Fast Interpolation Operations in Non-Rigid Image Registration

Matthew W. Jacobson[a] and Jeffrey A. Fessler[a*]

[a]The University of Michigan, EECS Department, Ann Arbor 48109-2122, MI, USA

ABSTRACT

Much literature on image registration[1–3] has worked with purely geometric image deformation models. For such models, interpolation/resampling operations are often the computationally intensive steps when iteratively minimizing the deformation cost function. This article discusses some techniques for efficiently implementing and accelerating these operations. To simplify presentation, we discuss our ideas in the context of 2D imaging. However, the concepts readily generalize to 3D. Our central technique is a table-lookup scheme that makes somewhat liberal use of RAM, but should not strain the resources of modern processors if certain design parameters are appropriately selected. The technique works by pre-interpolating and tabulating the grid values of the reference image onto a finer grid along one of the axes of the image. The lookup table can be rapidly constructed using FFTs. Our results show that this technique reduces iterative computation by an order of magnitude. When a minimization algorithm employing coordinate block alternation is used, one can obtain still faster computation by storing certain intermediate quantities as state variables. We refer to this technique as *state variable hold-over*. When combined with table-lookup, state variable hold-over reduces CPU time by about a factor two, as compared to table-lookup alone.

1. INTRODUCTION AND MOTIVATION

Let $\mathbb{R}^p$ denote the space of length p column vectors, for whatever $p \in \mathbb{N}$, and similarly let $\mathbb{R}^{M \times N}$ denote the space of $M \times N$ real image arrays. This article is concerned with fast methods of deforming and sampling a continuous image $u(x, y)$, having the parametric form

$$u(x,y) = \sum_{m=1}^{M} \sum_{n=1}^{N} \boldsymbol{u}_{mn} f(x-m) g(y-n), \tag{1.1}$$

where $\boldsymbol{u} \in \mathbb{R}^{M \times N}$ is an array of grid values $\boldsymbol{u}_{mn}$, and $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ are continuous, locally supported interpolating functions, all of which are known. We restrict attention to a parametric family $\mathcal{W}(\boldsymbol{\alpha})u$ of deformation operations

$$\left[\mathcal{W}(\boldsymbol{\alpha})u\right](x,y) \triangleq u(d^X(x,y|\boldsymbol{\alpha}_X), d^Y(x,y|\boldsymbol{\alpha}_Y)). \tag{1.2}$$

Here $\boldsymbol{\alpha}_X$, $\boldsymbol{\alpha}_Y$, and $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_X, \boldsymbol{\alpha}_Y)$ are vectors of deformation parameters and, given the x-direction deformation basis functions $\left\{b_k^X(x,y)\right\}_{k=1}^{K_x}$ (and similarly for the y-direction),

$$d^X(x,y|\boldsymbol{\alpha}_X) \triangleq x + \sum_{k=1}^{K_x} \alpha_{X,k} b_k^X(x,y) \tag{1.3}$$

$$d^Y(x,y|\boldsymbol{\alpha}_Y) \triangleq y + \sum_{k=1}^{K_y} \alpha_{Y,k} b_k^Y(x,y). \tag{1.4}$$

Such image transformations are a commonly considered family[2,3] in non-rigid intensity-based image registration. When $\mathcal{W}(\boldsymbol{\alpha})u$ is sampled at a set of test points $\{(x_j, y_j)\}_{j=1}^{J}$ then, combining (1.1) and (1.2), we obtain, for each j,

$$\mathcal{W}(\boldsymbol{\alpha})u(x_j,y_j) = \sum_{m,n} f(d^X(x_j,y_j|\boldsymbol{\alpha}_X) - m) g(d^Y(x_j,y_j|\boldsymbol{\alpha}_Y) - n) \boldsymbol{u}_{mn} \triangleq \left[\boldsymbol{W}_{fg}(\boldsymbol{\alpha})\boldsymbol{u}\right]_j, \tag{1.5}$$

This work was supported in part by NIH/NCI grant 1P01 CA87634.

