Yael Cohen
Naama Levy
Guidance: Prof. Danny Barash

Introduction

Abstract

The topic we chose for our mini-project is bilateral filtering. Bilateral filter is a non-linear filter which aims to blur and denoise an image while preserving its strong edges. Our work is based on *Tomasi and Manduchi* paper from 1998[1].

Their work introduces the simplicity and flexibility of the method, which can be implemented in applications from various domains, such as denoising of medical imaging and movie restoration, image styling adjusting for a variety of display capacities of different devices, image contrast management such as texture separation, tone mapping and management and detail enhancement [2].

By using this filter, each pixel value is being replace by a weighted average of its neighbors, when the weights have inverse proportion to their distance from the central pixel.

The weight for each pixel depends on the Euclidean distance of pixels and the range (intensity) differences. The first intends to smooth the image, while the latter was added for edge preservation.

The bilateral filter, denoted by BF convolution is defined by:

$$BF[I]_p = \frac{1}{W_p}\sum_{q\in S} G_{\sigma_s}(\|p-q\|)\, G_{\sigma_r}(|I_p - I_q|) I_q$$

Where:

$\boldsymbol{I}$ is the original image to be filtered, $\boldsymbol{p}$ is a pixel to be reevaluated, $\boldsymbol{S}$ is the pixel neighborhood.

$\boldsymbol{G_{\sigma_s}}$ is the spatial (or domain) kernel for smoothing differences in coordinates, and $\boldsymbol{G_{\sigma_r}}$ is the range kernel for smoothing differences in intensities, both can be calculated by Gaussian function, and specify how much will the image be filtered.

The normalization factor: $W_p = \sum_{q\in S} G_{\sigma_s}(\|p-q\|)\, G_{\sigma_r}(|I_p - I_q|)$, ensures the weights will sum to 1

Parameters setting:

- Increasing the σ_r parameter will cause the filter to become more similar to Gaussian convolution (blur), as it makes the Gaussian $\boldsymbol{G_{\sigma_r}}$ nearly constant over the intensity interval of the image.
- Increasing the σ_s parameter will make larger features smoother (Kornprobst, Pierre & Tumblin, Jack & Durand, Frédo, 2009).

Methods

The algorithm:[1]

The algorithm will run over all pixels in the image I:

For each pixel p in I:

1) Initialization: $BF[I]_p\ , W_p = 0$
2) For each neighbor q in the window S compute:
   a.
      - For BW image:

$$w_q = e^{-\frac{\|p-q\|^2}{2\sigma_s^2}} \cdot e^{-\frac{|I_p-I_q|^2}{2\sigma_r^2}}$$

      - For RGB image:

        For each channel c:

$$w_{q_c} = e^{-\frac{\|p-q\|^2}{2\sigma_s^2}} \cdot e^{-\frac{|c_p-c_q|^2}{2\sigma_r^2}}$$

   b. $BF[I]_p += w_q \cdot I_q$
   c. $W_p += w_q$
3) $BF[I]_p = \frac{I_p}{W_p}$

Efficient algorithm suggestion

The brute-force algorithm we described above takes $O(r^2)$ calculations per pixel, where r is the radius chosen for the neighbors' box. We would like to describe the box kernel method, introduced by Ben Weiss in 2006 [3]. This algorithm is based on the Huang's algorithm (see figure 1[i]) from 1981 which decreased the computational complexity to $O(r)$. Huang uses the sequential overlap of adjacent windows to reduce the computational complexity (Weiss, 2006).

Weiss's approach suggests an algorithm which takes $O(r)$ space and $O(\log(r))$ runtime per pixel when the pixel value is represented by 8-bit data.

The main concept enables this fast algorithm is the observation that if multiple columns are processed at once, the redundant calculations become sequential. Is this algorithm, Huang's method is adapted to process N columns at once using N histograms, each per output column. Due to the distributive property of histograms, we do not need to maintain each histogram. For example, if an image window W is a union of two disjoint regions A and B, then the H_w histogram = $H_A + H_B$.

