Seppä, M. (2007). High-quality two-stage resampling for 3-D volumes in medical imaging. Medical Image Analysis, 11: 346-360.

Available online at www.sciencedirect.com
ScienceDirect
Medical Image Analysis 11 (2007) 346–360
MEDICAL IMAGE ANALYSIS
www.elsevier.com/locate/media

High-quality two-stage resampling for 3-D volumes in medical imaging

Mika Seppä *

Brain Research Unit, Low Temperature Laboratory, P.O. Box 2200, FIN-02015 HUT, Espoo, Finland

Received 10 May 2006; received in revised form 18 January 2007; accepted 21 March 2007
Available online 30 March 2007

Abstract

This paper introduces a simple method of two-stage resampling where Fourier-domain up-sampling is followed by traditional resampling. Practical aspects as well as efficient implementation techniques are considered. A new version of pruned FFT algorithms to calculate the up-sampling stage is also introduced. The suggested two-stage resampling method provides very high-quality results exceeding those of the previous algorithms. It excels with higher dimensional datasets due to its ability to employ small-support kernels. The applied FFT algorithms make the method most efficient with dataset sizes of powers of two. These reasons and the importance of minimal resampling artifacts make the suggested method especially suitable for 3-D volumes in medical imaging. Furthermore, for repeated uses, only the second stage is recalculated allowing an increase in performance for motion correction applications in functional magnetic resonance imaging (fMRI), for example.

Keywords: FFT; Resampling; MRI

1. Introduction

Resampling is a common operation in all signal and image processing applications. Available methods vary in their computational complexity, speed, and quality. The fastest methods employ nearest neighbor and linear interpolation. Theoretically optimal but computationally one of the most expensive methods is full-width sinc interpolation that is applied in this paper.

Sinc interpolation can be used to perfectly restore the underlying periodic and band-limited continuous signal from digitized samples. This restoration assumes, of course, that the sampling frequency has been high enough to capture all signal frequencies. The reason why full-width sinc interpolation is not so widely used lies in its computational complexity and in the adverse effects that noise can cause (i.e. noise ringing). As the sinc function extends to infinity, theoretically all the samples have an effect on each output value. A common approach is to use windowed, instead of full-width, sinc interpolation so that a finite-support interpolation kernel can be used.

In addition to windowed sinc and linear interpolations, other common finite-support methods include cubic (Keys, 1981), spline (Hou and Andrews, 1978), and polynomial (Meijering et al., 1999) interpolation. Surveys by Lehmann et al. (1999, 2001) and by Meijering et al. (2001) provide a comprehensive comparison of different interpolation kernels. Specialized methods also exist for certain image processing tasks. For example, in 2-D image rotation the rotation matrix can be decomposed into two or three one-dimensional image shears (Catmull and Smith, 1980; Tsuchida et al., 1987; Unser et al., 1995). The interpolation kernels are typically separable, but non-separable two-dimensional cubic kernels have been studied as well (Reichenbach and Geng, 2003). In shape-based interpolation (Goshtasby et al., 1992; Grevera and Udupa, 1996, 1998), the interpolation process itself is influenced by the underlying data.

The so-called generalized interpolation methods (Blu et al., 1999, 2001; Thévenaz et al., 2000) represent interesting and conceptually new advances in interpolation. They replace the normal interpolation kernel by a non-

* Tel.: +358 9 4516150; fax: +358 9 4512969.
E-mail address: mika.seppa@hut.fi

1361-8415/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.media.2007.03.002

interpolating one and prefilter the data. A very original improvement to the well-known linear interpolation has been also introduced (Blu et al., 2004) where simple shift improves the interpolation results.

In this paper, I explore the practical aspects of a simple two-stage resampling method where the original data are first up-sampled in Fourier domain to obtain a high-quality intermediate stage. This Fourier domain up-sampling is equivalent to employing full-width sinc kernel in signal domain. Then follows a typical compact-support resampling step to calculate the final result. Fast implementation as well as other practical issues are discussed. The results prove the proposed two-stage resampling method to be a very applicable solution providing high-quality resampling, especially for 3-D volumes in medical imaging.

2. Methods

This section introduces the resampling methods used in this work. First, the traditional interpolation and the generalized interpolation are briefly described. Then the up-sampling in Fourier domain is explained and its practical implementations and limitations are discussed. The last part of this section explores the idea of two-stage resampling.

The main novelties of this work are the combination of Fourier domain up-sampling with normal resampling, the details that need to be considered, and the efficient implementation of the up-sampling stage. This section also introduces a novel modification to the fast Fourier transform (FFT) algorithm that offers faster processing in the special case of up-sampling. Therefore, the main focus here is in the Fourier domain processing.

All of the interpolation kernels studied in this work are separable and therefore the theory can be explored easily in one dimension. For practical applications, the number of original and final samples is assumed to be roughly the same. Thus, for example, a 3-D volume is resampled to yield a new 3-D volume of approximately the same size. Applications such as motion correction in functional magnetic resonance imaging (fMRI) and volume registration in general are good examples of these cases. However, if only one or few output samples would be required, some of the methods here would not be efficient as they preprocess the full dataset.

2.1. Traditional interpolation

The problem of interpolating a discrete data sequence f_k can be written as a product with an interpolating kernel function $\varphi_{\text{int}}(x)$ (Thévenaz et al., 2000) giving for $x \in \mathbb{R}$

$$f(x) = \sum_{k \in \mathbb{Z}} f_k \varphi_{\text{int}}(x - k). \tag{1}$$

For kernel $\varphi_{\text{int}}(x)$ to be interpolating one, it must satisfy $\varphi_{\text{int}}(0) = 1$ and $\varphi_{\text{int}}(k) = 0$ for all integer $k \neq 0$.

Support is the most crucial property of $\varphi_{\text{int}}(x)$ affecting the computational efficiency of the traditional interpolation. Support is the range of x where the kernel $\varphi_{\text{int}}(x)$ assumes non-zero values and its size specifies how many samples and how many calculations are required for each output value. If the size of the support is S, the number of input samples consulted and the number of multiplications needed per output value is S^D for D dimensional data. For example, in three dimensional case, each output value requires $2^3 = 8$ multiplications for linear interpolation (support 2) and $4^3 = 64$ multiplications for cubic interpolation (support 4). The size of the support also affects the interpolation quality so that larger support allows higher quality results.

The computational complexity of evaluating $\varphi_{\text{int}}(x)$ for a given x also affects the efficiency of the traditional interpolation. Typically, $\varphi_{\text{int}}(x)$ is polynomial and the differences in speed are negligible between different kernels with the same support size. However, some interpolators which use trigonometric functions in their kernel are computationally more costly; the windowed sinc interpolation is a good example of these methods.

For the algorithmic implementation of the traditional interpolation, the values outside the dataset are typically assumed to be constant (zero). Other possibilities are to assume that the dataset is mirrored or repeated (see Fig. 1). The selected implementation affects values that are computed at the edge and outside the dataset. Mirroring allows the smoothest continuation of the signal and thus causes the smallest edge effects. On the other hand, in some applications it is useful to clearly see when output contains samples from outside the original dataset. These applications typically apply implementations that assume constant (zero) outside values.

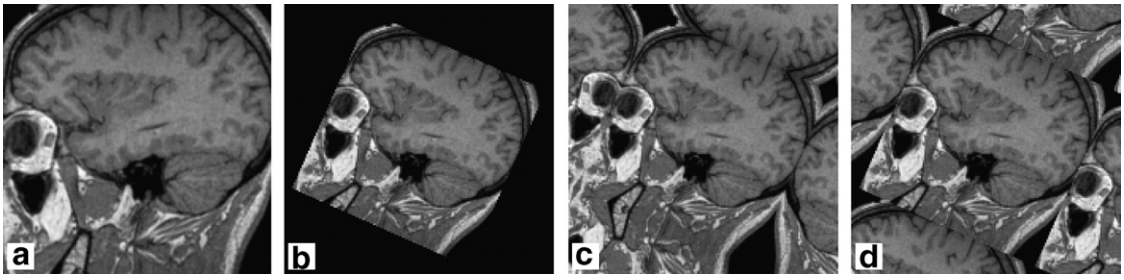

Fig. 1. Examples of handling samples outside dataset boundaries. Original image (a) is rotated and scaled down so that the results will contain regions outside the dataset. The panels are for zero filling (b), mirror boundary extension (c), and dataset repetition (d).

For more information about traditional interpolation and for comparisons of different kernels with several support sizes, see surveys by Lehmann et al. (1999, 2001) and by Meijering et al. (2001).

2.2. Generalized interpolation

Similarly as above, the generalized interpolation (Blu et al., 1999, 2001; Thévenaz et al., 2000) is also written as a product

$$f(x) = \sum_{k\in\mathbb{Z}} c_k \varphi(x-k). \tag{2}$$

Here, the original samples f_k of Eq. (1) are replaced by prefiltered samples c_k, and the interpolating kernel $\varphi_{\text{int}}(x)$ is replaced by a general kernel $\varphi(x)$. The prefiltering creating c_k is linked to $\varphi(x)$ by the interpolation condition $f(k) = f_k$, i.e. the function $f(x)$ must reproduce the original samples.

The benefit of the generalized interpolation is that a small-support kernel $\varphi(x)$ can be used for computational efficiency. Due to the prefiltering, the generalized interpolation has information available from larger area than a traditional interpolation of equal support. As shown by Blu et al. (1999), the prefiltering can be implemented efficiently as succession of forward and backward recursive filters. Thus, the additional time for the prefiltering step is minimal. Furthermore, if repeated computation is performed on a dataset, the data can be prefiltered just once and then kept in this prefiltered form.

