Image and Video Technology

Dr.-Ing. Colas Schretter, `cschrett@etrovub.be`
Pleinlaan 9, 2nd floor (PL9-2.25)

Exercises (WPO) 2017-2018

Content

1. (Week 07) Get started, create an image and experiment with noise distributions
2. (Week 08) Blur, noise (and interpolation) image artifacts
3. (Week 09) Lossless image compression and run-length encoding (RLE)
4. (Week 11) Variable-length codes (VLC) and entropy coding
5. (Week 12) Lossy JPEG image approximation with the discrete cosine transform (DCT)
6. (Week 13) Intra-frame and inter-frame prediction techniques for video compression

Objectives

- Write **by yourself** portable C++ code using the standard library
  – *"A program that has not been tested does not work", Bjarne Stroustrup*
- Understand **digital image** and video processing basics
  – *"Programming is understanding", Kristen Nygaard*
- Acquire **new skills** through practice instead of new knowledge
  – *"I know how to do it, but I can't do it", Fritz Oser*
- Focus on **accuracy** and usability instead of efficiency
  – *"Premature optimization is the root of all evil", Donald Knuth*

Deliverables

An archive with your source code in plain C++ and your lab workbook in PDF format or a web blog including all your results images, qualitative observations and quantitative figures.

– Submission deadline: Sunday **7th January 2018** before 00:00 UTC
– All files must be prefixed by "`ivt2017_name_surname_`" (plug in your own name / surname)
– Compile without warning with `g++ -Wall -Wextra -pedantic`, provide a Makefile if necessary
– Reference images will be given on Pointcarre for checking if programs satisfy requirements

Projects

Design an **original** encoder/decoder executable tool for **lossless bilevel image compression**.

Challenge by groups of two students. For less implementation clutter, only power-of-two square images must be supported (i.e. 64×64 pixels, 256×256 pixels, ...). For eager competitive spirit, a **ranking of the compression rates** on test set images will be shared on Pointcarre.

Evaluations

An external assessor will read your report and review your source code. He will ask questions and request modifications as programming task. Criteria are: acquisition of skills, respect of specifications, clever/beautiful design of your solutions, coding style, quality of the documentation.

1 Get started

1. Write, compile without warning and execute a "Hello World" C++ program
2. Install the ImageJ viewer and import the 256×256 pixels 32 bits .raw image of "Lena"
3. Explore qualitatively in ImageJ by setting the "Window/Level" in the "Adjustment" menu
4. Measure quantitatively in ImageJ with "Measure" and "Histogram" in the "Analyze" menu

2 Create and store RAW 32bpp grayscale images

1. Generate a 256×256 pixels image I containing normalized grayscale values $I(x, y) \in [0, 1]$ depending on pixel coordinates $(x, y) \in [0..255] \times [0..255]$ with the following formula:

$$I(x,y) = \frac{1}{2} + \frac{1}{2}\cos\left(x\frac{\pi}{32}\right)\cos\left(y\frac{\pi}{64}\right)$$

2. Write a `store` function for saving your result as a .raw file of 32 bits values in raster order
   – What is the size in bytes of your file?
3. Explore qualitatively your image by adjusting the "Window/Level"
   – Which value of level and window width encloses the range of pixel values?
4. Measure quantitatively your image with the "Measure" and "Histogram" tools

3 Generate uniform and Gaussian-distributed random images

1. Write a `psnr` function that compute the PSNR given a reference image and a "MAX" value
2. Generate a 256×256 pixels constant reference image
   – What is the expected mean of a uniform random realization in $[0, 1)$?
3. Generate a 256×256 pixels image with uniform-distributed random pixel values in $[0, 1)$
   – What is the expected PSNR (MAX = 1) of the random image, compared to the reference?
4. Generate a 256×256 pixels image with Gaussian-distributed random numbers
   – Set the mean to the expected value and experiment various variances
   – Which standard deviation matches the expected PSNR of the uniform random image?
5. Display in ImageJ the two uniform and Gaussian-distributed noise images, side-by-side
   – Use a level of $\frac{1}{2}$ and a window width of 1, compare the histograms of the two noise images
6. Measure statistics of the noise realizations with the "Measure" tool in the "Analyze" menu
   – Compare the mean, variance, minimum and maximum values for the two cases

4 Add Gaussian noise

1. Write a `load` function for reading RAW 32bpp image files
2. Load in memory the 256×256 pixels picture of "Lena"
3. Write a `normalize8bpp` function for mapping 8bpp grayscale values to the real range $[0, 1]$
4. Add Gaussian random noise with standard deviation $\sigma = 0.024$
5. Compare the noisy image with the original, in terms of PSNR

5 Blur and sharpen with 3×3 kernel convolution

1. What are the values of a normalized 3×3 blurring kernel, using the Normal distribution?
2. Write a `blur` function, applying a 3×3 kernel convolution in a inscribed rectangular region
   – Hint: You should not care for a solution handling pixels at the borders

3. Update your implementation for applying the 3×3 kernel convolution on the whole image
   – What is the PSNR of the blurred image compared to the original?
   – Which result looks best, at equal PSNR, to your naked eyes: the blurry or noisy image?
4. Use the blurred image and very simple arithmetic operations for sharpening the input image
   – Hint: check out "unsharp masking"
   – What is the PSNR of the sharpened image compared to the original?

6 Image capture artifacts

1. Load a 256×256 pixels picture and add Gaussian noise to the original image, play with the noise variance to get roughly equal PSNR than blurring with 3×3 kernel convolution
   – What is the standard deviation of the additive Gaussian noise you retained?
2. After adding Gaussian noise, then apply the 3×3 blurring kernel
3. Generate a second image where you apply first blur then add Gaussian noise
   – Which sequence models well artifacts of a SLR camera: "noise-blur" or "blur-noise"?
4. Compare visually your two results
   – Why there is such a difference in quality?
5. Compare the PSNR of the two results against the original picture

7 Downscaling and upscaling

1. Write a `downscale` function for reducing the size by a factor 2
2. Write a `upscale` function for back by a factor 2
   – Which interpolation method did your implement?
3. Compare in terms of PSNR the upscaled version with the original picture
4. Implement a better image interpolation method for the upscaling operation
   – What is the improvement in terms of PSNR?
5. After applying successively a downscale then upscale operation
   – What happened to the upscaled image?
6. Compute the difference image between the upscaled version and the original picture
   – Where are located the differences?
7. Apply again successively a downscale then upscale operations
   – Do you confirm your initial observations?
8. Compare the result in terms of PSNR using either the original and blurred images as reference