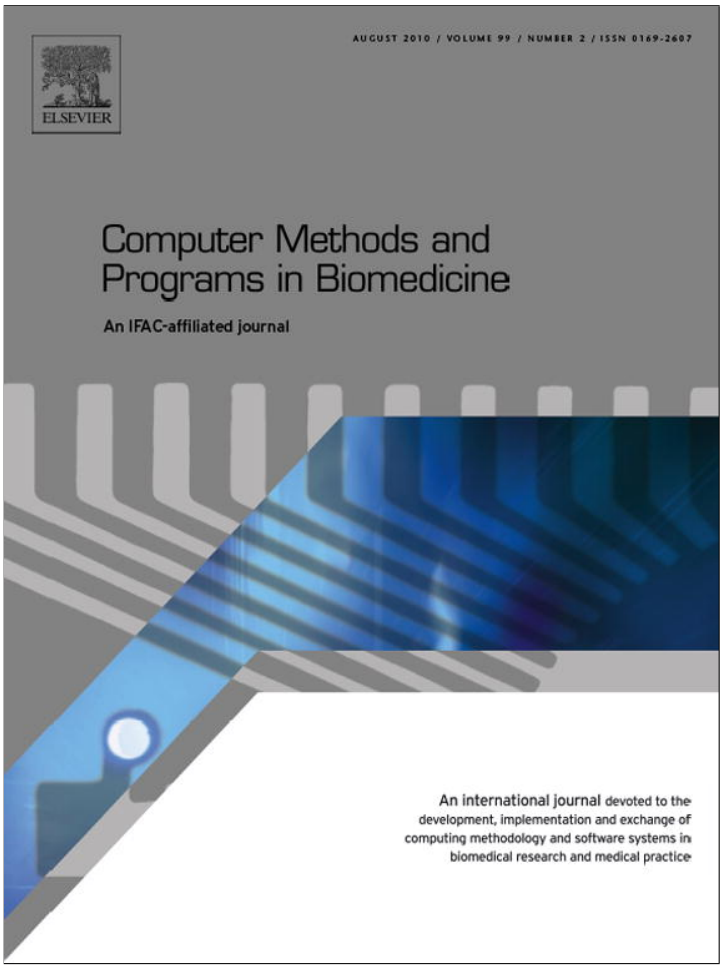
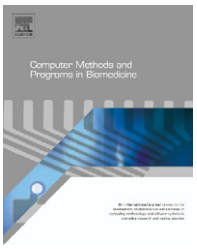


COMPUTER METHODS AND PROGRAMS IN BIOMEDICINE 99 (2010) 172–178

journal homepage: www.intl.elsevierhealth.com/journals/cmpb



B-LUT: Fast and low memory B-spline image interpolation

David Sarrut[a,b,c,*], *Jef Vandemeulebroucke*[a,b,c]

[a] *Université de Lyon, F-69622 Lyon, France*
[b] *Creatis, CNRS UMR 5220, F-69622 Villeurbanne, France*
[c] *Léon Bérard Cancer Center, 28 rue Laennec, F-69373 Lyon cedex 08, France*

ARTICLE INFO

Article history:
Received 15 July 2009
Received in revised form
24 November 2009
Accepted 24 November 2009

Keywords:
B-spline image transformation
B-spline image interpolation

ABSTRACT

We propose a fast alternative to B-splines in image processing based on an approximate calculation using precomputed B-spline weights. During B-spline indirect transformation, these weights are efficiently retrieved in a nearest-neighbor fashion from a look-up table, greatly reducing overall computation time. Depending on the application, calculating a B-spline using a look-up table, called B-LUT, will result in an exact or approximate B-spline calculation. In case of the latter the obtained accuracy can be controlled by the user. The method is applicable to a wide range of B-spline applications and has very low memory requirements compared to other proposed accelerations. The performance of the proposed B-LUTs was compared to conventional B-splines as implemented in the popular ITK toolkit for the general case of image intensity interpolation. Experiments illustrated that highly accurate B-spline approximation can be obtained all while computation time is reduced with a factor of 5–6. The B-LUT source code, compatible with the ITK toolkit, has been made freely available to the community.

1. Introduction

B-splines are widely used in image processing for manipulating a continuous version of a discrete image [1]. Using B-splines, an n-dimensional (n D) image (or signal) can be represented through a set of n D coefficients. Obtaining the image value at any continuous coordinate involves a linear combination of coefficients and basis function (B-spline) weights. Thanks to its compact support, this involves only a finite and usually small number of coefficients.

Considering the case of B-spline interpolation, two processes can be distinguished. The first one is referred to as the *direct transformation* and consists in computing a set of B-spline coefficients from the initial image. Very efficient digital filtering schemes have been proposed [2,3] to solve this issue. The second process, called *indirect transformation* consists in combining the found coefficients and weights for a given position. This latter process remains relatively slow. For example, performing a rotation of a three-dimensional (3D) image with size $512 \times 512 \times 200$ (≈ 50 million voxels) using cubic B-spline interpolation takes about 200 s. The computation of coefficients (the direct transformation) only takes about 10 s (both performed on a 2 GHz PC, using the ITK toolkit [4], see Section 3).

