SRFP FINAL REPORT

TITLE: ANALYSIS AND IMPLEMENTATION OF A FAST BILATERAL FILTER

Name: ANMOL POPLI

SRFP Application No: ENGS7022

Email id: anmol.ap020@gmail.com

Guide: Prof. KUNAL NARAYAN CHAUDHURY

Institution: Indian Institute of Science, Bengaluru

REPORT ON ANALYSIS AND IMPLEMENTATION OF A FAST BILATERAL FILTER

Linear Filtering with Gaussian Blur (GB)

The basic operation in linear image filtering is convolution by a positive kernel. Convolution with a Gaussian kernel amounts to estimate at each position a local average of intensities and corresponds to low pass filtering. With input image h(x), the Gaussian blur filtered image is defined by:

$$\hat{h}(x)=\int_{\Omega} g_{\sigma}(y)h(x-y)dy$$

where $g_{\sigma}(t)$ denotes the two-dimensional Gaussian kernel and Ω is its support.

$$g_{\sigma}(t)=\frac{1}{2\pi\sigma^2}\exp\left(\frac{-t^2}{2\sigma^2}\right)$$

The standard deviation sigma defines the extension of the neighborhood, with weights decreasing with the spatial distance to the center position x. As a result, image edges are blurred.

Nonlinear Filtering with Bilateral Filter (BF)

Similar to the Gaussian convolution, the bilateral filter also performs a weighted average of pixels. The difference is that the bilateral filter takes into account the variation of intensities to preserve edges. The rationale of bilateral filtering is that two pixels are close to each other not only if they occupy nearby spatial locations but also if they have some similarity in the photometric range. With input image f(x), the Bilateral Filter output is defined by:

$$\hat{f}(x)=\frac{1}{\eta}\int_{\Omega} g_{\sigma s}(y)g_{\sigma r}(f(x-y)-f(x))f(x-y)dy$$

where the normalization factor:

$$\eta=\int_{\Omega} g_{\sigma s}(y)g_{\sigma r}(f(x-y)-f(x))dy$$

$g_{\sigma s}$ is a spatial Gaussian that decreases the influence of distant pixels, $g_{\sigma r}$ a range Gaussian that decreases the influence of pixels with an intensity value different from f(x).

An advantage of linear filters is that they can be implemented efficiently even for large values of standard deviation using various techniques such as FFT and IIR approximations. Unfortunately, these acceleration techniques do not apply to nonlinear filters such as the bilateral filter. As the neighborhood of bilateral filter increases, the filtering operation becomes computationally intensive, with the complexity $O(r^2)$ per pixel, where r is the width of neighborhood. Also, unlike 2D Gaussian filter, the 2D bilateral filter is not separable into 1D filters.

The aim of my project is to implement and optimize fast bilateral filtering algorithms, wherein the number of operations per point does not scale with the size of the filter, i.e. these algorithms have O(1) complexity.

Shiftable Bilateral Filters

The property of shiftability for a kernel $\phi(s)$ means that for a given N, we can find a fixed set of basis functions $\phi_1(s)$, . . . , $\phi_N(s)$ and coefficients c_1 , . . . , c_N , so that for any translation τ, we can write

$$\phi(s-\tau)=c_1(\tau)\phi_1(s)+...+c_N(\tau)\phi_N(s)$$

The coefficients depend continuously on τ, but the basis functions have no dependence on τ. In this approach, the Gaussian range kernel is approximated by certain kernels that possess this property of shiftability.

Let $\bar{f}(x)$ denote the output of the Gaussian filter $g_{\sigma s}(x)$ with neighborhood Ω,

$$\bar{f}(x)=\int_{\Omega} g_{\sigma s}(y) f(x-y)\,dy$$

By replacing $g_{\sigma r}(s)$ with $\phi(s)$, BF output can be written as:

$$\hat{f}(x)=\frac{1}{\eta}[c_1(f(x))\overline{F_1}(x)+...+c_N(f(x))\overline{F_N}(x)]$$

where $F_i(x)=f(x)\phi_i(f(x))$. Similarly, by setting $G_i(x)=\phi_i(f(x))$,

$$\eta=c_1(f(x))\overline{G_1}(x)+...+c_N(f(x))\overline{G_N}(x)$$

Through this algorithm, the nonlinear filtering problem is converted into a problem of computing a number of linear Gaussian convolutions. Gaussian convolutions can be approximated using certain algorithms which have O(1) complexity.

The research papers I studied in this regard were [4], [5], [6], and [7]. All these make use of the shiftability property of range kernels to disintegrate the bilateral filter into linear Gaussian convolutions. [4] employs Taylor series approximation of the Gaussian kernel, with the Taylor polynomials being shiftable. In [5], raised cosines are used to approximate the Gaussian kernel. The constituent cosines are shiftable. [6] presents a fast algorithm to compute the maximum local dynamic range of the image, T and a method to reduce the number of raised cosine coefficients for low sigmar. In [7], Fourier bases are used to approximate the Gaussian kernel, with the complex exponentials being shiftable.

Fast Gaussian Filter Module

The preliminary part of my project was to study and implement fast algorithms of gaussian convolution, compare them and develop an optimal C implementation of Gaussian filter, which would then serve as an abstract module for the C implementation of the fast bilateral filter. The algorithms of Gaussian filtering I implemented are the following:

- **Deriche Recursive Gaussian Filter**
  In this [1] algorithm, the Gaussian is approximated by a sum of weighted exponentials with complex coefficients and exponents, from which a rational transfer function can be derived. This rational transfer function is then converted to a difference equation which can be implemented recursively. The order of filter depends on the number of exponentials in the sum. I implemented the 3rd order and 4th order recursive filters. In the paper, the expressions of transfer function coefficients for only the 4th order are given. I derived the 3rd order transfer function coefficients from the 3rd order approximation impulse response.

$$n_{22}^{c}=c_0\exp\left(\frac{-2b_0}{\sigma}\right)+\left(a_0\cos\left(\frac{\omega_0}{\sigma}\right)-a_1\sin\left(\frac{\omega_0}{\sigma}\right)\right)\exp\left(\frac{-(b_0+b_1)}{\sigma}\right)$$

$$n_{11}^{c}=-\left(a_0\cos\left(\frac{\omega_0}{\sigma}\right)-a_1\sin\left(\frac{\omega_0}{\sigma}\right)\right)\exp\left(\frac{-b_0}{\sigma}\right)-a_0\exp\left(\frac{-b_1}{\sigma}\right)-2c_0\exp\left(\frac{-b_0}{\sigma}\right)\cos\left(\frac{\omega_0}{\sigma}\right)$$

$$n_{00}^{c}=a_0+c_0$$

$$d_{33}^{c}=-\exp\left(\frac{-(2b_0+b_1)}{\sigma}\right)$$

$$d_{22}^{c}=\exp\left(-2\frac{b_0}{\sigma}\right)+2\exp\left(\frac{-(b_0+b_1)}{\sigma}\right)\cos\left(\frac{\omega_0}{\sigma}\right)$$

$$d_{11}^{c}=-2\exp\left(\frac{-b_0}{\sigma}\right)\cos\left(\frac{\omega_0}{\sigma}\right)-\exp\left(\frac{-b_1}{\sigma}\right)$$

$$d_{11}^{a}=d_{11}^{c},\ d_{22}^{a}=d_{22}^{c},\ d_{33}^{a}=d_{33}^{c},\ n_{33}^{a}=-d_{33}^{c}n_{00}^{c},\ n_{22}^{a}=n_{22}^{c}-d_{22}^{c}n_{00}^{c},\ n_{11}^{a}=n_{11}^{c}-d_{11}^{c}n_{00}^{c}$$

