Volumetric Nonlinear Anisotropic Diffusion on GPUs

Andreas Schwarzkopf[1], Thomas Kalbe[1], Chandrajit Bajaj[2],
Arjan Kuijper[1,3], and Michael Goesele[1]

[1] Technische Universität Darmstadt, Germany
[2] ICES-CVC University of Texas at Austin, USA
[3] Fraunhofer IGD, Darmstadt, Germany

Abstract. We present an efficient implementation of volumetric nonlinear anisotropic image diffusion on modern programmable graphics processing units (GPUs). We avoid the computational bottleneck of a time consuming eigenvalue decomposition in $\mathbb{R}^3$. Instead, we use a projection of the Hessian matrix along the surface normal onto the tangent plane of the local isodensity surface and solve for the remaining two tangent space eigenvectors. We derive closed formulas to achieve this resulting in efficient GPU code. We show that our most complex volumetric nonlinear anisotropic diffusion gains a speed up of more than 600 compared to a CPU solution.

1 Introduction and Motivation

Diffusion equations smooth out noise effectively and provide a scale space representation [1–5] of the image, when time is considered as a natural, continuous scale space parameter. They are well known in the field of image processing and have been subject to many enhancements during the last decades.

These equations are widely used for 2D images processing, see e.g. [3, 6] for an introduction. Recent publications apply this diffusion for smoothing of normal maps [7], and fairing of surfaces and functions on surfaces and meshes [8, 9], to mention only some possibilities. Anisotropic diffusion of whole volume images or general meshes [10, 11] and smoothing vector valued volume images [12] are also common tasks arising in medical applications.

As this diffusion requires to solve second order partial differential equations (PDEs) numerically for a rapidly increasing amount of discretized data, it is a perfect application for modern graphic cards, which can easily handle large data sets. The current GPU SIMD architecture allows to solve each iteration in a few milliseconds due to massively parallel processing. This holds for equations that lead to an efficient parallelization. However, this is at least difficult for most interesting, non-linear, PDEs due to the local structure in each voxel that determines in which direction smoothing can be performed.

1.1 Contribution

As main contribution we show how one can obtain the so-called local structure frame for volumetric data sets easily. This leads to nearly unconditional code, performing

A.M. Bruckstein et al. (Eds.): SSVM 2011, LNCS 6667, pp. 62–73, 2012.

extremely well on GPUs. For this purpose, we build on a technique described by Hadwiger et al. [13]. Anisotropic nonlinear diffusion on symmetric multiprocessor (SMP) clusters for volumetric data was discussed in [14], showing a maximum speedup of 20 on SMP clusters with up to 30 processors. In our GPU approach, we do not need to slice the volume and distribute it across a platform, as all shading processors of modern GPUs can access the same memory. We therefore achieve speedups of up to 640 on a comparably cheap GPU.

Volumetric anisotropic diffusion on GPUs using shader programs in the standard graphics pipeline has been discussed in, e.g, the works by Jeong et al. [15], Zhao [16] or Beyer et al. [17]. In contrast to this, our approach is based on NVidia's *CUDA* which is better suited for GPGPU (general purpose GPU) algorithms like volumetric diffusion. Intermediate values, such as the Hessian, are recomputed on-the-fly in each iteration and are stored temporarily in per-thread local memory. Therefore, larger data sets can be processed on the GPU. Further, to our knowledge, we are the first to compute the local structure frame in 2D in the context of volumetric diffusion. This significantly simplifies the algorithm while the results of the diffusion are very good, see Fig. 1, right, and 4.

2 Prerequisites

The linear homogeneous diffusion equation removes noise from images by solving the heat equation, a second order parabolic PDE. Initial and boundary conditions are required to find a particular solution. A general diffusion equation can then be defined as follows:

$$\frac{\partial}{\partial t}\Phi(\boldsymbol{x},t) = \operatorname{div}\left(D\nabla\Phi(\boldsymbol{x},t)\right) \text{ for } \boldsymbol{x}\in\Omega, t>0, \tag{1}$$

$$\Phi(\boldsymbol{x},0) = \Phi_0(\boldsymbol{x}) \text{ for } \boldsymbol{x}\in\Omega, \tag{2}$$

$$\frac{\partial}{\partial \boldsymbol{n}}\Phi(\boldsymbol{x},0) = 0 \text{ for } \boldsymbol{x}\in\partial\Omega. \tag{3}$$

Here, Φ denotes the noisy image function defined on a region Ω of the Euclidean space. D is a function, which determines the diffusion speed through the medium. D is constant (usually 1 or $1/2$) in the linear case. The initial condition (2) initializes the function at time $t = 0$ with the original noisy image Φ_0. The boundary values are defined in (3) by their derivative in normal direction $\boldsymbol{n}$ to the border of the considered volume: Since the directional derivative is assumed to be 0, no flow through the boundary $\partial\Omega$ is induced.