To our knowledge, few studies directly address the indirect computational time issue. Acceleration of processes which include B-spline interpolation, such as deformable image registration, are generally addressed in a parallel framework with hardware-based methods: with clusters of workstations [5], with multi-processors shared-memory systems [6], or with graphical processing units or GPUs [7,8]. Some authors [9] mentioned the use of precomputed weights, but we do not find any publication describing such work.

The computational cost of the indirect transformation is caused by the high number of operations performed for each

* *Corresponding author at*: Léon Bérard Cancer Center, CREATIS, 28 rue Laennec, F-69373 Lyon cedex 08, France. Tel.: +33 478785151.
E-mail address: david.sarrut@creatis.insa-lyon.fr (D. Sarrut).
0169-2607/$ – see front matter
doi:10.1016/j.cmpb.2009.11.013

interpolated value. For simplicity we will assume in the following that the same spline degree was chosen along all dimensions. Observations can however easily be extended to the general case.

During the first step of the indirect transformation we compute the tensor products of the B-spline basis function values according to the distance between the current position and each contributing control point in the support region (Eq. (1)).

$$\text{Step1} \quad \boldsymbol{\beta}_i^r(x) = \prod_j^d \beta^r(p_{ij} - x_j). \tag{1}$$

β^r is the B-spline basis function of degree r, d is the image dimension, i is the index of a control point with coordinates p_{ij}, x is the position at which we want to evaluate the function and x_j are its coordinates. For a given position x, $(r+1)^d$ different tensor products are computed, one for each control point with non-zero weight at x.

The second step required to compute the interpolated value $v(x)$ involves the linear combination of the tensor products ($\boldsymbol{\beta}_i^r$) with the corresponding coefficients c_i, previously computed during the direct transformation (Eq. (2)). In the following the term weights will refer exclusively to the B-spline tensor products $\boldsymbol{\beta}_i^r$.

$$\text{Step2} \quad v(x) = \sum_i^{(r+1)^d} c_i \, \boldsymbol{\beta}_i^r(x). \tag{2}$$

Evaluating the basis function $\beta^r(e)$ involves the computation of a polynomial of degree r. The initial definition of the B-spline functions is obtained recursively by convolving β^0 $(n+1)$ times with itself. An analytic expression can also be obtained by applying the recursive Cox-de Boor formula (Eq. (3)).

$$\boldsymbol{\beta}^r(e) = \boldsymbol{u}\boldsymbol{M} \text{ with } \begin{cases} \boldsymbol{u} = [e^r \; e^{r-1} \; \dots \; e \; 1] \\ \boldsymbol{M} \text{ matrix of size} k \times k \text{ with } k = r+1 \end{cases} \tag{3}$$

$$\boldsymbol{M} = [M_{ij}] = \left[\frac{1}{(k-1)!} C_{k-1,i} \sum_{m=j}^{k-1} (k-(m+1))^i \, (-1)^{m-j} \, C_{k,m-j} \right] \tag{4}$$

$$C_{i,j} = \frac{i!}{j!(i-j)!} = \text{binomial coefficient} \tag{5}$$

Step 1 requires about $2 \times r$ operations (additions or multiplications) for computing one β^r, thus $2 \times r \times d$ for the tensor product $\boldsymbol{\beta}^r$. There are $(r+1)^d$ different weights. This leads to a total complexity of $O(r \times d \times (r+1)^d)$ operations for step 1, while only $O((r+1)^d)$ for step 2.

If the interpolation is performed with a regular sampling rate which is a multiple of the control point spacing, the required weights will reoccur across the image region due to symmetry. In this case they can be precomputed, stored and reused, allowing to avoid most of the computational part of step 1, without any loss of accuracy. However, as generally this is not the case (e.g. when performing an image rotation or during image warping), the exact weights cannot be precomputed.

We propose to extend the range of applications for which splines can be efficiently calculated using precomputed weights. For applications where no exact calculation can be achieved the approximation error is controlled by oversampling the control point grid. Through efficient design of the weights look-up table (LUT), memory requirements are kept to a minimum further extending the use of the method to large scale problems. As we will show in Section 4, the proposed B-LUT framework can obtain high interpolation accuracy while offering considerable reduction in computation time.

2. Method

The method consists in approximating the tensor product (Eq. (1)) by a precomputed one. At interpolation time, step 1 is replaced by finding the closest precomputed weights in the LUT. Step 2 can then be applied in a conventional way by looping over the coefficients and their corresponding weights.

2.1. LUT computation

The computation of the weights is made only once before the interpolation. By choosing the (*over*)sampling rate of the precomputed weights with respect to the control point grid, it is possible to control the trade-off between LUT size and approximation accuracy. Note that the whole image region does not need to be sampled. Since we are assuming uniform B-splines, it suffices to sample one n-dimensional B-spline support.

