Lab 11

Basic Image Processing
Fall 2018

Disclaimer

Accurate and important details regarding Lab11 materials (both theory & practice) were presented on the whiteboard of the classroom, upcoming slides are only sketchy summary of those informations.

SIFT

Scale Invariant Feature Transform

Feature: “is a piece of information which is relevant for solving the computational task related to a certain application” (Wikipedia)

Descriptor: characterization of a feature or keypoint

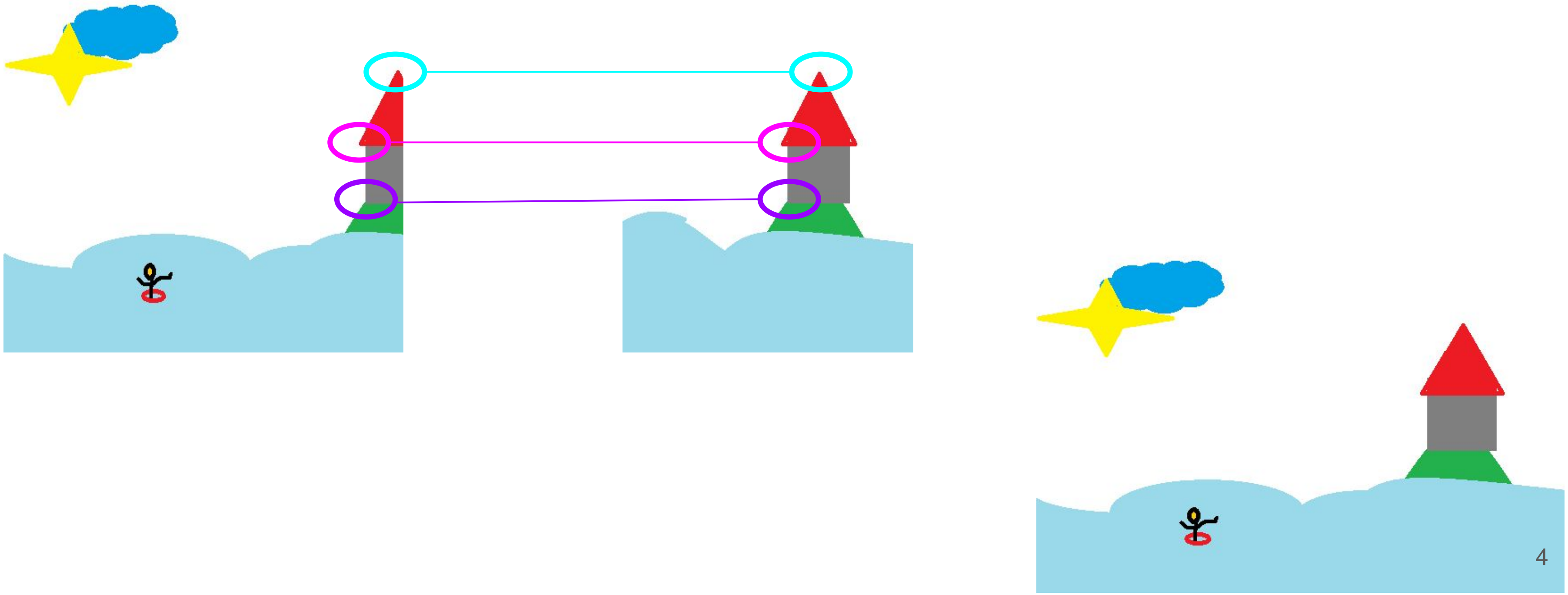
(these are superficial definitions, please see Lecture slides for more accurate shaping)

Main steps:

1. scale-space extrema detection (*build the pyramid*)
2. keypoint localization (*local extrema selection + rejection of weak candidates*)
3. orientation assignment
4. keypoint description

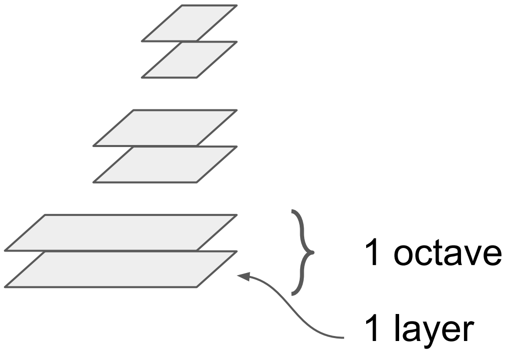
Why is it good for us?

