The Fast Bilateral Solver

Numerical imaging - Project report

Guillaume Dalle, MVA 2018-2019

Contents

Code available in the following GitHub repository.

Introduction

Filtering is one of the most fundamental building blocks of image processing. The simplest smoothing method is a Gaussian filter, averaging pixels that are close together in a geometric sense. However, such a filter doesn't differentiate between smooth areas and sharp edges, which become blurred in the process.

The present report is based on "The Fast Bilateral Solver", by Jonathan T. Barron & Ben Poole [3]. The goal of the article is to define and efficiently implement an edge-aware smoother. This technology has countless applications, ranging from simple image "cartooning" or sharpening to automatic colorization or segmentation.

1 Algorithm description

1.1 The model

In this section, we introduce the problem we want to solve. In simple terms, given a reference image and a target image, we want to produce a new result that is chromatically similar to the target image while respecting the structure (including edges) of the reference image.

For instance, in a colorization task:

- the reference image will be the original black and white image (because we want the coloring to respect geometric proximity and stop at edges)
- the target image will be the color channels of a user-marked black and white image (giving us a clue of the real colors in certain locations)

1.1.1 A high-dimensional optimization problem

From now on, we will work with a reference image $\mathbf{p} = (p_i)_{i\in[N]}$ made of pixels with several channels, e.g. in the LUV space (luminance - chrominance). To avoid the trap of purely geometric filtering, we will consider the position $p_i \in \mathbf{R}^5$ of pixel i as being the concatenation of:

- its geometric coordinates $(p_i^x, p_i^y) \in [0, x_{max}] \times [0, y_{max}]$
- its channel intensities, e.g. $(p_i^l, p_i^u, p_i^v) \in [0, 255]^3$ in the LUV case

This combination idea was first introduced by [9], among others.

With this in mind, we can define an edge-aware measure of similarity between two pixels of the reference image, taking into account not only their spatial proximity but also their color proximity. Given a vector $\sigma = (\sigma_{xy}, \sigma_l, \sigma_{uv})$ of standard deviations, we define a gaussian affinity between pixels i and j in the following way:

$$W_{i,j} = \exp\left(-\frac{||(p_i^x, p_i^y) - (p_j^x, p_j^y)||^2}{2\sigma_{xy}^2} - \frac{|p_i^l - p_j^l|^2}{2\sigma_l^2} - \frac{||(p_i^u, p_i^v) - (p_j^u, p_j^v)||^2}{2\sigma_{uv}^2}\right) \tag{1}$$

Suppose now that we want to build another image $\mathbf{x} = (x_i)_{i\in[N]}$ with the same shape as $\mathbf{p}$ but just one channel, that is $x_i \in [0, 255]$ (the multi-channel case is an easy generalization).

We wish this new image to be both:

1. Smooth with respect to the edge-aware similarity between pixels of the reference image $\mathbf{p}$.
2. Close to a target image $\mathbf{t}$ in terms of channel values.

This translates into the following optimization problem:

$$\min_{x \in \mathbf{R}^N} \frac{\lambda}{2} \sum_{1 \leq i,j \leq N} \hat{W}_{i,j}(x_i - x_j)^2 + \sum_{1 \leq i \leq N} c_i(x_i - t_i)^2 \tag{2}$$

Here t_i is the target value of pixel i, c_i expresses the confidence we have in the value of that target. λ defines the balance between both objectives of this minimization. Finally, $\hat{W}$ is a bistochastized version of W, an operation which is meant to improve smoothing performance (for instance by ensuring conservation of the mean).

Unfortunately, this optimization problem is not tractable, since $N = x_{max} \cdot y_{max}$ is usually very large (a few millions). We therefore have to find an approximation that enables us to solve it in reasonable time.

1.1.2 Splat-blur-slice and the bilateral space

This is where the bilateral space comes in. The main idea, described in [2], is to go from a huge set of pixels to a smaller set of "vertices" that approximate them.

Essentially, we can view a LUV image $\mathbf{p}$ as a set of points in a a 5-dimensional space of positions $p = (p^x, p^y, p^l, p^u, p^v)$. To make coordinates homogeneous, we first scale them by their respective standard deviations σ. We then approximate the scaled positions with a periodic lattice, assigning each pixel to one or more of its nearest neighbors.

The lattice points are called *vertices*, and they live in the *bilateral space*. The goal is that the number of vertices V should be much smaller than the number of pixels N.