Medical Imaging 2005: Image Processing, edited by J. Michael Fitzpatrick, Joseph M. Reinhardt, Proc. of SPIE Vol. 5747 (SPIE, Bellingham, WA, 2005)
1605-7422/05/$15 · doi: 10.1117/12.592243

Thus, deforming and sampling the continuous image u is equivalent to applying a linear transformation $\boldsymbol{W}_{fg}(\boldsymbol{\alpha})\boldsymbol{u} : \mathbb{R}^{M\times N} \rightarrow \mathbb{R}^J$ to the discrete parameter array $\boldsymbol{u}$. The form of $\boldsymbol{W}_{fg}(\boldsymbol{\alpha})$, however, depends non-linearly on $\boldsymbol{\alpha}$. We shall call operators of this form *discrete warping operators*.

Sampling at all (x_j, y_j) in (1.3) and (1.4) also leads to the quantities

$$\boldsymbol{d}_j^X(\boldsymbol{\alpha}_X) \stackrel{\triangle}{=} x_j + \sum_{k=1}^{K_x} \alpha_{X,k} b_k^X(x_j, y_j) \tag{1.6}$$

$$\boldsymbol{d}_j^Y(\boldsymbol{\alpha}_Y) \stackrel{\triangle}{=} y_j + \sum_{k=1}^{K_y} \alpha_{Y,k} b_k^Y(x_j, y_j). \tag{1.7}$$

or in matrix-vector form,

$$\boldsymbol{d}^X(\boldsymbol{\alpha}_X) \stackrel{\triangle}{=} \boldsymbol{x} + \boldsymbol{B}_X\boldsymbol{\alpha}_X \tag{1.8}$$

$$\boldsymbol{d}^Y(\boldsymbol{\alpha}_Y) \stackrel{\triangle}{=} \boldsymbol{y} + \boldsymbol{B}_Y\boldsymbol{\alpha}_Y. \tag{1.9}$$

That is, $\boldsymbol{B}_X$ is a $J \times K_x$ matrix whos columns are the basis function samples $b_k^X(x_j, y_j)$ (and analogously for $\boldsymbol{B}_Y$). We refer to $\boldsymbol{B}_X$ and $\boldsymbol{B}_Y$ as *deformation basis matrices*. We also define $\boldsymbol{d}(\alpha) \stackrel{\triangle}{=} [\boldsymbol{d}^X(\boldsymbol{\alpha}_X)|\boldsymbol{d}^Y(\boldsymbol{\alpha}_Y)]$ as the $J \times 2$ matrix whose j-th row $\boldsymbol{d}_j(\boldsymbol{\alpha}) = (\boldsymbol{d}_j^X(\boldsymbol{\alpha}_X), \boldsymbol{d}_j^Y(\boldsymbol{\alpha}_Y))$ is the transformation of the test point (x_j, y_j).

Based on (1.5), we can see that evaluating operators of the form $\boldsymbol{W}_{fg}(\boldsymbol{\alpha})\boldsymbol{u}$ simply involve interpolating the grid values $\boldsymbol{u}_{mn}$ at various locations $\boldsymbol{d}_j(\boldsymbol{\alpha})$. Fast methods for doing so are relevant to registration problems of the form

$$\text{min.} \quad \Phi(\boldsymbol{\alpha}) = \psi(\boldsymbol{W}_{hh}(\boldsymbol{\alpha})\boldsymbol{u}) \quad \text{s.t.} \quad \boldsymbol{\alpha} \in \Theta. \tag{1.10}$$

Here h is an interpolator, Θ is a feasible set of deformations, and $\psi(\cdot) : \mathbb{R}^J \rightarrow \mathbb{R}$ assigns a cost to the deformed image samples $[\boldsymbol{W}_{hh}(\boldsymbol{\alpha})\boldsymbol{u}]_j = \mathcal{W}(\boldsymbol{\alpha})u(x_j, y_j)$ reflecting how well they fit some target data set. Iterative optimization procedures for solving (1.10) will require evaluating Φ and some number of its derivatives at various $\boldsymbol{\alpha}$. It is clear from (1.10) that it is necessary to carry out the discrete warping operation $\boldsymbol{W}_{hh}(\boldsymbol{\alpha})\boldsymbol{u}$ when evaluating Φ. Furthermore, differentiating Φ requires additional discrete warpings associated with the derivatives of h. This is a straightforward consequence of the chain rule. For example, differentiating with respect to $\boldsymbol{\alpha}_X$ leads to,

