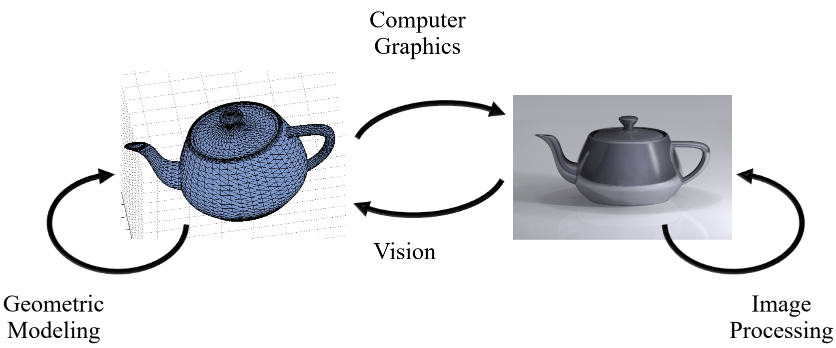
Modeling & Optimization

Ariel Shamir
Efi Arazi School of
Computer science

Computer Graphics & Other Related Fields



Problem: Aspect Ratio Change

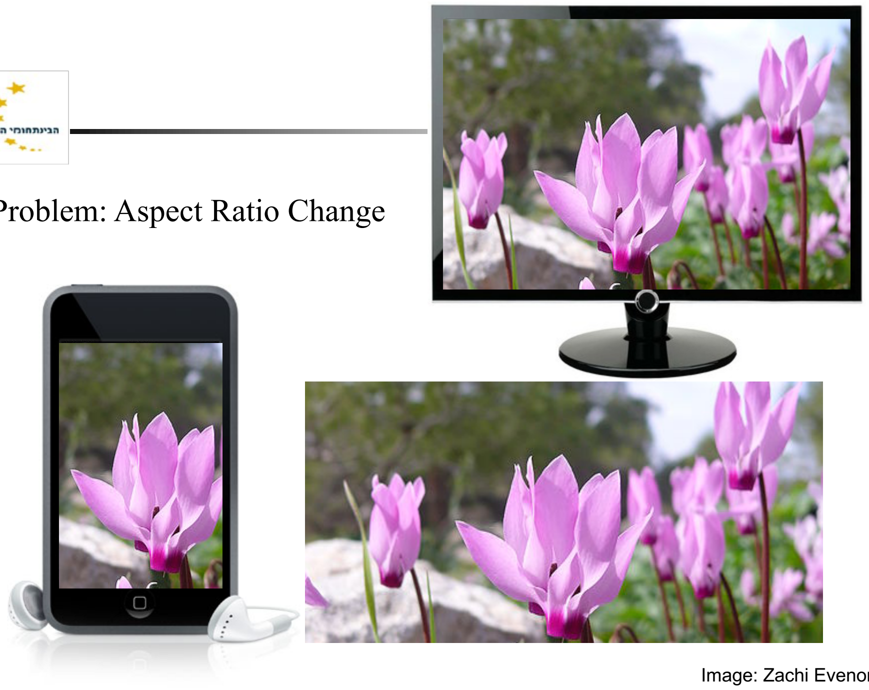


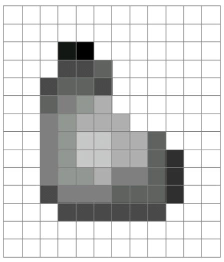
Image: Zachi Evenor

Content Aware | Scale | Crop

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?**

What is an Image?



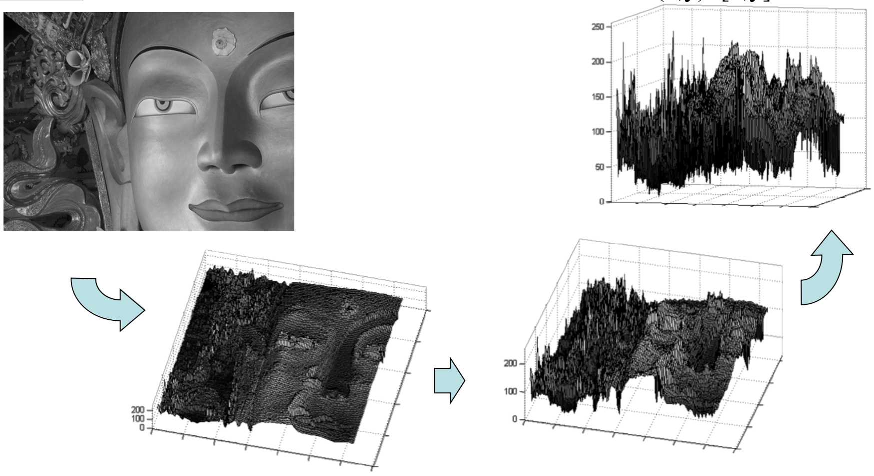
=

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

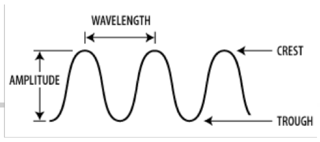
Common to use one byte per value: 0 = black, 255 = white

