CSC320H: Intro to Visual Computing

Lecture 2

Good to know

A1 is now due Monday Jan 27 (midnight)

- Check grace-day lateness policy (ask me about it if unclear)
- Check academic honesty section of the course info sheet (this is very important stuff, better know now than to be sorry later!).

Please come to my office hours.

- You're welcome to come only to say hi, but it's great if you ask questions… there's never a bad one!

Slides are password now password protected. Same as the assignments. (usr: visual, pwd: computation)

Today's Topics

2. Color

3. Image matting

1. High Dynamic Photography

Topic 2:

Color

- The spectral power distribution function
- Color image sensors

Image Acquisition

- Light goes through a lens and onto the sensor

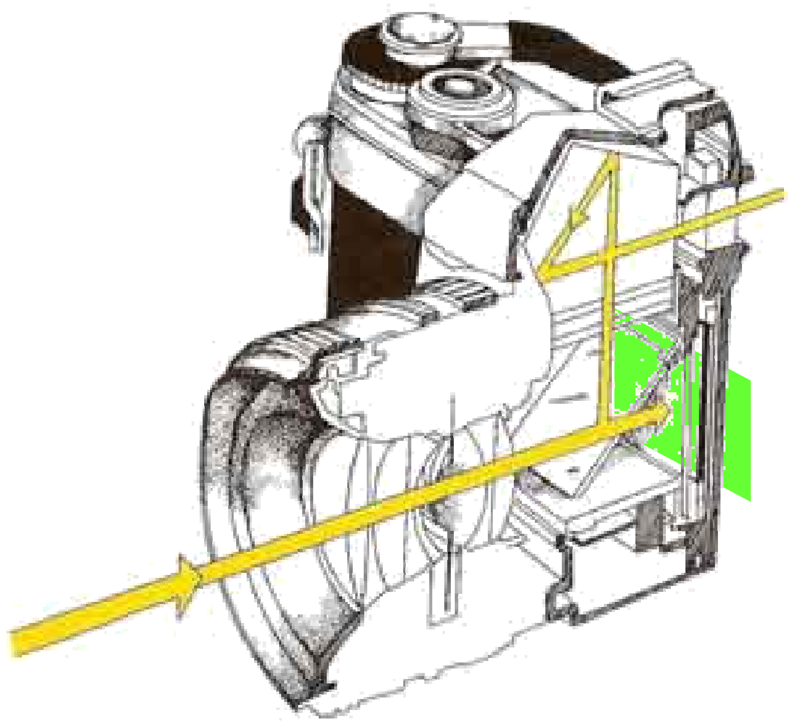
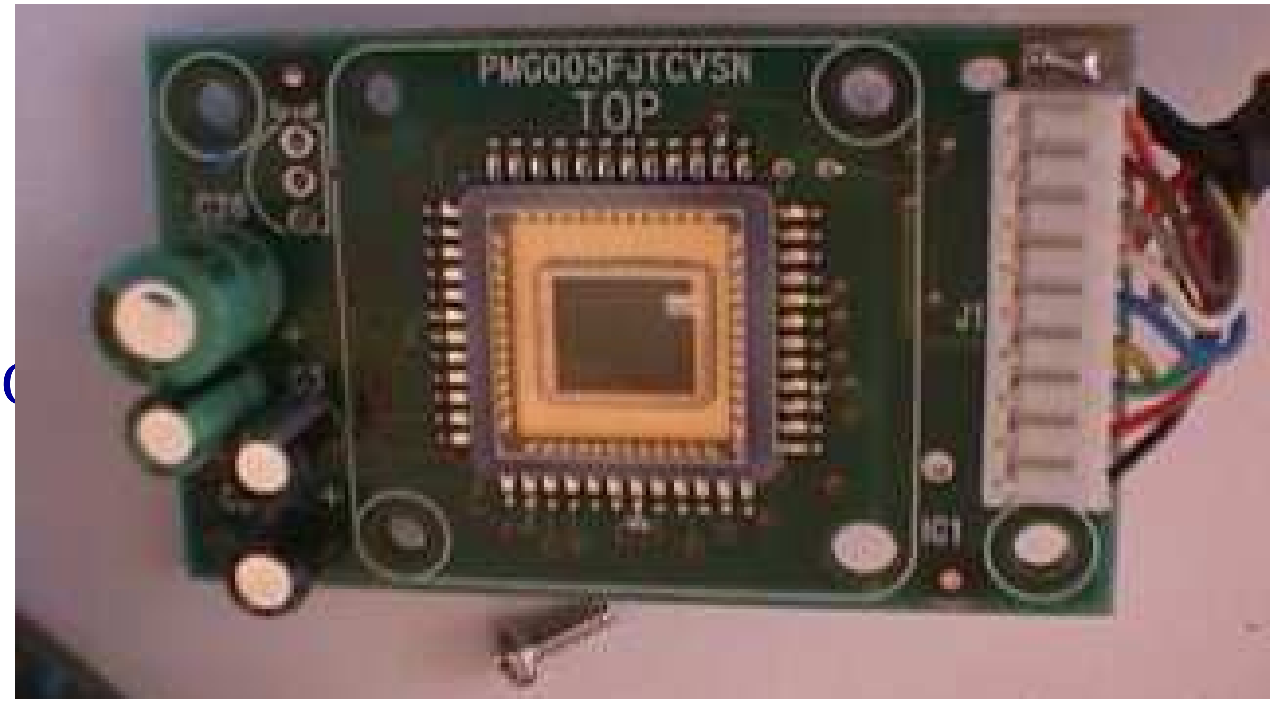


Image Acquisition: B&W Cameras

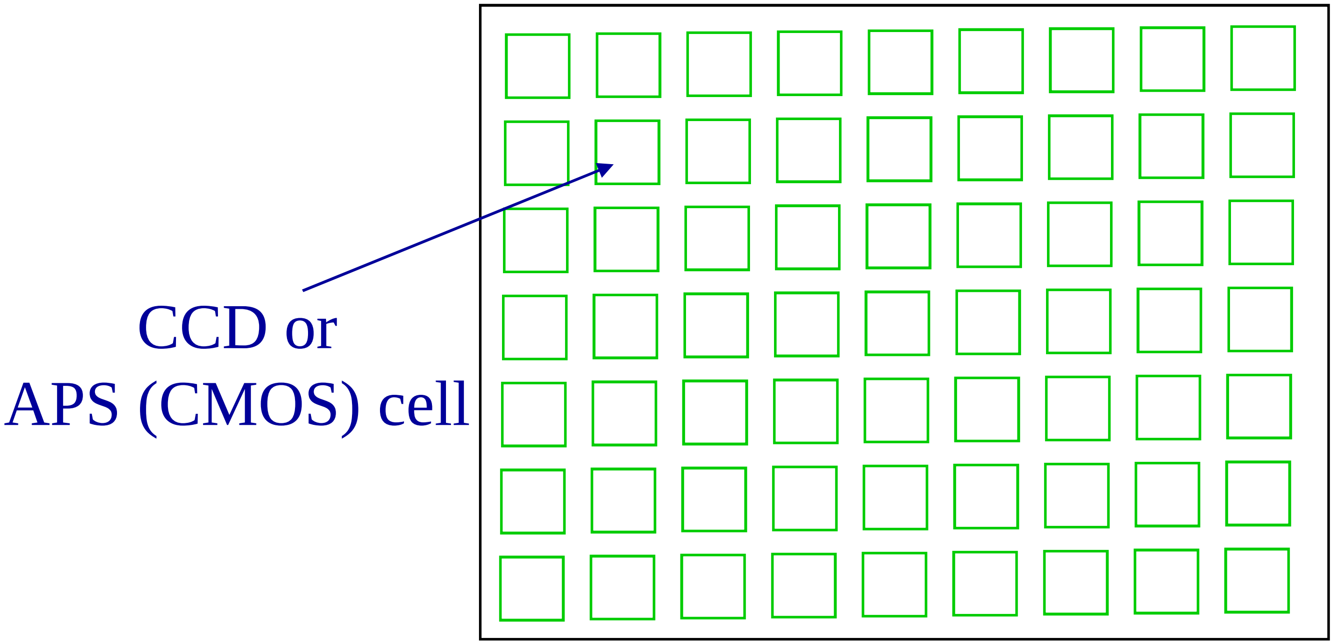
- Image sensors: 2-dimensional array of photo-sensitive cells, each corresponding to one pixel



- Light falling onto cell footprint causes generation of voltage that depends on intensity of incident light
- Voltage reading converted to digital signal within a sensor-specific range (usually an 8-, 10- or 14-bit number)

Image Acquisition: B&W Cameras

- Image sensors: 2-dimensional array of photo-sensitive cells, each corresponding to one pixel



- Light falling onto cell footprint causes generation of voltage that depends on intensity of incident light
- Voltage reading converted to digital signal within a sensor-specific range (usually an 8-, 10- or 14-bit number)

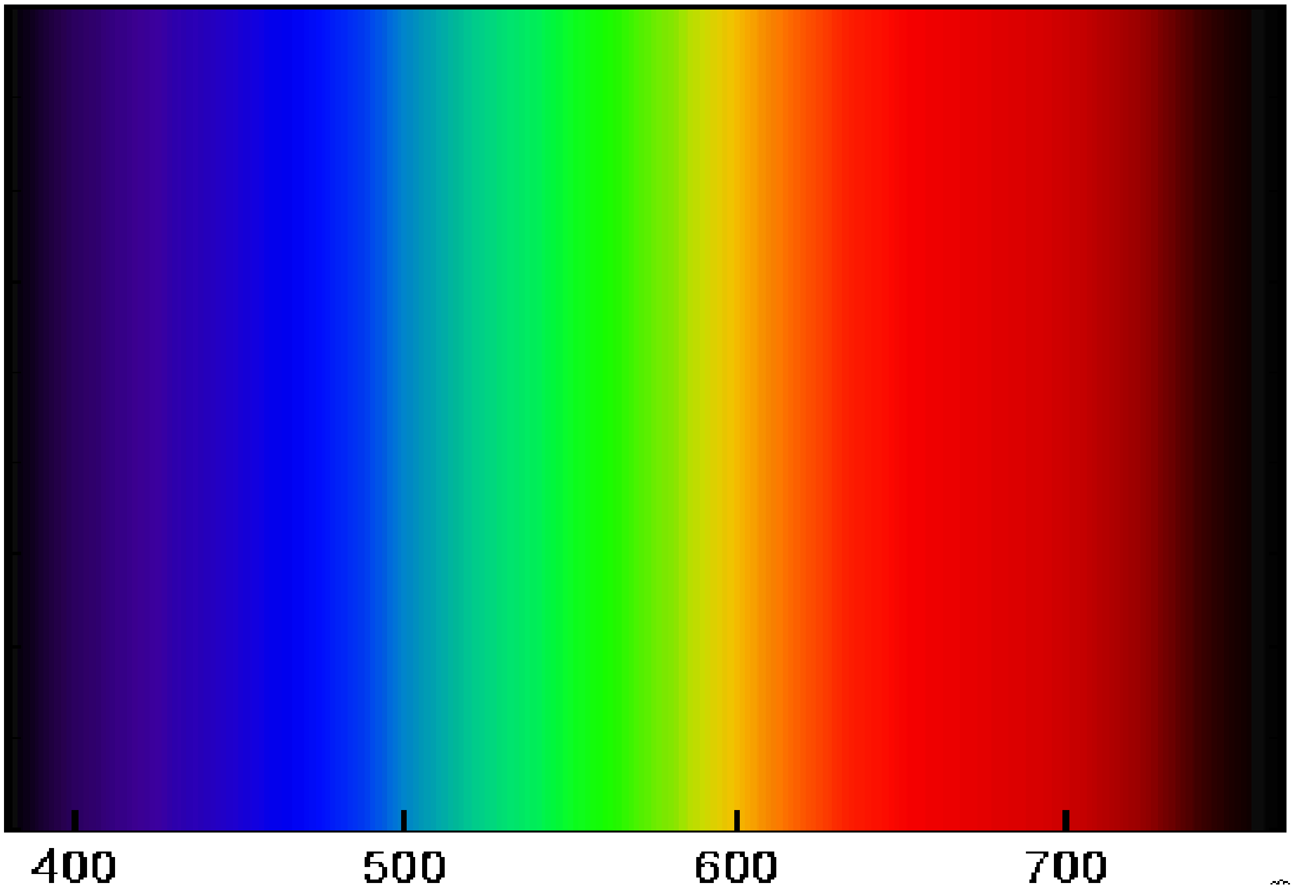
Image Acquisition: B&W Cameras

- Image sensors: 2-dimensional array of photo-sensitive cells, each corresponding to one pixel

- Light falling onto cell footprint causes generation of voltage that depends on intensity of incident light
- Voltage reading converted to digital signal within a sensor-specific range (usually an 8-, 10- or 14-bit number)

Color

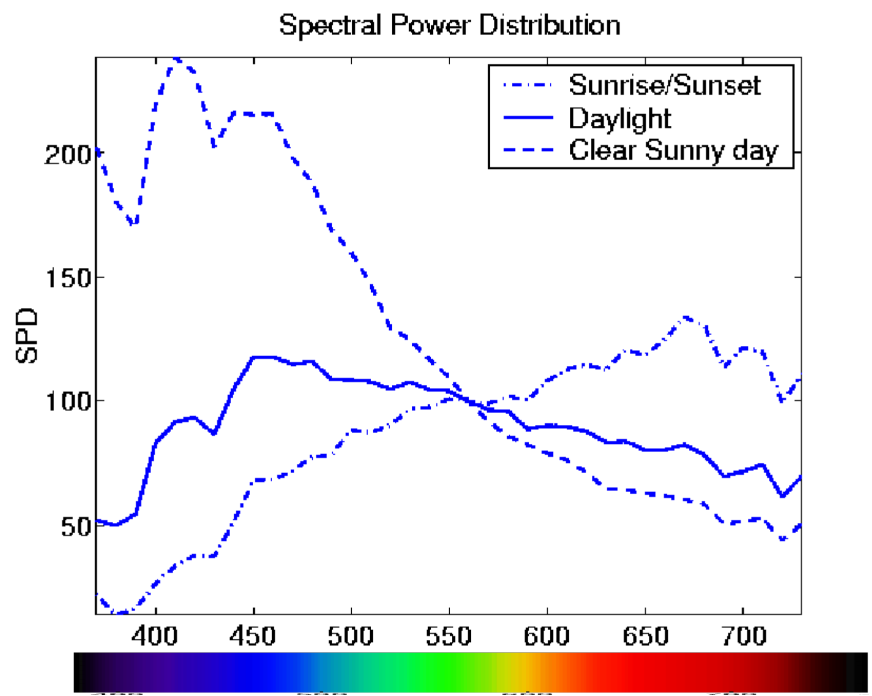
- The visible spectrum of light corresponds to wavelengths roughly from 400 to 700nm

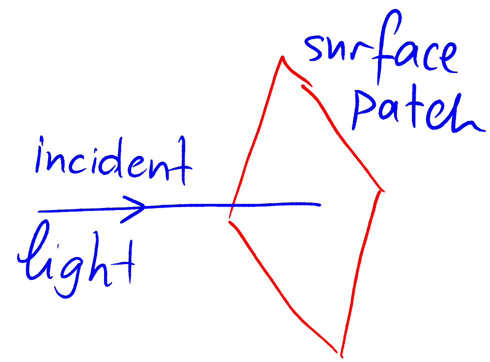


(adapted from CS320 notes, Jepson, 2005)

Color

- We describe light in terms of the power present at each wavelength in the visible spectrum, I(λ)
- I(λ) is called the spectral power distribution (SPD). It is measured in units of Watts per unit wavelength per unit cross-sectional area (m^2)



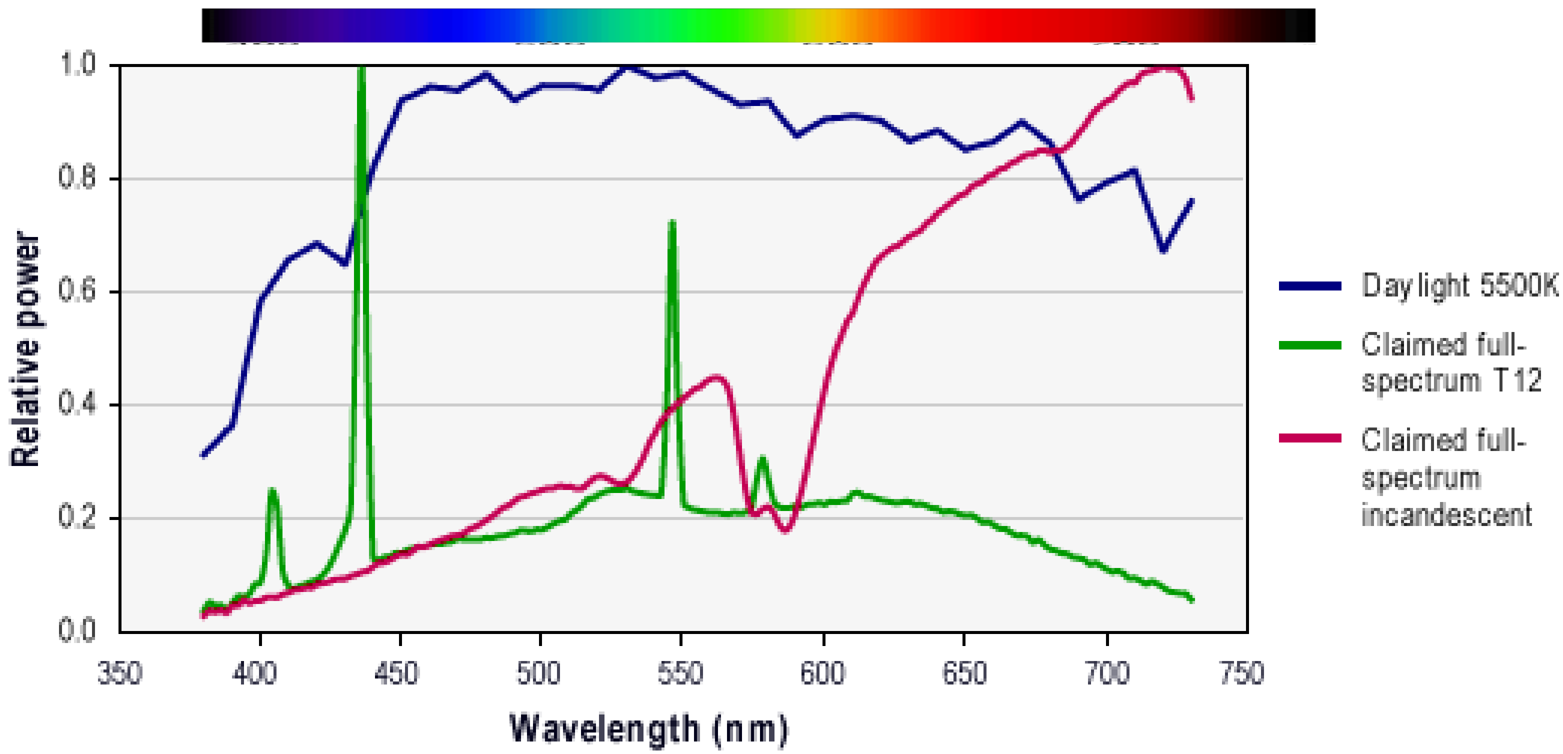


$I(\lambda)$ measures energy/sec flowing through the cross-section per m^2

(adapted from CS320 notes, Jepson, 2005)

Spectral Power Distribution

- Are “full-spectrum” light sources close to “natural” day light?

