Image Processing

CS475 / 675, Fall 2016

Siddhartha Chaudhuri

Image as Signal

- An image can be thought of as
  - Piecewise constant function, *or*
  - Uniform sampling of some underlying function

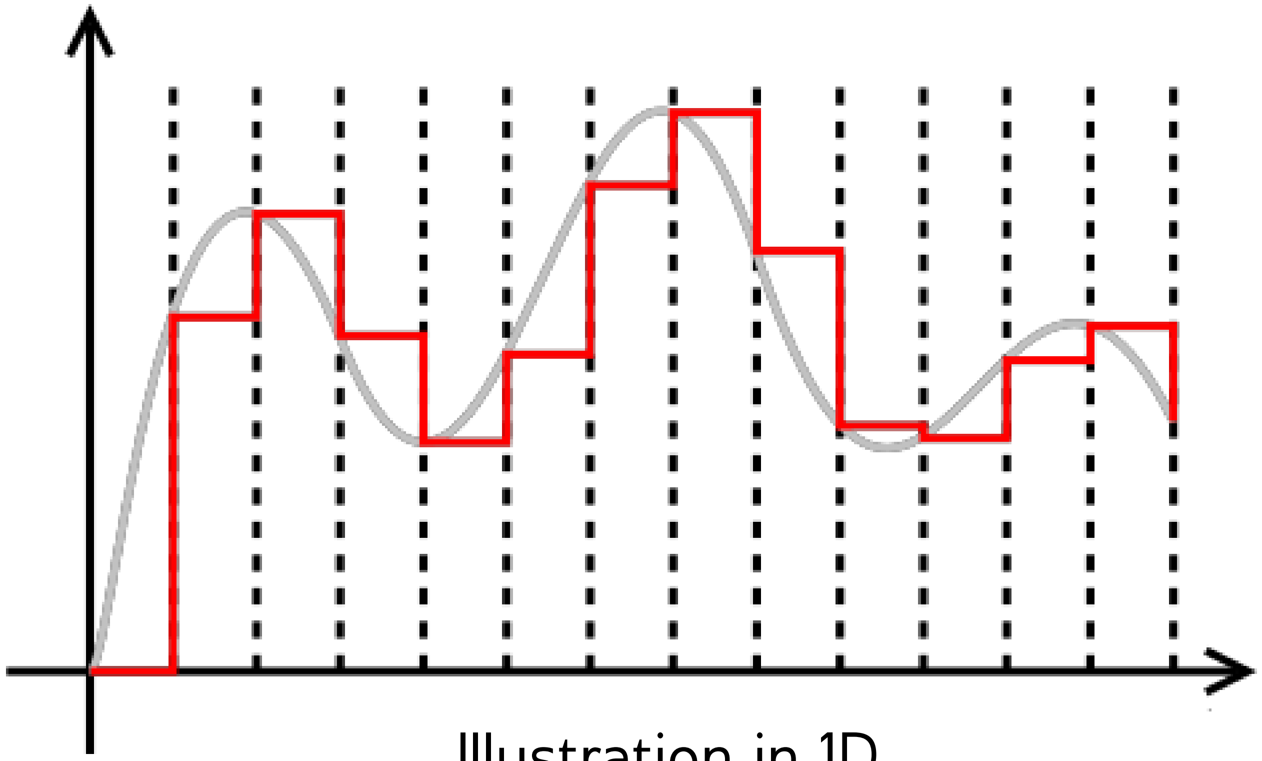
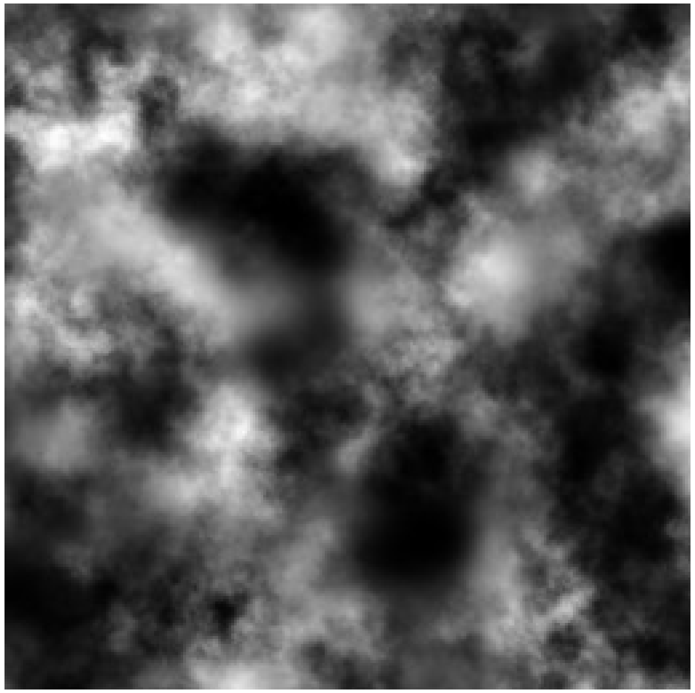
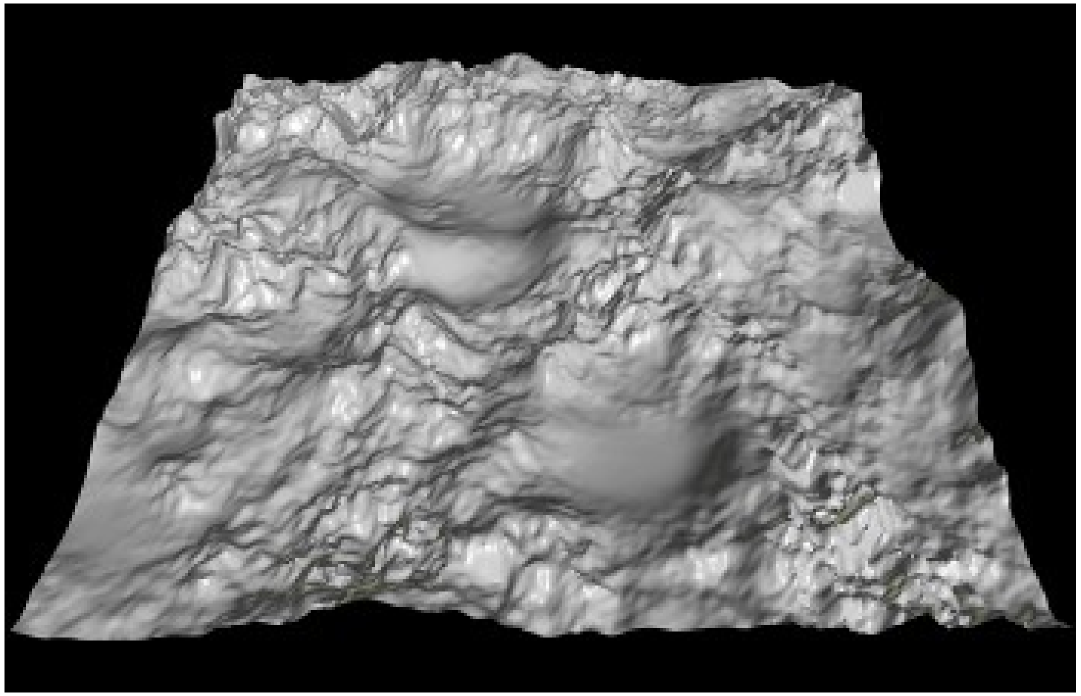


Illustration in 1D

(Wikipedia)

Image as Signal


2D heightmap image

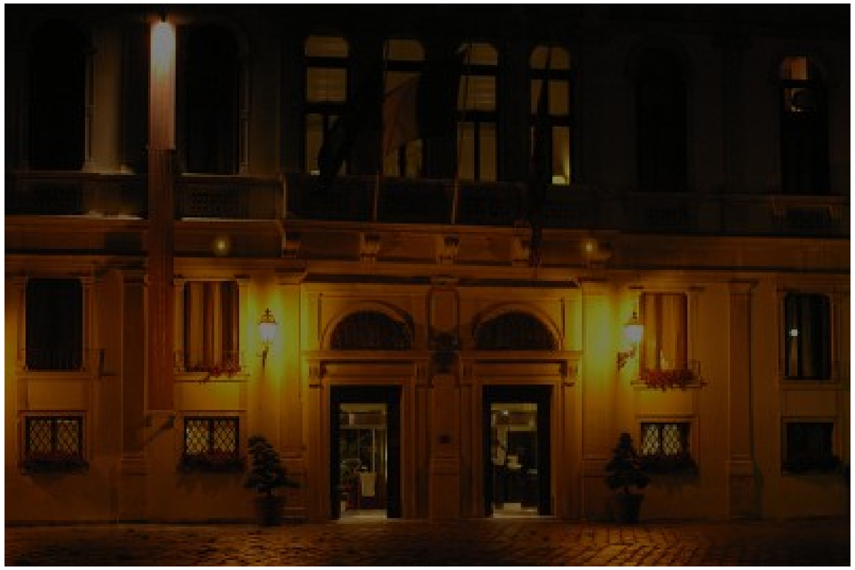

Visualized in 3D as heightfield

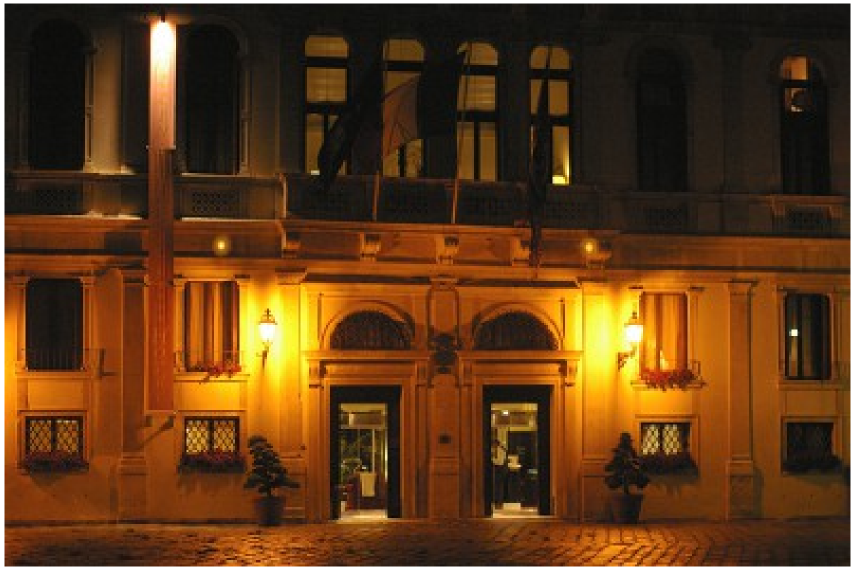
(Wikipedia)