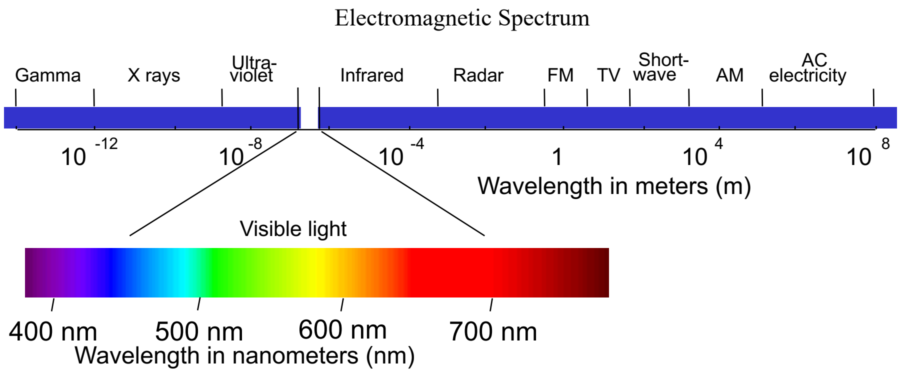
An Image as a 2D Function

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



What is Color?

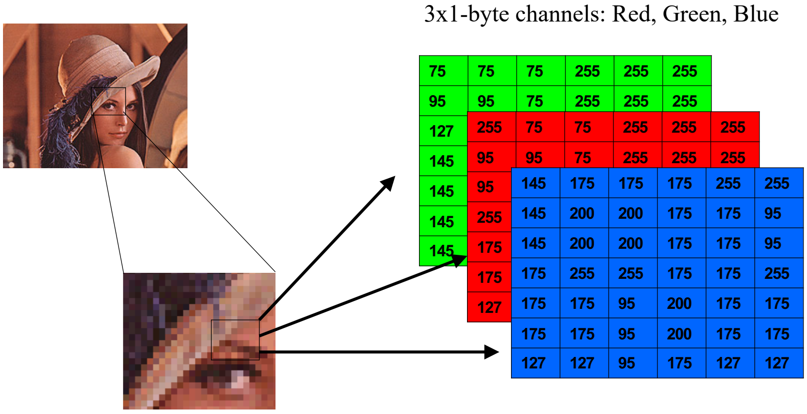




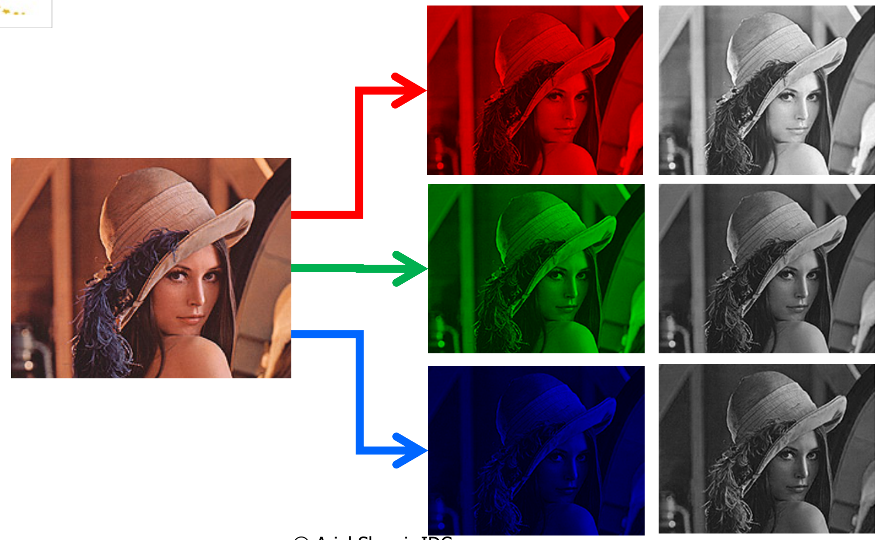
Color Images

F(x,y): [x,y] → R^3

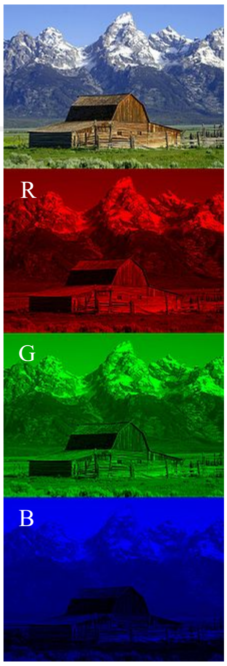
3x1-byte channels: Red, Green, Blue



Color Images



Converting to Grayscale

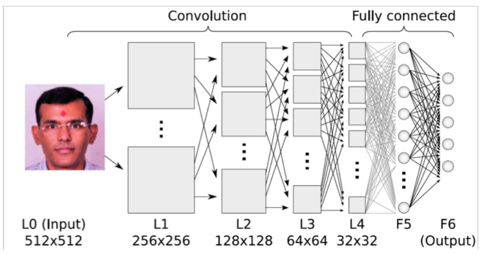


- Max(C,G,B)
- Min(R,G,B)
- Mean(R,G,B)
- Weighted Average
- Common (PhotoShop):

0.2989 * R + 0.5870 * G + 0.1140 * B

Image Importance

- Today: face detectors, object detectors, scene recognition etc…

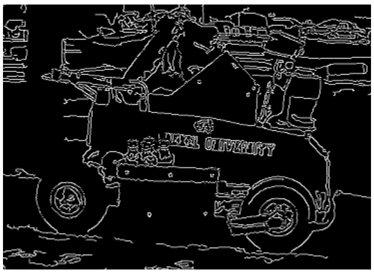


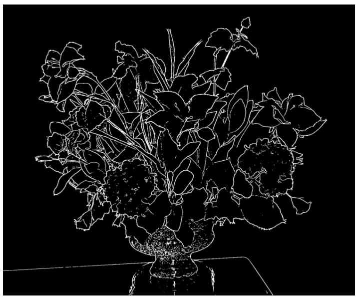
Low Level

- Where is most of the information?
- Most of the world is smooth = information content is low (a pixel is the same as its neighbor)
- Where is it not?

