Roussos–Maragos Tensor-Driven Diffusion for Image Interpolation

Pascal Getreuer

CMLA, ENS Cachan (pascal.getreuer@cmla.ens-cachan.fr)

Abstract

Roussos and Maragos proposed a method for image interpolation in "Reversible interpolation of vectorial images by an anisotropic diffusion-projection PDE." An earlier version was also published in conference paper (Roussos and Maragos, "Vector-Valued Image Interpolation by an Anisotropic Diffusion-Projection PDE," 2007).

Given a discretely sampled image v, the method finds an image u such that $v_n = (h * u)(n)$, for all $n \in \mathbb{Z}^2$, where h is the (assumed known) point spread function and $*$ denotes convolution.

The method is inspired by tensor-driven diffusion works of Tschumperlé and Weickert. Roussos and Maragos propose interpolation by evolving a diffusion equation to steady state, $\partial_t u = P_0\big(\mathrm{div}(T\nabla u)\big)$ where T is a tensor determined from image structure tensor and the diffusion is orthogonally projected to agree with the observed data. This diffusion is based on the general anisotropic diffusion model proposed by Weickert. The method can be applied to grayscale, color, or general vector-valued images.

Source Code

ANSI C source code to produce the same results as the demo is accessible on the article web page http://dx.doi.org/10.5201/ipol.2011.g_rmdi. Future software releases and updates will be posted at http://dev.ipol.im/~getreuer/code.

Keywords:

1 Initial Interpolation

First, an initial interpolation u_0 is computed by Fourier zero-padding with deconvolution,

$$\hat{u}_0(\xi) = \begin{cases} \dfrac{1}{\hat{h}(\xi)} \displaystyle\sum_{n \in \mathbb{Z}^2} v_n \mathrm{e}^{-i2\pi n\xi}, & \text{if } \xi \in [-\frac{1}{2}, +\frac{1}{2}] \times [-\frac{1}{2}, +\frac{1}{2}], \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where $\hat{h}$ is the two-dimensional Fourier transform of the point spread function h,

$$\hat{h}(\xi) := \int_{\mathbb{R}^2} h(x)\mathrm{e}^{-i2\pi(x \cdot \xi)} \, dx. \tag{2}$$

For the division, it is assumed that $\hat{h}(\xi) \neq 0$ in $[-\frac{1}{2}, +\frac{1}{2}] \times [-\frac{1}{2}, +\frac{1}{2}]$.

This interpolation is used as an initialization for the tensor-driven diffusion because it satisfies $v_n = (h * u_0)(n)$ for all n. It is well-known that Fourier interpolation produces significant ringing artifacts, so the goal of the diffusion is to remove the ringing.

2 Structure Tensor

As introduced by Bigün and Granlund [1] and Förstner and Gülch [2], the image structure tensor is

$$J(\nabla u) := \begin{pmatrix} \partial_{x_1} u \\ \partial_{x_2} u \end{pmatrix} \begin{pmatrix} \partial_{x_1} u & \partial_{x_2} u \end{pmatrix}. \tag{3}$$

At each point in the image, $J(\nabla u)$ is a 2×2 symmetric matrix. Roussos and Maragos [8, 9] use the smoothed structure tensor,

$$J_\rho(\nabla u_\sigma) = G_\rho * J\big(\nabla(G_\sigma * u)\big), \tag{4}$$

where G_σ and G_ρ are Gaussians with standard deviations σ and ρ, which control the amount of pre- and post-smoothing. The post-smoothing convolution with G_ρ is applied separately to each component of the tensor. The image gradient ∇u can be discretized using centered differences. Alternatively, the gradient may be incorporated into the pre-smoothing convolution as

$$\nabla(G_\sigma * u) = (\nabla G_\sigma) * u, \tag{5}$$

and then approximated with discrete convolutions.

For a color image, the smoothed structure tensor is computed as the sum of the smoothed structure tensors for each channel.

