© Ariel Shamir IDC

Computer Graphics

Ariel Shamir

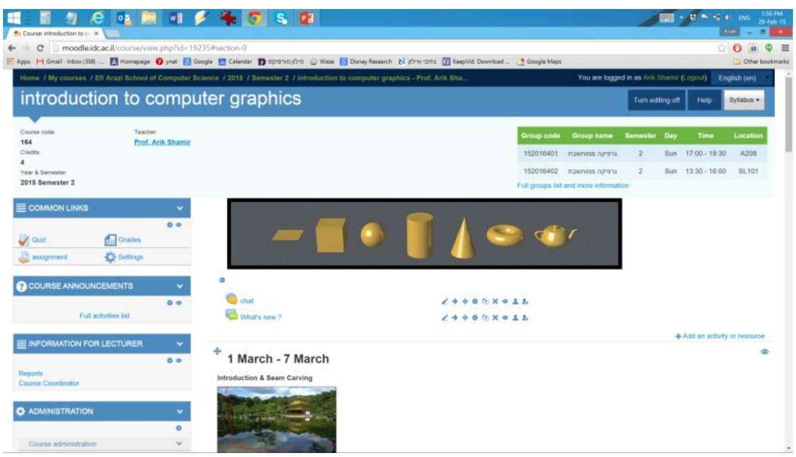
The Interdisciplinary Center

© Ariel Shamir IDC

Guidelines

- 3h lecture (Arik Shamir)
- 2h recitation (Anna Shtengel)
- 6 Exercises (50%)
- Use Java + OpenGL
- Course site: moodle!
- Find PDF of lectures & recitations, exercises etc.
- Questions about exercises using forum
- Exam (50%, must pass!)

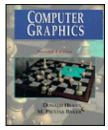
Course Site

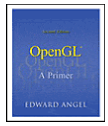


© Ariel Shamir IDC

Some Books

- Computer Graphics
  Hearn and Baker
  Second Edition, Prentice Hall, 1994.
- Computer Graphics Principles and Practice,
  Foley, Van Dam, Feiner, and Hughes
  Second Edition, Addison Wesley, 1996.
- Computer Graphics Using Open GL
  Francis S. Hill, Jr.
  Prentice Hall
- OpenGL- a primer
  E. Angel
  Addison Wesley
- OpenGL SuperBible
  R. Wright, B. Lipchak

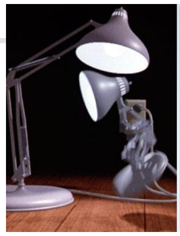




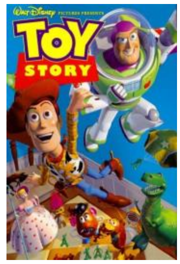
Why Graphics?

1. Math in Action
2. Abundant and almost ubiquitous
3. Cool!

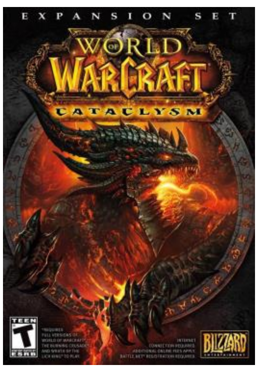
Movies Examples

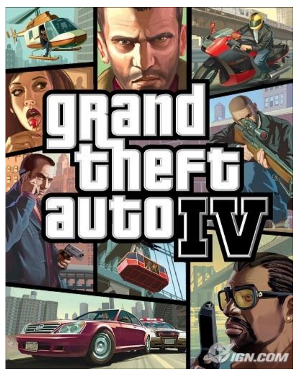


- Luxo movie
- Toy Story
- Final Fantasy
- Avatar

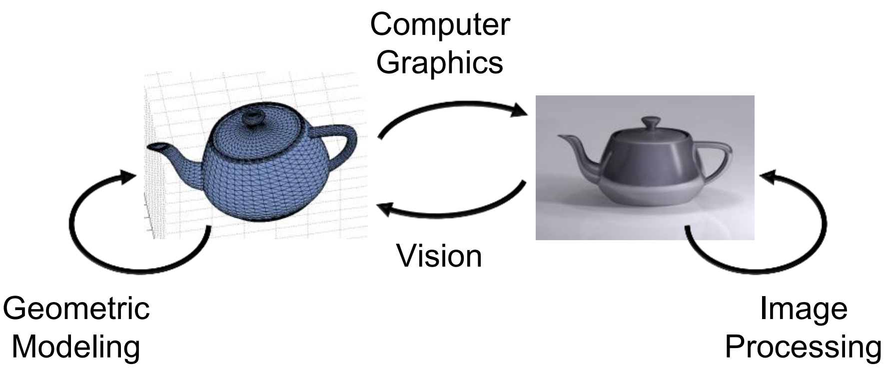


Games



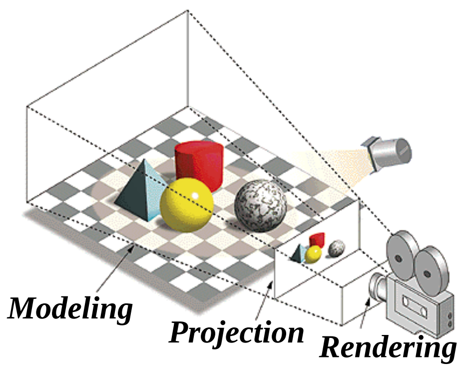


Computer Graphics & Other Related Fields



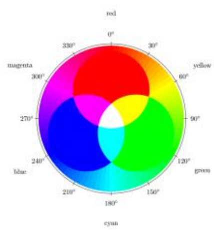
Classic Computer Graphics

1. Modeling
2. Transformations & Projections
3. Rendering



Other Subjects…

4. Color
5. Image processing
6. GUI & Event driven programming



Today: Seam Carving (Image Processing)

- What is an image?
- Resizing problem & approaches
- Seam Carving:
  - Backward energy
  - Forward energy
- Exercise: Seam Carving with Forward Energy.