Color Space Operations

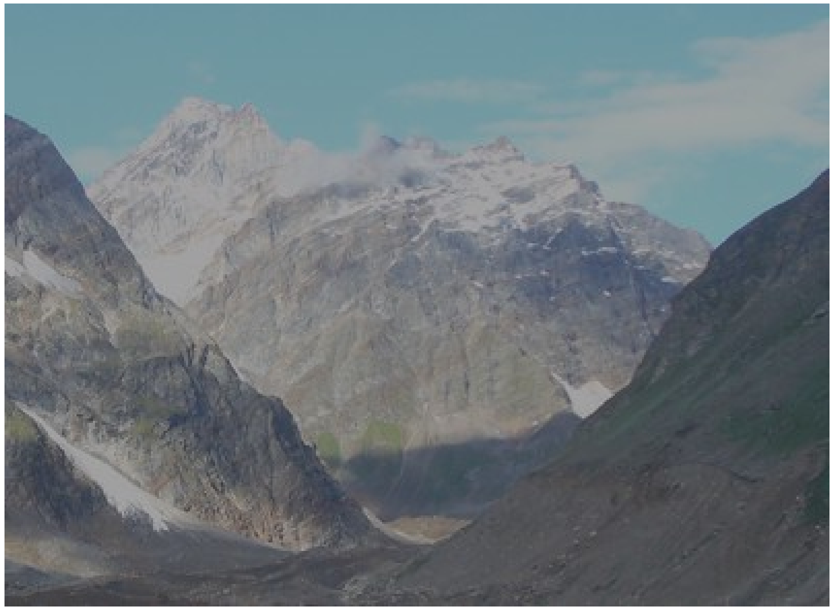
- Change pixel color based only on the current value at that position
  - No looking at any other pixels
- View as composition of functions
  - Image function $f : \mathbb{R}^d \to \mathrm{C}$ maps position $\mathbf{u} \in \mathbb{R}^d$ to color $f(\mathbf{u}) \in \mathrm{C}$
  - Let's apply the function $g : \mathrm{C} \to \mathrm{C}'$ to the image
    - e.g. g increases brightness (luminance)
    - C' may or may not be the same color space as C
  - The color at $\mathbf{u}$ is then $g(f(\mathbf{u}))$
  - In other words, the image is now the function $g \circ f$

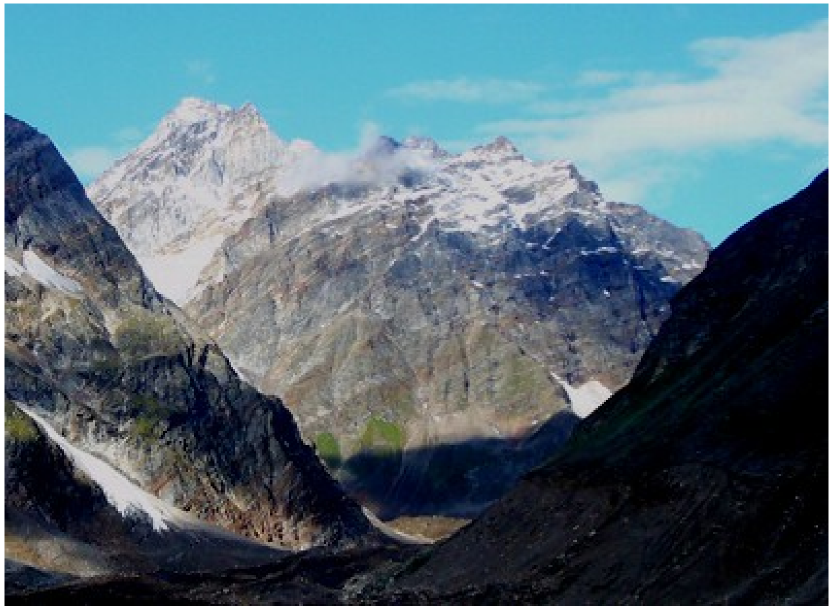
Example: Increase Brightness


Source


Result

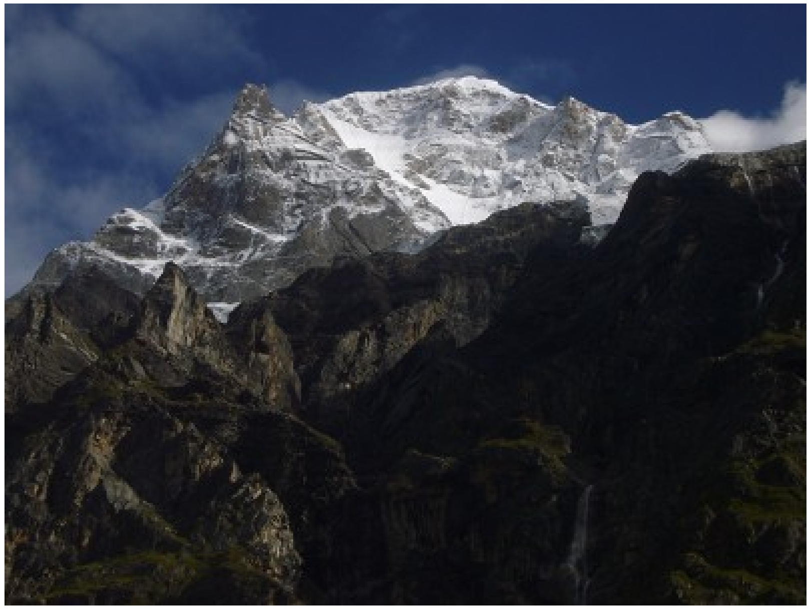
Example: Increase Contrast

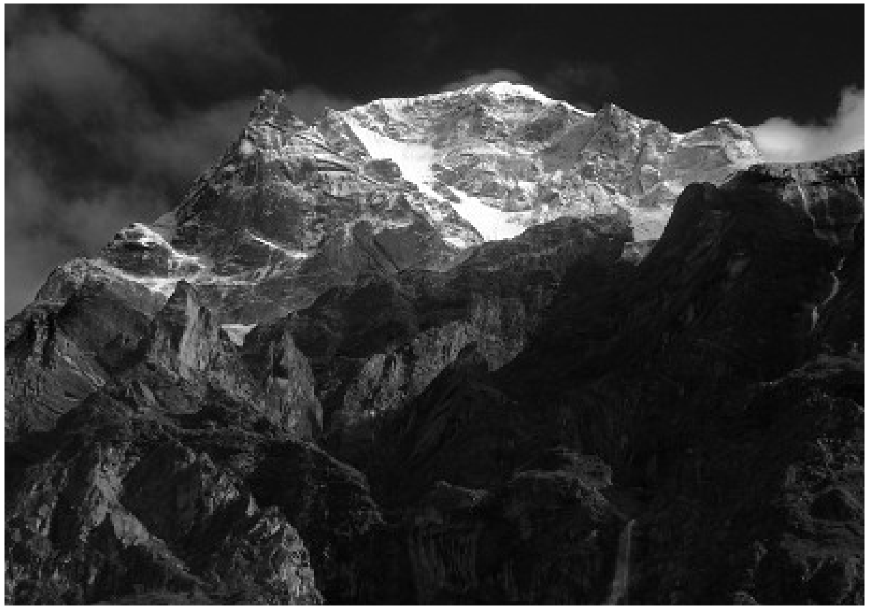

Source


Result

Note: The "levels" control can be used for a similar effect

Example: Desaturation ($C' \neq C$)

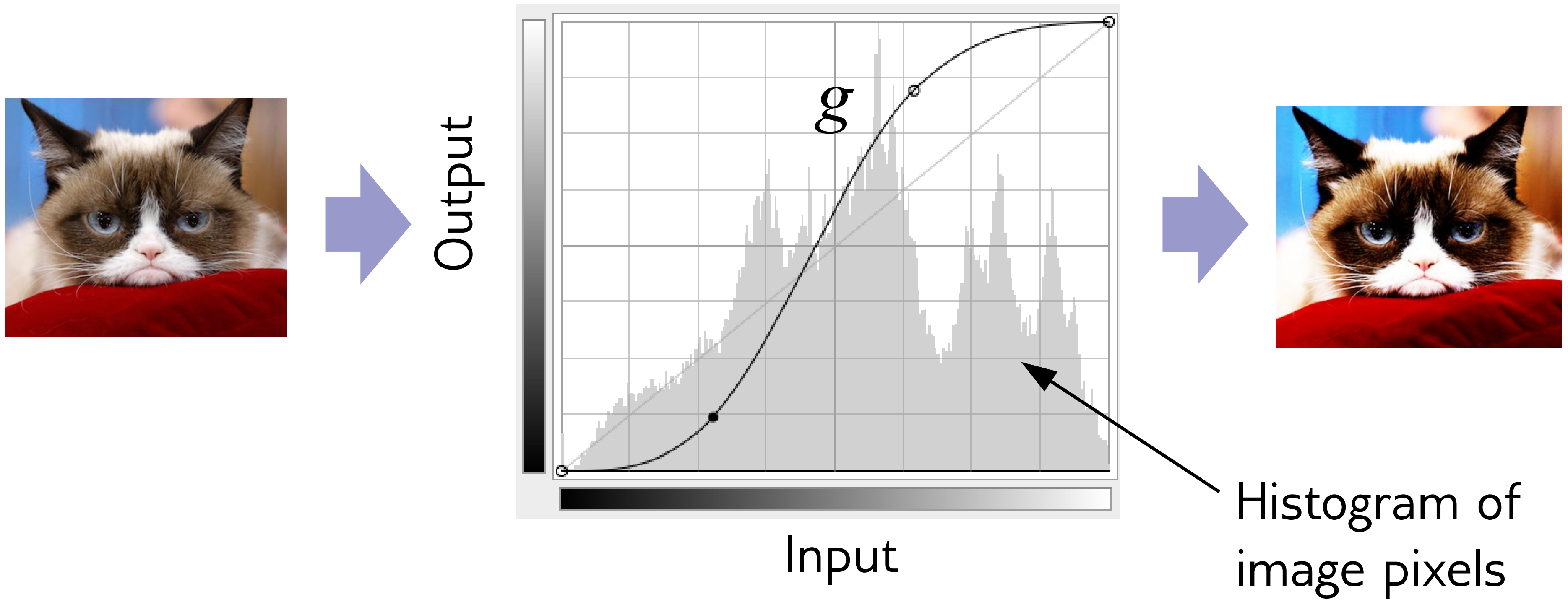

Source


Result

A common mapping is $Y = 0.3R + 0.59G + 0.11B$

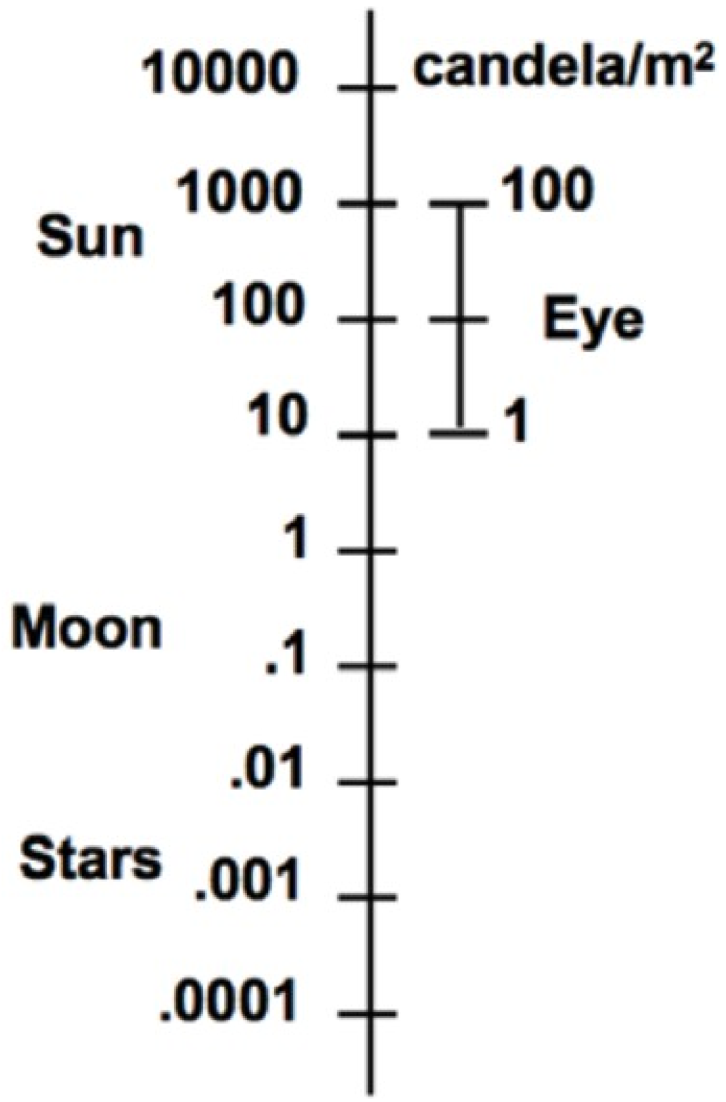
Curves: The Swiss Army Knife

- Visual manipulation of the function g
- Offers most fine-grained control
- (Demo in GIMP)