Due to the required prefiltering, also spline interpolation (Hou and Andrews, 1978; Unser et al., 1993a,b; Unser, 1999) and the shifted linear interpolation (Blu et al., 2004) fall into this generalized interpolation category. Maximal-order-minimal-support (MOMS) functions made of linear combinations of B-splines and its derivatives are an interesting class of functions. These functions were introduced in Blu et al. (2001) and were shown to have minimal support for a given interpolation accuracy. The optimal versions were labeled as OMOMS which are also employed in this study.

Prefilterings employed in the generalized interpolation typically amplify high frequencies and thus discontinuities at the edges of the dataset may cause artifacts. As suggested by Blu et al. (1999), it is reasonable to use mirror boundary extension (Thévenaz et al., 2000) for the generalized interpolation to maximize continuity at the edges (Fig. 1c). The selected boundary extension method must be used both by the prefiltering implementation and by the actual kernel multiplication algorithm for consistent results.

2.3. Up-sampling in Fourier domain

The most efficient ways to employ full-width sinc interpolation utilize Fourier transforms. One alternative is to use sub-sample shifts which can be implemented as phase-shifts in Fourier domain. Another class of methods use zero filling in Fourier domain combined with pruned fast Fourier transform (FFT) algorithms. Below, both of these approaches are described after a short introduction of some basic concepts. Details about the new fast sinc magnification (FSM) implementation of the pruned FFT algorithm are then described. Finally, some notes are given about the use and limitations of all these methods.

2.3.1. DFT

The discrete Fourier transform (DFT) $G(u)$ of an N sample array $g(x)$ and the inverse transform are defined as

$$G(u) = \mathscr{F}\{g(x)\} = \sum_{x=0}^{N-1} g(x) \mathrm{e}^{-\mathrm{i}2\pi ux/N} \tag{3}$$

$$g(x) = \mathscr{F}^{-1}\{G(u)\} = \frac{1}{N}\sum_{u=0}^{N-1} G(u) \mathrm{e}^{\mathrm{i}2\pi ux/N} \tag{4}$$

where $\mathrm{i} = \sqrt{-1}$, and $\mathscr{F}\{\ldots\}$ and $\mathscr{F}^{-1}\{\ldots\}$ denote the Fourier and the inverse Fourier transforms, respectively. The Fourier transform and the inverse Fourier transform have several well-known properties (Gonzalez and Woods, 1992) such as separability, translation, periodicity, and conjugate symmetry.

2.3.2. Zero-filled up-sampling

The Fourier-transformed array $G(u)$ holds the transform for zero frequency at position $u = 0$ and the transforms for positive frequencies at $u = 1,\ldots,u = N/2 - 1$ in increasing order. Similarly, positions $u = N-1, u = N-2,\ldots, u = N/2+1$ hold the transforms for negative frequencies so that $u = N-1$ is the smallest absolute frequency. The transform at position $u = N/2$ contains aliased data for the most positive and the most negative frequency. In general, this folding is of no concern as those frequencies are actually the same ($\mathrm{e}^{-\mathrm{i}\pi} = \mathrm{e}^{\mathrm{i}\pi}$), but for up-sampling the data array is extended in frequency domain and the aliased frequencies need to be separated.

Fig. 2 illustrates the up-sampling by zero filling in the frequency domain. The positive frequencies map to the positive frequencies in the extended array and the negative frequencies, respectively, to the negative frequencies at the end of the new array. The value at location $u = N/2$ is mapped with both the positive and the negative frequencies. This value can be handled by three different ways (Yaroslavsky, 1997) while maintaining the conjugate symmetry of the transform. The value can be set explicitly to zero, which naturally alters the data and the operation cannot be inverted, or it can be mapped at whole or at half with the positive and negative frequencies. The methods in this paper use the way of mapping half of the transformed value at $u = N/2$ with both the frequencies (Fig. 2). This way is considered to be the best (Yaroslavsky, 1997) as it produces the smallest boundary effects and it also maintains the total power (sum of absolute values) of the transform constant.

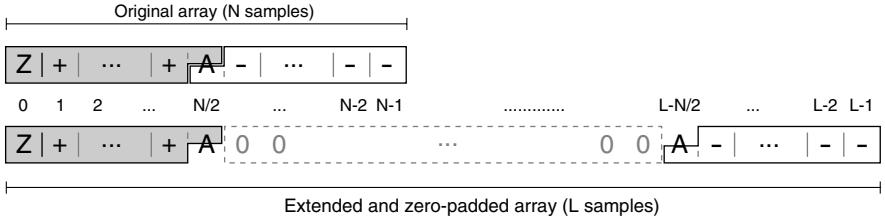

Fig. 2. Illustration of extending the original array of length N in frequency domain to a new array of size $L > N$. Position 0 holds the value for zero frequency (Z), positions *1* through $N/2$ in the original array the positive frequencies, and positions $N/2$ through $N-1$ in the original array the negative frequencies. Half of the value at position $N/2$ (marked A) goes with the positive part and half with the negative part. The values between these parts in the new array are set to zero.

The zero-filled frequency data $G'(u)$ created from $G(u)$ is now an array of L samples wide. The inverse transform $g'(x) = \mathscr{F}^{-1}\{G'(u)\}$ (Eq. (4)) uses thus scaling term $1/L$. For maintaining the original amplitude of the up-sampled signal, this result $g'(x)$ must be scaled by L/N where N is the size of the original signal array $g(x)$.

2.3.3. Shifted DFT (SDFT) algorithm

Up-sampling can also be performed by sub-sample shifts (Du et al., 1994; Yaroslavsky, 1997). The signal is first Fourier-transformed and the signal-space shift is then applied as a multiplication by an appropriate phase-shift in the Fourier domain. Inverse Fourier transform yields the shifted signal. The final up-sampled signal is composed of several sub-sample shifted signals. For example, the original signal and versions shifted by 1/3 and 2/3 samples can be composed together to form a up-sampled signal with magnification factor of 3.

With this method, it is crucial to note that the typically seen Fourier space phase-shift equation $H_d(u) = \exp(\mathrm{i}2\pi ud/N)$ is valid only for shifts of discrete number d of samples. Sub-sample shift s in our case, when the sample at position $N/2$ (if N is even) is handled as mentioned above (Fig. 2), can be performed with a Fourier-space transfer function (Yaroslavsky, 1997)

$$H_s(u) = \begin{cases} \exp(\mathrm{i}2\pi us/N) & \text{if } 0 \leqslant u < N/2 \\ \cos(\pi s) & \text{if } u = N/2 \\ \exp(-\mathrm{i}2\pi(N-u)s/N) & \text{if } N/2 < u \leqslant N-1 \end{cases} \tag{5}$$

Notice that $H_s(u)$ is conjugate symmetric, i.e. $H_s(N-u) = H_s(u)^*$ where $H_s(u)^*$ marks the complex conjugate of $H_s(u)$. In case N is odd, the middle term $\cos(\pi s)$ is left out in Eq. (5).

2.3.4. Pruned FFT algorithms

As already mentioned, up-sampling can be performed by Fourier transforming the sample array, by extending and zero filling the transformed array in the Fourier domain (Fig. 2), and by inverse transforming this new array back to signal domain. The introduced zeros in the frequency domain lead to unnecessary multiplications and additions in the normal FFT algorithms. To remediate this drawback, Markel (1971) developed a pruned decimation-in-frequency (DIF) type FFT algorithm. Similarly, Skinner (1976) introduced a slightly more efficient version for the decimation-in-time (DIT) type FFT algorithm. These two methods can be combined for both input and output pruning (Sreenivas and Rao, 1979; Jaroslavski, 1981; Smith and Nichols, 1988) if only a part of the output is needed. Originally, pruned FFT algorithms were developed for computation of high-resolution spectra and worked on end-padded data. For up-sampling, center-padding is used and modified versions of the pruned FFT algorithms have been created for this purpose (Nagai, 1986; Smit et al., 1990). A generalized method for pruning FFT type transforms has also been introduced (Rangarajan and Srinivasan, 1997).

2.3.5. FSM – Fast sinc magnification algorithm

Fig. 3 illustrates the differences between typical pruned FFT versions and the FSM algorithm introduced here. Case (a) shows the normal way of taking Fourier transform either with a DIF or DIT FFT algorithm which differ in whether the necessary bit-reversal reordering (Press et al., 1988) is performed after (DIF) or before (DIT) the recursive "butterfly" processing. After FFT, the array is expanded and zero filled as shown in Fig. 2 and then inverse transformed with pruned FFT algorithm of either DIF or DIT type. To better illustrate the new FSM algorithm, Fig. 3b shows the same as case (a) with the exception of specifically using a DIF-type algorithm for forward transform and a DIT-type algorithm for inverse transform.

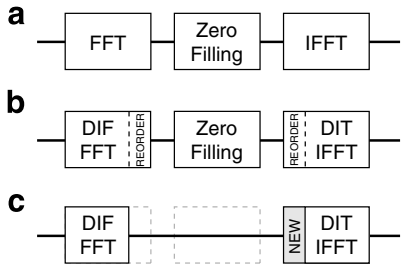

Fig. 3. Schematic view of the up-sampling in Fourier domain. Cases (a) and (b) show the usual way of separate Fourier transforms and zero filling. The inverse transform (IFFT) uses pruned FFT algorithms to skip unnecessary calculations with zeroes. Case (c) shows the proposed FSM (fast sinc magnification) algorithm. See text for more details.

The included reordering phases are illustrated in the algorithm boxes.