(example) aim at: have a match between the “appropriate” parts of the images



Definitions

image pyramid:



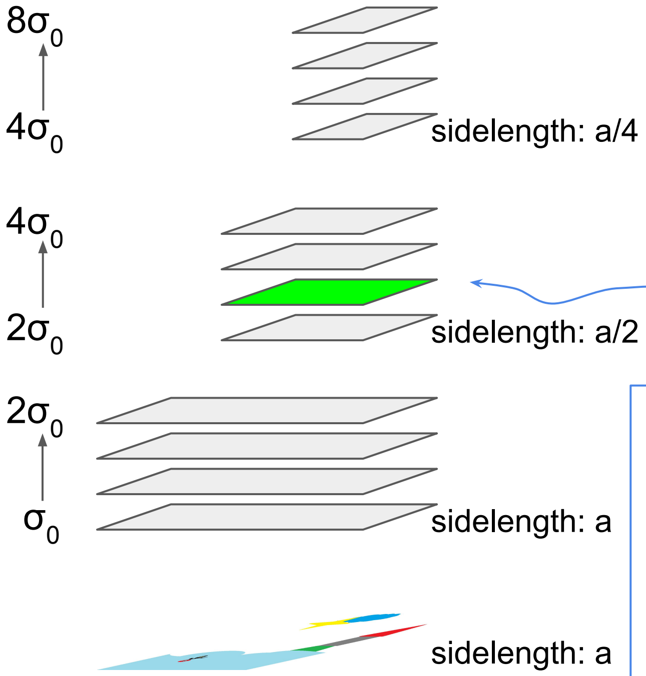
σ: parameter of Gaussian smoothing

# of layers (inside an octave) --- *this is the desired/final number of layers*

# of octaves

How to produce a pyramid?

the pyramid initially:



a layer inside *octave2*, blurred with a Gaussian, using $\hat{\sigma} \in [2\sigma_0, 4\sigma_0]$ standard deviation

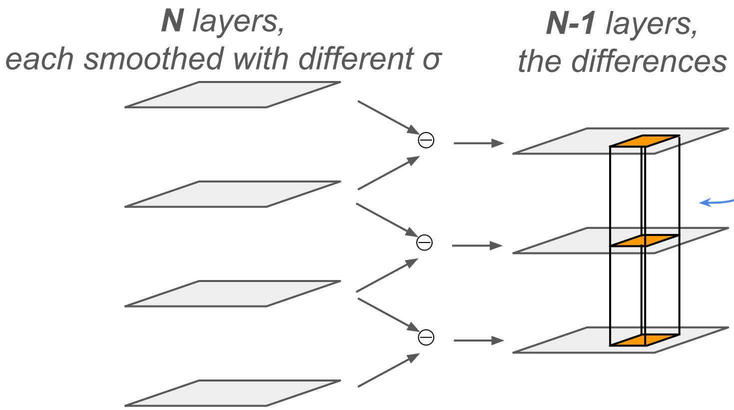
$$\hat{\sigma} = \sigma_{scaled} * 2^{\frac{s-2}{NOL}}$$

where

σ_{scaled}	is the octave-starting σ	(= $2\sigma_0$)
s	is the layer index inside the octave	(= 2)
NOL	is the number of layers inside one octave	

inside one octave:

- LoG* would be welcomed as edge enhancement on the layers …
- … but DoG** is computationally far more cheaper