Next, at every point in the image, the eigenvectors and eigenvalues of the the 2×2 matrix $J_\rho(\nabla u_\sigma)$ are computed. Since the matrices are guaranteed to be symmetric and real, the eigenvalues are real and the eigenvectors are real and orthogonal. Following Tschumperlé [6, 7], Roussos and Maragos construct the tensor T according to this eigensystem,

$$\begin{aligned} T\big(J_\rho(\nabla u_\sigma)\big) = {}& \big(1 + \tfrac{1}{K^2}(\lambda_1 + \lambda_2)\big)^{-1/2} J\big(\tfrac{w_1}{|w_1|}\big) \\ & + \big(1 + \tfrac{1}{K^2}(\lambda_1 + \lambda_2)\big)^{-1} J\big(\tfrac{w_2}{|w_2|}\big), \end{aligned} \tag{6}$$

where K is a parameter, $\lambda_1 \leq \lambda_2$ are the eigenvalues and w_1 and w_2 are the corresponding eigenvectors. The tensor is recomputed every n time steps of the diffusion.

To compute the eigensystem of a matrix $\left(\begin{smallmatrix} a & b \\ b & c \end{smallmatrix}\right)$, the eigenvalues are

$$\lambda_1 = \tfrac{1}{2}(a+c) - \sqrt{\tfrac{1}{4}(a+c)^2 - (ac - b^2)}, \tag{7}$$

$$\lambda_2 = \tfrac{1}{2}(a+c) + \sqrt{\tfrac{1}{4}(a+c)^2 - (ac - b^2)}, \tag{8}$$

and provided $b \neq 0$, the eigenvector corresponding to λ_1 is

$$w_1 = \begin{pmatrix} b \\ \lambda_1 - a \end{pmatrix}. \tag{9}$$

The other eigenvector w_2 is found as the orthogonal complement.

3 Tensor-Driven Diffusion

The interpolation is diffused according to the tensor as

$$\partial_t u = \operatorname{div}\!\left(\left(\begin{smallmatrix} a & b \\ b & c \end{smallmatrix}\right)\nabla u\right), \tag{10}$$

where $\left(\begin{smallmatrix} a & b \\ b & c \end{smallmatrix}\right)$ denotes the components of the tensor field T from the previous section. To implement the diffusion, one approach is proposed in Weickert's book [3]. More recently, Weickert and Scharr [4] proposed a method that is faster yet also simpler: define the derivative approximations

$$F_x u = \frac{1}{32}\begin{pmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{pmatrix} * u, \quad F_y u = \frac{1}{32}\begin{pmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{pmatrix} * u, \tag{11}$$

then the diffusion is implemented explicitly using these approximations as

$$u^{\text{next}} = u + dt\left(F_x(aF_xu + bF_yu) + F_y(bF_xu + cF_yu)\right). \tag{12}$$

The filters F_x and F_y have optimal approximate rotation invariance among all 3×3 linear filters. Weickert and Scharr claim that rotation invariance is crucial to avoid blur artifacts in the diffusion. Since F_x and F_y are 3×3 linear filters, the composition used in the formula above is effectively a 5×5 scheme.

Note that F_x and F_y are separable filters and have only two different filter coefficient values, so they can be implemented efficiently:

$$\begin{aligned} \tilde{u}_x(i,j) &= u(i+1,j) - u(i-1,j), \\ (F_xu)(i,j) &= \tfrac{10}{32}\tilde{u}_x(i,j) + \tfrac{3}{32}\left(\tilde{u}_x(i,j+1) + \tilde{u}_x(i,j-1)\right), \end{aligned} \tag{13}$$

and similarly for F_y.

For a color image, the diffusion scheme is applied independently to each channel. The channels are nevertheless coupled since they are all guided by the same tensor field, which is computed jointly over the channels.

4 Projection onto the Solution Set

The input is modeled as sampling the underlying continuous-domain image by $v = \operatorname{sample}(h * u)$, so the solution is required to belong to the affine set

$$\mathcal{W}_v = \{u : v(n) = (h * u)(n), \forall n \in \mathbb{Z}^2\}. \tag{14}$$

To impose this requirement, the diffusion is orthogonally projected onto $\mathcal{W}_v$,

$$\partial_t u = P_0\big(\mathrm{div}(T\nabla u)\big). \tag{15}$$

where P_0 denotes orthogonal projection onto $\mathcal{W}_0$ and the initial interpolation u_0 computed with Fourier zero-padding is in $\mathcal{W}_v$. Then u is in $\mathcal{W}_v$ for all $t > 0$.