Let $\lambda_j \in \mathbb{N}$ be the LUT sampling rate for the dimension j. The size of the LUT is the size of the B-spline support, $(r+1)^d$, multiplied by the sampling rate in each dimension $\prod_j^d \lambda_j$. The overall computation time of interpolating an entire image once can only be reduced if the number of weights to precompute is $< (r+1)^d$ times the number of pixels to interpolate. In practice however, this is often the case. For example, to interpolate a 256^3 3D image having about 1.6×10^7 pixels, the number of weights to precompute for a sampling rate equal to 20 is 5.12×10^5 ($20^3 \times (3+1)^3$), in comparison to 1.024×10^9 ($16 \times 10^6 \times (3+1)^3$) weights in total. Assuming single precision, the LUT memory size requirement is $4\,bytes \times (r+1)^d \times \prod_j^d \lambda_j$, which leads to 2 MB in this case.

2.2. LUT look-up

After computing the weights, step 1 in Eq. (1) is replaced by finding the optimal precomputed weight in the LUT. The interpolation position x is then transformed to its corresponding position x' relative to the sampled B-spline support region.

For efficiency, the LUT is indexed such that a single rounding operation, denoted $\lfloor a \rceil$, on each of the coordinates of x' leads to the index in the LUT l of the first element of the list of all $(r+1)^d$ weights corresponding to the current position, see Eq. (6). This implies that for each evaluation position a single look-up replaces step 1, providing all the weights required for the calculation in step 2 (Eq. (2)).

$$l = \sum_{j=1}^{d} \left(k_j \prod_i^j \lambda_{i-1} \right) \quad \text{with} \quad k_j = \lfloor x'_j \rceil \quad \text{and} \quad \lambda_0 = 1 \tag{6}$$

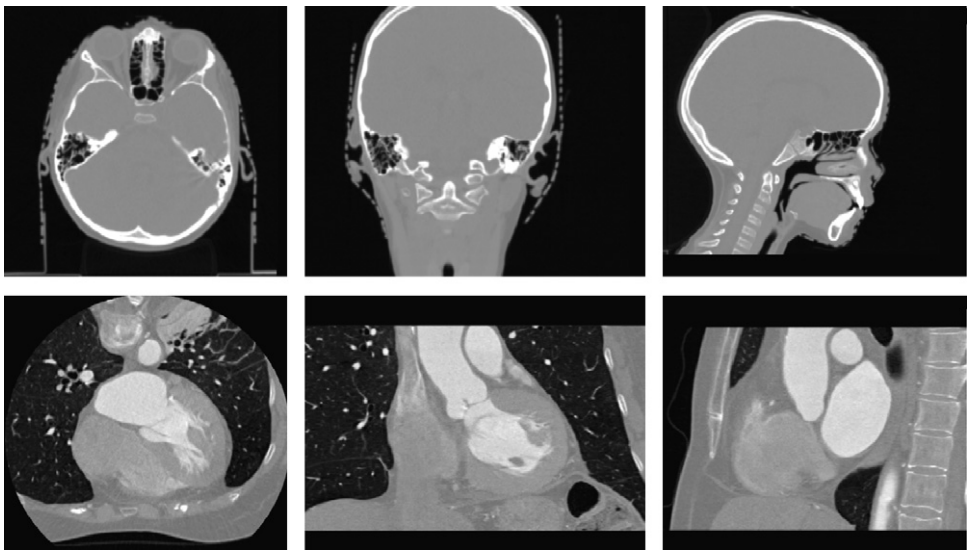


Fig. 1 – The two images used for the tests : a CT image of a head (top) and a CT image of heart/thorax (bottom).

3. Design considerations

We implemented the method inside the Insight Segmentation and Registration Toolkit (ITK[1][4]). This toolkit is widely used in medical image analysis and uses the notion of filters to represent a process chain. A new filter was created named `BSplineInterpolateImageFunctionWithLUT` which inherits from the original `BSplineInterpolateImageFunction`. This way, users only have to change three lines inside their usual code to use B-LUTs instead of B-splines. The code and procedure are available on the following web page http://www.creatis.insa-lyon.fr/rio/b-lut under the CeCILL open source license[2].

4. Experiments

In this work the performance of the proposed B-LUTs is assessed in the context of image intensity interpolation. Similar results can be expected in other application areas (see Section 5 for an example on deformable image registration). In addition to a quantitative accuracy evaluation, we provide an analysis of the time gain over conventional B-splines. The latter should be considered as an example as the obtained gain was found to be highly dependent on the computer architecture.

4.1. *B-LUT interpolation accuracy*

In order to illustrate the inherent loss of accuracy caused by the use of approximated weights, we performed the following tests, inspired from [9]. A 3D image was rotated several times around an arbitrary axis with a series of 16 different angles[3], resulting in a complete rotation of 360°. This procedure provides us with a gold standard to compare the rotated image with: the original image. Moreover, the image is used at its intrinsic resolution and the use of different rotation angles allows to avoid bias due to precomputed weight positions.

