Bilateral Filter Report

wfjv99

The bilateral filter is an image processing method which smooths out areas of an image which are similar in pixel intensity and geometrically near each other, thus preserving edges.[1] For this reason, it is often used in lieu of the Gaussian filter, which blurs areas of an image that are geometrically near each other but does not take into account pixel intensity similarity and does not preserve edges[4]. This is an implementation of the bilateral filter in Python 2 for .png images.

Part 1

Implementation

For my implementation, I targeted Python 2.7.5 and used the external libraries pypng[2], numpy[3] and functools32[5]. I used pypng for reading and writing .png images, and the numpy library for representing the images as matrices and then manipulating those matrices. The functools32 library provided a function *lru_cache* for memoising my padded matrices (I.e. when I had to pad the edges of a image matrix so that I could calculate responses at the boundary of the matrix) so that the bilateral filtering process was greatly speeded up. In my program, I treat greyscale images as RGB images where all three channels are equal, so I implement only one version of the bilateral filter.

A bilateral filter can be applied to an image using the instance method *Image.apply_bilateral_filter(size, distance_sigma, intensity_sigma).* The order (size) of the filter is specified as the 1st argument, and must be an odd number n, the resulting filter then being n x n in size. The standard deviations of the distance and intensity difference Gaussian functions can also be specified as the 2nd and 3rd arguments, respectively.

Input/Output Functionality

My implementation of the bilateral filter has been tested on and at least works with two types of .png files: 8-bit grayscale, no alpha channel and 24-bit RGB, no alpha channel. A png file is loaded as an instance of the Image class using the static *Image.read_from(path_to_png_file)* method, and after an instance has been manipulated (i.e. had a bilateral filter applied to it using *Image.apply_bilateral_filter()*), it is written back to the filesystem using the instance method *Image.write_to(path_to_write_out_png_to)*. For simplicity, all images (both greyscale and RGB) are written out as 24-bit RGB .png files with no alpha channel.

Construction of Gaussian functions and the bilateral filters

A Gaussian function G(x) (equivalent to as in Illustration 1) is returned by the function *gaussian_function(sigma)* which is defined in the body of the instance method *Image.apply_bilateral_filter().*

$$\frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{x^2}{2\sigma^2}}$$

Computed by Wolfram|Alpha

Illustration 1: G(x) for some defined σ (image generated with Wolfram Alpha)

The bilateral filter can be thought of as applying the combination of two different masks at every pixel p; the distance-difference ('domain' distance)[1] Gaussian filter D, which is the same regardless of what pixel it is being applied to, and the intensity-difference ('range' distance)[1] Gaussian filter I, which depends on the intensity of the pixel it's being applied to and the intensities of the neighbouring pixels. D is constructed once, while a different I is constructed at every single pixel. The applied mask C is the element wise multiplication of D x I, weighted by the sum of it's own components.

Boundary Handling

I arbitrarily decided to use the 'edge padding' method to handle boundaries, and the results turned out acceptable. I did this by using the *numpy.pad()* function, which can automatically pad the edges of a matrix by some specified amount of rows and columns. When applying a b x b bilateral filter to a n x m image, I pre-emptively padded the image's matrix by b/2 rows top and bottom and by b/2 columns either side before calculating any responses.

Part 2

All generated images are in the "out/" directory. The names of each image are formatted as

[letter indicating which image has been processed]_s[size of bilateral filter]_d[distance sigma]_i[intensity sigma].png

I generated them with the *do_all(size)* function.

Parameters

I tested bilateral filters of sizes 3x3 and 5x5, with distance sigmas (σ_d) of 0.1, 1, 10, 50 and 250 and intensity difference sigmas (σ_i) of 0.1, 1, 10, 50, and 250, on two greyscale images (testA.png and testB.png) and an RGB image (testC.png).

Discussion

Low σ_d

I noticed that for σ_d = 0.1, neither of the greyscale images changed much regardless of the values of the other parameters. It was only when the distance sigma was at least 1 that there was a noticeable smoothing in both greyscale images.

Increased blurring with increasing σ_d

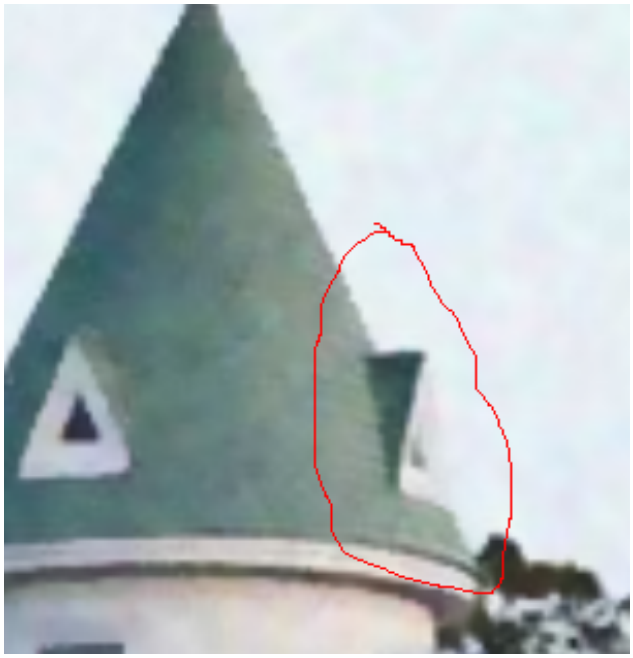
Generally, as σ_d was increased, there was an increased blurring/smoothing effect, and edges were not preserved as well. This may have been because with increasing σ_d, there is increasing blurring of local pixels without regard to similarity of pixel intensity.

Cartoon Effect in testC.png

For testC.png, when σ_i was 0.1, the result of applying the bilateral filter with a σ_d of 10 or greater was a cartoonish version of testC.png, where by cartoonish I mean areas of the same texture of the image became very smoothed out while the edges remained fairly sharp.

Original testC.png *5x5: σ_d=250, σ_i=0.1*

Noticeably the striated appearance of the walls and roof almost completely disappeared. One place where this did not occur in the cartoon image is on the lighter, right hand edge of the cone roof near the window (circled in red); this may have been because the lighter edge of the roof was closer in intensity values to the neighbouring sky, and so was 'saved' from indiscriminate blurring by the distance blurring.



References

[1] - http://users.soe.ucsc.edu/~manduchi/Papers/ICCV98.pdf – by C. Tomasi & R. Panducci

[2] - https://github.com/drj11/pypng – by David Jones and others

[3] - http://www.numpy.org/ - multiple developers

[4] - http://people.csail.mit.edu/sparis/bf_course/slides/03_definition_bf.pdf (pages 2-3)

[5] - https://github.com/MiCHiLU/python-functools32 – by Endoh Takanao

Other resources used:

- the lecture slides

- Python 2.7.5 documentation (https://docs.python.org/release/2.7.5/)

- inline documentation of the pypng library (e.g. documentation strings for the png.Writer class as in the file "libraries/png2.py")