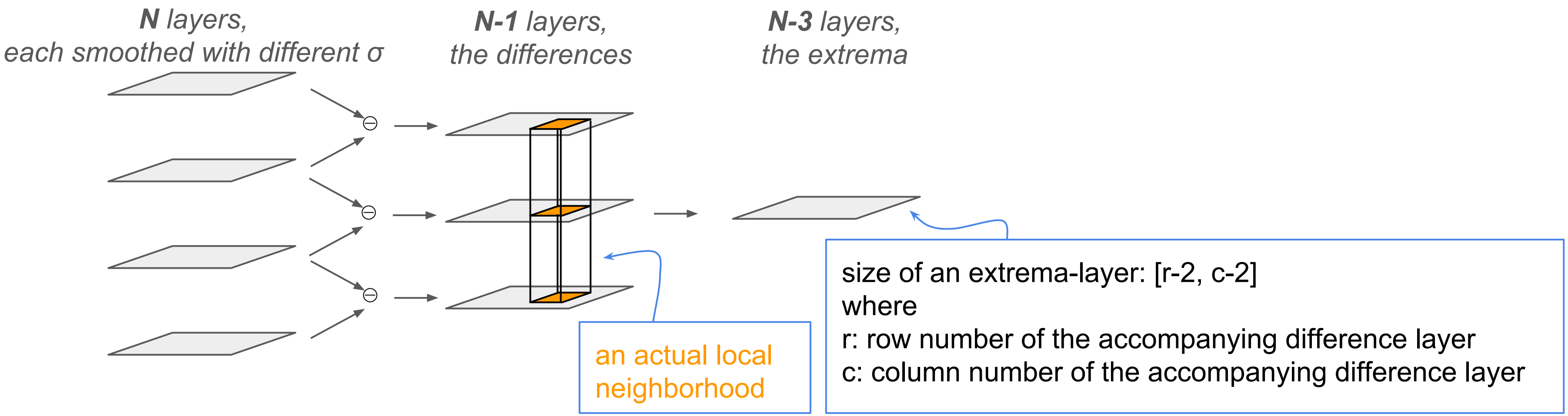
an actual local neighborhood: they are used for local etrema search --- see upcoming slide

* LoG: Laplacian of Gaussian (see slides 53-56 of Lecture 2)
** DoG: difference of Gaussians

Local extrema search

… the magical moment of the foam rubber made from polystyrene

If a central pixel in the 3x3x3 local neighborhood is extremum (min or max), it will be a keypoint-candidate.



Thoughts about the different layer-numbers inside the same octave:

1. initially we use N layers (and smooth them with different σ-s)
2. after calculating the differences, we lose 1 layer
3. during the extrema-search: we are interested only in “true” local neighborhoods, so the top and bottom difference layers in the octave could not be analyzed for extrema (as they do not have real upper/lower neighboring layer, resp.), furthermore: for the same reason, the boundary pixels could not be extrema, too ⇒ we lose 2 more layers

This “N-3” will be equal to the *desired* number of layers (Definitions on slide 5)

Now please

download the ‘Lab 11’ code package

from the

[**submission system**]

Exercise 1

Complete the function `sift_build_pyramid` in which implement the body of the loop:

- input and output of the function is the `params` structure, which contains all of the necessary parameters in its fields,
- the header of the loop iterates through all of the *octaves* in the pyramid, at every `octave` please:
  - calculate a `scaled_sigma` (σ_{scaled}= `DefaultSigma` * $2^{octave-1}$) and a `scaled_image` (`InputImage` resized with the built-in `imresize`, at scale $2^{-octave+1}$),
  - create the 3D array of layers (`layers`) with size [`r`, `c`, `NumberOfLayers`+3], where `r` and `c` are the row- and column-numbers of the `scaled_image` itself,
  - with a `for`-loop (cycle index: `s`), iterate along the 3rd dimension of `layers`, at every step
    - calculate the actual standard deviation ($\hat{\sigma}$ on Slide 6, *NOL* = `NumberOfLayers`),
    - overwrite the `s`-th layer of `layers` with the gauss-filtered `scaled_image` (use the built-in `imgaussfilt` with standard deviation $\hat{\sigma}$),
- after the outer `for`-loop, save the layer-wise differences of `layers` into the `octave`-th cell of the `pyramid` field of the `params` struct (use the built-in `diff` on `layers`, with parameters indicating 1st (order) differences and along 3rd dimension --- see Help).