Solving the heat equation for $D = 1$ at time $t = {}^1\!/_2\,\sigma^2$ equates to convolving the image function with a Gaussian of size σ (see [1, 3, 18]). Thus, this diffusion equation has exactly the same smoothing characteristics as the well known Gaussian filter. In particular boundaries blur out fast and therefore edge information gets lost quickly, see Fig. 1, left.

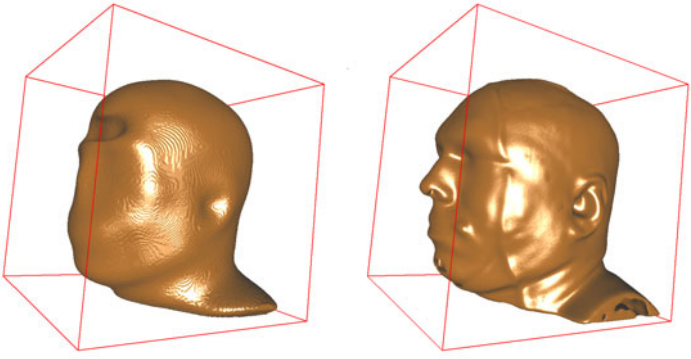

Fig. 1. 100 iterations of diffusion with a time step $\Delta t = 0.05$. *Left*: Homogeneous diffusion. The frontal sinus destroys the surface structure of the forehead and small scaled details (e.g. lips, nose, ears) are lost. *Right*: Nonlinear anisotropic diffusion (here: edge enhancing diffusion) steered by a diffusion tensor based on local structure preserves fine scaled features.

2.1 Inhomogeneous Diffusion

In the context of image processing, the heat equation was modified significantly by Perona and Malik [19] by replacing D in Eq. (1) with an edge detector, a monotonically decreasing non-negative real function g, which attenuates the induced flow close to edges and therefore effectively prevents edges from being washed out:

$$\frac{\partial}{\partial t}\Phi = \operatorname{div}\left(g(|\nabla\Phi|)\nabla\Phi\right), \text{ for } \boldsymbol{x} \in \Omega, t > 0. \tag{4}$$

Perona and Malik proposed the diffusivity functions $g(\nabla\Phi) = e^{-(|\nabla\Phi|/\lambda)^2}$ and $g(\nabla\Phi) = \frac{1}{1+(|\nabla\Phi|/\lambda)^2}$. They designated this diffusion anisotropic, but it is only locally adapting and still isotropic, as it is steered by a scalar diffusion coefficient. Weickert calls this locally adapting diffusion *inhomogeneous* [3]. Inhomogeneous diffusion is able to preserve edges over a long period of time, but its smoothing capabilities close to edges are rather poor.

2.2 Nonlinear Anisotropic Diffusion

Weickert introduced a new nonlinear anisotropic diffusion, using a tensor for D in Eq. (1). This allows for anisotropic adjustment of the diffusion flow [3, 6, 20]. The diffusion tensor D aligns the diffusion flow along the surface structure and its exact definition is mainly dependent on the desired results of the smoothing process.

Edge enhancing diffusion (EED) attenuates diffusion flow normal to the edge or surface but promotes flow along the edge or parallel to the surface, see Fig. 1, right. Furthermore, *coherence enhancing diffusion* (CED) trys to steer diffusion along line-like structures and is able to reconnect interrupted lines [6]. In a hybrid approach, joining EED and CED to locally adapting diffusion, one is able to enhance edges, smooth out noise and to connect broken lines.

In all cases the definition of a useful diffusion tensor involves the construction of a local structure frame: One needs to find a transformation which aligns the coordinate system orthogonally to the surface of the submanifold. Let V be such a coordinate transformation, aligning the third axis normal to the surface, then we can define the EED tensor D in three dimensions as follows:

$$D = VD^*V^T = V \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & g(|\nabla \Phi|) \end{pmatrix} V^T. \tag{5}$$

Matrix D transformed to the new basis V is a diagonal matrix D^*, as the diffusion flow is aligned perfectly along the principal directions of the surface structure. The crucial point when designing anisotropic diffusion is the efficient construction of this frame V. The traditional way is to obtain this basis by analyzing the structure tensor, defined as the outer product of the gradient $\nabla \Phi$ with itself. In the next section, we will present a method to find such a frame by efficiently analyzing the Hessian, which holds structural information, as it describes the change of the surface normal.