Now imagine for a moment that we must filter a vector of pixel values $\mathbf{x}$ with a simple matrix multiplication: forget about the optimization problem (2), we just want to compute $W\mathbf{x}$. To approximate that operation, which we deem too expensive to perform exactly, we use the bilateral space through the following three-step procedure (represented graphically in figure 1):

1. *Splat*: Project each pixel value x_i onto the vertices that are closest to p_i
2. *Blur*: Perform a smoothing operation of vertex values y_v in the bilateral space
3. *Slice* : Interpolate the new pixel value $\tilde{x}_i$ from the blurred values $\tilde{y}_v$ of the vertices closest to p_i

In the simplified bilateral grid (adapted by [2] from [4]) which is the one we implemented, the lattice is composed of the integer points $\mathbf{Z}^5$. As for the blurring operation, it simply averages a vertex with its 2 closest neighbors on each of the 5 axes: this boils down to a sum of 5 $(1, 2, 1)$ filters. The example in figure 1 uses this approach.

We indeed observe a huge dimension reduction, because we are actually sampling from a 2-dimensional structure (manifold) inside a 5D lattice. Indeed, only vertices that are close to the original image will actually matter, so that we can discard the rest. In addition, the scaling by σ groups vertices together and contributes to the small number of useful vertices.

A final remark is that contrary to standard gaussian filtering, the smoothing parameters do not affect the blurring operation per se, since the bilateral blur scheme (1, 2, 1) remains constant. Instead, σ defines the scaling of the bilateral grid and therefore the precision of nearest neighbor assignment.

If for instance σ_l is large, then lots of scaled positions p/σ will have a nearest vertex with the same luminance, and therefore the smoothing radius on luminance will be very wide.

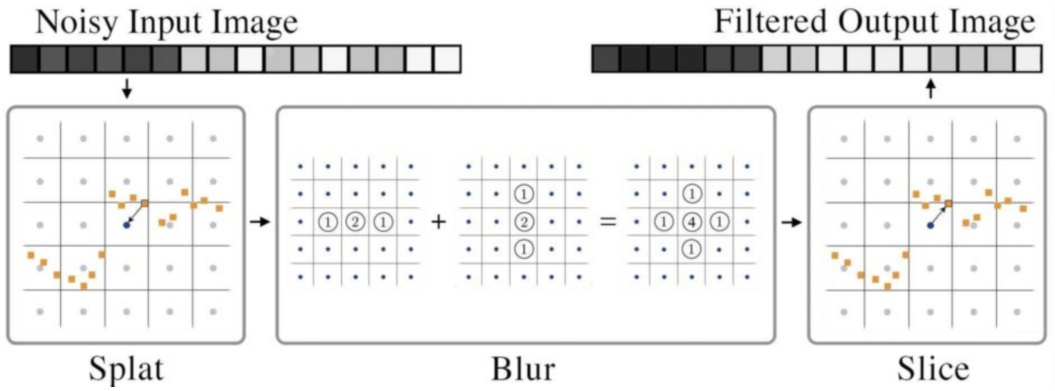

Figure 1: Illustration of the splat-blur-slice procedure on a 1D image (taken from [2])
The orange squares are the positions of the image pixels, the grey circles are all the vertices of the bilateral space. The cells containing no image pixel are associated to vertices we later discard.

1.1.3 A low-dimensional linear system

Although the original articles introducing splat-blur-slice techniques did not see it that way, the example of matrix multiplication suggests that this procedure can be thought of as an approximate matrix factorization of W:

$$W \simeq S^T B S \tag{3}$$

This interpretation was proposed by [2] and it is illustrated in figure 2.

Here S is the matrix of pixel-to-vertex assignment (responsible for splatting and slicing) while B performs the blur in bilateral space. Note that it is possible to efficiently bistochastize this factorization with diagonal matrices D_n and D_m, in order to approximately factor $\hat{W}$ as well:

$$\hat{W} \simeq S^T D_m^{-1} D_n B D_n D_m^{-1} S \quad \text{with} \quad S^T S = D_m \tag{4}$$

Using this factorization idea, we define a bilateral space optimization variable $\mathbf{y} = S\mathbf{x} \in \mathbf{R}^V$ which has a much lower dimension than $\mathbf{x} \in \mathbf{R}^N$. The original optimization problem can then be expressed as a quadratic minimization performed directly in the bilateral space:

$$\min_{\mathbf{y} \in \mathbf{R}^V} \left(\frac{1}{2} \mathbf{y}^T A \mathbf{y} - \mathbf{b}^T y + c \right) \tag{5}$$