Now, the proposed FSM algorithm works as shown in Fig. 3c. The data are first transformed with a DIF FFT algorithm without the reordering phase. Thus, the transform in Fourier domain is left in bit-reversed order. The explicit array extension and zero filling are excluded and performed implicitly by a modified DIT-type FFT algorithm. The modification, replacing the leading bit-reversal reordering of DIT, performs the array extension, zero-filling, possible transfer function multiplication, and the pruned FFT rounds in this bit-reversal ordered Fourier domain. This new part (gray box labeled "new" in Fig. 3c) is followed immediately by the normal DIT-type FFT butterfly rounds that finish the transform. C-language code for the algorithm is provided on-line (Seppä, 2007).

2.3.6. Practical issues

The FFT, pruned FFT, SDFT, and FSM algorithms all process complex-valued arrays. Since the explained up-sampling methods maintain conjugate symmetry in Fourier space, a purely real signal is up-sampled into a purely real signal. Respectively, a purely imaginary signal is up-sampled into a purely imaginary signal. Due to linearity of the Fourier transform, two separate real-valued signals can be simultaneously up-sampled by encoding them into the real and imaginary parts of the complex-valued array. This well-known trick reduces the amount of calculations to half.

Another issue to consider is how the multi-dimensional data are divided into one-dimensional magnifications. Interestingly, the total computational cost depends on the processing order of the dimensions if the data extents are different. For example, let us consider a simple 2-D case with image width w, image height h, and magnification factor M along both dimensions. To simplify the calculations, let us assume that the computational cost for the algorithm used would be relative to $N\log N$ where N is the original size of the array to be magnified. For the total computational costs we have then

$$\begin{aligned} C_\mathrm{r} &= h\cdot w\log w + wM\cdot h\log h \\ &= hw(\log w + \log h + (M-1)\log h) \\ C_\mathrm{c} &= w\cdot h\log h + hM\cdot w\log w \\ &= hw(\log w + \log h + (M-1)\log w) \end{aligned}$$

These equations give the relative total computational effort when magnifying takes first place along the row direction (C_r) or along the column direction (C_c). As can be seen, the cost is smaller if magnification is performed first along the dimension where the data extent is larger. The calculation and comparison of the relative costs are more difficult in a general case with high-dimensional data, with different magnification factors along the dimensions, and with a more accurate cost function for the algorithms. Furthermore, the computer memory and CPU cache latencies also have an effect on performance when accessing data from big arrays.

Tests in 2-D and 3-D using different data sizes and magnification factors along the axes revealed that no simple decision logic can be formulated to choose the best processing order of dimensions. In most of the cases, the memory speed was the key factor and only for highly asymmetric data the differences in the relative computational effort dictated the optimal order.

For optimal CPU cache utilization, it is reasonable to access memory in close-by regions as often as possible. Thus, the best way is to start magnification along the dimension that has its samples most scattered in the memory. The amount of data increases after each processed dimension and so the final magnification, with the amount of data at its maximum, is performed along the dimension where the samples are closest to each other.

2.3.7. Limitations

Up-sampling in the Fourier domain bears some limitations that arise from the FFT algorithms used. First of all, the data array sizes need to be powers of two which naturally can be circumvented by the normal FFT trick of (zero) padding the array to the next suitable size. This padding, of course, increases the computational burden as the transformed array gets longer. Also, the magnification factors for up-sampling need to be powers of two for FFT, pruned FFT, and FSM algorithms and an integer for SDFT. Basically, fractional magnification factors can be also used by discarding part of the up-sampled data. For example, factor 5/3 can be realized with SDFT by first magnifying by 5 and then selecting every third sample.

With Fourier up-sampling, the sample grid in the 2-D or higher dimensional case is always aligned along the original axes. Free placement and sizing of the grid is not possible. The FFT-based up-sampling methods also typically calculate the result for the whole data array even if the value in only one or a few locations would be needed. The output-pruned FFT algorithms have been developed to speed-up these cases. Nevertheless, this property can be partly seen as a limitation of the full-width sinc kernel as every output value depends on all the data samples, not just the local ones.

FFT-based methods share the Fourier transform property that the up-sampled data array repeats itself (periodicity). Thus, in practice, the output values at one edge of an image depend on the values at the opposite edge. For most of the images this is a clear drawback, although MRI data can be considered as an exception. Due to the nature how MR images are originally created as an inverse Fourier transform, these images can be considered to carry this property of 'wrapping-around' already inherently. Thus, reverting back to frequency domain, where the data were collected, and performing operations there is easily justified.

2.4. Two-stage resampling

The two-stage resampling studied in this work consists of an up-sampling stage followed by a normal resampling stage. The theoretically perfect sinc kernel is employed efficiently in Fourier domain with either SDFT or FSM up-sampling algorithms. This intermediate stage is then used for a compact-support normal resampling that allows free placement of the resampling grid and thus circumvents the limitations of the Fourier space up-sampling methods mentioned above. Due to the increased sampling rate, relatively small-support kernels can be used for speed and still the outcome is of very high quality.

Fig. 4 visualizes the middle-stage of the two-stage resampling. On the left, an illustrative 4×4 image (gray pixels) is to be resampled into rotated 5×5 image. The white frame outlines the position of the new resampled image. White dot marks the center of a new pixel and the black dots mark the corresponding surrounding pixels in the original image. For example, these pixels would be used by a support-2 (per dimension) method, such as linear interpolation, to calculate the output pixel value. This output value is a weighted sum of the original pixel values, and the weights are drawn from the selected kernel function. Naturally, kernels with larger support size would use more surrounding pixels. On the right, the middle-stage of the two-stage resampling is shown. The original image has been up-sampled by a factor of 2 along both dimensions. From this 8×8 data, the final 5×5 result is calculated with a normal resampling method, just as above.

2.4.1. Up-sampling stage

The first stage performs up-sampling in Fourier domain with discrete magnification factor M. The choice of the algorithm is FSM or SDFT for power-of-two magnification factors and SDFT for others. All the practical issues discussed above for SDFT and FSM apply here as well.

In particular, the dimensions of the dataset are processed in the order dictated by the memory and cache usage. Along each dimension, the data are magnified two rows at a time by encoding them into the real and imaginary parts of the complex-valued array. If necessary, the array is padded into a power-of-two size. The padding contains a linear slope between the last and the first sample to decrease possible edge effects. After up-sampling, the resulting two rows are copied into respective places of the magnified dataset, discarding the possible padding at the end.

One further detail to consider is the exact sample placement of this up-sampling stage. With Fourier-space zero-filling methods, a sample that is originally at position x maps to position Mx in the up-sampled signal, where M is the discrete magnification factor. Fig. 5a illustrates this in a case where 2-D image is magnified with factor 2 along each axis. The samples are assumed to span from left to right and from top to bottom. Thus, the original sample value in each pixel maps to the top-left sub-pixel (show in gray) of the magnified pixel. When the magnification factor is increased, this sub-pixel approaches the black dot shown in the upper left corner of the original pixels. This dot is the stationary point and thus the assumed pinpoint location for the original pixel value.

Fig. 5b illustrates a situation where the stationary point is considered to be in the center of the pixel. Therefore, with even values of magnification factor M, none of the new pixels will have exactly the same value as the original pixel. The underlying continuous function reaches that value in the corner of middle pixels. The gray area in the lower row shows the location where the upper left corner pixels of the normal zero-filled up-sampling should be translated to for obtaining the desired effect. The amount of translation is $M/2 - 1/2 = (M - 1)/2$ new up-sampled pixels to right and down. This translation can be performed with Fourier-space transfer function during the zero-filled up-sampling. As non-discrete amount of translation results when M is even, Eq. (5) must be used for correct results.

Because Fourier transforms are periodic, the signal at one edge affects the up-sampled values at the other edge. For symmetry, the case shown in Fig. 5b is the best unless the resampling at the second stage employs an algorithm that considers its data periodic, too. In any case, the selected way of positioning image details in the up-sam-

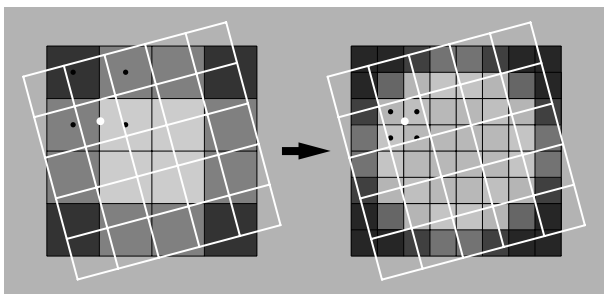

Fig. 4. Illustration of the middle-stage of the two-stage resampling.

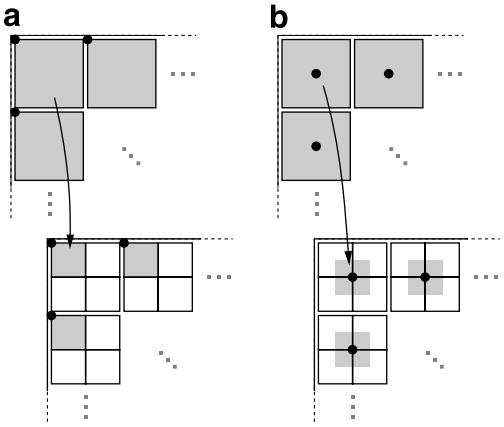

Fig. 5. Position of image details in different cases. The stationary point giving the exact location of image details is in the upper left-hand corner for case (a) and in the middle of the original pixel for case (b).

pling stage must be taken into account when positioning the second stage resampling grid.

2.4.2. Resampling stage

This second stage can use either traditional interpolation kernels or kernels of the generalized interpolation method. For the generalized interpolation, the prefiltering step can be conveniently performed with virtually no extra cost as a transfer function multiplication in the Fourier domain during the up-sampling stage. However, due to the periodicity of the Fourier transform, this method assumes implicitly that the original data are periodic and wrap around. Thus, for correct results, the multiplication algorithm applying the generalized interpolation kernel must treat the data as periodic, not as mirrored which is typical with sample-space prefiltering.