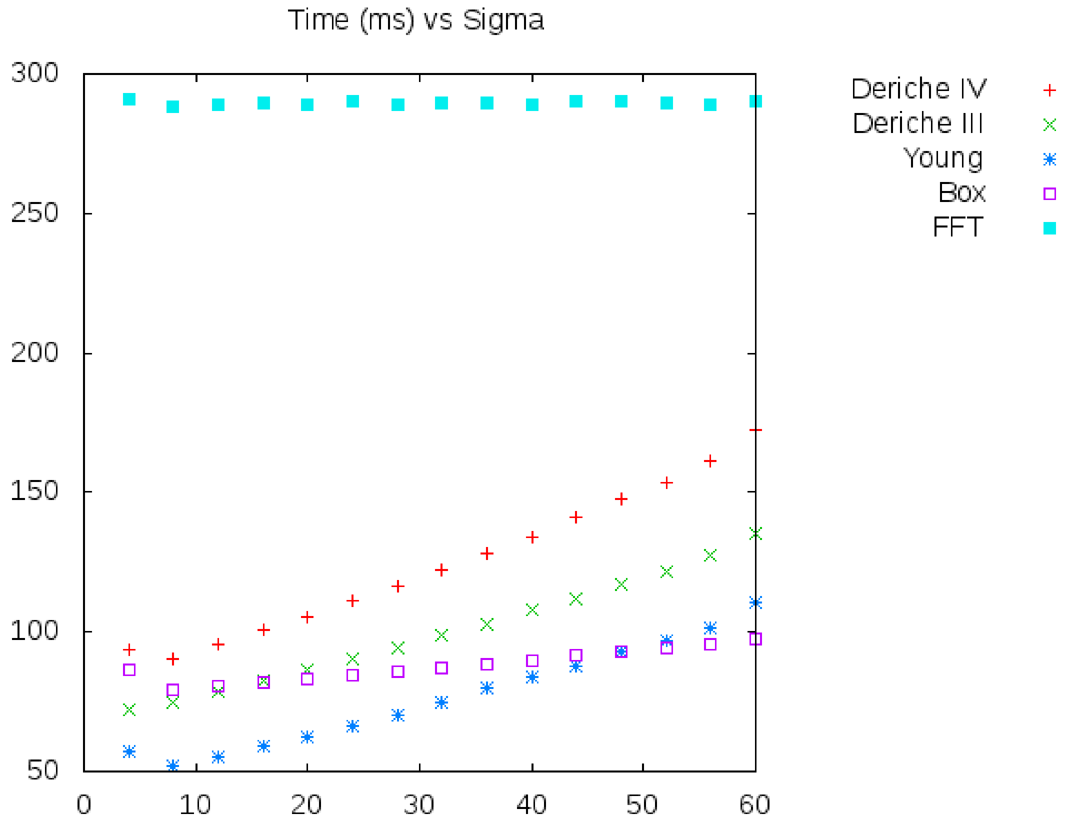
Image Padding: First, I implemented this with replicate padding. There were no extra computations required as the first output element can be computed just by integrating the impulse response, which in turn is the value of transfer function at frequency 0. Next, I opted for symmetric padding, with padding width equal to the filter radius, i.e. 3*sigma. This required the filtering operation to be done for the whole padded image and thus increased number of computations proportionate to the padding width. But since symmetric padding is the one that smooths boundary features on the basis of their actual neighborhood, I proceeded with it. One optimization I introduced in the 3rd order Deriche filter was to compute the first 3 output values of the actual part of the image using normal gaussian convolution, and start the recursion from the 4th pixel. For a causal filter applied on w padded pixels, 6w MADDS are reduced to 3w MADDS.

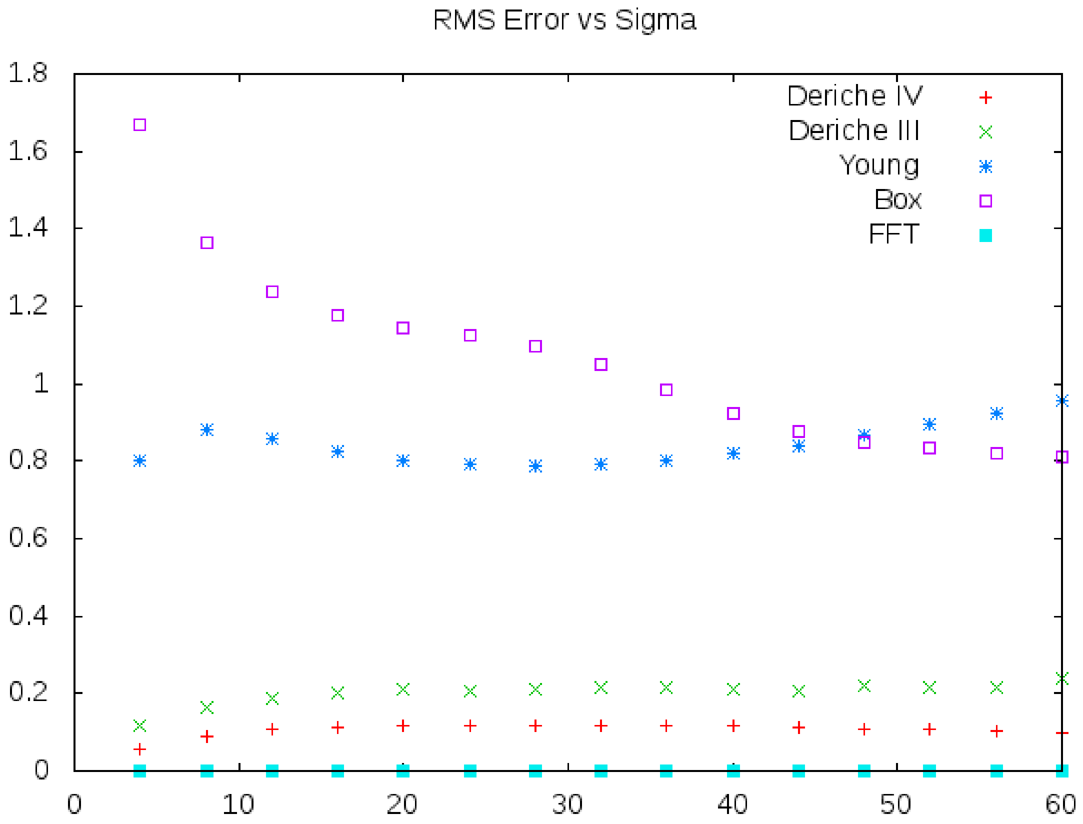
- **Young & van Vliet Recursive Gaussian Filter**
  In this [2] algorithm, the frequency domain Gaussian is approximated by a rational approximation from the book 'Handbook of Mathematical Functions' by Abramowitz and Stegun. This leads to the formulation of a causal and an anticausal transfer function, both of 3rd order. These transfer functions are converted to difference equations which are implemented recursively. The replicate and symmetric paddings were tried in Young as well.

- **Cascaded Uniform Filters**
  This [3] algorithm is based on the Central Limit Theorem, which suggests that the Gaussian filter can be approximated by a cascade of simple filters. A uniform-coefficient FIR filter is used as the simple filter as it can be implemented recursively. In my implementation, I convolved the uniform filter with the image 3 times to approximate the Gaussian filter. From the fact that variance adds on convolving, the following expression is used for the uniform filter width: $N=\sqrt{4\sigma^2+1}$.

- **FFT Implementation**
  This is based on the fact that convolving in the spatial domain corresponds to multiplication in the frequency domain. In my implementation, I computed the real FFTs of the image and the filter, multiplied them and computed the inverse real FFT. The number of operations per point does not depend on the filter size, but depends on the size of the image. Since the result obtained from this corresponds to periodic convolution, I padded the image as {f1, f2, ..., fn, fn, ..., f2, f1} so that even with periodic convolution happening, the result I get is that of linear convolution with symmetric padding.

In all the above algorithms, the fact that the 2D Gaussian filter is separable into two 1D Gaussian filters was made use of and so, all algorithms involved 1D convolution row wise and column wise. First, these algorithms were implemented in MATLAB for preliminary evaluation. Then these were implemented in C and the runtime of each algorithm was recorded and the errors computed with respect to the direct implementation of Gaussian FIR filter. Plots of results obtained by applying the filter (C Implementation) on a 512x512 image 'Lena' for a range of values of sigma are presented here.

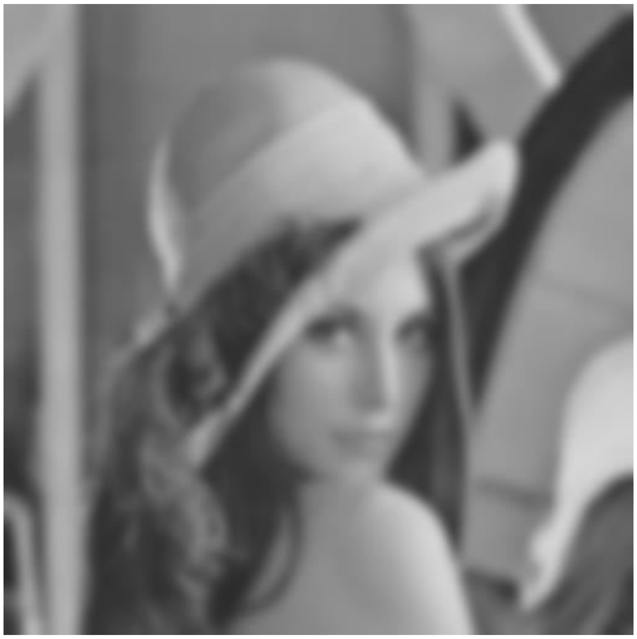




Since there is not much error difference between the 3rd and 4th order Deriche filters, order 3 is optimal as it results in a significant reduction in runtime. The Young filter requires an even lesser runtime but produces more error than Deriche. Any one of the two: Deriche 3rd order or Young can be a good choice of an O(1) Gaussian filter depending on the application. If speed is more important, Young filter can be used. If better accuracy is desired, Deriche filter will be applicable.

