SCC0251/5830— Prof. Moacir A. Ponti

Teaching Assistant: Leo Sampaio Ferraz Ribeiro

Assignment 1 : image generation

Code the assignment by yourself. Ask if you need help. Plagiarism is not tolerated.

1 Image generator

1.1 Goal

To familiarize the students with the tools they will be using for the remainder of the course and drive home the concept of image sampling.

1.2 Task

In this assignment you have to implement both an **Image Generator** using mathematical functions and an **Image Sampler**. Read the instructions for each step. Use `python` with the `numpy` library.

Your program will generate a synthetic image and then sample from it to create a "digitized" version. Follow the steps:

1. **Generate Synthetic Image**, f, according to the selected function F and parameters C and Q,
2. **Sample from** f to create g, with sampling and quantisation parameters N and B,
3. **Compare** g against reference image r, using root squared error (RSE),
4. **Print** the computed RSE between g and r.

1.3 Parameters

The following parameters will be input to your program in the following order through `stdin`, as usual for run.codes:

1. filename for the reference image r
2. lateral size of the synthesized image C (the image is assumed to be square so that its size is $C \times C$)
3. the function F to be used (1, 2, 3, 4 or 5)
4. parameter Q, used for image generation
5. lateral size of the sampled image N, where $N \leq C$
6. number of bits per pixel B, with $1 \leq B \leq 8$
7. seed S to be used for random functions

Even if a parameter is not going to be used by the selected function, it will be input for convenience.

2 Generating a Synthetic Image

Five image generation functions need to be implemented and the test cases will select them accordingly with the F parameter:

1. $f(x, y) = (xy + 2y)$;
2. $f(x, y) = |\cos(x/Q) + 2\sin(y/Q)|$;
3. $f(x, y) = |3(x/Q) - \sqrt[3]{y/Q}|$;
4. $f(x, y) = \text{rand}(0, 1, \text{S})$:

   The random function is uniform betwen 0 and 1. Use the seed S to initialize the `random` package in python once, before the first number is sampled. Use `random.random()` for this function, and `random.seed(S)` to set the initial seed. Build the image in *raster order*, i.e. line by line.

5. $f(x, y) = \text{randomwalk(S)}$, The random walk function should behave as implied by it's name:
   a) Start from a zero-filled image (see `numpy.zeros`),
   b) Set the value of the first position ($x = 0, y = 0$) to 1, i.e. $f(0, 0) = 1$
   c) Each step consists of randomized single steps on both x and y directions. Use `random.randint()` to get two random integers in the $[-1, 1]$ range, one for each direction, called dx and dy,

d) Use dx and dy to compute next positions $x = [(x + dx) \bmod C]$, $y = [(y + dy) \bmod C]$[1] and then set this position to 1: $f(x, y) = 1$,

e) Keep "walking" randomly for $1 + C^2$ steps

Note that, as with the previous function, you should use the seed S to initialize the `random` package once before the first integer is sampled.

Synthesized images f must be computed using `float` type values. **After f is computed, normalize values so that the minimum is 0 and the maximum is $2^{16} - 1 = 65535$**

3 Sampling and Quantizing the Image

This step consists in simulating the "digitisation" process. We want to take the f synthesized in the previous step and sample an integer matrix g with size $N \times N$, with its pixels using at most B bits (with B between 1 and 8).

3.1 Downsampling

Because g will have lower resolution ($N \leq C$) than f a downsampling operator must be implemented. The simplest operator (and the one you should implement) takes the first element of each C/N-sized sampling region and throws the remaining ones away.

For example, consider a matrix g with $C = 4$.

$$\begin{bmatrix} 5 & 15 & 36 & 0 \\ 18 & 0 & 0 & 1 \\ 0 & 100 & 154 & 0 \\ 0 & 99 & 159 & 100 \end{bmatrix}$$

With $N = 2$, this image has $(C/N)^2 = 4$ regions:

$$\begin{bmatrix} 5 & 15 & 36 & 0 \\ 18 & 0 & 0 & 1 \\ 0 & 100 & 154 & 0 \\ 0 & 99 & 159 & 100 \end{bmatrix}$$

This downsampling operator takes only first pixel each region, yielding image:

$$\begin{bmatrix} 5 & 36 \\ 0 & 154 \end{bmatrix}$$

Note that we start from $g(0, 0) = f(0, 0)$; then $g(0, 1)$ is computed by skipping $\lfloor C/N \rfloor$ pixels in the y direction, while $g(1, 0)$ is obtained by skipping $\lfloor C/N \rfloor$ pixels in the x direction, and so on.

