Pattern Recognition Lab

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

Panorama & Seam Carving

Making Bigger Images & Making Smaller Images

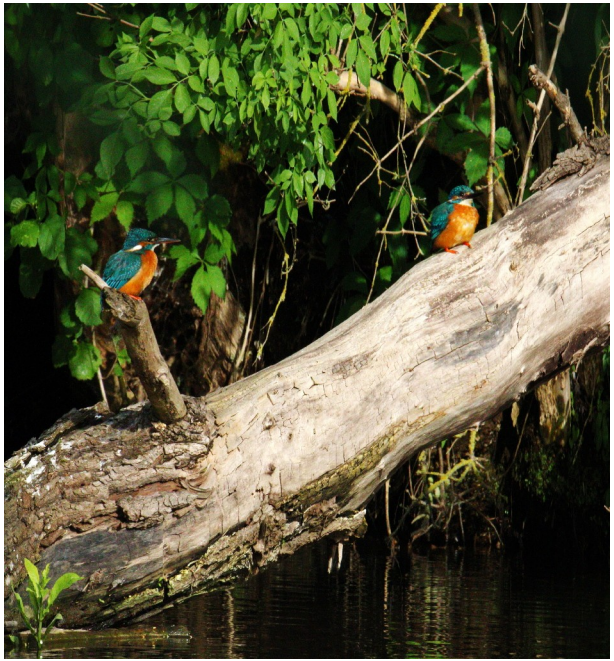
Mathias Seuret

Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

Teaser




Teaser



Menu

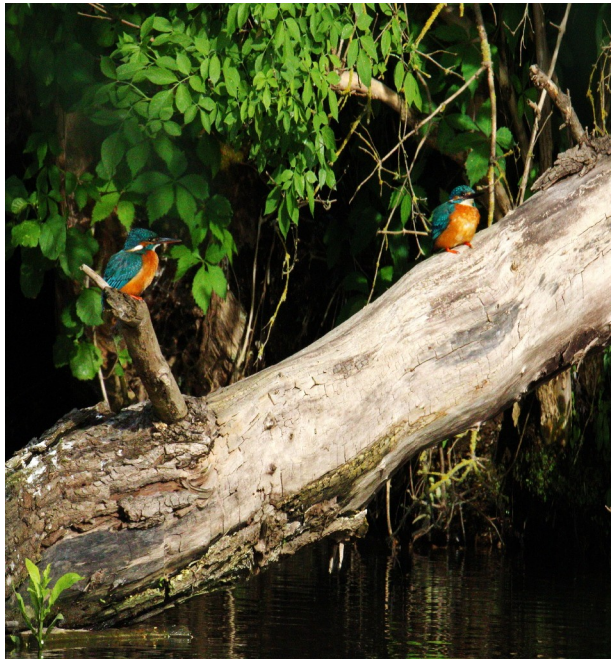
Today's Menu

Panorama creation (starter)
Seam Carving (main course)
Q & A (dessert)

Menu

Today's Menu

Panorama creation (starter)
Seam Carving (main course)
Q & A (dessert)



Part 1

Panorama

Part 1

Panorama

You are guinea pigs for this part

Check for updates on the forum & any honest feedback will be welcome

Problem introduction

We want to align two (or more) images.

Problem introduction

We want to align two (or more) images.

Idea: find corresponding points in both images, and align them

Problem introduction

We want to align two (or more) images.

Idea: find corresponding points in both images, and align them

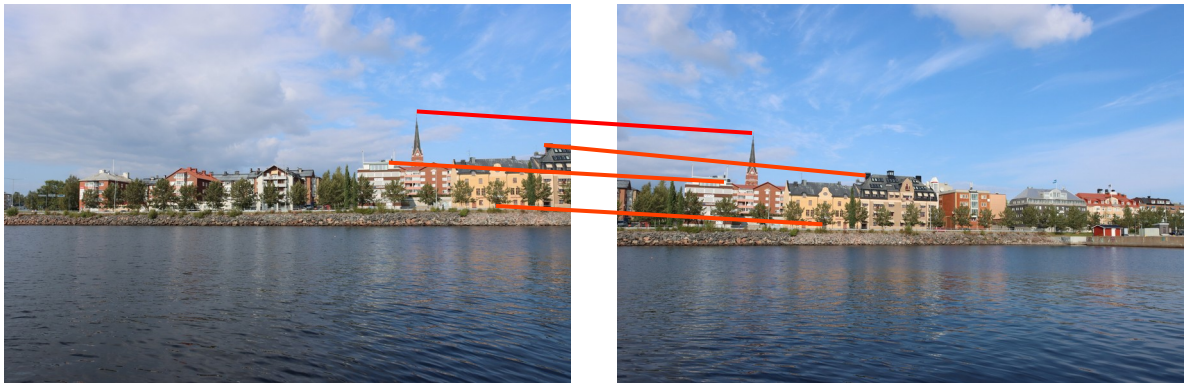
Implication:
- Points detection & matching
- Image transformation

Point Detection & Matching

Points should be discriminative & easy-to-compare

Angles might vary in the photos

Scales as well, especially with wide angle photos



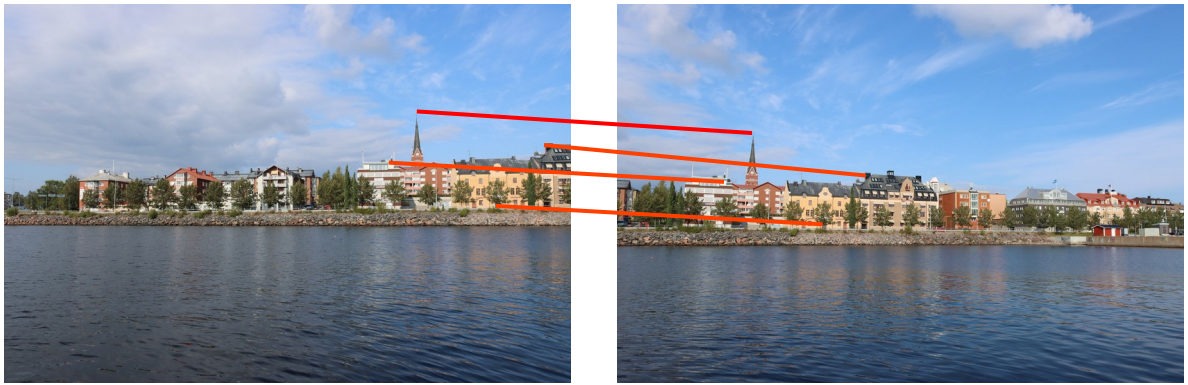
Point Detection & Matching

Points should be discriminative & easy-to-compare

Angles might vary in the photos

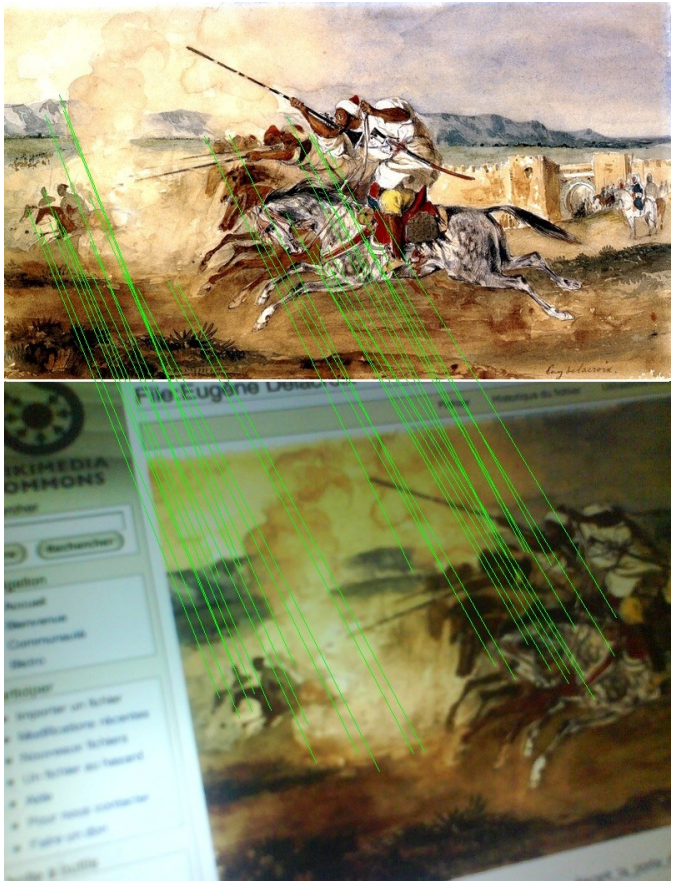
Scales as well, especially with wide angle photos