Edges

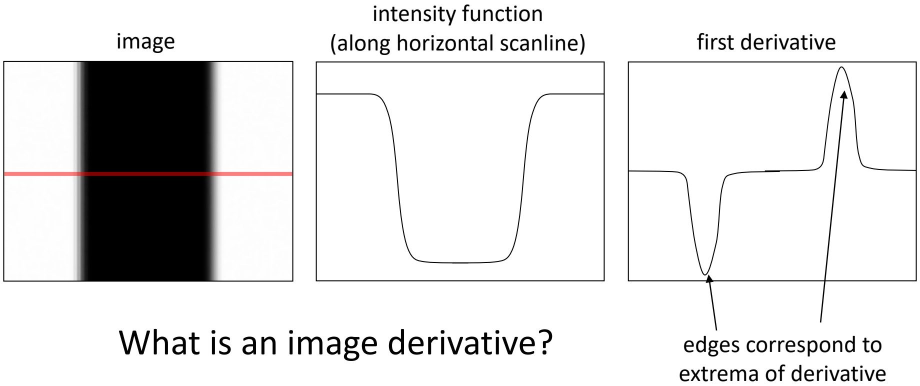
Edges carry most information in the scene





Characterizing Edges

- An edge is a place of rapid change in the image intensity function



What is an image derivative?

Finding Edges

- Edges = discontinuity of various forms
- Function discontinuity → large derivatives

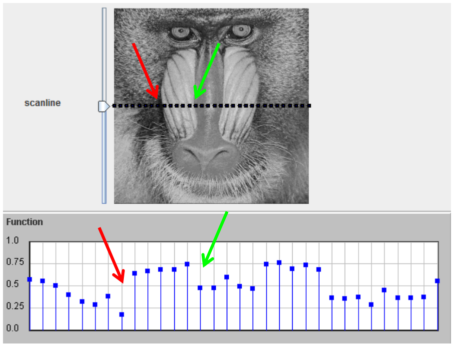


Image Derivatives?

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

© Ariel Shamir IDC

Derivative Approximations

- Remember the definition of the derivative:

$$f'(x) = Lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

- For a small enough Δx the following is a good approximation for the derivative:

$$\frac{f(x+\Delta x) - f(x)}{\Delta x}$$

Finite Difference

- We can approximate the first derivative by

Forward difference: $f'(x) = \frac{f(x+\Delta x) - f(x)}{\Delta x} + O(\Delta x)$

Backward difference: $f'(x) = \frac{f(x) - f(x-\Delta x)}{\Delta x} + O(\Delta x)$

- Or by adding the two (central difference):

$$f'(x) = \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} + O(\Delta x)$$

Pixel Differences

- In an image the smallest Δx (or Δy) is 1 so:

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

We get values:

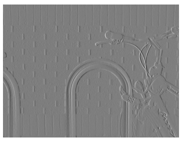
- $I(x,y) \in [0,255] \rightarrow d(x,y) \in [-255,255]$

Mapping to an Image?

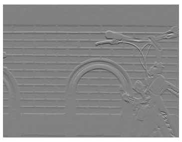
- The values are now between -255 to 255
- How can we visualize these differences?
- We map it back to [0,255] by adding 255 and dividing by 2.
  - Negative values are dark
  - Positive values are light
  - Zero is gray!
- (or we can just take the absolute value – black remains 0)

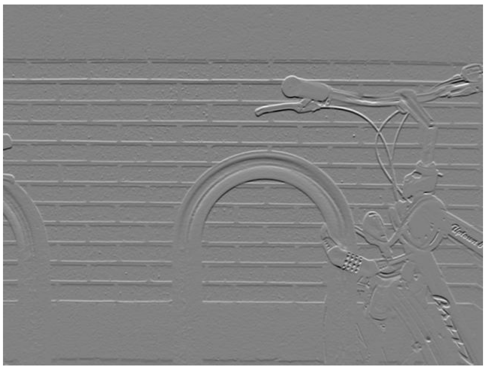
Pixel Differences

dx



dy

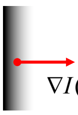




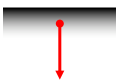
Gradient

- 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.