For numerical implementation, the solution is projected onto $\mathcal{W}_v$ after every n time steps and after the final time step. The projection is expensive to compute compared to the other steps of the algorithm, so it is helpful to reduce the number of projections that need to be computed.

We use the following normalization of the Fourier transform,

$$\hat{f}(\xi) = \int_{\mathbb{R}^2} f(x)\mathrm{e}^{-2\pi i x \cdot \xi}\, dx. \tag{16}$$

Before the main iteration, a function φ is precomputed,

$$\hat{\varphi}(\xi) = \Big(\sum_{k\in\mathbb{Z}^2} |\hat{h}(\xi - k)|^2\Big)^{-1/2} \hat{h}(\xi). \tag{17}$$

The projection onto $\mathcal{W}_v$ can then be implemented as

$$\big(P_v(u)\big)\hat{}(\xi) = \hat{u}(\xi) - \overline{\hat{\varphi}(\xi)} \sum_{n\in\mathbb{Z}^2} \hat{\varphi}(\xi - n)\big(\hat{u}(\xi - n) - \hat{u}_0(\xi - n)\big). \tag{18}$$

4.1 Derivation of P_v

Let H denote the coarsening operation and H^* its adjoint,

$$(Hu)(n) := (h * u)(n), \quad (H^*v)(x) = \sum_{m\in\mathbb{Z}^2} h(m - x)v_m. \tag{19}$$

The orthogonal projection onto $\mathcal{W}_0 = \ker(H)^\perp$ is

$$P_0(u) = u - H^*(HH^*)^{-1}Hu. \tag{20}$$

Let q be the discrete autocorrelation

$$q_n = \int_{\mathbb{R}^2} h(x)h(x - n)\, dx, \quad n \in \mathbb{Z}^2, \tag{21}$$

so that $q * u = H^*Hu$. Then the inverse $(HH^*)^{-1}$ can be expressed as

$$\big((HH^*)^{-1}v\big)_m = \sum_{n\in\mathbb{Z}^2} (q)^{-1}_{m-n}v_n, \tag{22}$$

where $(q)^{-1}$ is the discrete convolution inverse such that $\big((q)^{-1} * q\big) = 1$ for $n = 0$ and zero otherwise. Therefore, the projection onto $\mathcal{W}_0$ is

$$
\begin{aligned}
P_0(u)(x) &= u(x) - H^*(HH^*)^{-1}Hu(x) \\
&= u(x) - \sum_{m\in\mathbb{Z}^2} h(m - x) \sum_{n\in\mathbb{Z}^2} (q)^{-1}_{m-n}(h * u)(n), \\
\big(P_0(u)\big)\hat{}(\xi) &= \hat{u}(\xi) - \overline{\hat{h}(\xi)}\frac{\sum_{n\in\mathbb{Z}^2} \hat{h}(\xi - n)\hat{u}(\xi - n)}{\sum_{n\in\mathbb{Z}^2} |\hat{h}(\xi - n)|^2} \\
&= \hat{u}(\xi) - \overline{\hat{\varphi}(\xi)} \sum_{n\in\mathbb{Z}^2} \hat{\varphi}(\xi - n)\hat{u}(\xi - n),
\end{aligned}
\tag{23}
$$

where the Fourier transform of the discrete image $\big((HH^*)^{-1}Hu\big)_n$ follows from the Poisson summation formula,

$$\sum_{n\in\mathbb{Z}^2} f(n)\mathrm{e}^{-2\pi in\cdot\xi} = \sum_{n\in\mathbb{Z}^2} \hat{f}(\xi - n). \tag{24}$$

The projection onto $\mathcal{W}_v$ follows from P_0 as

$$P_v(u) = u_0 + P_0(u - u_0),$$

$$\big(P_v(u)\big)\hat{\ }(\xi) = \hat{u}_0(\xi) + \big(\hat{u}(\xi) - \hat{u}_0(\xi)\big) - \overline{\hat{\varphi}(\xi)} \sum_{n\in\mathbb{Z}^2} \hat{\varphi}(\xi - n)\big(\hat{u}(\xi - n) - \hat{u}_0(\xi - n)\big)$$

$$= \hat{u}(\xi) - \overline{\hat{\varphi}(\xi)} \sum_{n\in\mathbb{Z}^2} \hat{\varphi}(\xi - n)\big(\hat{u}(\xi - n) - \hat{u}_0(\xi - n)\big). \tag{25}$$

