RESEARCH

# Discrete Shearlet Transform on GPU with Applications in Anomaly Detection and Denoising

Xavier Gibert Serra[1*], Vishal M. Patel[1], Demetrio Labate[2] and Rama Chellappa[1]

*Correspondence: gibert@umiacs.umd.edu
[1]UMIACS, University of Maryland, College Park, MD 20742-3275, USA
Full list of author information is available at the end of the article
†Equal contributor

**Abstract**

Shearlets have emerged in recent years as one of of the most successful methods for the multiscale analysis of multidimensional signals. Unlike wavelets, shearlets form a pyramid of well-localized functions defined not only over a range of scales and locations, but also over a range of orientations and with highly anisotropic supports. As a result, shearlets are much more effective than traditional wavelets in handling the geometry of multidimensional data and this was exploited in a wide range of applications from image and signal processing. However, despite their desirable properties, the wider applicability of shearlets is limited by its computational complexity. For example, denoising a single $512 \times 512$ image using a current implementation of the shearlet-based shrinkage algorithm can take between 10 seconds and 2 minutes, depending on the number of CPU cores, and much longer processing times are required for video denoising. However, due to the parallel nature of the shearlet transform, it is possible to use Graphical Processing Units (GPU) to accelerate it. We provide an open source standalone implementation of the 2D shearlet using CUDA C++ as well as GPU-accelerated MATLAB implementations of the 2D and 3D shearlet transforms. We have instrumented the code so that we can analyze the running time of each kernel under different GPU hardware. In addition to denoising, we describe a novel application of shearlets for detecting anomalies on textured images. In this application, computation times can be reduced by a factor of 50 or more, compared to multi-core CPU implementations.

**Keywords:** shearlets; wavelets; image processing; parallelism; multi-core; GPU

## 1 Introduction

During the last decade, a new generation of multiscale systems has emerged which combines the power of the classical multiresolution analysis with the ability to process directional information with very high efficiency. Some of the most notable examples of such systems include the *curvelets* [1], the *contourlets* [2] and the *shearlets* [3]. Unlike classical wavelets, the elements of such systems form a pyramid of well localized waveforms ranging not only across various scales and locations, but also across various orientations and with highly anisotropic shapes. Thanks to their richer structure, these more sophisticated multiscale systems are able to overcome the poor directional sensitivity of traditional multiscale systems and have been used to derive several state-of-the-art algorithms in image and signal processing (cf. [4, 5]).

Shearlets, in particular, offer a unique combination of very remarkable features: they have a simple and well understood mathematical structure derived from the theory of affine systems [6, 3], they provide optimally sparse representations, in a precise sense, for a large class of images and other multidimensional data where wavelets are suboptimal [7, 8] and the directionality is controlled by shear matrices rather than rotations. This last property, in particular, enables a unified framework for continuum and discrete setting since shear transformations preserve the rectangular lattice and is an advantage in deriving faithful digital implementations [9, 10].

The shearlet decomposition has been successfully employed in many problems from applied mathematics and signal processing, including decomposition of operators [11], inverse problems [12, 13], edge detection [14, 15, 16], image separation [17] and image restoration [18, 19, 20]. However, one major bottleneck to the wider applicability of the shearlet transform is that current discrete implementations tend to be very time consuming making its use impractical for large data sets and for real-time applications. For instance, the current (CPU-based) MATLAB implementation [1] of the 2D shearlet transform, run on a typical desktop PC, takes about two minutes to denoise a noisy image of size $512 \times 512$ [9, 21]. The running time of the current (CPU-based) MATLAB implementation of the 3D shearlet transform for denoising a video sequence of size $192^3$ is about five minutes [20]. Running times for alternative shearlet implementations from Shearlab [10] as well as for the current implementation of the curvelet transform [22] are comparable.

In recent years, General Purpose Graphics Processing Units (GPGPUs) have become ubiquitous not only on High Performance Computing (HPC) clusters, but also on workstations. For example, Titan, which was until recently the world's fastest supercomputer, contains 18,688 NVIDIA Tesla K20X GPUs. These GPUs provide about 90% of Titan's peak computing performance, which is greater than 20 PetaFLOPS (quadrillion floating point operations per second). Due to their energy efficiency and capabilities, GPGPUs are also becoming mainstream on mobile platforms, such as iOS and Android devices. There are two main architectures for GPGPU computing: CUDA and OpenCL. CUDA was designed by NVIDIA, and has been around since 2006. OpenCL was originally designed by Apple, Inc, and was introduced in 2008. OpenCL is an open standard maintained by the Khronos Group, whose members include Intel, AMD, NVIDIA, and many others, so it has broader industry acceptance than any other architecture. In 2009, Microsoft introduced DirectCompute as an alternative architecture for GPGPU computing, which is only available in Windows Vista and later. OpenCL has been designed to provide the developer with a common framework for doing computation on heterogeneous devices. One of the advantages of OpenCL is that it can potentially support any computing device, such as CPUs, GPUs, and FPGAs, as long as there is an OpenCL compiler available for such processor. NVIDIA provides CUDA/OpenCL drivers, libraries and development tools for the three major Operating Systems (Linux, Windows and Mac OS X), while AMD/ATI$^{\text{TM}}$and Intel provide OpenCL drivers and tools for their respective GPUs.

The objective of this paper is to introduce and demonstrate a new implementation of the 2D and 3D discrete shearlet transform which takes advantage of the computational capabilities of the Graphics Processing Unit (GPU). To demonstrate the effectiveness of the proposed implementations, we will illustrate its application

[1]Note that this code also includes some C routines to speed-up the computation time. This is true both for the 2D and 3D implementations.

on problems of image and video denoising and on a problem of feature recognition aiming at crack detection of railway components. In particular, we will show that our new implementation takes about 40 milliseconds to denoise an image of size $512 \times 512$, which is a $233\times$ speed-up compared to single core CPU, and about 3 seconds to denoise a video of size $192^3$, which is a $551\times$ speed-up compared to single core CPU.

The organization of the paper is as follows. In Section 2, we recall the construction of 2D and 3D shearlets. Next, in Section 3, we present our implementation of the discrete shearlet transform and, in Section 4, we benchmark our implementation using three specific applications. Finally, concluding remarks and future work are discussed in Section 5.

## 2 Shearlets

In this section, we recall the construction of 2D and 3D shearlets (cf. [7, 6]).

### 2.1 2D Shearlets

To construct a smooth Parseval frames of shearlets for $L^2(\mathbb{R}^2)$, we start by defining appropriate multiscale function systems supported in the following cone-shaped regions of the Fourier domain $\widehat{\mathbb{R}}^2$:

$$\mathcal{P}_1 = \left\{(\xi_1,\xi_2) \in \mathbb{R}^2 : |\frac{\xi_2}{\xi_1}| \le 1\right\},\ \mathcal{P}_2 = \left\{(\xi_1,\xi_2) \in \mathbb{R}^2 : |\frac{\xi_2}{\xi_1}| > 1\right\}.$$

Let $\phi \in C^\infty([0,1])$ be a 'bump' function with $\operatorname{supp}\phi \subset [-\frac{1}{8},\frac{1}{8}]$ and $\phi = 1$ on $[-\frac{1}{16},\frac{1}{16}]$. For $\xi = (\xi_1,\xi_2) \in \widehat{\mathbb{R}}^2$, let $\Phi(\xi) = \Phi(\xi_1,\xi_2) = \phi(\xi_1)\,\phi(\xi_2)$ and define the function

$$W(\xi) = W(\xi_1,\xi_2) = \sqrt{\Phi^2(2^{-2}\xi_1, 2^{-2}\xi_2) - \Phi^2(\xi_1,\xi_2)}.$$

Note that the functions $W_j^2 = W^2(2^{-2j}\cdot)$, $j \ge 0$, have support inside the Cartesian coronae

$$C_j = [-2^{2j-1}, 2^{2j-1}]^2 \setminus [-2^{2j-4}, 2^{2j-4}]^2$$

and that they produce a smooth tiling of the frequency plane:

$$\Phi^2(\xi_1,\xi_2) + \sum_{j\ge 0} W^2(2^{-2j}\xi_1, 2^{-2j}\xi_2) = 1 \ \text{ for } (\xi_1,\xi_2) \in \widehat{\mathbb{R}}^2.$$