→ Scale-invariant feature transform (SIFT)



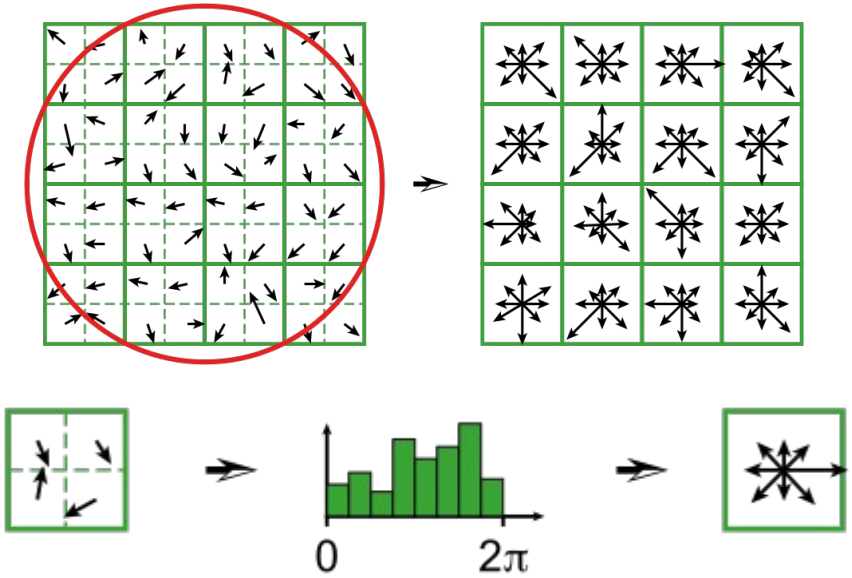
Point Detection

Selection of extremum points (with scales
Detection of elements of ≈ bluring radius

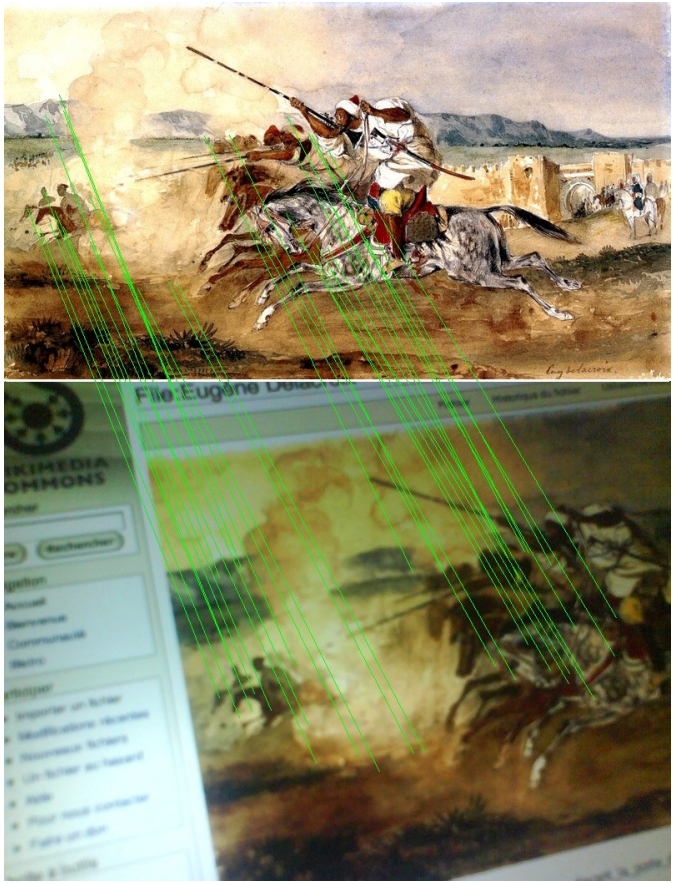


Source of the images: Wikipedia

Descriptor

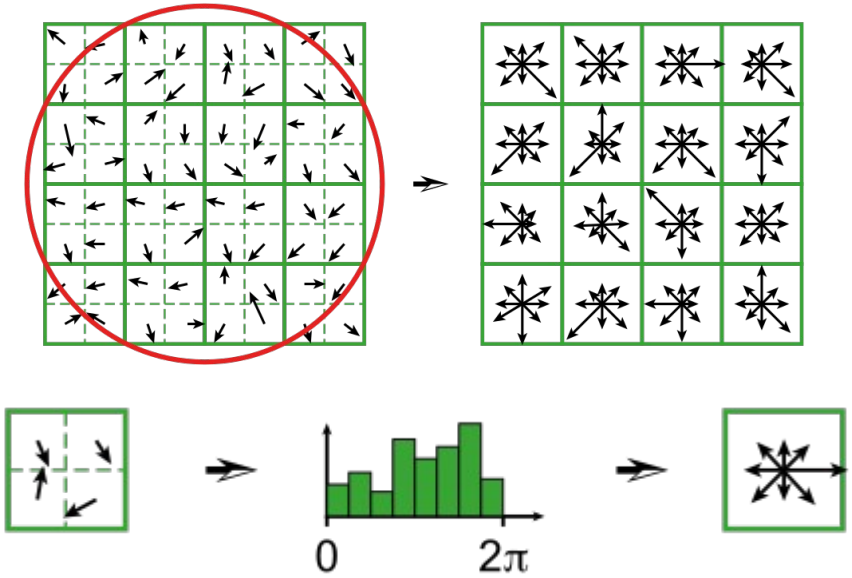


Gradients of 16x16 pixels around selected points are considered
Rotation invariance: gradient-based rotation of patch
Descriptor: concatenation of 8-bins histograms

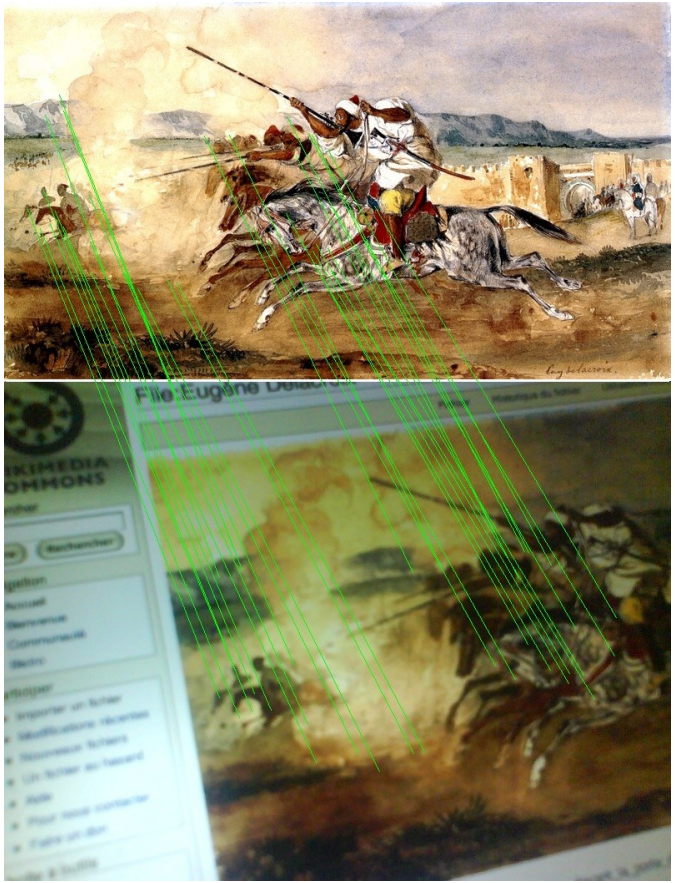


Source of the images: Wikipedia

Descriptor



Matching: comparison of descriptors (e.g., distance)
Warning: there *will* be wrong matches