Here A, $\mathbf{b}$ and c depend on the splat and blur matrices S and B, as well as the confidence level $\mathbf{c}$, the target image $\mathbf{t}$ and the regularization parameter λ. More precisely:

$$A = \lambda(D_m - D_n B D_n) + \mathrm{diag}(S\mathbf{c}) \tag{6}$$

From this expression, we see that A is a symmetric matrix. Assuming it is also positive, the minimum exists and is unique. In that case, by differentiation, the minimization is equivalent to solving the following linear system:

$$A\mathbf{y} = \mathbf{b} \tag{7}$$

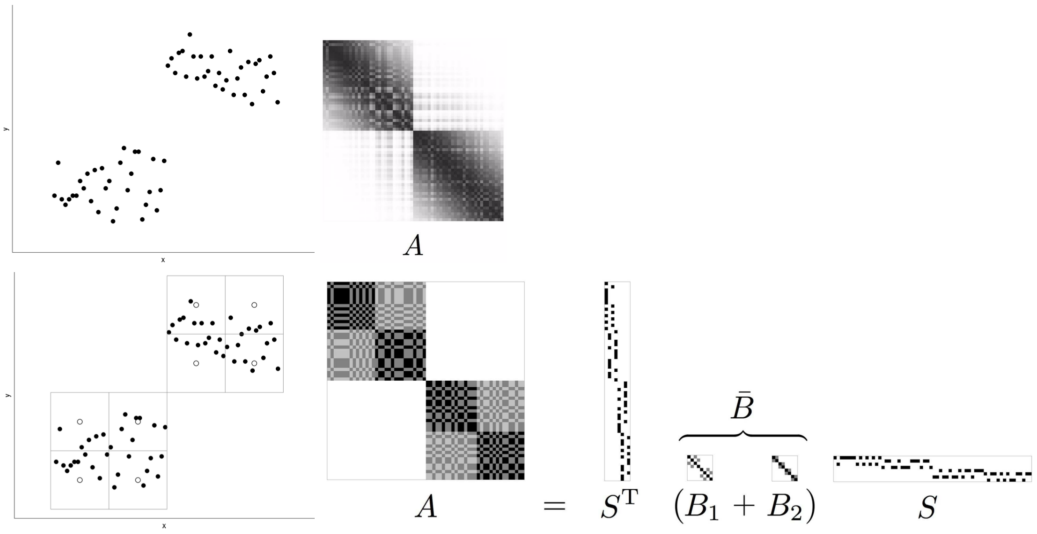

Figure 2: Illustration of the splat-blur-slice factorization on a 1D image (taken from [2])
The black dots are the pixel positions, the white dots are the vertices of the bilateral grid. The x-axis is the spatial position of the pixels/vertices, and the y-axis is the luminance position. The true matrix W (here A) is shown above, and below is its splat-blur-slice approximation. The matrix S links each pixel to one of the 8 bilateral vertices, and the matrix B is the sum of two (1, 2, 1) blurs in the bilateral space: one along the x-axis and one along the y-axis.

1.2 The optimization method

Now we move to describing the actual method used to solve the optimization problem (5).

1.2.1 Conjugate gradient algorithm

A key observation is that the matrix A we work with is very sparse. Therefore, classical methods for solving the linear system (7) will not perform well. Instead, we turn to the conjugate gradient algorithm, thoroughly described in [8].

Basically, this algorithm consists in a series of improvements of standard gradient descent, aimed at finding the solution to (5).

1. Gradient descent with line search: Choose the steepest descent direction, and optimize the step size to maximize decrease.
2. Conjugate directions: Choose the descent directions to be "orthogonal", so that no step can undo what the previous step did: in $\dim(y)$ steps, the optimum is reached.
3. Conjugate gradients: Enumerate the "orthogonal" descent directions in the right order, so that only a small number of steps is needed to achieve target precision.

1.2.2 Multiscale optimization

Remember that $\mathbf{y}$, the vector of vertex values, was obtained by putting together pixels that had a similar position in a 5-dimensional XYLUV space. We can iterate the process, and group vertices that have a similar position in the lattice. Of course, the dimensions are now

homogeneous, so the choice of the reduction factor is arbitrary: for instance, by choosing $\sigma = 2$, at each step of the recursion we put the vertices together by groups of 2^5.