Let $V \in C^\infty(\mathbb{R})$ so that $\operatorname{supp} V \subset [-1,1]$, $V(0) = 1$, $V^{(n)}(0) = 0$, for all $n \ge 1$ and

$$|V(u-1)|^2 + |V(u)|^2 + |V(u+1)|^2 = 1 \quad \text{for } |u| \le 1.$$

For $F_{(1)}(\xi_1,\xi_2) = V(\frac{\xi_2}{\xi_1})$ and $F_{(2)}(\xi_1,\xi_2) = V(\frac{\xi_1}{\xi_2})$, the *shearlet systems associated with the cone-shaped regions* $\mathcal{P}_\nu$, $\nu = 1,2$ are defined as the countable collection of functions

$$\{\psi^{(\nu)}_{j,\ell,k} : j \ge 0, -2^j \le \ell \le 2^j, k \in \mathbb{Z}^2\}, \tag{1}$$

where

$$\hat{\psi}^{(\nu)}_{j,\ell,k}(\xi) = |\det A_{(\nu)}|^{-j/2}\, W(2^{-j}\xi)\, F_{(\nu)}(\xi A^{-j}_{(\nu)} B^{-\ell}_{(\nu)})\, e^{2\pi i \xi A^{-j}_{(\nu)} B^{-\ell}_{(\nu)} k}, \tag{2}$$

and

$$A_{(1)} = \begin{pmatrix} 4 & 0 \\ 0 & 2 \end{pmatrix}, \quad B_{(1)} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad A_{(2)} = \begin{pmatrix} 2 & 0 \\ 0 & 4 \end{pmatrix}, \quad B_{(2)} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

Note that the dilation matrices $A_{(1)}, A_{(2)}$ produce anisotropic dilations, namely, *parabolic scaling* dilations; by contrast, the *shear matrices* $B_{(1)}, B_{(2)}$ are non-expanding and their integer powers control the directional features of the shearlet system. Hence, the systems (1) form collections of well-localized functions defined at various scales, orientations and locations, controlled by the indices $j, \ell, k$ respectively. In particular, the functions $\hat{\psi}^{(1)}_{j,\ell,k}$, given by (2) with $\nu = 1$, can be written explicitly as

$$\hat{\psi}^{(1)}_{j,\ell,k}(\xi) = 2^{-2j}\, W(2^{-2j}\xi)\, V\Big(2^j \frac{\xi_2}{\xi_1} - \ell\Big)\, e^{2\pi i \xi A^{-j}_{(1)} B^{-\ell}_{(1)} k},$$

showing that their supports are contained inside the trapezoidal regions

$$\Sigma_{j,\ell} := \{(\xi_1,\xi_2) : \xi_1 \in [-2^{2j-1}, -2^{2j-4}] \cup [2^{2j-4}, 2^{2j-1}], |\tfrac{\xi_2}{\xi_1} - \ell 2^{-j}| \le 2^{-j}\}$$

in the Fourier plane (see Fig. 1). Similar properties hold for the functions $\hat{\psi}^{(2)}_{j,\ell,k}$.

A smooth Parseval frame for the whole space $L^2(\mathbb{R}^2)$ is obtained by combining the two shearlet systems associated with the cone-based regions $\mathcal{P}_1$ and $\mathcal{P}_2$ together with a coarse scale system, associated with the low frequency region. To ensure that all elements of this combined shearlet system are $C^\infty_c$ in the Fourier domain, the elements whose supports overlap the boundaries of the cone regions in the frequency domain are slightly modified. That is, we define a *shearlet system for* $L^2(\mathbb{R}^2)$ as

$$\begin{aligned} &\Big\{\widetilde{\psi}_{-1,k} : k \in \mathbb{Z}^2\Big\} \bigcup \Big\{\widetilde{\psi}_{j,\ell,k,\nu} : j \ge 0, |\ell| < 2^j, k \in \mathbb{Z}^2, \nu = 1,2\Big\} \\ &\bigcup \Big\{\widetilde{\psi}_{j,\ell,k} : j \ge 0, \ell = \pm 2^j, k \in \mathbb{Z}^2\Big\}, \end{aligned} \tag{3}$$

consisting of:

- the *coarse-scale shearlets* $\{\widetilde{\psi}_{-1,k} = \Phi(\cdot - k) : k \in \mathbb{Z}^2\}$;
- the *interior shearlets* $\{\widetilde{\psi}_{j,\ell,k,\nu} = \psi^{(\nu)}_{j,\ell,k} : j \ge 0, |\ell| < 2^j, k \in \mathbb{Z}^2, \nu = 1,2\}$, where the functions $\psi^{(\nu)}_{j,\ell,k}$ are given by (2);
- the *boundary shearlets* $\{\widetilde{\psi}_{j,\ell,k} : j \ge 0, \ell = \pm 2^j, k \in \mathbb{Z}^2\}$, obtained by joining together slightly modified versions of $\psi^{(1)}_{j,\ell,k}$ and $\psi^{(2)}_{j,\ell,k}$, for $\ell = \pm 2^j$, after that they have been restricted in the Fourier domain to the cones $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. We refer to [6] for details.

For brevity, let us denote the system (3) using the compact notation

$$\{\widetilde{\psi}_\mu,\ \mu \in M\},$$

where $M = M_C \cup M_I \cup M_B$ are the indices associated with *coarse scale shearlets*, *interior shearlets*, and *boundary shearlets*, respectively. We have the following result from [6]:

**Theorem 2.1** *The system of shearlets* (3) *is a Parseval frame for* $L^2(\mathbb{R}^2)$. *That is, for any* $f \in L^2(\mathbb{R}^2)$,

$$\sum_{\mu \in M} |\langle f, \widetilde{\psi}_\mu \rangle|^2 = \|f\|^2.$$

*All elements* $\{\widetilde{\psi}_\mu,\ \mu \in M\}$ *are* $C^\infty$ *and compactly supported in the Fourier domain.*

As mentioned above, it is proved in [7] that the 2D Parseval frame of shearlets $\{\widetilde{\psi}_\mu,\ \mu \in M\}$ provides essentially optimal approximations for functions of 2 variables which are $C^2$ regular away from discontinuities along $C^2$ curves.

The mapping from $f \in L^2(\mathbb{R}^2)$ into the elements $\langle f, \widetilde{\psi}_\mu \rangle$, $\mu \in M$, is called the *2D shearlet transform.*

## 2.2 3D Shearlets

The construction outlined above extends to higher dimensions. In 3D, a shearlet system is obtained by appropriately combining 3 systems of functions associated with the pyramidal regions

$$\mathcal{P}_1 = \left\{ (\xi_1, \xi_2, \xi_3) \in \mathbb{R}^3 : |\frac{\xi_2}{\xi_1}| \leq 1, |\frac{\xi_3}{\xi_1}| \leq 1 \right\},$$

$$\mathcal{P}_2 = \left\{ (\xi_1, \xi_2, \xi_3) \in \mathbb{R}^3 : |\frac{\xi_1}{\xi_2}| < 1, |\frac{\xi_3}{\xi_2}| \leq 1 \right\},$$

$$\mathcal{P}_3 = \left\{ (\xi_1, \xi_2, \xi_3) \in \mathbb{R}^3 : |\frac{\xi_1}{\xi_3}| < 1, |\frac{\xi_2}{\xi_3}| < 1 \right\},$$

in which the Fourier space $\widehat{\mathbb{R}}^3$ is partitioned. With $\phi$ defined as above, for $\xi = (\xi_1, \xi_2, \xi_3) \in \widehat{\mathbb{R}}^3$, we now let

$$\Phi(\xi) = \Phi(\xi_1, \xi_2, \xi_3) = \phi(\xi_1)\, \phi(\xi_2)\, \phi(\xi_3)$$

and $W(\xi) = \sqrt{\Phi^2(2^{-2}\xi) - \Phi^2(\xi)}$. As in the 2-dimensional case, we have the smooth tiling condition

$$\Phi^2(\xi) + \sum_{j \geq 0} W^2(2^{-2j}\xi) = 1 \ \text{ for } \xi \in \widehat{\mathbb{R}}^3.$$

Hence, for $d = 1, 2, 3$, $\ell = (\ell_1, \ell_2) \in \mathbb{Z}^2$, the 3D *shearlet systems associated with the pyramidal regions* $\mathcal{P}_d$ are defined as the collections

$$\{\psi^{(d)}_{j,\ell,k} : j \geq 0, -2^j \leq \ell_1, \ell_2 \leq 2^j, k \in \mathbb{Z}^3\},$$

where

$$\hat{\psi}^{(d)}_{j,\ell,k}(\xi) = |\det A_{(d)}|^{-j/2}\, W(2^{-2j}\xi)\, F_{(d)}(\xi A^{-j}_{(d)} B^{[-\ell]}_{(d)})\, e^{2\pi i \xi A^{-j}_{(d)} B^{[-\ell]}_{(d)} k},$$

$F_{(1)}(\xi_1,\xi_2,\xi_3) = V(\frac{\xi_2}{\xi_1})V(\frac{\xi_3}{\xi_1})$, $F_{(2)}(\xi_1,\xi_2,\xi_3) = V(\frac{\xi_1}{\xi_2})V(\frac{\xi_3}{\xi_2})$, $F_{(3)}(\xi_1,\xi_2,\xi_3) = V(\frac{\xi_1}{\xi_3})V(\frac{\xi_2}{\xi_3})$, the anisotropic dilation matrices $A_{(d)}$ are given by

$$A_{(1)} = \begin{pmatrix} 4 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix},\ A_{(2)} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 2 \end{pmatrix},\ A_{(3)} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{pmatrix},$$

and the *shear matrices* are defined by

$$B^{[\ell]}_{(1)} = \begin{pmatrix} 1 & \ell_1 & \ell_2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix},\ B^{[\ell]}_{(2)} = \begin{pmatrix} 1 & 0 & 0 \\ \ell_1 & 1 & \ell_2 \\ 0 & 0 & 1 \end{pmatrix},\ B^{[\ell]}_{(3)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \ell_1 & \ell_2 & 1 \end{pmatrix}.$$