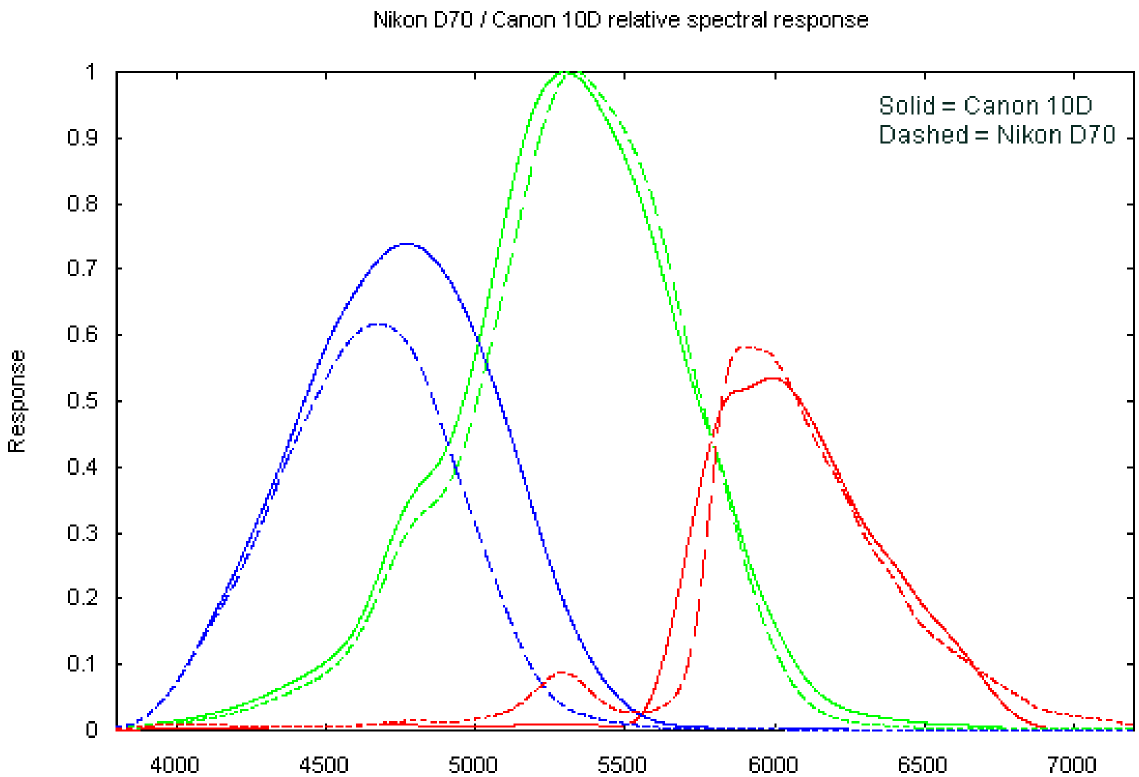


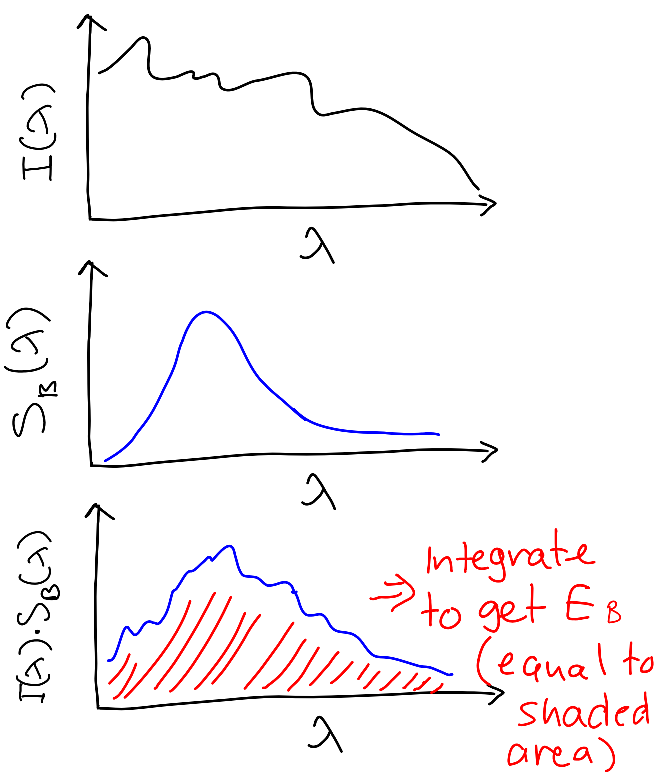
(from http://www.lrc.rpi.edu/programs/nlpip/lightinganswers/fullspectrum/lightSources.asp)

Sensor irradiance

- Visual cells (cones) respond as a function of the energy absorbed. This energy, per unit time (i.e., sensor irradiance) is given by:

$$E_\mu = \int_0^\infty I(\lambda) \cdot S_\mu(\lambda)\, d\lambda \quad \text{for} \quad \mu = R, G, B$$



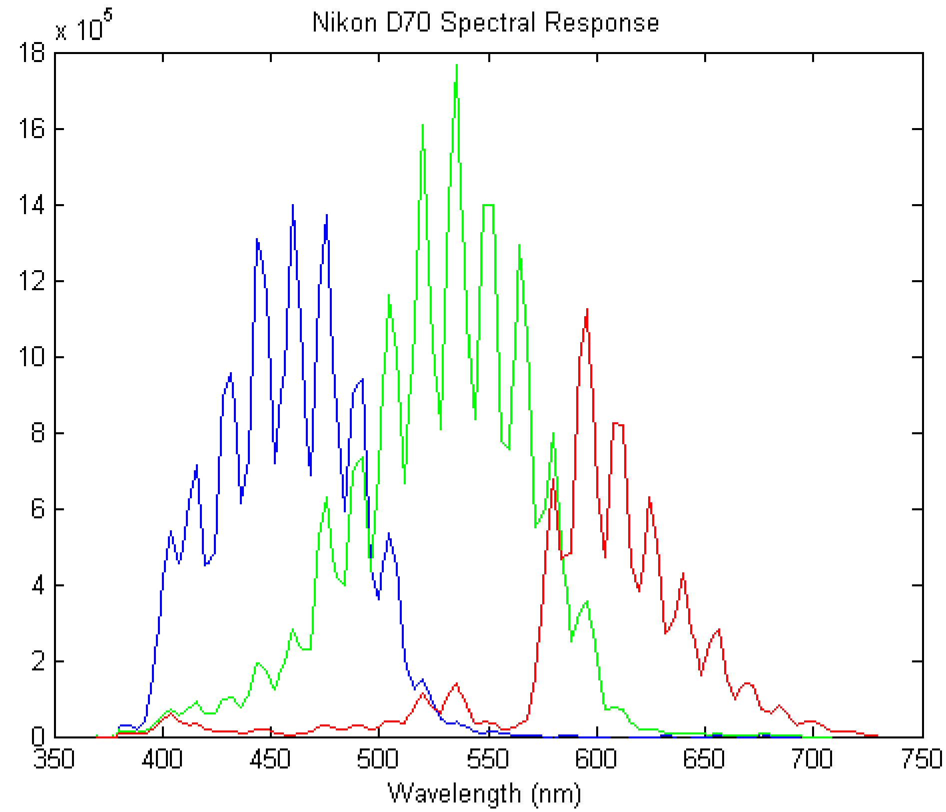


From: astrosurf.com/buil/d70v10d/eval.htm

Color Image Acquisition

- The same is true for electronic image sensors: sensor irradiance is given by

$$E_\mu = \int_0^\infty I(\lambda) \cdot S_\mu(\lambda)\, d\lambda \quad \text{for} \quad \mu = R, G, B$$



(from http://scien.stanford.edu/pages/labsite/2007/psych221/projects/07/camera_characterization/spectral_sensitivity.html)

Image Acquisition: Color Cameras?

- Imaging sensor: 2-dimensional array of photo-sensitive cells, each corresponding to one pixel

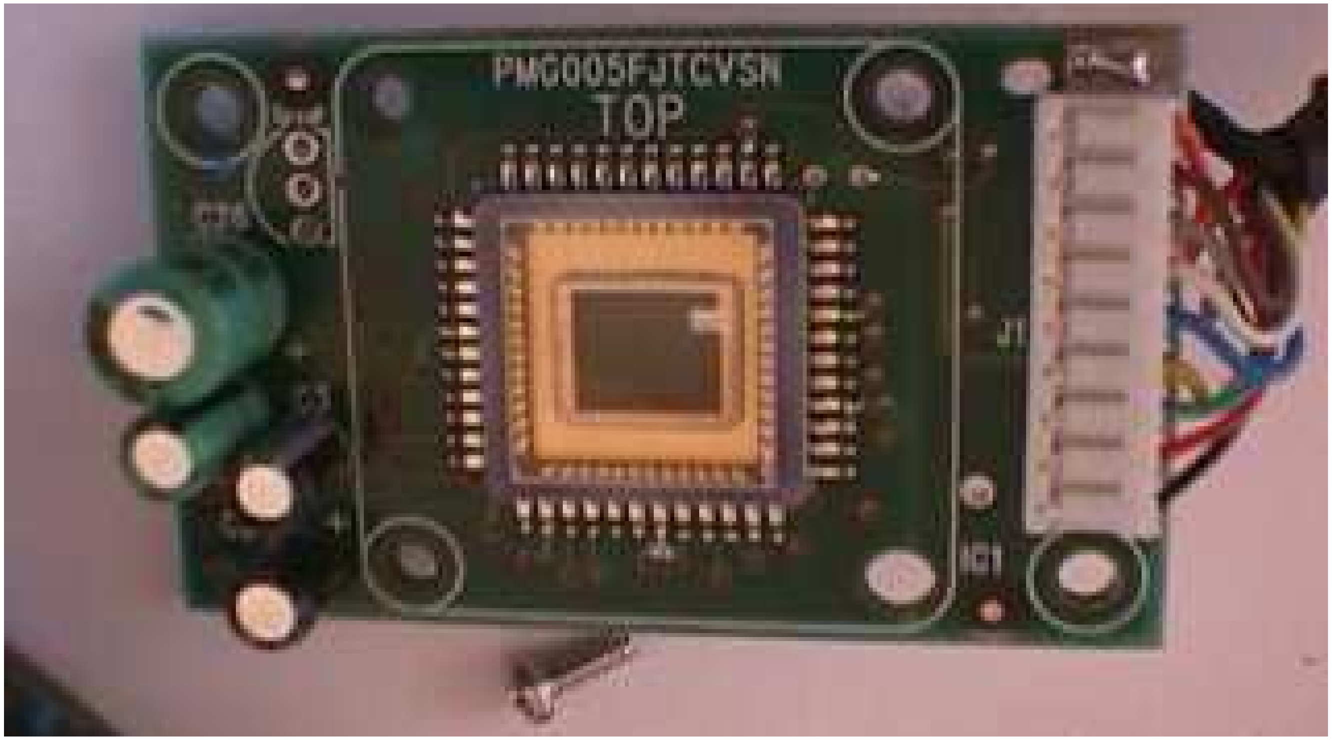
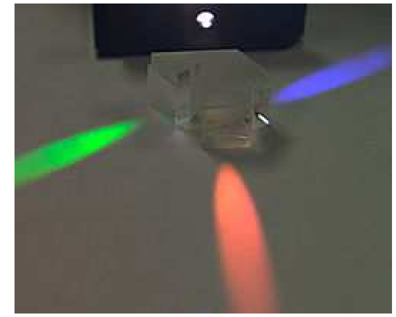
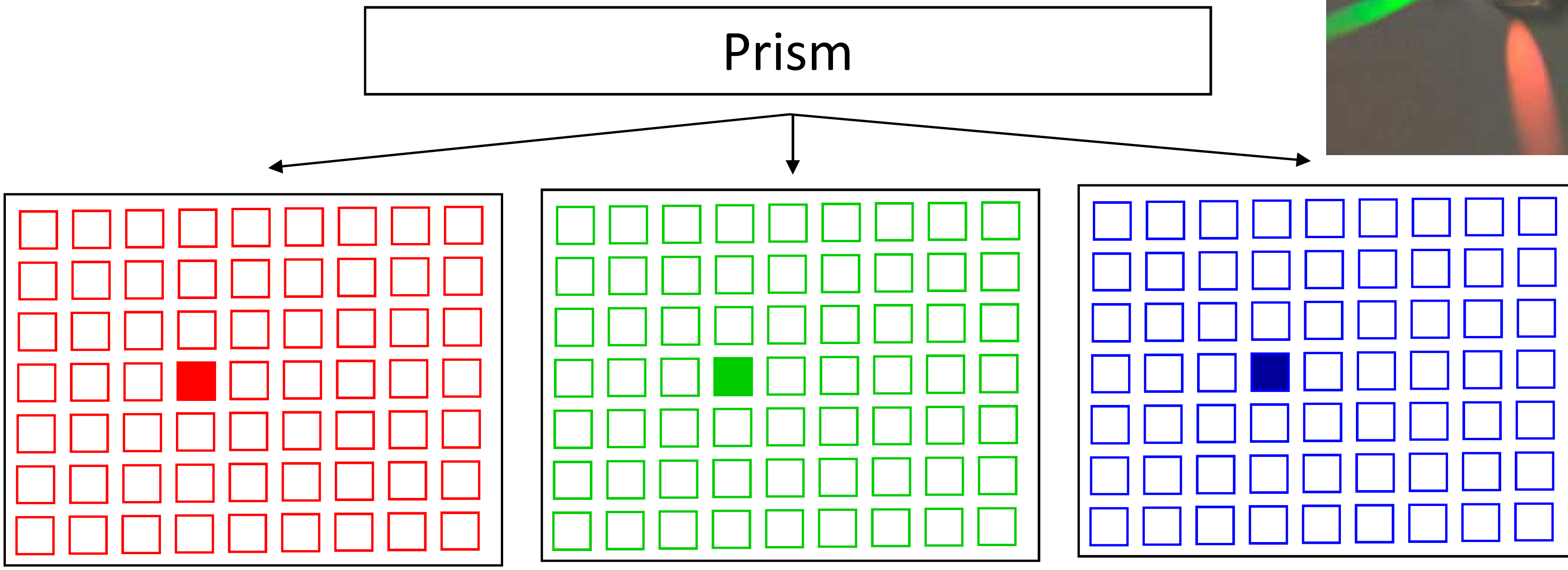


Image Acquisition: Color Cameras

- Solution #1: 3-chip color cameras

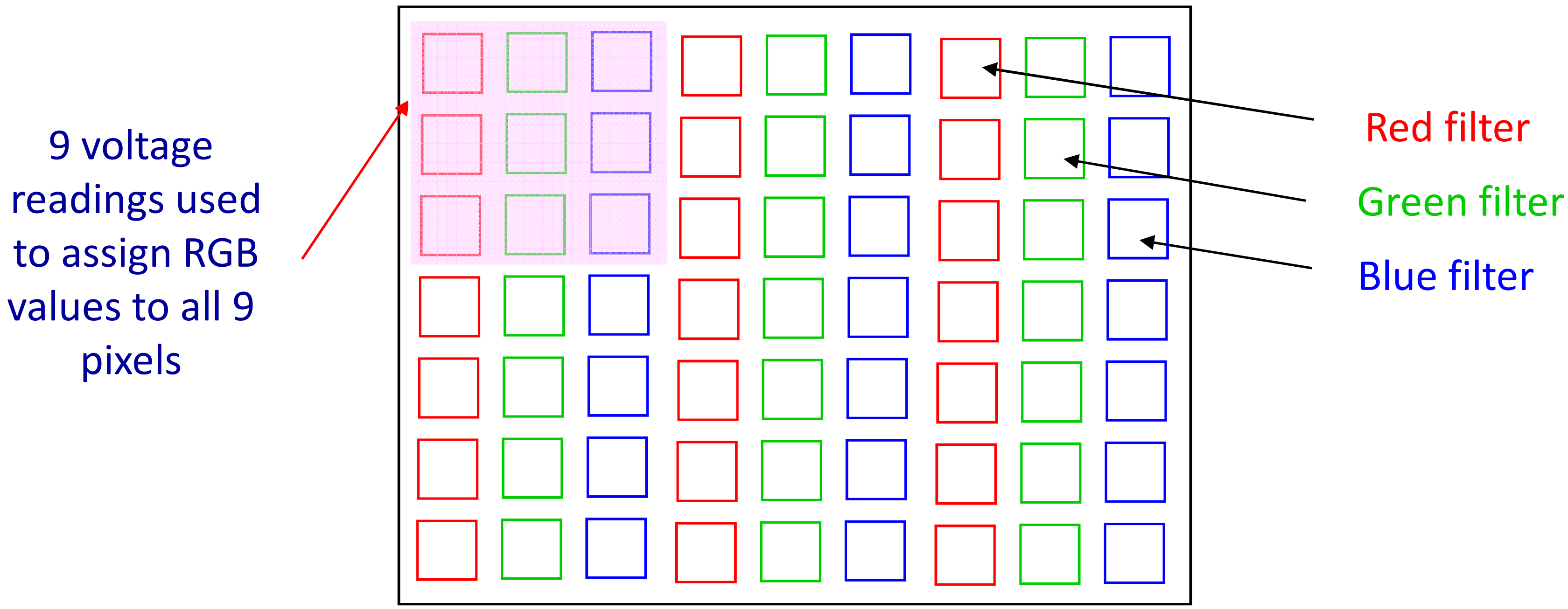




- NxM color images generated by 3 NxM sensor arrays
- Each array covered by single color filter (R, G, or B)
- R,G,B color of a pixel corresponds to color of light falling precisely at a single position of the sensor array(s)
- Prism & sensor arrays must be aligned very carefully