3 Surface Structure and the Hessian

When defining a diffusion tensor, it is utterly important to find a basis V whose axes are aligned exactly along the principal curvature directions of the surface. Theoretically, this was also possible by eigen-decomposition of the structure tensor in 3D space. As the structure tensor is a real symmetric matrix, the eigenvalues are real and the eigenvectors are existent. But the characteristic polynomial of a 3×3 matrix has degree 3 and therefore it is rather time consuming to solve for the roots.

On the other hand, the eigenvalues of a 2×2 matrix are computed easily by evaluating only a few closed formulas. Especially for machine code executed on modern GPUs, this is of advantage as the single execution paths are not divergent (not branching) and parallel execution on the hardware is achieved ideally. In the following, we will show how to obtain the structure frame V by evaluating closed formulas only.

3.1 Tangent Space Projection of the Hessian

The following considerations are aimed at finding a basis transformation $V : \mathbb{R}^3 \mapsto \mathbb{R}^3$ with as few computations as possible and which will describe a coordinate system normal to the tangent plane of the isosurface at a given point. Despite that, the remaining two basis vectors of V spanning the tangent plane should be aligned with the orthogonal principal curvature directions.

Assuming that the inner region of a volume consists of higher density volumes, we define the *surface normal* by the gradient $\nabla \Phi = \left[\frac{\partial \Phi}{\partial x} \; \frac{\partial \Phi}{\partial y} \; \frac{\partial \Phi}{\partial z} \right]^T$ as $\boldsymbol{n} = -\nabla \Phi / |\nabla \Phi|$. Since $\nabla \Phi$ points towards the direction of the greatest density ascent inside the volume, $\boldsymbol{n}$ lies inside the linear span of the gradient. Therefore it points into the direction the surface moves when the iso value is increased. We can choose $\boldsymbol{n}(\boldsymbol{x})$ to be the first vector of our frame V.

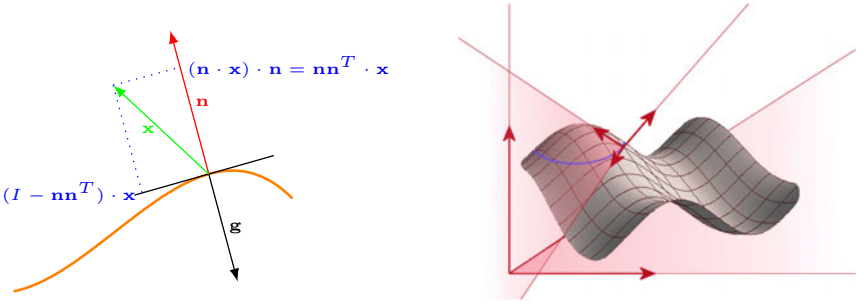

Fig. 2. *Left:* Projecting a point $\boldsymbol{x}$ into the span of the normal $\boldsymbol{n}$ and its complement. *Right:* The local surface frame.

The *curvature* of a surface is defined as the ratio between change of surface normal and change of position, which is described by the gradient of $\boldsymbol{n}$: If we move in an infinitesimal close area around the point $\boldsymbol{x}$, the normal will change according to the surface.

In [13] and [21] one finds methods on how to characterize the curvature of a surface based on gradient informations and, moreover, how to obtain the principal curvature directions. The derivative of the normal field $\nabla \boldsymbol{n}^T$ at some point $\boldsymbol{x}$ contains curvature information of the surface. Note that $\nabla \boldsymbol{n}^T$ is a 3×3 matrix. According to Kindlmann et al. [21] it holds that

$$\nabla \boldsymbol{n}^T = -\frac{1}{|\nabla \Phi|}(I - \boldsymbol{n}\boldsymbol{n}^T)H. \tag{6}$$

Here, I denotes the 3×3 identity matrix, and $H = \nabla(\nabla \Phi)^T$ is the Hessian containing all combinations of partial second order derivatives of the image Φ.

While the gradient describes the amount of change of Φ, the Hessian describes the amount of change of the gradient, that is the amount of change of the surface normal in an infinitesimal close region to a given point $\boldsymbol{x}$. This amount of change of the gradient can be decomposed into two components, namely the changes along the gradient direction and changes in the tangent space. Only the latter is required for isosurface curvature computation.