Similar to the 2D case, the shearlets $\hat{\psi}^{(1)}_{j,\ell,k}(\xi)$ can be written explicitly as

$$\hat{\psi}^{(1)}_{j,\ell_1,\ell_2,k}(\xi) = 2^{-2j}\, W(2^{-2j}\xi)\, V\Big(2^j \frac{\xi_2}{\xi_1} - \ell_1\Big)\, V\Big(2^j \frac{\xi_3}{\xi_1} - \ell_2\Big)\, e^{2\pi i \xi A^{-j}_{(1)} B^{[-\ell_1,-\ell_2]}_{(1)} k}, \quad (4)$$

showing that their supports are contained inside the trapezoidal regions

$$\{\xi : \xi_1 \in [-2^{2j-1}, -2^{2j-4}] \cup [2^{2j-4}, 2^{2j-1}], |\frac{\xi_2}{\xi_1} - \ell_1 2^{-j}| \le 2^{-j}, |\frac{\xi_3}{\xi_1} - \ell_2 2^{-j}| \le 2^{-j}\}.$$

Note that these support regions become increasingly more elongated at fine scales, due to the action of the anisotropic dilation matrices $A^j_{(1)}$, wand the orientations of these regions are controlled by the shear parameters $\ell_1, \ell_2$. A typical support region is illustrated in Fig. 2. Similar properties hold for the elements associated with the regions $\mathcal{P}_2$ and $\mathcal{P}_3$.

A Parseval frame of shearlets for $L^2(\mathbb{R}^3)$ is obtained by using an appropriate combination of the systems of shearlets associated with the 3 pyramidal regions $\mathcal{P}_d$, $d = 1, 2, 3$, together with a coarse scale system, which will take care of the low frequency region. Similar to the 2D case, in order to build such system in a way that all its elements are smooth in the Fourier domain, one has to appropriately define the elements of the shearlet systems overlapping the boundaries of the pyramidal regions $\mathcal{P}_d$ in the Fourier domain. We refer to [8, 15] for details. Hence, we define the *3D shearlet systems for* $L^2(\mathbb{R}^3)$ as the collections

$$\Big\{\widetilde{\psi}_{-1,k} : k \in \mathbb{Z}^3\Big\} \bigcup \Big\{\widetilde{\psi}_{j,\ell,k,d} : j \ge 0, |\ell_1| < 2^j, |\ell_2| \le 2^j, k \in \mathbb{Z}^3, d = 1,2,3\Big\}$$
$$\bigcup \Big\{\widetilde{\psi}_{j,\ell,k} : j \ge 0, \ell_1, \ell_2 = \pm 2^j, k \in \mathbb{Z}^3\Big\},$$

which again can be identified as the coarse-scale, interior and boundary shearlets. It turns out that the 3D system of shearlets is a Parseval frame of $L^2(\mathbb{R}^3)$ [6] and it provides essentially optimal approximations for functions of 3 variables which are $C^2$ regular away from discontinuities along $C^2$ surfaces [8].

## 3 Discrete Implementation

A faithful numerical implementation of the 2D shearlet transform was originally presented in [9]. Let us briefly recall the main steps of this implementation.

### 3.1 2D Discrete Shearlet Transform

Recall that the shearlet coefficients associated with the interior shearlets can be expressed as:

$$\langle f, \psi_{j,\ell,k}^{\nu}\rangle = 2^{3j/2} \int_{\widehat{\mathbb{R}}^2} \hat{f}(\xi)\, W(2^{-2j}\xi)\, F_{(\nu)}(\xi A_{(\nu)}^{-j} B^{-\ell_{(\nu)}})\, e^{2\pi i \xi A_{(\nu)}^{-j} B^{-\ell_{(\nu)}} k}\, d\xi.$$

First, to compute $\hat{f}(\xi_1,\xi_2)\, W(2^{-2j}\xi)$ in the discrete domain, at the resolution level $j$, we apply the Laplacian pyramid algorithm [23], which is implemented in space-domain. Let $\hat{f}[k_1,k_2]$ denote 2D Discrete Fourier Transform (DFT) of $f \in \ell^2(\mathbb{Z}_N^2)$, where we adopt the convention that brackets $[\cdot,\cdot]$ denote arrays of indices, parentheses $(\cdot,\cdot)$ denote function evaluations, and where we interpret the numbers $\hat{f}[k_1,k_2]$ as samples $\hat{f}[k_1,k_2] = \hat{f}(k_1,k_2)$ from the trigonometric polynomial

$$\hat{f}(\xi_1,\xi_2) = \sum_{n_1,n_2=0}^{N-1} f[n_1,n_2]\, e^{-2\pi i(\frac{n_1}{N}\xi_1 + \frac{n_1}{N}\xi_2)}.$$

The Laplacian pyramid algorithm will accomplish the multiscale partition illustrated in Figure 3, by decomposing $f_a^{j-1}[n_1,n_2]$, $0 \le n_1,n_2 < N_{j-1}$, into a low pass filtered image $f_a^j[n_1,n_2]$, a quarter of the size of $f_a^{j-1}[n_1,n_2]$, and a high pass filtered image $f_d^j[n_1,n_2]$. Observe that the matrix $f_a^j[n_1,n_2]$ has size $N_j \times N_j$, where $N_j = 2^{-2j}N$, and $f_a^0[n_1,n_2] = f[n_1,n_2]$ has size $N \times N$. In particular, we have that

$$\widehat{f_d^j}(\xi_1,\xi_2) = \hat{f}(\xi_1,\xi_2)\, W(2^{-2j}(\xi_1,\xi_2))$$

and thus, $f_d^j[n_1,n_2]$ are the discrete samples of a function $f_d^j(x_1,x_2)$, whose Fourier transform is $\widehat{f_d^j}(\xi_1,\xi_2)$. Since this operation is implemented as a convolution in space-domain, this step of the algorithm is one of the most computationally expensive.

The next step produces the directional filtering and this is achieved by computing the DFT on the pseudo-polar grid, and then applying a one-dimensional band-pass filter to the components of the signal with respect to this grid. More precisely, let us define the pseudo-polar coordinates $(u,v) \in \mathbb{R}^2$ as follows:

$$\begin{aligned}(u,w) &= (\xi_1, \tfrac{\xi_2}{\xi_1}) \quad \text{if } (\xi_1,\xi_2) \in \mathcal{P}_1,\\ (u,w) &= (\xi_2, \tfrac{\xi_1}{\xi_2}) \quad \text{if } (\xi_1,\xi_2) \in \mathcal{P}_2.\end{aligned}$$

After performing this change of coordinates, we obtain

$$\hat{f}(\xi_1,\xi_2)\, W(2^{-2j}\xi_1, 2^{-2j}\xi_2)\, F_{(\nu)}(\xi A_{(\nu)}^{-j} B^{-\ell_{(\nu)}}) = g_j(u,w)\, V(2^j w - \ell), \qquad (5)$$

where $g_j(u,w) = \hat{f}_d^j(\xi_1,\xi_2)$. This shows that the directional components are obtained by simply translating the window function $V$. The discrete samples

$g_j[n_1, n_2] = g_j(n_1, n_2)$ are the values of the DFT of $f_d^j[n_1, n_2]$ on a pseudo-polar grid.

Now let $\{v_{j,\ell}[n] : n \in \mathbb{Z}\}$ be the sequence whose discrete Fourier transform gives the samples of the window function $V(2^j k - \ell)$, i.e., $\hat{v}_{j,\ell}[k] = V(2^j k - \ell)$. Then, for fixed $n_1 \in \mathbb{Z}$, we have

$$\mathcal{F}_1\left(\mathcal{F}_1^{-1}\left(g_j[n_1, n_2]\right) * v_{j\ell}[n_2]\right) = g_j[n_1, n_2]\,\mathcal{F}_1\left(v_{j\ell}[n_2]\right), \tag{6}$$

where $*$ denotes the one-dimensional convolution along the $n_2$ axis and $\mathcal{F}_1$ is the one-dimensional discrete Fourier transform Thus (6) gives the algorithmic implementation for computing the discrete samples of $g_j(u, w)\, v(2^j w - \ell)$. At this point, to compute the shearlet coefficient in the discrete domain, it suffices to compute the inverse PDFT or directly re-assemble the Cartesian sampled values and apply the inverse two-dimensional FFT.