That way, we build a pyramid of ever coarser samplings of the 5D space we work in, each sampling being defined by the splatting matrix S_k, until only one vertex is left at the top. The set of vertex values at each floor k of the pyramid constitutes is then normalized by the mass of each floor (stored in the diagonal matrix Q) to build the pyramid-space vector $\mathbf{z}$:

$$\mathbf{z} = QP(y) = Q \begin{pmatrix} (S_K S_{K-1} \cdots S_1)y \\ (S_{K-1} \cdots S_1)y \\ \vdots \\ S_1 y \\ y \end{pmatrix} \tag{8}$$

The gradient of the loss in pyramid space is then given by:

$$\nabla \ell(\mathbf{z}) = QP(\nabla \ell(QP^T(\mathbf{z}))) \tag{9}$$

Since P is a linear operation, performing the operation in the pyramid space is actually equivalent to using a preconditioner in the conjugate gradient method. Luckily, it is known that the performance of the conjugate gradient method in large-scale problems can be substantially improved by preconditioning, a behaviour which is indeed observed with this hierarchical scheme.

1.2.3 Refinements

Several tweaks are made to the algorithm in order to improve its performance and generalize it. Not all of them were taken into account for our implementation.

Initialization: The initialization can help speed up the convergence if it is well chosen. Here, the heuristic is to select the solution of the optimization problem (5) in the case $\lambda = 0$, which corresponds to

$$y_{init} = S(\mathbf{c} \otimes \mathbf{t})/S(\mathbf{c}) \tag{10}$$

Multiple outputs: If we want to reconstruct a multi-channel image, for instance smooth a *RGB* picture or recover the UV channels $(\mathbf{x}^u, \mathbf{x}^v)$ of a black & white picture, as long as the reference picture stays the same we can simply stack the variables together and solve the least-squares / linear systems separately:

$$AY = B \quad \text{where} \quad X = (\mathbf{x}^u, \mathbf{x}^v) \quad \text{and} \quad B = (\mathbf{b}^u, \mathbf{b}^v) \tag{11}$$

To deal with the case of very wide matrices B, a low-rank approximation is proposed to speed up resolution.

Robustness: While it makes calculations easier, the quadratic loss function $||\mathbf{c} \otimes (\mathbf{x} - \mathbf{t})||^2$ is vulnerable to outliers. We can replace it by a robust loss function $\sum_i \rho(x_i - t_i)$, which is quadratic around zero but asymptotically linear. The price to pay is that we have to run several iterations of the conjugate gradient to solve successive least-squares problems, as part of an Iterative Reweighted Least Squares procedure.

1.2.4 Post-processing with the domain transform

Due to the hard assignment of a pixel to its nearest-neighbor in the bilateral grid, the results of edge-aware smoothing often exhibit visible square artefacts. To counter this effect, a post-processing phase is added using *domain transform*, another kind of very efficient edge-aware filter presented in [6].

The idea behind the domain transform is to define an isotropic one-dimensional transform $\mathbf{R}^D \to \mathbf{R}$, which preserves the distance between the positions (coordinates and channel values) of an image's pixels.

This one-dimensional transform is then used to apply a bilateral filter alternately on the rows and columns of the image, with steadily decreasing spatial standard deviation σ'_s but constant chromatic standard deviation σ'_r (which can also be set channel per channel). The transform being a real-valued increasing function, it allows for an efficient computation of the bilateral filter by reducing it (approximately) to a box filter.

2 Applications and discussion

2.1 Implementation

As part of the present project, the Fast Bilateral Solver was re-implemented from scratch using Python. Given our choice of programming language, a major hurdle was efficiency, which led to the tricks listed below.

Owing to the high dimension of image data, the splat-slice matrix S and the blur matrix B didn't fit into standard `numpy` arrays. Thankfully, most of their coefficients were zeros, and the dedicated library `scipy.sparse` allowed for much faster computations by only considering non-zero entries in any operation.

Another difficulty resided in the constant distinction between "useful vertices" and other vertices. Indeed, the impressive efficiency of the algorithm is based on the fact that only a tiny fraction of the bilateral grid is actually located next to an image pixel.

In terms of implementation however, this required to keep juggling between the coordinates of a vertex, the index of a vertex among the entire grid, and the index of a vertex among the subset of useful vertices. This is one of the reasons why the code for the bilateral grid is so obfuscated.

Regarding the domain transform, which is a post-processing phase we also coded, the major challenge was an speedy version of the box filter (used to smooth rows and columns alternately).

Although the naive implementation with two for-loops (for every pixel of the row, for every other pixel not too far away, add it to the average) has a theoretical runtime of $O(N)$ with a small box size, the low efficiency of for-loops in Python made it unbearably slow. Therefore, the recursive version of the filter suggested in [6] was first implemented, bringing it down to only one for-loop.