To perform the *tangent space projection*, we proceed as follows: We may omit the scaling factor $|\nabla \Phi|^{-1}$ in Eq. (6) and concentrate on the remaining term. It is easy to see that $(\boldsymbol{n}\boldsymbol{n}^T)\boldsymbol{x} = (\boldsymbol{n}\boldsymbol{x})\boldsymbol{n}$, and the operator $(\boldsymbol{n}\boldsymbol{n}^T)$ projects any point $\boldsymbol{x} \in \Omega$ onto the linear span of the normal. Therefore we are able to define a linear map

$$P = (I - \boldsymbol{n}\boldsymbol{n}^T) = \left(I - \frac{\nabla \Phi (\nabla \Phi)^T}{|\nabla \Phi|^2}\right). \tag{7}$$

which projects any given point $\boldsymbol{x}$ into the complement of the linear span of $\boldsymbol{n}$, which is the iso surface (see Fig. 2).

Projection P extracts the gradients change of direction from the Hessian inside the tangent space. By using P we define the *shape operator*

$$S = P^T \frac{H}{|\nabla \Phi|} P. \tag{8}$$

Since S is symmetric, solving the characteristic polynomial gives us three real roots and associated orthogonal eigenvectors. Still, the computational overhead of a full eigendecomposition of a 3×3 matrix is rather high, especially as one eigenvector, the normal $\boldsymbol{n}$, is already known. The remaining eigenvectors in the tangent plane are the principle curvature directions with corresponding eigenvalues $\lambda_{1,2}$ which amount to the principle curvatures.

According to Hadwiger et al. [13], we can solve for the eigenvalues directly in 2D tangent space without explicitly computing S. The transformation of S into any arbitrary orthogonal basis $(\boldsymbol{u}, \boldsymbol{v})$ of the tangent space is defined as

$$S' = \begin{pmatrix} s_{11} & s_{12} \\ s_{12} & s_{22} \end{pmatrix} = (\boldsymbol{u}, \boldsymbol{v})^T \frac{H}{|\nabla \Phi|} (\boldsymbol{u}, \boldsymbol{v}). \tag{9}$$

For finding an arbitrary orthogonal basis $\boldsymbol{u}$, $\boldsymbol{v}$ in the tangent plane, we may proceed as follows: We choose the canonical unit vector $\boldsymbol{e}_1 = (1,0,0)^T$ assuming that $\boldsymbol{e}_1 \nparallel \boldsymbol{n}$ holds and compute the cross product $\boldsymbol{u} = \boldsymbol{e}_1 \times \boldsymbol{n}$. In case $\boldsymbol{u} = \boldsymbol{0}$ we compute the cross product again, now using the second unit vector, $\boldsymbol{e}_2 = (0,1,0)^T$. $\boldsymbol{u}$ is now normal to $\boldsymbol{n}$ and therefore it must be part of the tangent plane. We finish the new basis by adding $\boldsymbol{v} = \boldsymbol{u} \times \boldsymbol{n}$.

By using Eq. (9) we are now able to compute the eigenvalues $\lambda_{1,2}$ of S' by solving the characteristic polynomial

$$\det(S' - \lambda I) = \begin{vmatrix} s_{11} - \lambda & s_{12} \\ s_{12} & s_{22} - \lambda \end{vmatrix} = 0$$

$$\Rightarrow \lambda_{1,2} = \frac{\text{trace}(S')}{2} \pm \sqrt{\frac{\text{trace}(S')^2}{4} - \det(S')}. \tag{10}$$

From the eigenvalues $\lambda_{1,2}$ we compute the corresponding eigenvectors. The appropriate formula in [13] is incorrect and can be found in the correct formulation in [22, p. 96]. The eigenvectors $\boldsymbol{w}^*_{1,2}$ are computed with reference to the basis $(\boldsymbol{u}, \boldsymbol{v})$ at first, and afterwards they are transformed back into 3D space:

$$\begin{aligned} \boldsymbol{w}^*_1 = \begin{pmatrix} w^*_{1u} \\ w^*_{1v} \end{pmatrix} &= \begin{cases} \begin{pmatrix} \lambda_1 - s_{22} \\ s_{12} \end{pmatrix}, & \text{for } s_{12} \neq 0 \\ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, & \text{for } s_{12} = 0 \end{cases} \\ \boldsymbol{w}^*_2 = \begin{pmatrix} w^*_{2u} \\ w^*_{2v} \end{pmatrix} &= \begin{cases} \begin{pmatrix} \lambda_2 - s_{22} \\ s_{12} \end{pmatrix}, & \text{for } s_{12} \neq 0 \\ \begin{pmatrix} 0 \\ 1 \end{pmatrix}, & \text{for } s_{12} = 0 \end{cases} \end{aligned} \tag{11}$$