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

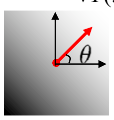


$\nabla I(x,y) = (dx,0)$



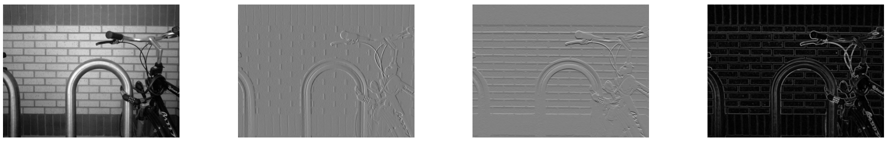
$\nabla I(x,y) = (0,dy)$

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

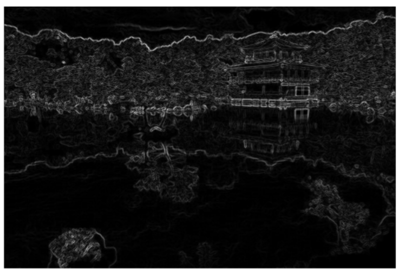


Gradient Size

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

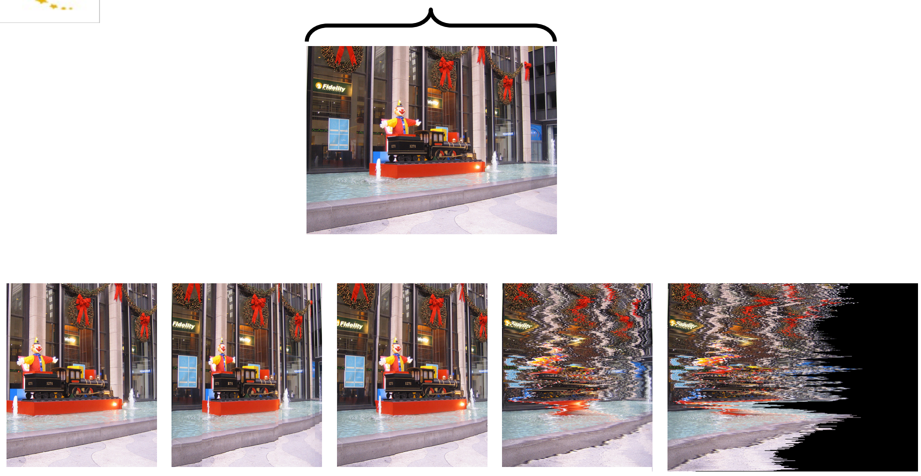


Back to Resizing



Reducing Size?

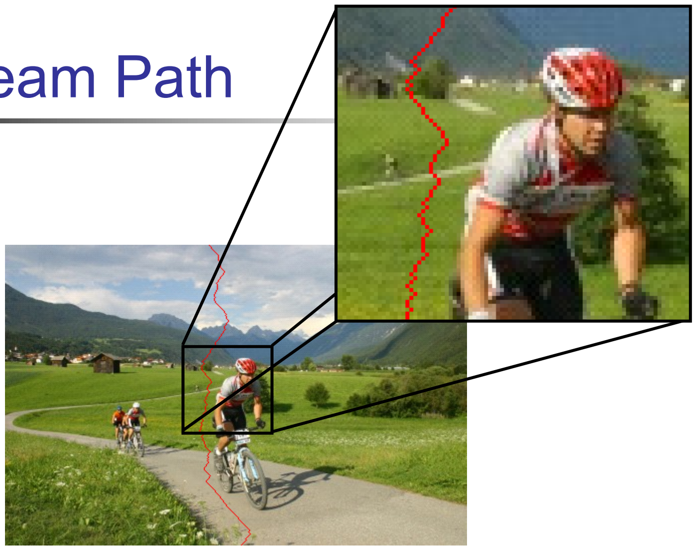
Reduce Width



crop column seam Row pixels Best pixels

Removing Columns

A Seam Path

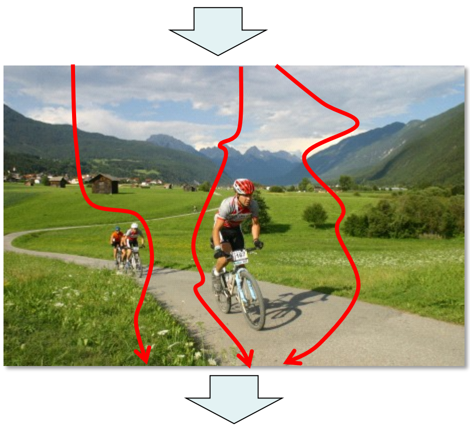


Carving Out One Seam

Width is one pixel smaller

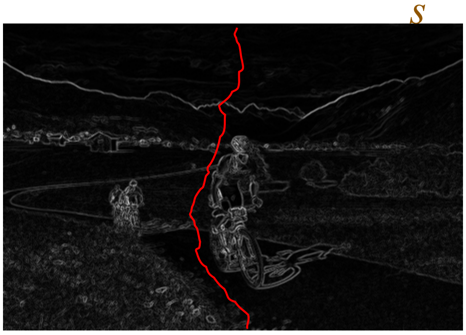
Carving Out Many Seams

Finding the Seam?



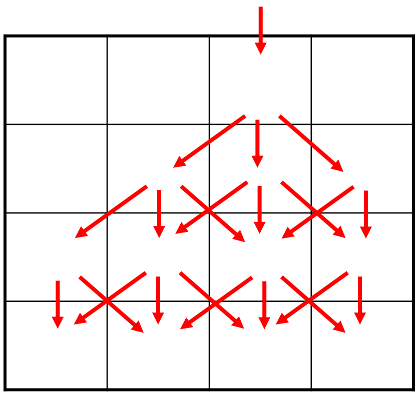
The Optimal Seam

$$E(\mathbf{I}) = \left|\frac{\partial}{\partial x}\mathbf{I}\right| + \left|\frac{\partial}{\partial y}\mathbf{I}\right| \quad \Rightarrow \quad s^* = \arg\min_{s} E(s)$$



Naïve Approach

- Loop over all seams and check their energy E(s). Choose the one with smallest energy.