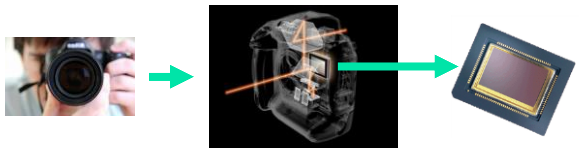
Step1: Example for Modeling

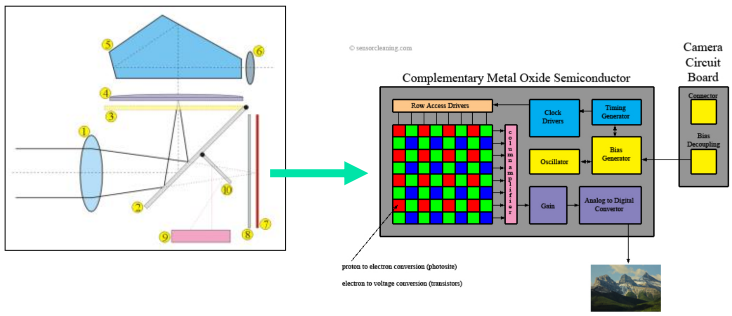
- What is an image?
- How is it represented?
- What is color?
- How is it represented?
- How to convert colors to grayscale?
- How do we model noise?

Step2: Example for Math

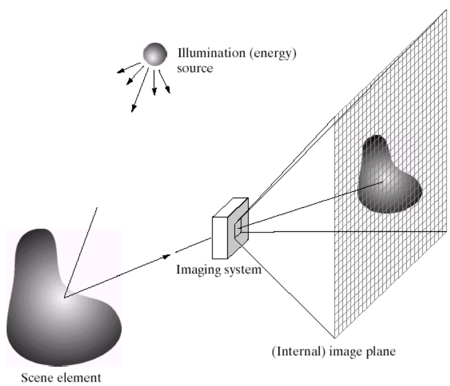
- How can we apply math operators on images?
- Images as functions
- What is an image derivative?
- Some signal processing

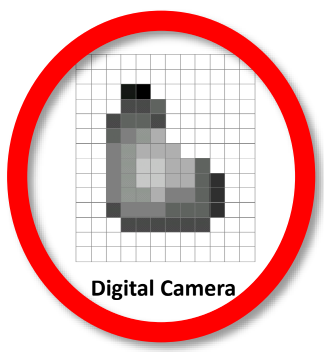
Image & Video Acquisition





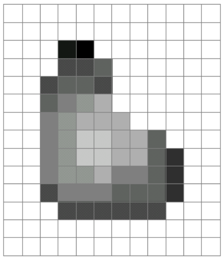
What is an image?





Source: A. Efros

A Grid of Intensity Values



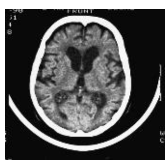
=

255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	20	0	255	255	255	255	255	255	255
255	255	255	75	75	75	255	255	255	255	255	255
255	255	75	95	95	75	255	255	255	255	255	255
255	255	96	127	145	175	255	255	255	255	255	255
255	255	127	145	175	175	175	255	255	255	255	255
255	255	127	145	200	200	175	175	95	255	255	255
255	255	127	145	200	200	175	175	95	47	255	255
255	255	127	145	145	175	127	127	95	47	255	255
255	255	74	127	127	127	95	95	95	47	255	255
255	255	255	74	74	74	74	74	74	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255

Common to use one byte per value: 0 = black, 255 = white

Images As Samples (Raster Graphics)

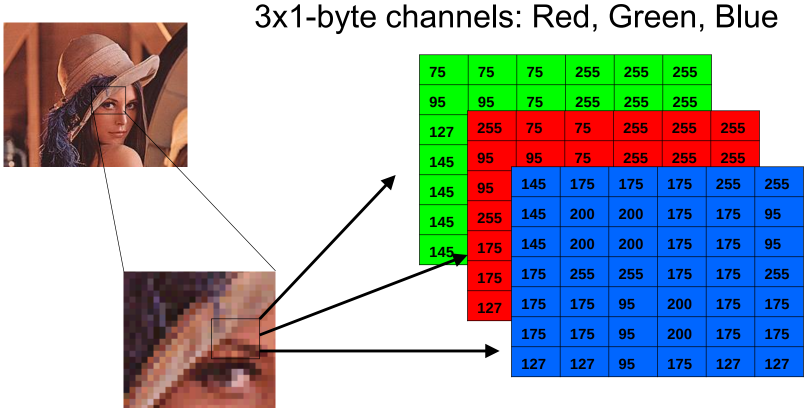
- All images can in fact be seen as point sample representation of some continuous function
- They are mostly defined on planar regular grids
- We assume some blending function to reconstruct the function on the whole space.



Color Images (Later)

