Blurring and Deblurring

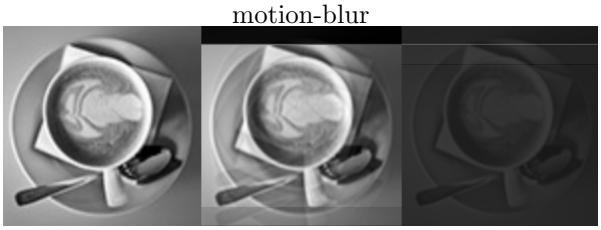
Benjamin Li

April 2020

1 Results

The execution-time for motion-blur and focus-blur were roughly 1 second for a 100x100 image; while, deblurring took 60 minutes.

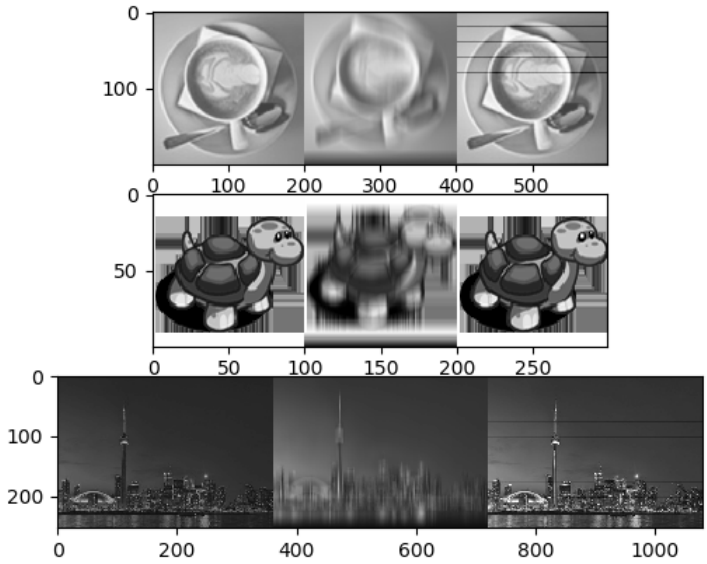
motion-blur



out-of-focus blur

1.1 Accelerated Motion-blur Algorithm

Three images (100x100, 200x200, 360x254) were blurred/deblurred in **two secs**. See **blur_fast.py**



2 Design

An n x m image A is multiplied by an $(n \cdot m)$ x $(n \cdot m)$ mask X to create a blurred-image. Images were converted from RGB to greyscale because processing-time is proportional to the number of channels.

$$X \cdot A = Y$$
$$X^{-1} \cdot Y = X$$

2.1 Blurring

The blurring-masks follow a simple pattern that allow us to create each row in $O(1)$ time-complexity. Furthermore, deblurring-masks X^{-1} can be easily derived to reverse the process.

The **motion-blur algorithm** increases the pixels used in blurring proportionally with input resolution to keep output consistent. Specifically, a pixel in the blurred image, $Y_{i,j}$ maps to the average of $A_{i,j}$ to $A_{i,j+N}$, where N is 10% of A's height. **Out-of-focus blur** used the same vector [1,0,0,1,4,1,0,0,1] regardless of input resolution to maintain simplicity.

Let i = number of pixels in X and X^{-1}:

- **Operations:** list flattening + $\sqrt{i}$ row operations $= i + \sqrt{i} \cdot N$
- **Time:** $O(i)$
- **Space:** $O(i)$

With the above, blurring a 100x100 image involves 11,000 operations.

2.2 Deblurring

The deblurring mask X^{-1} is computed with Doolittle's LU Decomposition algorithm. Let i = number of pixels in X and X^{-1}:

- L computation: $i \cdot (1 + 2 + ...i - 1) = i\frac{i(i+1)}{2}$ operations $= O(i^3)$
- U computation: $i \cdot (1 + 2 + ...i - 1) = i\frac{i(i+1)}{2}$ operations $= O(i^3)$
- Y computation: i vector dot-products $= i \cdot n$ operations $= O(i^{1.5})$
- X^{-1} computation: i vector dot-products $= i \cdot n$ operations $= O(i^{1.5})$
- **Total operations:** $i^3 + i^2 + 2i^{1.5} = O(i^3)$

With the above, deblurring a 100x100 image involves 1.00e+12 operations.

3 Complexity Analysis

The space-time complexity for an n x m image is $O((n \cdot m)^3)$ and $O((n \cdot m)^2)$.

4 Observations

4.0.1 Impact of Erroneous Blurring Matrix

In the final y rows of blurring kernel X, mistakes may lead to a **singular matrix**—meaning it is impossible to derive a conventional inverse-matrix X^{-1} for deblurring.

The value of y is arbitrary, but the below example will illustrate an example, where $y = 3$ for a motion-blur kernel. Note that the *incorrect* kernel has no 1s in the last 2 rows; thus, it is **singular**.

$$\text{Correct} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{Wrong} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Python: ‘LinAlgError: Singular matrix‘