Source of the images: Wikipedia

Image Transformation

Assuming that we have 4 matches

Compute an **homography**

Divide u and v by s

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ s \end{bmatrix}$$

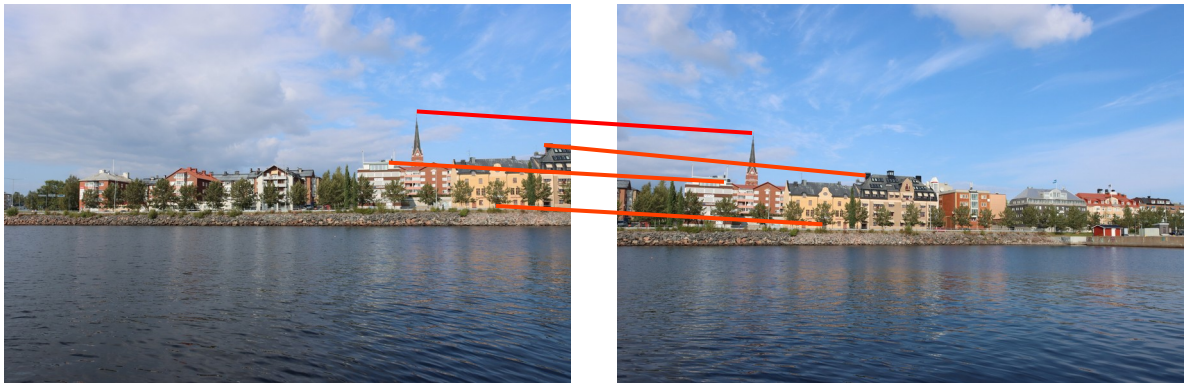


Image Transformation

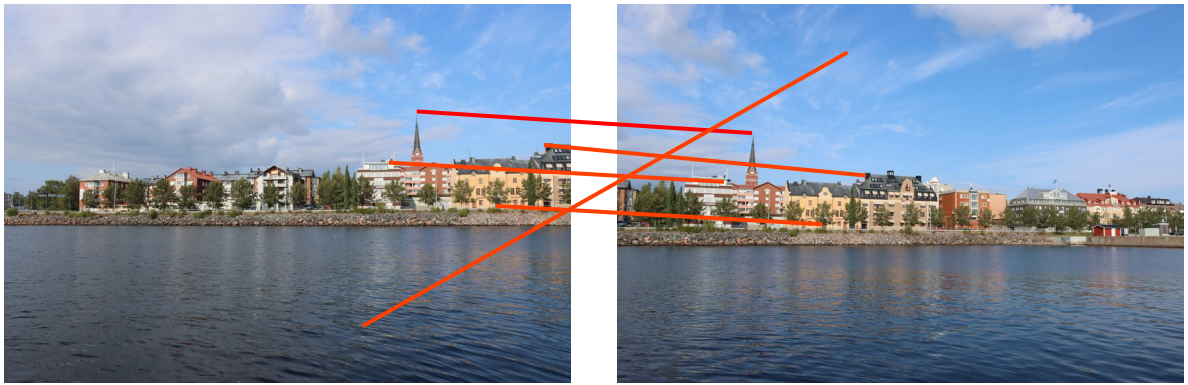
Assuming that we have 4 matches

Compute an **homography**

Divide u and v by s

Do other points get to the correct spot?

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ s \end{bmatrix}$$

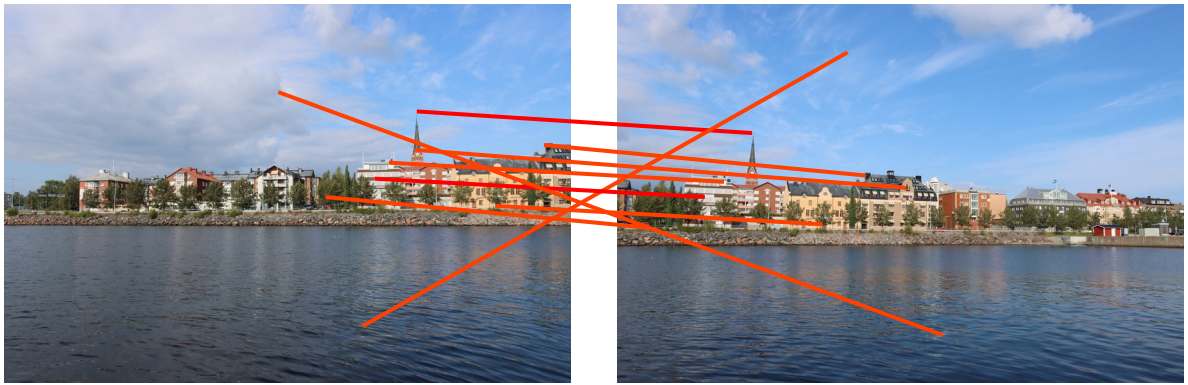


RANdom SAmple Consensus - RANSAC

Some matches are *really* bad

Idea: use 4 random matches, transform all points, count how many get *close enough* (inliers), repeat N times

Recompute the homography using the largest set of inliers



RANdom SAmple Consensus - RANSAC

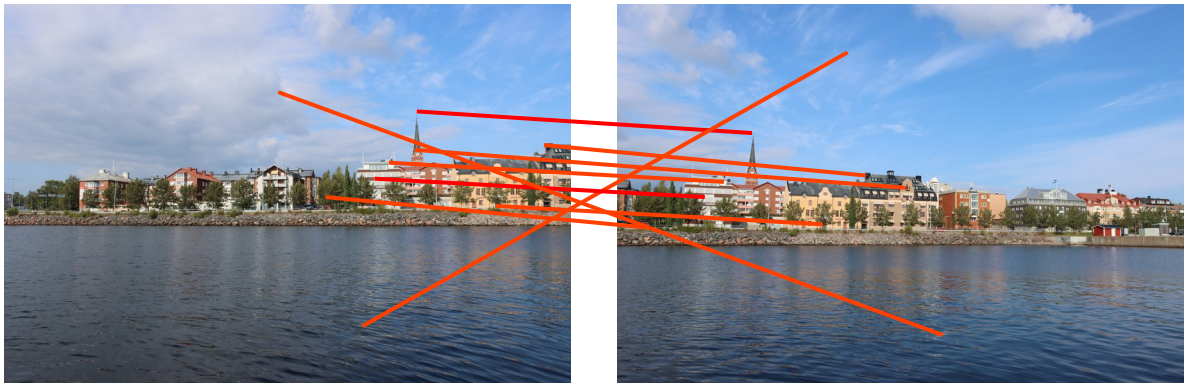
Some matches are *really* bad

Idea: use 4 random matches, transform all points, count how many get *close enough* (inliers), repeat N times

Recompute the homography using the largest set of inliers

Suggestion: $\alpha \cdot$ median distance

Either fixed number of iterations, or
until the set size does not change for a while



Stitching

Images are overlapping; some pixels belong to both

Options:

- Averaging
- Brightest wins
- Furthest from image border wins
- Seam carving (see exercise sheet)

Cheat Sheet

Make full use of Python's useful OpenCV library.