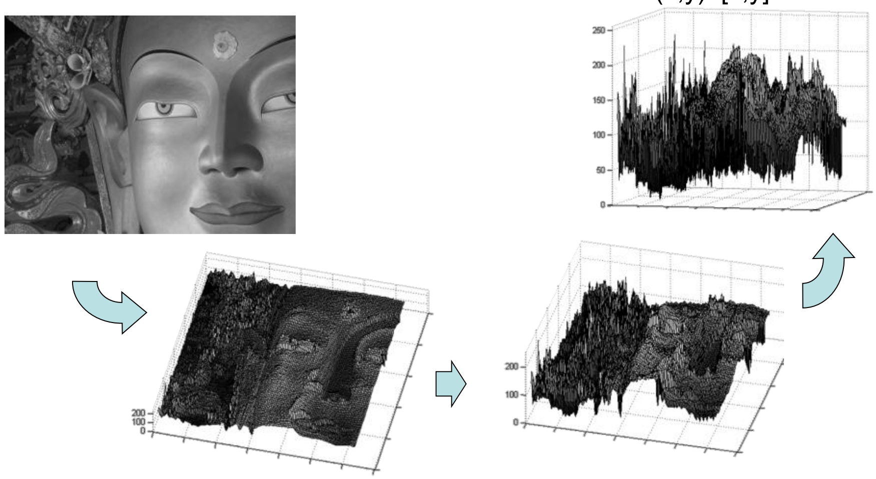
$F(x,y): [x,y] \rightarrow R^3$

3x1-byte channels: Red, Green, Blue



An Image as a 2D Function

$F(x,y): [x,y] \rightarrow R$



Sampling and Reconstruction

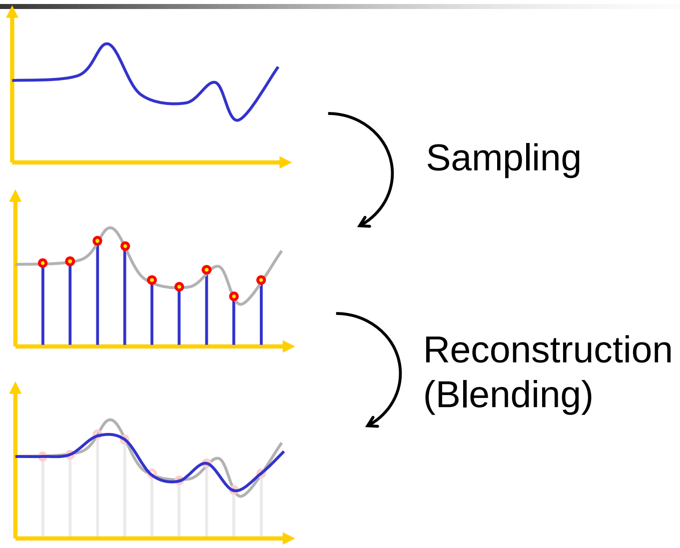


Image Derivatives?

- Derivative of an image is the derivative of the function of the image
- But: derivatives are defined on smooth functions.
- Defined using discrete differences

Pixel Differences

$dx(x,y) = I(x,y) - I(x-1,y)$

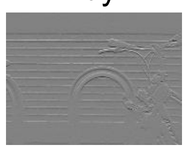
$dy(x,y) = I(x,y) - I(x,y-1)$

- $I(x,y) \in [0,255] \rightarrow d(x,y) \in [-255,255]$
- How can we visualize these differences?
- We map it back to [0,255] by adding 255 and dividing by 2.
  - Negative values are dark
  - Positive values are light
  - Zero is gray!

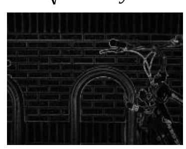
Pixel Differences

dx

dy



$\sqrt{dx^2 + dy^2}$



Example: Scale Down

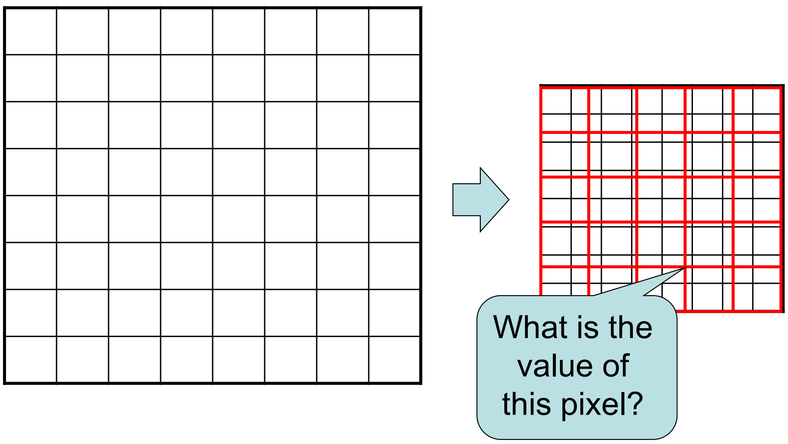
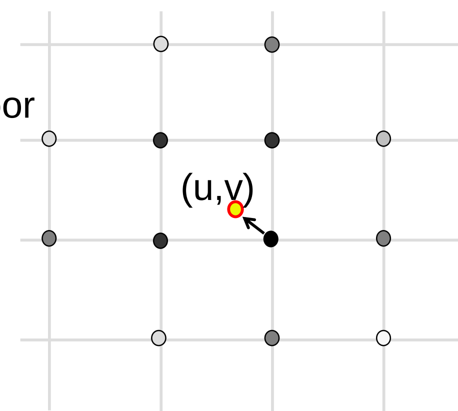


Image Resampling

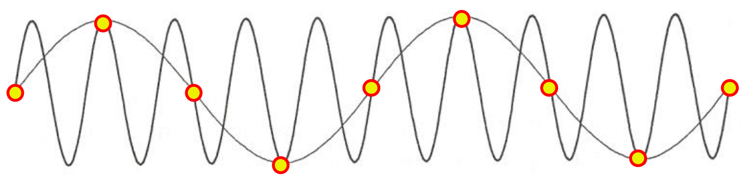
- What is the value at (u,v)?
- One Solution:
  Nearest Neighbor
- Problem:
  Aliasing!



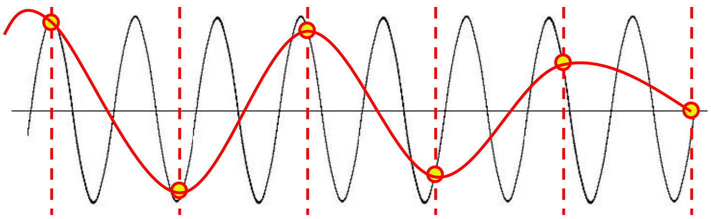
Nearest Neighbor: Aliasing

Aliasing Artifacts (1)

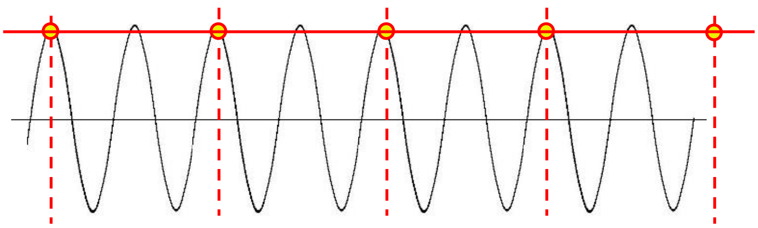
- Aliasing is what happens when we use too few samples: we cannot reconstruct the original function:



Aliasing Artifacts (2)

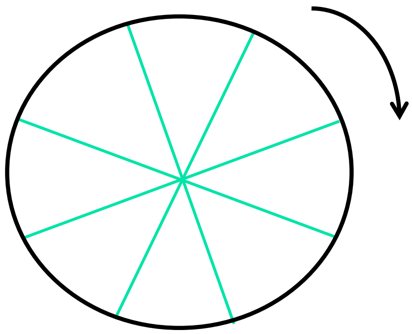


Aliasing Artifacts (3)