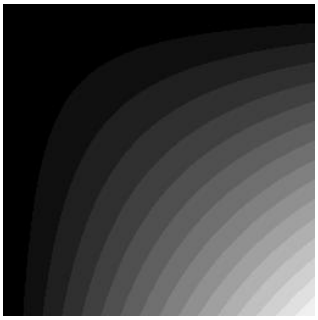
[1] Note that we use mod to avoid x and y referencing a position outside the image boundary

3.2 Quantizing

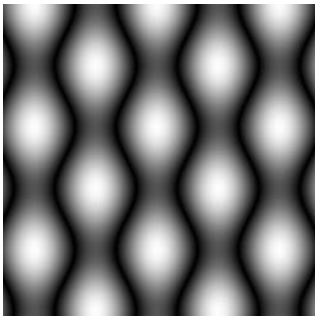
Finally, the image must be quantized to use only B bits per pixel. This can be easily achieved by:

1. Normalization of the values to the range $[0, 255]$ to avoid overflow in the next step,
2. Conversion from float to 8-bit unsigned integers (see numpy's `.astype` and `numpy.uint8`),
3. Bitwise shift to leave only the B most significant bits active while zero'ing the others (see the `«` and `»` operators in `python`). To achieve the quantizing effect, a right shift followed by a left shift will zero the least significant bits and leave the most significant ones intact.

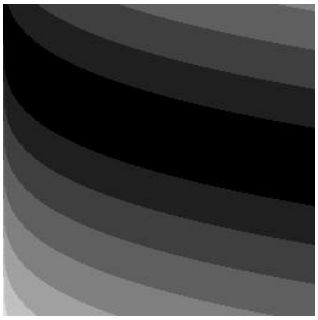
Examples of figures generated by the 5 different functions can be seen below:



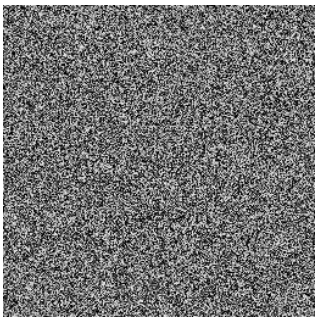
1 ($B = 4$)



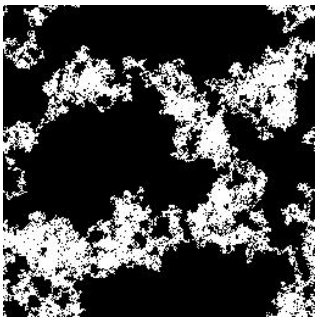
2 ($Q = 32$, $B = 6$)



3 ($Q = 1001$, $B = 3$)



4 ($S = 13$, $B = 3$)



5 ($S = 6666$, $B = 8$)

4 Comparing against reference

Your program must compare the generated image against a reference image r. This comparison must use the root squared error (RSE). Print this error in the screen, rounding to 4 decimal places.

$$RSE = \sqrt{\sum_i \sum_j (g(i,j) - R(i,j))^2}$$

Note this formula does not divide the error by the number of pixels. It is a modification of the Root Mean Squared Error, showing the sum of the errors in all pixels.

The reference image is stored in form of `numpy` matrix. You can load it using `np.load` as below:

```
import numpy as np

filename = input().rstrip()
R = np.load(filename)
```

In this code, `input()` reads from `stdin` and `rstrip` removes trailing characters from a string, which is very useful since `input()` appends a `\n` at the end of each line.

5 Input/output examples

Input example: reference image in the file `ex1.npy`, $C = 1024$, function $F = 1$, plus parameters: $Q = 2$, $N = 720$, $B = 6$, $S = 1$

```
ex1.npy
1024
1
2
720
6
1
```

Output example: only the RSE value with 4 decimal places

Example 1 (high RMSE, indicating the generate image is too different from the reference):

```
7468.7864
```

Example 2 (lower RMSE, indicating a similar image and a correct result):

```
4.1000
```

6 Submission

Submit your source code to run.codes (only the `.py` file).

1. **Comment your code**. Use a header with name, USP number, course code, year/semestre and the title of the assignment. A penalty on the evaluation will be applied if your code is missing the header and comments.

2. **Organize your code in programming functions**. Use one function per type of image to be generated (1,2,3,4,5).