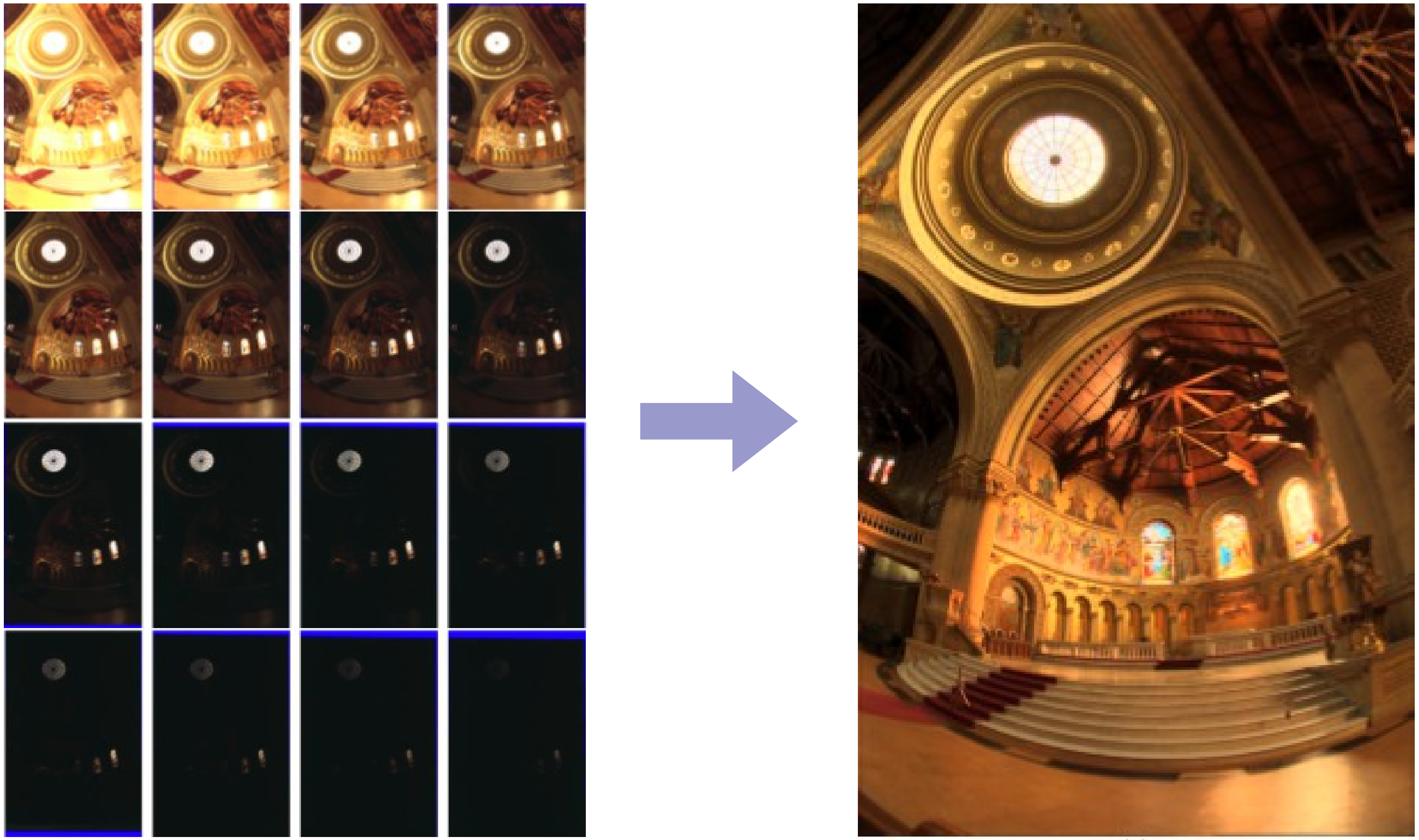


High Dynamic Range Images (HDR)

- The real world contains a far greater range of intensities (dynamic range) than normal displays/printers can reproduce
- Solution:
  - Capture a large range
    - Normal cameras are limited (best is about 5,000 : 1), so we must use tricks
  - Compress to displayable range
    - This is called *tone mapping*



High Dynamic Range Images (HDR)



16 photographs of Stanford Memorial Church, each with double the exposure of the previous one, merged into an HDR image that shows detail in both shadows and highlights

(Debevec, Malik and Ward)

Capturing an HDR Image

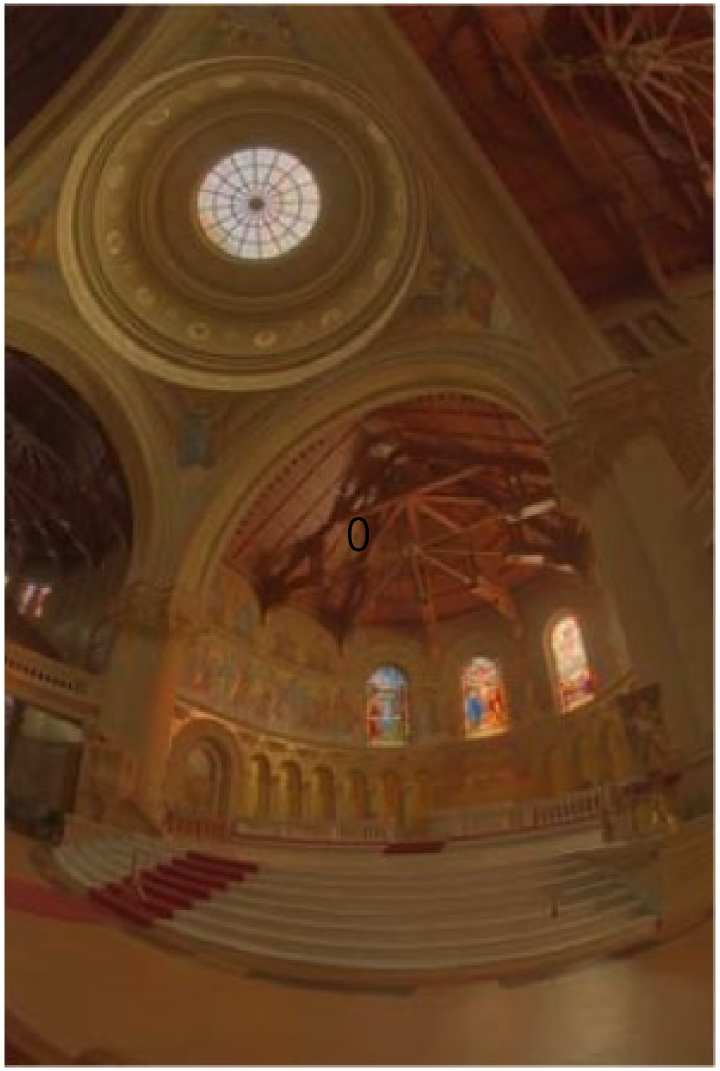
- Take many images from the same location, with different exposures
  - Vary shutter-speed or sensitivity, not aperture! (we don't want different amounts of blur in different images)
  - Longer exposures capture shadow detail, but highlights are clipped to white
  - Shorter exposures capture highlight detail, but shadows are clipped to black
- Merge into a single floating-point image that represents the entire range of intensities

Visual Response to Dynamic Range

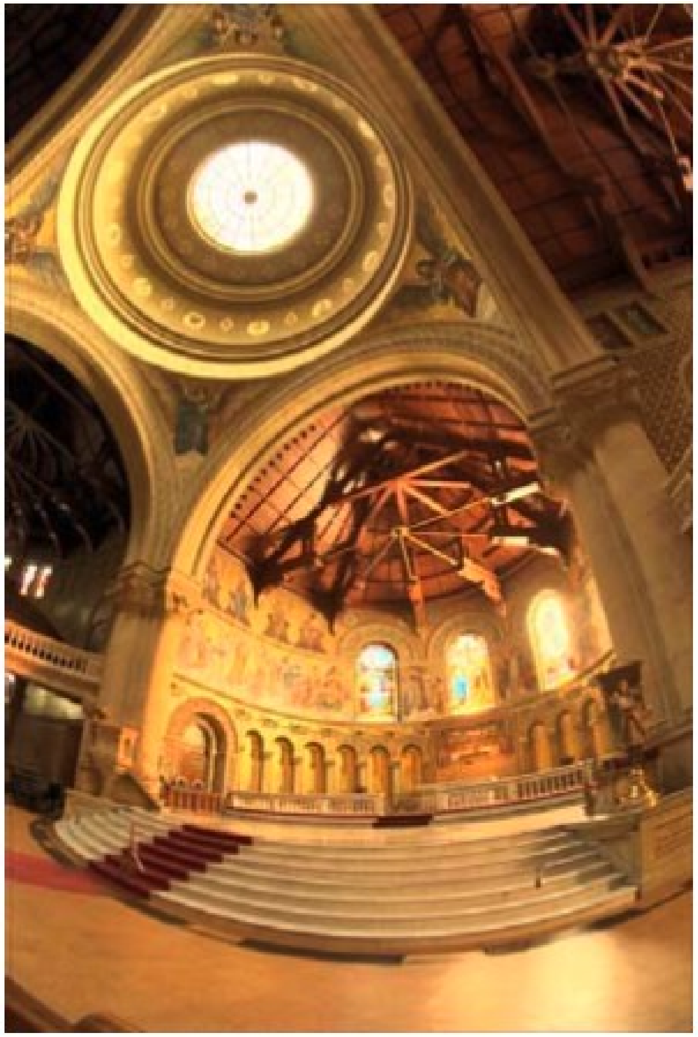
- Our eyes have roughly logarithmic response to intensity of light
  - (and many other stimuli as well – see Weber-Fechner Law and Stevens' Power Law)
- Doubling any intensity produces (roughly) the same increment in perceived brightness
  - Hence we use logarithmic scales like *decibels* for sound and *stops* for exposure