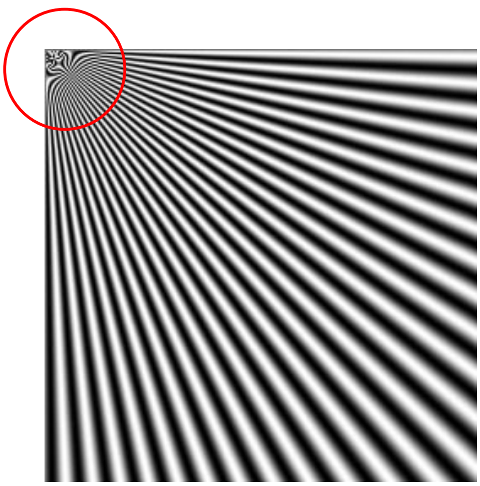


Temporal Aliasing

- Artifacts due to limited temporal resolution
  - Strobing (look stationary)
  - Flickering

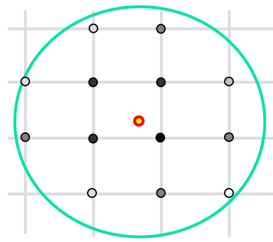


2D Aliasing



Solution:
Use Neighborhood Information

- Instead of using just one value or a small number of values we combine information from the neighborhood of the sample to get the sample value



© Ariel Shamir IDC

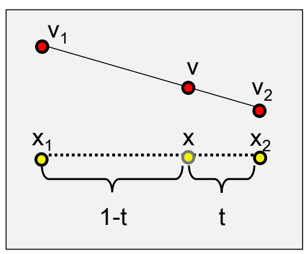
Linear interpolation

- Given two sample values at x_1 and x_2: $v_1 = f(x_1)$ and $v_2 = f(x_2)$
- How would you define the values at a position x where $x_1 < x < x_2$?
- Define the ratio based on how far x is from x_1 and x_2:

$$t = \frac{x_2 - x}{x_2 - x_1} \Rightarrow 1 - t = \frac{x - x_1}{x_2 - x_1}$$

- Use this ratio to interpolate the values v_1 and v_2:

$$v = f(x) = (1-t) \cdot v_1 + t\, v_2$$

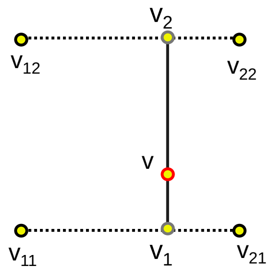


© Ariel Shamir IDC

Bilinear Interpolation

- Bi-linearly interpolation uses four values v_{11}, v_{21}, v_{12}, v_{22} of closest pixels.

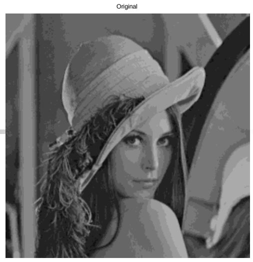
- v_1 = linear interpolation of values at v_{11} and v_{21}: $v_1 = (1-t)\cdot v_{11} + t\, v_{21}$
- v_2 = linear interpolation of values at v_{12} and v_{22}: $v_2 = (1-t)\cdot v_{12} + t\, v_{22}$
- Value at v = linear interpolation of v_1 and v_2: $v = (1-s)\cdot v_1 + s\, v_2$



Note that we use two parameters - (t,s)

Comparison

Original



Close up of scaling to 50%

Nearest Neighbor

Bilinear

Bilinear as Weighted Average

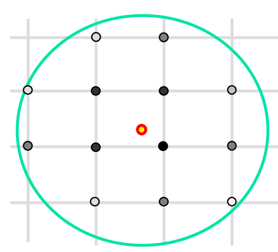
$$
\begin{aligned}
& v=(1-s)v_1+sv_2= \\
& (1-s)\big[(1-t)v_{11}+tv_{12}\big]+s\big[(1-t)v_{21}+tv_{22}\big]= \\
& (1-s)(1-t)v_{11}+(1-s)tv_{12}+s(1-t)v_{21}+stv_{22}= \\
& w_{11}v_{11}+w_{12}v_{12}+w_{21}v_{21}+w_{22}v_{22}=\sum_{1\le i,j\le 2} w_{ij}v_{ij}
\end{aligned}
$$

Key Idea:
Use Neighborhood Information

- Instead of using just one value or a small number of values we combine information from the neighborhood of the sample to get the sample value

$$\sum_{i,j\in \text{Neighborhood}} w_{ij}v_{ij}$$

The output is a weighted sum of values of pixels in neighborhood

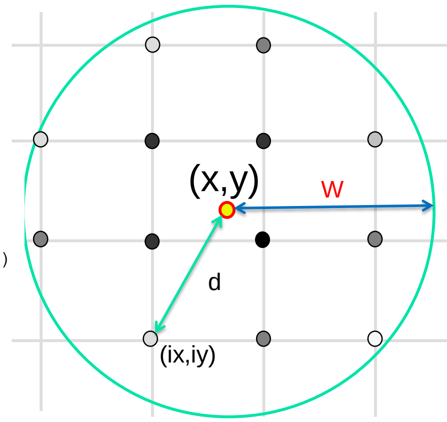


Kernel Resampling

- Calculating the weighted sum of values of pixels in neighborhood:

```
value(x,y)=0;
for(ix=x-w;ix<=x+w;ix++)
  for(iy=y-w;iy<=y+w;iy++)
    d=dist between (ix,iy) and (u,v)
    value(x,y) += k(d) *val(ix,iy)
```

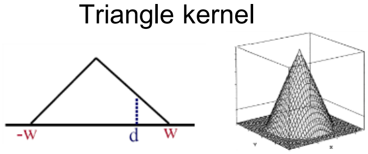
Kernel Function



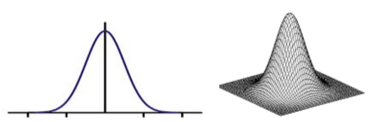
Kernels

- k() is a function of d (the distance) and w (the range)
- Uniform: K(d) = 1/N where N is the number of neighbors – creates a simple averaging
- More complex averaging gives a weight to each value (weighted average) for example based on the distance from the center:

Triangle kernel



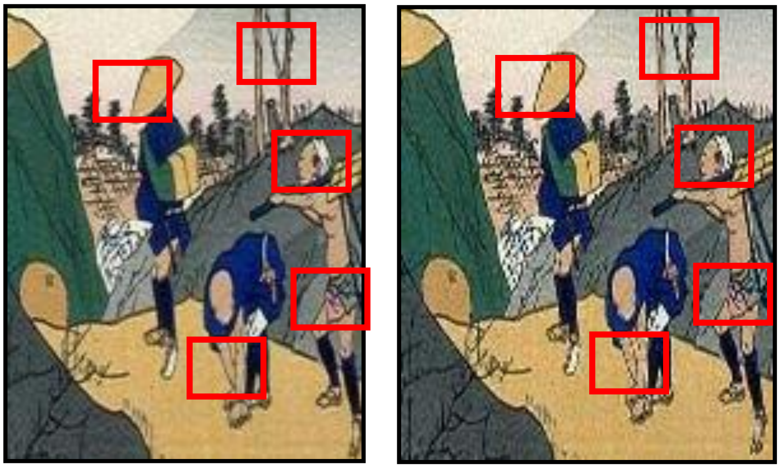
Gaussian kernel



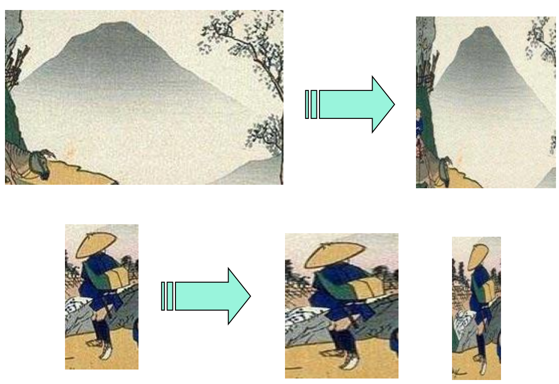
Non Uniform Resizing

Nearest Neighbor Vs. using Kernels (smoothing)

Comparison



Still, non-uniform artifacts!



Content Aware Retargeting

- Changing the aspect ratio while preserving details
- Given the original media in size mxn resize it to size m’xn’

  where m’ ≠ m or n’ ≠ n or both.

?

?

Simple Approaches?

- How can we reduce the size of the image in a non-uniform manner?

?

Simple approaches

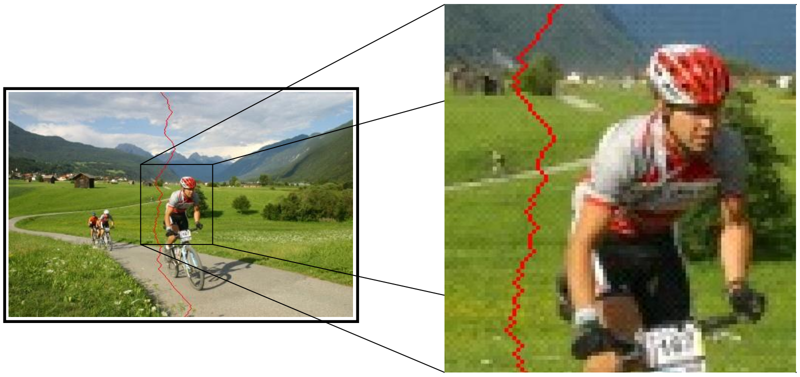
- Scaling creates artifacts
- Cropping looses information at the edges, and cannot support enlarging an image:

Cropping?

Enlarging?

Seam Carving

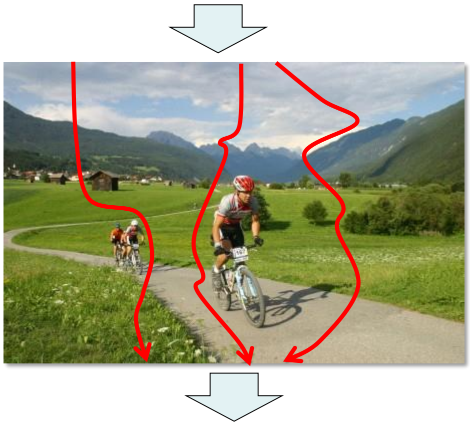
A Seam



Seam Carving

A Local Operator!

Finding the Seam?



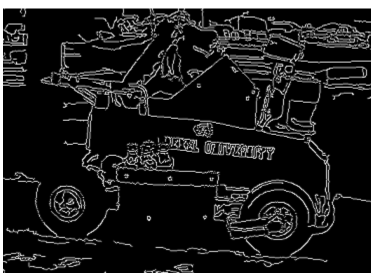
Key Idea: Content Aware

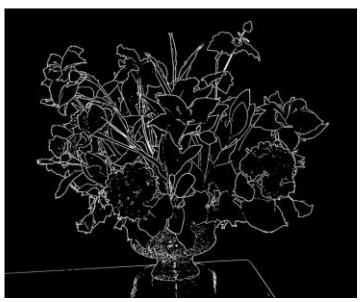
- Remove (or Insert) “less important” parts and preserve more important ones
- In effect this means we are creating ... ***content aware*** resizing

- **Key questions: what is important?**
  - **Edges are important**

What Is Important?

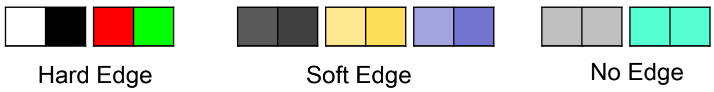
- Edges Carry most information in the scene:





Finding Edges

- Large difference in color values in an image constitutes an edge
- Examining two neighboring pixels:

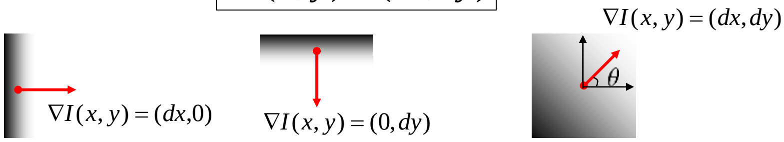


- Simple approach: assume we have a gray scale image, then measure horizontal and vertical pixel differences

Gradient

- For each pixel we have dx,dy values.
- Together they define a vector (dx,dy) that is called the gradient whose direction is the maximum change and magnitude is the amount of change.