Figure 3 illustrates the cascade of Laplacian pyramid and directional filtering.

Recall that, once the discrete shearlet coefficients are obtained, the inverse shearlet transform is computed using the following steps: (i) convolution of discrete shearlet coefficients and synthesis directional filters; (ii) sum of all directional components; (iii) reconstruction by inverse Laplacian pyramidal transformation.

### 3.2 2D GPU-based Implementation

Before implementing the 2D Discrete Shearket Transform algorithm on the GPU, we profiled the existing implementation available as a MATLAB toolbox at `http://www.math.uh.edu/~dlabate/shearlet_toolbox.zip`. Table 3 contains the breakdown of the processing times showing that the FFT computations used to perform directional filtering and the convolution part of the *à trous* algorithm used for pyramidal image decomposition and reconstruction take around 75% of the computation time. Hence they were the first candidates for porting into CUDA.

Since most of the computing time for performing a discrete shearlet transform is spent in FFT function calls, it is crucial to have the best possible library to perform FFTs. The main two GPU vendors provide optimized FFT libraries: NVIDIA provides *cuFFT* as part of its CUDA Toolkit, and AMD provides *clAmdFft* as part of its Accelerated Parallel Processing Math Libraries (APPML). We have decided to use CUDA as our development architecture both because there is better documentation and because of the availability of more mature development tools. We have implemented the device code in CUDA C++, while the host code is pure C++. Since both CUDA C/C++ and OpenCL are based on the C programming language, porting the code from CUDA to OpenCL should not be difficult. However, for code compactness, we have made extensive use of templates and operator overloading, which are supported in CUDA C++, but not in OpenCL, which is based on C99.

To facilitate the development, we have used GPUmat from the GP-you Group, a free (GPLv3) GPU engine for MATLAB® (source code is available from `http://sourceforge.net/projects/gpumat/`). This framework provides two new classes, *GPUsingle* and *GPUdouble*, which encapsulate vectors of numerical data allocated on GPU memory, and allow mathematical operations on objects of such classes via function and operator overloading. Transfers between CPU and GPU memory are as simple as doing type casting, and memory allocation and deallocation is done automatically. The idea is that existing MATLAB functions could be reused without any code changes. In practice, however, in order to get acceptable performance it is necessary to hand-tune the code or even use lower lever languages such as C/C++.

Fortunately, the GPUmat framework provides an interface for manipulating these objects from MEX files, and a mechanism for loading custom kernels. Although there are commercial alternatives to GPUmat such as Jacket from AccelerEyes, or the Parallel Computing Toolbox from Mathworks, we have found that GPUmat is pretty robust and adds very little overhead to the execution time as long as we follow good programming practices such as in-place operations and reuse of preallocated buffers.

Our implementation supports both single precision (32-bit) and double precision (64-bit) IEEE 754 floating point numbers. We generate the filter bank of directional filters using the Fourier-domain approach from [9], where directional filters are designed as Meyer-type window functions in the Fourier domain. Since this step only needs to be run once and does not depend on the image dimensions, we precompute these directional filters using the original MATLAB implementation.

For the Laplacian pyramidal decomposition, we ported the *à trous* algorithm using symmetric extension [2] into CUDA. This algorithm requires performing non-separable convolutions with decimated signals. For efficiency reasons, the kernel that performs *à trous* convolutions preloads blocks of data into shared memory, so that the memory is only accessed once from each GPU thread.

With the above GPU-based Laplacian pyramid and directional filter implementation, it is just a matter of applying convolutions in the GPU to find the forward and inverse shearlet transform.

The main steps of our GPU-based shearlet transform are as shown in table 1.

### 3.3 3D discrete shearlet transform

The algorithm for the discretization of the 3D shearlet transform is very similar to the 2D shearlet transform and our implementation of the 3D discrete shearlet transform adapts the code available from `http://www.math.uh.edu/~dlabate/3Dshearlet_toolbox.zip` and described in [20]. The main practical difference is that storing the 3D shearlet coefficients is much more memory-intensive. Since the memory requirement can be easily exceed the available GPU memory, in our algorithm we compute one convolution at a time in CUDA and add the result to the output.

## 4 Applications

In the following, we illustrate the advantages of our new implementation of the discrete shearlet transform by considering three applications: denoising of natural images corrupted with white Gaussian noise, detection of cracks in textured images and denoising of videos. The source code, sample data as well as the MATLAB scripts used to generate all the figures in this paper are publicly available at `http://www.umiacs.umd.edu/~gibert/ShearCuda.zip`.

For benchmark, we have evaluated the performance of the new discrete shearlet transform both on multicore CPUs and GPU. All CPU tests have been performed on a Dell PowerEdge C6145 with four-socket AMD Opteron$^{\mathrm{TM}}$6274 processors at 2.2GHz (64 cores total) and 256GB RAM, running Red Hat Enterprise

Linux (REHL) 6. This machine is one of 16 identical nodes in the High Performance Computing (HPC) cluster Euclid at the University of Maryland. During these benchmarks, we had exclusive access to this node, and no other processes were running, except for regular system services. To better understand the performance of this code when running on systems with different number of cores, we limited the number of available cores on some of the experiments. We found that neither MATLAB's *maxNumCompThreads* nor *–singleCompThread* work reliably on non-Intel processors, so we used the *taskset* Linux command to set the processor affinity to the desired number of cores. GPU tests were performed on different machines running RHEL 5 or 6, and CUDA 4.2 or 5.0. The tests include devices with CUDA Compute Capabilities (CC) between 1.3 and 3.5. Table 2 summarizes the configurations used in our experiments.

### 4.1 Image denoising

As a first test, we evaluated the performance of our implementation of the discrete shearlet transform on a problem of image denoising, using a standard denoising algorithm based on hard threshold of the shearlet coefficients. The setup is similar to the one described in [9]. That is, given an image $f \in \mathbb{R}^{N^2}$, we observe a noisy version of it given by $u = f + \epsilon$, where $\epsilon \in \mathbb{R}^{N^2}$ is an additive white Gaussian noise process which is independent of $f$, i.e., $\epsilon \sim N(0, \sigma^2 \mathbf{I}_{N^2 \times N^2})$. Our goal is to compute an estimate $\tilde{f}$ of $f$ from the noisy data $u$ by applying a classical hard thresholding scheme [24] on the shearlet coefficients of $u$. The threshold levels are given by $\tau_{i,j,n} = \sigma^2_{\epsilon_{i,j}} / \sigma^2_{i,j,n}$, as in [9, 2, 25], where $\sigma^2_{i,j,n}$ denotes the variance of the $n$-th coefficient at the $i$th directional subband in the $j$th scale, and $\sigma^2_{\epsilon_{i,j}}$ is the noise variance at scale $j$ and directional band $i$. The variances $\sigma^2_{\epsilon_{i,j}}$ are estimated by using a Monte-Carlo technique in which the variances are computed for several normalized noise images and then the estimates are averaged.

For our experiments, we used 5 levels of the Laplacian pyramid decomposition, and we applied a directional decomposition on 4 of the 5 scales. We used 8 shear filters of sizes $32 \times 32$ for the first two scales (coarser scales), and 16 shear filters of sizes $16 \times 16$ for the third and forth levels (fine scales). The shear filters are Meyer-type windows [9]. We used the $512 \times 512$ Barbara image to test our algorithm and, to assess its performace, we used the peak signal-to-noise ratio (PSNR), measured in decibels (dB), defined by

$$PSNR = 20 \log_{10} \frac{255N}{\|f - \tilde{f}\|_F},$$

where $\|\cdot\|_F$ is the Frobenius norm, the given image $f$ is of size $N \times N$ and $\tilde{f}$ denotes the estimated image.

In order to minimize latency as well as bandwidth usage on the PCIe bus, we first transferred the input image to GPU memory, then we let all the computation happen on the GPU and we finally transferred the results back to CPU memory. We have verified that both CPU and GPU implementations provide an output PSNR of 29.9dB when the input PSNR is 22.1dB. At these noise levels, there is no difference in PSNR between the single and the double precision implementations.

To verify the numerical accuracy, we ran the shearlet decomposition and reconstruction on a noise free image (without thresholding), and we got a reconstruction MSE (Mean Squared Error) of $9.197\times10^{-09}$ for single precision and $2.503\times10^{-12}$ for double precision on a GeForce GTX 690. On the CPU implementation, we get reconstruction errors of $9.1711\times10^{-09}$ and $1.6643\times10^{-26}$, respectively. This varifies that our implementation does provide the exact reconstruction.