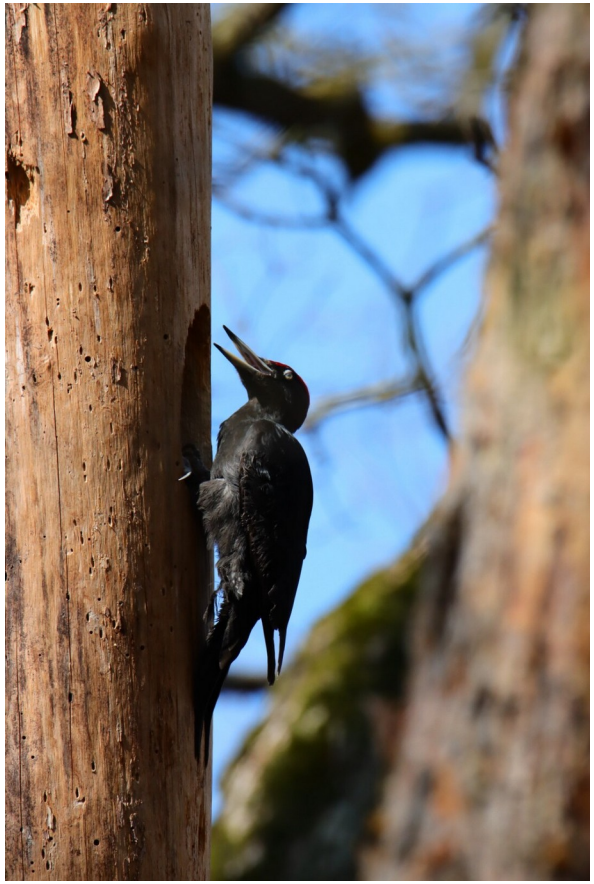
You will want to use:

- SIFT_create
- BFMatcher
- findHomography
- perspectiveTransform
- warpPerspective

Do not hesitate to look for code snippets on Google – there are many available. However, **keep track** of them. If a part of your code is taken from or heavily inspired from some online resource, add a comment with the URL.

Part 2

Seam Carving



Content

Problem introduction

Energy function

Minimum cost seam

Application

- Downscaling
- Upscaling

Problem introduction

This is a high-resolution image. Can we make it smaller?

Problem introduction

This is a high-resolution image. Can we make it smaller?

Resizing: poor results. The bird gets tiny.

Problem introduction

This is a high-resolution image. Can we make it smaller?

Resizing: poor results. The bird gets tiny.

Cropping: problem solved…

… for this image.

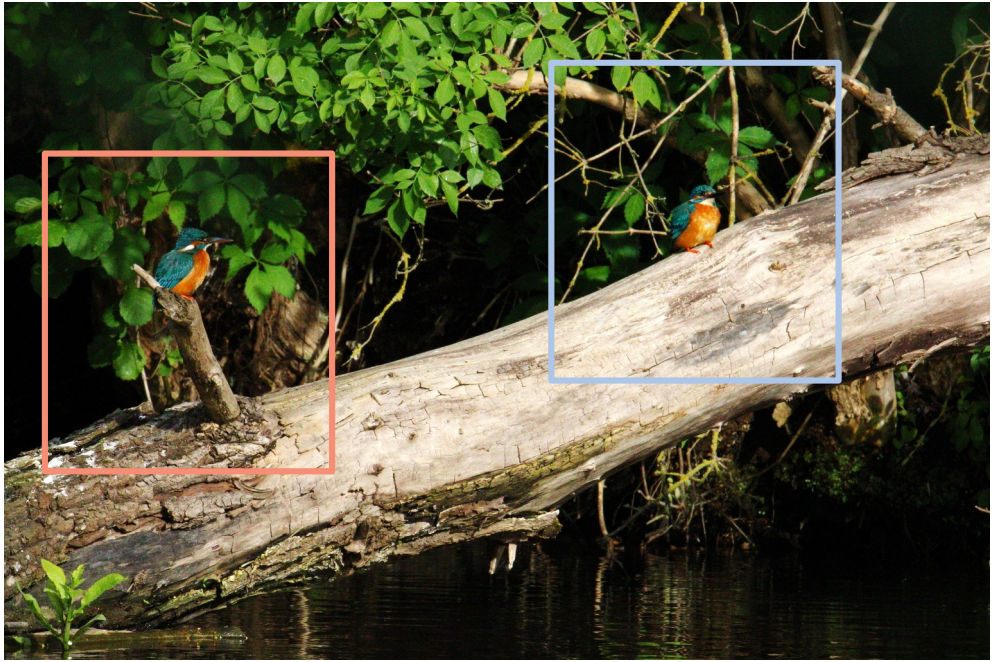
Problem introduction

What about this one?

Problem introduction

What about this one?

Which kingfisher do we keep if we crop?



Problem introduction

Going back to the common kestrel.

Problem introduction

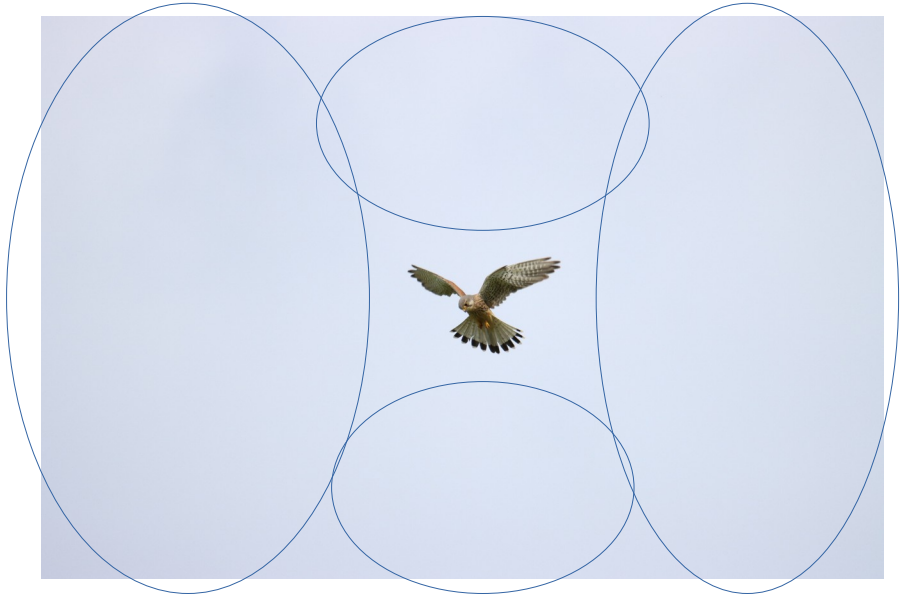
Going back to the common kestrel.

Cropping: removing unimportant pixels.

Problem introduction

Going back to the common kestrel.

Cropping: removing unimportant pixels.

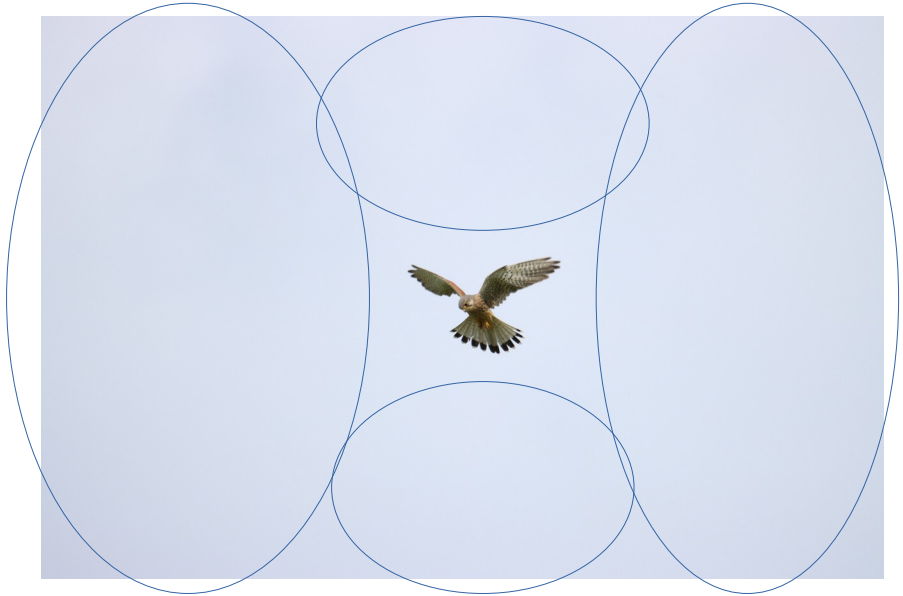


Problem introduction

Going back to the common kestrel.

Cropping: removing unimportant pixels.

Questions:
a) what is an unimportant pixel?
b) what is an important pixel?