$$\boxed{\nabla I(x,y) = (dx,dy)}$$



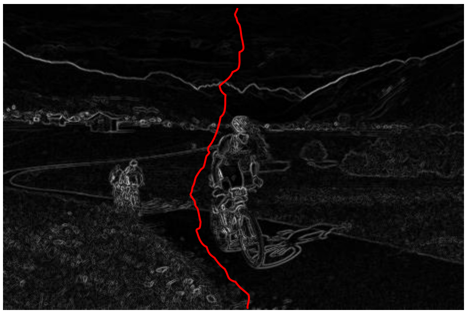
Gradient Magnitude

- A simple way to find edges in the image is to measure the magnitude of the gradient for each pixel (always positive)

$$\|\nabla I(x,y)\|_2 = \sqrt{dx^2 + dy^2}$$

$$\|\nabla I(x,y)\|_1 = |dx| + |dy|$$

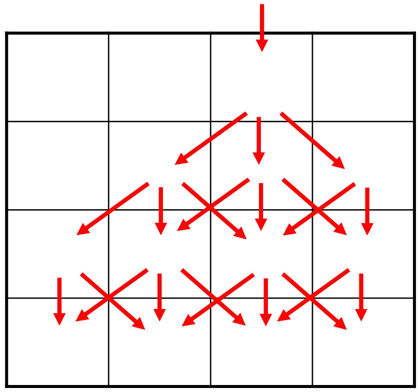
The Optimal Seam



$$E(\mathbf{I}) = |dx\mathbf{I}| + |dy\mathbf{I}| \quad \Rightarrow \quad s^* = \arg\min_s E(s)$$

Naïve Approach

- Loop over all seams and check their energy E(s). Choose the one with smallest energy.
- How many seams?
- Exponential (~3^h for wxh image)



However… Pixel Attributes → Dynamic Programming

$$M(i, j) = e(i, j) + \min(M(i-1, j-1), M(i-1, j), M(i-1, j+1))$$

5	8	12	3
9	2	3	9
7	3	4	2
6	5	7	8

Dynamic Programming

$$M(i, j) = e(i, j) + \min(M(i-1, j-1), M(i-1, j), M(i-1, j+1))$$

5	8	12	3
9	2+5	3	9
7	3	4	2
6	5	7	8

Dynamic Programming

$$M(i, j) = e(i, j) + \min(M(i-1, j-1), M(i-1, j), M(i-1, j+1))$$

5	8	12	3
9	7	3+3	9
7	3	4	2
6	5	7	8

Dynamic Programming

$M(i, j) = e(i, j) + \min(M(i-1, j-1), M(i-1, j), M(i-1, j+1))$

5	8	12	3
9	7	6	12
14	9	10	8
15	14	15	8+8

Searching for Minimum

$M(i, j) = e(i, j) + \min(M(i-1, j-1), M(i-1, j), M(i-1, j+1))$

5	8	12	3
9	7	6	12
14	9	10	8
15	14	15	16

Backtracking the Seam

$M(i, j) = e(i, j) + \min(M(i-1, j-1), M(i-1, j), M(i-1, j+1))$

5	8	12	3
9	7	6	12
14	9	10	8
15	14	15	16

Backtracking the Seam

$M(i, j) = e(i, j) + \min(M(i-1, j-1), M(i-1, j), M(i-1, j+1))$

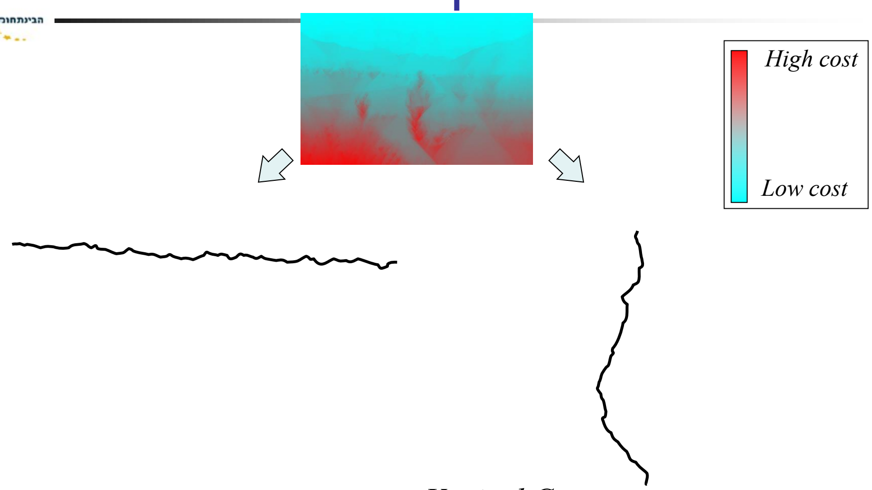
5	8	12	3
9	7	6	12
14	9	10	8
15	14	15	16

Backtracking the Seam

$M(i, j) = e(i, j) + \min(M(i-1, j-1), M(i-1, j), M(i-1, j+1))$

5	8	12	3
9	7	6	12
14	9	10	8
15	14	15	16

H & V Cost Maps



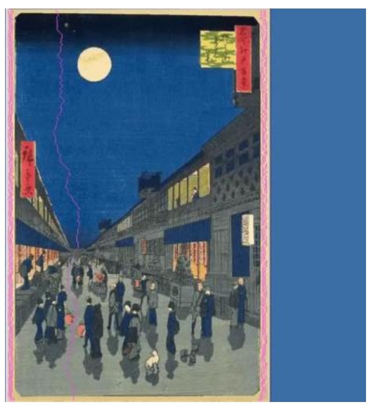
Horizontal Cost

Vertical Cost

Seam Insertion?

Seam Insertion

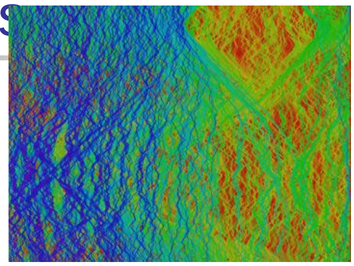
Duplicate seams in removal order

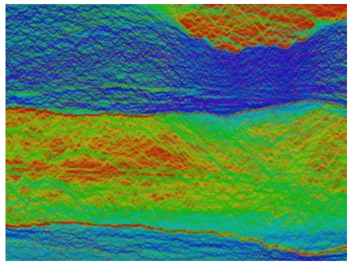


Seam Insertion

Enlarged or Reduced?

