A Fast Approximation to the Bilateral Filter

Release 0.10

Jordan Woehr

August 26, 2009

University of Calgary,
Calgary AB Canada

Abstract

This document describes an implementation of an alternative bilateral filtering algorithm in ITK.

This class implements the fast bilateral filter algorithm in ITK by taking the input image and organizing it into the required data structure as an ITK Image with dimension one greater than the original image. The `itk::DiscreteGaussianImageFilter` is then used to blur the higher dimensional image. The discrete Gaussian image filter was chosen because the kernel width used should usually be small due to the down-sampling involved when using this technique [1]. As described in the ITK documentation, this class should outperform the recursive Gaussian algorithm due to the small kernel size `itk::DiscreteGaussianImageFilter`. After the Gaussian blur has been completed the data will be interpolated and written to the output image.

By incorporating this class into the ITK framework, this more efficient algorithm will be made available to all ITK users.

Contents

1 Introduction

Currently a bilateral filter class (`itk::BilateralImageFilter`) exists within ITK. This class implements the bilateral filtering algorithm proposed by Tomasi and Manduchi [2]. The bilateral filter smooths an image while not blurring its edges. However, it is computationally expensive and requires many multiplication and add operations per pixel. Paris and Durand proposed a fast approximation for the bilateral filter and have shown that their implementation is more accurate than other accelerations [1], but it has not yet been implemented in ITK. This class implements the algorithm.

2 Comparison to the BilateralImageFilter

The FastBilateralImageFilter class was designed to be as compatible with the existing `itk::BilateralImageFilter` class as possible. Methods in the FastBilateralImageFilter class have methods with the same name as those in the `itk::BilateralImageFilter`. The units of the parameters that these methods takes are also the same to facilitate an easy conversion from one filter to the other if so desired.

Some methods of the `itk::BilateralImageFilter` could not be implemented using this algorithm, or did not make sense to implement.

- Get/Set FilterDimensionality - The dimensionality of the filter can not be set with the expectation of similar results to the bilateral filter because all data in the input still goes through the down-sample/interpolate process.
- Get/Set NumberOfRangeGaussianSamples - This variable is not used in the FastBilateralFilter algorithm.
- Get/Set AutomaticKernelSize - Unnecessary in the FastBilateralFilter algorithm. Due to the down-sampling a small kernel with a uniform size can be used every time.
- Get/Set Radius - Used in the `itk::BilateralImageFilter` when AutomaticaKernelSize is set to false. Unnecessary because there is no AutomaticKernelSize.

3 Usage

Use of the FastBilateralImageFilter is like most other image filters, and identical to the `itk::BilateralImageFilter` with the exception of the methods not implemented.

The FastBilateralImageFilter has two instance variables that can be set, DomainSigma and RangeSigma. The domain sigma specifies the standard deviation of the kernel used to blur the pixels based on their distance from one another. The domain sigma may be different in each dimension and is measured in the units of spacing of the image. The range sigma specifies the standard deviation of the kernel for the intensity. A larger domain sigma will result in a larger neighborhood of pixels being averaged and a larger range sigma will result in a greater number of intensities being averaged together.

4 Limitations

While the fast algorithm is typically faster than the classical algorithm, there are situations where the classical algorithm may perform better. When the sigma values are set very small, especially when less than one, the fast algorithm may take longer to run. This is due to the fact that with small sigma values less down-sampling can be done to the data. Using too small of sigma values also causes more bins to be generated and more memory to be used. On a large data-set this may result in not being able to allocate enough memory for the filter to run.

The fast algorithm may also run longer than the classical algorithm on images in more than two dimensions. This is due to the interpolation step of the algorithm because linear interpolation in too many dimensions becomes much more expensive computationally.

5 Example

```
typedef itk::FastBilateralImageFilter< InputImageType, OutputImageType > FilterType;
typename FilterType::Pointer filter = ImageType::New();
filter->SetDomainSigma(10.0);
filter->SetRangeSigma (10.0);
filter->SetInput(image);
filter->Update();
```

The bilateral filter and fast bilateral filter were run on the image in figure 1 both with a domain sigma of 10 and a range sigma of 10. The classical algorithm ran in 54.58 seconds and produced the result in figure 2 while the fast algorithm ran in 0.65 seconds and produced the result in figure 3. The difference of the two results is shown in figure 4.

References

[1] S. Paris and F. Durand. A fast approximation of the bilateral filter using a signal processing approach. *International Journal of Computer Vision*, 81:24–52, January 2009. (document), 1

[2] J. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the Sixth International Conference on Computer Vision*, page 839, January 1998. 1

Figure 1: The input image for both filters.

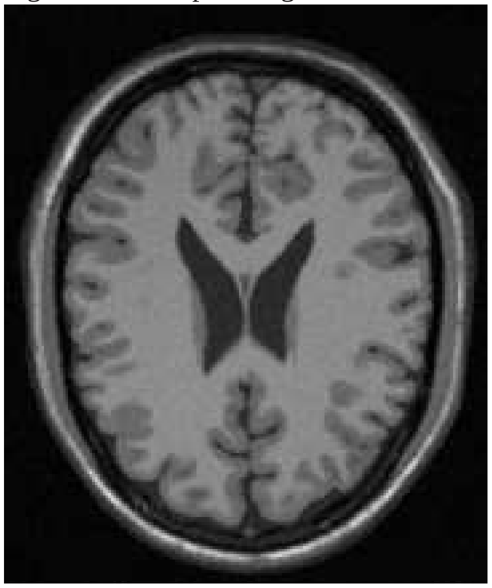

Figure 2: The bilateral filter result.

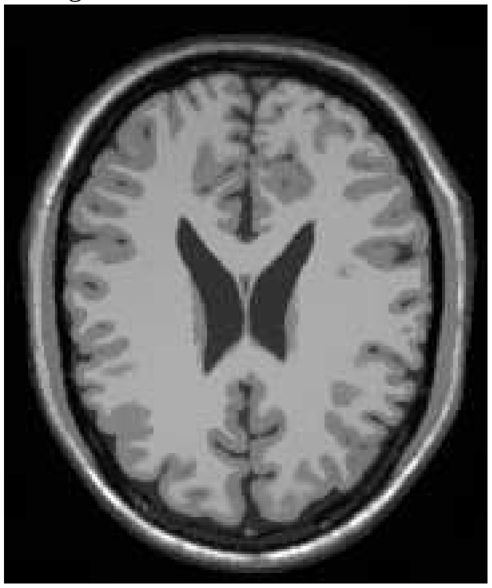

Figure 3: The fast bilateral filter result.

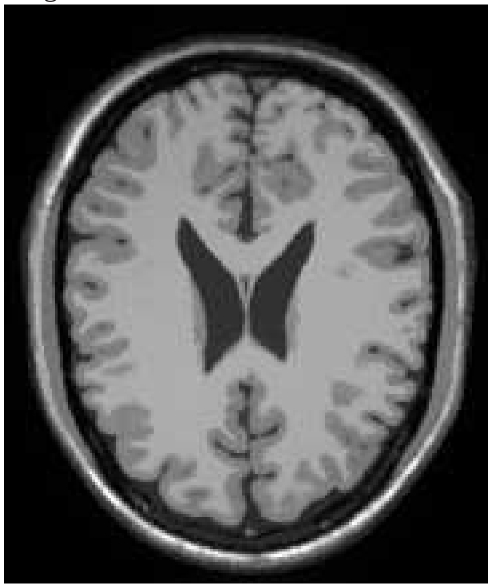

Figure 4: The difference of the two results.