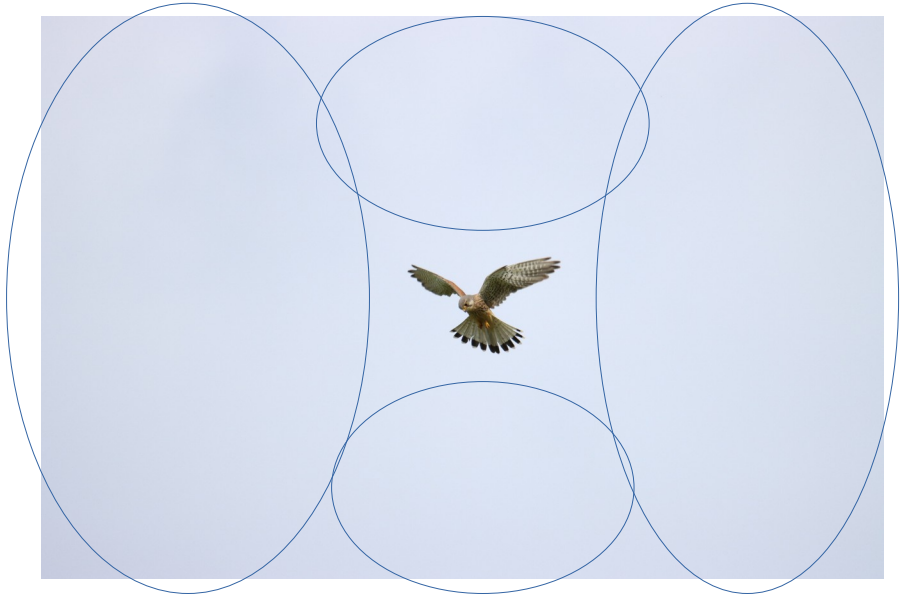
Problem introduction

Going back to the common kestrel.

Cropping: removing unimportant pixels.

Questions:
a) what is an unimportant pixel?
b) what is an important pixel?

(Only) for now:
The importance of a pixel is proportional to how much it differs from its neighbors.



Energy function

Let im be an image and (x,y) be a pixel. Differences with its neighbors are:

```
d_x1 = im(x-1, y) - im(x, y)
d_x2 = im(x+1, y) - im(x, y)
d_y1 = im(x, y-1) - im(x, y)
d_y2 = im(x, y+1) - im(x, y)
```

Energy function

Let im be an image and (x,y) be a pixel. Differences with its neighbors are:

```
d_x1 = im(x-1, y) - im(x, y)
d_x2 = im(x+1, y) - im(x, y)
d_y1 = im(x, y-1) - im(x, y)
d_y2 = im(x, y+1) - im(x, y)
```

We can compute the sum of all d_*

```
d = d_x1 + d_x2 + d_y1 + d_y2
  = im(x-1,y)+im(x+1,y)+im(x,y-1)+im(x,y+1)-4*im(x,y)
```

Energy function

Let im be an image and (x,y) be a pixel. Differences with its neighbors are:

```
d_x1 = im(x-1, y) - im(x, y)
d_x2 = im(x+1, y) - im(x, y)
d_y1 = im(x, y-1) - im(x, y)
d_y2 = im(x, y+1) - im(x, y)
```

We can compute the sum of all d_*

```
d = d_x1 + d_x2 + d_y1 + d_y2
  = im(x-1,y)+im(x+1,y)+im(x,y-1)+im(x,y+1)-4*im(x,y)
```

Surprise: this is the Laplacian operator.

Energy function

Laplacian as a convolution filter:

0	1	0
1	-4	1
0	1	0

Energy function

Laplacian as a convolution filter:

0	1	0
1	-4	1
0	1	0

```
from PIL import Image
import numpy as np
from scipy.signal import convolve
array =
np.array(Image.open('falcon.JPG').convert('L'))
laplacian = np.array([[0, 1, 0],[1, -4, 1],[0, 1, 0]])
res = np.abs(convolve(array, laplacian, 'same'))
res = np.sqrt(res)#for visualization purpose
res = (255*res/np.max(res)).astype(np.uint8)
Image.fromarray(res).save('output.jpg')
```

Energy function

Laplacian as a convolution filter:

0	1	0
1	-4	1
0	1	0

abs(): we care about ‘how much different’, not the sign

```
from PIL import Image
import numpy as np
from scipy.signal import convolve
array =
np.array(Image.open('falcon.JPG').convert('L'))
laplacian = np.array([[0, 1, 0],[1, -4, 1],[0, 1, 0]])
res = np.abs(convolve(array, laplacian, 'same'))
res = np.sqrt(res)#for visualization purpose
res = (255*res/np.max(res)).astype(np.uint8)
Image.fromarray(res).save('output.jpg')
```

Minimum cost seam

Let us start with a simplified approach.

We can delete rows and columns which have the lowest values. They are less important.

Deleting a row/column implies:
- Some pixels get new neighbors
- Update energy function

Minimum cost seam

Let us start with a simplified approach.

We can delete rows and columns which have the lowest values. They are less important.

Deleting a row/column implies:
- Some pixels get new neighbors
- Update energy function

Repeat until you get the desired image size

Minimum cost seam

Problem: terrible deformations!

Minimum cost seam

Problem: terrible deformations!

Poor kingfishers...

Some parts of the image should NOT be modified.

Minimum cost seam

Problem: terrible deformations!

Poor kingfishers...

Some parts of the image should NOT be modified.

Two possibilities:

- Use artificial intelligence to detect important areas (it’s hard)

Minimum cost seam

Problem: terrible deformations!

Poor kingfishers...

Some parts of the image should NOT be modified.

Two possibilities:
- Use artificial intelligence to detect important areas (it's hard)
- Ask the user about it (we'll do this)

Minimum cost seam

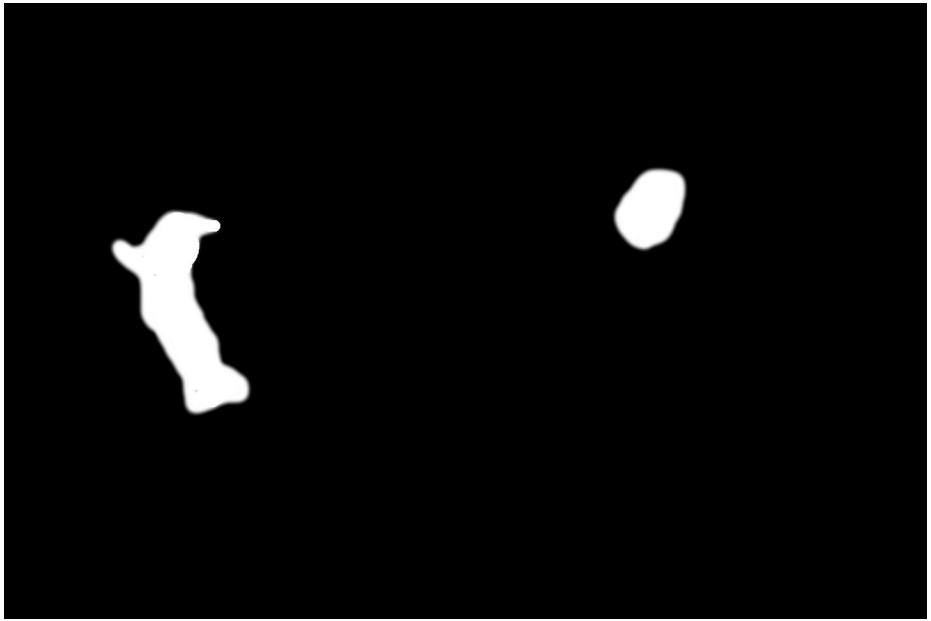
Add values to the energy function, so that the method does not cut through cute birds
The mask image can be hand drawn – no need to develop a GUI!

Minimum cost seam

Masks should have finite values!

Do not use float('inf')

If no cut avoiding white parts is possible, finite values make the cuts minimal.