But taking advantage of the sparse matrix structure turned out to be even more efficient for large problems: interpreting the recursion equation as a sparse linear system of equations enabled us to get rid of any for loop and parallelize some of the computational burden.

2.2 Image processing applications

We now turn to specific use cases of the Fast Bilateral Solver that we implemented and tested. For each of them, we detail the framework and the parameter choice, then give some illustrations.

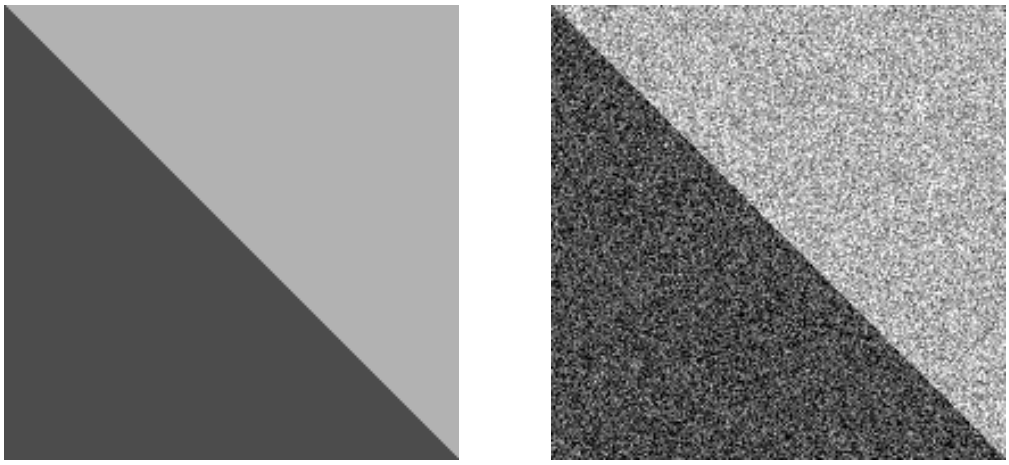

Figure 3: Dummy image with and without noise for edge-aware smoothing

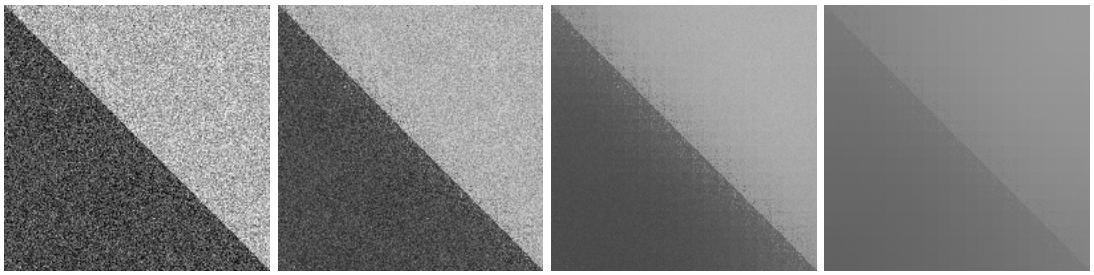

Figure 4: Result of the edge-aware smoothing with $\lambda = 1, 10, 100, 1000$

2.2.1 Edge-aware smoothing

This is the most straightforward application of the algorithm we present. In the edge-aware smoothing setting, the reference image $\mathbf{x}$ (which dictates the similarity between pixels in the regularization) is the same as the target image $\mathbf{t}$ (which dictates the starting point to imitate).

The effect of the filter is to smooth the image within regions without edges, while respecting sharp borders between areas of different color or luminance.

We tested the method using a dummy black and white image with a sharp diagonal edge separating two regions of different average luminance (0.3 and 0.7). The added noise is Gaussian with standard deviation 0.15. This image is depicted in figure 3. The images are not blurred, simply small : 200×200 pixels.

The default filter parameters are $\lambda = 100$, $\sigma_{xy} = 10$ and $\sigma_l = 50$. For each parameter, deviations around these standard settings are tested and commented.

First, the influence of the regularization parameter λ is studied in figure 4. For small values of λ, the regularization term doesn't have any effect, and the target image is simply copied. As λ grows larger, the regularization kicks in, but when it gets too large the target image becomes forgotten and all that matters is regularity, leading to a near-uniform result.