The transformation of the 2D eigenvectors into object space is accomplished by extending the tangent space basis with $\boldsymbol{n}$ to 3D and a retransformation into the original orientation by means of $V = \{\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{n}\}$:

$$\boldsymbol{w}_i = \begin{pmatrix} u_x & v_x & n_x \\ u_y & v_y & n_y \\ u_z & v_z & n_z \end{pmatrix} \begin{pmatrix} w^*_{iu} \\ w^*_{iv} \\ 0 \end{pmatrix} = \begin{pmatrix} u_x w^*_{iu} + v_x w^*_{iv} \\ u_y w^*_{iu} + v_y w^*_{iv} \\ u_z w^*_{iu} + v_z w^*_{iv} \end{pmatrix}. \tag{12}$$

3.2 Algorithm: Retrieving the Diffusion Tensor

Building on the results of the previous sections, we now depict a compact and easy to implement algorithm to define the anisotropic diffusion tensor. The core of our algorithm is a 2×2 eigen-decomposition of the Hessian projected into the tangent space of the iso surface, computed with simple, closed formulas. The algorithm can be outlined as follows:

1. Calculate the gradient $\nabla\Phi$ and the normal of the isosurface $\boldsymbol{n} = -{}^{\nabla\Phi}/_{|\nabla\Phi|}$
2. Build the Hessian $H = \nabla(\nabla\Phi)^T$ (see Sect. 4)
3. Complete $\boldsymbol{n}$ with any arbitrary $\boldsymbol{u}$ and $\boldsymbol{v}$ to an orthonormal basis, whose $\boldsymbol{u}, \boldsymbol{v}$ plane is tangential to the isosurface
4. Using Eq. (9), project H into the tangent plane to obtain the 2×2 matrix S'
5. Using Eq. (10) we can calculate the eigenvalues $\lambda_{1,2}$
6. Now, using Eq. (11), we obtain the corresponding eigenvectors $\boldsymbol{w}^*_{1,2}$ with respect to the $(\boldsymbol{u}, \boldsymbol{v})$ basis
7. W.l.o.g. we might – if this was necessary for the definition of our diffusion tensor – reorder the eigenvalues and eigenvectors: $\lambda_1 < \lambda_2$
8. Transform the 2D eigenvectors back to object space using Eq. (12), receiving the 3D eigenvectors $\boldsymbol{w}_1, \boldsymbol{w}_2$
9. Set $V = (\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{n})$ and $V^{-1} = V^T$
10. Define $D = V \cdot \mathrm{diag}(1, 1, g(\nabla\Phi)) \cdot V^{-1}$

The last step defines the EED diffusion tensor, smoothing along the isosurface and attenuating the diffusion flow normal to the edge. It is further possible to define other diffusion tensors with different properties upon the frame V.

4 Implementation

Our prototype was implemented using C for CUDA which allows for high-parallel computations on NVidia GPUs. As divergent program execution – arising from conditional code which leads to branching – and sequential calculations of the GPU multiprocessors could eliminate speed advantages it is important to find a reduction of the dimension: As one surface frame basis vector – the normal – is known, we can search for the remaining ones in the hyper plane.

The volume data was stored as a 3D texture on the GPU. Coalesced memory access is not possible when processing volumetric data, so the best speedup was achieved by using cached texture memory. Clamping the textures automatically keeps track of all

the border values: As gradients at the borders equal zero, no flow will be induced and we avoid conditional code which would slow down the program.

Besides, we can access values between the grid centers and request the hardware to do trilinear interpolation. This gives us a slight speedup for discretizing the Hessian as explained below. Finally, and most important, texture memory is cached, which compensates for the uncoalesced access and results in faster access to neighboring voxels. Performing multiple iterations can be achieved by synchronizing all threads and copying the output back to the texture memory. Copying memory within the device is a fast solution to circumvent the read only issue of texture memory. Each CUDA thread processes one voxel at a time. The code was straight forward developed from the discretized formulation of the anisotropic diffusion.

4.1 Discretization

In the following $\widehat{\Phi}$ denotes the discretized image function $\Phi : \Omega \subset \mathbb{R}^3 \mapsto \mathbb{R}$. Each grid point is associated with a value $\widehat{\Phi}_{x,y,z}$. Neighboring voxels are labeled $\widehat{\Phi}_{x_-} = \widehat{\Phi}_{x-1,y,z}$, likewise $\widehat{\Phi}_{x_+}$, $\widehat{\Phi}_{y_-}$, $\widehat{\Phi}_{y_+}$, $\widehat{\Phi}_{z_-}$ and $\widehat{\Phi}_{z_+}$.