The running times vary significantly depending on the number of CPU cores available and the GPU model. Figure 8 shows a comparison of running times (wall times) of the image denoising algorithm on different hardware configurations. We can clearly see that the CPU implementation does not scale well as we increase the number of CPU cores due to parts of the algorithm running sequentially. For a fair comparison of multicore vs GPU, we would have to compare the performance to a fully optimized CPU implementation. It should be noted that there is enough coarse level parallelism on this algorithm to accomplish full CPU utilization without incurring in inter CPU communication issues. However, the trend reveals that on this application GPU is more efficient than CPU. In summary, the denoising algorithm takes 8.89 seconds on 4 CPU cores vs. 0.038 seconds on the GeForce GTX 690 (a 233× speed-up) when using single precision. For double precision, it takes 10.7 seconds on 4 CPU cores vs. 0.127 seconds on the GeForce GTX 690 (an 84× speed-up).

Table 3 shows the breakdown of different parts of the image denoising algorithm both on CPU and GPU.

### 4.2 Crack detection

Detection of cracks on concrete structures is a difficult problem due to the changes in width and direction of the cracks, as well as the variability in the surface texture. This problem has received considerable attention recently. Redundant representations, such as undecimated wavelets, have been extensively used for crack detection [26, 27]. However, wavelets have poor directional sensitivity and have difficulties in detecting weak diagonal cracks. To overcome this limitatation, Ma *et al.* [28] proposed the use of the *nonsubsampled contourlet transform* [2] for crack detection. However, all these methods rely on the assumption that the background surface can be modeled as additive white Gaussian noise and his assumption leads to matched filter solutions. As a matter of fact, on real images textures are highly correlated and applying linear filters leads to poor performance.

To address this problem, we propose a completely new approach to crack detection based on separating the image into morphological distinct components using sparse reprentations, adaptive thresholding and variational regularization. This technique was pioneered by Stark *et al.* [29] and later extended and generalized by many authors (e.g., [30, 18, 17]). In particular, we will use the Iterative Shrinkage Algorithm with a combined dictionary of shearlets and wavelets to separate cracks from background texture.

To demonstrate the performance of the GPU-accelerated Iterative Shrinkage Algorithm, we processed three $512\times512$ images. The images correspond to cracks on concrete railroad crossties collected by ENSCO Inc. during summer 2012 using four $2048\times1$ line-scan cameras, which were assembled into $8192\times3072$ frames. The cameras were triggered using a calibrated encoder, producing images with square pixels with a constant size of 0.43mm. We have manually cropped these images so that we can decouple crack detection from crosstie boundary tracking. As one can see from Figure 5, these cracks propagate in different directions and the background texture has a lot of variation. However, due to the fact that the information in these images is highly redundant, it is possible to separate the image into two components, that is, cracks and texture, by solving an $\ell_1$ optimization problem [17].

More precisely, we will model an image $x$ containing cracks on textural background as a superposition of a crack component $x_c$ with a textural component $x_t$:

$$x = x_c + x_t.$$

Let $\Phi_1$ and $\Phi_2$ be the dictionaries corresponding to wavelets and shearlets, respectively. We assume that $x_c$ is sparse in a shearlet dictionary $\Phi_1$ and similarly $x_t$ is sparse in a wavelet dictionary $\Phi_2$. That is, we assume that there are sparse coefficients $a_c$ and $a_t$ so that $x_c = \Phi_1 a_c$ and $x_t = \Phi_2 a_t$. Then, one can separate these components from an $x$ via the coefficients $a_c$ and $a_t$ by solving the following optimization problem:

$$(\hat{a}_c, \hat{a}_t) = \arg\min{}_{a_c, a_t} \lambda \|a_c\|_1 + \lambda \|a_t\|_1 + \frac{1}{2}\|x - \Phi_1 a_c - \Phi_2 a_t\|_2^2, \tag{7}$$

where for an $n$-dimensional vector $b$ the $\ell_1$ norm is defined as $\|b\|_1 = \sum_i |b_i|$. This image separation problem can be solved efficiently using an *iterative shrinkage algorithm* proposed in [17].

In our numerical experiments, we used symlet wavelets with 4 decomposition levels to generate $\Phi_2$ and a 4-level shearlet decomposition with Meyer filters of sizes $80 \times 80$ on all 4 scales, 8 directional filters on the first three scales, and 16 directional filters on the forth scale, to generate $\Phi_1$. To assess the performance of the separation algorithm, we calculated the ROC curves for each image using the following 2 detection methods.

a) *Shearlet-C*: This method takes advantage of the Parseval property of the shearlet transform and performs crack detection directly in the transform domain. We first decompose the image into cracks and texture components using Iterative Shrinkage with a shearlet dictionary and a wavelet one. Instead of using the reconstructed image, we analyze the values of the shearlet transform coefficients. For each scale in the shearlet transform domain, we analyze the directional components corresponding to each displacement and collect the maximum magnitude across all directions. If the sign of the shearlet coefficient corresponding to the maximum magnitude is positive, we classify the corresponding pixel as background, otherwise we assign the norm of the vector containing the maximum responses at each scale to each pixel and we apply a threshold.

b) *Shearlet-I*: We first decompose the image into cracks and texture components as described for the previous method. Then, we apply an intensity threshold on the reconstructed cracks image.

We compare our results to the following 2 basic methods not based on shearlets:

c) *Intensity*: This is the most basic approach, which only uses image intensity. After compensating for slow variations of intensity in the image, we apply a global threshold.
d) *Canny*: We use the Canny[31] edge detector as implemented in MATLAB using the default $\sigma = \sqrt{2}$ and the default high to low threshold ratio of 40%.

After using a low-level detector, it may be necessary to remove small isolated regions corresponding to false detections due to random noise. This postprocessing step may reduce the false detection rate on intensity-based methods. However, to provide an objective comparison, we have generated the experimental results without running any postprocessing. We leave the perfomance analysis of a complete crack detector for future work.

To evaluate the performance of each crack detector, we manually annotated the crack pixels in each image. To mitigate the effect of ambiguous segmentation boundaries, we annotated the boundaries around the cracks as tightly as possible (making sure that only pixels completely contained inside the crack boundaries are annotated as such) and defined an envelope region around each crack whose labels are treated as "do not care". Formally, let $\Omega$ denote the set of pixels in the image, and $F$ (foreground) denote the set of pixels labeled as cracks. We define the set $B$ (backgrond) as

$$B = \{x \in \Omega : \min_{f \in F} \|x - f\| > \delta\}.$$

where $\|x - f\|$ denotes the Euclidean distance between sites $x$ and $f$. In our experiments we used $\delta = 3$. To account for possible small inaccuracies in the ground truth, we performed a bipartite graph matching between the detected crack pixels and the crack pixels in the ground truth. For our experiments, we allow matching within a maximum distance of 2 pixels. This choice of matching metric does not penalize crack overestimation errors as long as these errors are contained in such envelope. This allows us to decouple errors in estimating the position of the crack centerline from errors in estimating the crack width, which is more sensitive to lighting variations. Let $D$ be the set of pixels detected as cracks by a given detector and

$$\begin{array}{lll} tp = |D \cap F| & fn = |\bar{D} \cap F| & p = tp + fn = |F| \\ tn = |\bar{D} \cap B| & fp = |D \cap B| & n = tn + fp = |B| \end{array}$$

The probability of detection ($PD$) and false alarm ($PF$) are defined as

$$PD = \frac{tp}{p} \qquad PF = \frac{fp}{n}$$

A sequence of admissible detectors $D|_{PF \leq \epsilon}$, for a given false detection rate $\epsilon$, $0 \leq \epsilon \leq 1$ would produce monotonically increasing detection rates, $PD|_{PF \leq \epsilon}$. The Receiving Operating Characteristic function (ROC curve) is defined as $PD$ as a function of $PF$

$$ROC(x) = \max_{\epsilon \leq x} PD|_{PF=\epsilon}$$

One commonly used metric is the Area Under the ROC Curve (AUC), defined by

$$AUC = \int_0^1 ROC(x)\, dx,$$

which corresponds to the probability that a sample randomly drawn from $F$ will receive a score higher than a sample randomly drawn from $B$. $AUC$ provides a mesure of the average performance of the detection across all possible sensitivity settings. Although it is an important mesure, in practice we are interested in knowing how well the detector will work when we choose a particular sensitivy setting. For this reason, we have selected Constant False Alarm Rate (CFAR) detectors with $PF = 10^{-3}$ and $PF = 10^{-4}$ and we report the corresponding $PD$. For completeness, we also report the $F_1$ score (also know as the Dice similarity index), which is defined as

$$F_1 = \frac{2\,tp}{2\,tp + fn + fp}$$

The $F_1$ score is also known as the balanced $F-$score, since it is equivalent to the harmonic mean of the *precision* and *recall*:

$$F_1 = 2\,\frac{precision \cdot recall}{precision + recall}$$

where

$$precision = \frac{tp}{p} \qquad recall = \frac{tp}{tp + fn}.$$

In this paper, we report the peak $F_1$ score for all methods. The Canny edge detection method estimates the location of the crack boundary, while the other three methods estimate the location of the crack itself. To have a meaningful comparison, we have generated a separate ground truth masks for the crack outline, so we can use the same matching metric on the Canny method. For each method, we have used the same algorithm parameters on all the images.