Then, the influence of the spatial filter radius σ_{xy} is analyzed in figure 5. For small values of σ_{xy}, the spatial smoothing is limited, allowing several outliers to exist when they should

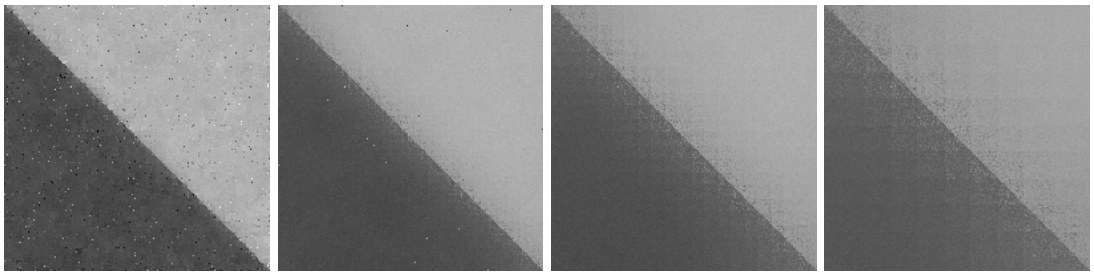

Figure 5: Result of the edge-aware smoothing with $\sigma_{xy} = 2, 5, 10, 20$

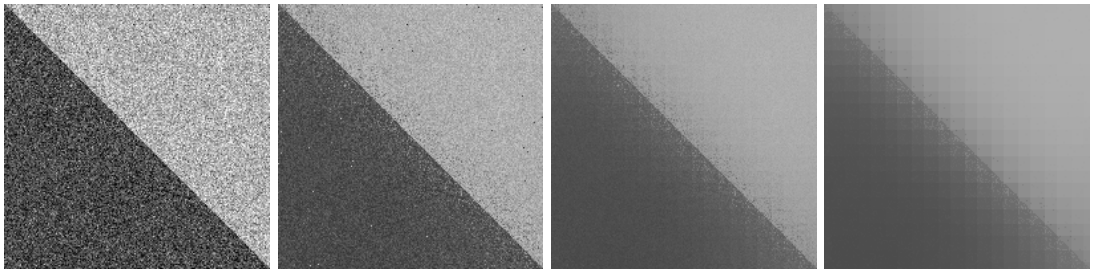

Figure 6: Result of the edge-aware smoothing with $\sigma_l = 5, 20, 50, 200$

have been averaged with their neighbors. As σ_{xy} grows larger, the interior of the regions becomes smoother but square artefacts appear. They are the result of the hard nearest-neighbor assignment: the groups of pixels whose associated vertices have the same spatial coordinates are large, and the transition between them is sudden.

Finally, the influence of the luminance filter radius σ_l is demonstrated in figure 6. For small values of σ_l, the filter is very edge-aware (but not very meaningful since the default value of σ_{xy} is also quite small). As σ_l grows larger, the nearest-neighbor assignment is more and more controlled by the spatial filter radius σ_{xy}, which means the artefacts reappear, and the edge becomes less sharp.

2.2.2 Cartooning and sharpening

Besides analyzing the behaviour of the algorithm, there are other applications to edge-aware smoothing. Even when there is no need for denoising, it can be used to give images a cartoon aspect, or sharpen their details.

Below is an example of cartooning and sharpening applied to a landscape photograph (see figure 7) taken in one of France's most beautiful regions: the Auvergne [1]. The algorithm parameters were $\lambda = 100$, $\sigma_{xy} = 10$, $\sigma_{rgb} = 50$.

Figure 2.2.2 presents the results of smoothing, and figure 9 the results of subsequent sharpening. For enhancing the details, we use a superposition of the original image I and the difference between I and its smoothed version $S(I)$: the result $I + [S(I) - I]$ is an image where the details of the original are more salient.

[1] Picture taken from `https://www.pleinevie.fr/loisirs/tourisme/france/volcans-d-auvergne-cap-sur-des-puys-a-la-chaine-21316`

Figure 7: Original picture taken from the Puy de Pariou

Figure 8: Result of edge-aware smoothing

Figure 9: Result of sharpening based on edge-aware smoothing

If we zoom in on figure 2.2.2, we can see the square artefacts induced by the choice of $\sigma_{xy} = 10$. For instance, in the sky, supposedly homogeneous regions are tessellated with squares of size 10*10 pixels. This is what the domain transform post-processing was introduced to solve however it comes at the cost of a new filtering step which further eliminated details. Figure 10 shows the result of this last step, with parameters $\sigma'_s = 20$ and $\sigma'_r = 50$.