$$\begin{aligned}\nabla_{\boldsymbol{\alpha}_X}\Phi(\boldsymbol{\alpha}) &= \nabla_{\boldsymbol{\alpha}_X}\boldsymbol{d}^X(\boldsymbol{\alpha}_X) \cdot \nabla_{\boldsymbol{d}^X}\boldsymbol{W}_{hh}(\boldsymbol{\alpha})\boldsymbol{u} \cdot \nabla\psi(\boldsymbol{W}_{hh}(\boldsymbol{\alpha})\boldsymbol{u}) \\ &= \boldsymbol{B}_X^T\left[\mathcal{D}\left\{\boldsymbol{W}_{\dot{h}h}(\boldsymbol{\alpha})\boldsymbol{u}\right\}\right]\nabla\psi(\boldsymbol{W}_{hh}(\boldsymbol{\alpha})\boldsymbol{u}).\end{aligned} \tag{1.11}$$

where, $\mathcal{D}\{\boldsymbol{z}\}$ denotes a the diagonal matrix whose diagonal is the vector $\boldsymbol{z}$. Thus, we see that taking a derivative leads to an expression involving not one, but two discrete warping operations, $\boldsymbol{W}_{hh}(\boldsymbol{\alpha})\boldsymbol{u}$ and $\boldsymbol{W}_{\dot{h}h}(\boldsymbol{\alpha})\boldsymbol{u}$. Obviously, taking higher derivatives will lead to an even greater preponderance of such operations.

The computational expense of optimization algorithms for (1.10) is therefore dictated by the time needed to manipulate $\boldsymbol{B}_X$ (or $\boldsymbol{B}_Y$), to carry out the various interpolation operations associated with $\boldsymbol{W}_{hh}(\boldsymbol{\alpha})\boldsymbol{u}$ and its derivatives, and to evaluate the requisite derivatives of $\psi(\cdot)$. As we shall discuss, the manipulation of $\boldsymbol{B}_X$ is typically a minor computational issue compared to the discrete warpings. These matrices usually have a small number of non-sparse elements or an otherwise favorable structure.

The computational expense of manipulating $\psi(\cdot)$ is problem-specific. In this article, we shall be concerned mainly with the least squares cost function

$$\Phi_{\text{LS}}(\boldsymbol{\alpha}) = \frac{1}{2}||\boldsymbol{W}_{hh}(\boldsymbol{\alpha})\boldsymbol{u} - \boldsymbol{v}||^2. \tag{1.12}$$

That is, $\psi(\boldsymbol{z}) = ||\boldsymbol{z} - \boldsymbol{v}||^2/2$, where $||\cdot||$ denotes the standard Euclidean norm and $\boldsymbol{v} \in \mathbb{R}^J$ is a given vector of target image values. Techniques for minimizing (1.12) have been explored in various works.[1–3] For this problem, the form of $\psi(\cdot)$ and its derivatives is trivial compared to the discrete warpings. Thus, the discrete warpings are the dominant computational burden, and the acceleration techniques that we propose make a significant difference in overall CPU time. For other $\psi(\cdot)$, the acceleration techniques may also be helpful to some degree.

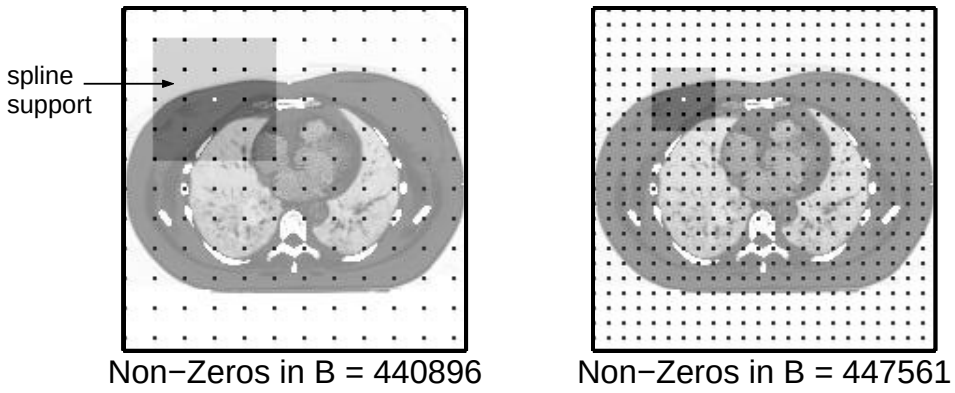

Figure 1. Depiction of a typical system of deformation basis functions, both coarse and fine, and the relationship of its geometry to the grid of $\boldsymbol{u}$. In this illustration, $\boldsymbol{u}$ is a 184×184 torso phantom. Irrespective of the fineness of the deformation grid, the number of non-zero entries in $\boldsymbol{B}_C$, $C \in \{X, Y\}$ is about 450000.