5 Algorithm

Here we summarize the algorithm. First, the initial interpolation u_0 is computed by Fourier zero-padding with deconvolution and the projection function $\hat{\varphi}$ is precomputed. The algorithm then iterates the main projection-diffusion loop:

1. Compute the tensor T.

2. Perform n explicit time steps of $\partial_t u = \mathrm{div}(T\nabla u)$ using the approximation

$$\mathrm{div}\big(\big(\begin{smallmatrix} a & b \\ b & c \end{smallmatrix}\big)\nabla u\big) \approx F_x(aF_x u + bF_y u) + F_y(bF_x u + cF_y u).$$

3. Orthogonally project the solution onto $\mathcal{W}_v$ with

$$\big(P_v(u)\big)\hat{\ }(\xi) = \hat{u}(\xi) - \overline{\hat{\varphi}(\xi)} \sum_{n\in\mathbb{Z}^2} \hat{\varphi}(\xi - n)\big(\hat{u}(\xi - n) - \hat{u}_0(\xi - n)\big).$$

The loop stops when either $\|u_{\mathrm{cur}} - u_{\mathrm{prev}}\|_2 \leq tol$ or when a maximum number of iterations N is reached.

For operations involving Fourier transforms, boundary artifacts are avoided by half-sample symmetric extension of the image. In the implementation, the image is extended by $(5 \times scalefactor)$ pixels on each of the four borders of the image.

6 Examples

The following examples demonstrate the method for factor-4 interpolation. The parameters used are point spread function h is a Gaussian with standard deviation 0.5 in units of input pixels, $K = 1$, $dt = 2$, $tol = 0.1$, $n = 5$ iterations, and $N = 50$ maximum iterations.

In practice, the standard deviation of the Gaussian point spread function h must be tuned to approximate how the input image was sampled. The examples are shown using standard deviation 0.5, which provides moderate antialiasing. In the online demo, the default value is 0.35, which is a reasonable model of the blurriness of typical images.

Input Image (86 × 79)
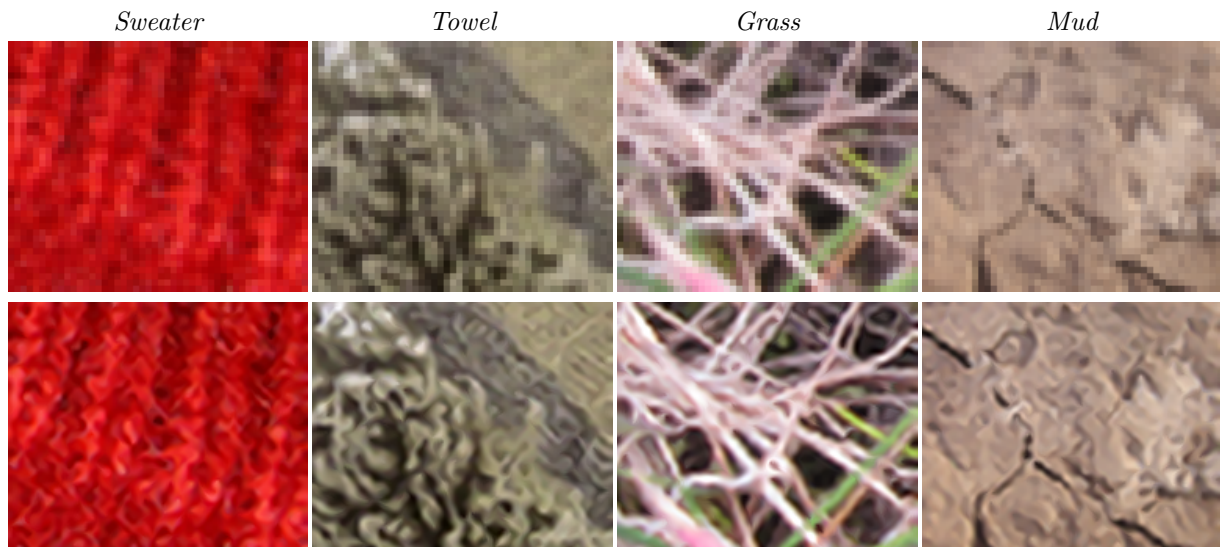
Tensor-Driven Diffusion, CPU time 5.216s

Input Image (86 × 79)

Tensor-Driven Diffusion, CPU time 2.149s

The next example demonstrates the method's good performance on oriented textures. The top row shows the input images and the bottom row shows the corresponding interpolations created using the method.