For traditional interpolation kernels, the algorithm can use either periodicity assumption or constant (zero) values outside the dataset. In most cases the constant value is sensible as it allows the detection of samples coming outside of the dataset. If a constant-value algorithm is used, the up-sampling stage must use the symmetrical positioning choice illustrated in Fig. 5b.

3. Results

This section compares several previously known methods with the two-stage resampling method of this study. First, the speeds of FSM and SDFT algorithms are explored. Then, the speed and quality of two-stage resampling is tested followed by an analysis of resampling errors.

3.1. Up-sampling

Table 1 compares different methods for Fourier domain up-sampling. Method labeled FFT uses normal FFT routines without pruning. FSM employs the pruning algorithm introduced in this study and SDFT is the shifted-DFT method. All the methods use similar algorithm design with tabulated trigonometric functions to allow reasonable comparison.

Unfortunately, tests against previously used pruned FFT methods were not possible since their implementations were not readily available. The only speed enhancement of FSM compared with other pruned FFT methods is its lack of Fourier-space reordering steps. To estimate the probable speed difference, columns labeled FSM* in Table 1 show the times for FSM including two extra reordering steps.

Method SDFT* is SDFT without data centering which avoids one round of Fourier transforms and is therefore faster than centering SDFT. However, this method cannot be used with all second-stage resamplers in the two-stage resampling method and, as such, has limited use.

The tests were run for 1–3 dimensional datasets with different sizes and different magnification factors. The exponents for the size give the dimensions of the dataset and the size was equal along each dimension. Test round with each algorithm was repeated multiple times to more accurately time a single round. The right-most set of columns in Table 1 give the relative differences in computation time between the other methods and the FSM. Positive value means that the compared method uses more time (i.e. is slower) than the FSM and negative means the opposite.

3.2. Two-stage resampling

The speed and quality of two-stage resampling was explored with 2-D images and 3-D volumes. In both cases, the data went through 15 successive resampling steps where the output from the previous step was used as the input for the next step. In each step, the resampled dataset size was the same as for the original and all computations were performed in floating-point format to minimize quantization effects. The computation times shown are for a single resampling step run on a 3 GHz Pentium 4 Linux workstation (512 kB L2 cache) with 2 GB RAM.

After the 15th resampling step, the 2-D and 3-D data were back in their original alignment and the errors induced by all resampling steps together were assessed with signal-to-noise ratio (SNR). In addition to SNR, also root-mean-square (RMS) error is provided for Lena (2-D) image to facilitate comparison to other published resampling methods not used in this study. The measure used for the SNR is

$$\mathrm{SNR} = 10\log_{10}\left(\sum s_i^2 / \sum e_i^2\right) \tag{6}$$

where s_i is the original sample, e_i is the error between the original and the final output sample, and the summations go through all the compared samples i. Because the resampling steps contained rotations, information in the dataset corners was lost in the successive steps as the next dataset did not totally cover the previous one. Due to this reason and to avoid boundary effects, the final comparison was performed only on a central portion of the data.

The 2-D images were a 256×256 T_1-weighted MR image (T_1) and five 512×512 images: T_2-weighted MRI (T_2), proton-density weighted MRI (PD), computed tomography image (CT), synthetic image (Syn.), and a gray-scale photograph (Lena) typically used in image processing tests. All the images experienced 24-degree rotations that together account for a full rotation. Fig. 6 shows the synthetic image that consist of expanding sine waves with wavelength of 2.1 pixels in the center and 10 pixels at the edges. The left image is without any noise and the intensity oscillates between peak values 0 and 255. The right image has added Gaussian noise with standard deviation of 20. The rotation test uses the noiseless image and the effects of the noise are studied in the next section.

The 3-D datasets are a $64 \times 64 \times 31$ fMRI volume ($3.1\ \mathrm{mm} \times 3.1\ \mathrm{mm} \times 4\ \mathrm{mm}$ voxel size), a $256 \times 256 \times 190$ anatomical ($1\ \mathrm{mm} \times 1\ \mathrm{mm} \times 1\ \mathrm{mm}$ voxel size) T_1-weighted

Table 1
Up-sampling time comparisons for different datasets (Size) and magnifications (M)

Size	M	Time (ms)					Diff. (%)			
		FFT	FSM	FSM*	SDFT	SDFT*	FFT	FSM*	SDFT	SDFT*
256	2	0.076	0.070	0.073	0.096	0.057	8.4	4.8	27.4	−23.0
	4	0.116	0.100	0.103	0.182	0.140	13.6	3.0	44.9	28.6
	8	0.229	0.185	0.189	0.358	0.315	19.1	1.7	48.3	41.1
512	2	0.156	0.144	0.152	0.199	0.117	7.7	4.9	27.4	−23.9
	4	0.271	0.237	0.245	0.373	0.287	12.8	3.3	36.6	17.5
	8	0.507	0.418	0.425	0.732	0.643	17.6	1.6	42.9	34.9
1024	2	0.352	0.324	0.341	0.408	0.238	7.8	5.0	20.4	−36.5
	4	0.587	0.515	0.531	0.773	0.594	12.3	3.1	33.4	13.4
	8	1.13	0.949	0.963	1.52	1.33	16.1	1.5	37.5	28.8
2048	2	0.770	0.715	0.749	0.902	0.524	7.2	4.6	20.8	−36.4
	4	1.31	1.16	1.19	1.73	1.33	11.3	2.9	32.9	13.0
	8	2.65	2.27	2.29	3.38	2.97	14.4	1.0	32.9	23.6
4096	2	1.69	1.57	1.64	1.97	1.15	7.2	4.7	20.6	−36.3
	4	3.03	2.70	2.77	3.68	2.86	10.7	2.4	26.6	5.3
	8	6.36	5.57	5.63	7.35	6.47	12.5	1.1	24.3	13.9
8192	2	3.87	3.60	3.75	4.39	2.59	7.1	4.2	18.0	−38.9
	4	7.23	6.50	6.64	8.24	6.35	10.2	2.2	21.2	−2.3
	8	31.3	27.7	27.9	17.3	15.3	11.5	0.9	−60.1	−80.2
128^2	2	4.43	3.76	4.12	3.98	2.62	15.2	8.8	5.5	−43.4
	4	15.9	13.2	13.7	14.2	12.0	16.9	3.7	7.0	−10.2
	8	53.0	41.7	42.5	45.8	42.0	21.3	1.7	9.0	0.5
256^2	2	23.7	21.2	22.7	21.9	16.2	10.4	6.7	3.4	−30.5
	4	68.1	58.0	60.3	60.5	51.0	14.9	3.8	4.1	−13.8
	8	255	205	210	193	176	19.6	2.2	−6.5	−16.7
512^2	2	106	96.4	102	99.3	74.2	8.7	5.8	2.9	−29.8
	4	325	281	292	261	220	13.6	3.8	−7.5	−27.8
	8	1150	945	967	834	760	17.7	2.3	−13.3	−24.4
32^3	2	19.8	15.6	17.1	17.7	12.4	21.4	8.7	12.1	−25.8
	4	107	77.7	81.7	91.6	76.2	27.3	4.9	15.2	−1.9
	8	708	482	494	598	544	31.9	2.5	19.3	11.4
64^3	2	182	156	169	165	120	14.3	7.8	5.4	−30.4
	4	966	771	808	850	714	20.1	4.5	9.2	−8.1
	8	6410	4860	4950	5550	5070	24.1	1.7	12.4	4.1

Differences show the relative change in times against the FSM method. The tests were run on a 3 GHz Pentium 4 (512 kB L2 cache) Linux workstation with 2 GB RAM.

MRI volume, a $512 \times 512 \times 58$ CT volume (0.7 mm × 0.7 mm × 3mm voxel size), and a synthetic $192 \times 192 \times 192$ volume (1 mm × 1 mm × 1 mm voxel size). The synthetic volume is similar to the synthetic 2-D image (Fig. 6) except that the waves are now expanding spheres in 3-D space. The wavelength increased from 2.1 voxels in the center to 5 voxels at the edges of the volume and no noise was added.

For the 3-D datasets, the first 14 resampling steps were pseudo-random and the 15th was calculated to take the volume back to perfect registration with the original one. Each pseudo-random transformation combined rotation with translations. The rotation was applied around a random 3-D rotation axis going through the center of the volume with the rotation angle uniformly distributed between 0 and 5°. The translations were along each axis with uniform distributions between −2 and 2 mm. The seed for the pseudo-random number generator was recorded to allow all the compared methods to apply exactly the same sequence of transformations.