Displaying an HDR Image: Tone Mapping

- Naïve solution: Linearly map HDR range to displayable range
  - 100,000 : 1 $\rightarrow$ 100 : 1
- Problem: Flat appearance
  - Linear scaling compresses the lower end of the range too much, so the reproduction lacks contrast in the midtones and shadows which form the bulk of the image

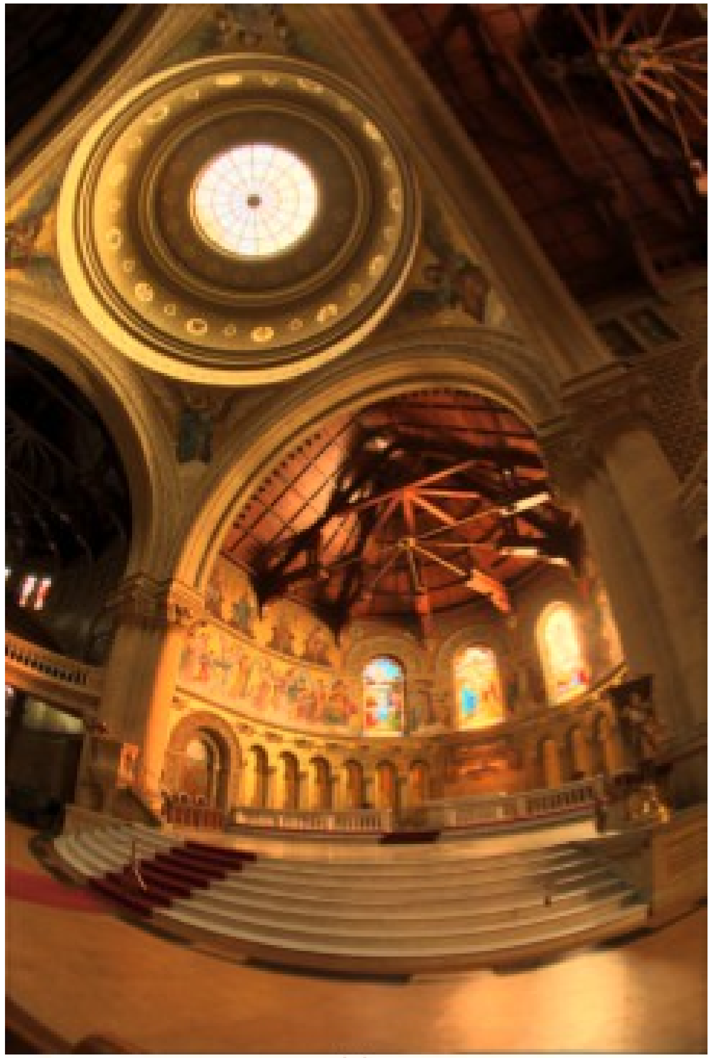


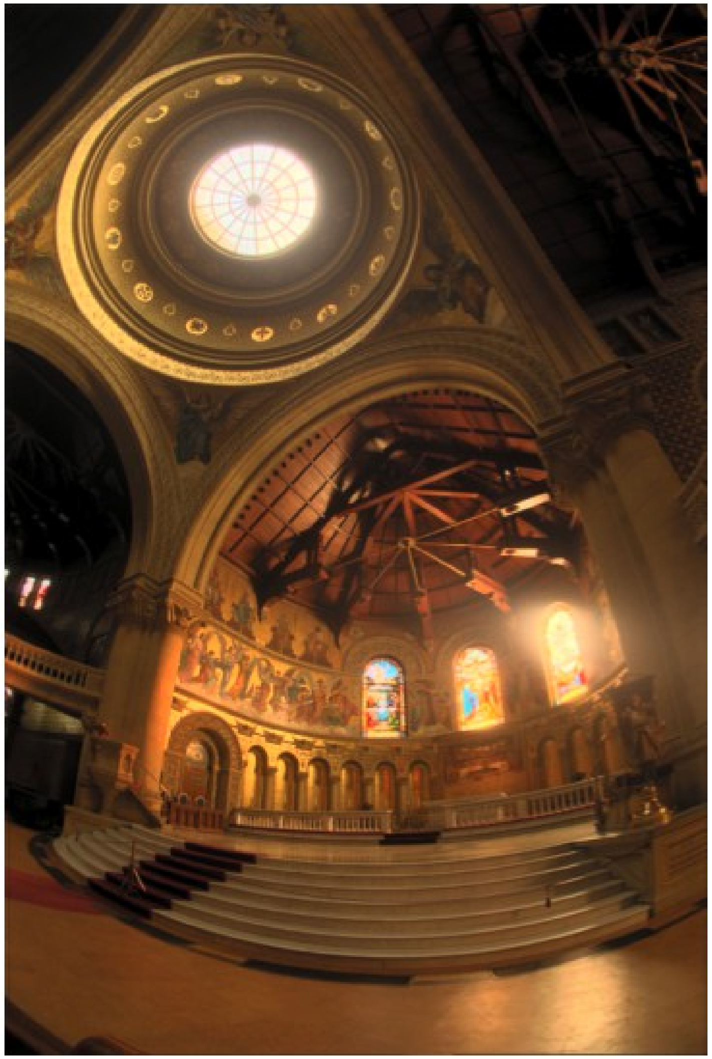
Improvement: Logarithmic Mapping



Opens up lower end of the range

Advanced Tone Mapping


Adaptive histogram compression


Taking into account glare, contrast, scotopic response etc.

(Greg Ward)

Color + Image Space Operations

- New color of pixel computed from
  - its current color
  - the colors of its neighbors
- We'll study an operation called *convolution*
- Widely used for image filters
  - e.g. blur, sharpen, edge-detect, emboss...

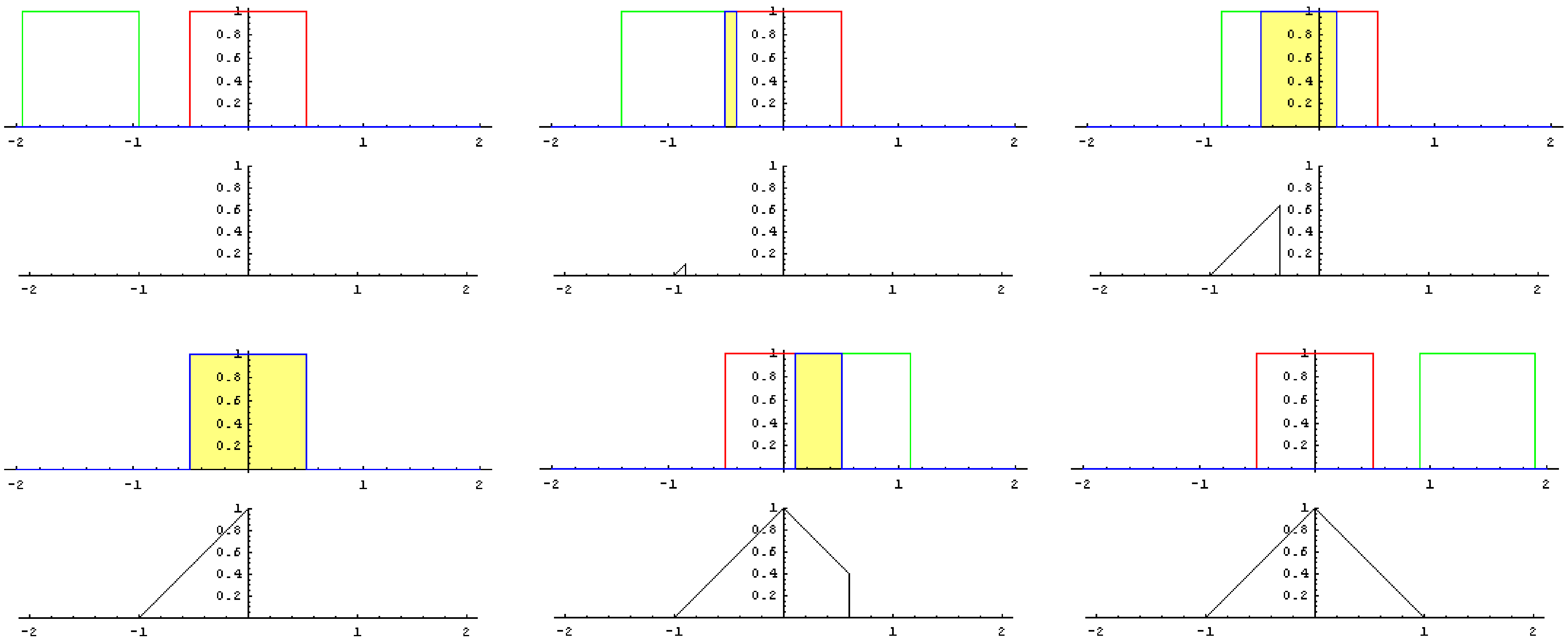
Convolution

- Recall: Image is function f mapping positions to colors
- Convolution measures overlap of f with another function g as it is (reversed and) shifted over f

$$[f * g](t) = \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau = \int_{-\infty}^{\infty} g(\tau)\, f(t-\tau)\, d\tau$$

- Note: The convolution of two functions f and g is itself a function $f * g$

Convolution Example



Tracing out the convolution of two box functions as the (reversed) green one is moved across the red one. The convolution, a triangular function, gives the area under the product of the functions for every position of the moving function

(Wikipedia)

Discrete Convolution

- If f and g are defined over integers $\mathbb{Z}$ (e.g. a 1D raster image), their *discrete convolution* is

$$[f * g](n) = \sum_{i=-\infty}^{\infty} f(i)\, g(n-i) = \sum_{i=-\infty}^{\infty} g(i)\, f(n-i)$$

- Intuition:
  - Center the *kernel/filter function* g at the n^{th} pixel
  - Weight every pixel in the image by the value of g there
  - Add up the weighted values to get the new color at the n^{th} pixel

Convolution in 2D

- Continuous

$$[f*g](s,t) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\sigma,\tau)\,g(s-\sigma,t-\tau)\,\mathrm{d}\sigma\,\mathrm{d}\tau$$

- Discrete (this is what we're going to look at)

$$[f*g](m,n) = \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} f(i,j)\,g(m-i,n-j)$$