How Many Seams?

- An image has n columns and m rows
- Start from any pixel at top row (n)
- For each one choose between 3 possible pixels in the next row
- For each one of those, choose between 3 in the next row…
- $n*3^{m-1}$ = exponential! ☹

Pixel Attribute → 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
5	4	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
5	4	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
5	4	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
14	13	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
14	13	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
14	13	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
14	13	15	16

Backtracking the Seam

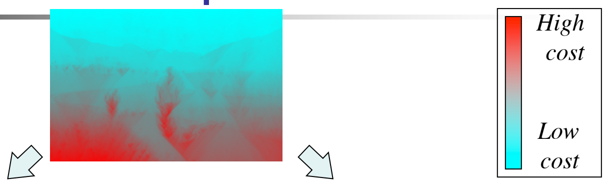
$$M(i, j) = e(i, j) + \min(M(i-1, j-1), M(i-1, j), M(i-1, j+1))$$

5	8	12	3
9	7	6	12
14	9	10	8
14	13	15	16

Dynamic Programming

- A method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions using a memory-based data structure (array, map,etc).
- A problem where the sub-solution is the optimal solution to the sub-problem.
- In our case?

H & V Cost Maps



Horizontal Cost

Vertical Cost

Aspect Ratio Change

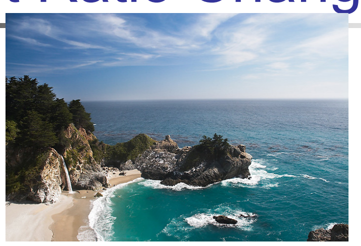
Aspect Ratio Change

Seam Carving

Original

Scaling

Aspect Ratio Change



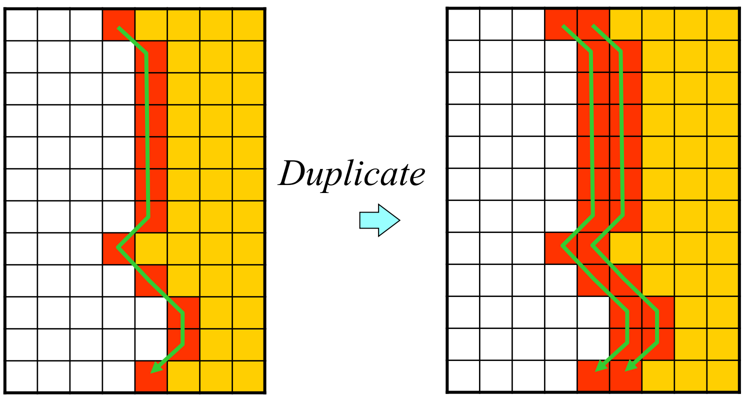
Cropping

Seams

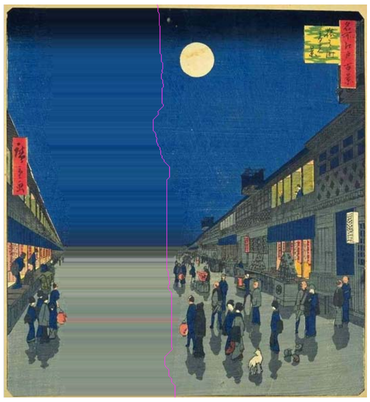
Scaling

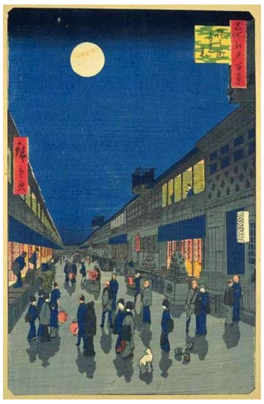
Enlarging an Image?

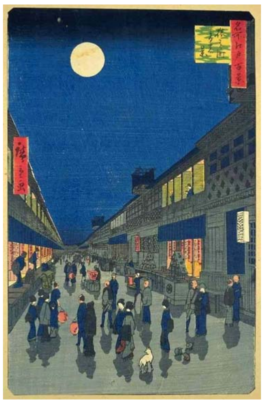
Inserting a Seam?



Width is one pixel larger



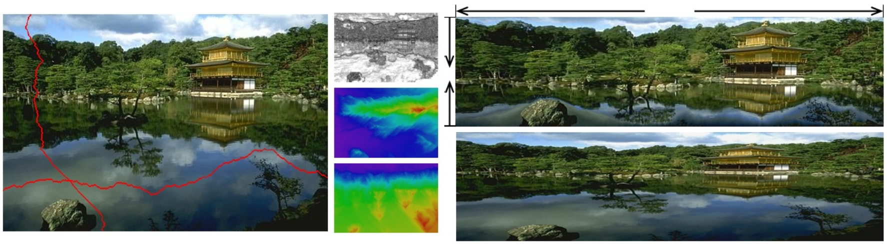




Enlarged or Reduced?

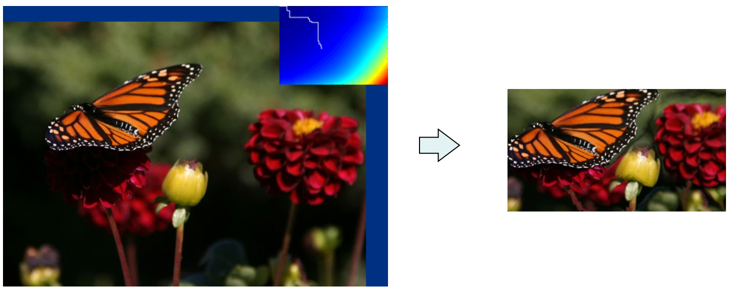
Both Dimensions

- Inserting & Removing



Both Dimensions?