2.2.3 Depth superresolution

Another use of the Fast Bilateral Solver is the improvement of low quality depth maps with the help of a color image. While color images taken by standard cameras often have little noise and high resolution, depth maps can be very noisy and have a much lower resolution.

The algorithm we present enables us to improve the depth map by drawing inspiration from the structure of the reference color image.

Approximately following the test procedure laid out by [5], we start with a pair (image, disparity map) taken from the Middlebury dataset [7]. From the disparity map we deduce a depth map, which is then downsampled by a factor of $f = 8$, combined with a Gaussian noise (whose variance is proportional to the local disparity) and eventually reupsampled using interpolation. These two starting points are displayed in figure 11.

The result of the procedure can be observed in figure 12. While it is globally convincing, some things are still not right. Regarding the "cloudy" aspect, it is probably caused by the high amplitude of the Gaussian noise, leading to irregularities even in the supposedly uniform regions.

However, the fact that the supposed depth of the chess board (left-hand side of the scene) varies so wildly is much more problematic. This is due to the denoising procedure working from a color image, and trying to abide by its edges. While the edges of the chess squares are indeed very sharp in the color reference, they do not correspond to limits between objects on separate

Figure 10: Application of domain transform on smoothed image to remove square artefacts

depth levels. This causes errors in the depth reconstruction performed by the algorithm.

2.2.4 Colorization

Finally, as mentioned in the beginning of this report, the Fast Bilateral Solver works very well for colorization. The target image is a black and white picture color-marked by the user in certain areas where the color is (approximately known). The algorithm's purpose is then to expand and conciliate these color stains, while respecting the structure of the reference black and white image.

We worked on a black and white picture of the Hindenburg's crash in 1937, taken by Murray Becker. The color markings were done in a very rough way, aiming at an esthetically pleasing result rather than a historically correct one: they are reproduced in figure 13.

As for the algorithm parameters, we used the following settings: $\lambda = 5$, $\sigma_{xy} = 10$, $\sigma_l = 50$, $\sigma'_s = 20$, $\sigma'_r = 50$.

The colorized result can be seen in figure 14. We notice that the color blue has spread across the whole night sky, changing its shade according to the luminance structure of the black and white reference. The yellow and red colors have extended their areas too, but always stopping at sharp edges of the original image (e.g. the limits of the aircraft or the smoke cloud).

2.3 Discussion

2.3.1 Many advantages

The Fast Bilateral Solver presents several advantages over previous methods, which make it interesting in a number of situations. Here are a few:

Figure 11: High-resolution color reference and noisy low-resolution depth map

Figure 12: High-resolution denoised depth map

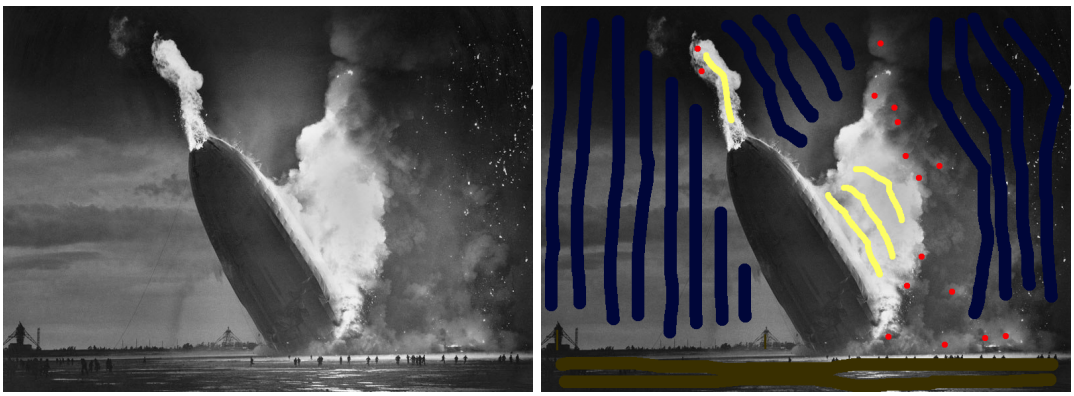

Figure 13: Black & white original and color-marked versions of the Hindenburg disaster

Figure 14: Automatically colorized version of the Hindenburg disaster

- As its name suggests, it is very fast. Compared to other baseline techniques on several applications (including those mentioned above), its runtime is diminished significantly without compromising the accuracy.