Minimum cost seam

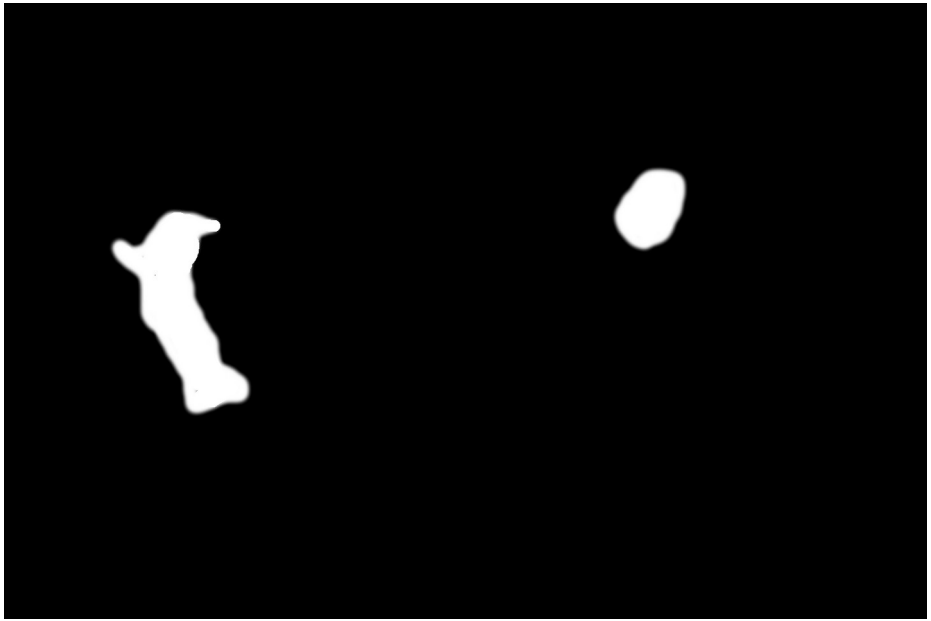
Masks should have finite values!

Do not use float('inf')

If no cut avoiding white parts is possible, finite values make the cuts minimal.

With infinite float values, cutting one or many pixels has the same (infinite) cost.

Extremely high integer values can lead to integer overflow.



Exercise 1

Compute energy function of an image

Remove rows and columns to reach a given size

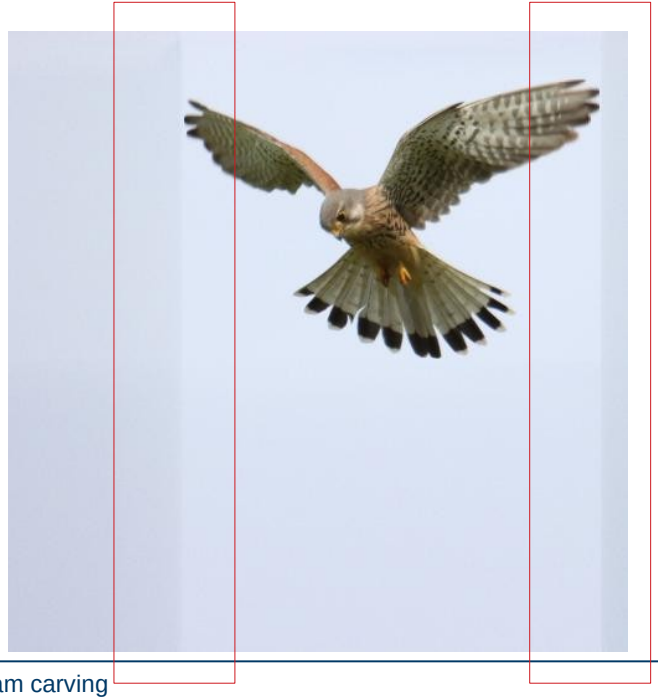
Use masks to preserve some elements

Minimum cost seam

Question for the class: if you have not peeked in the following slides, what is the big problem with the following two images? Hint: it is more visible on the left one

Minimum cost seam

Question for the class: if you have not peeked in the following slides, what is the big problem with the following two images? Hint: it is more visible on the left one



Minimum cost seam

Seam carving is about computing a path which:
a) goes from one side to the other of the image
b) minimizes the cost of the pixels it crosses
c) advances by 1 pixel at each step
d) and can move perpendicularly by N pixels (e.g., N=1)

Minimum cost seam

Seam carving is about computing a path which:
a) goes from one side to the other of the image
b) minimizes the cost of the pixels it crosses
c) advances by 1 pixel at each step
d) and can move perpendicularly by N pixels (e.g., N=1)

From *all* possible paths across the image, we want to find *the one* which has the lowest sum of energy, or cost.

Minimum cost seam

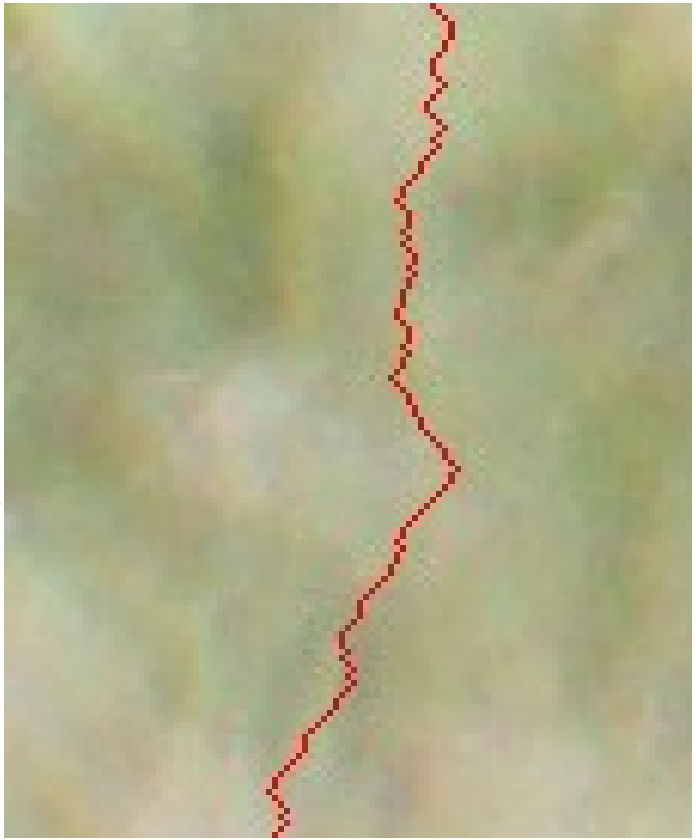
Seam carving is about computing a path which:
a) goes from one side to the other of the image
b) minimizes the cost of the pixels it crosses
c) advances by 1 pixel at each step
d) and can move perpendicularly by N pixels (e.g., N=1)

From *all* possible paths across the image, we want to find *the one* which has the lowest sum of energy, or cost.

Such algorithms are called ‘path search’, or ‘path finding’ algorithms

Minimum cost seam

Seam carving is about computing a path which:
a) goes from one side to the other of the image
b) minimizes the cost of the pixels it crosses
c) advances by 1 pixel at each step
d) and can move perpendicularly by N pixels (e.g., N=1)



From *all* possible paths across the image, we want to find *the one* which has the lowest sum of energy, or cost.

Such algorithms are called ‘path search’, or ‘path finding’ algorithms

Minimum cost seam

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

Minimum cost seam

Where would you cut vertically this cost array?

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

Minimum cost seam

Where would you cut vertically this cost array?

Do you know any algorithm for this?