The test images were two CT images, the first of the head and the second of the lungs and heart. Their respective sizes were $512 \times 512 \times 222$ and $512 \times 512 \times 403$, and their voxel sizes $0.5 \times 0.5 \times 1$ and $0.4 \times 0.4 \times 0.3$. The used images therefore contained about 58 and 105 million voxels respectively. Experiments were performed for B-spline degrees from linear ($r = 1$) to quintic ($r = 5$), including the popular cubic B-splines ($r = 3$). For each degree, we tested different LUT sampling values (λ equal to 1, 2, 4, 5, 10, 20 and 50 for each image dimension). The rotated image when using B-LUT interpolation was compared to the one obtained with B-spline interpolation and to the original reference image. Differences were quantified by computing the Root Mean Squared Error (RMSE) and the maximum difference (MAX). Pixels that go out of the image support during the rotation are discarded and thus not counted into the RMSE final values, by using a binary image mask.

Fig. 1.

Table 1 summarizes the results for the consecutive rotation experiment described above. In the two first tables we assess the accuracy of the B-LUT interpolation with respect to the conventional B-spline method, while in the following two tables a comparison with the original reference image is performed. Fig. 2 displays RMSE between the two methods (the RMSE axis is logarithmic) for the first image (similar results were obtained for the second). Fig. 3 illustrates the resulting images with the first test image.

Fig. 4.

[1] http://www.itk.org.
[2] http://www.cecill.info/index.en.html.
[3] Angles were 0.7°, 3.2°, 6.5°, 9.3°, 12.1°, 15.2°, 18.4°, 21.3°, 23.7°, 26.6°, 29.8°, 32.9°, 35.7°, 38.5°, 41.8°and 44.3°.

Table 1 – (Top) RMSE and maximum error (MAX) between fast B-spline and conventional B-spline. (Bottom) RMSE between reference image without rotation and images rotated by 360° with sampled (λ from 1 to 50) and conventional (denoted by 'Ref') B-spline interpolations.

	RMSE (and MAX) error of B-LUT according to conventional B-spline						
	$\lambda = 1$	$\lambda = 2$	$\lambda = 4$	$\lambda = 5$	$\lambda = 10$	$\lambda = 20$	$\lambda = 50$
Linear $r = 1$	133 (3679)	23 (702)	9 (276)	7 (190)	3 (91)	1 (40)	1 (14)
Quadratic $r = 2$	138 (4156)	48 (1805)	21 (862)	17 (700)	8 (328)	4 (149)	2 (63)
Cubic $r = 3$	140 (4216)	53 (2038)	23 (944)	18 (738)	9 (369)	4 (187)	2 (69)
Quartic $r = 4$	142 (4351)	59 (2422)	26 (1143)	21 (882)	10 (465)	5 (232)	2 (88)
Quintic $r = 5$	143 (4356)	61 (2635)	28 (1219)	22 (922)	11 (503)	5 (256)	2 (96)

	RMSE (and MAX) error according to original image							
	$\lambda = 1$	$\lambda = 2$	$\lambda = 4$	$\lambda = 5$	$\lambda = 10$	$\lambda = 20$	$\lambda = 50$	Ref
Linear $r = 1$	144 (3982)	87 (3059)	89 (3062)	89 (3077)	90 (3105)	90 (3104)	90 (3104)	90 (3102)
Quadratic $r = 2$	144 (3982)	57 (2933)	33 (1887)	29 (1889)	24 (1593)	23 (1616)	22 (1536)	22 (1542)
Cubic $r = 3$	144 (3982)	58 (3027)	30 (1840)	26 (1725)	20 (1465)	18 (1484)	17 (1386)	17 (1396)
Quartic $r = 4$	144 (3982)	61 (3122)	30 (1772)	25 (1600)	17 (1330)	14 (1349)	13 (1216)	13 (1252)
Quintic $r = 5$	144 (3982)	63 (3168)	31 (1781)	25 (1571)	16 (1281)	13 (1302)	12 (1162)	12 (1206)

	RMSE (and MAX) error of B-LUT according to conventional B-spline						
	$\lambda = 1$	$\lambda = 2$	$\lambda = 4$	$\lambda = 5$	$\lambda = 10$	$\lambda = 20$	$\lambda = 50$
Linear $r = 1$	91 (2517)	16 (382)	6 (188)	5 (120)	2 (70)	1 (28)	0 (12)
Quadratic $r = 2$	94 (2884)	31 (946)	15 (461)	12 (408)	5 (183)	3 (88)	1 (41)
Cubic $r = 3$	95 (2878)	33 (1084)	16 (474)	13 (419)	6 (196)	3 (98)	1 (44)
Quartic $r = 4$	96 (2872)	37 (1254)	19 (574)	15 (513)	7 (236)	4 (109)	1 (50)
Quintic $r = 5$	96 (2870)	39 (1330)	20 (630)	15 (562)	7 (259)	4 (115)	1 (52)