Image Acquisition: Color Cameras

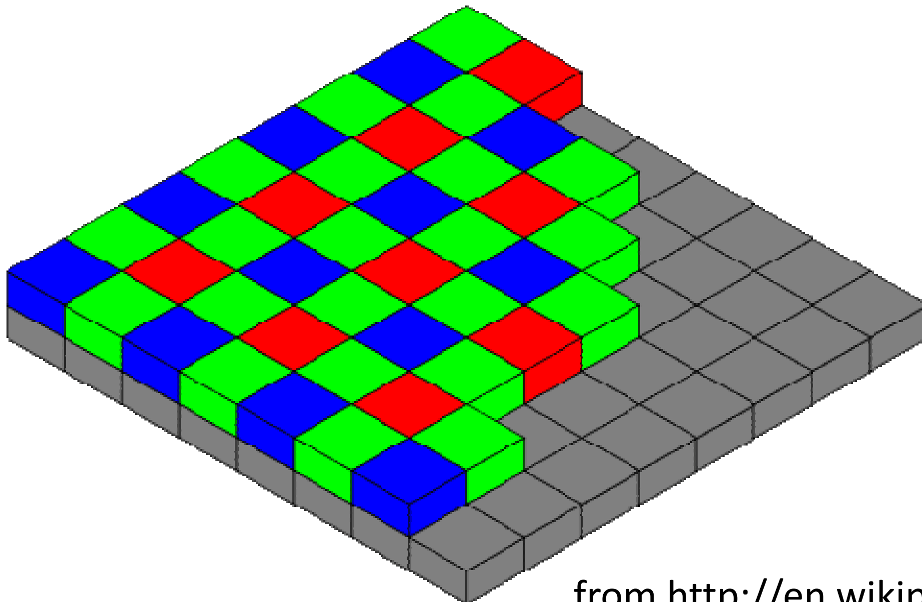
- Solution #2: Single-chip color cameras



- 3-filter "mask" placed on top of sensor array, each permitting only red, green, or blue light to go through
- Interpolation algorithms assign 3-band colors to every pixel
- The process is called de-mosaicing
- Result: NxM color image from an NxM sensor array

Image Acquisition: Color Cameras

- Solution #2: Single-chip color cameras



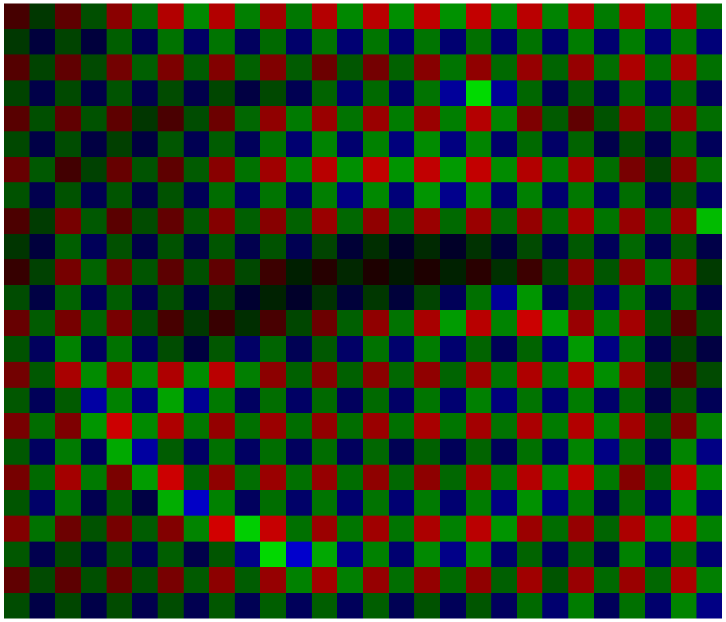
from http://en.wikipedia.org/wiki/Demosaicing

Bayern pattern

- In practice the R,G,B filters are laid out in a more complex pattern than just column-wise
- Usually there are more G filters than R or B filters

Image Acquisition: Color Cameras

- Solution #2: Single-chip color cameras

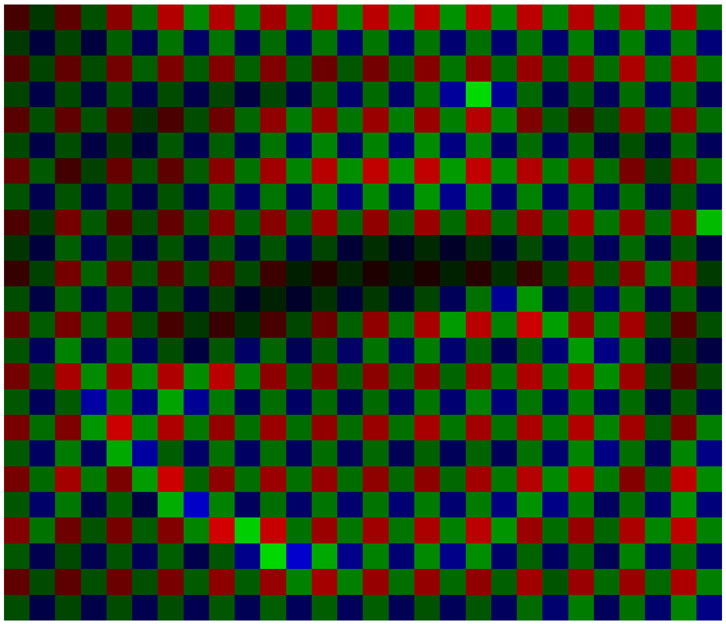


Raw sensor image

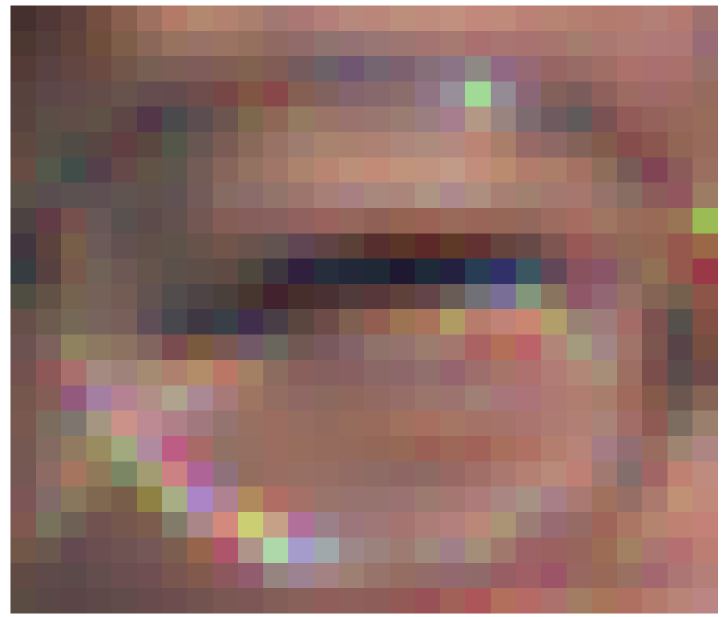
Demosaicing via Linear interpolation

Image Acquisition: Color Cameras

- Solution #2: Single-chip color cameras



Raw sensor image



Demosaicing

Almost every modern camera can output ‘raw’ images.

‘dcraw’ is an open source program that reads most ‘raw’ formats

RAW vs. Developed color images

A color image after “developing” demosaicing + intensity mapping

RAW vs. Developed color images

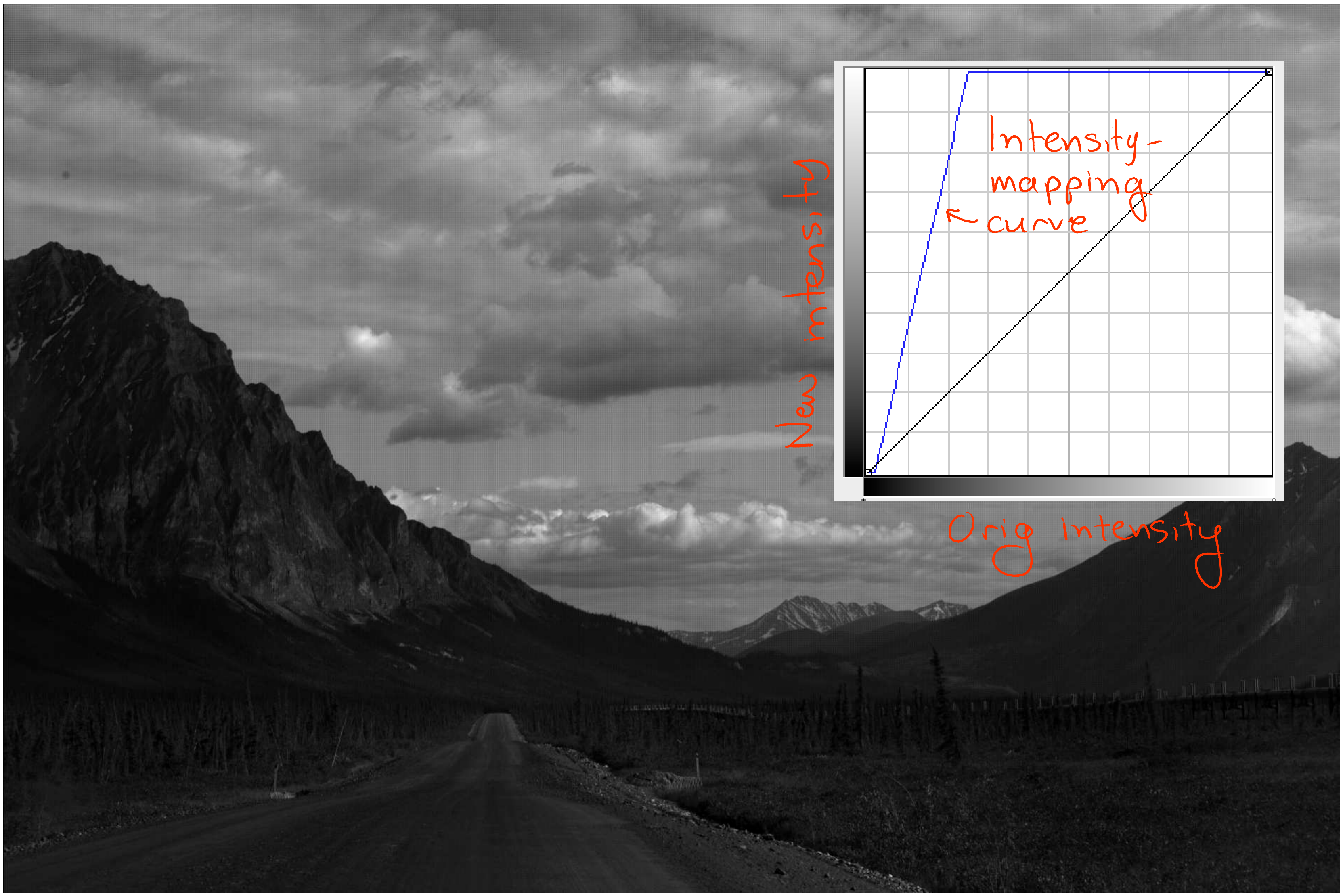
- The color image before “developing” (linear RAW image)

RAW vs. Developed color images

- The color image before “developing” (contrast-enhanced)

RAW vs. Developed color images

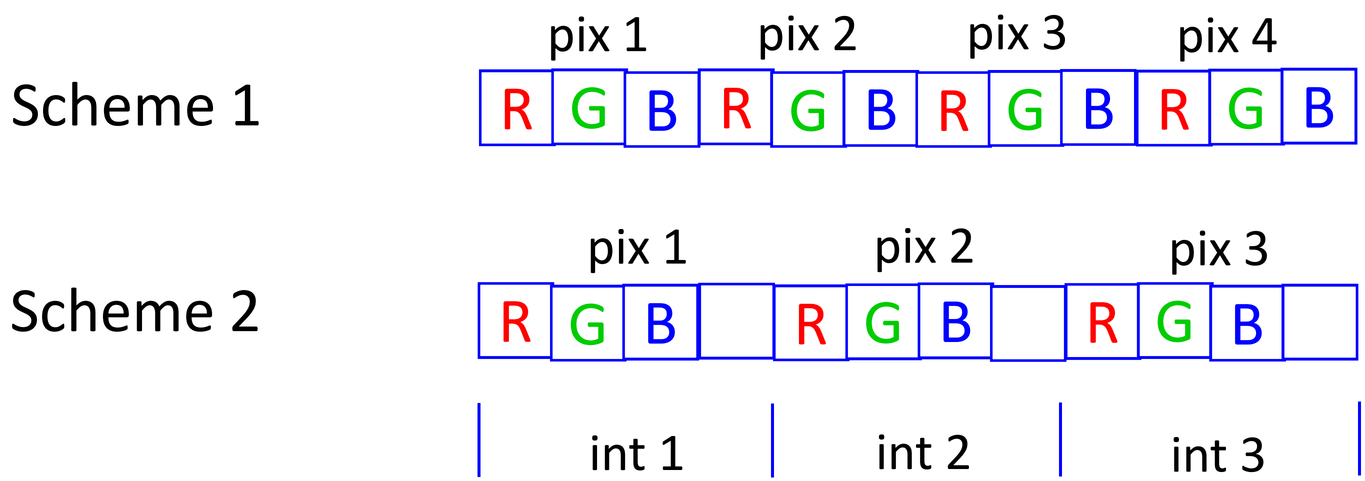
- The color image before "developing" (contrast-enhanced)



Computational Considerations

Suppose we allocate 1 byte per color component per pixel

Consider the following two image storage schemes:



Q: Is one scheme computationally faster?

Ans: Scheme 2 is often faster because pixels are aligned at 4-byte (i.e., integer) boundaries

Topic 3:

Image Matting

- The pixel alpha component
- The Matting Equation
- Four solutions to the Matting Equation
- The Triangulation Matting algorithm

The Alpha Pixel Component

Frequently in computer graphics, pixels are assumed to have 4 rather than 3 components

pix 1				pix 2				pix 3			
R	G	B	α	R	G	B	α	R	G	B	α

- The 4th component is called the alpha component
  - usually assumed to have fractional values between 0 &1
  - when represented as 1 byte, carries the values $\alpha = 0/255, 1/255, \ldots, 254/255, 255/255$
  - defines pixel 'transparency'

$$\begin{bmatrix} R \\ G \\ B \\ \alpha \end{bmatrix} \stackrel{def}{=} \begin{bmatrix} \alpha R \\ \alpha G \\ \alpha B \end{bmatrix}$$

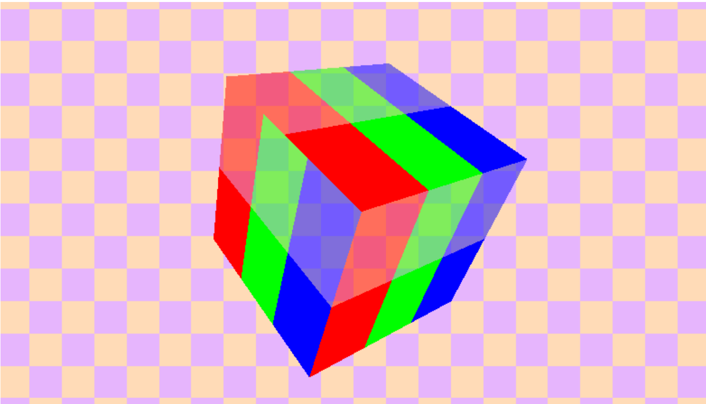


Image Matting

Image Matting is typically used to merge graphics with real images (or with more computer graphics)