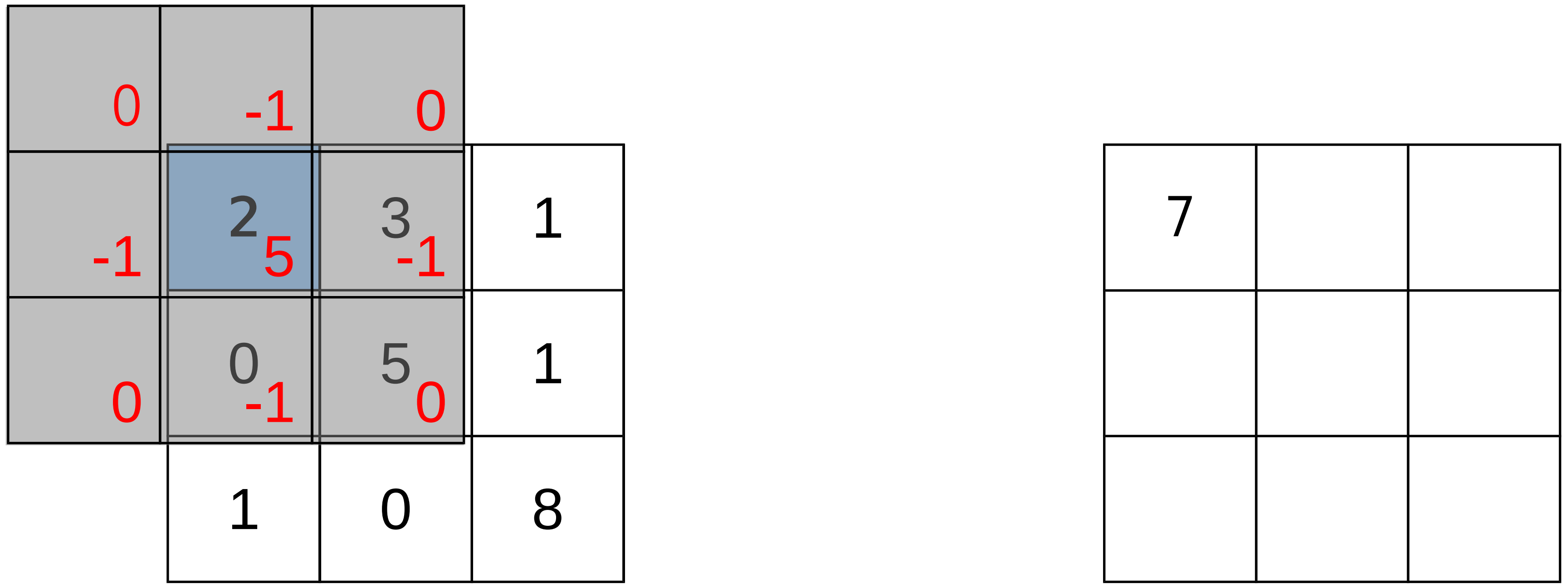
Discrete 2D Convolution: Demo

$$\begin{array}{|c|c|c|}\hline 2 & 3 & 1 \\ \hline 0 & 5 & 1 \\ \hline 1 & 0 & 8 \\ \hline \end{array} \ast \begin{array}{|c|c|c|}\hline 0 & -1 & 0 \\ \hline -1 & 5 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array} = ?$$

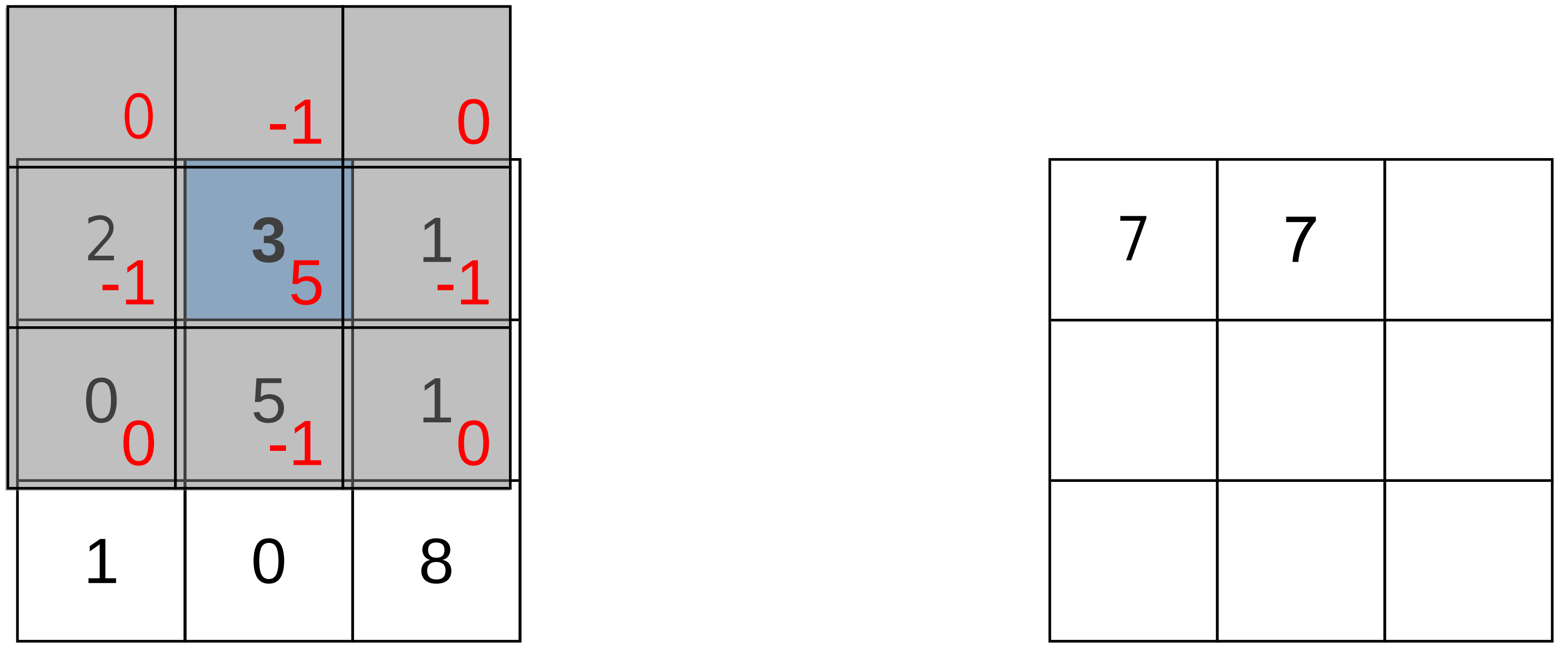
(we'll assume everything outside the 3x3 is zero)

Important: Here the kernel matrix is symmetric, but from now on any kernel matrix shown has ***already been flipped*** on both axes

