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) Lossy JPEG image approximation with the discrete cosine transform (DCT)
4. (Week 11) Lossless image compression with run-length encoding (RLE)
5. (Week 12) Variable-length codes (VLC) and entropy coding
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

Project

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.

Evaluation

An external assessor will read your report and review your source code. He will ask questions and request modifications as programming task. Criteria are: respect of exercise's specifications, design of solutions, testing procedures, skills in C++ programming, quality of the documentation.

Session 1

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 32bpp .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 should be (and is) the size in bytes of your file?
3. Explore qualitatively your image by adjusting the "Window/Level"
   – Which level and window width enclose the full range of pixel values?
4. Measure quantitatively your image with the "Measure" and "Histogram" tools

3 Generate uniform and Gaussian-distributed random images

1. Generate a 256×256 pixels image with uniform-distributed random numbers in $[0, 1)$
   – What is the expected MSE of the random image, compared to the reference?
2. Generate a 256×256 pixels image with Gaussian-distributed random numbers
   – Set the mean to $\frac{1}{2}$ and experiment with various noise variances
   – Which standard deviation value matches the expected MSE of the uniform random image?
3. Display 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
4. 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

Session 2

4 Additive 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 in $[0, 1]$
   – Apply this operator to convert pixel values from integers in $[0..255]$ to real values in $[0, 1]$
4. Write a `noise` function that adds zero-mean Gaussian noise of given standard deviation
   – Experiment with various noise variances and compare your results side-by-side
5. Write a `psnr` function that computes the PSNR between two images, given a "MAX" value
   – What is the PSNR (MAX=1) of the noisy image compared to the original?

5 Blur and sharpen with 3×3 kernel convolution

1. What are the values of a normalized 3×3 blur kernel, using the Normal distribution?
   – **Hint**: this is very unlikely you will find readily the right numbers on the Internet

2. Write a `blur` function that applies a 3×3 kernel convolution inside a rectangular region
   – **Hint**: you should not care at this moment 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 (MAX=1) of the blurred image compared to the original?
   – Which result looks best to your naked eyes (at equal PSNR): 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
   – Set the noise variance to get roughly equal PSNR than blur with 3×3 kernel convolution
   – Which standard deviation you retained for the additive Gaussian noise?

2. After adding Gaussian noise, then convolve the image with the 3×3 blur 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. Display your "noise-blur" and "blur-noise" result images, side-by-side
   – Why there is such a difference in visual quality?

5. Compare the PSNR of the two results against the original picture
   – Do you agree that *"Blur is a medication for noise"*?

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
   – Which interpolation method did your implement?
   – What is the upscaling quality 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

Session 3

8 Discrete cosine transform (DCT)

1. Create a matrix (dictionary) containing all DCT basis vectors for a 1D signal of length 256
   – Display the dictionary as an image: is it looking right?
   – Rescale your basis vectors to get the modified orthonormal DCT-II variant used in JPEG
   – What is the gain of the DC coefficient?

2. Write a `transform` function that produce DCT coefficients from the input image
   – Use the analysis DCT dictionary as a parameter to the function
   – **Hint**: the 2D transform is separable: split into 1D row-wise, then column-wise dot products

3. Write a `threshold` function that zero-out small (in absolute value) DCT coefficients
   – Threshold small DCT coefficients to zero for discarding information in the frequency domain
4. Write a `transpose` function to produce the inverse synthesis IDCT dictionary
   – Display the analysis and synthesis dictionaries side-by-side, do you spot the difference?
5. Reconstruct an image using again the transform function with the synthesis IDCT dictionary
   – What is the visual effect of hard thresholding on the reconstructed images?
   – Measure the approximation quality in terms of PSNR for various threshold values

9 Lossy JPEG image approximation

1. Create a 8×8 pixels image containing standard JPEG quantization weights Q at 50% quality
   – Visualize the pattern Q, zoom in, change the "Window/Level" setting, ...
   – What is the meaning of coefficients? Why the 8×8 pattern is not symmetric?
2. Write an `approximate` function that apply for each 8×8 pixels blocks: DCT, Q, IQ, IDCT
3. Write a `quantize8bpp` function for exporting normalized images to 8bpp grayscale values
   – **Hint**: after denormalization, quantize and clip integer values in the valid range [0..255]
4. Create a difference image of your result with a baseline JPEG file at 50% quality
   – **Hint**: set the quality of JPEG in the "Edit > Options > Input/Output..." menu in ImageJ
   – Is the difference image zero everywhere as expected?
5. Compare **qualitatively** (eyeballing candidly) the difference image and explain what you see
   – Are you surprised by what you see?
6. Compare **quantitatively** (measuring precisely) the difference image and interpret numbers
   – Do you understand more from your analyzes?

10 Packing DCT coefficients

1. Experiment various ways to layout the DCT coefficients (for an input 256×256 pixels image):
   1. 256×256 pixels layout with a 32×32 grid of 8×8 **contiguous** DCT coefficients
   2. 256×256 pixels layout with a 8×8 grid of 32×32 **interleaved** DCT coefficients
   – Compare visually side-by-side the two options 1. and 2.
   3. 64×1024 pixels layout with 1024 rows of 64 DCT coefficients per block in **raster** order
   4. 64×1024 pixels layout with 1024 rows of 64 DCT coefficients per block in **zigzag** order
   – Compare visually side-by-side the two options 3. and 4.