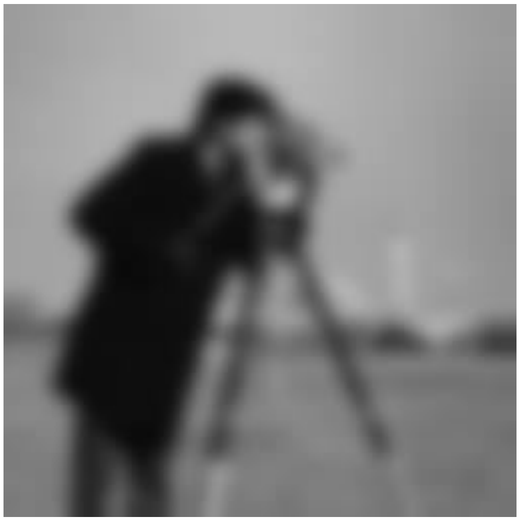
Fast Gaussian Filter Results on Natural Images

Input 'Lena' 512x512



Gaussian (Deriche) Filter Output (sigma=5)

Input 'Cameraman' 256x256



Gaussian (Deriche) Filter Output (sigma=5)

Fast Bilateral Filter Implementation

The next part of my project was to work on the implementation and optimization of algorithms presented in papers [5], [6], and [7], and integrate the fast Gaussian filter module with it. Hereunder are the brief descriptions of these papers.

Trigonometric Range Kernels [5]

The cosine is a shiftable kernel, $\phi(s)=\cos(\gamma s)$

$$\cos(\gamma(s-\tau))=\cos(\gamma\tau)\cos(\gamma s)+\sin(\gamma\tau)\sin(\gamma s)$$

This idea is extended to more general trigonometric functions of form $\phi(s)=a_0+a_1\cos(\gamma s)+\ldots+a_N\cos(N\gamma s)$. This is done by writing in terms of complex exponentials,

$$\phi(s)=\sum_{|n|\leq N} c_n \exp(jn\gamma s).$$

Coefficients must be real and symmetric since $\phi(s)$ is real and symmetric.

$$\phi(s-\tau)=\sum_{|n|\leqslant N} c_n \exp(-jn\gamma\tau)\exp(jn\gamma s)$$

The properties of symmetry, nonnegativity, and monotonicity required by a range kernel are offered by the family of raised cosines of the form:

$$\phi(s)=[\cos(\gamma s)]^N=\sum_{n=0}^{N} 2^{-N}\binom{N}{n}\exp\left(j(2n-N)\gamma s\right) \qquad \gamma=\frac{\pi}{2T} \qquad (-T\leqslant s\leqslant T)$$

Since $\phi(s)$ has a total of N+1 terms, this gives a total of 2(N+1) auxiliary images for spatial convolution.

Even though the raised cosine kernel becomes more Gaussian like as N increases but it converges pointwise to 0 at all points as N gets large, except for node points 0, $\pm\pi$, $\pm 2\pi$, …

The paper addresses this problem by scaling the raised cosine with increasing N and proves the following pointwise convergence:

$$\lim_{n\to\infty}\left[\cos\left(\frac{\gamma s}{\sqrt{N}}\right)\right]^N=\exp\left(\frac{-\gamma^2 s^2}{2}\right)$$

But practically, a good Gaussian approximation can be achieved using a reasonable value of N. The sigmar for above Gaussian is equal to (1/gamma). Hence in the implementation, gamma is used to control the width of the kernel. For the monotonicity and non-negativity of the kernel, the value of N is set such that half period of the raised cosine just lies on [-T,T], which requires $\gamma T/\sqrt{N}=\pi/2$, and then multiplied by a factor of 4 for further accuracy.

Acceleration of Shiftable Algorithms [6]

This paper describes a fast algorithm for finding T as a smaller T requires lesser number of coefficients.

$$T=\max_{x}\max_{\|y\|\leq R}|f(x-y)-f(x)|$$

$$T=\max_{x}\left[\max_{\|y\|\leqslant R} f(x-y)-f(x)\right]$$

The Max-Filter algorithm is used for computing $\max_{|y|\leq R} f(x-y)$ at every x. Further, T is computed using above equation.

The paper also describes a method to reduce the number of raised cosine coefficients for narrow kernels by truncating the smaller coefficients while keeping the oscillatory error at the tails within some tolerance limit.

Fourier Kernels [7]

The Fourier bases, i.e. the complex exponentials are shiftable as proved above in the description of [5]. The symmetric range kernel $\Phi(t)$ can be approximated on the interval [0,T] by the shiftable Fourier basis for some order N. A symmetric kernel can be approximated by the shiftable function:

$$\varphi_N(t) = d_0 + \sum_{n=1}^{N} d_n \cos(n\omega t)$$

This can further be written as a Fourier basis approximation:

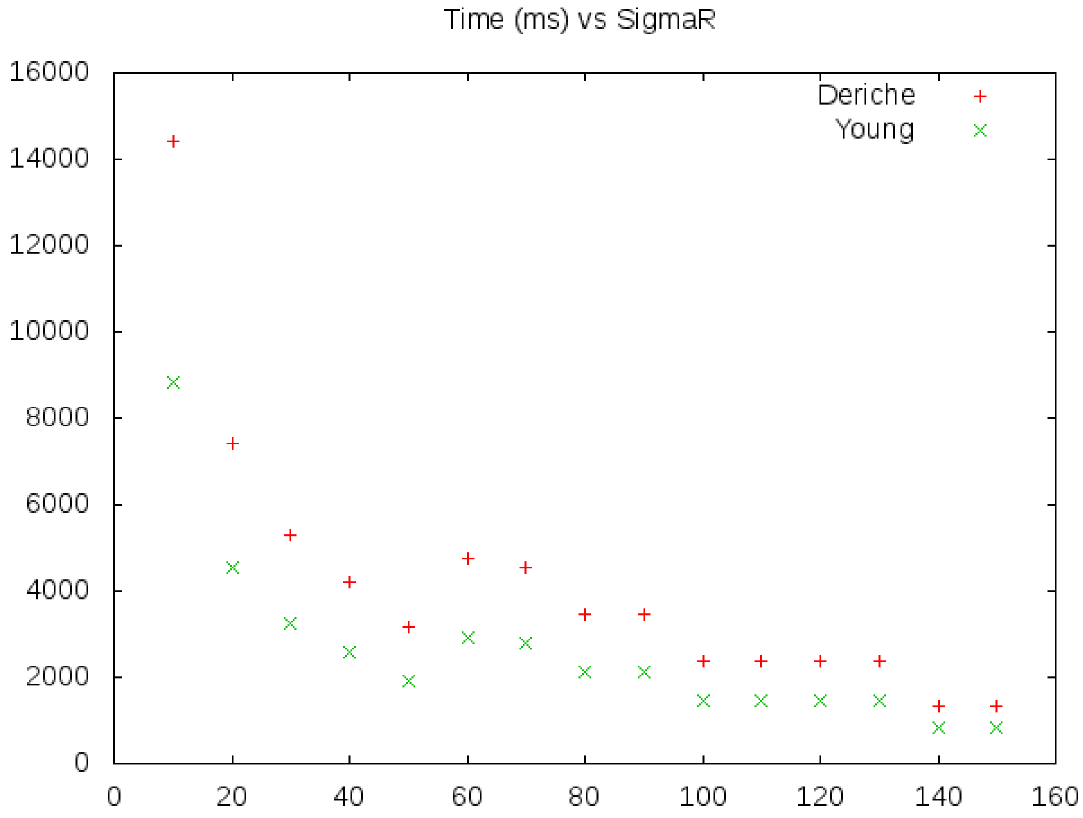
$$\varphi_N(t) = \sum_{n=-N}^{N} c_n \exp(\iota n\omega t)$$

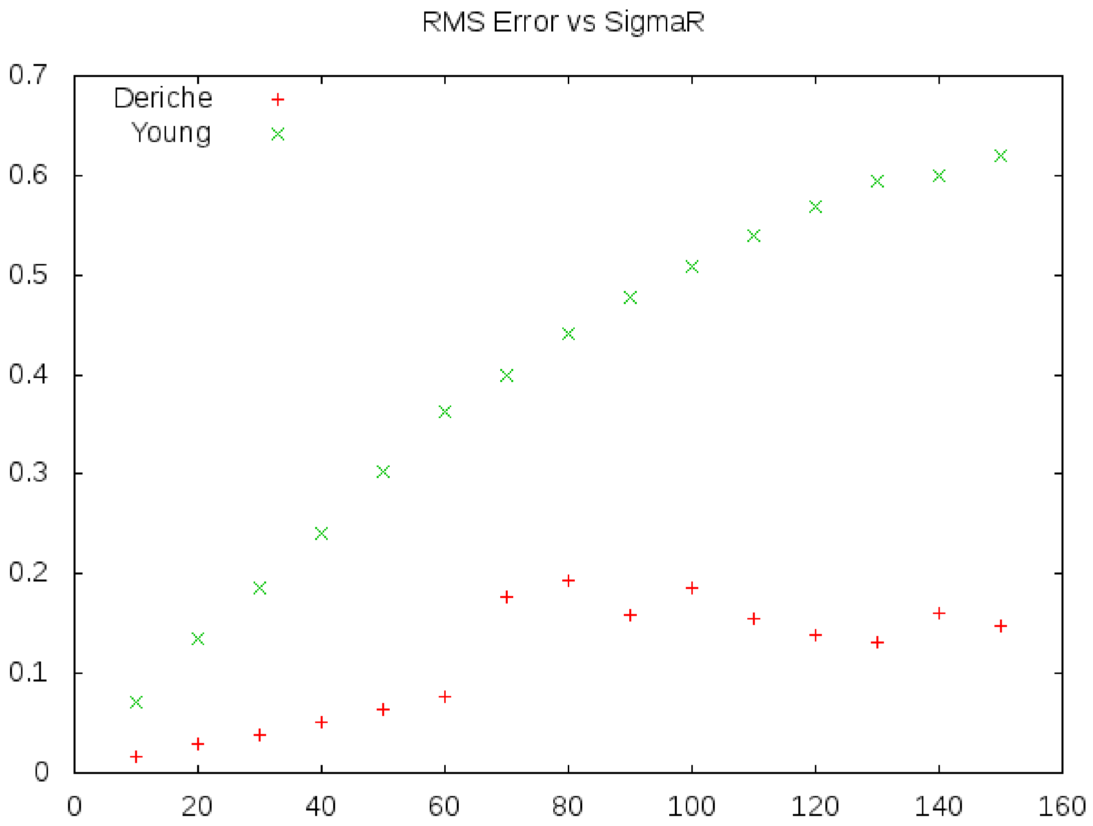
For a particular order N, a linear Least Squares problem is solved to obtain the coefficients $d_0, \ldots., d_N$. As $N \rightarrow \infty$, the Fourier basis represents the exact kernel (Fourier series of the kernel). As N is increased, it results in a better approximation of the kernel. In the implementation, we fix a tolerance limit eps and start with order N=1. Then the LS problem is solved and we check whether the square error is within the tolerance eps. If it is not, N is incremented and the procedure is continued until the LS square error is reduced to less than eps. The LS problems are solved via QR factorization recursively.