Multi-Size Images



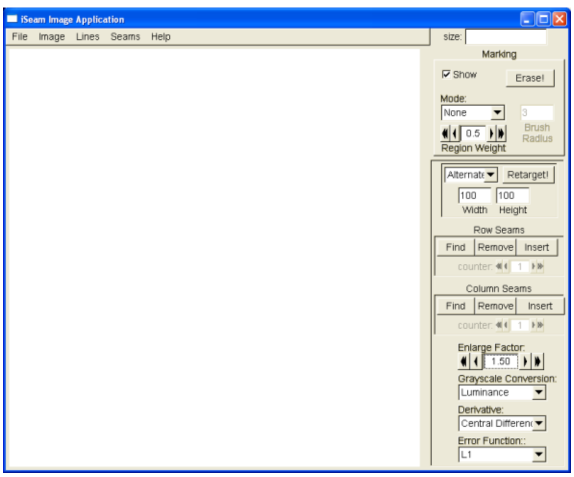


First to be removed

⋮

Last to be removed

Multi-Size Images & Demo



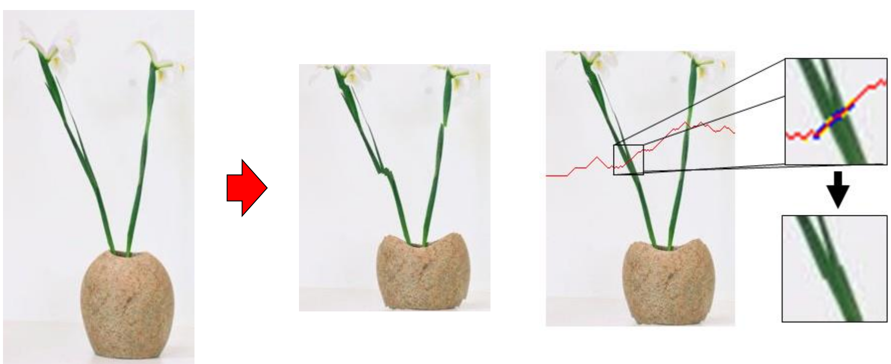
Not Always a Success

Image: Yehudit Garinkol

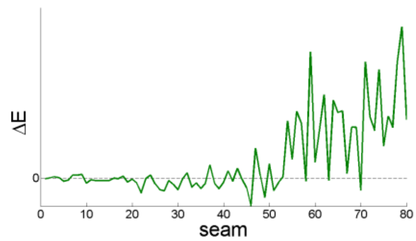
Find the Missing Shoe!

Solution

More Problems…



Change in Energy

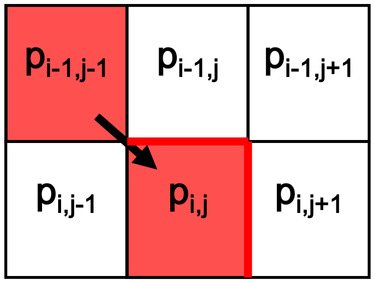


Tracking Inserted Energy

$p_{i-1,j-1}$	$p_{i-1,j}$	$p_{i-1,j+1}$
$p_{i,j-1}$	$p_{i,j}$	$p_{i,j+1}$

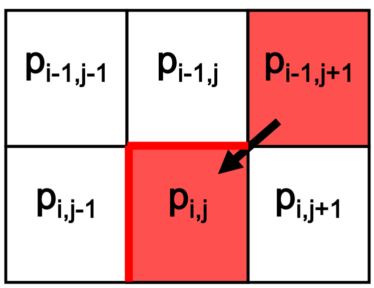
- Three possibilities when removing pixel $P_{i,j}$

Pixel $P_{i,j}$: Left Seam



$$C_L(i,j) = |I(i,j+1) - I(i,j-1)| + |I(i-1,j) - I(i,j-1)|$$

Pixel $P_{i,j}$: Right Seam



$$C_R(i,j) = |I(i,j+1) - I(i,j-1)| + |I(i-1,j) - I(i,j+1)|$$

Pixel $P_{i,j}$: Vertical Seam

$p_{i-1,j-1}$	$p_{i-1,j}$	$p_{i-1,j+1}$
$p_{i,j-1}$	$p_{i,j}$	$p_{i,j+1}$

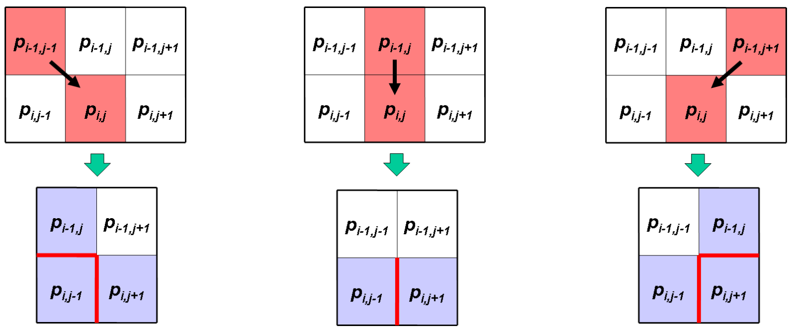
$$C_V(i,j) = |I(i,j+1) - I(i,j-1)|$$

“Backward” Energy Function

$$M(i,j) = E\ (i,j) + \min \begin{cases} M(i-1,j-1) \\ M(i-1,j) \\ M(i-1,j+1) \end{cases}$$

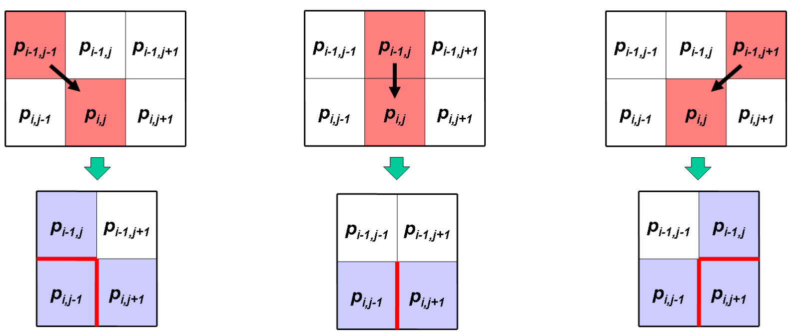
Forward Looking Energy

$$M(i,j) = \min \begin{cases} M(i-1,j-1) + C_L(i,j) \\ M(i-1,j) + C_U(i,j), \\ M(i-1,j+1) + C_R(i,j) \end{cases}$$