Table 2 shows the results in numerical format for the 2-D images. Figs. 7 and 8 show graphical illustrations of the results for the 2-D and 3-D datasets, respectively. The methods are labeled in the following fashion. The methods employing traditional interpolation are nearest neighbor (NN), linear (LI), Keys cubic (CU), and windowed sinc (WS_4 and WS_6). The width of the Hamming window used

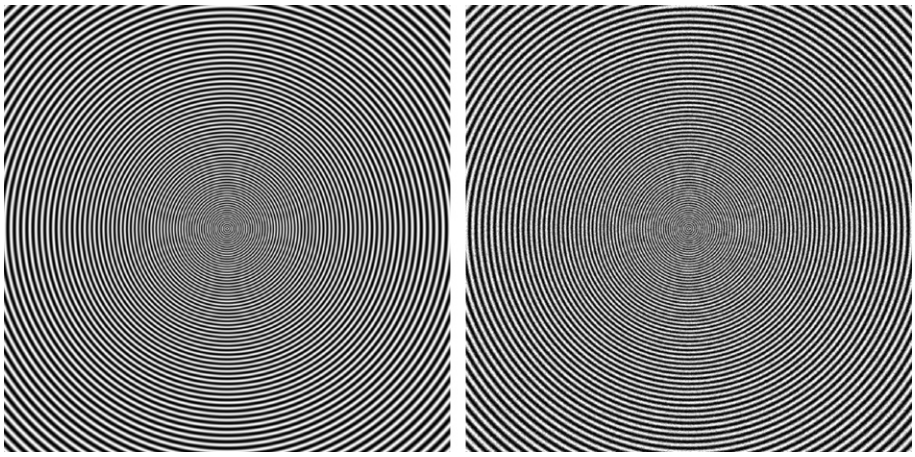

Fig. 6. Synthetic images used in the tests. Left one is without noise and right one contains Gaussian noise with standard deviation of 20.

was four samples for WS_4 and 6 samples for WS_6. The methods employing generalized interpolation are linear shifted (LS), B-spline (BS_x), and optimal MOMS ($OMOMS_x$) with the sub-indexes (x) showing the sizes of the support. The introduced two-stage resampling methods are labeled with letter–number–letter triplets. The first

Table 2
Comparison of different resampling methods for 2-D image rotation using one 256×256 pixel image (T_1) and five 512×512 pixel images (T_2, PD, CT, Synthetic, Lena)

Method	T_1		Time (ms)	T_2	PD	CT	Syn.	Lena	Lena
	Time (ms)	SNR (dB)		SNR (dB)	SNR (dB)	SNR (dB)	SNR (dB)	SNR (dB)	RMS
Traditional									
NN	4	12.1	16–19	14.0	19.5	15.7	3.2	18.0	16.6
LI	5	15.2	22–24	17.7	23.4	19.9	5.8	21.9	10.6
CU	11	18.7	47–50	23.7	30.0	29.6	11.5	28.6	4.9
WS_4	39	17.8	157–157	22.1	28.3	26.6	9.6	26.8	6.1
WS_6	54	21.3	210–215	28.4	34.3	36.3	17.9	32.8	3.0
Generalized									
LS	7	16.9	43–47	24.6	30.5	30.4	3.5	27.2	5.8
BS_4	18	21.6	97–101	28.9	34.7	36.7	18.1	33.1	2.9
$OMOMS_4$	19	23.6	100–103	33.6	38.5	41.8	24.1	35.8	2.2
BS_6	38	24.1	175–177	35.0	39.6	42.9	25.5	36.4	2.0
$OMOMS_6$	46	24.7	208–217	36.7	41.0	44.1	27.7	37.1	1.9
Two-stage									
F2N	26	15.7	119–122	18.8	24.4	21.0	7.6	22.9	9.5
F2L	29	19.0	127–132	23.4	29.4	27.4	11.1	27.9	5.3
F2S	29	25.5	129–135	37.2	43.5	42.5	26.7	38.1	1.6
F2C	35	26.4	149–156	37.7	43.5	44.8	27.7	38.8	1.5
$F2B_4$	37	26.9	156–160	43.9	50.9	48.9	42.4	40.3	1.3
$F2O_4$	37	26.5	158–162	44.0	51.1	48.6	48.8	40.0	1.3
S2N	27	15.7	120–123	18.8	24.4	21.0	7.6	22.9	9.5
S2N*	23	15.6	97–99	18.5	24.2	20.8	7.1	22.7	9.7
S2L	31	19.0	129–131	23.4	29.4	27.4	11.1	27.9	5.3
S2L*	26	19.0	104–111	23.4	29.4	27.4	11.1	27.9	5.3
S2S	31	25.5	132–135	37.2	43.5	42.5	26.7	38.1	1.6
S2C	37	26.4	151–156	37.7	43.5	44.8	27.7	38.8	1.5
S2C*	32	26.4	127–135	37.7	43.5	44.8	27.5	38.8	1.5
$S2B_4$	39	26.9	158–162	43.9	50.9	48.9	42.4	40.3	1.3
$S2O_4$	39	26.5	160–163	44.0	51.1	48.6	48.8	40.0	1.3
S3N	41	18.1	171–173	21.9	27.5	24.2	5.9	25.7	6.9
S3L	43	22.5	179–185	28.2	34.3	33.0	16.1	32.4	3.2
S3S	51	26.3	216–221	41.7	48.3	46.7	33.9	39.6	1.4
S3C	49	26.9	199–207	43.6	50.6	48.9	39.0	39.9	1.3
$S3B_4$	58	26.6	240–246	44.1	51.2	48.7	49.3	40.0	1.3
$S3O_4$	59	26.5	242–249	44.0	51.1	48.6	49.6	39.9	1.3

For each method, Time gives the calculation time for one rotation step. The data quality after all 15 resampling steps is shown with signal-to-noise ratio (SNR) and also with root-mean-square error (RMS) for Lena image. Times for the 512×512 images are collected as a single time range.

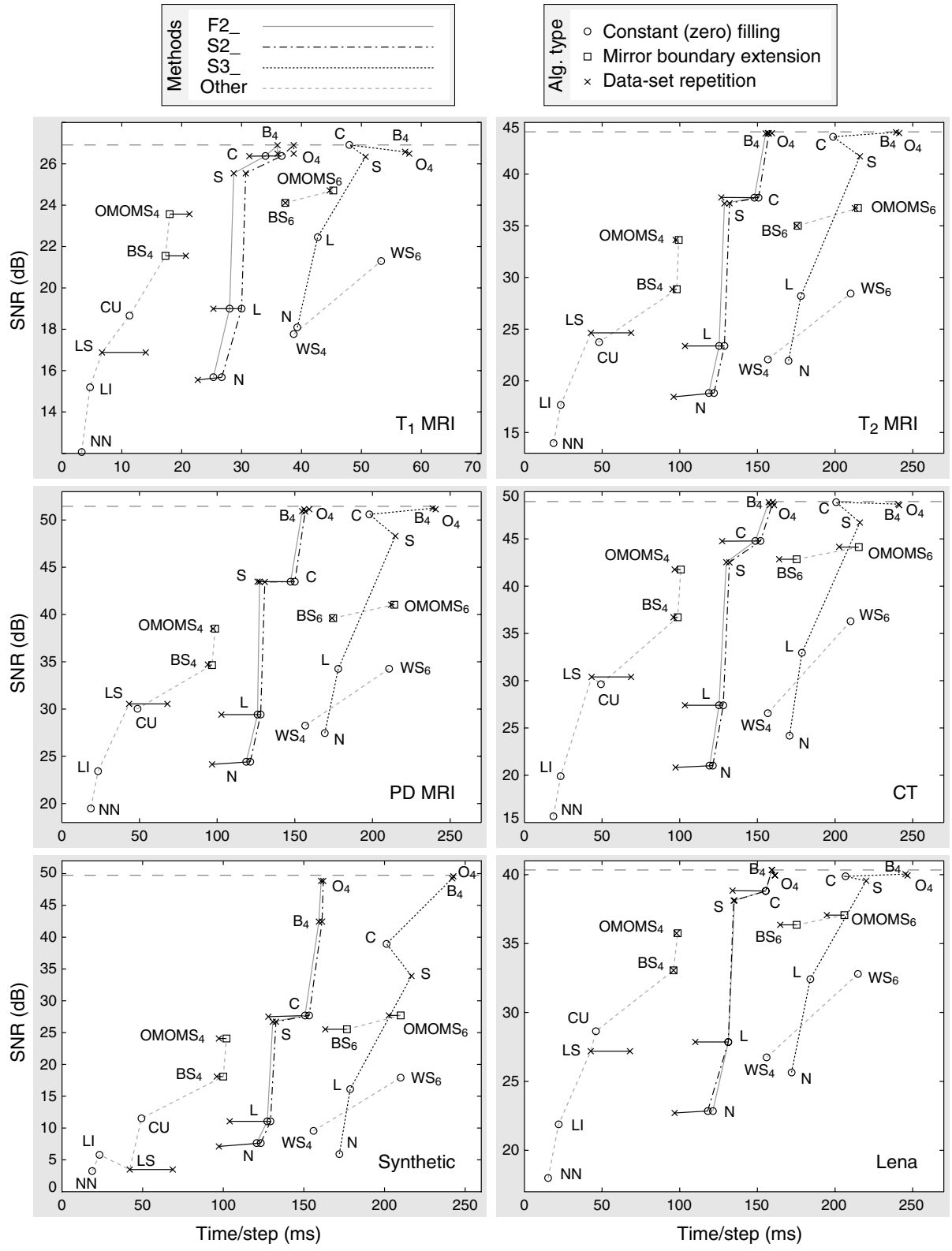

Fig. 7. Comparison of different resampling methods with 2-D datasets. See text for explanations.

letter (either F or S) stands for the algorithm used in the up-sampling stage (FSM or SDFT, respectively). The number identifies the magnification factor used for the up-sampling stage. The last letter designates the method used in the second-stage resampling step and is N (nearest neighbor), L (linear), S (shifted linear), C (Keys cubic), B