Existing MATLAB Implementation – FastBilateralFilterV2 on File Exchange (FBFV2)

It computes the maximum local dynamic range of the image, T using the algorithm described in [6]. Based on the ratio T/sigmar, Raised Cosine approximation is used for higher sigmar and Fourier Basis approximation for lower values of sigmar. The Gaussian spatial convolution is performed using *imfilter* from the Image Processing Toolbox. The reason for this bifurcation of algorithms is that it requires a very large power of cosine to approximate narrow Gaussian kernels whereas the same can be approximated by significantly lesser number of Fourier coefficients, Also the number of Fourier coefficients required starts to blow up as the extent of the Gaussian kernel increases beyond T. Since raised cosines can approximate wide kernels with lesser number of coefficients, it is employed for higher sigmar. The ratio used is 3.5.

I implemented the above algorithm in C using Deriche and Young as the Gaussian filters. Plots of results obtained on a 512x512 image 'Lena' for sigmas=10 and a range of values of sigmar are presented here. All the error plots henceforth compute the error with respect to the Direct Implementation of Bilateral Filter.



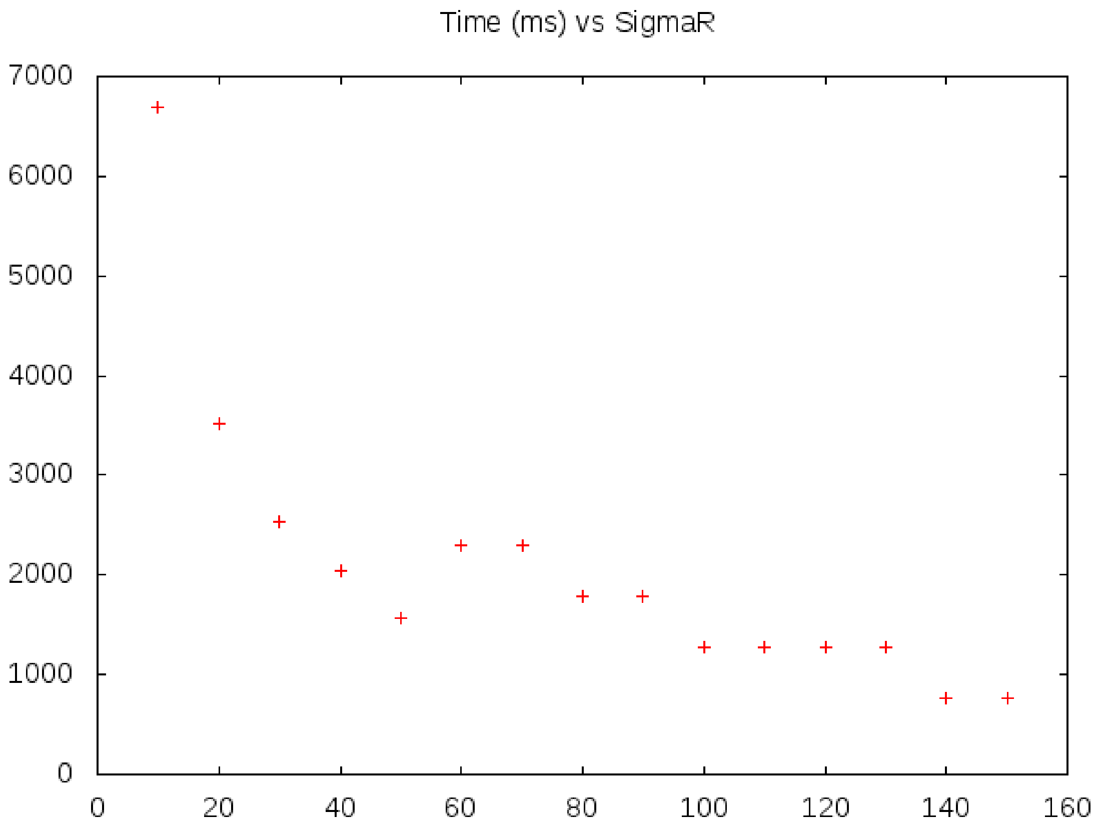


Due to the speed of Young being much superior to that of Deriche and accuracy also being quite reasonable, I decided to proceed with using Young's algorithm as the Gaussian filter module.

Auxiliary Image Recursion

The FBFV2 computes the auxiliary image $H=\exp(j\omega_k f(x))$ for each spatial convolution by exponentiating the input image. This consumes a significant amount of time as it requires calculating the exponential of each pixel of the image for every convolution. I introduced recursion into this computation of H. The input image needs to be exponentiated only twice, once for computing the first H and to compute the recursive auxiliary image $Ho=\exp(j\omega_o f(x))$. Further auxiliary images H can be computed by multiplying previous H by Ho.

C Implementation Plot on 512x512 'Lena', sigmas=10
It can be observed that the execution time has significantly decreased.

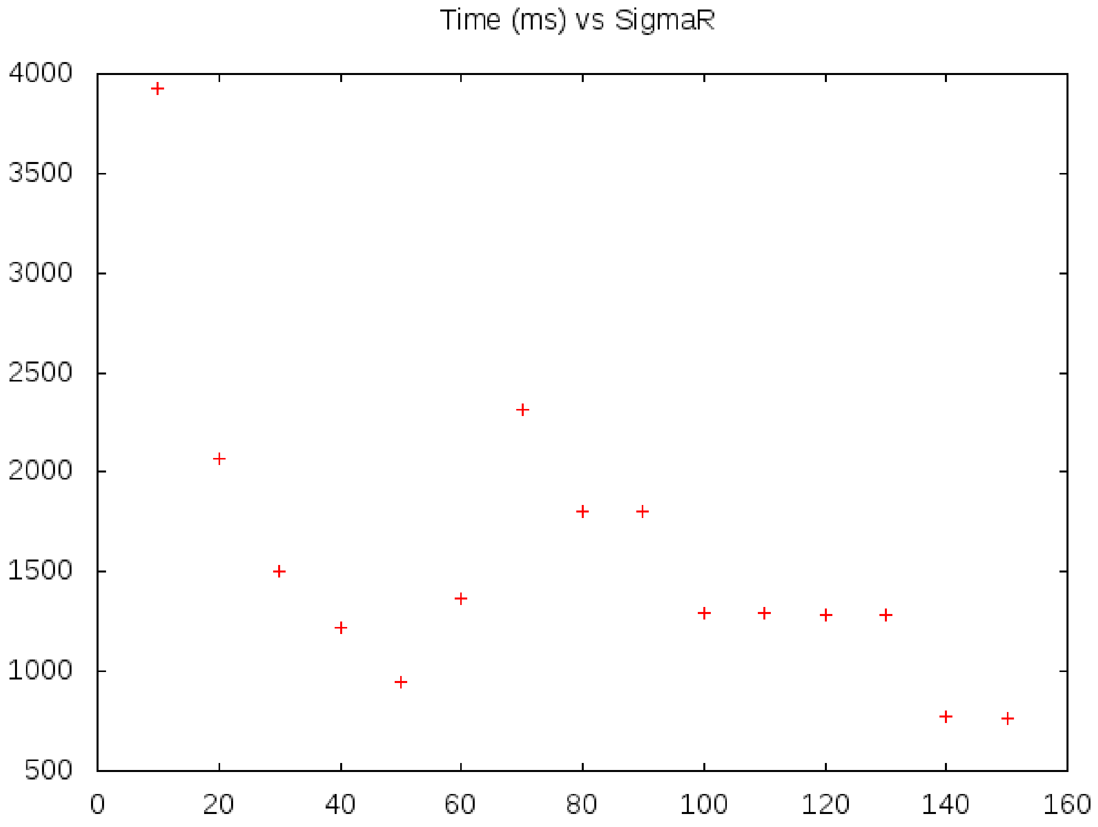


Negative Frequency Convolutions

In Fourier Basis algorithm, spatial convolutions are to be performed for both positive frequency exponentials and negative frequency exponentials. The FBFV2 computes the auxiliary images for the positive frequencies and applies spatial Gaussian filter to them. It then computes auxiliary images for negative frequencies and applies spatial filter to them. I optimized this computation, making use of the fact that the negative frequency gives conjugate of the auxiliary image given by positive frequency, and convolving the conjugate of an image with a filter is equivalent to computing the conjugate of the convolution. So, the computation of auxiliary images and spatial convolution is to be performed only for positive frequencies. The negative counterparts are computed just by taking conjugate of the positive ones. This reduced the execution time by almost half.