Test 1

Run the script `SIFT_script`:
if the implementation is correct, you should get the struct `params` and an empty vector `SIFT` in the workspace.

Let's examine the different layers inside the pyramid --- all the details are contained in the `params` struct:

```
figure;
subplot(2, 2, 1); imagesc(params.pyramid{1}(:, :, 1)); title('octave1, layer1');
subplot(2, 2, 2); imagesc(params.pyramid{1}(:, :, end-1)); title('octave1, layer last-1');
subplot(2, 2, 3); imagesc(params.pyramid{2}(:, :, 1)); title('octave2, layer1');
subplot(2, 2, 4); imagesc(params.pyramid{2}(:, :, end-1)); title('octave2, layer last-1');
```

Things to observe:

- upper row: same image-size, different blurring,
- lower-left: half of the side-length, but same blurring with upper-right.

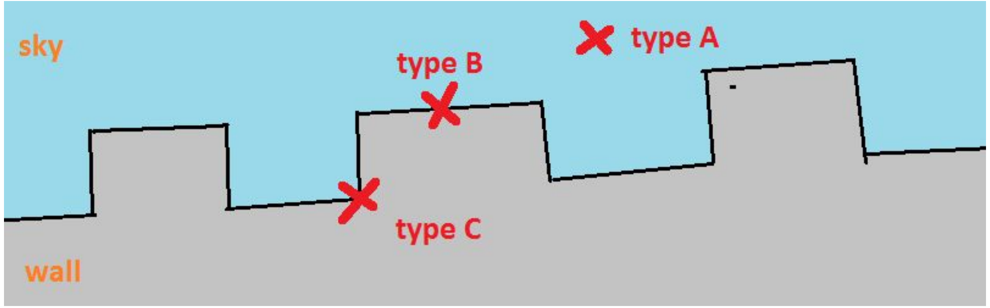
Exercise 2

Complete the function `sift_find_extrema` in which implement the body of the loop:

- input and output of the function is the `params` structure, which contains all of the necessary parameters in its fields,
- the already initialized 3D array `map` will contain the places of local extrema: if an element of it has value 1, it means that the corresponding point of the `layers` of the actually processed octave is a local minimum/maximum,
- write 3 nested `for`-loops to visit all inner elements of `layers` (1st: through 3rd dimension, 2nd: through rows, 3rd: through columns):
  - extract the 3x3x3 local neighborhood (`nbhd`) of the actual element being visited,
  - reorganize `nbhd` to a column-vector (operator `:`),
  - search for the *index* of the minimum and maximum value, if one of those indices is equal to 14, then the central pixel is an extrema (please consider: the properly reshaped 3x3x3 array will have its central element at the middle of the column-vector) ⇒ update `map` if necessary.

NB - along **3rd dim.**: `layers` contains `NumOfLayers`+2 slices while `map` should be indexed from 1 till `NumOfLayers`.

Test 2



Run the script `SIFT_script` again:
if the implementation is correct, you should got the struct `params` and the array `SIFT` with size Px128 in the workspace.
(200 < P < 800)

Let's examine the different fields of params: `extrema`, `filtered` and `eliminated`:

```
figure;
imshow(params.InputImage);
hold on;
spy(params.extrema{1}(:, :, 3), 'g');      % type A
spy(params.filtered{1}(:, :, 3), 'co');    % type B
spy(params.eliminated{1}(:, :, 3), 'yx'); % type C
```

Things to observe:

- type A: extrema even on the blue sky,
- type B: low-contrast extrema already filtered out, but edges are still problematic due to aperture problem,
- type C: only complex/2D structures, which can be located accurately.

THE END