The isotropic grid structure provides a natural spatial discretizing scheme for central differences. For temporal discretization we use forward differences. For nonlinear anisotropic diffusion we obtain the following discretization:

$$\begin{aligned}\widehat{\Phi}(t+\Delta t) &\approx \widehat{\Phi}(t) + \Delta t \cdot \operatorname{div}(D(\nabla\Phi)\nabla\Phi) \\ &= \widehat{\Phi}(t) + \Delta t \cdot \left(\frac{\partial}{\partial x}\left(d_{11}\frac{\partial\widehat{\Phi}}{\partial x} + d_{12}\frac{\partial\widehat{\Phi}}{\partial y} + d_{13}\frac{\partial\widehat{\Phi}}{\partial z}\right) + \right. \\ &\qquad \frac{\partial}{\partial y}\left(d_{12}\frac{\partial\widehat{\Phi}}{\partial x} + d_{22}\frac{\partial\widehat{\Phi}}{\partial y} + d_{23}\frac{\partial\widehat{\Phi}}{\partial z}\right) + \\ &\qquad \left. \frac{\partial}{\partial z}\left(d_{13}\frac{\partial\widehat{\Phi}}{\partial x} + d_{23}\frac{\partial\widehat{\Phi}}{\partial y} + d_{33}\frac{\partial\widehat{\Phi}}{\partial z}\right)\right) \end{aligned} \tag{13}$$

The entries d_{ij} represent the components of the diffusion tensor D. We discretize Eq. (13) over an isotropic grid with central differences.

The diffusion tensor of EED as defined in step 10 of the algorithm outlined in Sect. 3.2 yields a symmetric matrix D consisting of the eigenvectors $\boldsymbol{w_1}$ and $\boldsymbol{w_2}$ of the Hessian projected along $\boldsymbol{n}$ to the tangent plane of the isosurface:

$$D = \begin{pmatrix} d_{11} & d_{12} & d_{13} \\ d_{12} & d_{22} & d_{23} \\ d_{13} & d_{23} & d_{33} \end{pmatrix}, \tag{14}$$

with $d_{11} = w_{1x}w_{1x} + w_{2x}w_{2x} + n_x n_x g(\nabla\Phi)$ and accordingly for the remaining cases. The eigenvalues from Eq. (10) and eigenvectors from Eq. (11) are obtained directly in tangent space and are transformed back to object space, see Eq. (12).

The step size Δt needs to be small enough in order to guarantee numerical stability. Following [14], $\Delta t < 0.5/N_d$, with N_d the dimension of the problem, i.e. 3. We used the conservative value $\Delta t = 0.05$.

5 Results

The results of our GPU implementation as well as a CPU implementation for a volume with size 512^3 voxels are given in Table 1. Unless specified otherwise, the timings were taken on a system consisting of an Intel Xeon E5430 CPU (2.66 GHz) and a NVidia GeForce GTX 480 with 480 shader cores. The CPU variants were ported from the GPU code in a straightforward way without any further optimizations.

The table shows the timings, separated into transfer times of the volume to the GPU (obviously not applicable to CPU) and the times needed for one iteration of the code. The speedups are given for one iteration alone. Since diffusion equations typically need many iterations we neglect the transfer times and only take the iteration timings into account. We achieve a speedup of about 170 to 320 for the homogeneous and inhomogeneous diffusion. For the nonlinear anisotropic diffusion (EED) the results are even better: Five iterations require 10 minutes CPU time compared to 1 second on the GPU. As other anisotropic diffusion equations have even higher arithmetic intensity one can expect them to be even faster compared to their CPU variants.

The runtimes of single iterations are also visualized in Fig. 3. As the filters are independent of the data, execution times are proportional to volume sizes and particularly the fraction of CPU to GPU times is constant, so one can easily extrapolate to other data sizes. Modern GPUs provide up to 4 GB memory, so it is possible to process data sets with up to 800^3 voxels on the GPU.