Sweater   *Towel*   *Grass*   *Mud*

Here we compare the interpolation with several existing methods. A high resolution image is coarsened by convolving with h and the downsampling by factor 4 to create the input image. This

image is then interpolated with each of the methods and compared with the original using the PSNR and MSSIM metrics. The time to compute the interpolation is also shown.

Original Image (332 × 300)

Input Image (83 × 75)

Bicubic

PSNR 24.36, MSSIM 0.6311, CPU time 0.012s

Fractal Zooming [11]

PSNR 24.50, MSSIM 0.6317

Fourier Zero-Padding with Deconvolution

PSNR 25.70, MSSIM 0.7104, CPU time 0.049s

TV Minimization [5]

PSNR 25.87, MSSIM 0.7181, CPU Time 2.72s

Contour Stencils [10]

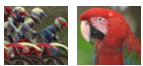PSNR 25.99, MSSIM 0.7256, CPU Time 0.077s

Tensor-Driven Diffusion

PSNR 26.01, MSSIM 0.7303, CPU Time 2.23s

Acknowledgments

Image Credits

Steve Kelly, Kodak Image Suite, images 5 and 23 (http://r0k.us/graphics/kodak/)

Pascal Getreuer

John D. Willson, USGS Amphibian Research and Monitoring Initiative, Public Domain (`http://armi.usgs.gov/gallery/detail.php?id=323`)

References

[1] J. Bigün and G. Granlund. "Optimal orientation detection of linear symmetry." In *IEEE First International Conference on Computer Vision*, London, Great Britain, pp. 433–438, 1987.

[2] W. Förstner and E. Gülch. "A fast operator for detection and precise location of distinct points, corners, and centers of circular features." In *Proceedings of the Intercomission Conference on Fast Processing of Photogrammetric Data*, pp. 281–305, 1987.

[3] J. Weickert. *Anisotropic Diffusion in Image Processing*. ECMI Series, Teubner-Verlag, Stuttgart, Germany, 1998. ISBN:3519026066, `http://www.mia.uni-saarland.de/weickert/book.html`

[4] J. Weickert and H. Scharr. "A Scheme for Coherence-Enhancing Diffusion Filtering with Optimized Rotation Invariance." *Journal of Visual Communication and Image Representation*, vol. 13, no. 1–2, pp. 103–118, 2002. `http://dx.doi.org/10.1006/jvci.2001.0495`

[5] F. Malgouyres and F. Guichard. "Edge direction preserving image zooming: A mathematical and numerical analysis." *SIAM Journal on Numerical Analysis*, 39, pp. 1–37, 2002. `http://dx.doi.org/10.1137/S0036142999362286`

[6] D. Tschumperlé. "PDE's based regularization of multivalued images and applications." Ph.D. Thesis, University of Nice-Sophia Antipolis, 2002. `http://tel.archives-ouvertes.fr/tel-00002396`

[7] D. Tschumperlé and R. Deriche. "Vector-Valued Image Regularization with PDE's: A Common Framework for Different Applications." *IEEE Pattern Analysis and Machine Intelligence*, vol. 27, no. 4, pp. 506–517, 2005. `http://doi.ieeecomputersociety.org/10.1109/TPAMI.2005.87`

[8] A. Roussos and P. Maragos. "Vector-Valued Image Interpolation by an Anisotropic Diffusion-Projection PDE." In *Lecture Notes in Computer Science, SSVM proceedings*, vol. 4485, pp. 104–115, 2007. `http://dx.doi.org/10.1007/978-3-540-72823-8_10`

[9] A. Roussos and P. Maragos. "Reversible interpolation of vectorial images by an anisotropic diffusion-projection PDE." *International Journal of Computer Vision*, 2008. `http://dx.doi.org/10.1007/s11263-008-0132-x`

[10] P. Getreuer. "Image Interpolation with Geometric Contour Stencils." *Image Processing On Line*, 2011. DOI: `http://dx.doi.org/10.5201/ipol.2011.g_igcs`

[11] onOne software. "Genuine Fractals." `http://www.ononesoftware.com`