Table 4 summarizes our results. We observe that our shearlet-based detectors perform consistently well on all evaluation metrics. Note that, on Image 3, the Shearlet-I method, which is based on intensity in the reconstructed image, produces better results than all other methods. Due to its simplicity, the intensity-based methods is still being used by the industry. For example, the system recently proposed in [32] uses pixel intensities to detect cracks on road pavement. Based on the results from Table 4, we can conclude that, with the proper image preprocessing, intensity can still be a powerful feature for crack detection. However, the detection performance provided by shearlet-based features is more consistent across images. In future work, we will further explore the potential of combining both intensity and shearlet-based features.

## 4.3 Video denoising

Video denosing can be performed using the same type of algorithm described above for image denoising and consisting, essentially, in computing the shearlet coefficients of the noisy data, followed by hard threshoding and reconstruction from the thresholded coefficients. Similar to the previous section, a noisy video is obtained by adding white Gaussian noise to a video sequence.

We have tested our GPU-based implementation of the 3D shearlet video denoising algorithm using the $192 \times 192 \times 192$ waterfall video sequence. Figure 7 shows frame 96 before and after denoising. Figure 9 compares the running times of the video denoising algorithm based on CPU vs. GPU. One can notice that, when we go from single core to dual core, the run time drops from 27.5 minutes to 14.4 minutes on single precision (a $1.91\times$ speed-up). However, when going from dual-core to quad core we only get $1.62\times$ speed-up, and the rate of improvement as we keep doubling the number of cores keeps diminishing, to the point where the improvement from single core to 64 cores is just a $9.45\times$ speed-up. On the other hand, a GeForce 480 produces the same result in just 3 seconds, a remarkable $551\times$ speed-up compared to single core CPU, and $58\times$ speed-up over 64 CPU cores.

## 5 Discussion and Conclusion

The shearlet transform is an advanced multiscale method which has emerged in recent years as a refinement of the traditional wavelet transform and was shown to perform very competitively over a wide range of image and data processing problems. However, standard CPU-based numerical implementations are very time-consuming and make the application of this method to large data sets and real-time problems very impractical.

In this paper, we have described how to speed-up the computation of the 2D/3D discrete shearlet transform by using GPU-based implementations. The development of algorithms on GPU used to be tedious and require a very specialized knowledge of the hardware. Using CUDA this is no longer the case and scientists with C/C++ programming skills can quickly develop efficient GPU implementations of data-intensive algorithms. In this paper, we have taken advantage of the GPU-based implementation of the Fast Fourier Transform and used the capabilities of MATLAB for quick prototyping. The results presented in this paper illustrate the practical benefits of this approach. For example, a GeForce 480 GTX, a $200 graphics card, can perform video denoising 58 times faster than an expensive 64-core machine while consuming much less power.

Our new implementation enables the efficient application of the sherleat decomposition to a variety of image and data processing tasks for which the required CPU resources would be prohibitive. There are further improvements and extensions that can be achieved such as pre-calculating the filter coefficients and porting the code to OpenCL so it can also run on AMD and Intel GPUs, but this would go beyond the scope of this paper.

**Competing interests**
The authors declare that they have no competing interests.

**Acknowledgements**
The authors thank Amtrak, ENSCO, Inc. and the Federal Railroad Administration for providing the images used in section 4.2. This work was supported by the Federal Railroad Administration under contract DTFR53-13-C-00032. DL acknowledges support from NSF grant DMS 1008900/1008907 and DMS 1005799.

**Author details**
[1]UMIACS, University of Maryland, College Park, MD 20742-3275, USA. [2]Department of Mathematics, University of Houston, Houston, TX 77204-3008, USA.

**References**

1. Candès, E.J., Donoho, D.L.: New tight frames of curvelets and optimal representations of objects with $c^2$ singularities. Comm. Pure Appl. Math. **57**, 219–266 (2004)
2. Cunha, A.L., Zhou, J., Do, M.N.: The nonsubsampled contourlet transform: Theory, design, and applications. IEEE Transactions on Image Processing **15**, 3089–3101 (2006)
3. Labate, D., Lim, W., Kutyniok, G., Weiss, G.: Sparse multidimensional representation using shearlets. In: Wavelets XI (San Diego, CA, 2005) vol. SPIE Proc. 5914, pp. 254–262 (2005). SPIE, Bellingham, WA
4. Kutyniok, G., Labate, D.: Shearlets: Multiscale Analysis for Multivariate Data. Birkhäuser, Boston (2012)
5. Starck, J.-L., Murtagh, F., Fadili, J.M.: Sparse image and signal processing: Wavelets, curvelets, morphological diversity. Cambridge books online. Cambridge University Press (2010)
6. Guo, K., Labate, D.: The construction of smooth parseval frames of shearlets. Math. Model. Nat. Phenom. **8**(1), 82–105 (2013)
7. Guo, K., Labate, D.: Optimally sparse multidimensional representation using shearlets. Siam  J. Math. Anal. **9**, 298–318 (2007)
8. Guo, K., Labate, D.: Optimally sparse representations of 3d data with $c^2$ surface singularities using parseval frames of shearlets. Siam  J. Math. Anal. **44**, 851–886 (2012)
9. Easley, G.R., Labate, D., Lim, W.: Sparse directional image representations using the discrete shearlet transform. Appl. Comput. Harmon. Anal. **25**(1), 25–46 (2008)
10. Kutyniok, G., Shahram, M., Zhuang, X.: Shearlab: A rational design of a digital parabolic scaling algorithm. SIAM J. on Imaging Sciences **5**(4), 1291–1332 (2012)
11. Guo, K., Labate, D.: Representation of fourier integral operators using shearlets. J. Fourier Anal. Appl. **14**, 327–371 (2008)
12. Colonna, F., Easley, G.R., Guo, K., Labate, D.: Radon transform inversion using the shearlet representation. Appl. Comput. Harmon. Anal. **29**(2), 232–250 (2010)
13. Vandeghinste, B., Goossens, B., Van Holen, R., Vanhove, C., Pizurica, A., Vandenberghe, S., Staelens, S.: Iterative ct reconstruction using shearlet-based regularization. IEEE Transactions on Nuclear Science **60**(5), 3305–3317 (2013)
14. Guo, K., Labate, D.: Characterization and analysis of edges using the continuous shearlet transform. SIAM on Imaging Sciences **2**, 959–986 (2009)
15. Guo, K., Labate, D.: Analysis and detection of surface discontinuities using the 3d continuous shearlet transform. Appl. Comput. Harmon. Anal. **30**, 231–242 (2011)
16. Yi, S., Labate, D., Easley, G.R., Krim, H.: A shearlet approach to edge analysis and detection. IEEE Transactions on Image Processing **18**(5), 929–941 (2009)
17. Kutyniok, G., Lim, W.: Image separation using wavelets and shearlets. In: Curves and Surfaces (Avignon, France, 2010), Lecture Notes in Computer Science 6920, (2011). Springer
18. Easley, G., Labate, D., Negi, P.S.: 3d data denoising using combined sparse dictionaries. Math. Model. Nat. Phenom. **8**(1), 60–74 (2013)
19. Patel, V.M., Easley, G., Healy, D.M.: Shearlet-based deconvolution. IEEE Transactions on Image Processing **18**, 2673–2685 (2009)
20. Negi, P.S., Labate, D.: 3-d discrete shearlet transform and video processing. IEEE Transactions on Image Processing **21**, 2944–2954 (2012)
21. Easley, G., Labate, D., Patel, V.M.: Directional multiscale processing of images using wavelets with composite dilations. Journal of Mathematical Imaging and Vision (2012)
22. Candès, E.J., Demanet, L., Donoho, D., Ying, L.: Fast discrete curvelet transforms. SIAM Multiscale Model. Simul. **(5)(3)**, 861–899 (2006)
23. Burt, P.J., Adelson, E.H.: The laplacian pyramid as a compact image code. IEEE Transactions on Communications **16**(11), 2675–2681 (2007)
24. Donoho, D., Johnstone, I.: Adapting to unknown smoothness via wavelet shrinkage. J. Amer. Statist. Assoc. **90**, 1200–1224 (1995)
25. Chang, S.G., Yu, B., Vetterli, M.: Adaptive wavelet thresholding for image denoising and compression. IEEE Transactions on Image Processing **9**(9), 1532–1546 (2000)
26. Subirats, P., Dumoulin, J., Legeay, V., Barba, D.: Automation of pavement surface crack detection using the continuous wavelet transform. In: IEEE International Conference on Image Processing, pp. 3037–3040 (2006)
27. Chambon, S., Moliard, J.: Automatic road pavement assessment with image processing: Review and comparison. Int. Journal of Geophysics (doi:10.1155/2011/989354) (2011)
28. Ma, C., Zhao, C., Hou, Y.: Pavement distress detection based on nonsubsampled contourlet transform. Int. Conf. on Computer Science and Software Engineering **1**, 28–31 (2008)
29. Starck, J.-L., Elad, M., Donoho, D.L.: Image decomposition via the combination of sparse representation and a variational approach. IEEE Transactions on Image Processing **14**(10), 1570–1582 (2005)
30. Bobin, J., Starck, J.-L., Fadili, M.J., Moudden, Y., Donoho, D.L.: Morphological component analysis: an adaptive thresholding strategy. IEEE Transactions on Image Processing **16**(11), 2675–2681 (2007)
31. Canny, J.: A computational approach to edge detection. IEEE Trans. Pattern Analysis and Machine Intelligence **8**(6), 679–698 (1986)
32. Oliveira, H., Correia, P.L.: Automatic road crack detection and characterization. IEEE Transactions on Intelligent Transportation Systems **14**(1), 155–168 (2013)