Adding “Pixel Energy”

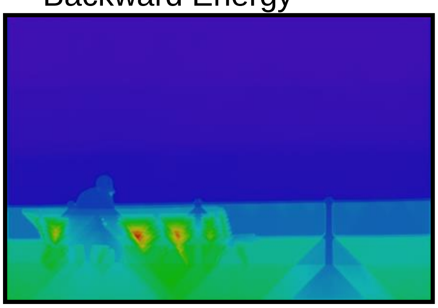
$$M(i,j) = P(i,j) + \min \begin{cases} M(i-1,j-1) + C_L(i,j) \\ M(i-1,j) + C_U(i,j), \\ M(i-1,j+1) + C_R(i,j) \end{cases}$$



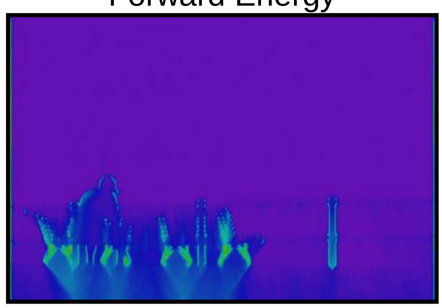
Backward Energy

Forward Energy

Backward Energy

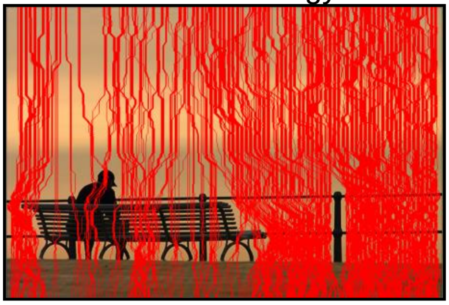


Forward Energy



Backward Energy

Forward Energy



Backward

Forward

Backward

Forward

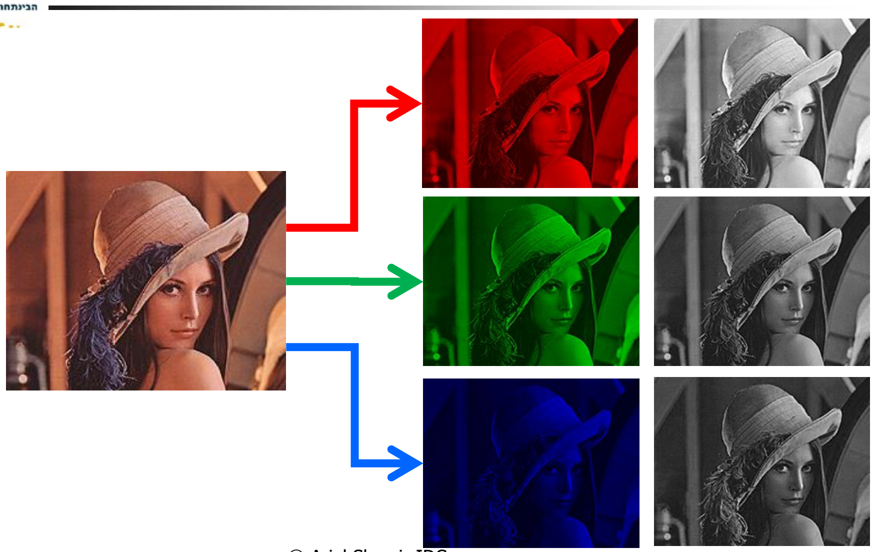
Backward

Forward

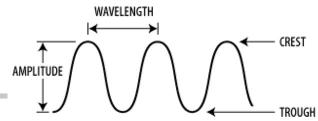
Expand

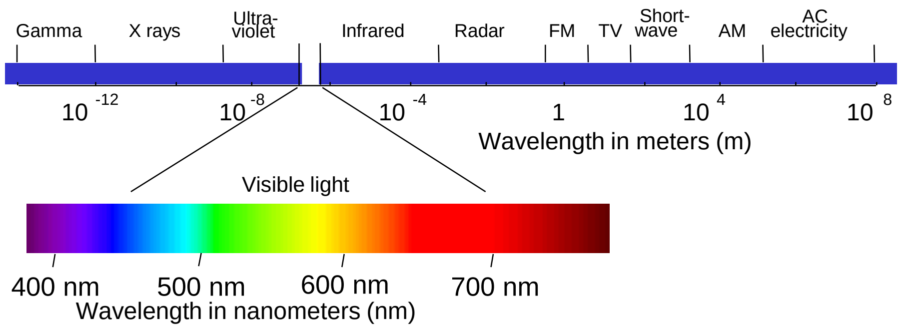
Expand

Colors?



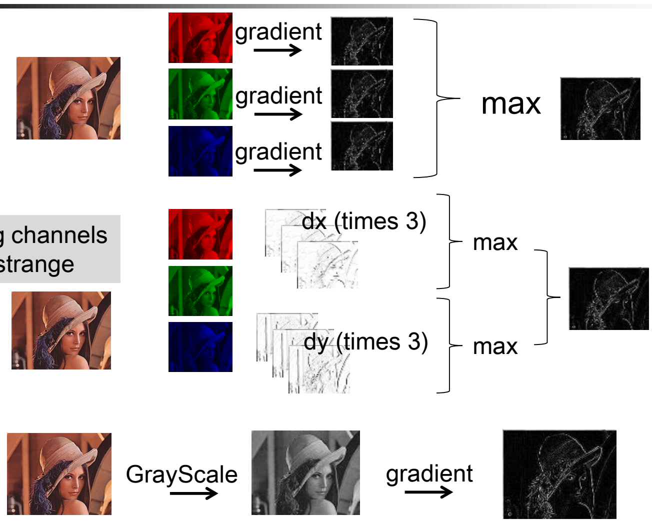
What is Color?





Possibilities

Combining channels – may be strange



Converting to Grayscale

- Max(C,G,B)
- Min(R,G,B)
- Mean(R,G,B)
- Sum(R,G,B) (normalized)
- Common (PhotoShop):

0.2989 * R + 0.5870 * G + 0.1140 * B