This solution[ii] forms a set H^* of partial histograms $P_0 \ldots P_{n-1}$, such that each histogram $H_0 \ldots H_{N-1}$ is representable as the sum of T partial histograms from H^*. Each input pixel is added/subtracted to each histogram intersecting its column.

When applying this approach on bilateral filtering, it relies on the box spatial kernel method. In this method, every neighbor has the same weight, which means we consider G_{σ_s} as constant. Under this condition, the histogram of each spatial window becomes sufficient to perform the filtering operation. This $O(log\ r)$ median-filtering algorithm already generates these histograms, so the bilateral convolution can be appended in constant time per pixel, scaling with the support of the intensity function G_{σ_r}. Weiss's paper

[1]Our mini-project implementation on GitHub: https://github.com/yaelco94/bilateralFilter

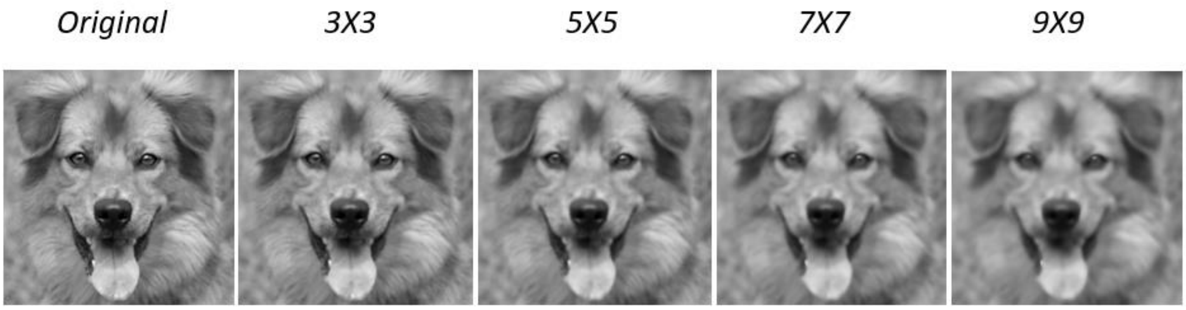
demonstrates this improvement comparing to Photoshop's Surface Blur filter[iii], which makes the bilateral filter 20 times faster than the other. This achieved thanks to the reduction of time spent on calculating each window's histogram by the intensity function.

Results and Discussion:

In our research we wanted to explore the effect of 3 different parameters on the image:

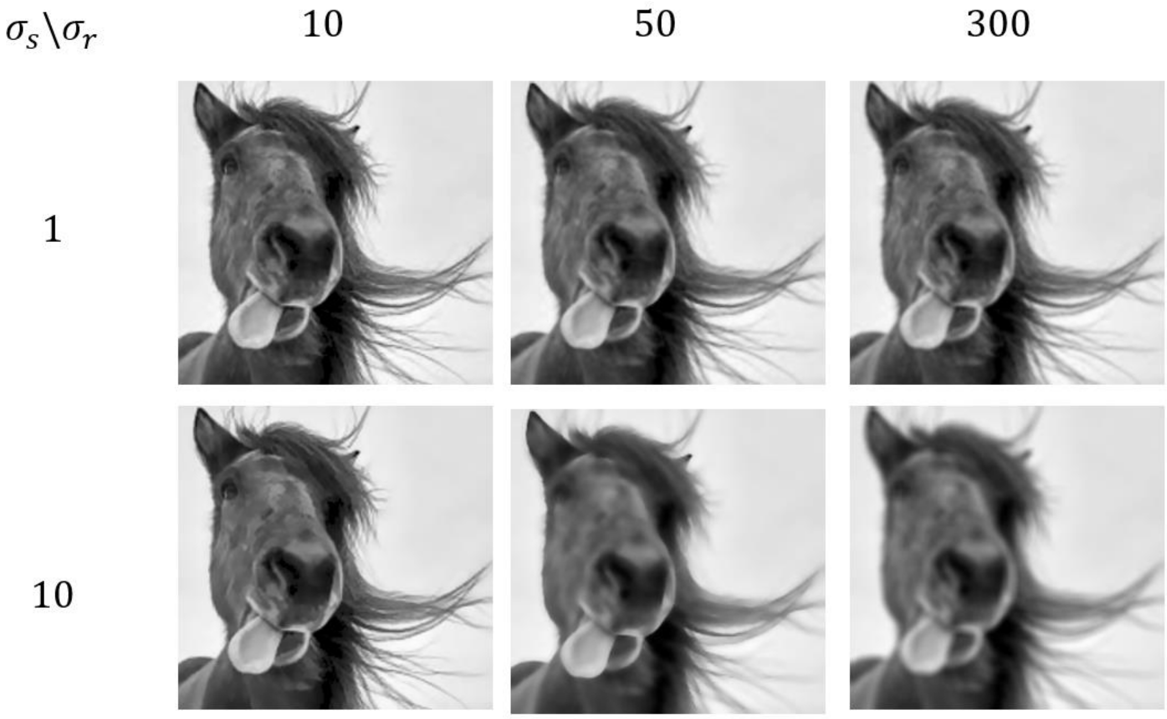
window size, spatial sigma and tonal sigma.