C Implementation Plot on 512x512 'Lena', sigmas=10

It can be observed that the execution time in the Fourier basis range has decreased by almost half.



Parallel Programming in C Implementation

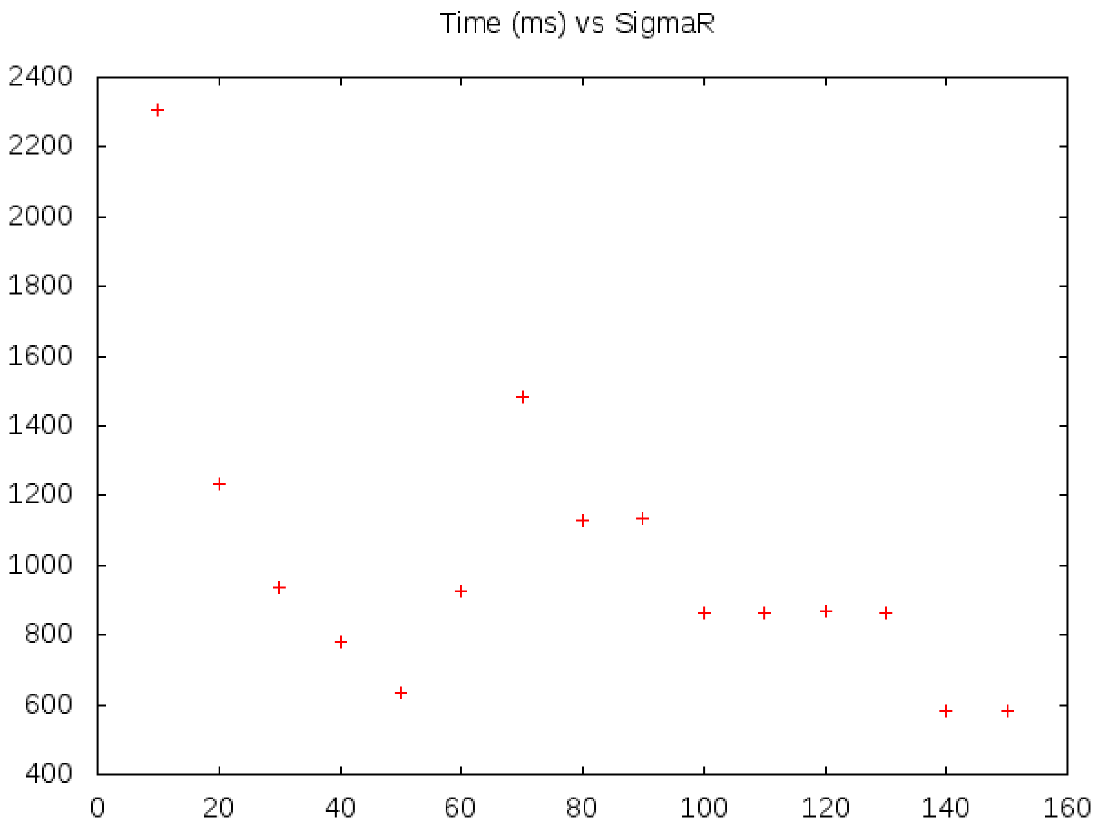
The spatial convolutions performed on the auxiliary images generated by distinct frequency terms are independent of each other. Hence they could be performed simultaneously on separate cores of the system. The *imfilter* routine of MATLAB is hardware optimized and multi-threaded. So its performance is already very optimum. But the aforementioned independence of spatial convolutions could be taken advantage of in the C implementation to make it even faster.

I implemented multi-threading in C using the OpenMP API. By default, when a parallel region is forked using *'#paragma omp parallel'* directive, the number of threads created are equal to the number of hyperthreads on the system. But this number may or may not be equal to the number of physical cores on the system. We would achieve the best performance by forking only as many threads as the number of physical cores and assigning each thread to a distinct physical core. This is because every thread occupies part of the RAM to store auxiliary images and their outputs and as the number of threads increase, RAM consumption increases. So it must be ensured that there are no redundant threads running on the same core. Simply reducing the number of threads wouldn't suffice as they may be assigned to hyperthreads of the same physical core, which would not produce the best performance possible. To get round this issue, I wrote a routine that extracts the number of physical cores from the system info, forks a parallel region with as many threads and sets the affinity of each thread to a hyperthread on a distinct physical core. I also limited the number of running threads to 4 so as to have a reasonable limit on RAM consumption.

To distribute the spatial convolutions among threads, I used static scheduling. Dynamic scheduling of threads randomly distributes the iterations to threads depending on whichever thread is free at the moment. But this would have prevented me from doing the Auxiliary Image recursion. Whereas in case of static scheduling, we know beforehand which iteration will be running on which thread. At fork, I distributed chunks of convolutions to each thread depending on the number of threads. Hence the input image is required to be exponentiated only once for each thread and the subsequent auxiliary images are computed by recursion.

C Implementation Plot on 512x512 'Lena', sigmas=10

On my 1.7 GHz 2-Core system, the execution time further decreased to almost half after I implemented multi-threading with 2 threads on the 2 cores.

