the sum will become 0 after differentiation, as they do not contain the differentiated variable $I_{x,y}$. For a given pixel (x, y), we need to consider only 4 partial derivates: two belonging to the pixel (x, y), x-derivative for the pixel on the left $(x - 1, y)$ and y-derivative for the pixel in the top $(x, y - 1)$, as shown in Figure 3. This gives us:

$$\frac{\delta F}{\delta I_{x,y}} = -2(I_{x+1,y} - I_{x,y} - G^{(x)}_{x,y}) - 2(I_{x,y+1} - I_{x,y} - G^{(y)}_{x,y}) + \tag{4}$$

$$2(I_{x,y} - I_{x-1,y} - G^{(x)}_{x-1,y}) + 2(I_{x,y} - I_{x,y-1} - G^{(y)}_{x,y-1})\,. \tag{5}$$

After rearanging the terms and equating $\frac{\delta F}{\delta I_{x,y}}$ to 0, we get:

$$I_{x-1,y} + I_{x+1,y} + I_{x,y-1} + I_{x,y+1} - 4I_{x,y} = G^{(x)}_{x,y} - G^{(x)}_{x-1,y} + G^{(y)}_{x,y} - G^{(y)}_{x,y-1} \,. \quad (6)$$

In these few steps we derived a discrete Poisson equation, which can be found in many engineering problems. The Poisson equation is often written as:

$$\nabla^2 I = \mathrm{div} G \,, \quad (7)$$

where $\nabla^2 I$ is the discrete Laplace operator:

$$\nabla^2 I_{x,y} = I_{x-1,y} + I_{x+1,y} + I_{x,y-1} + I_{x,y+1} - 4I_{x,y} \,, \quad (8)$$

and $\mathrm{div} G$ is the divergence of the vector field:

$$\mathrm{div} G_{x,y} = G^{(x)}_{x,y} - G^{(x)}_{x-1,y} + G^{(y)}_{x,y} - G^{(y)}_{x,y-1} \,. \quad (9)$$

We can also write the equation using discrete convolution operators:

$$I * \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = G^{(x)} * \begin{bmatrix} -1 & 1 & 0 \end{bmatrix} + G^{(y)} * \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix} . \quad (10)$$

Note that the covolution flips the order of elements in the kernel, thus the row and column vectors on the right hand side are also flipped.

When equation 6 is satisfied for every pixel, it forms a system of linear equations:

$$A \cdot \begin{bmatrix} I_{1,1} \\ I_{2,1} \\ \cdots \\ I_{N,M} \end{bmatrix} = b \quad (11)$$

Here we represent an image as a very large column vector, in which image pixels are stacked column-after-column (in an equivalent manner they can be stacked row-after-row). Every row of matrix A contains the Laplace operator for a corresponding pixel. But the matrix also needs to account for the boundary conditions, that is handle pixels that are at the image edge and therefore do not contain neighbour on one of the sides. Matrix A for a tiny

3x3 image looks like this:

$$A = \begin{bmatrix} -2 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -3 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -3 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -3 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -3 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -2 \end{bmatrix} \tag{12}$$

Obviously, the matrix is enormous for normal size images. However, most matrix elements are 0, so it can be easily stored using a sparse matrix representation. Note that only the pixel in the center of the image (5th row) contains the full Laplace operator; all other pixels are missing neighbours so the operator is adjusted accordingly. Accounting for all boundary cases is probably the most difficult and error-prone part in formulating gradient-field reconstruction problem. The column vector b corresponds to the right hand side of equation 6.

2 Solving linear system

There is a large number of methods and software libraries, which can solve a sparse linear problem given in Equation 11. The Poisson equation is typically solved using multi-grid methods, which iteratively update the solution at different scales. Those, however, are rarther difficult to implement and tailored to one particular shape of a matrix. Alternatively, the solution can be readily found after transformation to the frequency domain (discrete cosine transform). However, such a method does not allow introducing weights, importance of which will be discussed in the next section. Finally, conjugate gradient and biconjugate gradient [1, sec. 2.7] methods provide a fast-converging iterative method for solving sparse systems, which can be very memory efficient. Those methods require providing only a way to compute multiplication of the matrix A and its transpose with an arbitrary vector. Such operation can be realized in an arbitrary way without the need to store the sparse matrix (which can be very large even if it is sparse). The conjugate gradient requires fewer operations than the biconjugate gradient method, but

(a) Uniform weights (b) Higher weights at low contrast

Figure 4: The solution of gradient field reconstruction often contain "pinching" artefacts, such as shown in figure (a). The artefacts can be avoided if small gradient magnitudes are weighted more than large magnitudes.

it should be used only with positive definite matrices. Matrix A is not positive definite so in principle the biconjugate gradient method should be used. However, in practice, conjugate gradient method converges equally well.