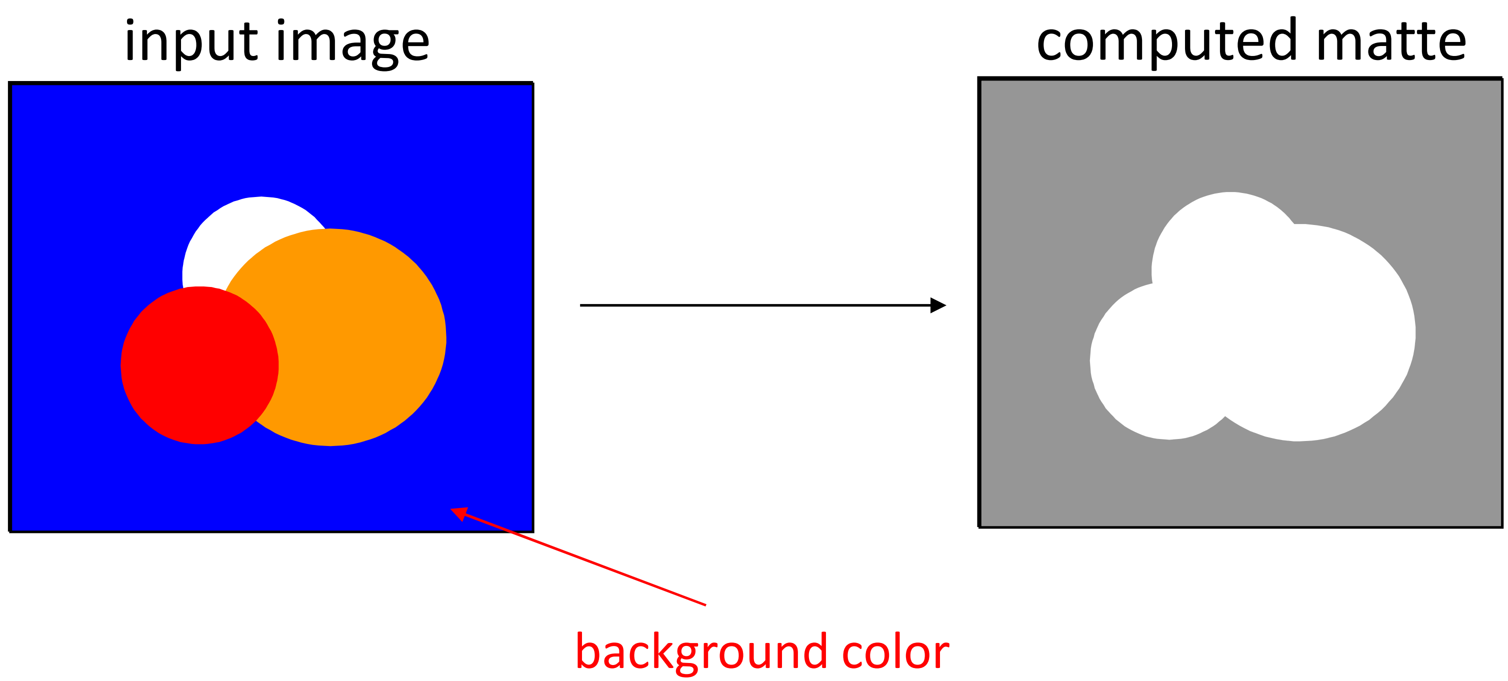
Technology has many applications:

- Sports advertising
- Medical applications (“x-ray vision”)
- Special visual effects

Constant-Color Matting

<u>Goal:</u>

Separating a desired foreground image from a background of constant (or almost constant) color



<u>Blue-Screen Matting:</u>

Matting images whose background color is blue

Computing Semi-Transparent Mattes

In general, we might want to obtain a semi-transparent matte from an original image (eg. image on the left)

compositing glass with
portrait using
a semi-transparent matte

$\Rightarrow$ Matte can be thought of as a gray-scale image with values between 0 & 1 (ie. the alpha component of an RGBα image)

Matting Problem: Mathematical Definition

Intuitively, the Matting Problem amounts to extracting a foreground image & its matte from a given composite image

For every pixel in the composite image,

<u>given</u>

– background color

– composite pixel color

<u>compute</u>

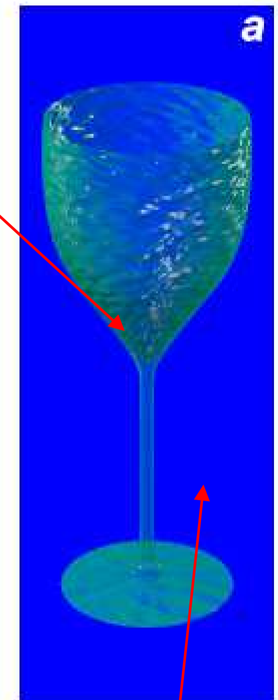
– object pixel color
and alpha such that

$$C_k = \begin{bmatrix} R_k \\ G_k \\ B_k \end{bmatrix}$$

$$C = \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$C_o = \begin{bmatrix} R_o \\ G_o \\ B_o \end{bmatrix} \overset{def}{=} \begin{bmatrix} R_o/a \\ G_o/a \\ B_o/a \\ a \end{bmatrix}$$

$\alpha_o \cong 1$



$\alpha_o = 0$

Matting Equation

$$\boxed{C = C_o + C(1-\alpha_o)C_k}$$

$$\boxed{\text{Reminder: } \begin{bmatrix} R \\ G \\ B \\ \alpha \end{bmatrix} = \begin{bmatrix} \alpha R \\ \alpha G \\ \alpha B \end{bmatrix}}$$

Why is Matting Hard?

$$\boxed{\begin{array}{l}\text{Matting Equation} \\ C = C_o + (1-\alpha_o)\, C_k\end{array}}$$

C: composite pixel color
C_o: object pixel color
C_k: background pixel color

Example:

$$C = C_K = \begin{bmatrix} 100 \\ 100 \\ 200 \end{bmatrix}$$

What is the solution of the matting eq?

$\alpha = 1$, $C_o = \begin{bmatrix} 100 \\ 100 \\ 200 \end{bmatrix}$ $C_K =$ arbitrary

$\alpha = 0$, $C_o =$ arbitrary $C_K = \begin{bmatrix} 100 \\ 100 \\ 200 \end{bmatrix}$

$\alpha = 0.5$ $C_o = \begin{bmatrix} 50 \\ 50 \\ 100 \end{bmatrix}$ $C_K = \begin{bmatrix} 100 \\ 100 \\ 200 \end{bmatrix}$

Why is Matting Hard?

$$\boxed{\text{Matting Equation} \quad C = C_o + (1-\alpha_o)\, C_K}$$

More generally,

$$C = C_o + C_K - \alpha_o C_K$$

define $C_\Delta = C - C_K$

$$C_\Delta = C_o - \alpha_o C_K$$

In matrix form,

$$\begin{bmatrix} R_\Delta \\ G_\Delta \\ B_\Delta \end{bmatrix} = \begin{bmatrix} 1 & & & -R_K \\ & 1 & & -G_K \\ & & 1 & -B_K \end{bmatrix} \begin{bmatrix} R_o \\ G_o \\ B_o \\ \alpha_o \end{bmatrix}$$

This 3x4 matrix is <u>not</u> invertible

\# equations < \# unknowns

$\Rightarrow$ infinite solutions

$\Rightarrow$ To solve system, we must increase the number of equations

Matting Solutions

Matting Equation

$$C = C_o + (1-\alpha_o)\,C_K$$

More generally,

$$C = C_o + C_K - \alpha_o C_K$$

define $C_\Delta = C - C_K$

$$C_\Delta = C_o - \alpha_o C_K$$

In matrix form,

$$\begin{bmatrix} R_\Delta \\ G_\Delta \\ B_\Delta \end{bmatrix} = \begin{bmatrix} 1 & & & -R_K \\ & 1 & & -G_K \\ & & 1 & -B_K \end{bmatrix} \begin{bmatrix} R_o \\ G_o \\ B_o \\ \alpha_o \end{bmatrix}$$

Solution #1

$$\alpha = \begin{cases} 0 & C = C_K \\ 1 & \text{otherwise} \end{cases}$$

Solution #2

No blue ($B_o = 0$)

Solution #3

Gray ($R_o = G_o = B_o$)

or

Flesh $C_o = \begin{bmatrix} d \\ 0.5d \\ 0.5d \end{bmatrix}$

Solution #4

Triangulation matting.

Solution #1: Known Background Color

- "Trivial" approach that never solves the matting equation:

  set α_o=0 if pixel color is equal to the backing color
  set α_o=1 otherwise

- Blue or green backing colors typically used

Limitations:

- Background color must be known very accurately & must be constant (and, even then, pixels will still contain noise!)
- Foreground subject (e.g., weather-person) cannot wear anything with color similar to the backing color!!!
- No "mixed" pixels allowed (i.e., with α_o between 0 & 1)

Solution #2: No Blue

Matting Equation:

$$\boxed{C = C_o + (1 - \alpha_o)\, C_k}$$

- If we know that the foreground contains no blue, we have $B_o = 0$
- This leaves us with 3 equations and 3 unknowns, which has exactly one solution

$$\boxed{\begin{aligned} R &= \alpha_o R_o + (1 - \alpha_o)\, R_k \\ G &= \alpha_o G_o + (1 - \alpha_o)\, G_k \\ B &= B_k - \alpha_o B_k \end{aligned}}$$

$\leftarrow$ 3. Solve for R_o

$\leftarrow$ 2. Solve for G_o

$\leftarrow$ 1. Solve for α_o

Main difficulty:

- The “no blue” assumption is very restrictive!!!!
- Excludes all gray colors except black, about 2/3 of all hues, and all pastels & tints of the remaining hues (because white contains blue)

Solution #3: Gray or Flesh

Matting Equation:

$$\boxed{C = C_o + (1 - \alpha_o)\, C_k}$$

- If we know that the foreground contains gray, that means that $R_o = B_o = G_o$
- This leaves us with 3 equations and 2 unknowns

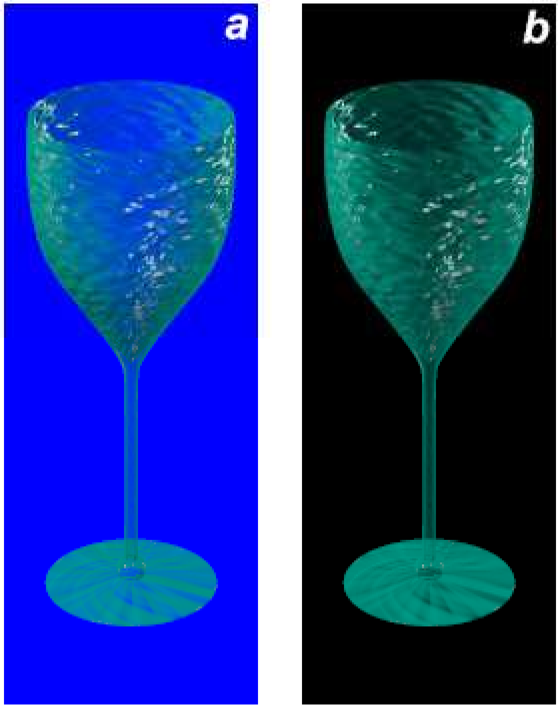
- Flesh typically has color of the form [d 0.5d 0.5d], where d determines the darkening due to skin color differences among races
- Again, this reduces the number of unknowns to 2

Solution #4: Triangulation Matting

Matting Equation:

$$\boxed{C = C_o + (1 - \alpha_o)\, C_k}$$

- Instead of reducing the number of unknowns, we could attempt to increase the number of equations
- One way to do this is to photograph an object of interest in front of two known but distinct backgrounds



- Equations: 6 (3 for each composite)
- Unknowns: 4
- Since each pixel is processed independently, the backgrounds don't need to be a constant backing color

Triangulation Matting

$$\boxed{\text{Matting Equation} \quad C = C_o + (1-\alpha_o) C_K}$$

- Eqs for composite 1

$$\begin{bmatrix} R_\Delta \\ G_\Delta \\ B_\Delta \end{bmatrix} = \begin{bmatrix} 1 & & & -R_K \\ & 1 & & -G_K \\ & & 1 & -B_K \end{bmatrix} \begin{bmatrix} R_o \\ G_o \\ B_o \\ \alpha_o \end{bmatrix}$$

- Eqs for composite 2

$$\begin{bmatrix} R_\Delta' \\ G_\Delta' \\ B_\Delta' \end{bmatrix} = \begin{bmatrix} 1 & & & -R_K' \\ & 1 & & -G_K' \\ & & 1 & -B_K' \end{bmatrix} \begin{bmatrix} R_o \\ G_o \\ B_o \\ \alpha_o \end{bmatrix}$$

Solution #1

$$\alpha = \begin{cases} 0 & C = C_K \\ 1 & \text{otherwise} \end{cases}$$

Solution #2

No blue ($B_o = 0$)

Solution #3

Gray ($R_o = G_o = B_o$)

or

Flesh $C_o = \begin{bmatrix} d \\ 0.5d \\ 0.5d \end{bmatrix}$

$$\boxed{\text{Solution \#4} \quad \text{Triangulation matting}}$$

Triangulation Matting

Matting Equation

$$\boxed{C = C_o + (1-\alpha_o) C_k}$$

- Putting the two eqs together

$$\begin{bmatrix} R_\Delta \\ G_\Delta \\ B_\Delta \\ R_\Delta' \\ G_\Delta' \\ B_\Delta' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -R_k \\ 0 & 1 & 0 & -G_k \\ 0 & 0 & 1 & -B_k \\ 1 & 0 & 0 & -R_k' \\ 0 & 1 & 0 & -G_k' \\ 0 & 0 & 1 & -B_k' \end{bmatrix} \begin{bmatrix} R_o \\ G_o \\ B_o \\ \alpha_o \end{bmatrix}$$

- Solution found by computing the pseduo-inverse of the coefficient matrix.

Solution #1

$$\alpha = \begin{cases} 0 & C = C_k \\ 1 & \text{otherwise} \end{cases}$$

Solution #2

No blue ($B_o = 0$)

Solution #3

Gray ($R_o = G_o = B_o$)

or

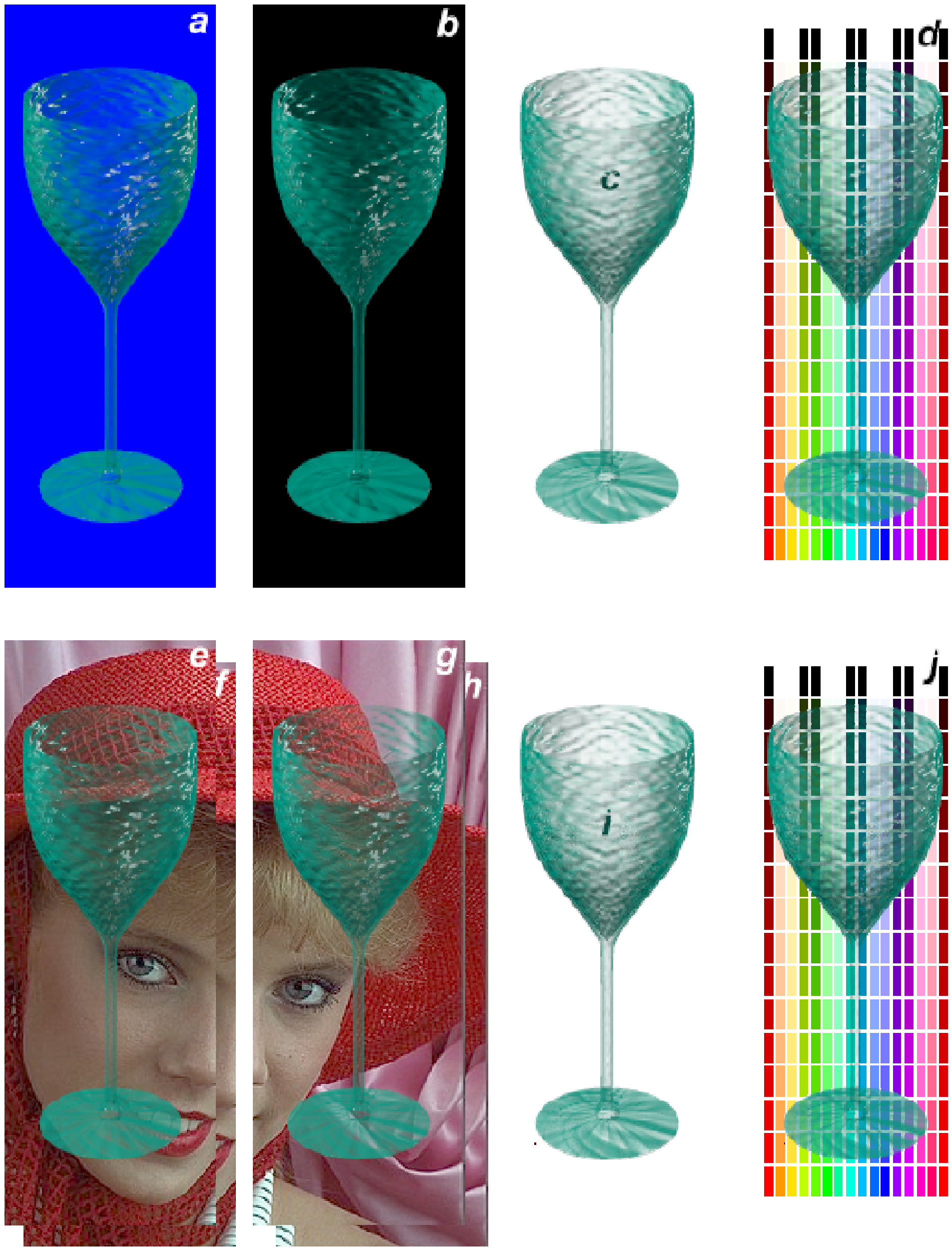
Flesh $C_o = \begin{bmatrix} d \\ 0.5d \\ 0.5d \end{bmatrix}$

Solution #4

Triangulation matting

Triangulation Matting Examples

From Smith & Blinn's SIGGRAPH'96 paper



Triangulation Matting Examples

From Smith & Blinn's SIGGRAPH'96 paper

Triangulation Matting Examples

From Smith & Blinn's SIGGRAPH'96 paper

Video Matting

Topic 01:

Image formation & HDR Photography

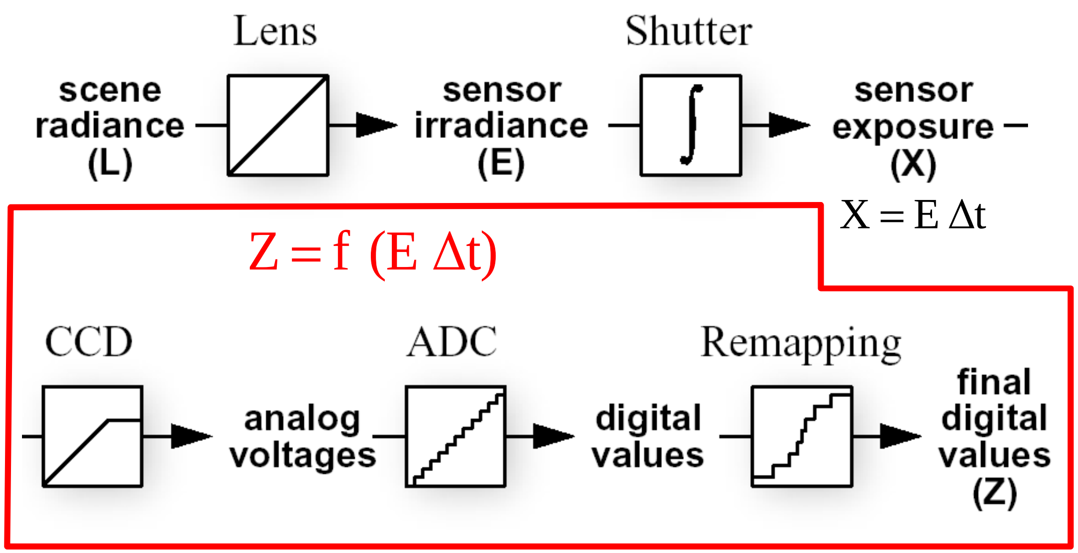
Computing the camera response function

High Dynamic Range Photography

Goal: to increase the ability of an imaging procedure to adequately image both high lights and dark shadows in a scene.

