GPU Based Bilateral Filtering for Images

Paul Logas

Computer Engineering Department

University of Florida

Gainesville, FL

jacobpaullogas@outlook.com

*Abstract*—A Bilateral filter smooths images while preserving edges, this is accomplished through nonlinear combination of a neighborhood of image values. The method combines colors or intensities based on geometric closeness and photometric similarities. Bilateral filters can enforce the perception metric of the CIE-Lab color space to smooth colors and preserve edges tuned to human perception. Bilateral filtering has the added benefit of an absence of phantom colors along the edges of a color image. A naïve, iterative solution to Bilateral filtering will iterates over all the input image’s pixels. The iterative and local nature of this algorithm has tremendous potential for parallel GPU implementations.

Keywords—bilateral; parallel; filtering; GPU

# Introduction

One of the most fundamental operations of computer vision and image processing is filtering. Broadly defined, a value of a filtered image at some location is a function of color or intensity values in a neighborhood about the location. For example, a Gaussian low-pass filter, one of the most commonly used filters, computes a weighted average value for the location in question. The weights are computed to have a higher weight for values nearby the location and lower weights for values further away. A more formal explanation of the weight fall-off can be found [1], intuitively an image varies little over a small space, thus nearby pixels are going to be similar. By averaging these pixels together, the image function is smoothed and noise is removed.

The assumption of a slow spatial variation fails at edge cases, a pure Gaussian filter will blur image edges as it does not consider legitimate variations in pixel intensities or color values. This is a major issue for computer vision applications, cameras inherently introduce noise which could negatively alter feature matching techniques however if a standard Gaussian filter is applied to the image edge information can be completely lost. Edge based segmentation for example look for abrupt changes in intensity. However, if an image is not smoothed than false edges can be detected. There have been attempts to reduce the effect of smoothing as well as do without smoothing altogether [2] [3].

One popular answer to this problem is anisotropic diffusion [4]. In this approach, the local image variation is measured at every point with pixel values being averaged from neighborhoods with size and shape depending on local variation. Diffusion methods average over an extended region through solving partial differential equations and are thus iterative inherently. In addition to possible efficiency issues, iteration can raise issues of stability.

A scheme introduced by C. Tomasi and R. Manduchi [5] provides a non-iterative and simple scheme for smoothing that preserves edges. The basic idea of this approach is to do what traditional filters do in the image domain, in the image range. Two pixels can either be spatially near each other or have values near each other, in a perceptually meaningful way. Closeness is specific to the vicinity in the domain, while similarity refers to the vicinity in the range. A traditional filter is filters by domain, enforcing closeness by weighing pixel values with coefficients which fall off over a distance. Range filtering on the other hand, averages image values with weights decaying by similarity. A range filter is nonlinear as weights depend on intensity or color of the image.

In the scheme, spatial locality is not altogether disregarded. Range filtering alone will only distort the image’s color map. Thus, the scheme combines range and domain filtering. This is what is referred to as *bilateral* filtering [5].

As this *bilateral* filters have an explicit definition for distance in an image function’s domain and range, they can be applied to any function in which these distances can be defined. Applying this filter to a color image is just as simple as applying to black-and-white images. The CIE-Lab color space gives the color space a meaningful measure of color similarity which in short distances correlate with human color discrimination performance. A bilateral filter often uses this metric as it provides a smooth image with edges that are preserved in a way that is tuned to human performance. In other words, colors that are perceptually similar are averaged together and perceptually visible edges are preserved.

This scheme introduced by C. Tomasi and R. Manduchi [5] is non-iterative and so each pixel must only be evaluated once. This is a perfect candidate for implementation in a massively multi-threaded environment such as a GPU which is made up of hundreds if not thousands of computation units providing excellent throughput [6]. With no one output pixel depending on the output of another pixel and a pixel’s output depending upon a single set of calculations, a GPU’s high throughput can be used to great effect.

# Overview

## Premise

A low-pass filter applied to an image produces an output defined as:

|  |  |  |
| --- | --- | --- |
|  |  | 1 |

where is the measure of the closeness in geometric terms between the center and a point nearby . Both the output function **h** and input function **f** can be multiband. If low-pass filtering is to preserve the dc component of low-pass signals

|  |  |  |
| --- | --- | --- |
|  |  | 2 |

If filter is shift-invariant than is purely a function of the vector difference and will be constant.