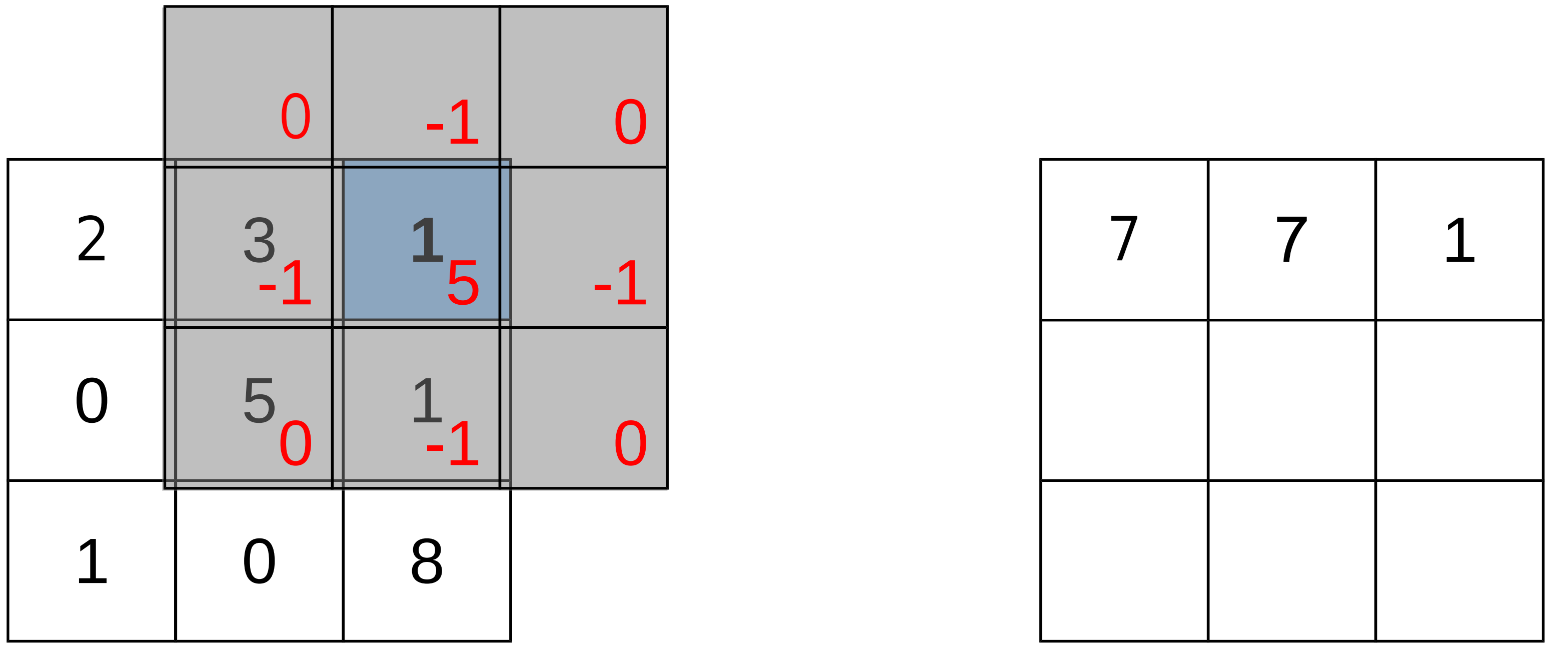
Discrete 2D Convolution: Demo



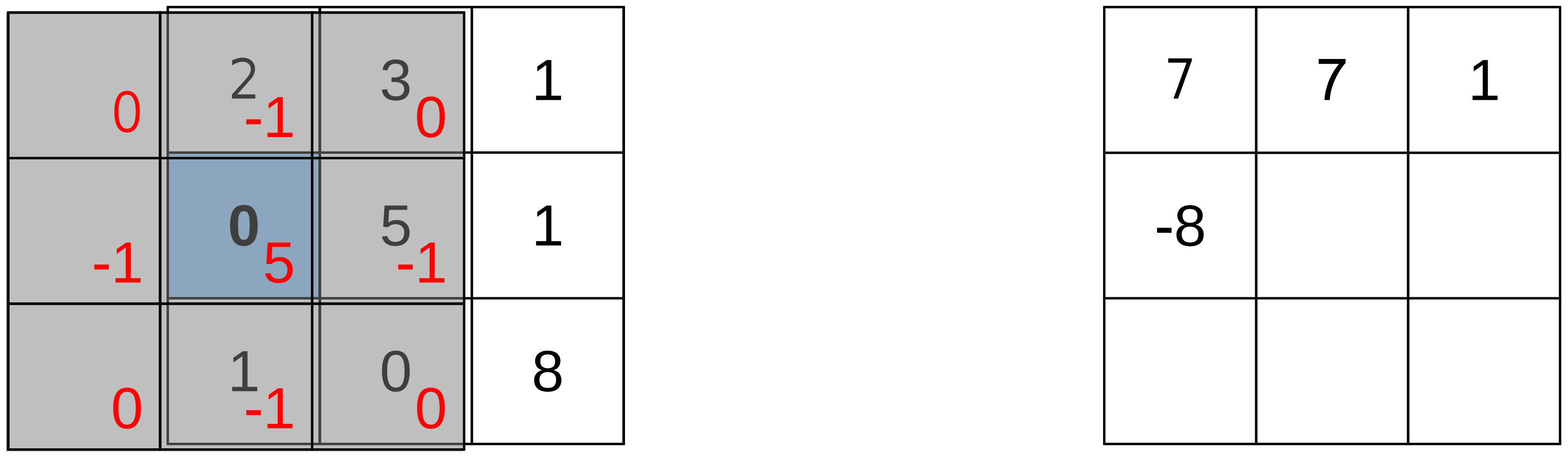
Discrete 2D Convolution: Demo



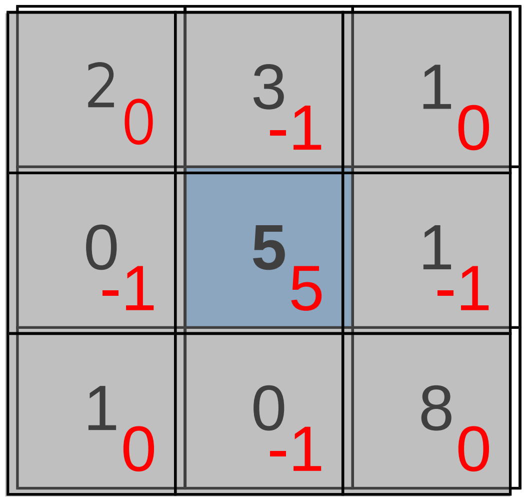
Discrete 2D Convolution: Demo



Discrete 2D Convolution: Demo

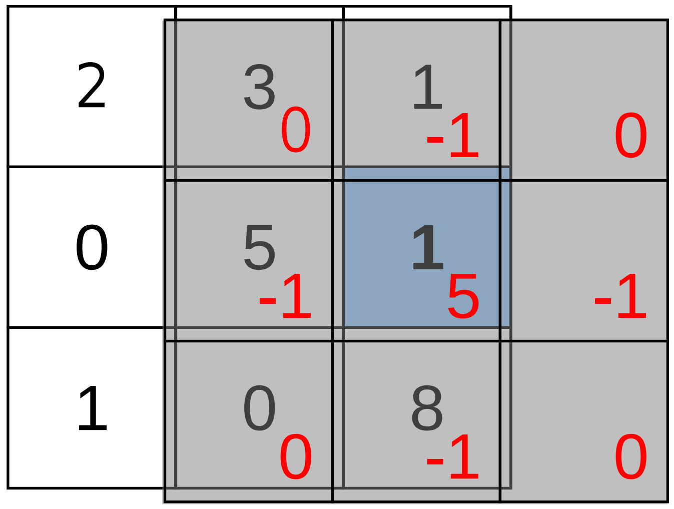


Discrete 2D Convolution: Demo



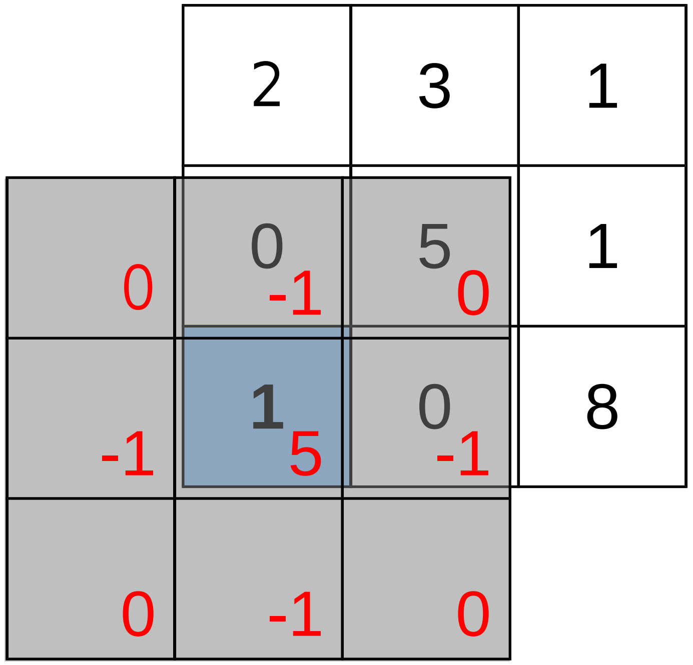
7	7	1
-8	21	

Discrete 2D Convolution: Demo



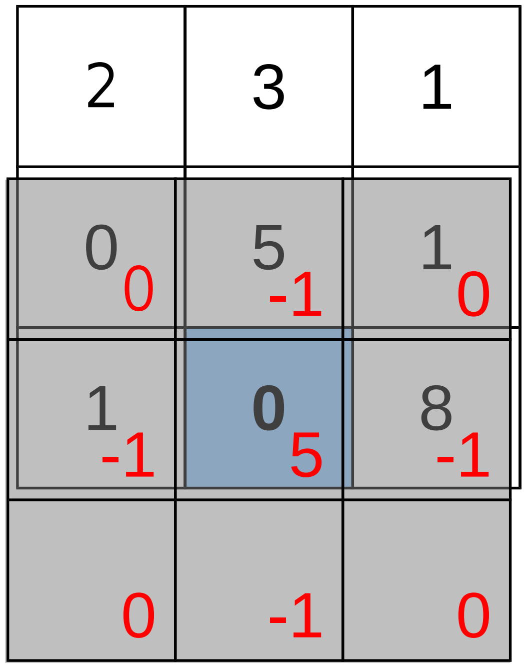
7	7	1
-8	21	-9

Discrete 2D Convolution: Demo



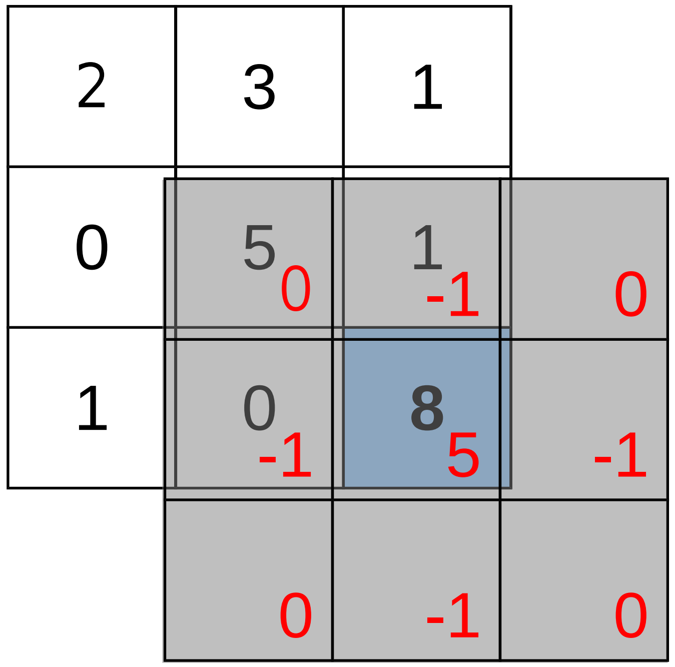
7	7	1
-8	21	-9
5		

Discrete 2D Convolution: Demo



7	7	1
-8	21	-9
5	-14	

Discrete 2D Convolution: Demo

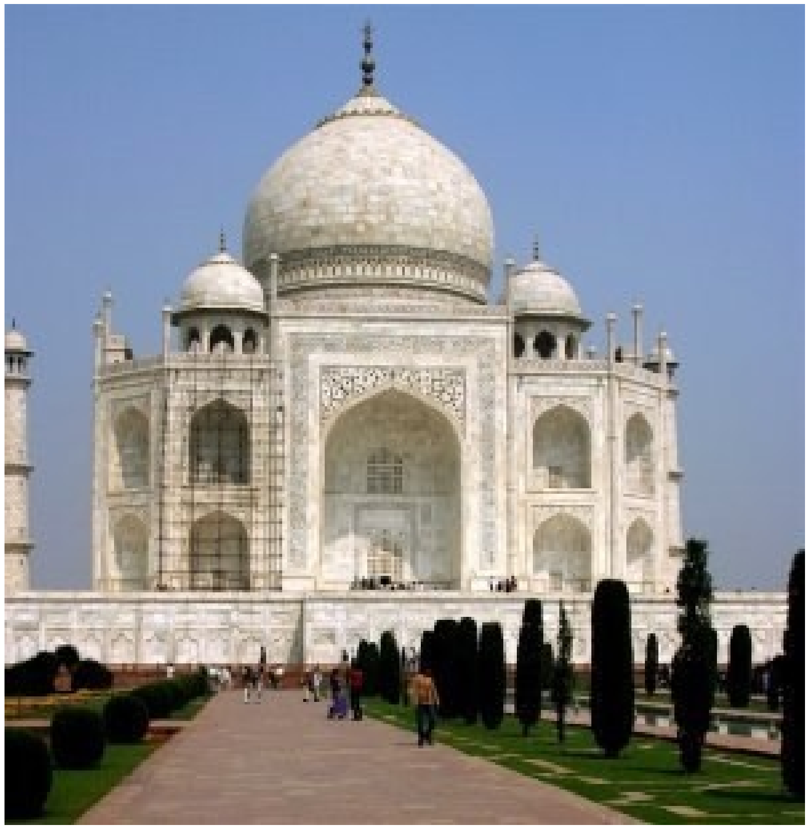


7	7	1
-8	21	-9
5	-14	39

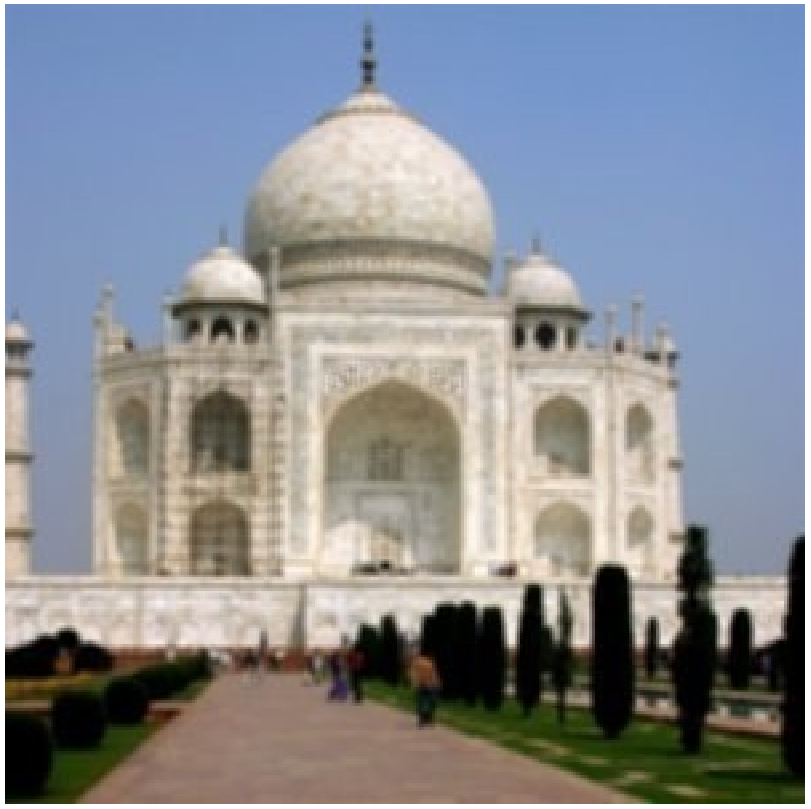
Discrete 2D Convolution: Demo

$$
\begin{array}{|c|c|c|}
\hline
2 & 3 & 1 \\
\hline
0 & 5 & 1 \\
\hline
1 & 0 & 8 \\
\hline
\end{array}
\;*\;
\begin{array}{|c|c|c|}
\hline
0 & -1 & 0 \\
\hline
-1 & 5 & -1 \\
\hline
0 & -1 & 0 \\
\hline
\end{array}
\;=\;
\begin{array}{|c|c|c|}
\hline
7 & 7 & 1 \\
\hline
-8 & 21 & -9 \\
\hline
5 & -14 & 39 \\
\hline
\end{array}
$$