- Remove horizontal seam first?
- Remove vertical seams first?
- Alternate between the two?
- The optimal order can be found! → Dynamic Prog.



Optimal Order Map

Removal of ***vertical*** *seams*

Removal of ***horizontal*** *seams*

0	13	16	19		
16	17	22	28		
19	31	25	35		
24	28	29	???		
32	35	33			
41	38	35			

Optimal?

- Greedy in iterative sense we assume the cost function is monotonic!
- In fact there are many (exponential) ways to get to the desired size (m x n) – we must check all of them but we store only the best of two:
  - (m+1 x n) + (row seam cost)
  - (m x n+1) + (col seam cost)
- Key idea: ratio (of row & column) is more important than order

How Many Paths to (3,2)?

	R	RR	…
		RRC	
	…		
…			

	R		…
	RC	**RCR**	
	…		
…			

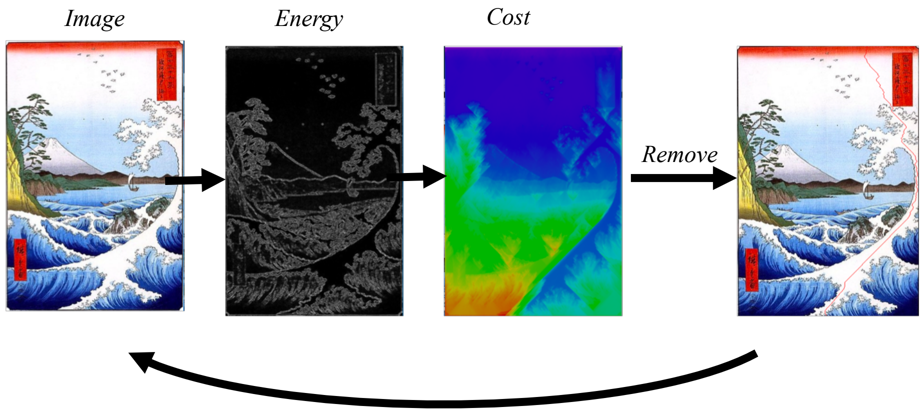
			…
C	CR	**CRR**	
	…		
…			

What did we check?

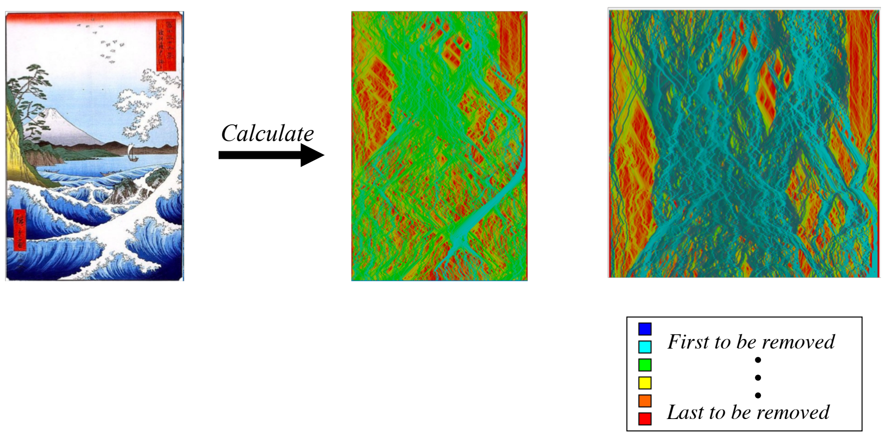
- We find best path to (3,2) by checking **RCR** against **RRC only**
- but maybe **CRR** is better than them? – we didn't check it because we chose **RC** over **CR** to get to the (2,2) entry in the previous stage – and we are bound to this choice!

0		RR	…
C	RC	**?**	
	…		
…			

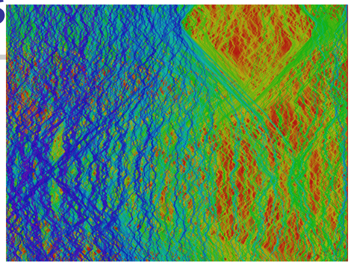
Exercise: Implementation

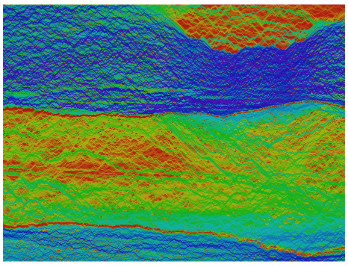


Save the Order



Multi-Size Images





First to be removed

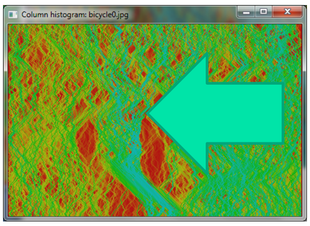
⋮

Last to be removed

Gathering Pixels Row by Row

3	~~1~~	6	5	4	~~2~~	8	7

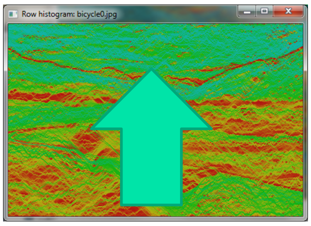
3	6	5	4	8	7



```
Resize width from m to m'
For each row r from 0 to n
    For each column c from 0 to m
        c' = 0
        If seam_index(r,c) > (m-m')
            Copy pixel (r,c) to (r,c')
            c' = c'+1
```