Like domain filtering, range filtering is defined as:

|  |  |  |
| --- | --- | --- |
|  |  | 3 |

where is the measure of the photometric similarity between the pixel value at the center and some nearby point . The similarity function therefore deals with the range of the image function, while closeness function operates in the domain of the image function. The constant for normalization ( 2 ) is replaced in the range domain by

|  |  |  |
| --- | --- | --- |
|  |  | 4 |

Instead of depending on the distance from center as the function does, the normalization for the similarity function is dependent upon the image function. The similarity function is said to be *unbiased* if it depends on the difference alone.

In these equations, the spatial distribution of the image intensities has no role in range filtering alone. However, a combination of intensities from the entire image does not make much sense as values a large distance from should have no bearing on the final value at location . An appropriate solution to this shortcoming in range filtering is to combine both range and domain filtering into one filter. This achieves both geometric and photometric locality. Combination of the filters is simply:

**Figure 1**: (a) A level step image with Gaussian noise. (b) The surface plot of the unfiltered image. (c) Surface plot after Gaussian filter applied. (d) Surface plot after bilateral filter applied. (e) The output image after bilateral filter applied.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| (a) | (b) | (c) | (d) | (e) |

|  |  |  |
| --- | --- | --- |
|  |  | 5 |

with the normalization as a simple combination of domain and range normalization

|  |  |  |
| --- | --- | --- |
|  |  | 6 |

The combination of the two filters is termed *bilateral filtering*. This filter replaces the pixel value at a point with an average of similar and nearby pixel values. Regions without edges have pixel values that are similar to each other which results in ( 4 ) being close to one. Thus, the filter in these regions is essentially a domain filter, averaging away weakly correlated pixel differences that are caused by noise.

Now considering an image function such as the one in Figure 1(a), which has a sharp boundary between a dark and a light region. When the similarity function is centered on a pixel on or near the edge, weights for the pixel values on the same side as the center will be near one. Conversely, weights on the other side of the image edge will be near zero. Thus, the desired effect is achieved, as shown in Figure 1(d), when the range filtering is combined with the domain filtering.

Also, it can be observed that equations (1) and (3) are independent of other computed values of x. This means for all x can be computed in any order. Thus, it can be computed concurrently and with current GPU technologies, it is not unreasonable for a compute thread to be launched for each pixel.

## Gaussian

The shift-invariant Gaussian filter, in which both the domain, , and range, , closeness functions are Gaussian functions of Euclidean distance between the arguments. The function is radially symmetric

where the Euclidean distance between point and is

This equation is recognizable as the Gaussian point-spread function without the normalization coefficient [1]. Now we wish to define in a way that will provide a point-spread of the range. The similarity function as it turns out is analogous to the function so it is defined as

where

Instead of the measure of the Euclidean distance as it was for the domain function, is the measure of distance between two intensity or color values and .

There are two values, and , in the bilateral filter. The value for is the for the domain part of the filter, thus it acts exactly as the in a standard Gaussian blurring filter would. A large blurs more, as it combines pixel values from a greater distance in the image. The must be changed if one desires the same results from an image that is scaled up or down. The other in a bilateral filter, , is termed the photometric spread. This value is set in order to achieve the desired combination of pixel values. Pixels with values much closer to each other than are mixed while values more distant than are not. Similar to when the image is edited the value must be changed to achieve similar results. In this case, must be changed if an image is amplified or attenuated. [5]

# Parallel Implementation

A naïve implementation of the bilateral filter was implemented to run serially on the CPU as a reference. This program runs a bilateral filter iteratively over all pixels in the image, one at a time. The spatial Gaussian kernel is calculated at startup to reduce redundant computations while the range Gaussian kernel is calculated for each pixel.

As the naïve serial implementation is iterative over all pixels in the image, the transfer to a parallel GPU implementation was straightforward. Basically, the code for each individual pixel is used as a basis for the GPU kernel. Like the serial implementation, the spatial Gaussian kernel is precomputed.

Most of the computation time in the GPU kernel is spent calculating the Gaussian of color intensities. The CUDA framework provides faster implementations of these functions at a negligible precision loss.