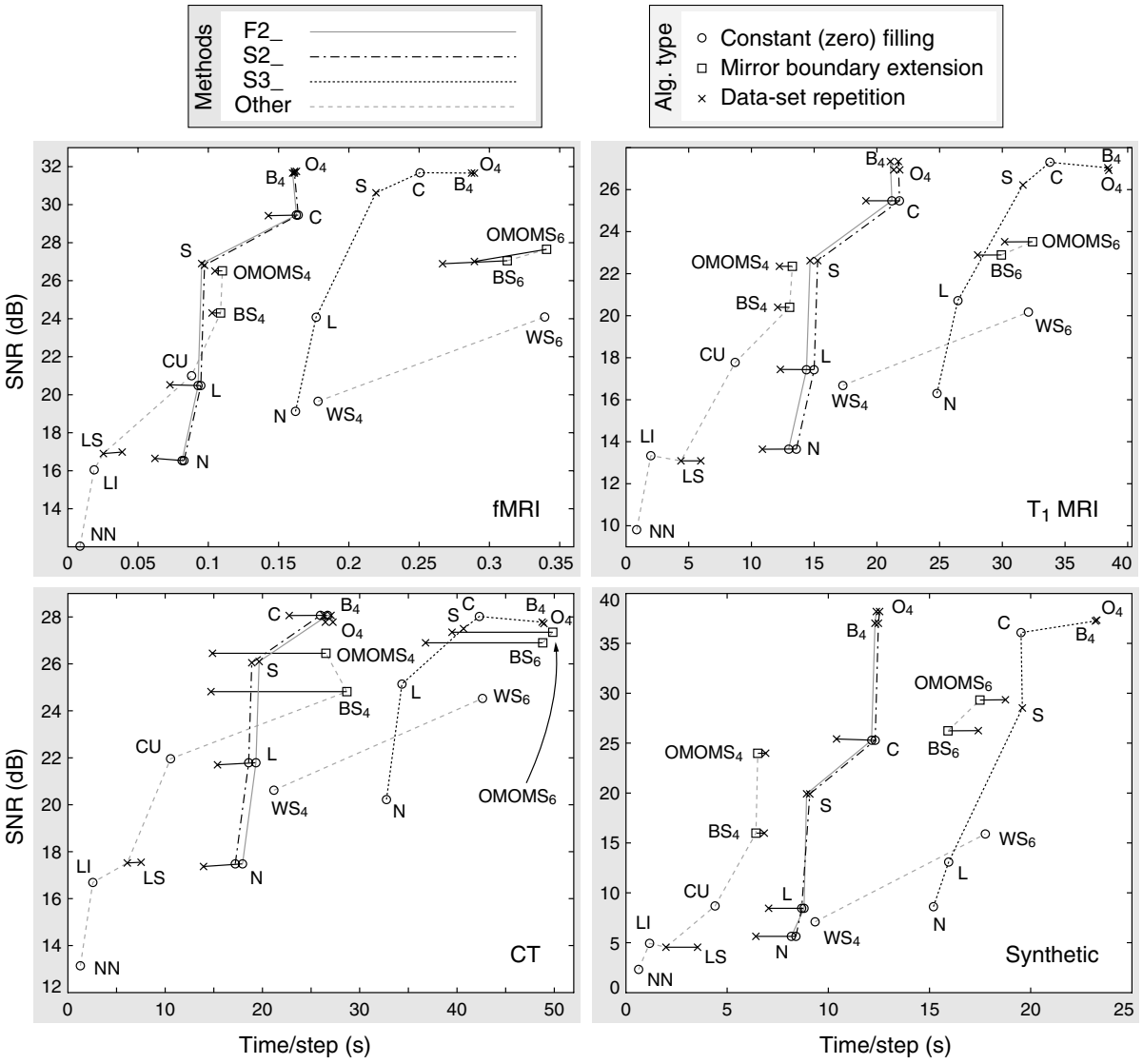

Fig. 8. Comparison of different resampling methods with 3-D datasets. See text for explanations.

(B-spline), or O (OMOMS). The subindex specifies the size of the support for B-spline and OMOMS kernels and asterisk marks non-centered version of SDFT-based methods. For example, $F2O_4$ means up-sampling with FSM algorithm using magnification factor 2 followed by OMOMS resampling of support 4. The preprocessing necessary for OMOMS is incorporated as transfer function multiplication in the FSM algorithm.

In Figs. 7 and 8 similar methods are connected by lines to enhance readability. The two-stage resampling methods are grouped by the up-sampling stage (up-sampling algorithm and magnification) and the letter in the graphs specify the second-stage resampling method. The legends identify the line and marker types used. The previously known methods shown for comparison are grouped under title "Other" which contains methods employing both traditional and generalized interpolation. Those methods are divided into three sub-groups and the left-most containing 6 methods (NN through $OMOMS_4$) is identical to the second-stage methods of the two-stage resampling (N through O_4). The remaining two sub-groups are the support-6 B-spline based methods (BS_6 and $OMOMS_6$) and the windowed-sinc methods (WS_4 and WS_6).

The solid horizontal lines in Figs. 7 and 8 connect alternative versions of the same method. For the two-stage resampling methods, the line links to the non-centered version identified by the asterisks in Table 2. For generalized interpolation methods, it links to the same method with preprocessing performed through FFT instead of forward and backward recursive filters. The three different mark types in Figs. 7 and 8 identify the type of the kernel multiplication algorithm (see Fig. 1) employed.

For the 2-D cases, the two-stage resampling was tested with magnification factors up to 8. The calculation time became progressively slower without any further gain in SNR. The maximum SNR for all images was reached already with factor 2 or 3 and is shown in Fig. 7 with horizontal broken line at the top of each graph.

In Fig. 8 the graph of CT data shows relatively slow performance for the BS and OMOMS methods implemented through the recursive prefiltering (box markers), especially for support size 4. This performance issue and the timing difference between the BS_4 and $OMOMS_4$ are related to the CPU cache (512 kB) usage with the big CT dataset ($512 \times 512 \times 58$). When the CT test-case was run on another platform with 1 MB CPU cache, the timing difference between the BS_4 and the $OMOMS_4$ methods disappeared and the execution times were as expected from the other graphs.

To obtain high quality resampling results, Figs. 7 and 8 clearly show that the two-stage scheme is better than just increasing the size of the support for the other methods. Especially in 3-D (Fig. 8), the computational cost for large support (WS_6, BS_6, and $OMOMS_6$) becomes obvious. These figures also show that the two-stage resampling can practically reach its maximum SNR already with magnification factor 2, at least for these datasets.

3.3. Resampling errors

Fig. 9 shows difference images for the compared central areas of the 2-D T_1 MRI dataset after 15 resampling steps. Zero difference is mapped to middle gray and the deviations are magnified 6-fold and manifest as darker and lighter colors. Both magnitude and distribution of errors can be assessed from these images. Best methods should produce difference images as close to middle gray as possible with minimal variation and no visual structure in the resampling error. The methods are labeled as mentioned above. Fig. 9 visually confirms the numerical results of SNR and shows that practically no structure is left in the errors of $F2B_4$ and $F2O_4$. On the other hand, various borders show up clearly in the errors of other methods, even for normal B-spline and OMOMS methods with support size 6.

Fig. 10 shows the response of selected methods to Gaussian noise added to the 2-D images. Panels (a) and (b) verify that the two-stage resampling methods ($F2^*$/$S3^*$) maintain their superior quality to other methods even with increasing noise level. Panels (c)–(h) compare different two-stage resampling methods using the T_1-weighted MRI (c), T_2-weighted MRI (d), proton-density weighted MRI (e), CT image (f), the synthetic image (g), and the gray-scale photograph Lena (h). In these six panels, the difference between each method and the F2C method is plotted as a function of the standard deviation of the

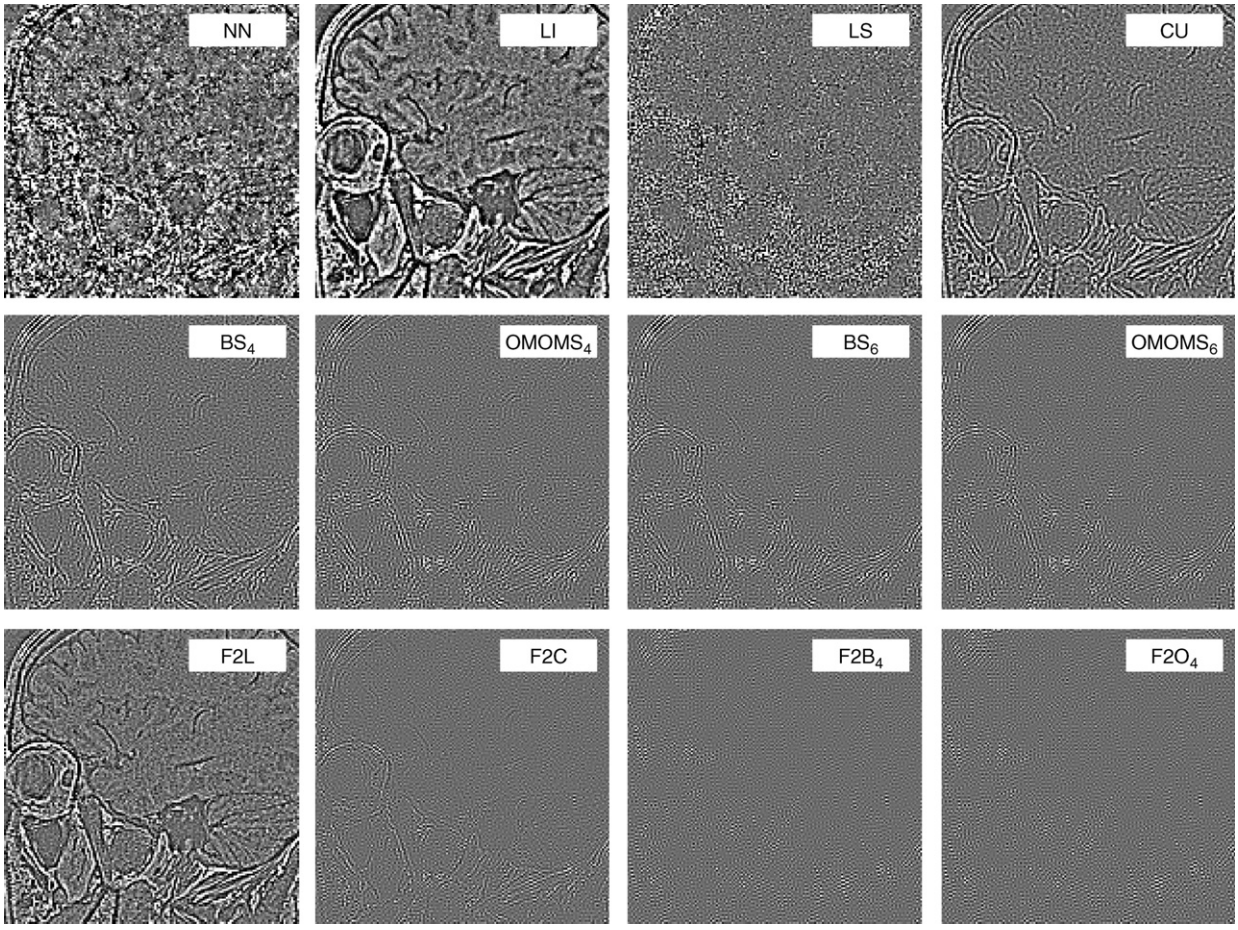

Fig. 9. Difference images showing the resampling error for the 2-D T_1 MRI slice rotated 15 times. The error is magnified 6-fold to make it more visible and the zero difference is mapped to middle gray. Area shown is the same as used for calculating the SNR of the methods.