Gathering Pixels by Columns

```
Resize height from n to n'
For each column c from 0 to m
   For each row r from 0 to n
        r' = 0
        If seam_index(r,c) > (n-n')
            Copy pixel (r,c) to (r',c)
            r' = r'+1
```

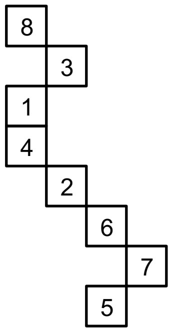


Combining Both Directions?

- Why can't we just interchange?
- When we remove one row seam we must remove one pixel from each column seam!
- Similarly the opposite: when we remove one column seam we must remove one pixel from each row seam!
- This will ensure that we can interchange the operations
- This means that each row seam must contain one pixel from each column seam and vice verse!

“Seam Sudoku”

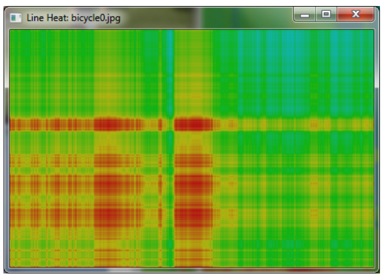
- Each Row seam must include numbers 1…m
- Each Column seam must include numbers 1…n
- Can this be done?



Trivial Solution: Rows & Columns

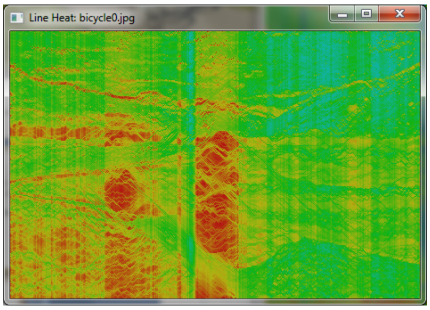
Any permutation of rows & columns:

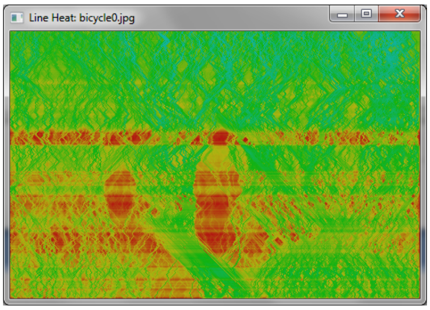
3,2	3,5	3,3	3,1	3,4
1,2	1,5	1,3	1,1	1,4
4,2	4,5	4,3	4,1	4,4
2,2	2,5	2,3	2,1	2,4



Also…

Row seams + columns | Column seams + rows





OPEN QUESTIONS

- **Seams Sudoku**: Row seams & Column seams together?
- Possible directions:
  - Given a non-constrained row-seam order maybe constrain the column seam while we build them (and vice verse)
  - Given a constrained solution (e.g. start with rows & columns) – switch pixel orders to get better seams while preserving the constraints

Not Always a Success

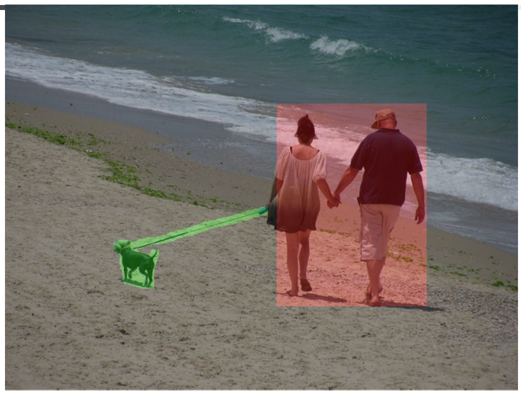
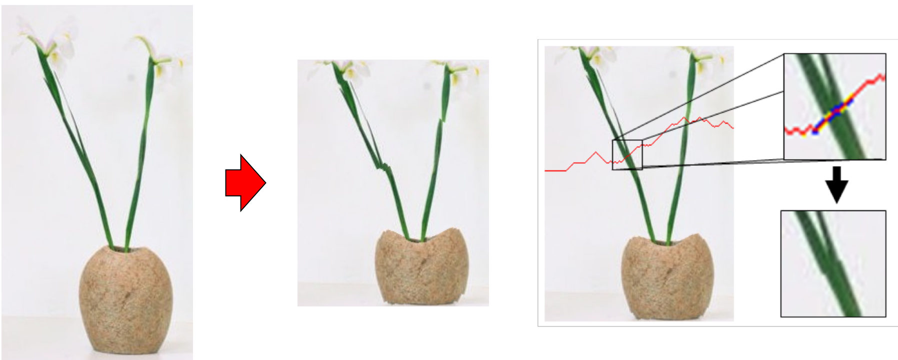
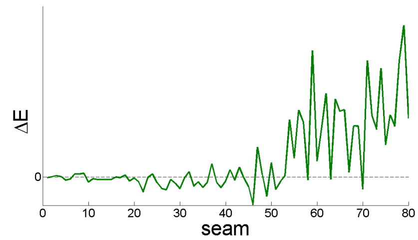