3 Weighted reconstruction

An image resulting from solving Equation 11 often contains undesirable "pinching" artefacts, such as those shown in Figure 4a. Those artefacts are inherent to the nature of gradient field reconstruction — the solution is just the best approximation of the desired gradient field but it hardly ever exactly matches the desired gradient field. As we minimize squared differences, tiny inaccuracies for many pixels introduce less error than large inaccuracies for few pixels. This in turn introduces smooth gradients in the areas, where the desired gradient field is inconsistent (cannot form an image). Such gradients produce "pinching" artefacts.

The problem is that the error in reconstructed gradients is penalized the same regardless of whether the value of the gradient is small or large. This is opposite to how the visual system perceives differences in color values: we are more likely to spot tiny difference between two similar pixel values than the same tiny difference between two very different pixel values. We could account for that effect by introducing some form of non-linear metric, however, that would make our problem non-linear and non-linear problems are in general much slower to solve. However, the same can be achieved by introducing weights to our objective function:

$$\arg\min_{I} \sum_{x,y} \left[w^{(x)}_{x,y} \left(I_{x+1,y} - I_{x,y} - G^{(x)}_{x,y} \right)^2 + w^{(y)}_{x,y} \left(I_{x,y+1} - I_{x,y} - G^{(y)}_{x,y} \right)^2 \right] , \tag{13}$$

where $w^{(x)}_{x,y}$ and $w^{(y)}_{x,y}$ are the weights or importance we assign to each gradient, for horizontal and vertical partial derivatives respectively. Usually the weights are kept the same for both orientations, i.e. $w^{(x)}_{x,y} = w^{(y)}_{x,y}$. To account for the contrast perception of the visual system, we need to assign a higher weight to small gradient magnitudes. For example, we could use the weight:

$$w^{(x)}_{x,y} = w^{(y)}_{x,y} = \frac{1}{||G_{x,y}|| + \epsilon} \tag{14}$$

where $||G_{x,y}||$ is the magnitude of the desired (target) gradient at pixel (x, y) and ϵ is a small constant (0.0001), which prevents division by 0.

4 Matrix notation

We could follow the same procedure as in the previous section and differentiate Equation 13 to find the linear system that minimizes our objective. However, the process starts to be tedious and error-prone. As the objective functions gets more and more complex, it is worth switching to the matrix notation. Let us consider first our original problem without the weights $w_{x,y}$, which we will add later. Equation 3 in the matrix notation can be written as:

$$\arg\min_{I} \left|\left| \begin{bmatrix} \nabla_x \\ \nabla_y \end{bmatrix} I - \begin{bmatrix} G^{(x)} \\ G^{(y)} \end{bmatrix} \right|\right|^2 . \tag{15}$$

In the equation I, $G^{(x)}$ and $G^{(y)}$ are stacked column vectors, representing columns of the resulting image or desired gradient field. The square brackets denote vertical concatenation of the matrices or vectors. Operator $||\cdot||^2$ is the L_2-norm, which squares and sums the elements of the resulting column vector. ∇_x and ∇_y are differential operators, which are represented as $N \times N$ matrices, where N is the number of pixels. Each row of those sparse matrices tells us which pixels need to be subtracted from one another to compute forward gradients along horizontal and vertical directions. For a tiny 3×3 pixel image those operators are:

$$\nabla_x = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{16}$$

$$\nabla_y = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{17}$$

Note that the rows contain all zeros for pixels on the boundary, for which no gradient can be computed: the last column of pixels for ∇_x and the last row of pixels for ∇_y.