- Firstly, we observed the effect of the image window. We set the spatial sigma and tonal sigma to constant values ($\sigma_r = 100, \sigma_d = 10$), and changed the window size. Here are the results:



In this experience we notice that as we increase the window size the image is more blur, and the edges are less clear. On the other hand, on a small window size, the changes are almost unseen. Therefore, we found our optimal window size for the rest of experience to be **5x5 or 7x7**.

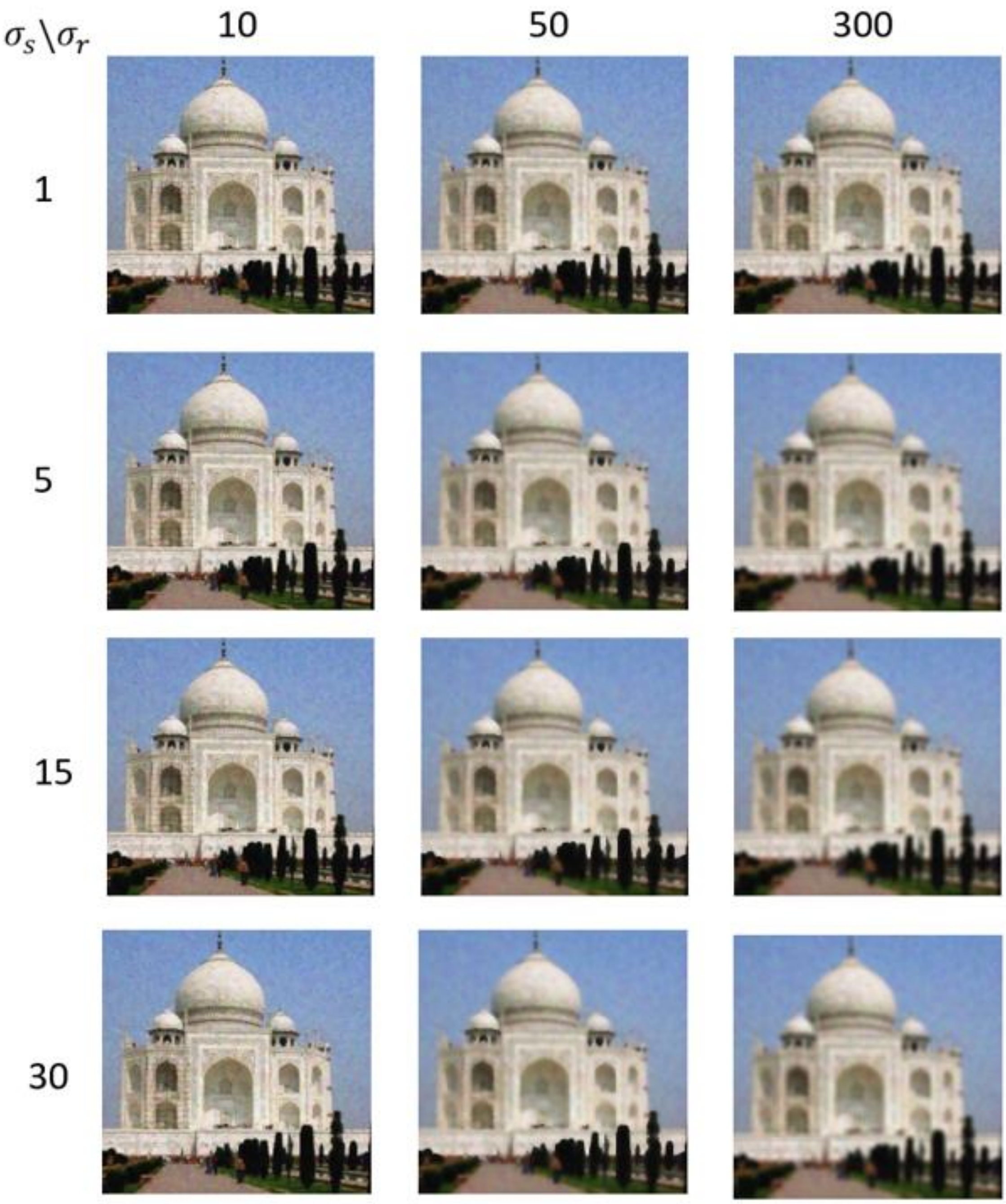
- Our next phase was to examine the effect of the spatial sigma and tonal sigma. We set our window size to be 7x7 and changed the sigma values as you can see in the image below:



We can observe that when σ_s is low, there is almost no effect of σ_r and all the images are remarkably similar to the original photo. That occurs since we almost do not consider further pixels. As σ_s increases the affect of σ_r becomes much more significant.

In addition, we saw that if we set a very high σ_r, the image returns very blur, that happens since we almost ignore the tonal differences, hence the edges are not preserved.

- In our last test we wanted to examine the effect on colourful images. From our previous experiences we decided to use larger spatial sigma and smaller tonal sigma while setting our window size to 5x5.



- In addition, we examined the changes on a single pixel. We observed the values of the pixel in location (100, 100) for each one of the pictures presented. The original pixel was: [89 126 133].

	10	50	300
1	(90.85, 125.46, 133.84)	(97.17, 129.16, 137.63)	(98.56, 129.97, 138.92)
10	(91.01, 125.94, 134.61)	(100.16, 131.6, 139.64)	(104.52, 134.37, 144.08)
15	(91, 125.95, 134.61)	(100.17, 131.61, 139.64)	(104.55, 134.4, 144.11)
30	(91, 125.94, 134.62)	(100.17, 131.61, 139.64)	(104.57, 134.41, 144.12)

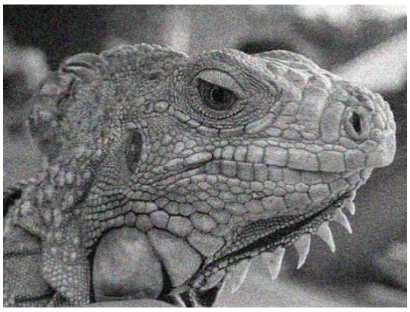
- We can see that as we increase the values of σ_r and σ_s the neighbourhood affect on this pixel RGB vector values increase as well.