	RMSE (and MAX) error according to original image							
	$\lambda = 1$	$\lambda = 2$	$\lambda = 4$	$\lambda = 5$	$\lambda = 10$	$\lambda = 20$	$\lambda = 50$	Ref
Linear $r = 1$	97 (2848)	55 (1154)	56 (1008)	56 (1069)	56 (1053)	56 (1051)	56 (1054)	56 (1054)
Quadratic $r = 2$	97 (2848)	36 (984)	23 (519)	20 (480)	17 (306)	16 (266)	16 (272)	16 (267)
Cubic $r = 3$	97 (2848)	37 (1087)	22 (508)	19 (471)	15 (269)	14 (224)	13 (221)	13 (213)
Quartic $r = 4$	97 (2848)	39 (1211)	22 (565)	18 (477)	13 (283)	11 (189)	11 (167)	11 (162)
Quintic $r = 5$	97 (2848)	40 (1279)	22 (620)	19 (526)	12 (288)	11 (176)	10 (146)	10 (143)

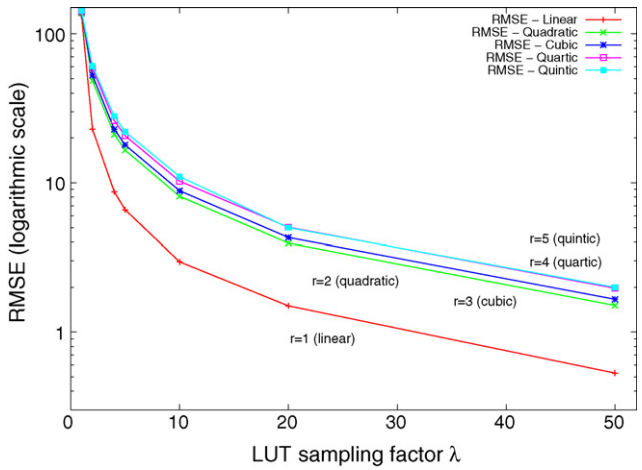


Fig. 2 – RMSE between image rotated with fast B-LUT and image rotated with corresponding conventional B-splines, for various degrees and sampling values. The RMSE axis is logarithmic scaled.

A sampling equal to λ means that the distance between the position used to compute the approximated weight and the position of the real weight is at maximum (1/2λ) (half the distance between two samples). In order to illustrate the intrinsic error of the method, we compute RMSE between an image and the same image displaced by such maximum distance. Fig. 5 displays the values for cubic B-splines computed with a 2D slice.

4.2. *B-LUT interpolation efficiency*

We performed time measurements for both B-LUT and B-spline interpolation. Tests were performed on the previously described 3D images for a rotation transformation. We observed computation time variations when successive interpolations were performed due to processor cache effects. Times were thus measured for 10 different transformations and averaged. Fig. 6 displays the mean computation time of image 1 test, for interpolations with different B-spline degrees (from linear to quintic), for conventional B-spline interpolation and fast B-LUT interpolation with two sampling rates ($\lambda = 10$ and $\lambda = 20$). We also indicate the time due to coefficient computation (boxes at the bottom). The machine was an

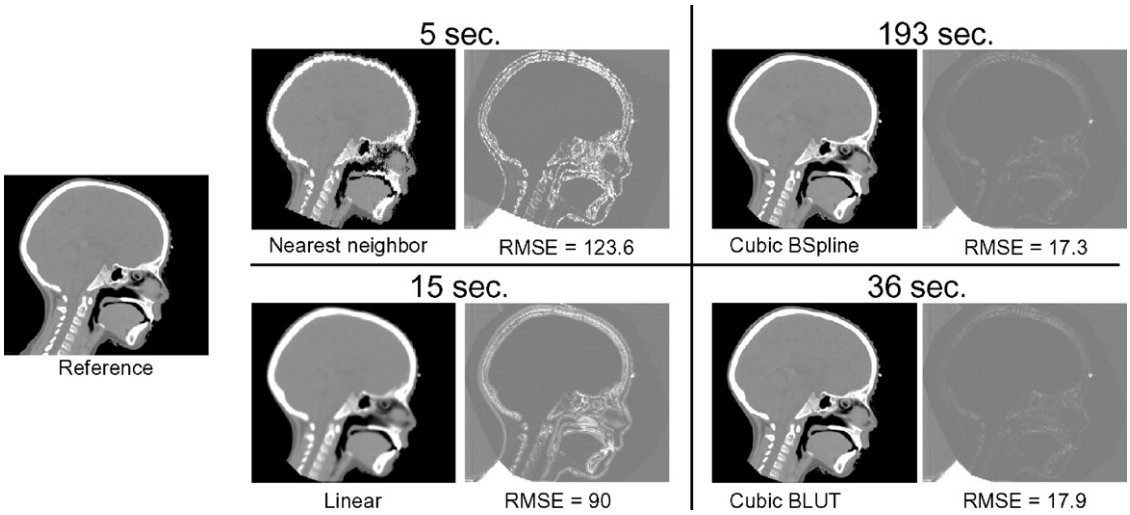


Fig. 3 – The images show the results of the application of 16 successive rotations around an arbitrary axis (360°), with different interpolation methods. Numbers are RMSE errors and computational time of one rotation in seconds (PC 2 GHz). Initial image is 512 × 512 × 222 pixels.