Equation 15 is in the format $||Ax - b||^2$, which can be directly solved by some sparse matrix libraries, such as `SciPy.sparse` or the "\" operator in matlab Matlab. However, to reduce the size of the sparse matrix and to speed-up computation, it is worth taking one more step and transform the least-square optimization into a linear problem. For overdetermined systems, such as ours, the solution of the optimization problem:

$$\arg\min_x ||Ax - b||^2 \tag{18}$$

can be found by solving a linear system:

$$A'Ax = A'b\,. \tag{19}$$

Note that $'$ denotes a matrix transpose and $A'A$ is a square matrix. If we replace A and b with the corresponding operators and gradient values from our problem, we get the following linear system:

$$\begin{bmatrix}\nabla'_x & \nabla'_y\end{bmatrix} \begin{bmatrix}\nabla_x \\ \nabla_y\end{bmatrix} I = \begin{bmatrix}\nabla'_x & \nabla'_y\end{bmatrix} \begin{bmatrix}G^{(x)} \\ G^{(y)}\end{bmatrix}\,, \tag{20}$$

which, after multiplying stacked matrices, gives us:

$$\left(\nabla'_x\,\nabla_x + \nabla'_y\,\nabla_y\right)\, I = \nabla'_x\,G^{(x)} + \nabla'_y\,G^{(y)}\,. \tag{21}$$

Weights can be added to such a system by inserting a sparse diagonal matrix W. For simplicity we use the same weights for vertical and horizontal derivatives:

$$\left(\nabla'_x\,W\,\nabla_x + \nabla'_y\,W\,\nabla_y\right)\, I = \nabla'_x\,W\,G^{(x)} + \nabla'_y\,W\,G^{(y)}\,. \tag{22}$$

The above operations can be performed using a sparse matrix library (or SciPy/Matlab), thus saving us effort of computing operators by hand.

There is still one problem remaining: our equation does not have a unique solution. This is because the target gradient field contains relative information about differences between pixels, but it does not say what the absolute value of the pixels should be. For that reason, we need to constrain the absolute value, for example by ensuring that a value of a first reconstructed pixel is the same as in the source image (I_{src}):

$$\begin{bmatrix}1 & 0 & \ldots & 0\end{bmatrix}\, I = I_{src}(1,1)\,. \tag{23}$$

If we denote the vector on the left-hand side of the equation as C, the final linear problem can be written as:

$$\left(\nabla'_x\,W\,\nabla_x + \nabla'_y\,W\,\nabla_y + C'\,C\right)\, I = \nabla'_x\,W\,G^{(x)} + \nabla'_y\,W\,G^{(y)} + C'\,I_{src}(1,1)\,. \tag{24}$$

The resulting equation can be solved using a sparse solver in Python or Matlab.

References

[1] S. A. Teukolsky, B. P. Flannery, W. H. Press, and W. T. Vetterling. *Numerical recipes in C*. Cambridge University Press, Cambridge, vol. 2 edition, 1992.

Advanced Graphics and Image Processing - Lecture notes

Rafał Mantiuk

Lent term 2018/19

1 Light field rendering using homographic transformation

This section explains how to render a light field for a novel view position using a parametrization with a focal plane. The method is explained on a rather high level in [1]. These notes are meant to provide a practical guide on how to perform the required calculations and in particular how to find a homographic transformation between the virtual and array cameras.

The scenario and selected symbols are illustrated in Figure 1. We want to render our light field "as seen" by camera K. We have N images captured by N cameras in the array (only 4 shown in the figure), all of which have their apertures on the camera array plane C. We further assume that our array cameras are pin-hole cameras to simplify the explanation. The novel view is rendered assuming focal plane F. The focal plane has a similar function as the focus distance in a regular camera: objects on the focal plane will be rendered sharp, while objects that and in front or behind that plane will appear blurry (in practice they will appear ghosted because of the limited number of cameras). The focal plane F does not need to be parallel to the camera plane; it can be titled, unlike in a traditional camera with a regular lens. Because we have a limited number of cameras, we need to use reconstruction functions A_0, ..., A_1 (only two shown) for each camera. The functions shown contain the weights in the range 0-1 that are used to interpolate between two neighboring views.