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

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

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

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

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

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

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

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

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

Min
1

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

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

Min
1

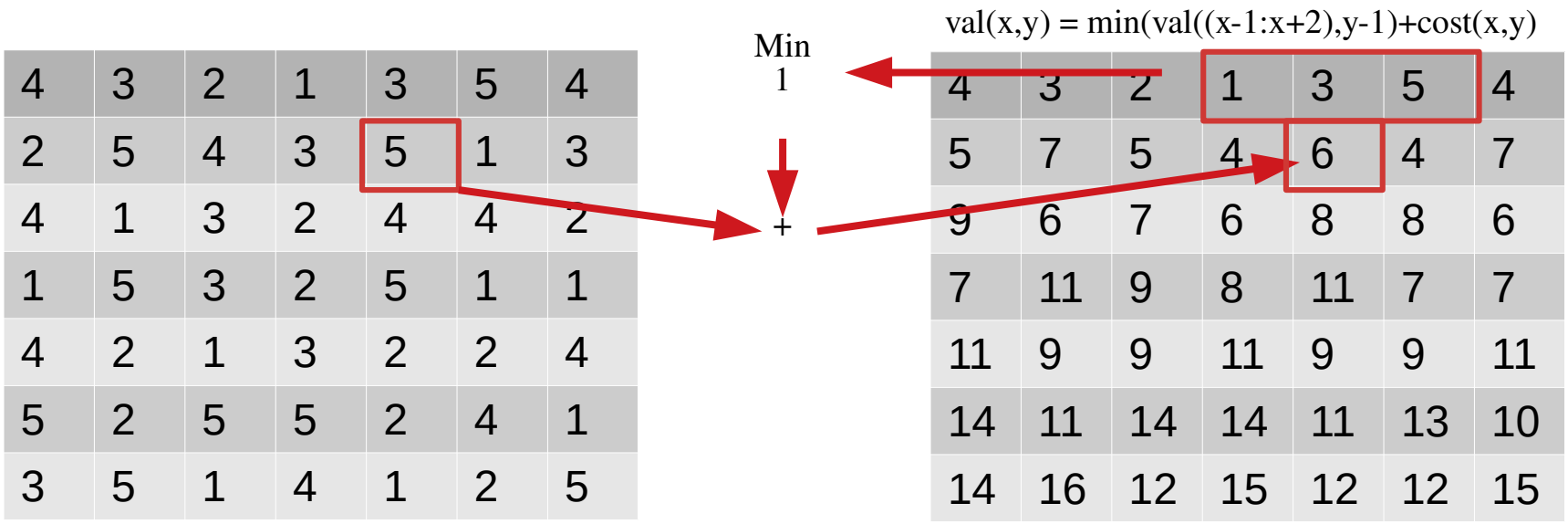
val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.



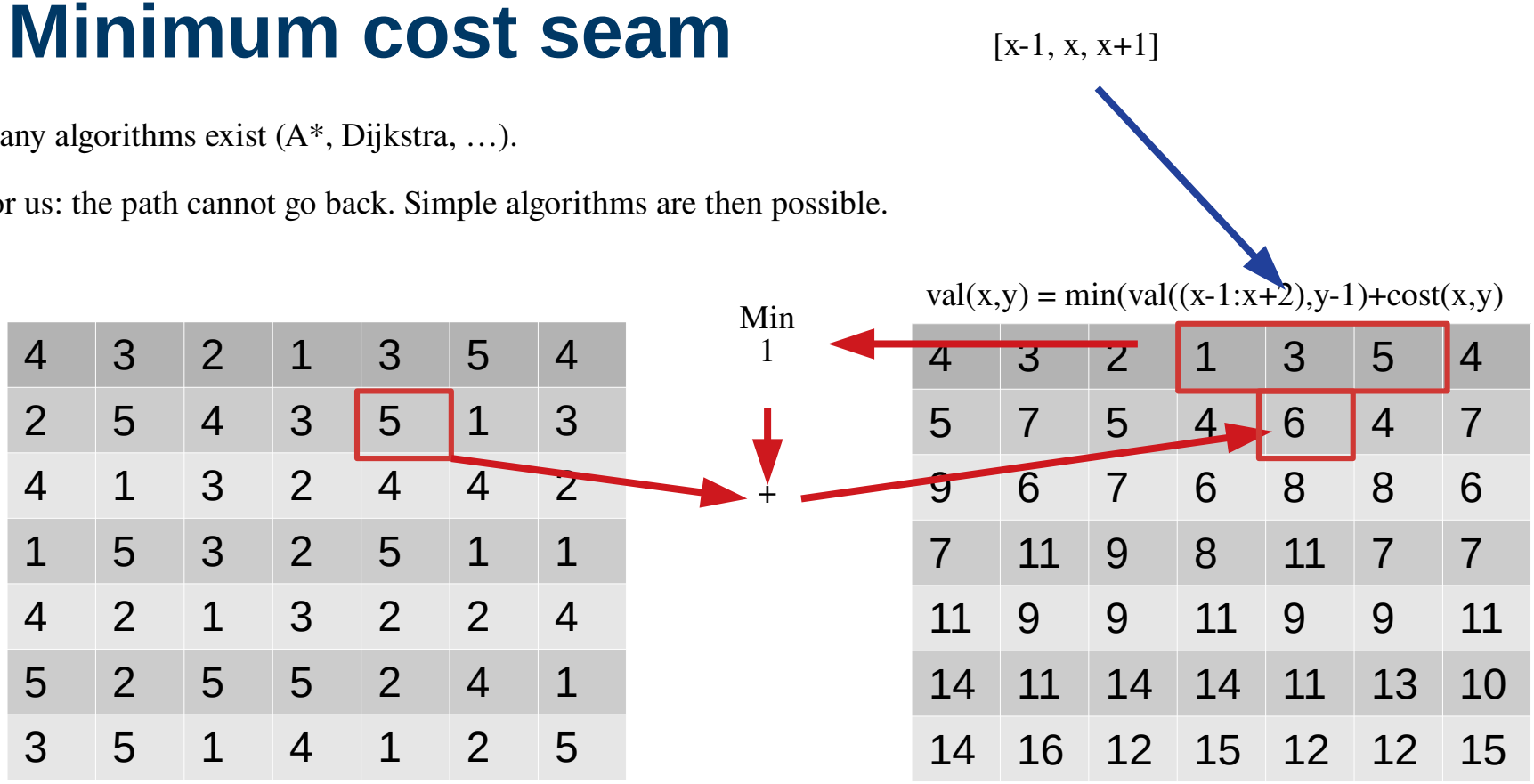
4	3	2	1	3	5	4
2	5	4	3	5	1	3
4	1	3	2	4	4	2
1	5	3	2	5	1	1
4	2	1	3	2	2	4
5	2	5	5	2	4	1
3	5	1	4	1	2	5

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.