Dynamic range: The ratio of the largest non-saturating input signal to the smallest detectable input signal.

Computing The Camera Response Function



Δt_1 Δt_2

General procedure

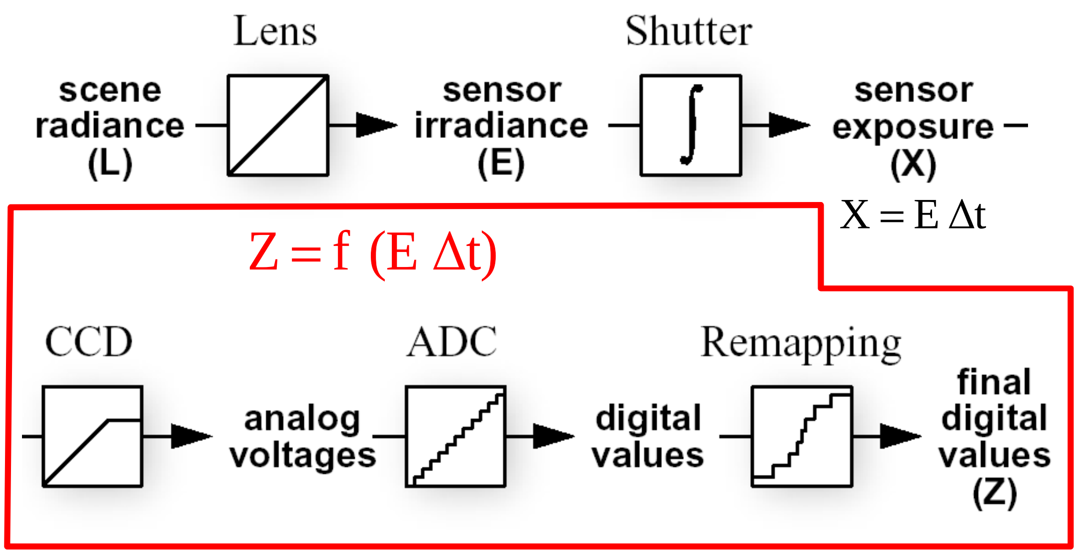
① Collect photos for many exposure intervals $\Delta t_1, \Delta t_2, \ldots$ w/out moving the camera

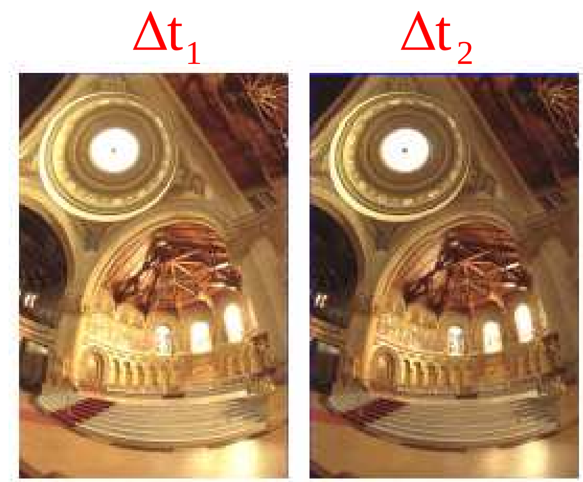
② Process photos to compute f

Problem:

For a given photo, we will know Δt_j and Z but we have no way of measuring E!

Idea #1: The log-Inverse Response Function g()





Idea #1: Invert & use log's

$$Z = f(E \cdot \Delta t)$$

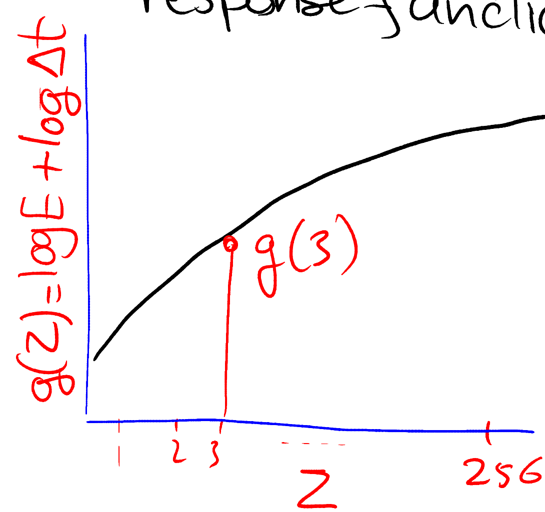
$$f^{-1}(Z) = E \cdot \Delta t$$

$$\log f^{-1}(Z) = \log(E) + \log \Delta t$$

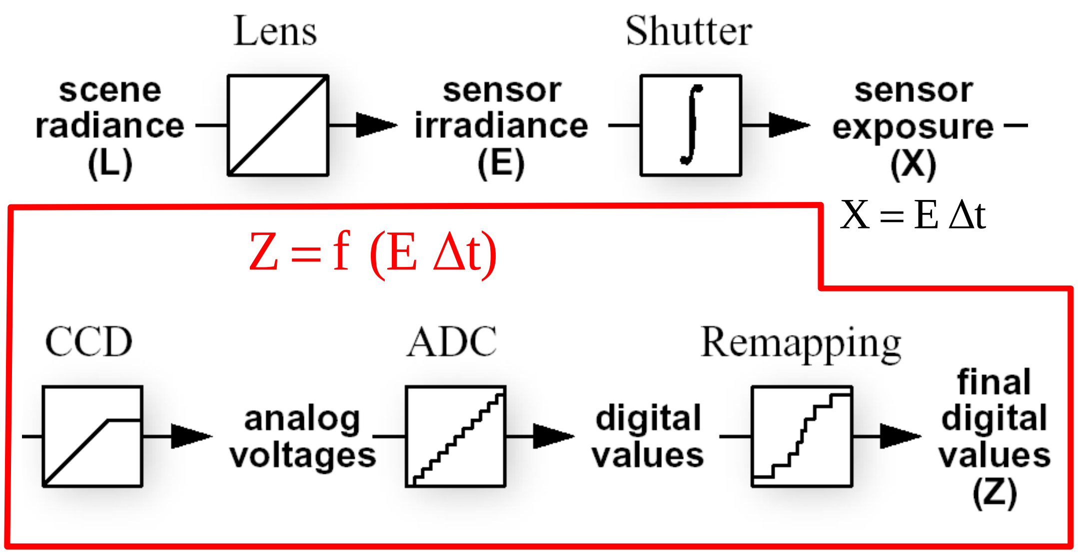
log-inverse response function:

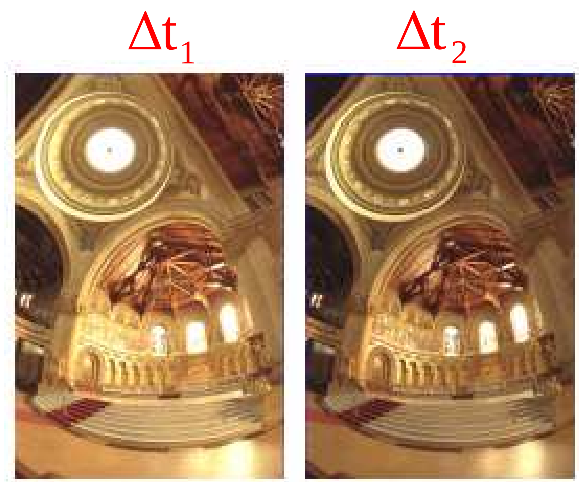
$$\boxed{g(Z) = \log(E) + \log \Delta t}$$

The log-inverse response function



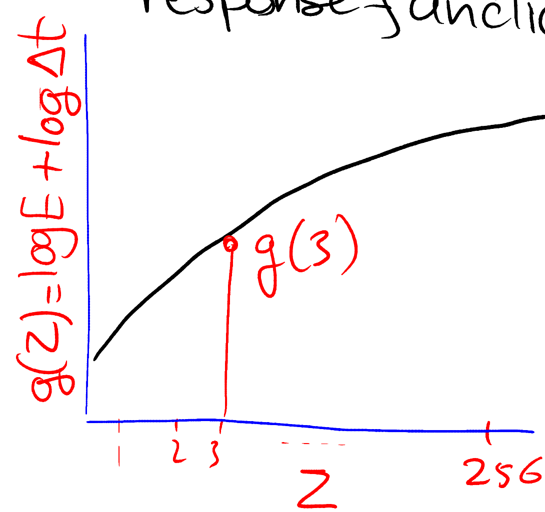
Idea #2: Compute Discrete Values of g(), Not f()



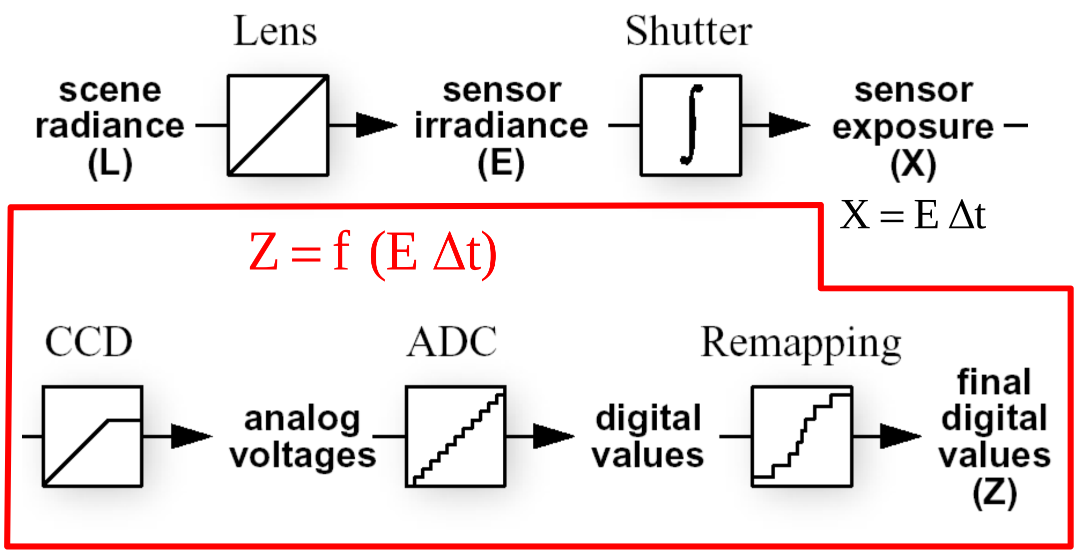


256 values of g only:
g(0), g(1), …, g(255)

The log-inverse response function



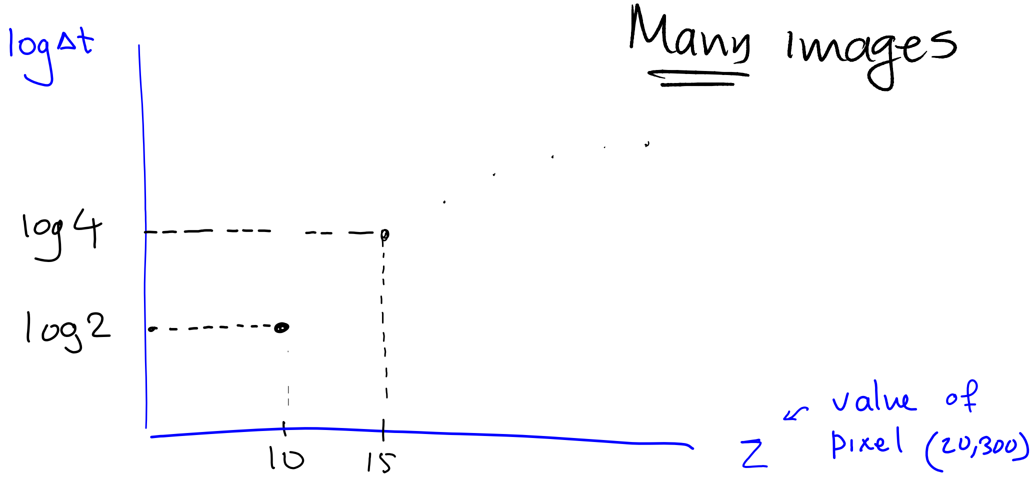
Computing The log-Inverse Response Function

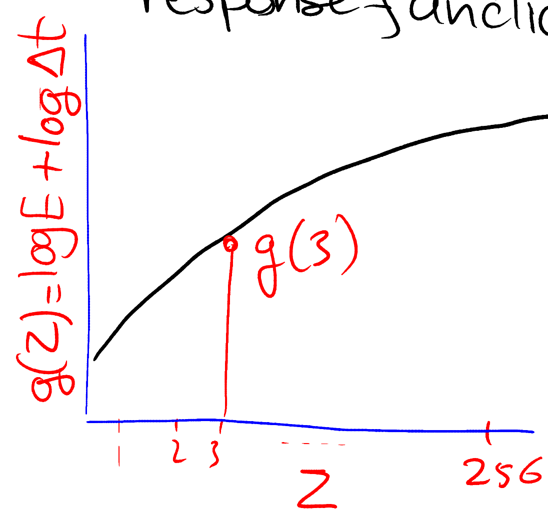


The log-inverse response function

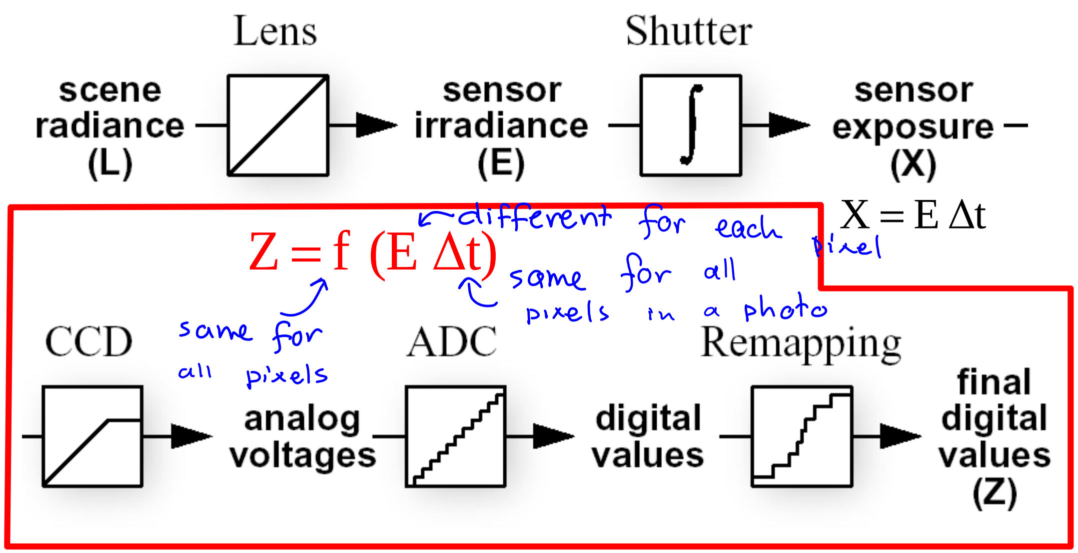
Approach #1: One pixel

Many images

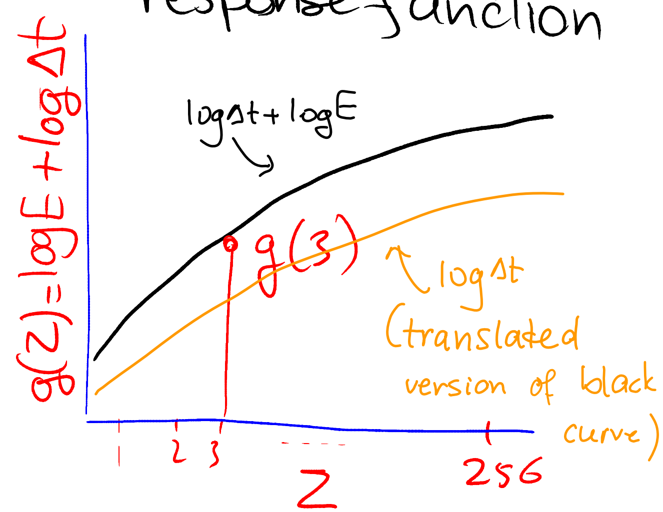




Computing The log-Inverse Response Function



The log-inverse response function



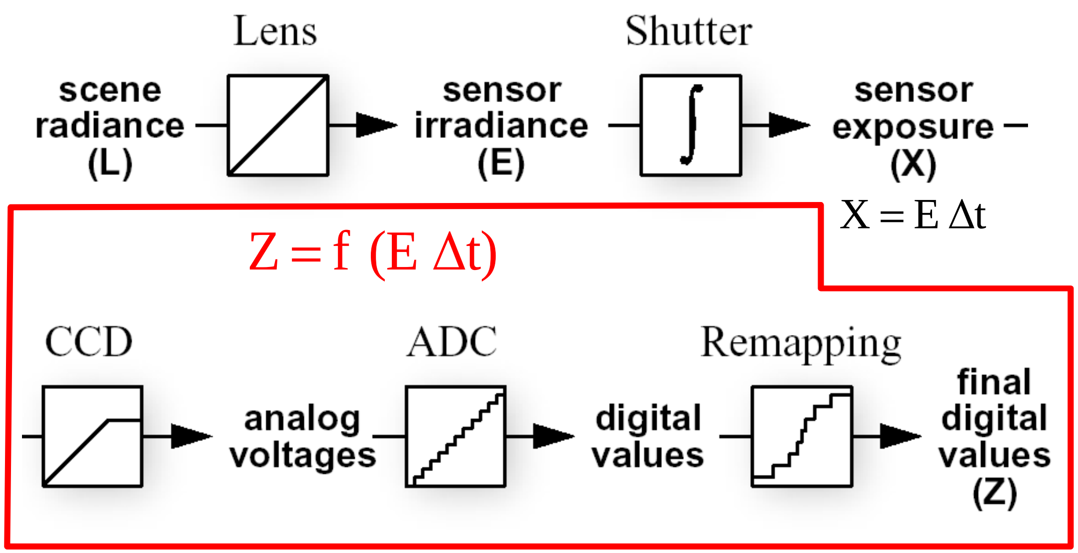
Approach #1: One pixel

Many images

1. Finely adjust Δt in range ($\frac{1}{1000}$ sec, 30 sec)
2. Plot $\log \Delta t$ as a function of the pixel's observed intensity

$\Rightarrow$ But $\log E$ is unknown! Does it matter?