To intuitively understand how light field rendering is performed, imagine the following hypothetical scenario. Each camera in the array captures the

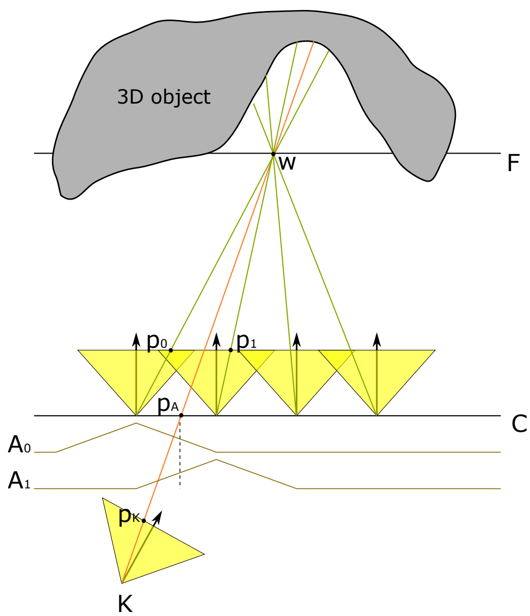


Figure 1: Light field rendering for the novel view represented by camera K. The pixels P_K in the rendered image is the weighted average of the pixels values p_1, ..., p_N from the images captured by the camera array.

image of the scene. Then, all objects in the scene are removed and you put a large projection screen where the focal plane F should be. Then, you swap all cameras for projectors, which project the captured images on the projection screen F. Finally, you put a new camera K at the desired location and capture the image of the projection screen. The projection screen (focal plane) is needed to form an image. Ideally, to obtain a sharp image, we would like to project the camera array images on a geometry. However, such a geometry is not readily available when capturing scenes with a camera array. In this situation a single plane is often a good-enough proxy, which has its analogy in physical cameras (focal distance). More advanced light field rendering methods attempt to reconstruct a more accurate proxy geometry using multi-view stereo algorithms and then project camera images on that geometry [3].

This simplified scenario misses one step, which is modulating each projected image by the reconstruction function A, as such modulation has no physical counterpart. However, this scenario should give you a good idea what operations need to be performed in order to render a light field from a

Data: Camera array images J_1, J_2, ..., J_N
Result: Rendered image I
for *each pixel at the coordinates* $\boldsymbol{p}_K$ *in the novel view* **do**
  $I(\boldsymbol{p}_K) \leftarrow 0$;
  $w(\boldsymbol{p}_K) \leftarrow 0$;
  for *each camera* i *in the array* **do**
    Find the coordinates $\boldsymbol{p}_i$ in the i-th camera image corresponding to the pixel $\boldsymbol{p}_K$;
    Find the coordinates $\boldsymbol{p}_A$ on the aperture plane A corresponding to the pixel $\boldsymbol{p}_K$;
    $I(\boldsymbol{p}_K) \leftarrow I(\boldsymbol{p}_K) + A(\boldsymbol{p}_A)\, J_i(\boldsymbol{p}_i)$;
    $W(\boldsymbol{p}_K) \leftarrow W(\boldsymbol{p}_K) + A(\boldsymbol{p}_A)$;
  end
  $I(\boldsymbol{p}_K) \leftarrow I(\boldsymbol{p}_K)/W(\boldsymbol{p}_K)$;
end

Algorithm 1: Light field rendering algorithm

novel view position.

Now let us see how we can turn such a high-level explanation into a practical algorithm. One way to render a light field is shown in Algorithm 1. The algorithm iterates over all pixels in the rendered image, then for each pixel it iterates over all cameras in the array. The resulting image is the weighted average of the camera images that are warped using homographic transformations. The weights are determined by the reconstruction functions A_i. The algorithm is straightforward, except for the mapping from pixel coordinates in the novel view p_K to coordinates in each camera image p_i and the coordinates on the aperture plane p_A. The following paragraphs explain how to find such transformations.

1.1 Homographic transformation between the virtual and array cameras

The text below assumes that you are familiar with homogeneous coordinates and geometric transformations, both commonly used in computer graphics and computer vision. If these topics are still unclear, refer to Section 2.1 in [4] (this book is available online) or Chapter 6 in [2].