Performance comparing

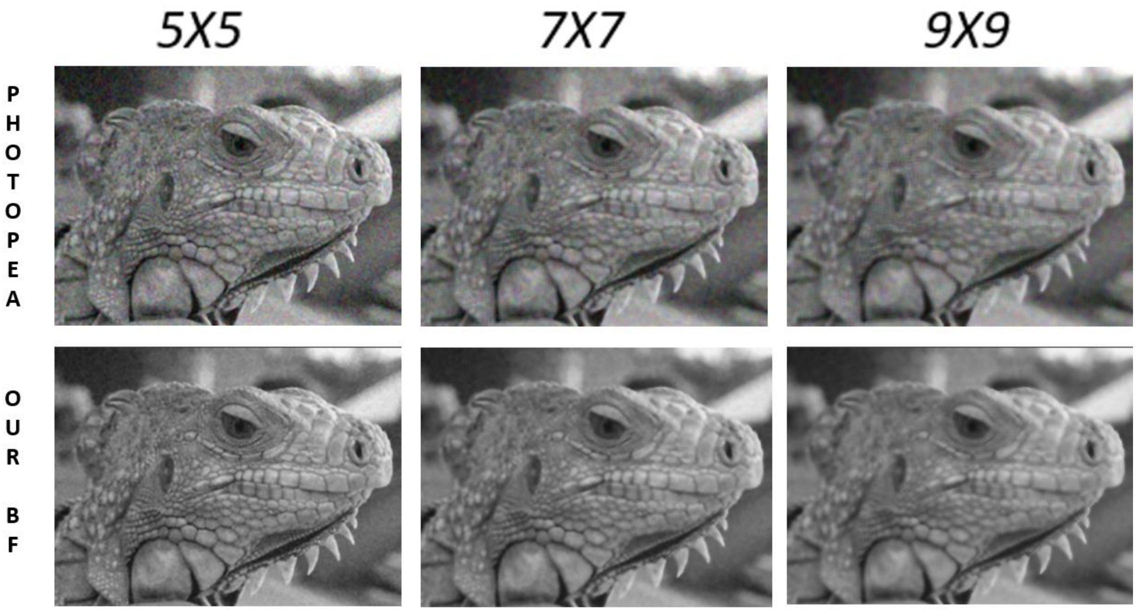
Finally, we compared the results of our implementation of the biliteral filtering, with the results of median denoising filter of “Photopea” application (free photoshop application).

We could not find online the parameters the app uses (besides the option to change radius/window[2] size), so we applied a variety of sigma values until we found the values that supplied the most similar results. The values we used for our implementation are $\sigma_r = 100, \sigma_d = 10$.

Below the original photo:



Here you can see the results of “Photopea” application [5] compared to our implementation of biliteral filter, with a different window size:



[2] denote: the application “Photopea” uses “radius size” parameter instead of “window size”. We did the conversion to window size in order to retain the same terms as the rest of the paper.

Yael Cohen
Naama Levy
Guidance: Prof. Danny Barash

Conclusions

In our project, we practice the application of image filtering using computational method which aimed to denoise is while conserve its edges. We implement our code in python, chose different photos, and set various of parameters in order to examine our filter effect on them.

We saw that for different photos, in different size and resolution, we may set different parameters values for the desired results. For example, in a very noisy image, we may want to increase range constant, so the photo will become more blur. But when considering a photo with many thin details we want to preserve, we may pick smaller values for the constants.

As for the window size, we found 5-7px neighborhood size to be the best for image filtering application for 300X300px square, as the changes did not damage the image edges, and the computational complexity was not too heavy to run on a domestic computer.

For other applications, we may consider changing the filter parameters and use a stronger computational device.

Yael Cohen
Naama Levy
Guidance: Prof. Danny Barash

References

[1] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, 1998, pp. 839-846, doi: 10.1109/ICCV.1998.710815.

[2] Kornprobst, Pierre & Tumblin, Jack & Durand, Frédo. (2009). Bilateral Filtering: Theory and Applications. Foundations and Trends in Computer Graphics and Vision. 4. 1-74. 10.1561/0600000020.