In the next section, we discuss, in a bit more detail, the general considerations affecting the speed of the discrete warping operations. In Section 3, we present some techniques for accelerating the interpolation operations involved. One of these techniques, which we call *state variable hold-over*, applies to algorithms that alternately update $\boldsymbol{\alpha}_X$ and $\boldsymbol{\alpha}_Y$, and is based on saving in memory certain intermediate quantities associated with whichever variable is being held fixed. The second technique is a table-lookup scheme in which $\boldsymbol{u}$ is pre-interpolated at small intervals along (at least) one axis. As we show, the pre-interpolation can be done rapidly using FFT operations. Finally, in Section 4, we test our various techniques on the registration of two simulated torso phantoms.

2. GENERAL COMPUTATIONAL ASPECTS

2.1. The Role of the Deformation Basis Matrix

It is apparent from (1.11), that differentiating the cost function Φ requires multiplication by $\boldsymbol{B}_X^T$ and $\boldsymbol{B}_Y^T$. In addition, it is important to remember that a discrete warping $\boldsymbol{W}_{fg}(\boldsymbol{\alpha})\boldsymbol{u}$ depends on $\boldsymbol{\alpha}$ only through $\boldsymbol{d}^X(\boldsymbol{\alpha}_X)$ and $\boldsymbol{d}^X(\boldsymbol{\alpha}_Y)$ (see (1.5)), which in turn depend on $\boldsymbol{\alpha}$ only through $\boldsymbol{B}_X\boldsymbol{\alpha}_X$ and $\boldsymbol{B}_Y\boldsymbol{\alpha}_Y$ (see (1.8) and (1.9)), respectively. It is therefore pertinent to consider the structure of $\boldsymbol{B}_X$ and $\boldsymbol{B}_Y$ and the computational demands that manipulating them entails.

At this time, it is convenient to define

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_X & 0 \\ 0 & \boldsymbol{B}_Y \end{bmatrix}.$$

As mentioned, the columns of $\boldsymbol{B}_X$ are the samples $b_k^X(x_j, y_j)$ of the x-direction basis functions (and similarly for $\boldsymbol{B}_Y$). The form of $\boldsymbol{B}$ is therefore dictated by the choice of deformation basis functions. A typical scheme by which basis functions are chosen is depicted in Figure 1. Shifted copies of locally supported functions (typically tensor products of B-splines) are centered at the nodes of a deformation space grid, usually a coarse subset of the grid of the $\boldsymbol{u}_{mn}$ parameters. One makes the grid finer (reducing the support of the basis functions accordingly) to span finer deformations. Since the entries of $\boldsymbol{B}_X$ and $\boldsymbol{B}_Y$ are derived from shifted-copies of functions, these matrices have a circulant-like structure.

When the test points $\{(x_j, y_j)\}_{j=1}^J$ are simply the set of integer cordinates $\{(m, n)\}$, $1 \le m \le M$, $1 \le n \le N$, the number of non-zero entries in $\boldsymbol{B}_X$ (or $\boldsymbol{B}_Y$) is approximately MN times the area (as measured in number of nodes) of each of the basis functions. (Note that this number is constant irrespective of the fineness of the deformation space grid). Thus, for example, when the basis functions are cubic B-splines, the number of non-zero entries of $\boldsymbol{B}_X$ is about $16MN$ and the matrix can easily be stored in RAM. In 3D problems, the storage requirements may be prohibitive. However, because the $\boldsymbol{B}_X$ and $\boldsymbol{B}_Y$ matrices have a circulant-like structure, fast convolution techniques can be used to multiply with these matrices and their transposes.