Table 1. Timings and speedups of one iteration step of the discretized PDE, Eq. (13), with a data set of size 512^3 voxels (CPU: Intel Xeon E5430; GPU: NVIDIA GTX 480)

		Time ([ms])		Speedup
		MemCpy	Iteration	Iteration
Linear homogeneous	cpu	0	1 917	1
	gpu	256	11	173
Nonlinear inhomogeneous (Perona-Malik)	cpu	0	33 499	1
	gpu	281	105	320
Nonlinear anisotropic (EED)	cpu	0	118 717	1
	gpu	276	185	641

Overall it is important to acquire high-quality first and second order derivatives for the gradient as well as the Hessian. Especially at the beginning of the diffusion process significant noise components may disturb the discrete computation of the derivatives enormously. To initialize the diffusion process optimally, usually some sort of (pre-) filtering is applied to the image. for instance using a box, a Gaussian, or a median filter. Bajaj et al. propose to use bilateral filtering, since it removes noise while preserving edge or curvature information, which is important for constructing the diffusion tensor [18]. The bilateral filter can be seen as an expansion to Gaussian filtering by applying an additional edge term [23–25]. In our GPU implementation we achieve a speedup

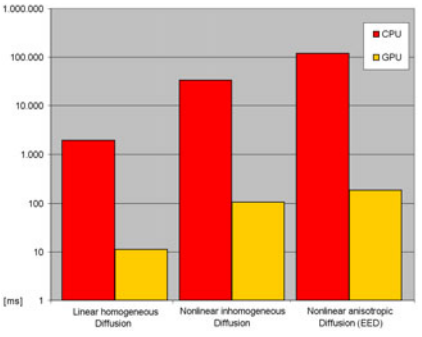

Fig. 3. Runtimes for one iteration of the presented kernels from table 1 on a logarithmic scale (CPU: Intel Xeon E5430; GPU: NVIDIA GTX 480)

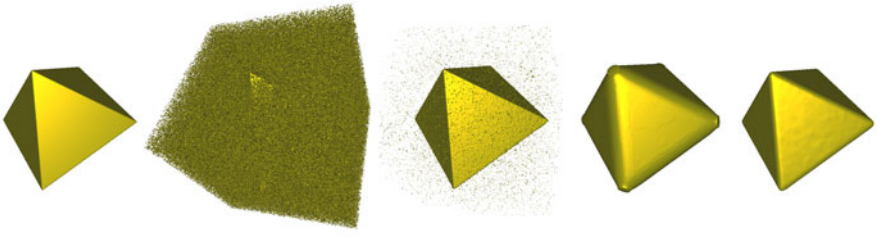

Fig. 4. Restoring the functional data $\Phi(x, y, z) = |x| + |y| + |z|$ (256^3 voxels). *From Left to Right*: Ground truth, Gaussian noise added ($\approx 20\%$ voxels affected, variance 30%), after bilateral prefiltering, after 100 iterations ($\Delta t = 0.05$) of homogeneous and edge enhancing diffusion, respectively, of the prefiltered data set.

of 800 for the bilateral filter. Simpler filters also benefit from massively parallelization, albeit less due to their simplicity. The box filter has a speed up factor of 43, the Gaussian filter one of 151, and the median filter is 76 times faster. All filters took around 1 to 2 seconds on the 512^3 data set.

In Fig. 4 an example of a 3D model endowed with a significant amount of noise is given. Here applying the bilateral filter before the diffusion definitely makes sense.

6 Conclusions

Using the example of EED, we have shown that volumetric nonlinear anisotropic diffusion can be mapped *efficiently* onto the GPU. As efficient GPU code should avoid branching if possible, we derived *closed formulas for the 3D eigenvalue analysis* of the shape operator that allows for reducing the problem from 3D object space onto 2D tangent space: We have presented closed formulas for creating a structure frame along the three principal curvature directions. Building on that, we defined the diffusion tensor for nonlinear anisotropic diffusion and achieved over 600 times the speed compared to a conventional CPU solution. Among the different possible pre-filters for very noisy

images we have seen that the bilateral filter is a promising candidate for being processed on the GPU, achieving 830 times the speed of our CPU solution.

As the technical development of GPUs is rapidly progressing and available memory expands, increasingly larger data sets can be processed directly on the GPU. Apart from scalar volume data, one could also process vector data sets on the GPU, as they arise, for example in DW-MRI (diffusion-weighted magnetic resonance imaging). A starting point for this could be 3D-RGBA-textures, representing 4D vectors. Another question concerns automatic parameter detection. Presumably it was necessary to construct and analyze complete or statistically representative image and gradient histograms, which could be done directly on the GPU. Also, one could examine how CED or hybrid diffusion performs on GPUs, as the arithmetic intensity is higher. Building on successful (pre-)filtering and the diffusion process one could try to deal with segmentation as well, in order to present a seamless GPU solution.