**Figures**

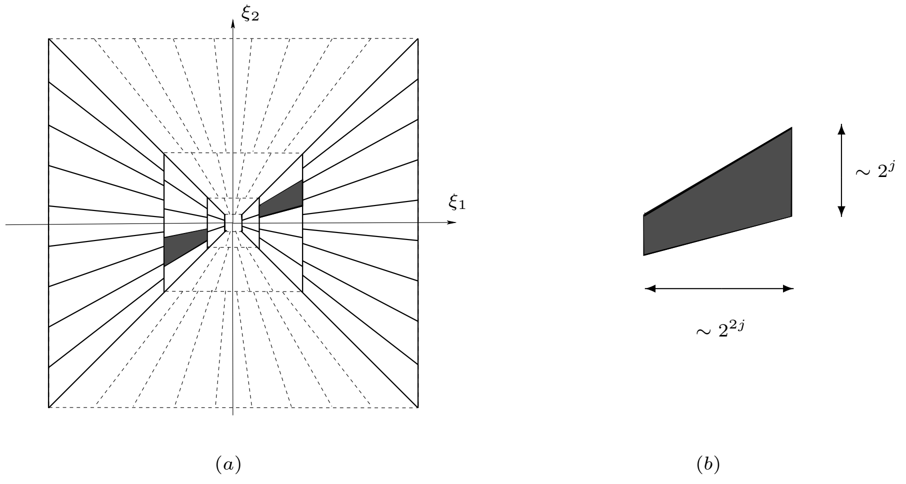

**Figure 1** (a) The tiling of the frequency plane $\widehat{\mathbb{R}}^2$ induced by the shearlets. (b) Frequency support $\Sigma_{j,\ell}$ of a shearlet $\psi^{(1)}_{j,\ell,k}$, for $\xi_1 > 0$. The other half of the support, for $\xi_1 < 0$, is symmetrical.

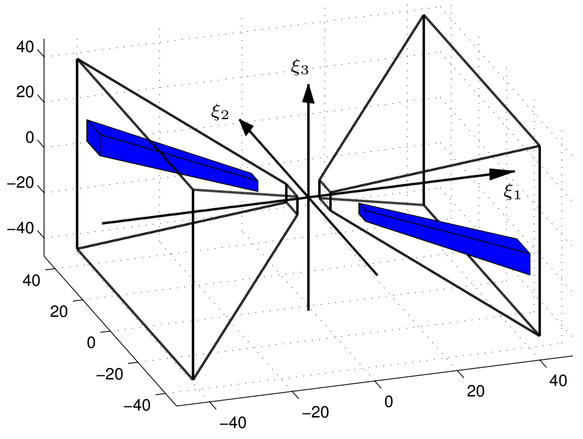

**Figure 2** Frequency support of a representative shearlet function $\psi_{j,\ell,k}$, inside the pyramidal region $\mathcal{P}_1$. The orientation of the support region is controlled by $\ell = (\ell_1, \ell_2)$; its shape is becoming more elongated as $j$ increases ($j = 4$ in this plot).

**Table 1** Main steps of the shearlet transform

| | Forward transform | | Inverse transform |
|---|---|---|---|
| 1. | *Laplacian decomposition* | 1. | *Forward FFT of directional components* |
| 2. | *Forward FFT of Laplacian components* | 2. | *Modulation with complex conjugate directional filter bank* |
| 3. | *Modulation of Laplacian components with directional filter bank* | 3. | *Inverse FFT of directional components* |
| 4. | *Inverse FFT of directional components* | 4. | *Laplacian reconstruction* |

**Tables**

**Additional Files**

Additional file 1 — Sample additional file title

Additional file descriptions text (including details of how to view the file, if it is in a non-standard format or the file extension). This might refer to a multi-page table or a figure.

[1] Although the GeForce GTX 690 is a dual-GPU with a total of 4GB and 3072 cores, we have only used one of the 2 devices in the GPU for our experiments.

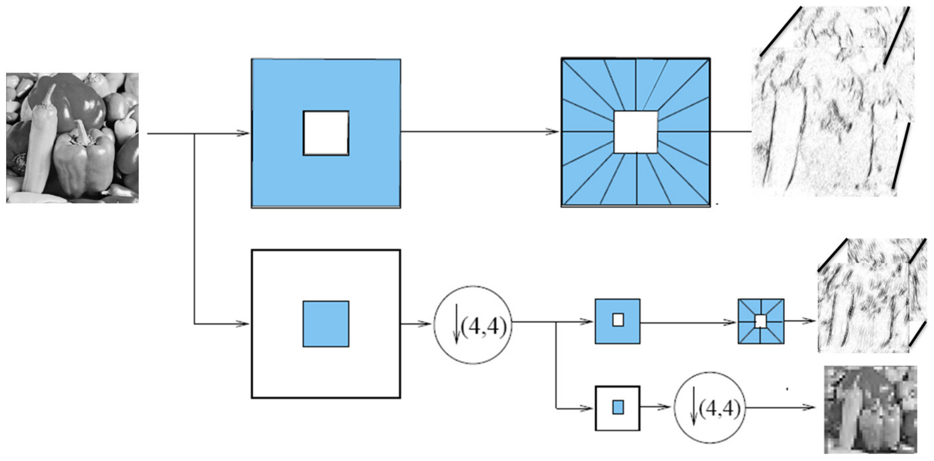

**Figure 3** The figure illustrating the succession of Laplacian pyramid and directional filtering.

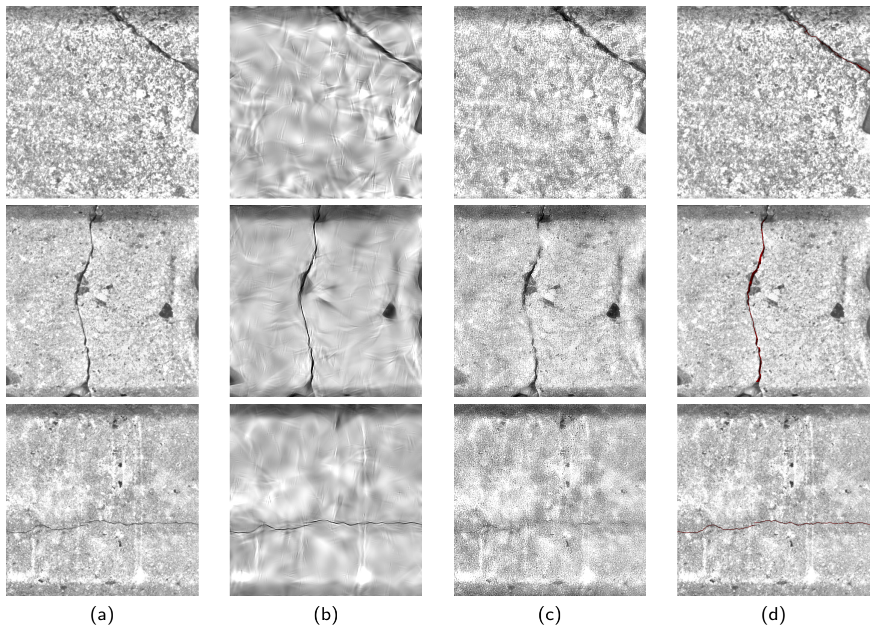

**Figure 4 Image separation.** (a) Original images separated into (b) Cracks and (c) Textural background components (d) Crack ground truth

**Table 2** Specifications and computing environments for each of the graphics processors used on our benchmarks

| GPU Model | Memory | #Cores | CC | OS | CUDA |
|---|---|---|---|---|---|
| Tesla C1060 | 4GB | 240 | 1.3 | RHEL 5 | 5.0.35 |
| GeForce GTX 480 | 1.5GB | 448 | 2.0 | RHEL 6 | 4.2.9 |
| Tesla C2050 | 3GB | 448 | 2.0 | RHEL 6 | 4.2.9 |
| GeForce GTX 690[1] | 2GB | 1536 | 3.0 | RHEL 6 | 5.0.35 |
| Tesla K20c | 4.8GB | 2496 | 3.5 | RHEL 6 | 5.0.35 |

Additional file 2 — Sample additional file title
Additional file descriptions text.