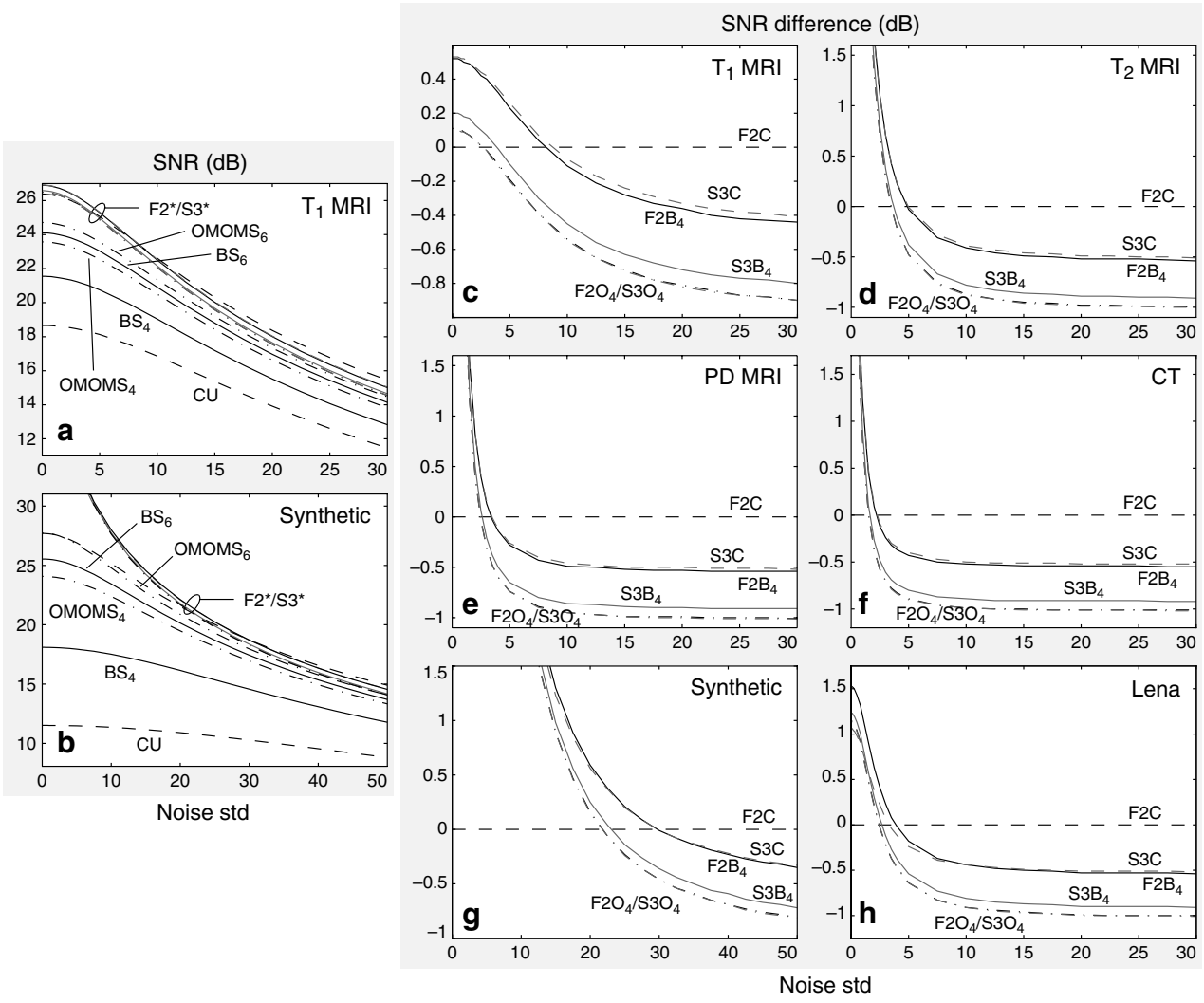

Fig. 10. Effect of image noise to the resampling results. Panels (a) and (b) show the absolute SNR of the methods as a function of standard deviation of the added Gaussian noise. Panels (c)–(h) show the difference in SNR between other methods and F2C as a function of the noise.

added Gaussian noise. The original 2-D T_1 MRI had gray-values from 0 to 255, with mean of 68 and standard deviation of 46. Respectively, the T_2 MRI had range 0–255 (mean 50, std 47), PD MRI had range 0–255 (mean 75, std 56), the CT image had range 0–255 (mean 57, std 62), the synthetic image had range 0–255 (mean 128, std 90), and the Lena image had range 16–242 (mean 129, std 48). Notice that the synthetic image has different horizontal scale than the other images. As can be seen in Fig. 6 (right), the synthetic image does not appear very noisy even at noise level 20 when the other two-stage resampling methods drop to the level of the F2C method (Fig. 10b and g).

Fig. 11 shows error histograms after 15 resampling steps for the 3-D fMRI dataset. Best methods should have small RMS value, small maximum absolute error, and an as narrow as possible distribution centered on zero. The center area of the fMRI used for comparison has values in range 0–2148, with mean of 607 and standard deviation of 629.

4. Discussion

The tests (Table 1) demonstrate that the FSM algorithm works well and offers a clear speed-up to normal (unpruned) FFT implementations and a slight (1–8%) speed-up to previously used pruned FFT methods. In most cases, FSM is also faster than the SDFT method and seems to be inferior only with high dataset and magnification sizes. In those cases, the SDFT method benefits from better CPU cache performance as it processes smaller arrays than FSM (but multiple times). With relatively small array sizes and especially in the one-dimensional case the performance of SDFT is diminished by the overhead of calculating trigonometric functions in the transfer function (Eq. 5). The overhead of arranging the interleaved shifted arrays also has its effect, especially when final sample spacing increases with higher dimensional data.

The non-centered version SDFT* clearly benefits from the possibility to copy the original data and to calculate the shift only for $M-1$ out of M cases where M is the

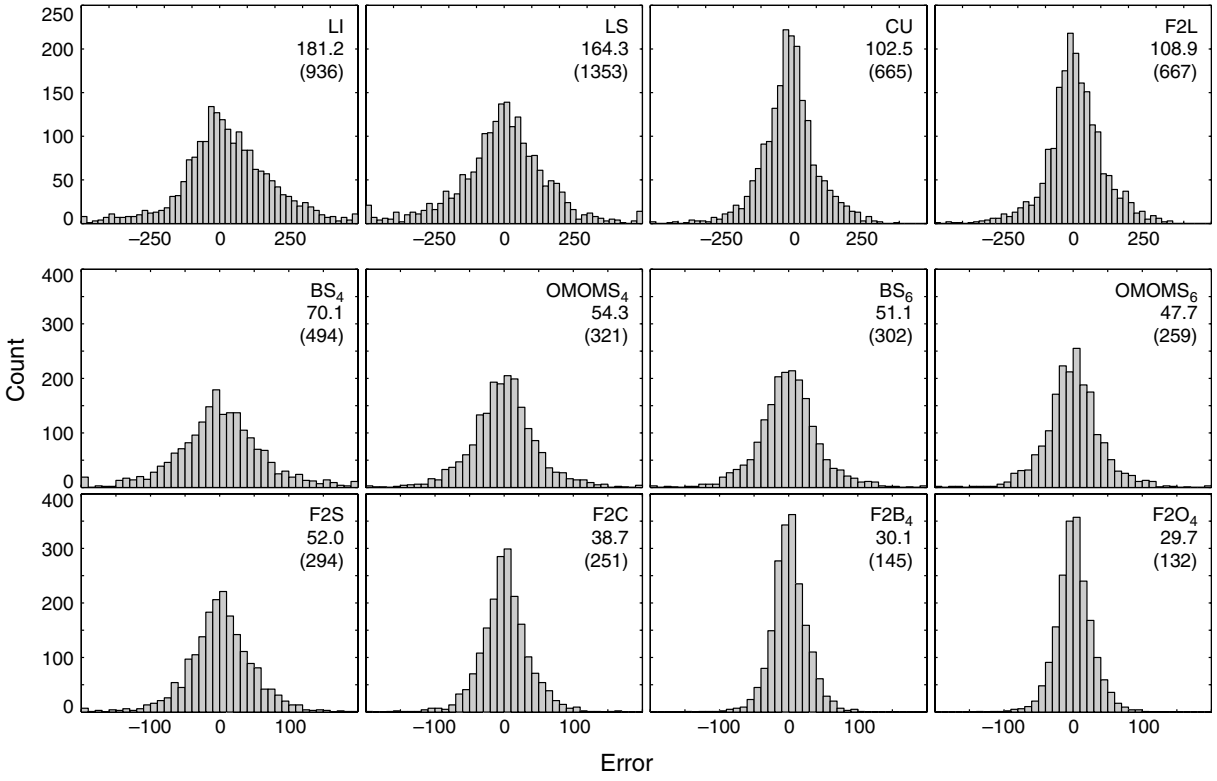

Fig. 11. Error histograms of 3-D fMRI data for different resampling methods after 15 resampling steps. The top right-hand corner of each panel shows the resampling method, root-mean-square (RMS) error, and the maximum absolute error in parenthesis. The top row has different scales on both axes than the bottom two rows.

magnification factor. Naturally, the effect is most notable with $M = 2$ which also seems to be the most usable factor for two-stage resampling, as is evident below. Selecting SDFT* requires a second-stage algorithm with dataset repetition for outside values. Otherwise, biasing is introduced at the boundaries.