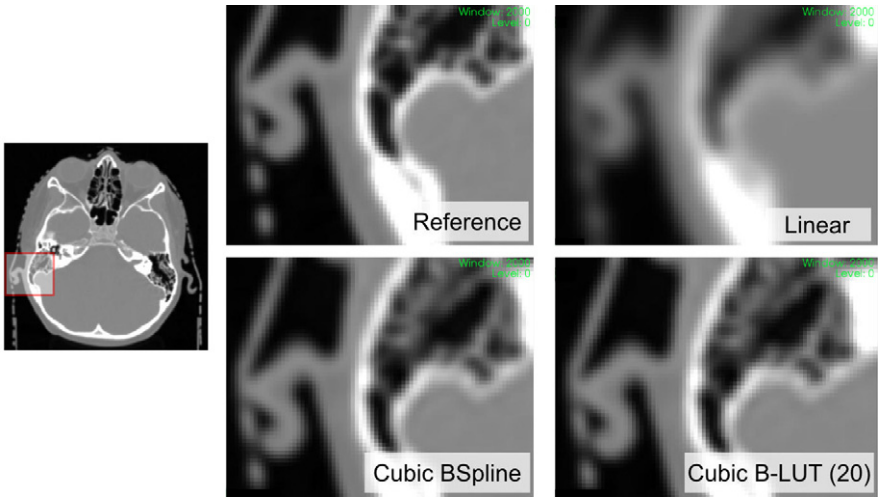


Fig. 4 – The images focus on a small area having important grey levels gradients (bone, soft tissues, air) in order to compare the reference image to the interpolated ones with linear, cubic B-splines and cubic B-LUT (with 20 as sample rate) interpolations.

Intel Core 2 Duo 2.0Ghz. Fig. 7 illustrates the loss of efficiency when increasing the LUT size for cubic B-LUT interpolation.

5. Discussion

Fig. 2 and Table 1 show that, with a sampling rate $\lambda > 10$, the difference between images interpolated with conventional B-splines and B-LUT is very low (visually they are indistinguishable in all tests). The mean of absolute differences was found to be < 3 Hounsfield units for cubic B-splines with $\lambda = 10$ or 2 with $\lambda = 20$. For lower sampling rates, the approximation became insufficient. For example, cubic B-LUT with $\lambda = 5$ performed worse than conventional quadratic B-spline. We also observed that the differences increased with spline degree.

Fig. 6 illustrates the important time gain that can be obtained using B-LUTs. For equivalent B-spline degrees the time is reduced by a factor between 5 (quadratic) and 6 (quintic). It can be seen that quintic B-LUT interpolation was more than twice as fast than conventional cubic B-spline interpolation. For linear interpolation, computational time was greater (factor 1.15) than conventional linear interpolation and we thus do not recommend to use the LUT method for a degree lower than 2. However, it should be mentioned that identical tests performed on a different computer architecture (AMD Athlon(tm) 64 bits 2 GHz) gave different results. For degrees 2 up to 5 similar though slightly lower speedup factors between 4 and 5 were found. For linear interpolation a speedup > 3 was in this case also observed. Such differences could be due to the processor cache size (2 MB for Intel and 512 kB for AMD), and specific handling of specific arithmetic operations. Finally

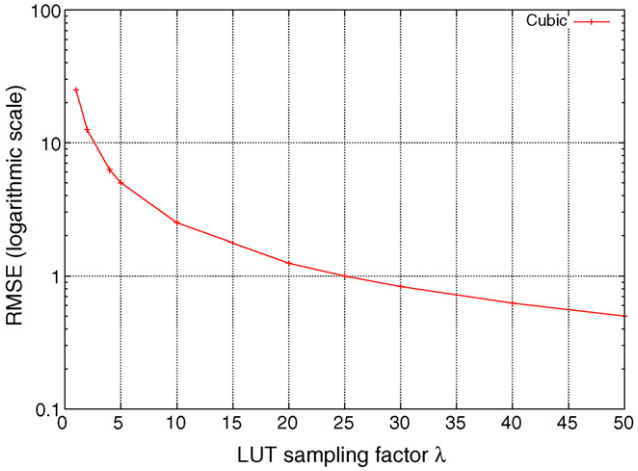


Fig. 5 – RMSE between an image and a translation of the maximum amount of error (1/2λ) for cubic B-spline interpolation.