Filter: Blur



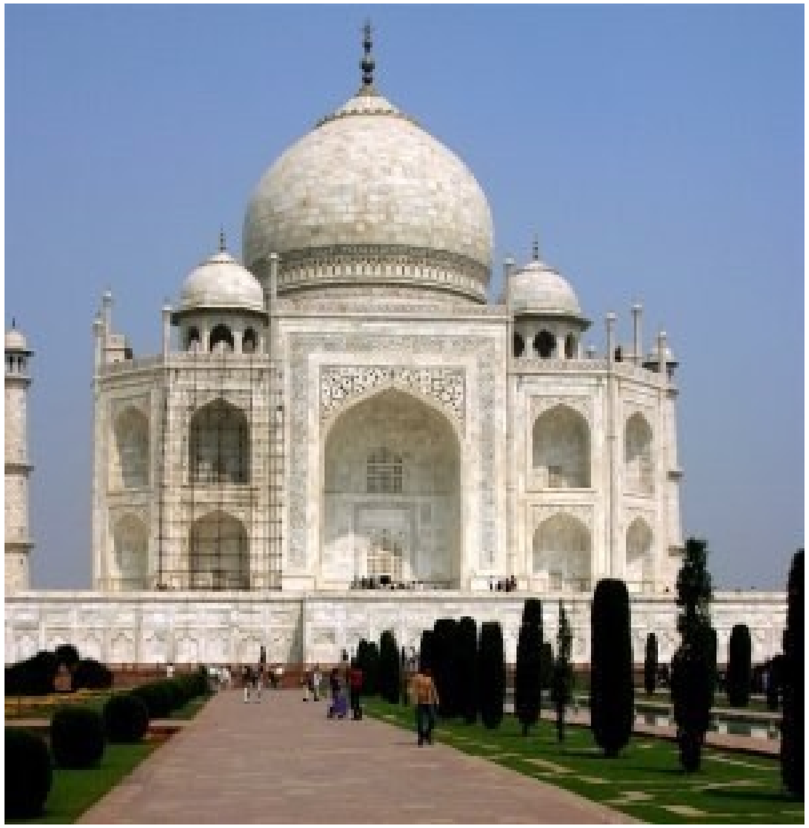
$$* \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} =$$



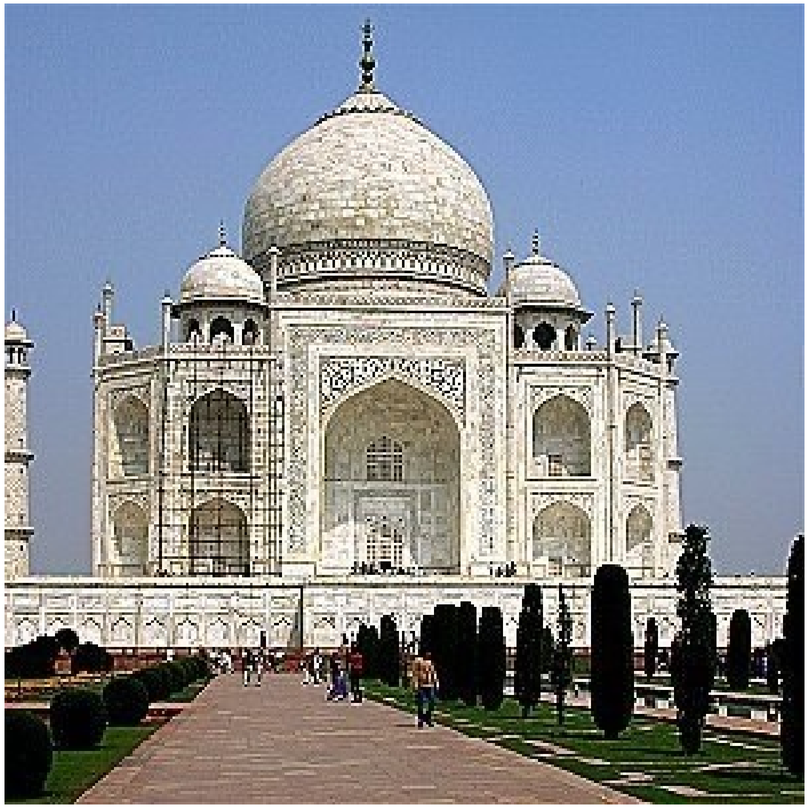
(We'll assume the kernel is normalized before convolution so the entries sum to 1)

(GIMP documentation)

Filter: Sharpen

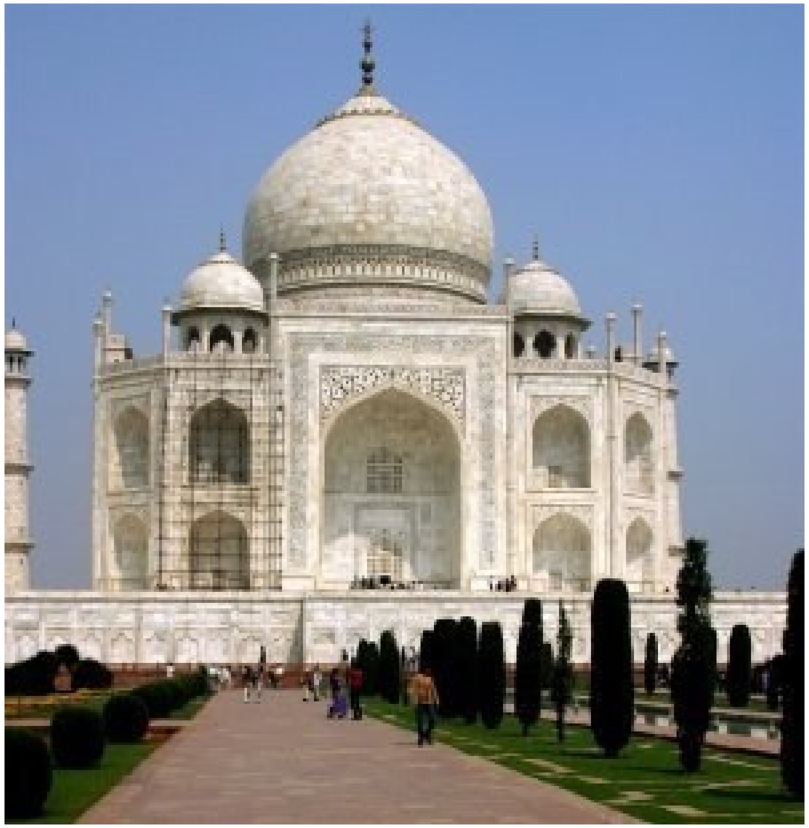


$$* \begin{array}{|c|c|c|} \hline 0 & -1 & 0 \\ \hline -1 & 5 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array} =$$

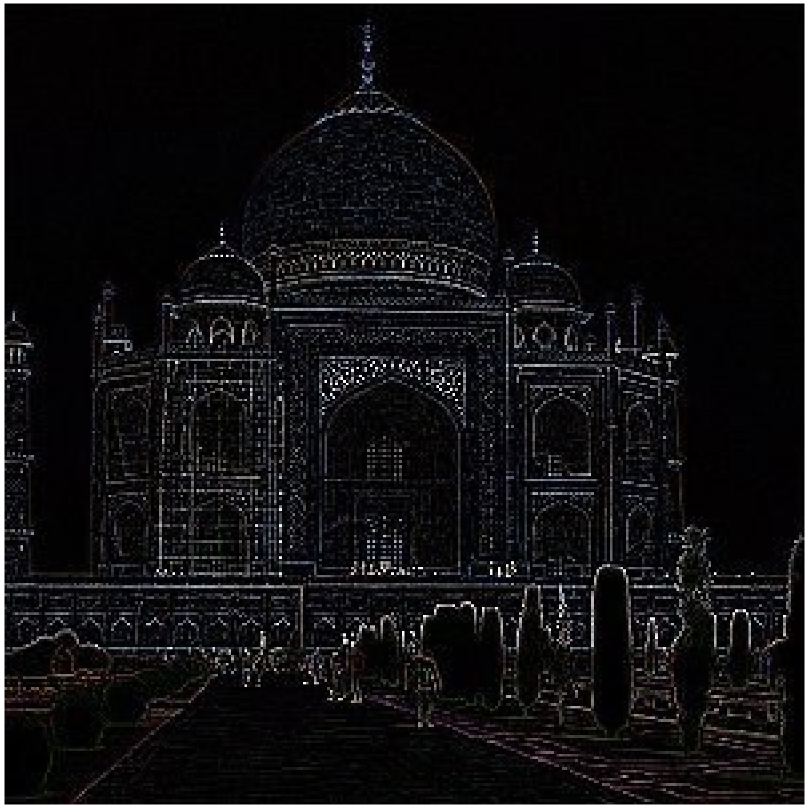


(GIMP documentation)

Filter: Edge-Detect

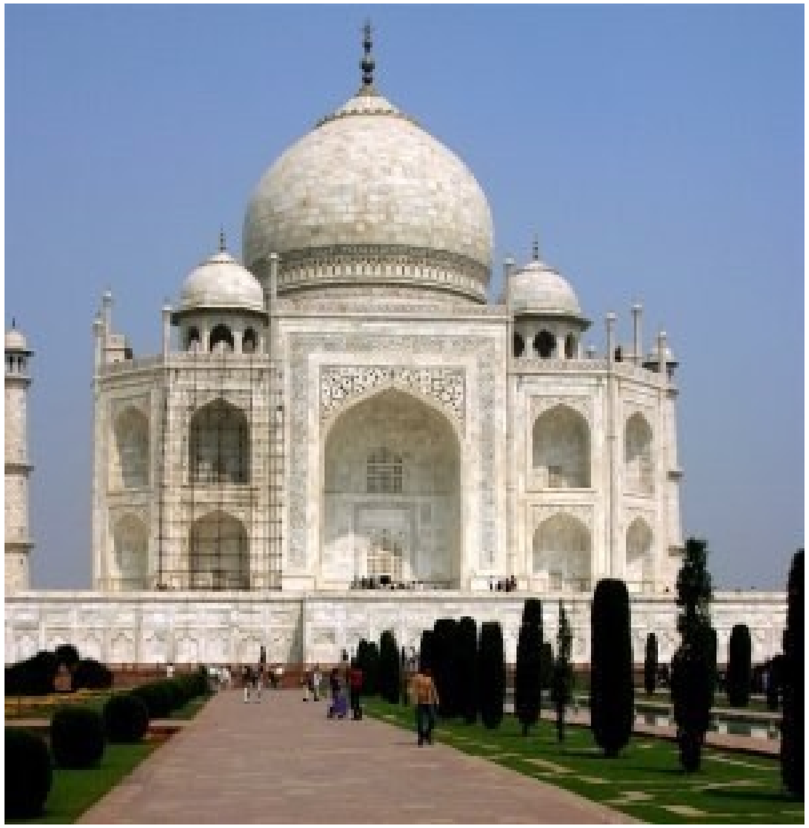


$$* \begin{array}{|c|c|c|} \hline 0 & -1 & 0 \\ \hline -1 & 4 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array} =$$