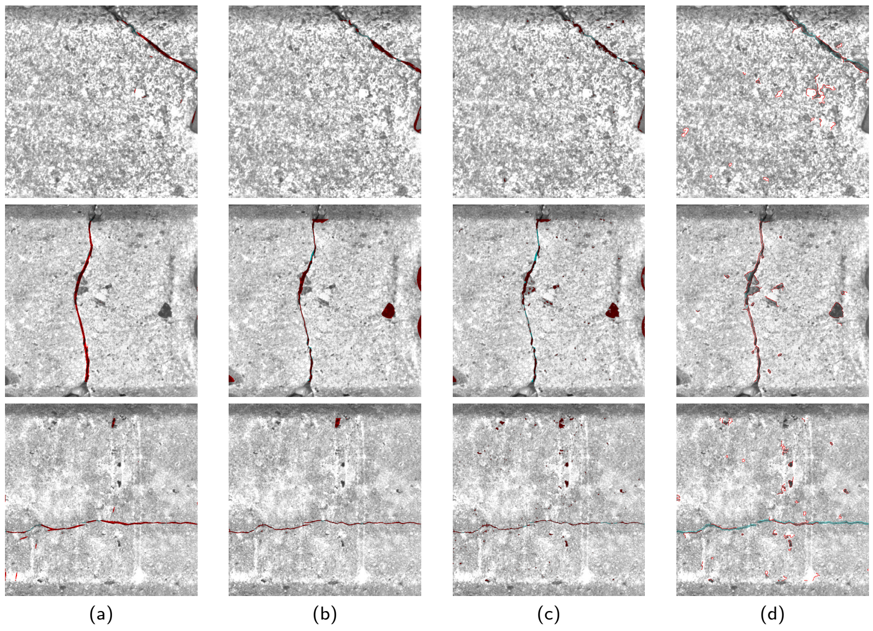

**Figure 5 Crack detection results.** (e) using shearlet coefficients (Shearlet-C) (f) using thresholding in the image reconstruction using shearlets (Shearlet-I) (g) using intensity thresholding in the original image (h) using Canny edge detection. All results are generated at peak $F_2$ score

**Table 3** Comparison of processing times for denoising a single precision $512 \times 512$ image on a multicore CPU using 4 CPU cores vs. a GeForce GTX 690 GPU.

| Step | 4-core CPU | | GTX 690 GPU | |
|---|---|---|---|---|
| | time (s) | % time | time (ms) | % time |
| Laplacian pyramid | 2.787 | 31.6% | 18.282 | 47.3% |
| Directional filters | 4.386 | 49.7% | 18.350 | 47.5% |
| Hard threshold | 0.375 | 4.2% | 1.967 | 5.1% |
| Other | 1.281 | 14.5 % | 0.063 | 0.2% |
| TOTAL TIME | 8.829 seconds | | 38.662 msec | |

**Table 4** Comparison of detection performance for different crack detection algorithms.

| Image | Method | AUC | $F_1$ score | $PD\|_{PF=10^{-3}}$ | $PD\|_{PF=10^{-4}}$ |
|---|---|---|---|---|---|
| 1 | Shearlet-C | **0.99915** | **0.79916** | **0.8398** | **0.6746** |
| | Shearlet-I | 0.99908 | 0.65810 | 0.7140 | 0.4247 |
| | Intensity | 0.99874 | 0.73188 | 0.7411 | 0.5722 |
| | Canny | 0.94457 | 0.27752 | 0.2114 | 0.1099 |
| 2 | Shearlet-C | **0.99999** | **0.98841** | **0.9989** | **0.9895** |
| | Shearlet-I | 0.99557 | 0.62705 | 0.4837 | 0.3964 |
| | Intensity | 0.99037 | 0.55404 | 0.4371 | 0.3342 |
| | Canny | 0.99043 | 0.81787 | 0.6425 | 0.4462 |
| 3 | Shearlet-C | 0.99934 | 0.76418 | 0.8368 | 0.5874 |
| | Shearlet-I | **0.99977** | **0.82353** | **0.9101** | **0.7098** |
| | Intensity | 0.99650 | 0.45992 | 0.0543 | 0.0000 |
| | Canny | 0.96248 | 0.19436 | 0.0000 | 0.0000 |

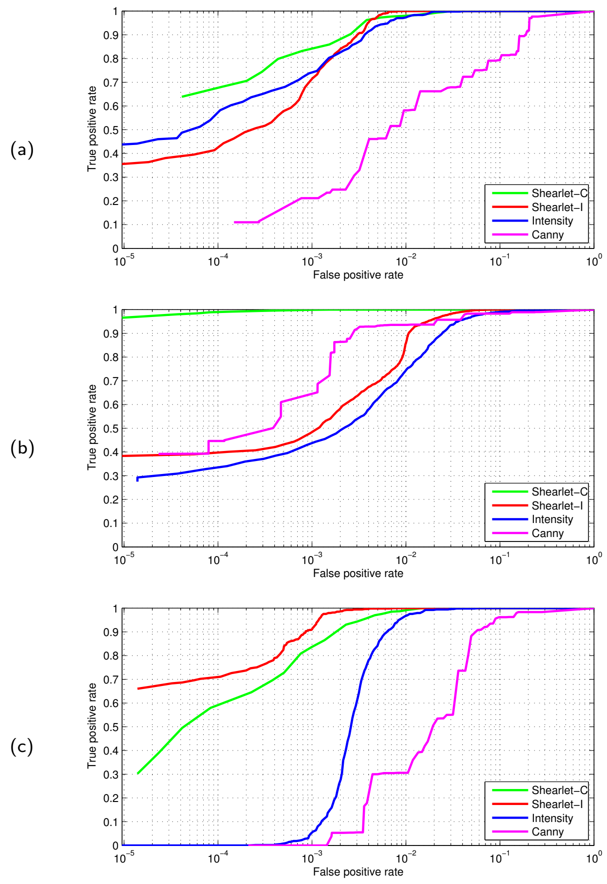

**Figure 6 ROC curves for crack detection.** (a) Image 1 (b) Image 2 (c) Image 3

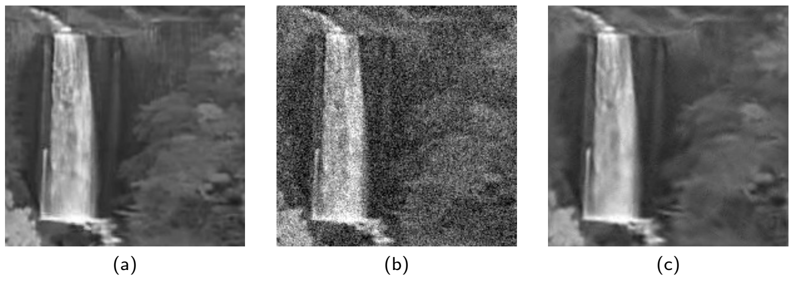

**Figure 7 Video denoising.** (a) Original video frame (b) Noise added (c) Denoised frame

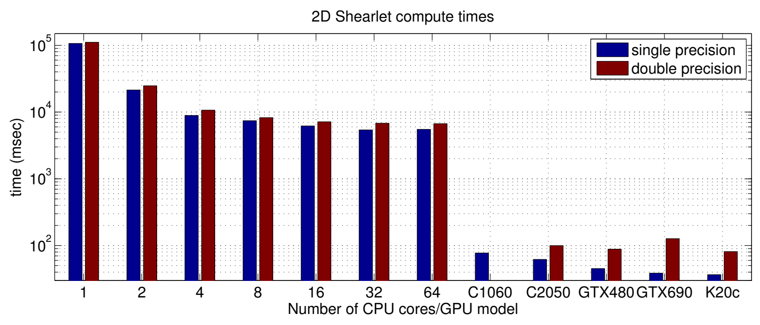

**Figure 8** Comparison of CPU vs GPU run times for denoising a $512 \times 512$ image using shearlets.

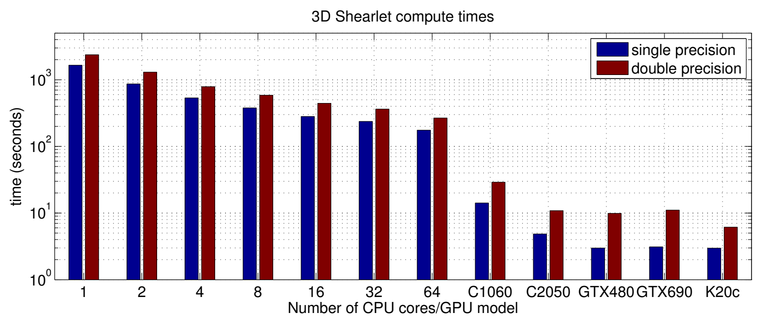

**Figure 9** Comparison of CPU vs GPU run times for denoising a $192^3$ video using 3D shearlets. Time includes all transfers between CPU and GPU.