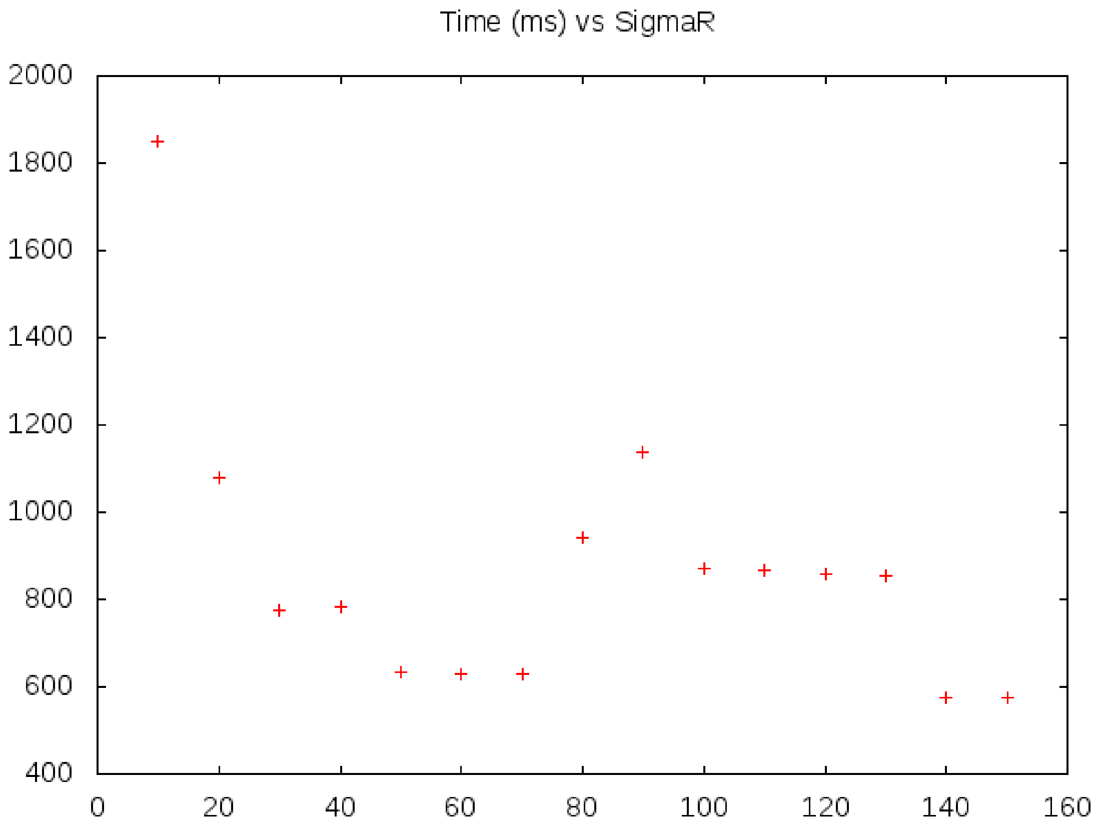


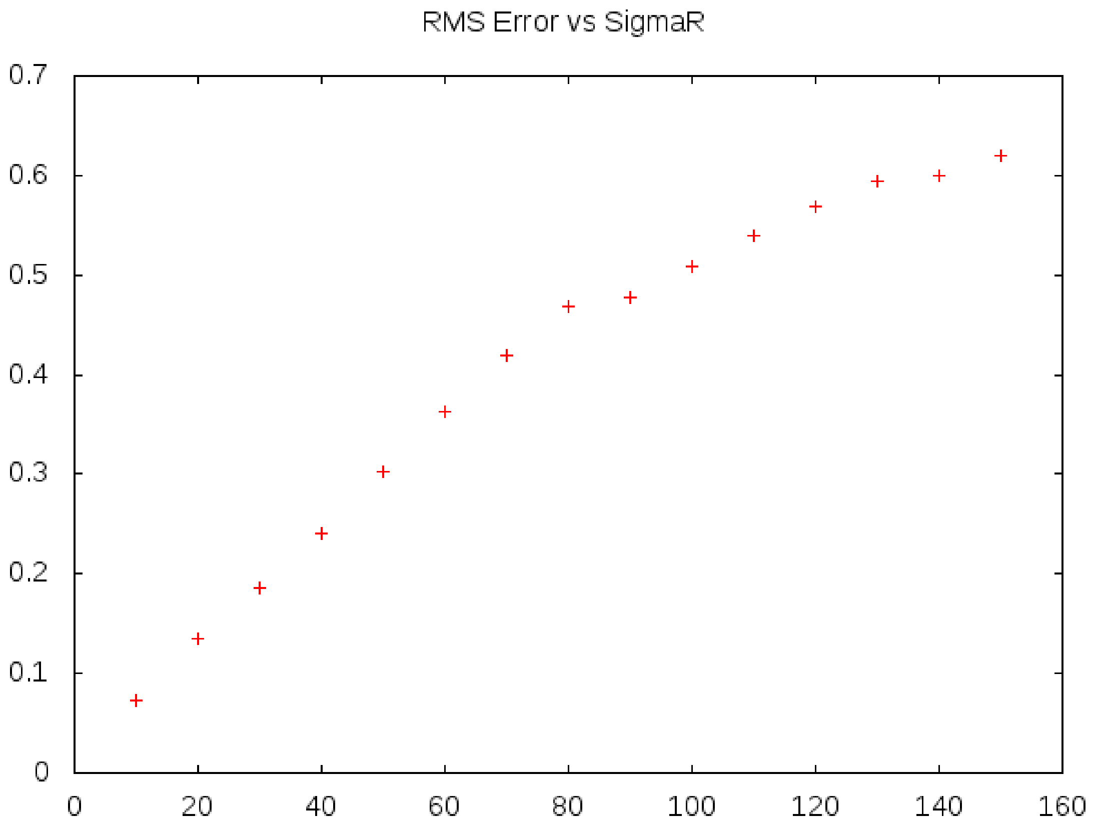
Optimizing Fourier Basis Tolerance Limit eps

The threshold of range kernel square error eps being used in FBFV2 – 1e-3 was too conservative. It required a large number of Fourier coefficients to approximate narrow kernels, and also kernels wider than the [-T,T] extent. I did some testing on higher values of eps and the results were as described. Higher values close to 1e-1 resulted in significant ripples and some significant negative weights near the tail of the kernel. Since our normalization is meant only for positive kernels, this resulted in pixel outputs going out of bounds, and large l_∞ errors. But values close to 1e-2 didn't produce any significant ripples and performed almost as well as 1e-3 in terms of accuracy, and notably reduced the number of Fourier coefficients required for both very narrow kernels and kernels exceeding the [-T,T] extent. The latter outcome gives a significant performance advantage to Fourier Basis over Raised Cosine, due to which I was able to extend the Fourier Basis algorithm over a larger range of sigmar and reduce the ratio T/sigmar to around 2.5. Hence the value of eps was modified from 1e-3 to 1e-2.

C Implementation Plots on 512x512 'Lena', sigmas=10

The decrease in execution time due to reduction in number of Fourier coefficients and extension of Fourier basis range can be observed from the plot. Also, the error plot is almost similar to the one plotted earlier.