  On the one hand, using the simplified bilateral grid makes the computation of the splat-blur-slice decomposition $W \simeq S^T B S$ near instantaneous even for large images. On the other hand, the bilateral space in which we work has a very low dimension compared to the original pixel space. Finally, the formulation of the problem as a least-squares optimization allows the use of very efficient minimization routines such as the conjugate gradient descent, further optimized by a hierarchical preconditioner.

- For the same reasons of dimensionality reduction, this algorithm requires low memory.

- It is rapidly implementable in any programming language: the only crucial requirement is an efficient implementation of sparse matrices.

- It has few hyperparameters, all of which have a clear intuitive meaning: λ for regularization and σ for the filter radius are the most important ones.

- It is general: the idea of splat-blur-slice factorization and the optimization problem per se can be used accross a whole range of image processing problems.

- It is easily integrated into a deep learning pipeline, because the backpropagation of the error on $\mathbf{x}$ onto the error on $(\mathbf{t}, \mathbf{c})$ can be computed efficiently. This is partly due to A depending on $\mathbf{t}$ and $\mathbf{c}$ only through its diagonal coefficients.

2.3.2 Some drawbacks

However, a few aspects still make the whole procedure perfectible, or weren't very clear in the article. We will try to list a few negative points below:

- The complexity of the algorithm is dependent upon the value of the filter radius σ. The smaller the values in σ, the more bilateral vertices there will be, which slows down the whole procedure.
- The simplified bilateral grid, while easy to implement, comes with caveats. The hard nearest-neighbor assignment causes square artefacts which must be removed through post-processing or smaller values of σ_{xy}.

  This can be solved through the use of more sophisticated grids, such as the permutohedral lattice of [1], which splats a pixel onto several vertices. However, the implementation becomes substantially more complicated as a result, and the runtime of the algorithm increases due to harder nearest-neighbor computations and a less sparse S matrix.
- The systematic use of domain transform post-processing in [3] makes it hard to separate between the merits of both procedures, which were both designed to solve the same problem. While in a few settings, we indeed observed that the domain transform allowed for a reduction of the square artefacts induced by the simplified bilateral grid, maybe changing the parameters of the grid would have helped as well. If the domain transform is always necessary, it may mean the initial algorithm is not sufficient.

  In addition, although our PYTHON code was heavily profiled and optimized, the domain transform remained (by far) the slowest part of the procedure, which is why (most of the time) we dispensed with it to focus on the Fast Bilateral Solver per se.

Conclusion

The Fast Bilateral Solver is a very general edge-aware filter than can be applied to a wide variety of image-related tasks. Its very general optimization problem, along with its ease of implementation and small computational cost, make it a valuable addition to any image-processing toolbox, including deep-learning frameworks.

References

[1] Andrew Adams, Jongmin Baek, and Myers Abraham Davis. Fast High-Dimensional Filtering Using the Permutohedral Lattice. *Computer Graphics Forum*, 29(2):753–762, May 2010.

[2] Jonathan T. Barron, Andrew Adams, YiChang Shih, and Carlos Hernandez. Fast bilateral-space stereo for synthetic defocus. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4466–4474, Boston, MA, USA, June 2015. IEEE.

[3] Jonathan T. Barron and Ben Poole. The Fast Bilateral Solver. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, pages 617–632. Springer International Publishing, 2016.

[4] Jiawen Chen, Sylvain Paris, and Frédo Durand. Real-time Edge-aware Image Processing with the Bilateral Grid. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[5] David Ferstl, Christian Reinbacher, Rene Ranftl, Matthias Ruether, and Horst Bischof. Image Guided Depth Upsampling Using Anisotropic Total Generalized Variation. In *2013 IEEE International Conference on Computer Vision*, pages 993–1000, Sydney, Australia, December 2013. IEEE.

[6] Eduardo S. L. Gastal and Manuel M. Oliveira. Domain Transform for Edge-aware Image and Video Processing. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH '11, pages 69:1–69:12, New York, NY, USA, 2011. ACM.

[7] Daniel Scharstein, Heiko Hirschmüller, York Kitajima, Greg Krathwohl, Nera Nešić, Xi Wang, and Porter Westling. High-Resolution Stereo Datasets with Subpixel-Accurate Ground Truth. In Xiaoyi Jiang, Joachim Hornegger, and Reinhard Koch, editors, *Pattern Recognition*, volume 8753, pages 31–42. Springer International Publishing, Cham, 2014.

[8] Jonathan R Shewchuk. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

[9] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 839–846, Bombay, India, 1998. Narosa Publishing House.