We assume that we know the position and pose of each camera in the array, so that homogeneous 3D coordinates of a point in the 3D word coordinate space w can be mapped to the 2D pixel coordinates p_i of camera i:

$$\boldsymbol{p}_i = \boldsymbol{K}\,\boldsymbol{P}\,\boldsymbol{V}_i\,\boldsymbol{w}\,. \tag{1}$$

where $\boldsymbol{V}$ is the view transformation, $\boldsymbol{P}$ is the projection matrix and $\boldsymbol{K}$ is the intrinsic camera matrix. Note that we will use bold lower case symbols to denote vectors, uppercase bold symbols for matrices and a regular font for scalars. The operation is easier to understand if the coordinates and matrices are expanded:

$$\begin{bmatrix} x_i \\ y_i \\ w_i \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} \\ v_{21} & v_{22} & v_{23} & v_{24} \\ v_{31} & v_{32} & v_{33} & v_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{2}$$

The view matrix $\boldsymbol{V}$ translates and rotates the 3D coordinates of the 3D point $\boldsymbol{w}$ so that the origin of the new coordinate system is at the camera centre, and camera's optical axis is aligned with the z-axis (as the view matrix in computer graphics). This matrix can be computed using a *LookAt* function, often available in graphics matrix libraries.

The projection matrix $\boldsymbol{P}$ may look like an odd version of an identity matrix, but it actually drops one dimension (projects from 3D to 2D) and copies the value of Z coordinate into the additional homogeneous coordinate w_i. Note that to compute Cartesian coordinates of the point from the homogeneous coordinates, we divide x_i/w_i and y_i/w_i. As w_i is now equal to the depth in the camera coordinates, this operation is equivalent to a perspective projection (you can refer to slides 88–92 in the Introduction to Graphics Course).

The intrinsic camera matrix $\boldsymbol{K}$ maps the projected 3D coordinates into pixel coordinates. f_x and f_y are focal lengths and c_x and c_y are the coordinates of optical center expressed in pixel coordinates. We assume that the intrinsic matrix is the same for all the cameras in the array.

Besides having all matrices for all cameras in the array, we also have a similar transformation for our virtual camera K, which represents the currently rendered view:

$$\boldsymbol{p}_K = \boldsymbol{K}_K\,\boldsymbol{P}\,\boldsymbol{V}_K\,\boldsymbol{w}\,. \tag{3}$$

Our first task is to find transformation matrices that could transform from pixel coordinates $\boldsymbol{p}_K$ in the virtual camera image into pixel coordinates $\boldsymbol{p}_i$

for each camera i. This is normally achieved by inverting the transformation matrix for the novel view and combining it with the camera array transformation. However, the problem is that the product of $\boldsymbol{K}_K \boldsymbol{P} \boldsymbol{V}_K$ is not a square matrix that can be inverted — it is missing one dimension. The dimension is missing because we are projecting from 3D to 2D and one dimension (depth) is lost.

Therefore, to map both coordinates, we need to reintroduce missing information. This is achieved by assuming that the 3D point lies on the focal plane F. Note that the plane equation can be expressed as $\boldsymbol{N} \cdot (\boldsymbol{w} - \boldsymbol{w}_F) = 0$, where $\boldsymbol{N}$ is the plane normal, and $\boldsymbol{w}_F$ specifies the position of the plane in the 3D space. Operator $\cdot$ is the dot product. If the homogeneous coordinates of the point $\boldsymbol{w}$ are $\begin{bmatrix} X & Y & Z & 1 \end{bmatrix}$, the plane equation can be expressed as

$$d = \begin{bmatrix} n_x & n_y & n_z & -\boldsymbol{N} \cdot \boldsymbol{w}_F \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{4}$$

where d is the distance to the plane and $\boldsymbol{N} = \begin{bmatrix} n_x & n_y & n_z \end{bmatrix}$. We can introduce the plane equation into the projection matrix from Equation 2:

$$\begin{bmatrix} x_i \\ y_i \\ d_i \\ w_i \end{bmatrix} = \begin{bmatrix} f_x & 0 & 0 & c_x \\ 0 & f_y & 0 & c_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ n_x^{(c)} & n_y^{(c)} & n_z^{(c)} & -\boldsymbol{N}^{(c)} \cdot \boldsymbol{w}_F^{(c)} \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} \\ v_{21} & v_{22} & v_{23} & v_{24} \\ v_{31} & v_{32} & v_{33} & v_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{5}$$

The product of the matrices in above is a full 4×4 transformation matrix, which is not rank-deficient and can be inverted. Note that the pixel coordinates $\boldsymbol{p}_K$ and $\boldsymbol{p}_i$ now have an extra dimension d, which should be set to 0 (because we constrain 3D point w to lie on the plane).