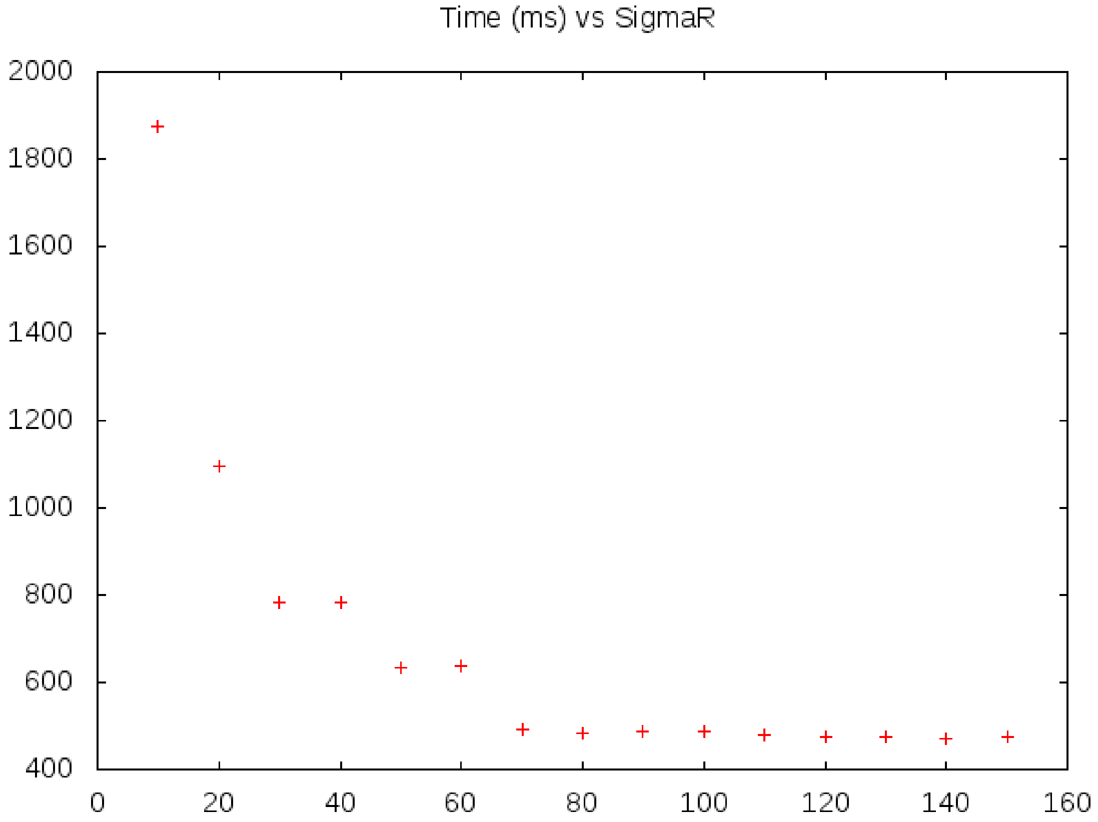


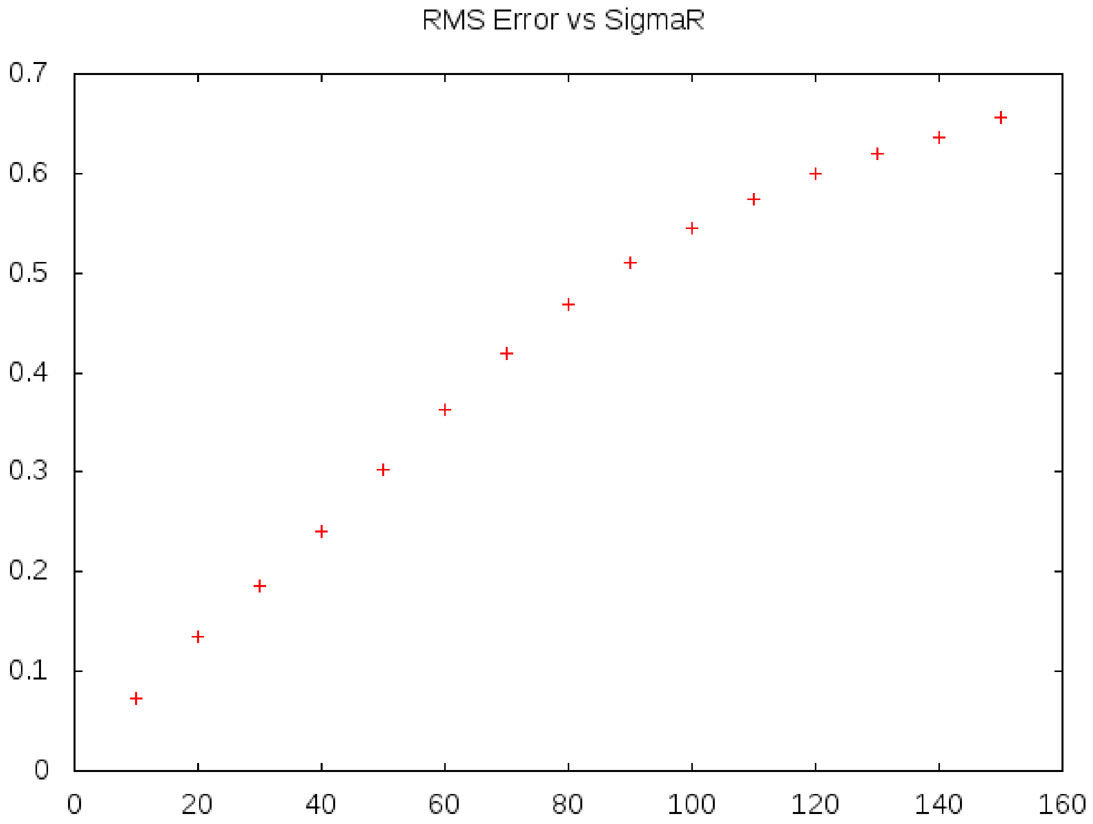
Removal of Raised Cosine

The Fourier Basis coefficients start to blow up after a certain value of sigmar. The reason for this is that as the extent of the kernel exceeds T and the period of the approximation is maintained constant at T, there is a sudden drop in the single-period kernel from some finite positive value to zero. This requires higher frequency Fourier bases to model the kernel. Keeping the period fixed at T requires a large number of Fourier bases to approximate wider kernels, and hence large number of spatial convolutions. This gave a performance advantage to Raised Cosine algorithm for wider kernels. I solved this problem by varying the period for wider kernels so as to enclose the whole extent of the kernel within one period. The new period T0 has to be varied in such a way that the plot of the number of coefficients vs sigmar for any T is monotonic and decreasing. I experimentally arrived at an optimal value of 3.2 for the ratio T0/sigmar which makes this plot monotonic. Of course, the minimum value of T0 is required to be T because we want to model the kernel for interval [-T,T]. So for kernels of sigmar less than T/3.2, the period T0 is kept fixed at T. The way this period variation works is that even though the fundamental frequency decreases with increasing sigmar, the width of the frequency domain kernel decreases in the same proportion. Thus the number of required Fourier bases remains almost constant. This optimization gave Fourier basis a significant performance advantage over Raised cosine even for wider kernels. And since the accuracy of the Fourier basis approximation is much superior to Raised cosine, there was no reason to continue with the latter. Now, Fourier Basis is the only algorithm being used for range kernel approximation.

C Implementation Plots on 512x512 'Lena', sigmas=10

The very low and almost constant execution times for wider kernels can be observed.



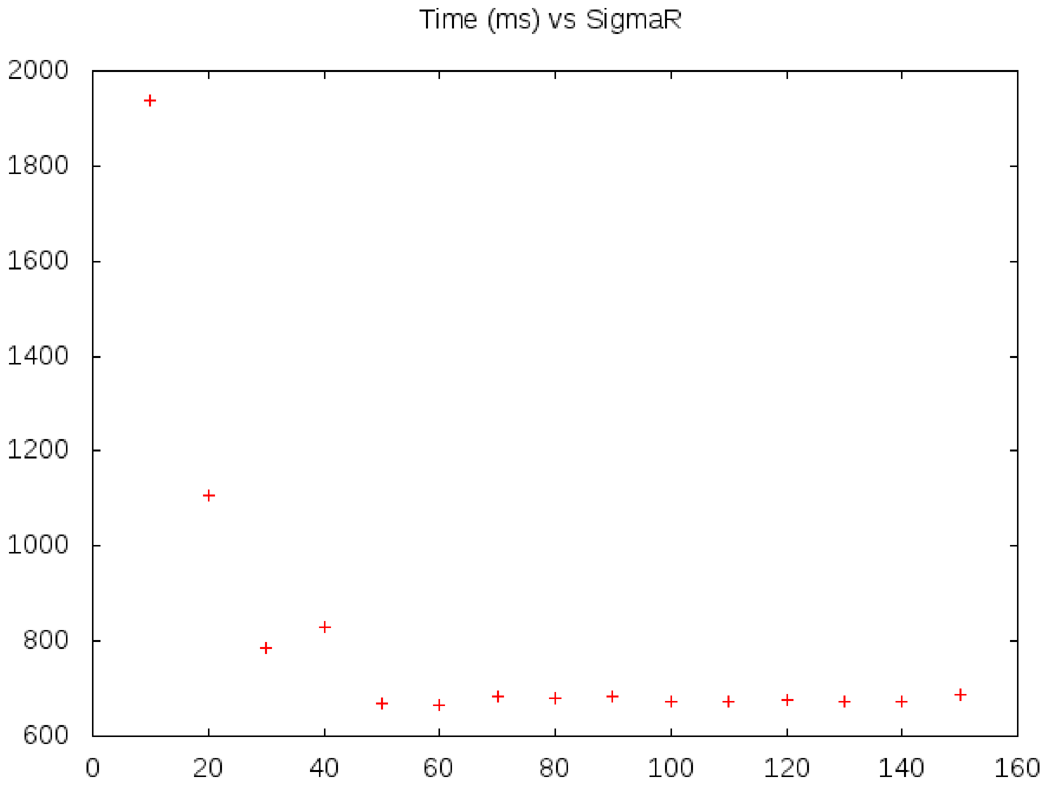


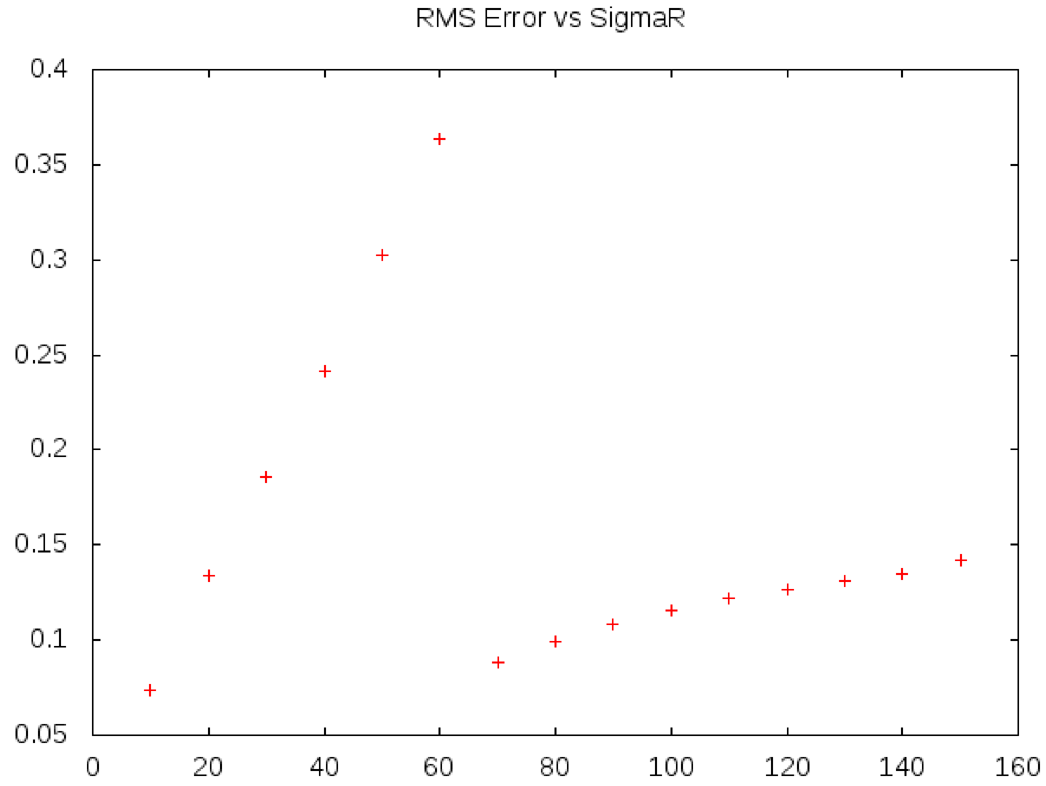
Bifurcation into Young and Deriche (C Implementation)

As can be seen from the plots above, the execution times for wider kernels are exceptionally good, while the RMS error goes on increasing with higher sigmar. The execution times are lower than we aimed for and so could be traded off for better accuracy. Using Deriche is a good choice in this scenario as it offers this very trade off. The bifurcation was to be done on the basis of number of coefficients as that is what governs the execution time. Since the maximum threads I fork is 4, the bifurcation had to be done at a multiple of 4. I decided to use Deriche for 4 or lesser number of coefficients and Young for higher number. Since the number of coefficients depends on the extent of the kernel with respect to T, I conducted a test on it and found that below T/sigmar = 3.5, the number of coefficients is 4 or less. So I use this optimal ratio of T/sigmar = 3.5 to determine the algorithm to be used for spatial convolution.

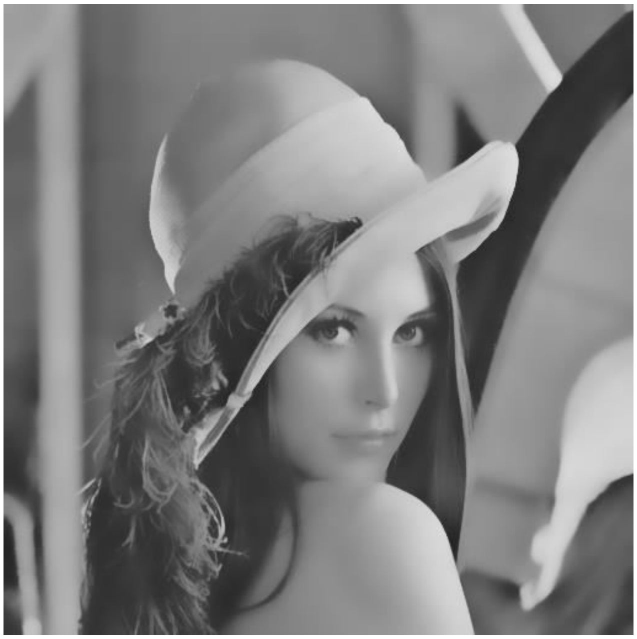
Final C Implementation Plots on 512x512 'Lena', sigmas=10

It can be observed that the execution times are still pretty good while the errors for large sigmar have decreased by a significant amount.