The GPU kernel takes as input three float pointers, a handful of integers, and a single float. The image matrix as well as the spatial kernel matrix are represented by float pointers with color images having the three values adjacent to each other. The last float pointer is the output array to which the calculated pixels are written. As the spatial kernel is already calculated and passed in its entirety to the GPU, only the range sigma is required which is represented as a float. Finally, the image width and height as well as the Gaussian kernel width are passed in as integers.

Two GPU kernels exist to calculate the binomial filter for two cases: color and grayscale. When discussing the algorithm, the grayscale implementation will be referenced for brevity’s sake.

The algorithm as stated before follows the naïve serial implementation of the bilateral filter. The current intensity value is retrieved from the input image. Then a two level for loop over [-w:w], where w is the width of the Gaussian kernel, is used to find the response of all pixels in the window of interest. For each iteration of the loop, the point to sample is found and its intensity is subtracted from the central intensity. This difference is then used to find the Gaussian weight of the range filter over range sigma. The total weight of the bilateral filter is then calculated using the newly found range weight and the already computed spatial weight. Finally, before the next loop iteration a global normalization factor has its value summed with the new weight and similarly a global response has its value summed with the product of the point intensity and the new weight. Once the loops exit, the output is set to the quotient of the response and the normalization factor.

# Results

The machine running the following tests:

* GPU: NVIDIA GeForce GTX 1080
* CPU: Intel Core i5 3.50 GHz
* RAM: 16 GB
* NVIDIA Driver: 376.19

Timing tests were carried out in groups of 5 with each group being of a set image size. Timing is begun before filtering begins and directly after it ends, therefore the GPU timings include memory transfer time to and from the device.

## Image Sizes

The first set of tests compare the image sizes with their execution time. The image sizes are 22.5KP, 160KP, and 2MP. Here, the values of , , and the filter radius are kept constant. As can be seen in **Table 1**, as the number of megapixels increase the execution time increases in serial at a much faster rate than the parallel implementation.

|  |  |  |
| --- | --- | --- |
| Megapixels | Parallel (ms) | Serial (ms) |
| 0.0225 | 210 | 1151 |
| 0.16 | 218 | 6651 |
| 0.388 | 256 | 18041 |
| 2 | 335 | 98351 |

**Table 1:** Average runtime for serial and parallel implementations.

## Image Output Results

Images in color were experimented with to explore the edge preservation in a variety of images. Figure 2 illustrates the effect of a variety of values for both the domain filter and range filter. An experiment with high values of with low values of is shown in Figure 3. This experiment seems to elicit a cartoon-like effect as almost no detail is preserved between the edges.

# Conclusions

In this paper, the concept of bilateral filters was discussed at length. Additionally, the potential for the algorithm to be highly parallelized on a GPU was introduced.

The bilateral filter’s naïve implementation is straightforward however the time of execution is linear to the size of both the image and kernel width. This is not to say highly parallelized implementations are the only avenue, there have been several other, successful, attempts at reducing the rate at which the algorithm resolves [7]. The increasing computational power of GPUs and the parallelizable nature of the algorithm; however, allows the naïve serial algorithm to perform at speeds at, or exceeding, those of more complex methodologies. Tests done with 2MP images shows that a parallel implementation gives roughly 293 times faster performance than its serial counterpart.

# References

|  |  |
| --- | --- |
| [1] | B. Klaus and P. Horn, Robot Vision, Cambridge, MA: MIT Press, 1986. |
| [2] | M. Tabb and N. Ahuja, "Multiscale Image Segmentation by Integrated Edge and Region Detection," *IEEE Transactions on Image Processing,* vol. 6, no. 5, 1997. |
| [3] | R. T. Chin and C. L. Yeh, "Quantitative evaluation of some edge-preserving noise-smoothing techniques," *CVGIP,* no. 23, pp. 67-91, 1983. |
| [4] | P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Trans.,* vol. 12, no. 7, pp. 629-639, 1990. |
| [5] | C. Tomasi and R. Manduchi, Bilateral Filtering for Gray and Color Images, Bombay: IEEE International, 1998. |
| [6] | D. Luebk, "GPU Architecture: Implications & Trends," in *Siggraph*, 2008. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Figure 2:** Figure processed with various range and domain parameter values with a kernel width of 5.

|  |  |
| --- | --- |
| C:\Users\Jacob\AppData\Local\Microsoft\Windows\INetCacheContent.Word\test2.jpg  (a) | (b) |
| (c) | | |

**Figure 3:** The initial image (a) filter with and (b) filter with and .