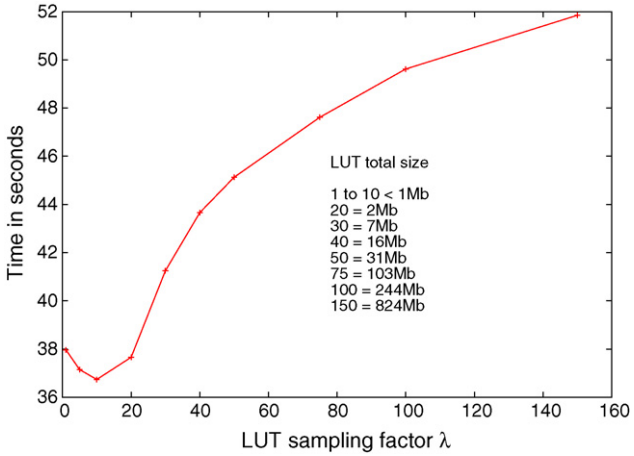


Fig. 7 – Computation time for one rotation of the 3D test image with cubic interpolation and several sampling values, from λ = 1 to λ = 150. Total sizes of the LUT for the different sampling λ are indicated in Mb.

we also draw the reader's attention to the time required for coefficients computation (direct transformation process using recursive filters) which is negligible compared to the time needed by the interpolation.

The influence of the LUT size on the computational time, controlled by the sampling factor λ is illustrated in Fig. 7. For $\lambda \leq 20$, the time is almost unchanged. For $\lambda = 10$, a light speedup compared to lower λ values was observed, but this is probably due to processor cache effect. Raising λ to 100 increased the computational time by 30%, but the whole computational time remains largely lower than the conventional implementation.

An important parameter when choosing an adequate value for the sampling factor λ is the relative density of pixels to be interpolated with respect to the B-spline control point grid d_r. In the presented experiments, d_r was equal to 1. When there are fewer control points than interpolated values ($d_r > 1$), the sampling rate should be increased accordingly to maintain the same level of accuracy. The results presented here should thus be interpreted as guidelines for choosing the *relative* sampling factor $\lambda_r = (\lambda/d_r)$. More explicitly, one can expect to obtain an accuracy corresponding to the measurements presented in Fig. 2 for sampling factor λ_r, when choosing the sampling factor λ such that $\lambda = d_r\lambda_r$. The computational efficiency shown in Fig. 7, will depend however on the actual LUT size, and so on the choice of λ.

The general application of image intensity interpolation was addressed in this work. The proposed B-LUT framework is however applicable in all areas where B-splines are used. A particular example is the case of deformable image registration using free form deformations (FFD) [10] where B-splines are used to represent the sought spatial transform. The optimization of the transform is usually done iteratively, requiring one image to be repeatedly transformed in order to evaluate the current solution. During each iteration the splines representing the transform will be evaluated at the same positions (the voxel positions of the refer-

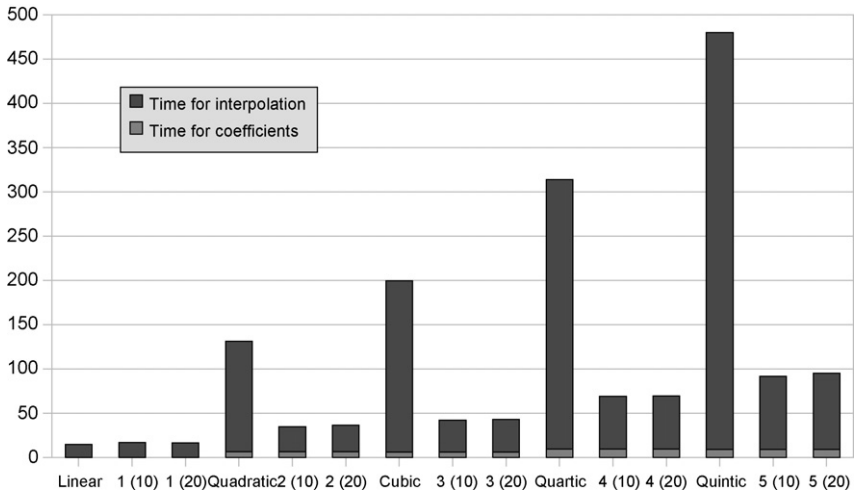


Fig. 6 – Time (in seconds) for different B-spline interpolations: from linear ($r = 1$) to quintic ($r = 5$) degree, with conventional method and fast methods ($\lambda = 10$ and $\lambda = 20$).

Table 2 – Computation time and memory requirement for FFD registration with conventional B-spline, optimized B-spline and the proposed B-LUT method.