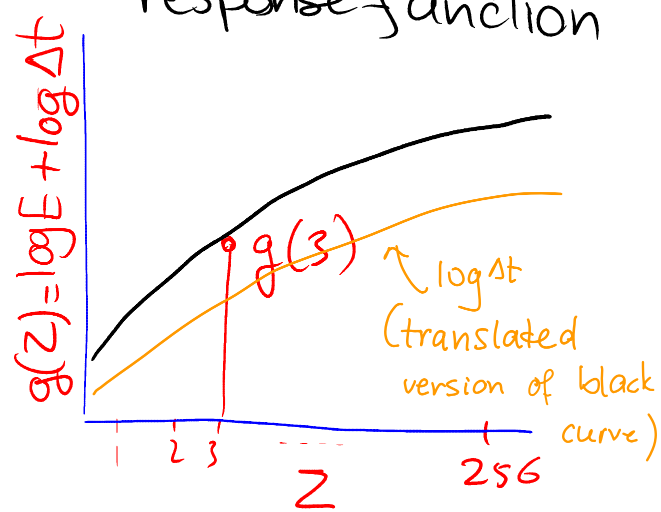
Computing The log-Inverse Response Function



Approach #1: One pixel
Many images

What are the problems with this approach? NEED LOTS OF IMAGES!
WASTED PIXELS (we only use one out of 1000s in photo)
POSSIBLE Δt VALUES DETERMINED BY CAMERA

The log-inverse response function



Computing The log-Inverse Response Function

Approach #2: Few pixels, Few images

different for each pixel

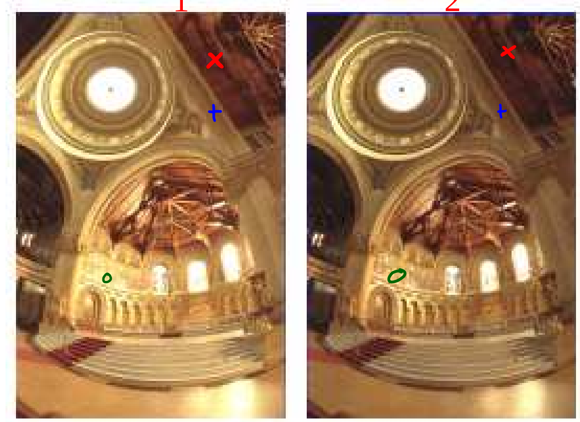
different for each pixel

$$g(Z) = \log E + \log \Delta t$$

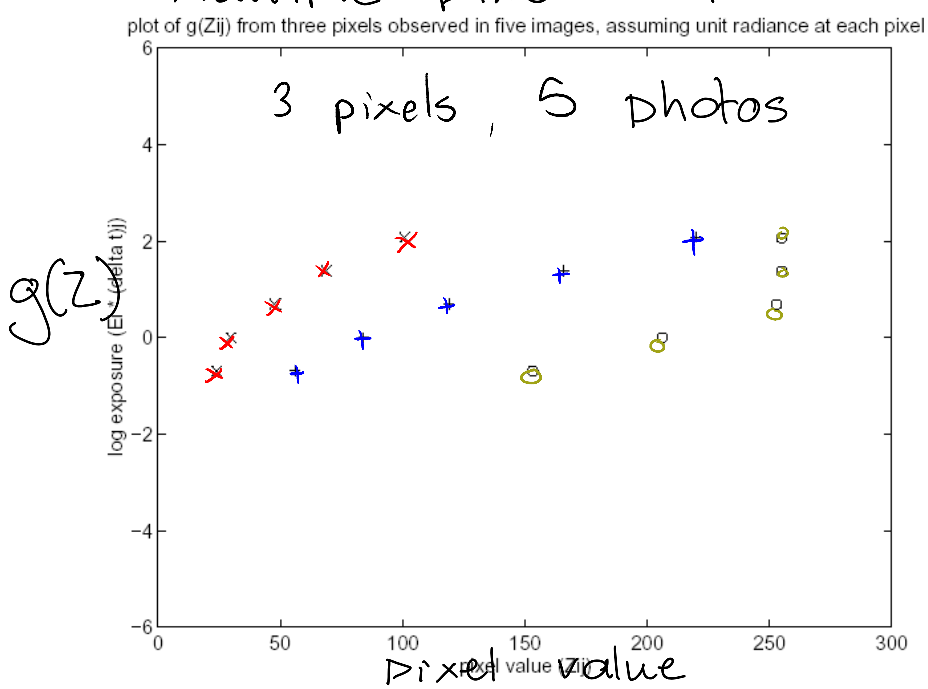
same for all pixels

same for all pixels in a photo

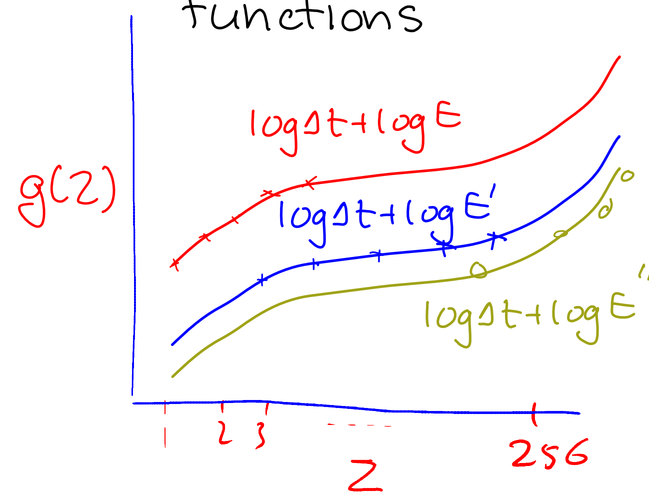
Δt_1 Δt_2



Samples of g(Z) for multiple pixels & photos



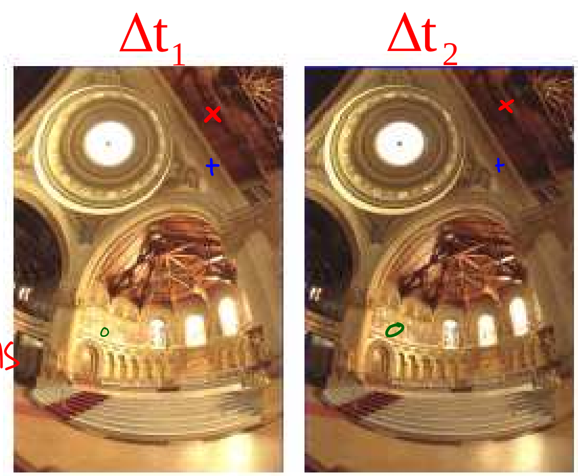
Relation between samples & the g(Z) functions



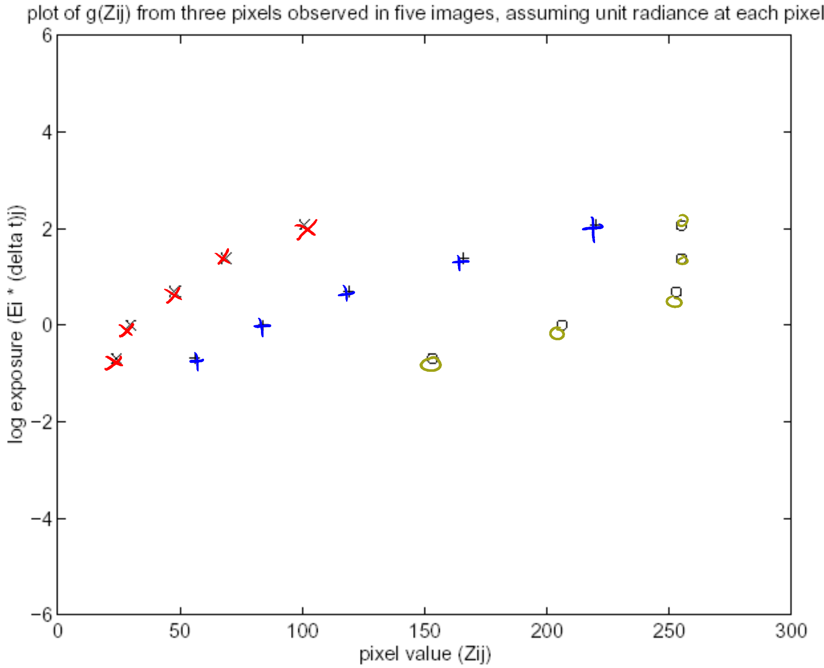
Computing The log-Inverse Response Function

Approach #2: Few pixels, Few images

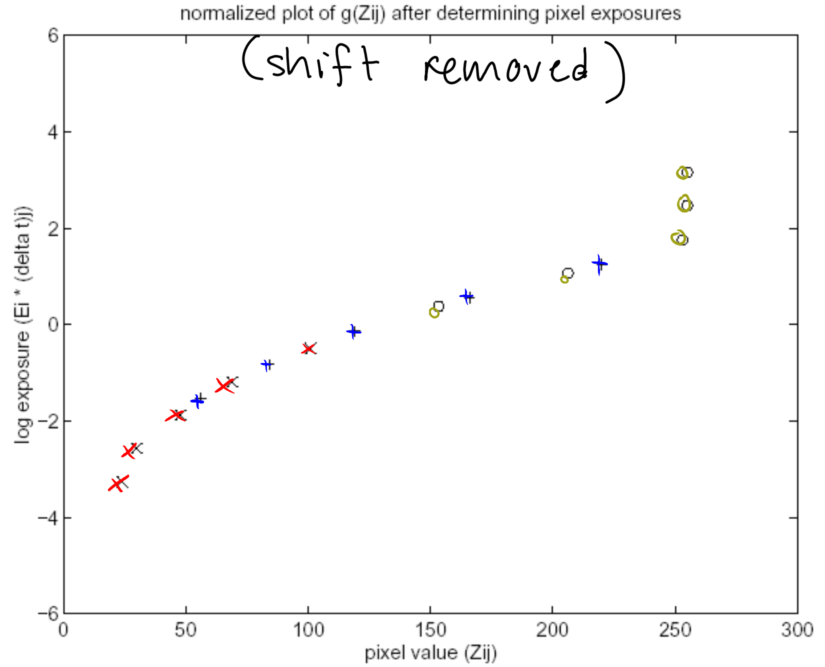
⇒ To compute the complete function, we must compute the relative vertical shift of the g( ) functions from individual pixels



3 pixels, 5 images



Computed Function



Computing The log-Inverse Response Function

Approach #2: N pixels, P images

pixel value (known)

unknown

$$g(Z_{ij}) = \log E_i + \log \Delta t_j$$

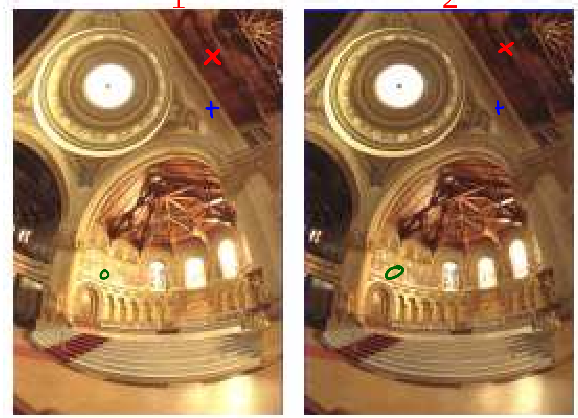
i-th pixel

In j-th image

irradiance of i-th pixel (unknown)

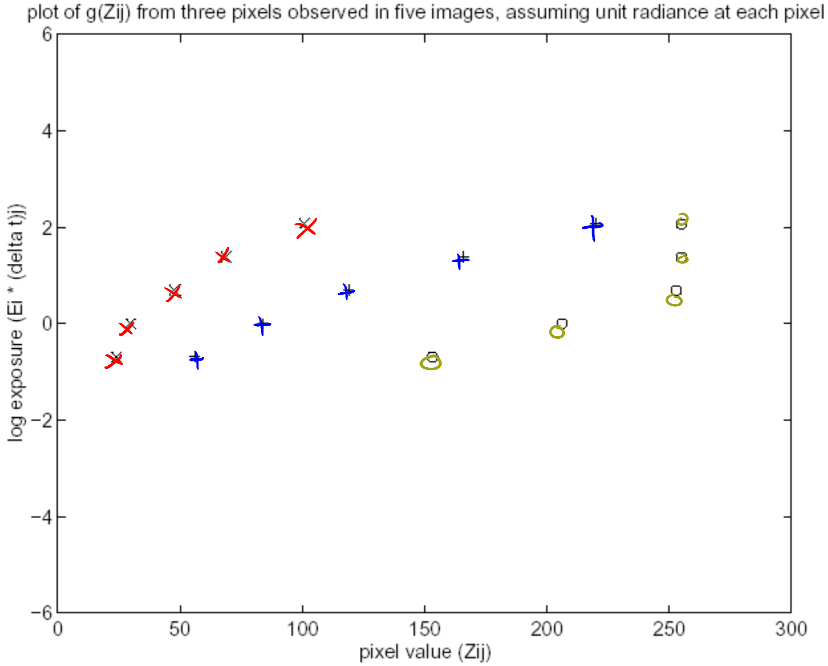
exposure interval of j-th image (known)

Δt_1 Δt_2



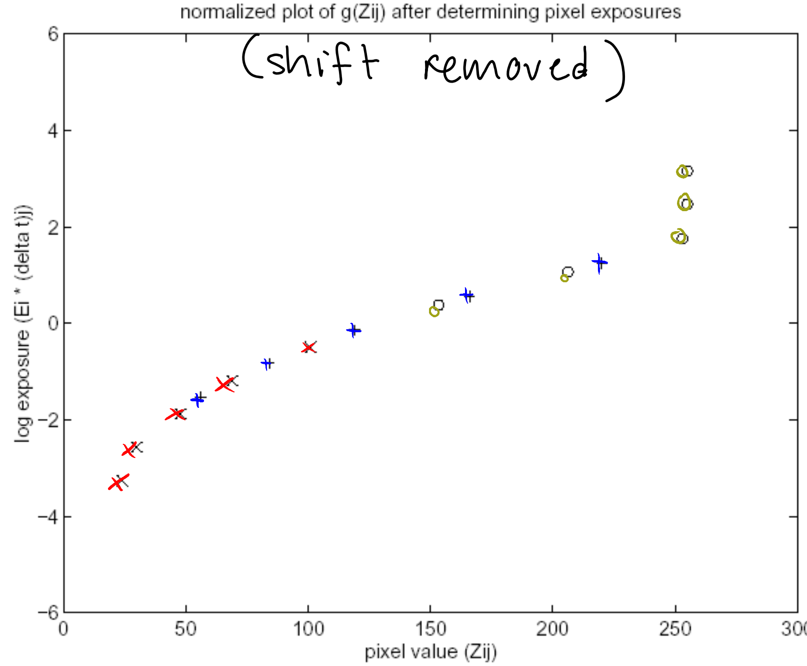
3 pixels, 5 images

plot of g(Zij) from three pixels observed in five images, assuming unit radiance at each pixel



Computed Function

normalized plot of g(Zij) after determining pixel exposures



Computing The log-Inverse Response Function

Approach #2: N pixels, P images

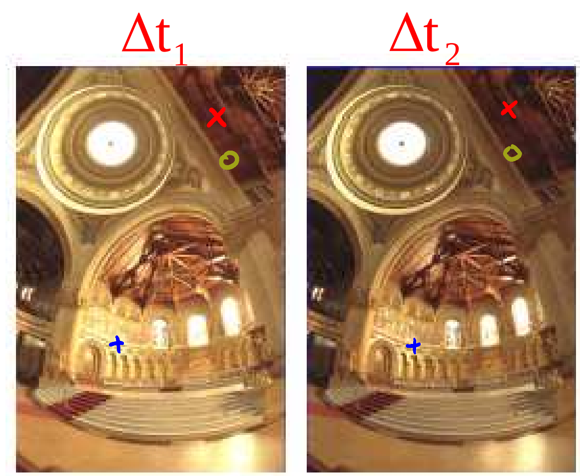
observed intensity

$$g(Z_{ij}) = \log E_i + \log \Delta t_j$$

i-th pixel

In j-th image

exposure interval of j-th image



Goal:

Compute $g(0), g(1), .. g(255)$

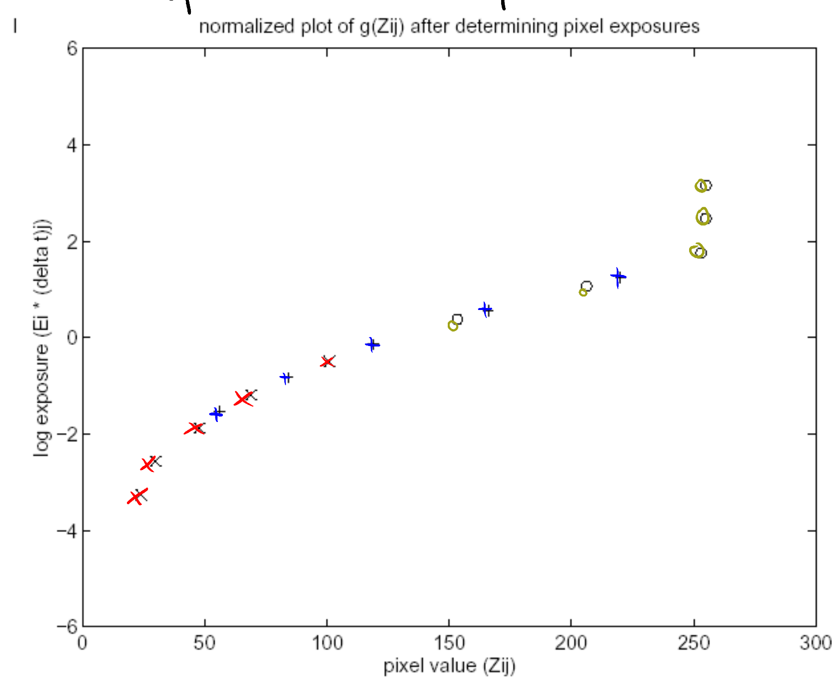
Compute $\log E_i$

Given:

N pixel intensities in

P images, known Δt_j

Computed Response Function



Computing The log-Inverse Response Function

Approach #2: N pixels, P images

observed intensity

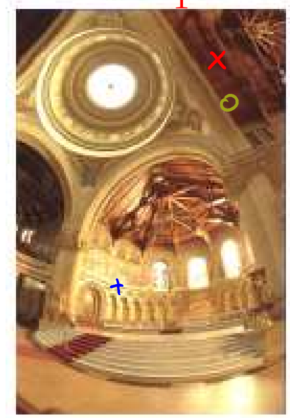
$$g(Z_{ij}) = \log E_i + \log \Delta t_j$$

i-th pixel

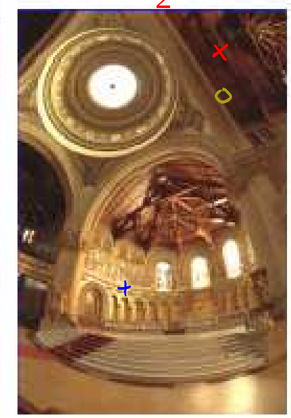
In j-th image

exposure interval of j-th image

Δt_1



Δt_2



Goal:

Compute $g(0), g(1), \ldots g(255)$

Compute $\log E_i$

Given:

N pixel intensities in P images, known Δt_j

we know that

$$\boxed{\forall_{ij} \quad g(Z_{ij}) - \log E_i = \log \Delta t_j \quad (*)}$$

$N \cdot P$ equations, $N + 256$ unknowns

Idea: Each pixel in each photo contributes one equation

Computing The log-Inverse Response Function

Approach #2: N pixels, P images

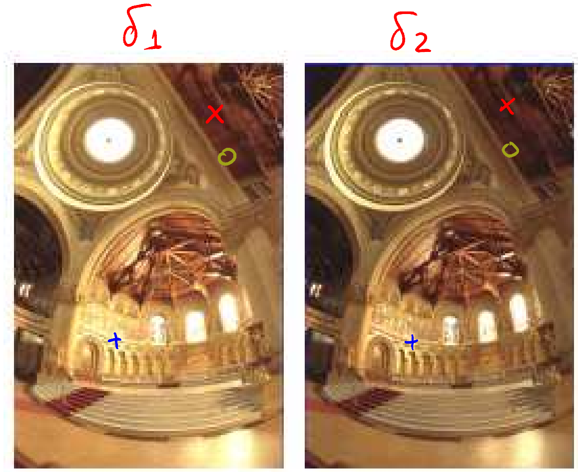
observed intensity

$$g(Z_{ij}) = \log E_i + \log \Delta t_j$$

i-th pixel

In j-th image

exposure interval of j-th image



Goal: simplified notation

Compute $\overset{g_0}{g(0)}, \overset{g_1}{g(1)}, \ldots \overset{g_{255}}{g(255)}$

Compute $\log E_i = e_i$

Given:

N pixel intensities in

P images, known Δt_j $\delta_j = \log \Delta t_j$

we know that

$$\boxed{\forall_{ij} \quad g_{Z_{ij}} - e_i = \delta_j \qquad (*)}$$

$N \cdot P$ equations, $N + 256$ unknowns

Example: Pixel 1000 in the 5th photo has intensity $Z_{1000,5} = 125$. The associated equation is

$$g_{125} - e_{1000} = \delta_5$$

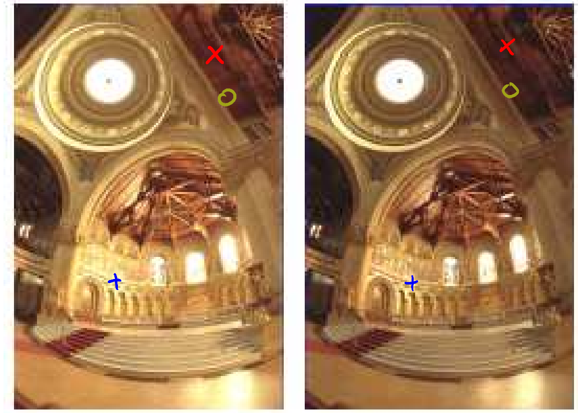
Computing The Camera Response Function

Single eq (*) in matrix form

$$\begin{array}{cccc} & \uparrow & & \\ & (z_{ij}+1)\text{-th column} & & (256+i)\text{-th column} \end{array}$$

$$\begin{bmatrix} 0 \cdots 0 & 1 & 0 \cdots 0 & -1 \cdots 0 \end{bmatrix} \begin{bmatrix} g_0 \\ \vdots \\ g_{255} \\ e_1 \\ \vdots \\ e_N \end{bmatrix} = \delta_j$$

δ_1 δ_2



Goal:

simplified notation: $g_0 \quad g_1 \quad g_{255}$

Compute $g(0), g(1), \ldots g(255)$

Compute $\log E_i = e_i$

Given:

N pixel intensities in

P images, known Δt_j $\quad \delta_j = \log \Delta t_j$

we know that

$$\forall_{ij} \quad g_{z_{ij}} - e_i = \delta_j \qquad (*)$$

$N \cdot P$ equations, $N + 256$ unknowns

Example: Pixel 1000 in the 5th photo has intensity $Z_{1000,5} = 125$. The associated equation is

$$g_{125} - e_{1000} = \delta_5$$

Computing The log-Inverse Response Function

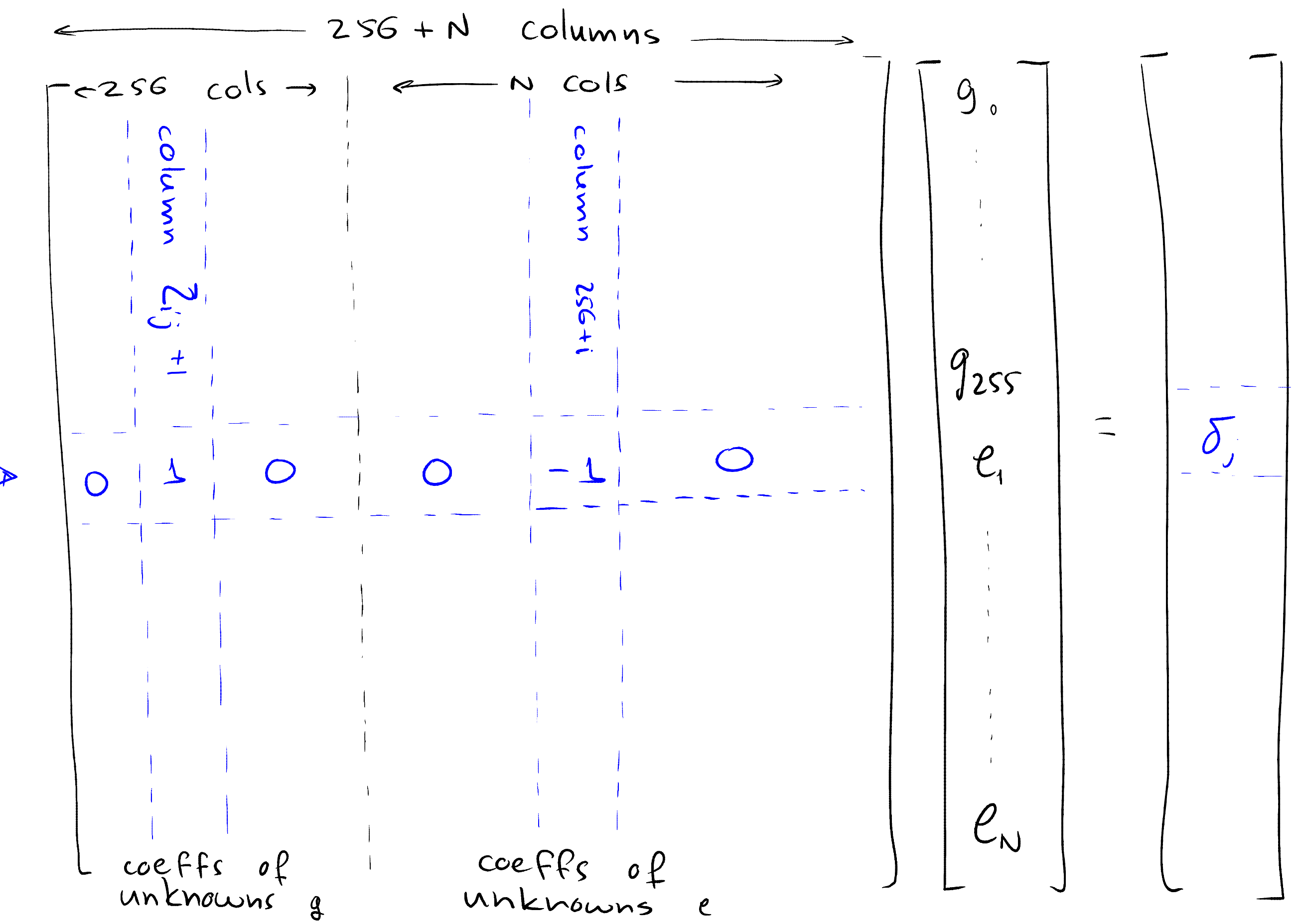
Single eq (*) in matrix form

$$\begin{array}{c} \left[\begin{array}{ccccccccc} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & -1 & \cdots 0 \end{array}\right] \\ \qquad\qquad (Z_{ij}+1)\text{-th column} \qquad (256+i)\text{-th column} \end{array} \begin{bmatrix} g_0 \\ \vdots \\ g_{255} \\ e_1 \\ \vdots \\ e_N \end{bmatrix} = \delta_j$$

$$\boxed{\forall i,j \qquad g_{Z_{ij}} - e_i = \delta_j \qquad (*)}$$

If pixel i has intensity Z_{ij} in image j

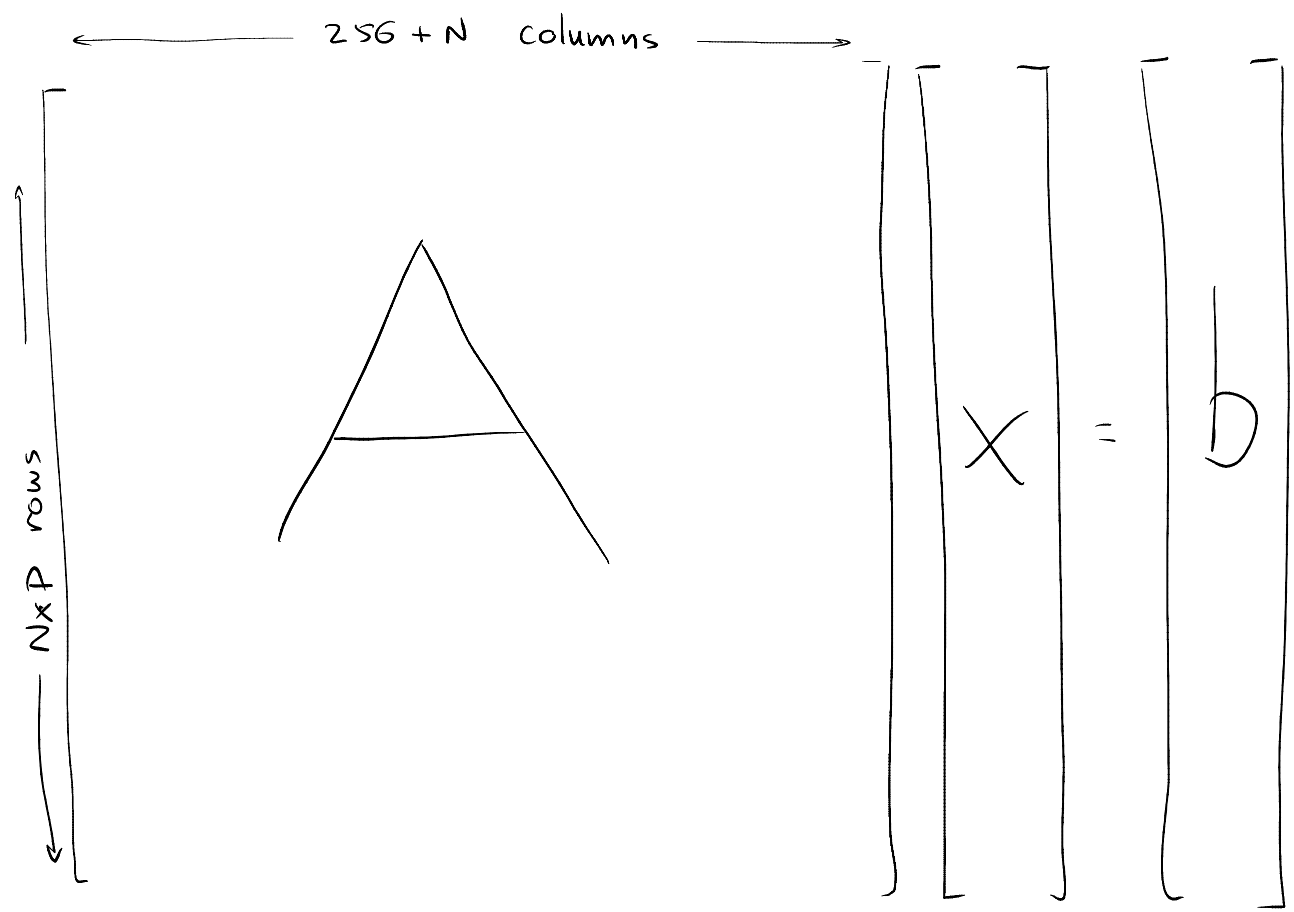
(ij)-th row



Computing The log-Inverse Response Function

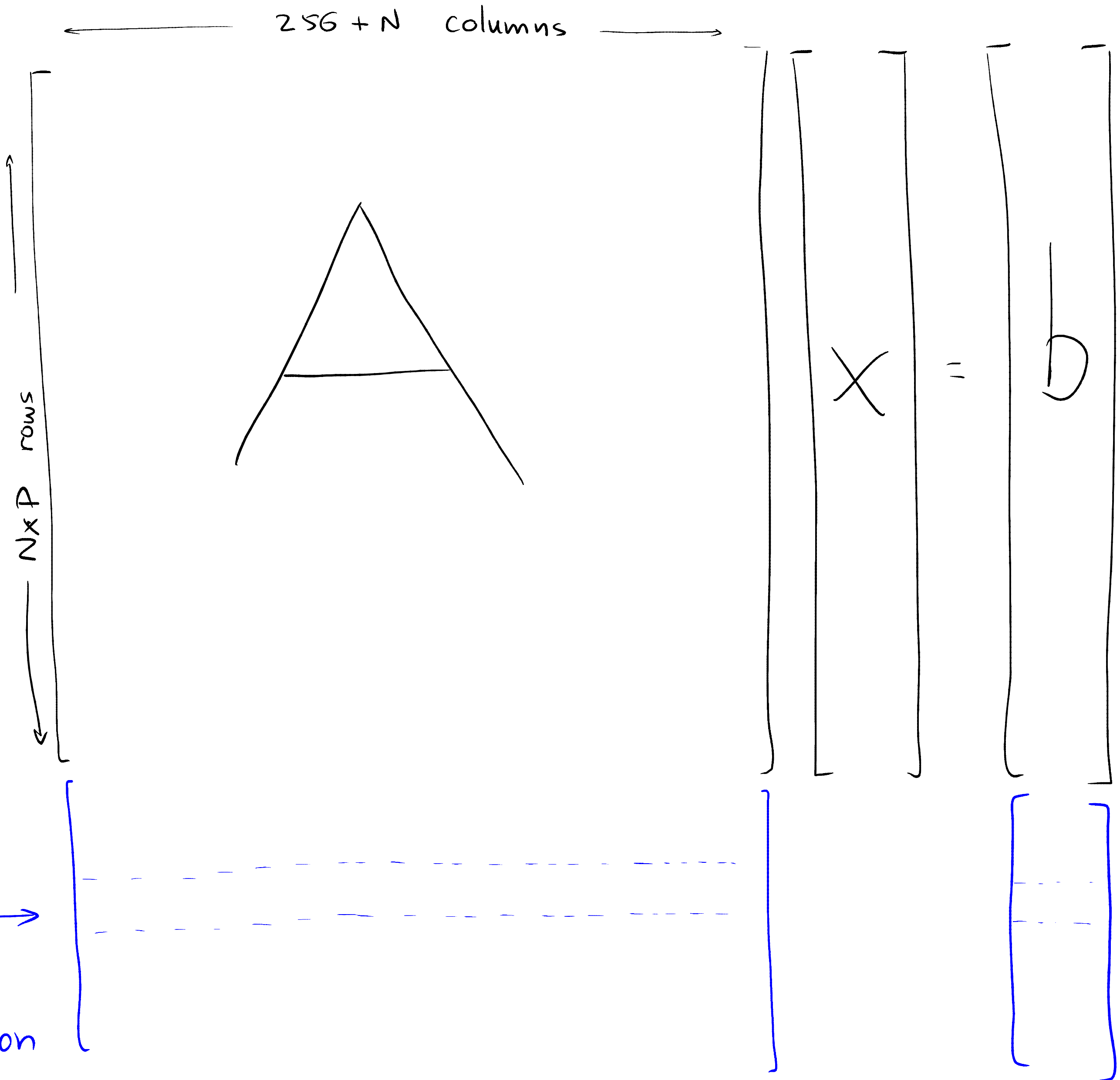
- To solve this system, matrix A must be non-singular
- We can achieve this by choosing N, P so that