With all the computational burden of the two-stage resampling summed up, both FSM and SDFT are quite equal for 2-D and 3-D datasets (Table 2, Figs. 7 and 8), with FSM showing only a minute advantage over SDFT. Thus, although the conclusions below are drawn for FSM-based versions, SDFT-based versions can be used just as well. Furthermore, if constant (zero) outside values are not necessary, the faster SDFT* version can be applied as well (horizontal solid lines in Figs. 7 and 8).

The quality provided by the two-stage resampling scheme proves to be superior (Table 2, Figs. 7–11) when compared with other commonly used resampling methods. The best-quality versions (F2C, $F2B_4$, and $F2O_4$) are clearly slower than the compared support 4 methods (CU, BS_4, $OMOMS_4$). However, these two-stage resampling methods are faster than the compared support 6 methods (WS_6, BS_6, $OMOMS_6$) and provide much better SNR.

Selection of the best two-stage version depends on whether dataset repetition is acceptable or if use of constant (zero) for outside values is necessary. If repetition is allowed, version $F2B_4$ seems to be the choice in normal circumstances. Among generalized interpolation methods, $OMOMS_4$ is clearly better than BS_4 in all the tests. Nevertheless, the two-stage version $F2O_4$ does not outperform $F2B_4$ except in the noiseless synthetic case. In fact, $F2B_4$ seems to provide slightly better results than FSO_4 for most of the natural datasets. Both of these methods use prefiltering that enhances high-frequency signals but the amplification factor for FSO_4 is higher making it more sensitive to noise. As can be seen in Fig. 10, F2C method outperforms all the other methods, even S3C, under extremely noisy conditions. Another benefit of F2C is the ability to use either dataset repetition (F2C*, marked with cross in Figs. 7 and 8) or constant (zero) outside values (marked with circle).

As a conclusion, the two-stage resampling method F2C (or S2C) is a solid choice in applications needing ultimate quality for resampling. It provides better results than any of the previously used methods under realistic or higher noise levels. Nevertheless, F2C is still faster than the support 6 (or above) methods, especially in 3-D where the size of the support becomes crucial. If dataset repetition is acceptable for outside values, version $F2B_4$ might provide even higher quality results unless the data contain excessive amounts of noise.

One specific field where two-stage resampling methods should be advantageous is volume registration applications in medical imaging, such as fMRI motion correction. In the registration process, the resampling of one volume is

repeated many times when the correct registration parameters are searched for. Thus, the up-sampling of the two-stage resampling method is necessary to perform just once and only the second stage is recalculated during iterations. In this special case, it is beneficial to use magnification factor of 3 or 4 for the up-sampling stage and use fast small-support shifted-linear (support 2) resampling for the second stage (e.g. S3S). Due to high temporary memory consumption, magnification factors 3 and 4 might not be practical or even possible for high-resolution volumes. On the other hand, fMRI volumes are typically relatively small and therefore are very fit to this kind of a method.

Acknowledgements

I thank Riitta Hari and Kimmo Uutela for comments on the manuscript and Ulla Ruotsalainen for useful advices. This work was supported by the Academy of Finland (National Centers of Excellence Program).

References

Blu, T., Thévenaz, P., Unser, M., 1999. Generalized interpolation: higher quality at no additional cost. In: Proceedings of IEEE International Conference Image Proc. '99, vol. III. Kobe, Japan, pp. 667–671.

Blu, T., Thévenaz, P., Unser, M., 2001. MOMS: maximal-order interpolation of minimal support. IEEE Trans. Image Proc. 10 (7), 1069–1080.

Blu, T., Thévenaz, P., Unser, M., 2004. Linear interpolation revitalized. IEEE Trans. Image Proc. 13 (5), 710–719.

Catmull, E., Smith, A.R., 1980. 3-D transformations of images in scanline order. Proc. SIGGRAPH '80. ACM Press, New York, USA, pp. 279–285.

Du, Y.P., Parker, D.L., Davis, W.L., Cao, G., 1994. Reduction of partial-volume artifacts with zero-filled interpolation in three-dimensional MR angiography. J. Magn. Reson. Imaging 4 (5), 733–741.

Gonzalez, R.C., Woods, R.E., 1992. Digital Image Processing. Addison Wesley, Reading, Massachusetts.

Goshtasby, A., Turner, D.A., Ackerman, L.V., 1992. Matching of tomographic slices for interpolation. IEEE Trans. Med. Imaging 11 (4), 507–516.

Grevera, G.J., Udupa, J.K., 1996. Shape-based interpolation of multidimensional grey-level images. IEEE Trans. Med. Imaging 15 (6), 881–892.

Grevera, G.J., Udupa, J.K., 1998. An objective comparison of 3-D image interpolation methods. IEEE Trans. Med. Imaging 17 (4), 642–652.

Hou, H.S., Andrews, H.C., 1978. Cubic splines for image interpolation and digital filtering. IEEE Trans. Acoust., Speech, Signal Proc. 26 (6), 508–517.

Jaroslavski, L.P., 1981. Comments on FFT algorithm for both input and output pruning. IEEE Trans. Acoust., Speech, Signal Proc. 29 (3), 448–449.

Keys, R.G., 1981. Cubic convolution interpolation for digital image processing. IEEE Trans. Acoust., Speech, Signal Proc. 29 (6), 1153–1160.

Lehmann, T.M., Gönner, C., Spitzer, K., 1999. Survey: Interpolation methods in medical image processing. IEEE Trans. Med. Imaging 18 (11), 1049–1075.

Lehmann, T.M., Gönner, C., Spitzer, K., 2001. Addendum: B-spline interpolation in medical image processing. IEEE Trans. Med. Imaging 20 (7), 660–665.

Markel, J.D., 1971. FFT pruning. IEEE Trans. Audio Electroacoust. 19 (4), 305–311.

Meijering, E.H., Zuiderveld, K.J., Viergever, M.A., 1999. Image reconstruction by convolution with symmetrical piecewise *n*th-order polynomial kernels. IEEE Trans. Image Proc. 8 (2), 192–201.

Meijering, E.H., Niessen, W.J., Viergever, M.A., 2001. Quantitative evaluation of convolution-based methods for medical image interpolation. Med. Image Anal. 5 (2), 111–126.

Nagai, K., 1986. Pruning the decimation-in-time FFT algorithm with frequency shift. IEEE Trans. Acoust., Speech, Signal Proc. 34 (4), 1008–1010.

Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T., 1988. Numerical Recipes in C. Cambridge University Press, Cambridge.

Rangarajan, S.R., Srinivasan, S., 1997. Generalised method for pruning an FFT type of transform. IEE Proc.-Vis. Image Signal Process. 144 (4), 189–192.

Reichenbach, S.E., Geng, F., 2003. Two-dimensional cubic convolution. IEEE Trans. Image Proc. 12 (8), 857–865.

Seppä, M., 2007. FSM – Fast sinc magnification algorithm in C. URL http://neuro.hut.fi/~mseppa/fsm/.

Skinner, D.P., 1976. Pruning the decimation-in-time FFT algorithm. IEEE Trans. Acoust., Speech, Signal Proc. 24, 193–194.

Smit, T., Smith, M.R., Nichols, S.T., 1990. Efficient sinc function interpolation technique for center padded data. IEEE Trans. Acoust., Speech, Signal Proc. 38 (9), 1512–1517.

Smith, M.R., Nichols, S.T., 1988. Efficient algorithms for generating interpolated (zoomed) MR images. Magn. Res. Med. 7 (2), 156–171.

Sreenivas, T.V., Rao, P.V.S., 1979. FFT algorithm for both input and output pruning. IEEE Trans. Acoust., Speech, Signal Proc. 27 (3), 291–292.

Thévenaz, P., Blu, T., Unser, M., 2000. Interpolation revisited. IEEE Trans. Med. Imaging 19 (7), 739–758.

Tsuchida, N., Yamada, Y., Ueda, M., 1987. Hardware for image rotation by twice skew transformations. IEEE Trans. Acoust., Speech, Signal Proc. 35 (4), 527–532.

Unser, M., 1999. Splines: a perfect fit for signal and image processing. IEEE Signal Proc. Mag. 16 (6), 22–38.

Unser, M., Aldroubi, A., Eden, M., 1993a. B-spline signal processing: Part I-theory. IEEE Trans. Signal Proc. 41 (2), 821–833.

Unser, M., Aldroubi, A., Eden, M., 1993b. B-spline signal processing: Part II-efficient design and applications. IEEE Trans. Signal Proc. 41 (2), 834–848.

Unser, M., Thévenaz, P., Yaroslavsky, L., 1995. Convolution-based interpolation for fast, high-quality rotation of images. IEEE Trans. Image Proc. 4 (10), 1371–1381.

Yaroslavsky, L.P., 1997. Efficient algorithm for discrete sinc interpolation. App. Opt. 36 (2), 460–463.