FFD methods	Computation time (mn)	Memory requirement (MB)
Conv. B-spline	73	578
Opt. B-spline	30	2721
B-LUT	17	597

ence image). Their corresponding tensor products can be computed once and reused in further iterations, providing considerable acceleration at run-time. Such an approach has been implemented in the optimized registration framework of the ITK toolkit [6], available in the ITK *Review* section. The B-spline weights corresponding to *all* considered reference image voxels are precomputed and stored. The memory requirements of this approach (about $(r+1)^d$ times the image memory size in double precision) become very quickly prohibitive, rendering it impossible to run the method for reasonably sized images. To remedy this, we implemented a B-LUT FFD, in which the B-spline spatial transform was replaced by a B-LUT spatial transform. Contrary to the previous approach, the B-LUT deformable registration, requires only one n-dimensional B-spline support region to be sampled and stored. By careful choice of the sampling factor, the B-LUT FFD yields an exact B-spline representation.

We compared the optimized ITK B-spline FFD registration method to the B-LUT FFD registration method. Note however that this comparison does not just assess the impact of caching all the B-spline weights with respect to the use of a compact LUT. This because several acceleration mechanisms are included in the optimized ITK B-spline FFD, also affecting the calculation of the Jacobian of the transform and the derivative of the cost function. These mechanisms were difficult and even in some cases impossible to reproduce in the case of the B-LUT FFD.

We ran both registration methods on image pairs which have 128^3 voxels. We placed 10 control points along each dimension (7 control points inside the image region placed every 20 voxels, 3 control points for the required border of the cubic splines), making a total of 1000 control points and parameters to optimize. In case of the B-LUT FFD, we set the sampling factor to 20 along each dimension, obtaining an exact B-spline representation. We ran both methods for 50 iterations on a single thread and recorded the CPU occupation time and memory consumption. In case of the B-spline FFD the method required 30 mn and occupied more than 2.7 GB, while conventional B-spline FFD, without optimization takes more than twice the time and about 578 MB. The B-LUT FFD finished in only 17 mn and requiring 597 MB, only 3% more memory (see Table 2). Tests on 256^3 image pairs could not be performed as the B-spline FFD memory requirements exceeded the 8 GB of RAM on the host machine. The B-LUT FFD reported a memory consumption below 2 GB. As mentioned before, these measurements should be interpreted as indicators of relative performance as the obtained performance may differ on other architectures or for different parameters.

6. Conclusion

We proposed a method to accelerate B-spline interpolation using look-up tables of precomputed B-spline tensor products. Depending on the application, the resulting interpolation will be exact or approximate in which case the accuracy of the B-spline approximation is fully controllable by the user, by varying the sampling factor λ. We obtained speedup factors between 5 and 6 compared to conventional method from quadratic to quintic interpolation, with a very low error. Attention should be given to the relative density of interpolated values with respect to the density of control points when choosing the sampling factor. For applications where the density of the samples is comparable to the density of control points, we recommend using $\lambda = 20$, offering a good compromise. For other relative densities, λ should be changed accordingly to maintain the same accuracy of approximation. In the future, this method could be used together with hardware-based acceleration with GPU [7] to speed up the processing of time critical applications even more.

REFERENCES

[1] M. Unser, Splines: a perfect fit for signal and image processing, IEEE Trans. Signal Process. 16 (6) (1999) 22–38.

[2] M. Unser, A. Aldroubi, M. Eden, B-spline signal processing: Part I. Theory, IEEE Trans. Med. Imaging 41 (2) (1993) 821–833.

[3] M. Unser, A. Aldroubi, M. Eden, B-spline signal processing: Part II. Efficient design and applications, IEEE Trans. Med. Imaging 41 (2) (1993) 834–848.

[4] L. Ibanez, W. Schroeder, L. Ng, J. Cates, The ITK Software Guide, 2nd ed., Kitware, Inc., ISBN 1-930934-15-7, 2005, http://www.itk.org/ItkSoftwareGuide.pdf.

[5] F. Inoa, K. Ooyamab, K. Hagiharaa, A data distributed parallel algorithm for nonrigid image registration, Parallel Comput. 31 (1) (2005) 19–43.

[6] S. Aylward, J. Jomier, S. Barre, B. Davis, L. Ibanez, Optimizing ITKs registration methods for multi-processor, shared-memory systems, Insight Journal http://hdl.handle.net/1926/566.

[7] G.C. Sharp, N. Kandasamy, H. Singh, M. Folkert, Gpu-based streaming architectures for fast cone-beam ct image reconstruction and demons deformable registration, Phys. Med. Biol. 52 (19) (2007) 5771–5783.

[8] M. Modat, G.R. Ridgway, Z.A. Taylor, M. Lehmann, J. Barnes, D.J. Hawkes, N.C. Fox, S. Ourselin, Fast free-form deformation using graphics processing units, Comput. Methods Programs Biomed., doi:10.1016/j.cmpb.2009.09.002, in press.

[9] E. Meijering, W. Niessen, M. Viergever, Quantitative evaluation of convolution-based methods for medical image interpolation, Med. Image Anal. (2001) 111–126.

[10] D. Rueckert, L. Sonoda, C. Hayes, D. Hill, M. Leach, D. Hawkes, Nonrigid, registration using free-form deformations: application to breast MR images, IEEE Trans. Med. Imaging 18 (8) (1999) 712–721.