  $NP \gg 256 + N$ (e.g. 1000 pixels, 20 exposure settings)



Computing The log-Inverse Response Function

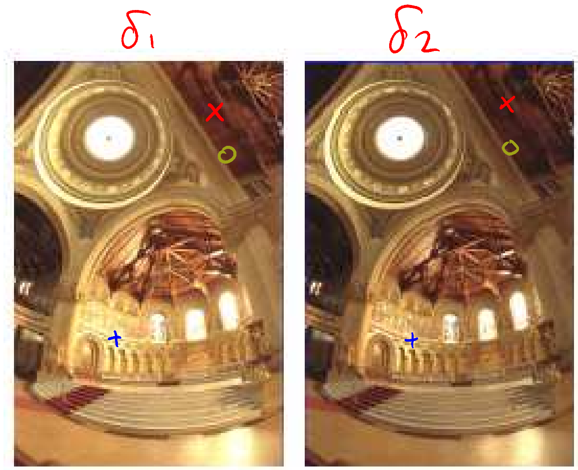
- In practice, we can add more constraints (i.e. equations) between the unknowns to account for pixel noise and to ensure the response curve is smooth

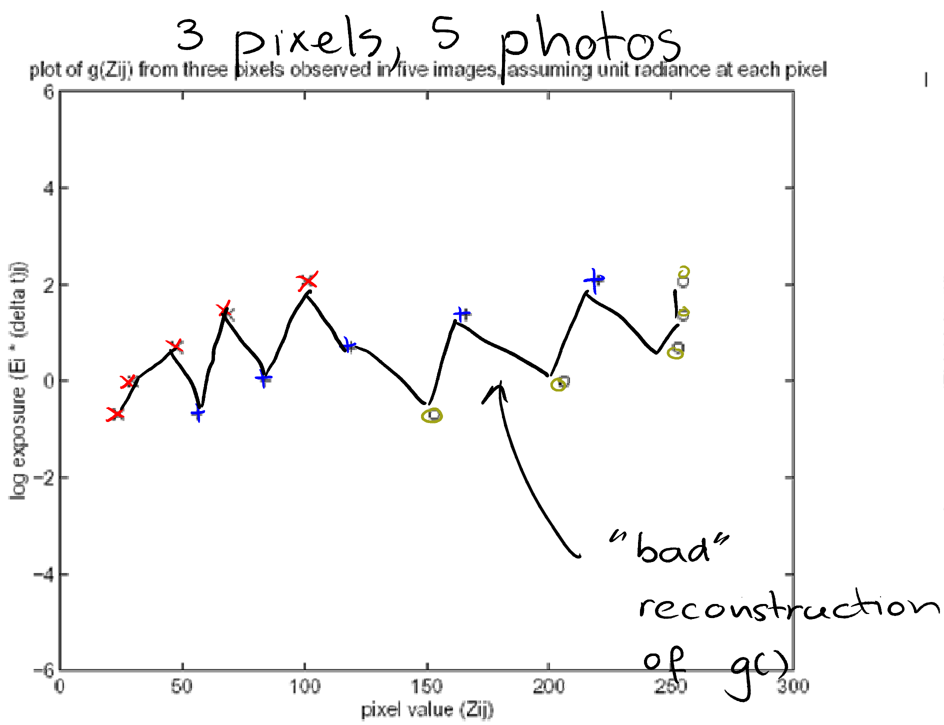


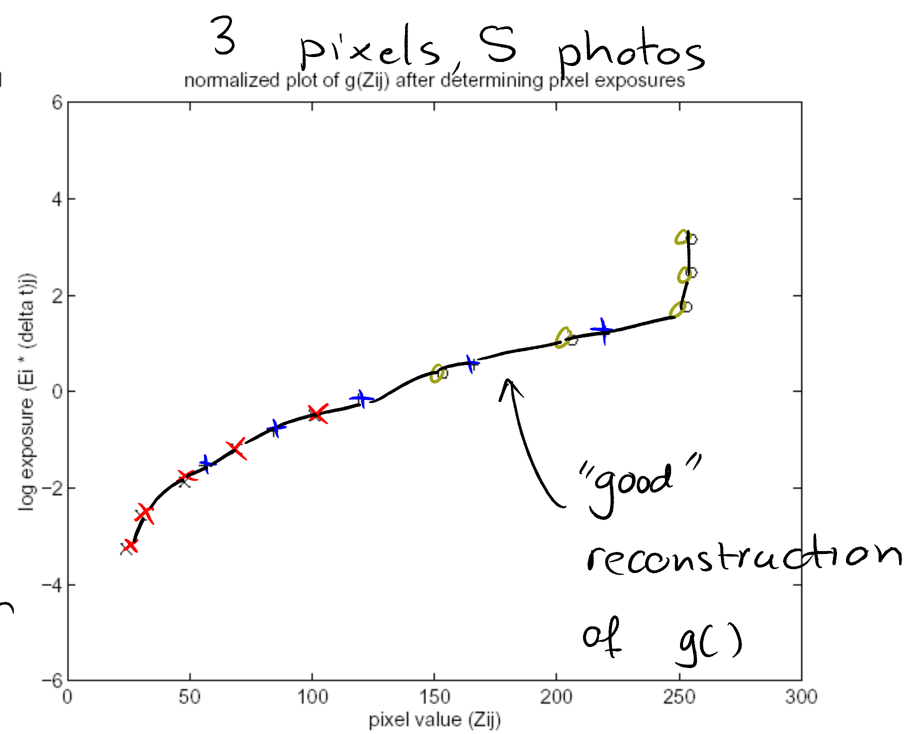
row representing one additional constraint equation

Smoothness Constraints (aka Regularization)

- Idea: The camera response function changes smoothly in real cameras
- We add more Eqs to enforce smoothness







Smoothness Constraints (aka Regularization)

- Idea: The camera response function changes smoothly in real cameras
- We add more Eqs to enforce smoothness

- Intuition: Force near-constant rate of change:

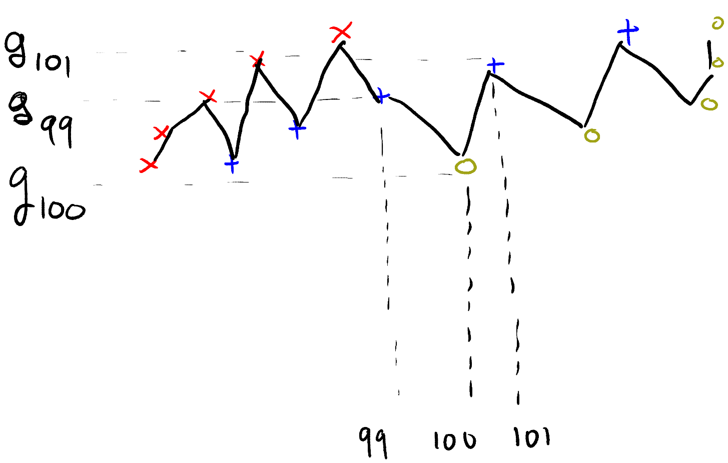
$$g_{100} - g_{99} \simeq g_{101} - g_{100}$$

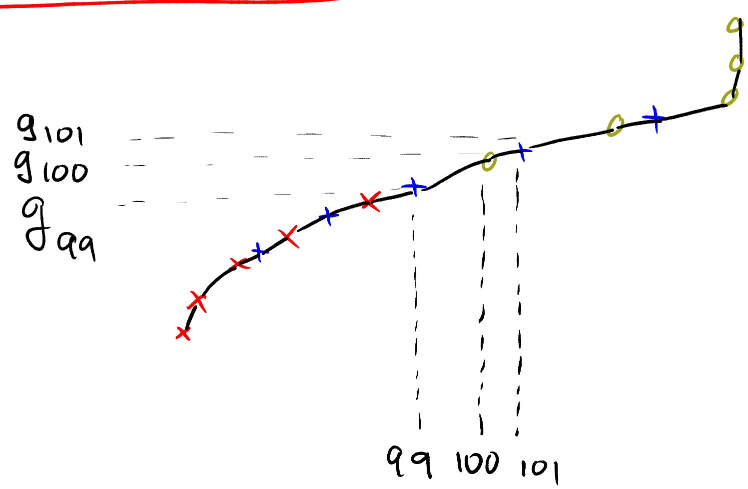
$$\Longleftrightarrow$$

$$2g_{100} - g_{99} - g_{101} \simeq 0$$

Add these Eqs to system:

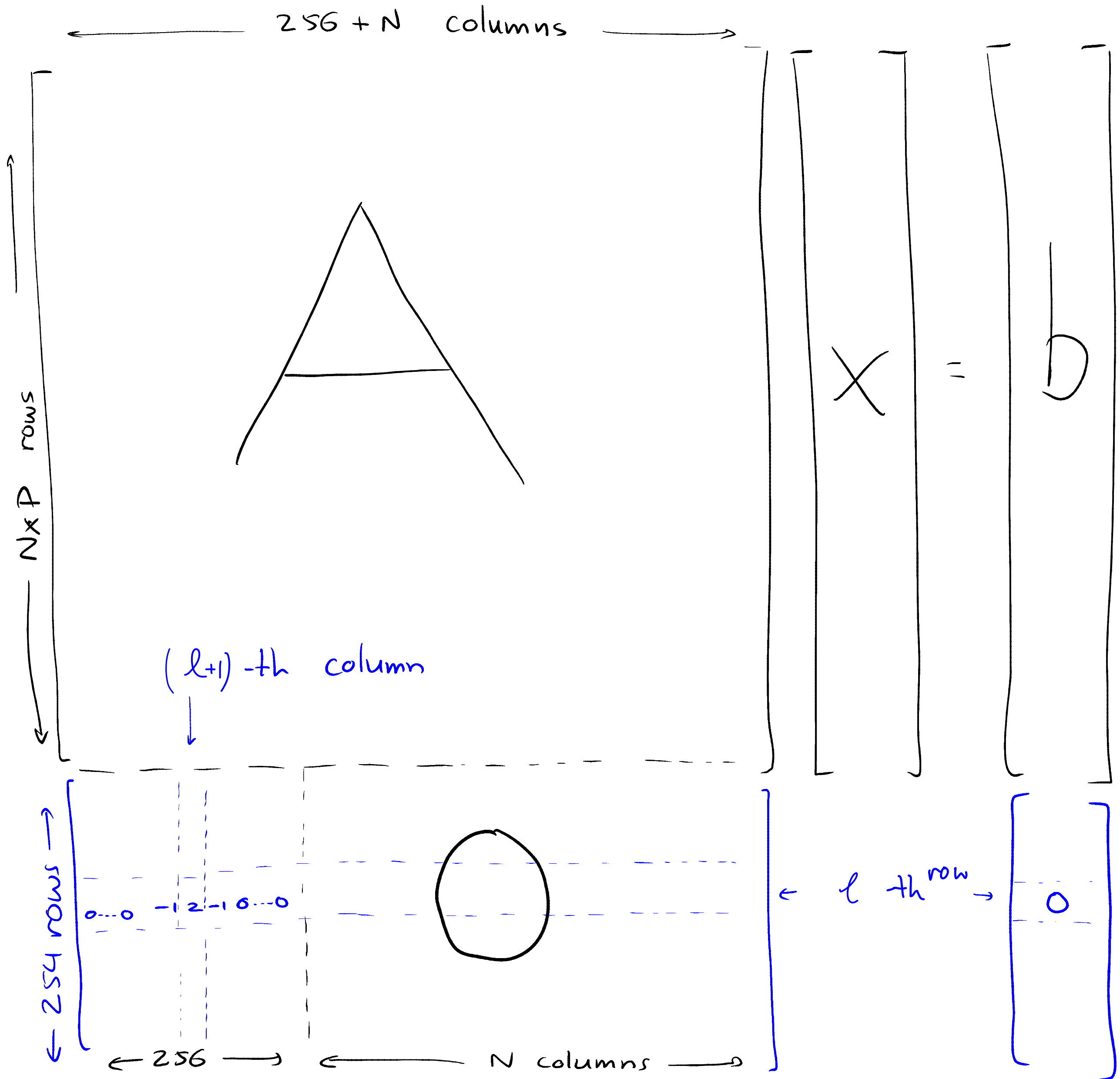
$$(l = 1, \ldots, 254) \quad 2\,g_l - g_{l+1} - g_{l-1} = 0 \qquad (**)$$





Smoothness Constraints (aka Regularization)

Original equations



Smoothness Equations

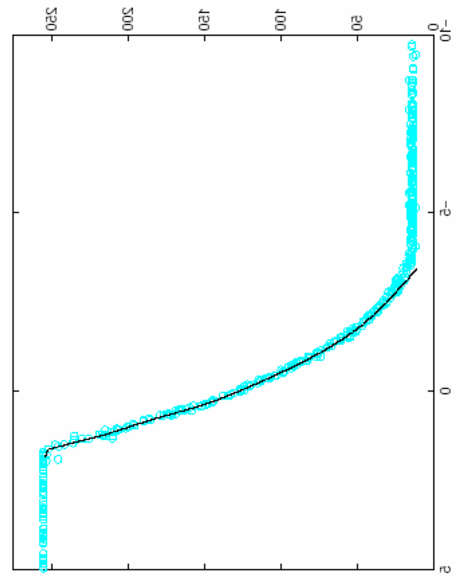
$2g_l - g_{l+1} - g_{l-1} = 0$

Computing The log-Inverse Response Function

Figure 6: *Sixteen photographs of a church taken at 1-stop increments from 30 sec to* $\frac{1}{1000}$ *sec. The sun is directly behind the rightmost stained glass window, making it especially bright. The blue borders seen in some of the image margins are induced by the image registration process.*

Result for red pixels

(from Debevec & Malik, 1997)



An Application of HDR Photography…

Light probe:

HDR photo of a reflective sphere

Useful for illuminating CG with natural light

For more information see http://www.debevec.org/Probes/

Rendering With Natural Light

Short Film from the SIGGRAPH Animation Theater

http://www.debevec.org/RNL/

- Uses a light probe to illuminate a synthetically generated scene

Aside: Representing a Linear Eq in Matrix Form

- Suppose we have N unknowns $x_1, \dots, x_N$ and an equation of the form

known quantities

$$a_1 x_1 + a_2 x_2 + \cdots + a_N x_N = b \qquad (**)$$

- Definitions

N-dimensional ROW vector is just an $1 \times N$ matrix

$$[\underbrace{\qquad\qquad\qquad}_{N \text{ elements}}]$$

N-dimensional COLUMN vector is just an $N \times 1$ matrix

$$\left.\begin{bmatrix} \\ \\ \\ \end{bmatrix}\right\} N \text{ elements}$$

- Convention:

Use a ROW vector to represent known coefficients

$$[a_1 \quad a_2 \quad \cdots \quad a_N]$$

Use a COLUMN vector to represent unknowns

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

Aside: Representing a Linear Eq in Matrix Form

- Suppose we have N unknowns $x_1, \ldots, x_N$ and an equation of the form

known quantities

$$a_1 x_1 + a_2 x_2 + \cdots + a_N x_N = b \qquad (**)$$

- Eq (**) in matrix form:

$$\boxed{\begin{bmatrix} a_1 & a_2 & \cdots & a_N \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = b \iff \sum_{j=1}^{N} a_j x_j = b}$$

- Convention:

Use a Row vector to represent known coefficients

$$\begin{bmatrix} a_1 & a_2 & \cdots & a_N \end{bmatrix}$$

Use a COLUMN vector to represent unknowns

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

Aside: Representing Linear Eqs in Matrix Form

- Suppose we have N unknowns $x_1, \ldots, x_N$ and a <u>system</u> of M equations

known quantities

$$(i = 1, \ldots, M) \quad a_{i1} x_1 + a_{i2} x_2 + \cdots + a_{iN} x_N = b_i \qquad (***)$$

$$\begin{array}{lll}
i=1 & \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \end{bmatrix} & = b_1 \\
i=k & \begin{bmatrix} a_{k1} & a_{k2} & & a_{kN} \end{bmatrix} & = b_k \\
i=M & \begin{bmatrix} a_{M1} & a_{M2} & & a_{MN} \end{bmatrix} & = b_M
\end{array}
\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

$$\Longleftrightarrow$$

$$(i = 1, \ldots, M) \quad \sum_{j=1}^{N} a_{ij} x_j = b_i$$

Aside: Representing Linear Eqs in Matrix Form

- Suppose we have N unknowns $x_1, \ldots, x_N$ and a system of M equations

known quantities

$$(i=1,\ldots,M) \qquad a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{iN}x_N = b_i \qquad (***)$$

$$\begin{array}{ll}
i=1 & \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \end{bmatrix} \\
i=k & \begin{bmatrix} a_{k1} & a_{k2} & & a_{kN} \end{bmatrix} \\
i=M & \begin{bmatrix} a_{M1} & a_{M2} & & a_{MN} \end{bmatrix}
\end{array}
\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}
\begin{array}{l} = b_1 \\ = b_k \\ = b_M \end{array}$$

$$\iff$$

$$(i=1,\ldots,M) \quad \sum_{j=1}^{N} a_{ij}x_j = b_i$$

- System (***) in matrix form:

$$\begin{array}{l}\\ \text{k-th equation} \\ \\ \end{array}$$

$$M \text{ rows} \left\{ \underbrace{\begin{bmatrix} a_{11} & \cdots & a_{1N} \\ \boxed{a_{k1} \cdots} & & \boxed{a_{kN}} \\ a_{M1} & \cdots & a_{MN} \end{bmatrix}}_{N \text{ columns } (= \#\text{unknowns})} \right. \boxed{\begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}} = \begin{bmatrix} b_1 \\ \boxed{b_k} \\ b_M \end{bmatrix}$$

$$\iff Ax = b$$