References

1. Koenderink, J.J.: The structure of images. Biological Cybernetics 50, 363–370 (1984)
2. Lindeberg, T.: Scale-Space Theory in Computer Vision. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Dordrecht (1994)
3. Weickert, J.: A review of nonlinear diffusion filtering. In: ter Haar Romeny, B.M., Florack, L.M.J., Viergever, M.A. (eds.) Scale-Space 1997. LNCS, vol. 1252, pp. 1–28. Springer, Heidelberg (1997)
4. Lindeberg, T.: Generalized Gaussian scale-space axiomatics comprising linear scale-space, affine scale-space and spatio-temporal scale-space. Journal of Mathematical Imaging and Vision, 1–46 (2010)
5. Kuijper, A.: Geometrical PDEs based on second order derivatives of gauge coordinates in image processing. Image and Vision Computing 27(8), 1023–1034 (2009)
6. Weickert, J.: Coherence enhancing diffusion filtering. International Journal of Computer Vision 31, 111–127 (1999)
7. Tasdizen, T., Whitaker, R., Burchard, P., Osher, S.: Geometric surface smoothing via anisotropic diffusion of normals. In: Proc. VIS 2002, pp. 125–132 (2002)
8. Bajaj, C.L., Xu, G.: Adaptive surfaces fairing by geometric diffusion. In: Symp. CAGD, pp. 731–737 (2001)
9. Bajaj, C.L., Xu, G.: Anisotropic diffusion of surfaces and functions on surfaces. ACM Trans. Graph. 22(1), 4–32 (2003)
10. Lipnikov, K., Shashkov, M., Svyatskiy, D., Vassilevski, Y.: Monotone finite volume schemes for diffusion equations on unstructured triangular and shape-regular polygonal meshes. Journal of Computational Physics 227(1), 492–512 (2007)
11. Agelas, L., Masson, R.: Convergence of the finite volume MPFA O scheme for heterogeneous anisotropic diffusion problems on general meshes. Comptes Rendus Mathematique 346(17-18), 1007–1012 (2008)
12. Zhang, X., Chen, W., Qian, L., Ye, H.: Affine invariant non-linear anisotropic diffusion smoothing strategy for vector-valued images. Imaging Science Journal 58(3), 119–124 (2010)
13. Hadwiger, M., Sigg, C., Scharsach, H., Bühler, K., Gross, M.H.: Real-time ray-casting and advanced shading of discrete isosurfaces. Comp. Graph. Forum 24(3), 303–312 (2005)

14. Tabik, S., Garzon, E., Garcia, I., Fernandez, J.: Implementation of anisotropic nonlinear diffusion for filtering 3D images in structural biology on SMP clusters. In: Proc. Int. Conf. Parallel Computing: Current & Future Issues of High-End Computing, ParCo., vol. 33, pp. 727–734 (2005)
15. Interactive 3D seismic fault detection on the graphics hardware. In: Proc. Volume Graphics (2006)
16. Zhao, Y.: Lattice Boltzman based PDE solver on the GPU. The Visual Computer 24(5), 323–333 (2008)
17. Beyer, J., Langer, C., Fritz, L., Hadwiger, M., Wolfsberger, S., Bühler, K.: Interactive diffusion-based smoothing and segmentation of volumetric datasets on graphics hardware. Methods Inf. Med. 46(3), 270–274
18. Bajaj, C.L., Wu, Q., Xu, G.: Level set based volumetric anisotropic diffusion for 3D image denoising. ICES TR03-10, UTexas, Austin USA (2003)
19. Perona, P., Malik, J.: Scale-space and edge detection using anisotropic diffusion. IEEE Trans. Pattern Analysis and Machine Intelligence 12, 629–639 (1990)
20. Weickert, J.: Anisotropic Diffusion in Image Processing. B.G. Teubne, Stuttgart (1998)
21. Kindlmann, G., Whitaker, R., Tasdizen, T., Mller, T.: Curvature-based transfer functions for direct volume rendering: Methods and applications. In: Proc. IEEE Vis., pp. 513–520 (2003)
22. Sigg, C.: Representation and Rendering of Implicit Surfaces. PhD thesis, ETH Zurich (2006)
23. Tomasi, C., Manduchi, R.: Bilateral filtering for gray and color images. In: Proc. IEEE Int. Conf. on Computer Vision 1998, pp. 839–846 (1998)
24. Durand, F., Dorsey, J.: Fast bilateral filtering for the display of high-dynamic-range images. ACM Trans. Graph. 21(3), 257–266 (2002)
25. Paris, S., Kornprobst, P., Tumblin, J., Durand, F.: A gentle introduction to bilateral filtering and its applications. In: SIGGRAPH Course (2007)