scc0251/5830— Prof. Moacir Ponti

Teaching Assistant: Aline Becher

Assignment 1 : image generation

Try to code the assignment by yourself. Plagiarism is not tolerated.

Image generator

In this assignment you have to implement an image generator using mathematical functions. Read the instructions for each step. Use `python` with the `numpy` library.

Your program must allow the user to provide parameters in order to generate images by the following steps

1. **Parameter input**:
   a) filename for the reference image r
   b) lateral size of the scene C (the scene is assumed to be square so that its size is $C \times C$),
   c) the function to be used f (1, 2, 3, 4 or 5),
   d) parameter Q;
   e) lateral size of the digital image N (also forming a square so that the size is $N \times N$), and $N \leq C$;
   f) number of bits per pixel B, with $1 \leq B \leq 8$;
   g) seed S to be used for the random function.
2. **Generate scene image**, f, according to the selected function and parameters,
3. **Generate digital image**, g, with sampling and quantisation defined by N and B,
4. **Compare** g, with the reference image r,
5. **Print** in the screen the root mean squared error between g and r.

Scene image, digital image

Scene image : functions to generate images

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, using seed S initialised once before the first number is sampled. Use `random.random()` for this function.

5. $f(x, y) = \text{randomwalk}(\text{S})$,

   Seed S is initialised once before the first number is sampled. Then, consider $f(x, y) = 0$ for all x, y. The random walk starts by setting the value 1 to the position $(x = 0, y = 0)$, i.e. $f(0, 0) = 1$. Then, random steps are computed considering at the same time x and y, generating a random number dx between -1 and 1 and a random number dy also between -1 and 1. Use `random.randint()` in this case. The program then sets $x = [(x + dx) \bmod C]$, $y = [(y + dy) \bmod C]$ and finally $f(x, y) = 1$. The module operator is important to avoid error of beyond matrix limits.

   The total number of steps (a step is given after each dx and dy sampling) is $1+(C \cdot C)$

Use the package `random`; The scene image f must be computed using `float` type values. **After f is computed, normalize values so that the minimum is 0 and the maximum is** $2^16 - 1 = 65535$

Sampling and quantisation steps : in this part, we simulate "digitising" the image, generating a integer matrix g with size $N \times N$ and storing pixels with a maximum value of B bits (B between 1 and 8). Because g may have lower resolution than f a downsampling pooling operator must be employed. For example, consider a matrix g with $C = 4$.

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

This downsampling operator takes the first pixel in a given region and skip the remaining ones. For an image f with $N = 2$ we would have:

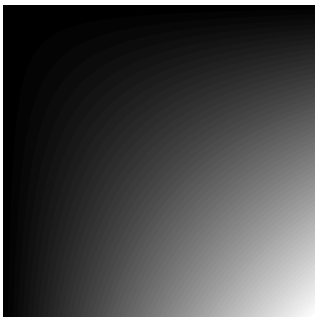
$$\begin{bmatrix} 5 & 36 \\ 0 & 154 \end{bmatrix}$$

The step can be defined as the integer ratio between C and N, i.e. $\lfloor C/N \rfloor$.

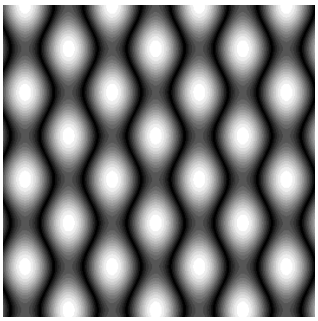
Note that $g(0,0) = f(0,0)$ and then $g(0,1)$ is obtained by skipping a number of pixels relative to the ratio of reduction between f and g.

In addition, f may contain values higher than 2^8. Thus, a quantisation is needed, using a bitwise shift. In order to perform that, firs convert values of f into a 8-bit unsigned integer, so that the maximum value is $(2^8) - 1 = 255$. Then, perform a bit-shift so that only the B most significant bits remain, and the other one are only zeros.

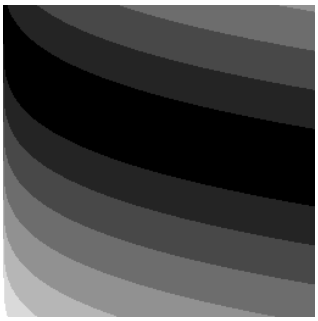
Examples of figures generated by the 5 different functions can be seen below:



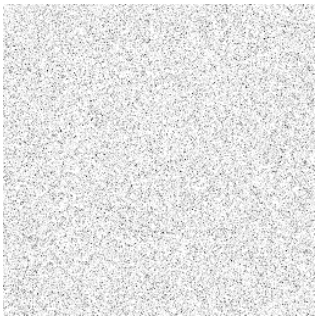
1



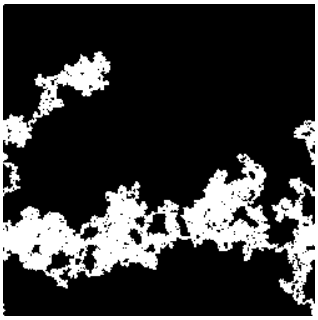
2 ($Q = 32$, $B = 6$)



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



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



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

Comparing with reference

Your program must compare the generated image with 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 should load and convert to the `uint8` to assure the comparison is valid, as below:

```
import numpy as np

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

Input/output examples

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

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

Note function 1 does not use parameters Q and S, still all must be read via keyboard.

Output example: only the RMSE value in format `float`

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

```
7468.7864
```

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

```
4.1000
```

Submission

Submit your source code using the 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 grading 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).