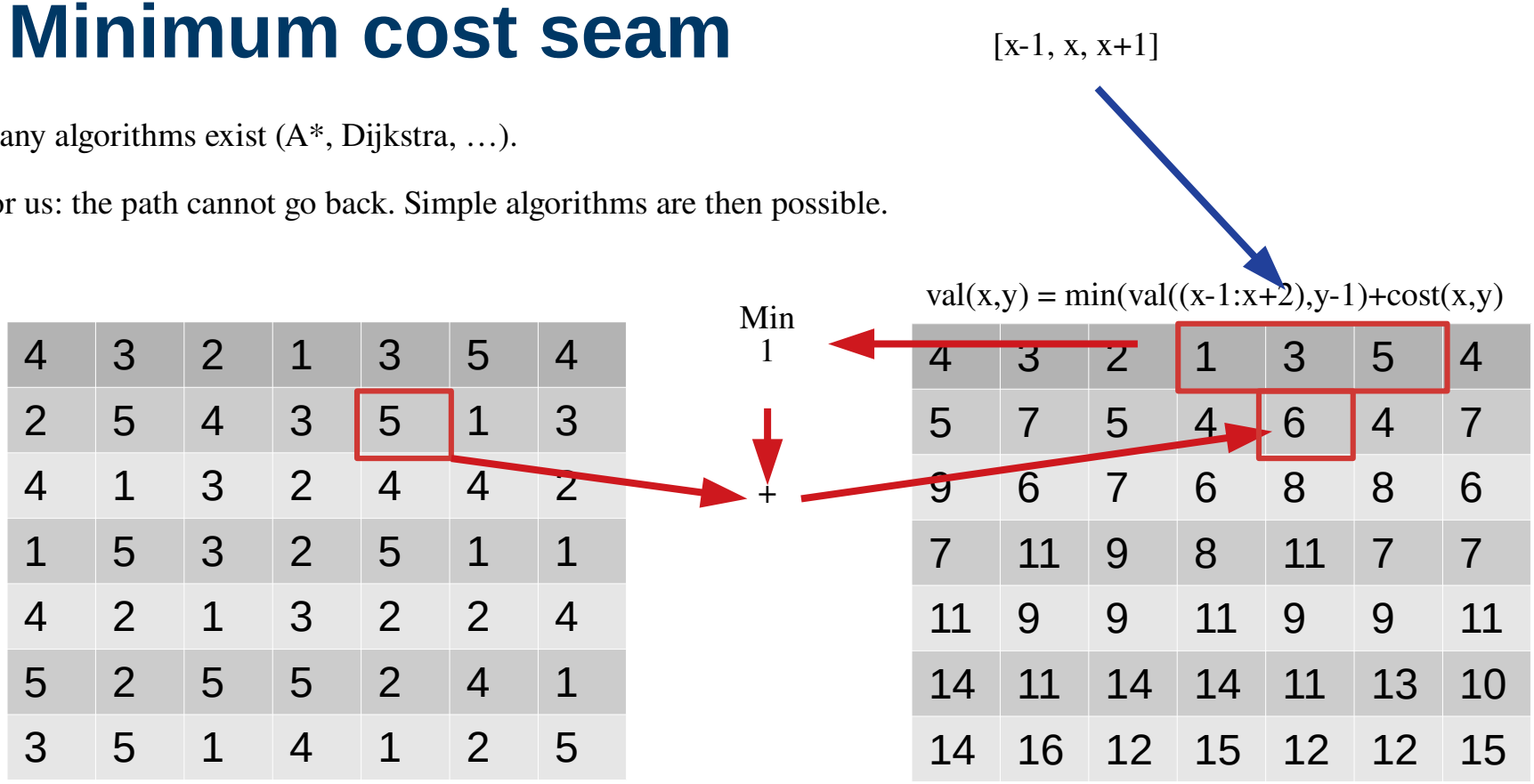
4	3	2	1	3	5	4
2	5	4	3	5	1	3
4	1	3	2	4	4	2
1	5	3	2	5	1	1
4	2	1	3	2	2	4
5	2	5	5	2	4	1
3	5	1	4	1	2	5

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.



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

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

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

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

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

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

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

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

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

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

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

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

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

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

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

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

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

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

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

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

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

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

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

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

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

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

4	**3**	2	1	3	5	4
2	5	4	3	5	1	3
4	**1**	3	2	4	4	2
1	5	3	2	5	1	1
4	**2**	1	3	2	2	4
5	**2**	5	5	2	4	1
3	5	**1**	4	1	2	5

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

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

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

4	**3**	2	1	3	5	4
2	5	4	3	5	1	3
4	**1**	3	2	4	4	2
1	5	3	2	5	1	1
4	**2**	1	3	2	2	4
5	**2**	5	5	2	4	1
3	5	**1**	4	1	2	5

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Minimum cost seam

Many algorithms exist (A*, Dijkstra, …).

For us: the path cannot go back. Simple algorithms are then possible.

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

val(x,y) = min(val((x-1:x+2),y-1)+cost(x,y)

4	3	2	1	3	5	4
5	7	5	4	6	4	7
9	6	7	6	8	8	6
7	11	9	8	11	7	7
11	9	9	11	9	9	11
14	11	14	14	11	13	10
14	16	12	15	12	12	15

Exercise 2

Implement this path finding algorithm

Try it on the illustration array

Apply it horizontally and vertically. Either:

- Simple: transpose the arrays „manually“
- Better: have specialized functions

Exercise 3

Rescale the array using seam carving

Compare the result of three horizontal and three vertical seam carvings, when:

- The vertical are applied first, then the horizontal
- Iterate one horizontal, then one vertival cut, and so on

Exercise 4

Adjust your method for images

- Take into account the mask to preserve elements
- Start with small, RGB images

You have three channels

- Compute energy in grayscale
- All channels must use the same seams

Include a mask for preserving some elements of the image

Try this on some pictures of your choice

Upscaling images

Images can also be upscaled with this method.

Instead of removing pixels, insert new ones.

Upscaling images

Images can also be upscaled with this method.

Instead of removing pixels, insert new ones.

Expand image, offset data, and fill the gap with mean value.

Upscaling images

Images can also be upscaled with this method.

Instead of removing pixels, insert new ones.

Expand image, offset data, and fill the gap with mean value.

What is the problem?

Upscaling images

Images can also be upscaled with this method.

Instead of removing pixels, insert new ones.

Expand image, offset data, and fill the gap with mean value.

What is the problem?

Upscaling images

Images can also be upscaled with this method.

Instead of removing pixels, insert new ones.

Expand image, offset data, and fill the gap with mean value.

What is the problem?

The expanded optimal path is still optimal. It is then expanded again, and again...

Upscaling images

Solution:

Do you remember the mask image?

Upscaling images

Solution:

Do you remember the mask image?

Upscaling images

Solution:

Do you remember the mask image?

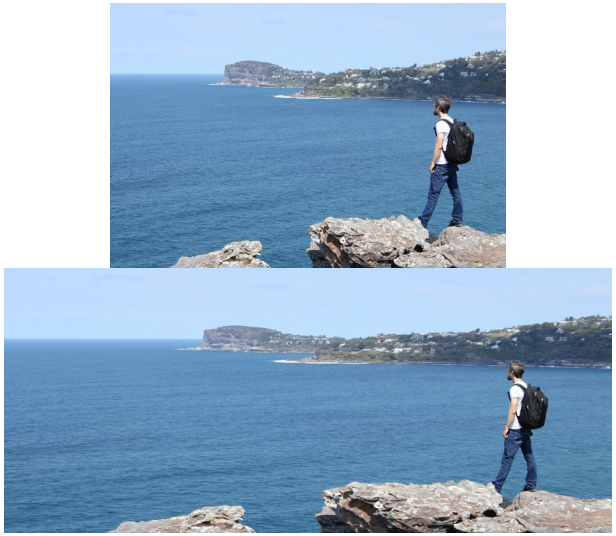
Add the seams to it with a low intensity.

It will prevent the algorithm to always expand the image at the same place.

Note: you can also use this when downscaling images.

Upscaling images

Upscaling images



Upscaling images

Exercise 5

Implement the image upscaling as described above.

Try it on landscapes and cityscapes

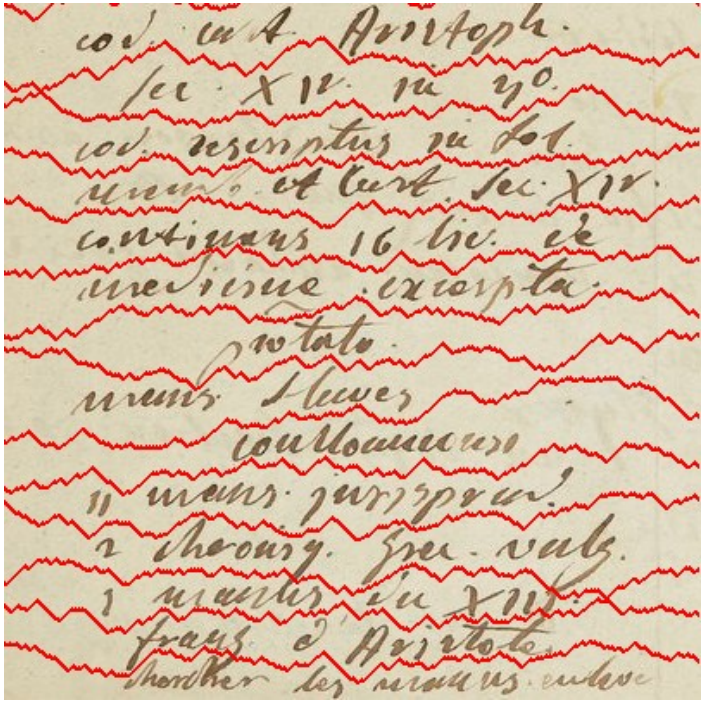
- Which kind of picture works best?
- Can you explain why?

Exercise 6 (optional)

Apply seam carving to some images of your choice

Post in the forum your best (or funniest) result

Other applications



Thank you for your attention

(they finally got closer without help)