Image: Yehudit Garinkol

Problems…



Change in Energy

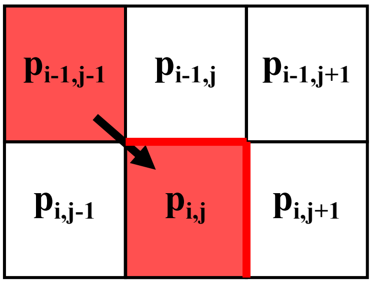


Tracking Inserted Energy

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

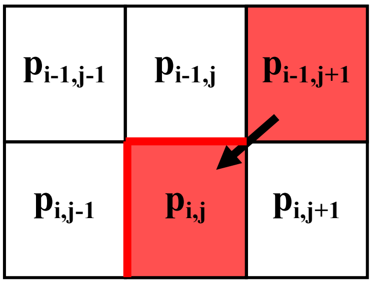
- Three possibilities when removing pixel $P_{i,j}$

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



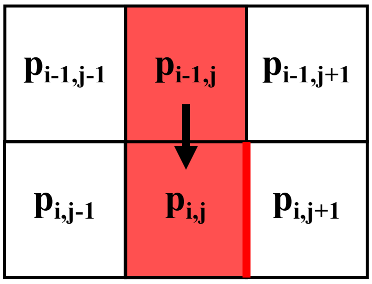
$$C_L(i,j) = |I(i,j+1) - I(i,j-1)|| + |I(i-1,j) - I(i,j-1)|$$

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



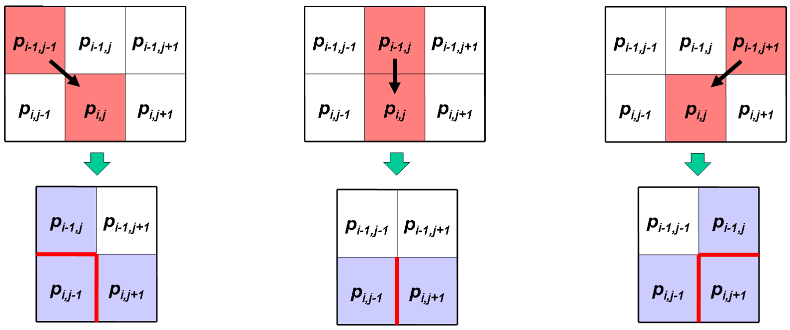
$$C_V(i,j) = |I(i,j+1) - I(i,j-1)|$$

Old "Backward" Energy Function

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

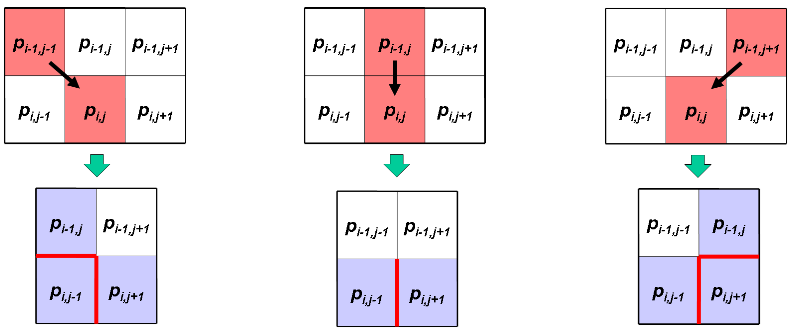
New Forward Looking Energy

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



Adding "Pixel Energy"

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

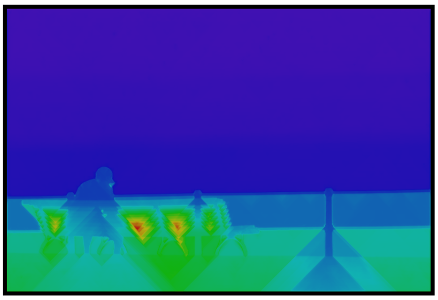


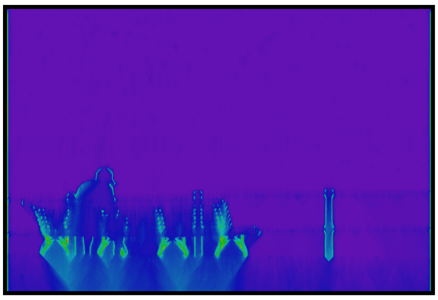
Backward Energy

Forward Energy

Backward Energy

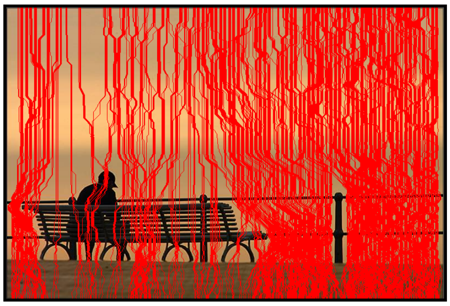
Forward Energy





Backward Energy

Forward Energy



Backward

(c) ariel shamir

Forward

(c) ariel shamir

Backward

(c) ariel shamir

Forward

(c) ariel shamir

Optimization Summary

- Seam Carving: simple dynamic programming
- Choosing Seam Order (H or V): exponential – but we can choose greedy using dynamic prog.
- Seam Soduko: creating multisize image in both direction: exponential + topological constraints

Thank You

- arik@idc.ac.il