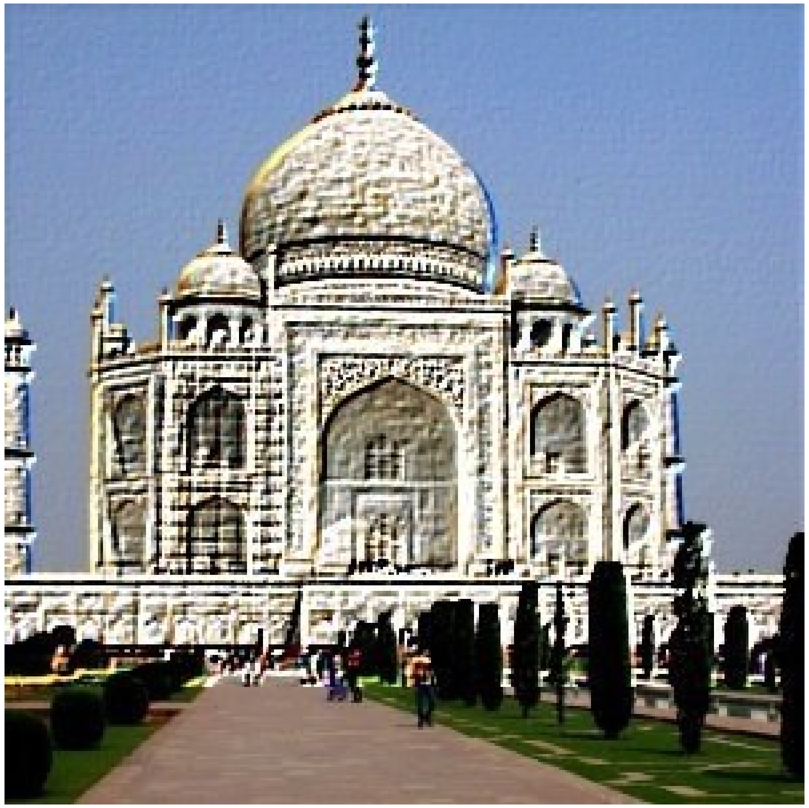


(GIMP documentation)

Filter: Emboss



$$* \begin{array}{|c|c|c|} \hline -2 & -1 & 0 \\ \hline -1 & 1 & 1 \\ \hline 0 & 1 & 2 \\ \hline \end{array} =$$



(GIMP documentation)

Resizing Images

We'll look at this during the class on sampling, aliasing etc.

How Does Superman Fly?

(Thanks to Alexei Efros for this slide)

Superhuman powers?

OR

Image Matting and Compositing?

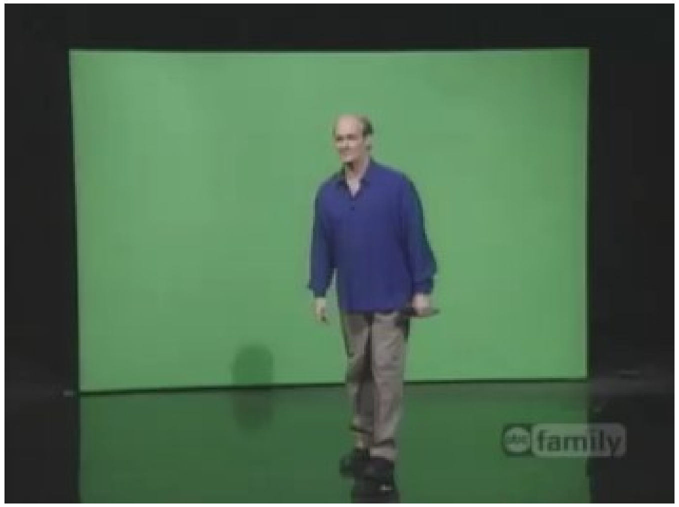
http://graphics.cs.cmu.edu/courses/15-463/2006_fall/www/Lectures/BSMatting.pdf

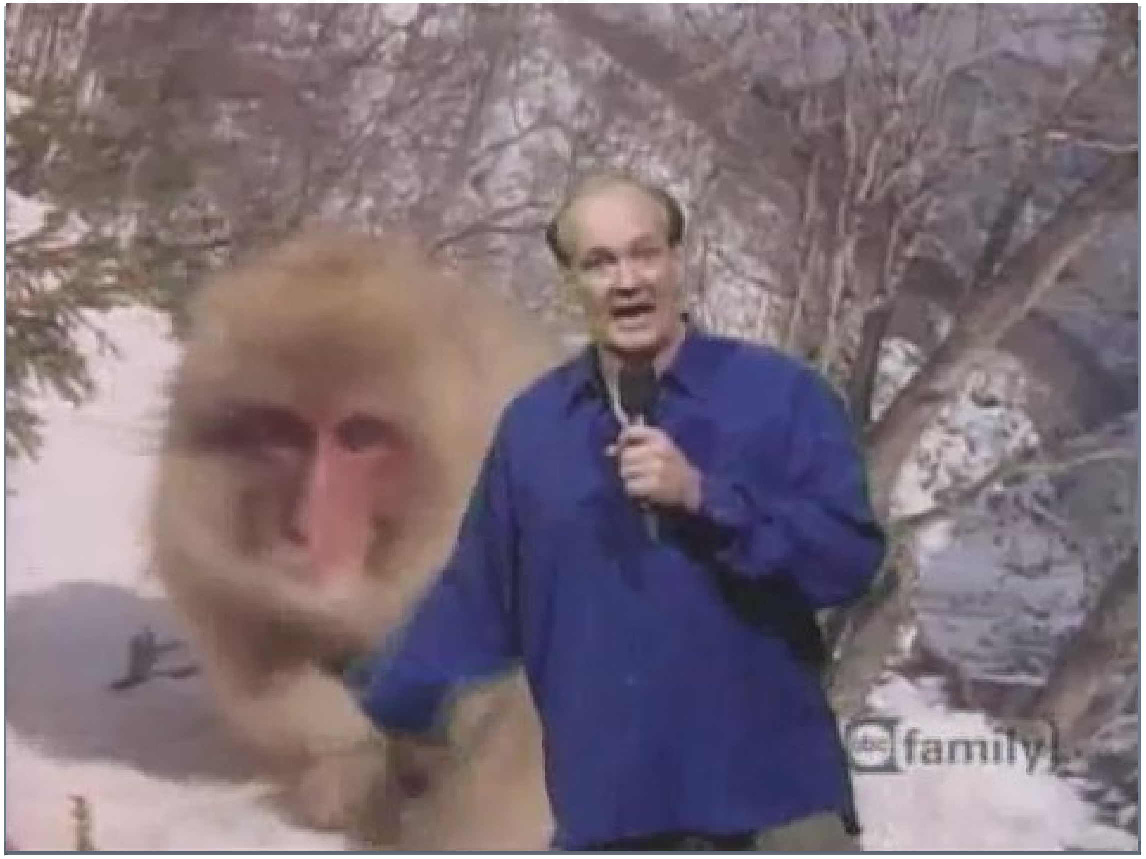
Background Subtraction and Matting

- General idea: Shoot someone in front of one background, make it look like (s)he's in front of another

Background Subtraction and Matting

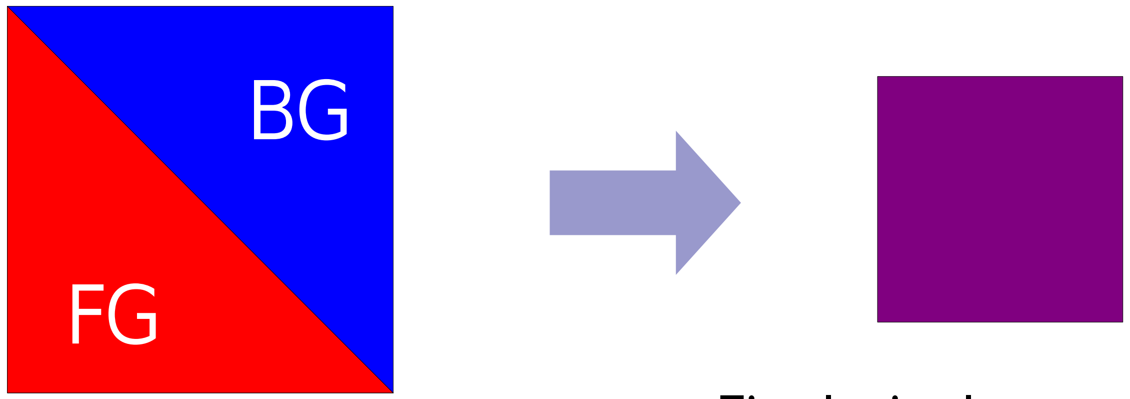
- General idea: Shoot someone in front of one background, make it look like (s)he's in front of another





Background Subtraction and Matting

- How does one remove the blue/green screen ("pull the matte")?
  - Possibility: Delete all approximately blue/green pixels
    - Don't wear a blue tie!
    - What about translucent parts of the foreground?
    - What about pixels at edges of foreground object, partially covering foreground and partially covering background?



Coverage of single pixel

Final pixel appearance

The Problem in the Abstract

- Foreground pixel has color (R_F, G_F, B_F), opacity/coverage α_F
- Background pixel has color (R_B, G_B, B_B)
- Final pixel has color (R, G, B)
- Solve:

$$\begin{aligned} R &= \alpha_F R_F + (1 - \alpha_F)\, R_B \\ G &= \alpha_F G_F + (1 - \alpha_F)\, G_B \\ B &= \alpha_F B_F + (1 - \alpha_F)\, B_B \end{aligned}$$

  for $(R_F, G_F, B_F, \alpha_F)$
- Impossible with 3 equations and 4 unknowns

Petros Vlahos Algorithm

- Vlahos invented blue screen matting
  - Founded Ultimatte, got an Oscar in 1964
- Vlahos Assumption: $B_F = \beta G_F$, for some user-specified $\beta \in [0.5, 1.5]$
  - Why???
    - Trial and error
    - Human skin tone mostly maintains such a ratio
  - With this assumption the equations are solvable
- Modern editions of Ultimatte use more refined versions of this assumption
- See Smith & Blinn, 1996, for a newer approach