It should be noted that the normal and the point in the plane equation have superscript $^{(c)}$, which denotes that the plane is given in the *camera* coordinate system, rather than in the world coordinate system. This is because the point $\begin{bmatrix} X & Y & Z & 1 \end{bmatrix}$ is transformed from the world to the camera coordinates by the view matrix $\boldsymbol{V_i}$ before it is multiplied by our modified projection matrix. This could be a desired behavior for the virtual camera, for example if we want the focal plane to follow the camera and be perpendicular to the camera's optical axis. But, if we simply want to specify the focal plane in the world coordinates, we have two options: either replace the third row in the final matrix (obtained after multiplying the three matrices in Equation 5) with our plane equation in the world coordinate system; or to transform the plane to the camera coordinates:

$$\boldsymbol{w}_F^{(c)} = \boldsymbol{V}_i \, \boldsymbol{w}_F \tag{6}$$

and

$$\boldsymbol{N}^{(c)} = \overline{\boldsymbol{V}}_i \, \boldsymbol{N} \,. \tag{7}$$

$\overline{\boldsymbol{V}}_i$ is the "normal" or direction transformation for the view matrix $\boldsymbol{V}_i$, which rotates the normal vector but it does not translate it. It is obtained by setting to zero the translation coefficients w_{14}, w_{24}, w_{34}.

Now let us find the final mapping from the virtual camera coordinates $\hat{\boldsymbol{p}_K}$ to the array camera coordinates $\hat{\boldsymbol{p}}_i$. We will denote the extended coordinates (with extra d) in Equation 5 as $\hat{\boldsymbol{p}_K}$ and $\hat{\boldsymbol{p}}_i$. We will also denote our new projection and intrinsic matrices in Equation 5 as $\hat{\boldsymbol{P}}$ and $\hat{\boldsymbol{K}}$. Given that, the mapping from $\boldsymbol{p}_K$ to $\boldsymbol{p}_i$ can be expressed as:

$$\hat{\boldsymbol{p}}_i = \hat{\boldsymbol{K}}_i \, \hat{\boldsymbol{P}} \, \boldsymbol{V}_i \, \boldsymbol{V}_K^{-1} \, \hat{\boldsymbol{P}}^{-1} \, \hat{\boldsymbol{K}}_K^{-1} \, \hat{\boldsymbol{p}_K}. \tag{8}$$

The position on the aperture plane $\boldsymbol{w}_A$ can be readily found from:

$$\boldsymbol{w}_A = \boldsymbol{V}_i^{-1} \hat{\boldsymbol{P}}_A^{-1} \hat{\boldsymbol{K}}_i^{-1} \hat{\boldsymbol{p}_K} \,, \tag{9}$$

where the projection matrix $\hat{\boldsymbol{P}}_A$ is modified to include the plane equation of the aperture plane, the same way as done in Equation 5.

1.2 Reconstruction functions

The choice of the reconstruction function A_i will determine how images from different cameras are mixed together. The functions shown in Figure 1 will perform bilinear-interpolation between the views. Although this could be a rational choice, it will result in ghosting for the parts of the scene that are further away from the focal plane F. Another choice is to simulate a wide-aperture camera and include all cameras in the generated view (set $A_i = 1$). This will produce an image with a very shallow depth of field. Another possibility is to use the nearest-neighbor strategy and a box-shaped reconstruction filter (the width of the boxes being equal to the distance between the cameras). This approach will avoid ghosting but will cause the views to jump sharply as the virtual camera moves over the scene. It is worth experimenting with different reconstruction startegies to choose the best for a given application but also for the given angular resolution of the light field (number of views).

References

[1] Aaron Isaksen, Leonard McMillan, and Steven J. Gortler. Dynamically reparameterized light fields. In *Proc of SIGGRAPH '00*, volume 7, pages 297–306, New York, New York, USA, 2000. ACM Press.

[2] Steve Marschner and Peter Shirley. *Fundamentals of Computer Graphics*. A K Peters/CRC Press, 4 edition edition, 2016.

[3] Ryan S. Overbeck, Daniel Erickson, Daniel Evangelakos, Matt Pharr, and Paul Debevec. A system for acquiring, processing, and rendering panoramic light field stills for virtual reality. *ACM Transactions on Graphics*, 37(6):1–15, dec 2018.

[4] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag New York Inc, 2010.