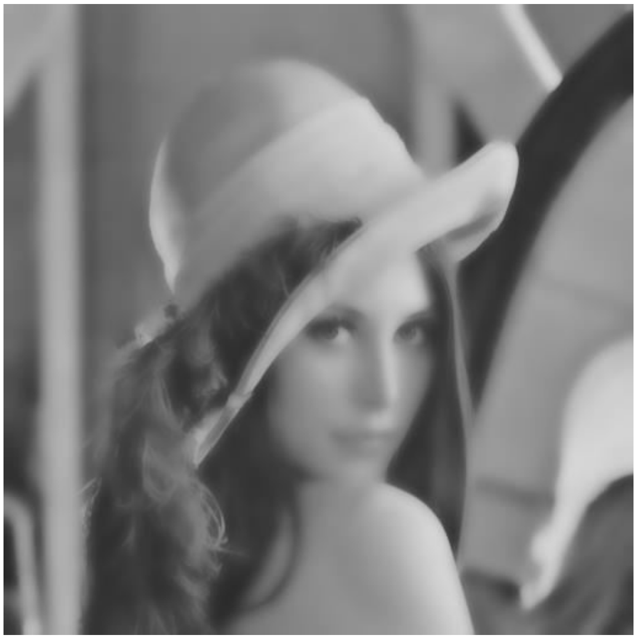

Time (ms) vs SigmaR
2000
1800
1600
1400
1200
1000
800
600
0
20
40
60
80
100
120
140
160


RMS Error vs SigmaR
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
20
40
60
80
100
120
140
160

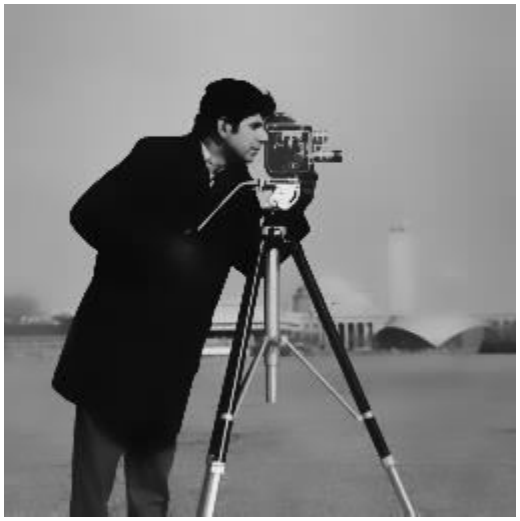
Fast Bilateral Filter Results on Natural Images



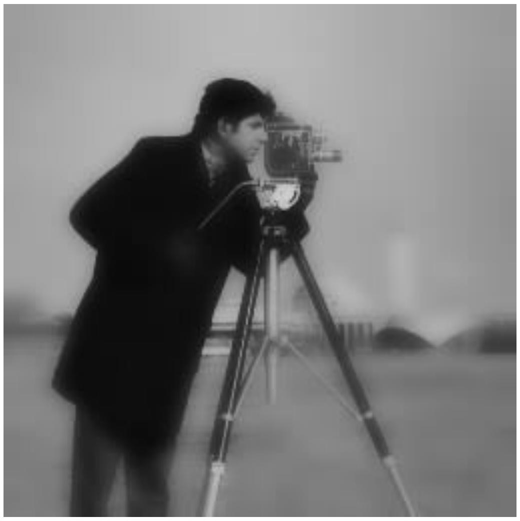
Bilateral Filter Output (sigmas=5, sigmar=40)



Bilateral Filter Output (sigmas=5, sigmar=100)



Bilateral Filter Output (sigmas=5, sigmar=40)

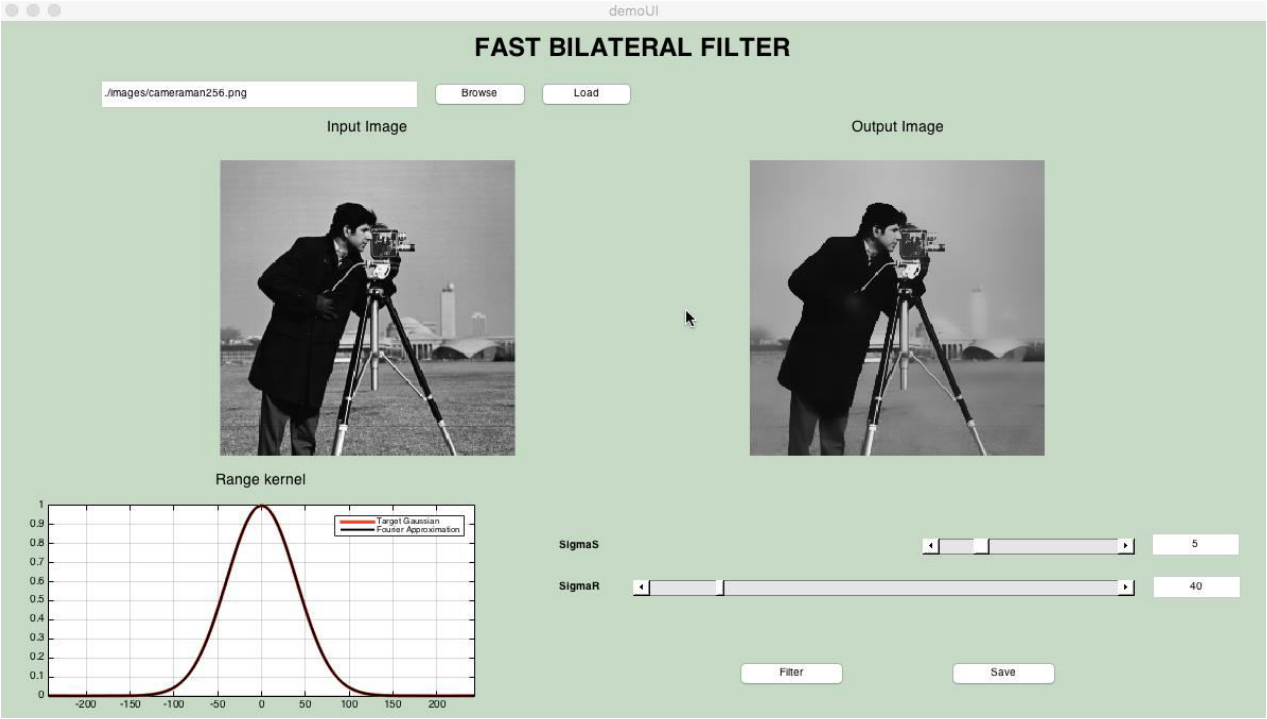


Bilateral Filter Output (sigmas=5, sigmar=100)

MATLAB GUI

As the optimized implementation is quite fast and gives results almost in real time, I developed an interactive GUI for the MATLAB code. It features UI controls to load an image from the memory, sliders to interactively set sigmas and sigmar, and on filtering displays the output image and plots the target and approximate range kernels. The output image can be saved to memory. The MATLAB implementation with all the optimized modifications and the GUI, has been updated on the File Exchange of MATLAB Central.
Link: http://www.mathworks.com/matlabcentral/fileexchange/36657-fast-bilateral-filter

MATLAB GUI Snapshot



./images/cameraman256.png

SigmaS 5

SigmaR 40

REFERENCES

1. Rachid Deriche. "Recursively implementating the Gaussian and its derivatives." [Research Report] RR-1893, INRIA. 1993, pp.24. <inria-00074778>

2. Ian T. Young, Lucas J. van Vliet. "Recursive implementation of the Gaussian filter." Signal Processing 44 (1995) 139-151.

3. William M. Wells. "Efficient Synthesis of Gaussian Filters by Cascaded Uniform Filters." IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-8, No. 2, March 1986, 234-239.

4. F. Porikli, “Constant time O(1) bilateral filtering,” Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp. 1-8, 2008.

5. K. N. Chaudhury, D. Sage, and M. Unser, “Fast O(1) bilateral filtering using trigonometric range kernels,” IEEE Transactions on Image Processing, vol. 20, no. 12, pp. 3376-3382, 2011.

6. K. N. Chaudhury, “Acceleration of the shiftable algorithm for bilateral filtering and nonlocal means,” IEEE Transactions on Image Processing, vol. 22, no. 4, pp. 1291-1300, 2013.

7. S. Ghosh, K. N. Chaudhury, "On Fast Bilateral Filtering Using Fourier Kernels," IEEE Signal Processing Letters, vol. 23, no. 5, pp. 570-573, 2016.