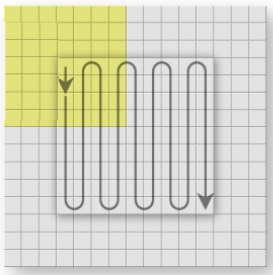
[3] Ben Weiss. 2006. Fast median and bilateral filtering. ACM Trans. Graph. 25, 3 (July 2006), 519–526. DOI:https://doi.org/10.1145/1141911.1141918

[4] Huang, T.S. 1981. *Two-Dimensional Signal Processing II: Transforms and Median Filters.* Berlin: Springer-Verlag, pp. 209-211.

[5] https://www.photopea.com

---

i Figure 1: Huang's algorithm



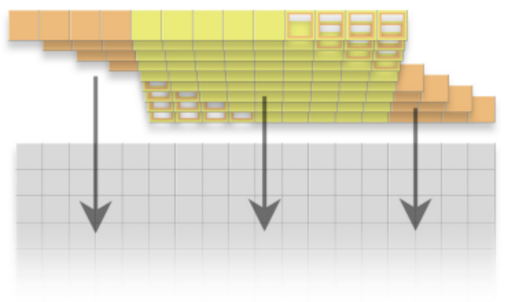
r: radius of median filter. (shown above as $r = 3$.)
H: 256-element histogram.
I: input image, $S + 2r$ pixels square.
O: output image [inset], S pixels square.

```
initialize H to I[0 .. 2r][0 .. 2r].  // yellow region
find median value m in H, write m to O[0][0].
   for row = 1 to S - 1:
      add values I[2r + row][0 .. 2r] to H.
      subtract values I[row - 1][0 .. 2r] from H.
      find median value m in H; write m to O[row][0].
   step sideways to next column (and process bottom to top, etc.).
```

Figure 3: Pseudocode for Huang's $O(r)$ Algorithm

ii Figure2: The $O(\log r)$ Algorithm

Yael Cohen
Naama Levy
Guidance: Prof. Danny Barash



r: radius of median filter.
H^*: Array of partial histograms, processing N columns.
I: input image, $N + 2r$ pixels square.
O: output image, N pixels square.

```
for each row in [0 .. 2r]:      // Initialize H*
    Add row, I[row][0 .. 2r + N - 1] to H*, as per Figure 4
for each output pixel in O[0][col]:  // compute first N median values
    scan H_col (implicit in H*) to the find the median m,
    write O[0][col] = m.
for row = 1 to N - 1: // step from top to bottom of image
    add new bottom row, I[row + 2r][0 .. 2r + N - 1], to H*.
    subtract old top row, I[row - 1][0 .. 2r + N - 1], from H*.
    find N new median values in H*; write to O[row][0 .. N - 1].
```

Figure 6: Pseudocode for $O(\log r)$ Algorithm

iii

Figure3: Comparing bilateral filter with $O(log\ r)$ Algorithm with Photoshop's blur filter.

With a single iteration and a fixed triangular intensity function (support 80 levels), our results numerically match Photoshop's Surface Blur output, with up to twenty-fold acceleration. The performance bottleneck (over 80% of the calculation) is the constant time spent multiplying each window's histogram by the intensity function, which accounts for the flatness of our performance curve. Reducing our implementation to 64 segments should nearly triple its speed, while maintaining very high quality results.

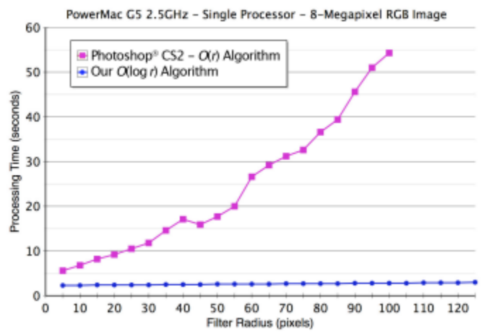


Figure 13: Bilateral Filter Performance