When a fine deformation grid is used, the support of the basis functions is relatively small. Together with the circulant-like structure of $\boldsymbol{B}_X$ and $\boldsymbol{B}_Y$, this implies that matrix products of the form $\boldsymbol{B}_C^T \mathcal{D}\{\boldsymbol{z}\} \boldsymbol{B}_C$, $C \in \{X, Y\}$, and hence also

$$B^T \mathcal{D}\{v\} B =$$

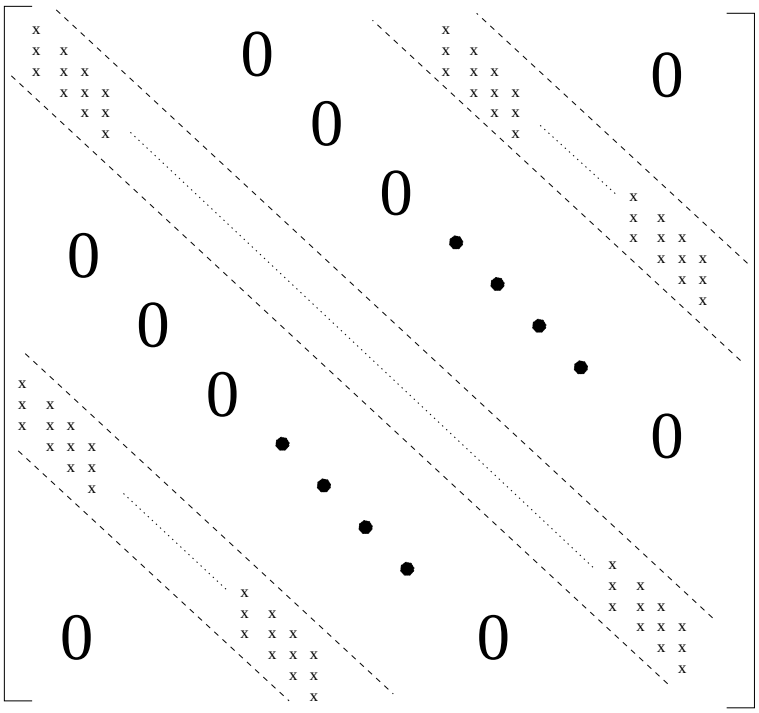

Figure 2. Depiction of the multi-banded structure of $\boldsymbol{B}^T \mathcal{D}\{\boldsymbol{z}\}\boldsymbol{B}$.

those of the form $\boldsymbol{B}^T \mathcal{D}\{\boldsymbol{z}\}\boldsymbol{B}$, have a sparse, multi-banded structure as depicted in Figure 2. Such matrices are rapidly inverted by sparse implementations of Gaussian-elimination. The advantage of this in the least squares problem will be made apparent in Section 2.3.

Manageable $\boldsymbol{B}$ matrices are also obtained when locally supported basis functions (of whatever form) are situated at only a few select locations in the image (such as when the significant deformation is known to occur in a particular region). They would be manageable simply because their size and number of non-zero entries may be comparatively small.

2.2. The Constituents of Interpolation/Sampling Operations

As mentioned, the evaluation of a discrete warping operator $\boldsymbol{W}_{fg}(\boldsymbol{\alpha})\boldsymbol{u}$ involves first computing $\boldsymbol{d}^X(\boldsymbol{\alpha}_X)$ and $\boldsymbol{d}^Y(\boldsymbol{\alpha}_Y)$ and then (see (1.5)) interpolating the parameters $\boldsymbol{u}_{mn}$ at every deformed test point $\boldsymbol{d}_j(\boldsymbol{\alpha}) = (\boldsymbol{d}_j^X(\boldsymbol{\alpha}_X), \boldsymbol{d}_j^Y(\boldsymbol{\alpha}_Y))$. Some of the ideas relevant to doing so are depicted in Figure 3.

There are four principal steps to each interpolation. Firstly, one locates the region of non-trivial grid values $\boldsymbol{u}_{mn}$ that contribute to the interpolated value $[\boldsymbol{W}_{fg}(\boldsymbol{\alpha})\boldsymbol{u}]_j$. Because multiplicatively separable interpolation is assumed here and because f and g are locally supported interpolating functions, there will be some local rectangular region of grid values $\boldsymbol{u}_{mn}$ near each given $\boldsymbol{d}_j$ that contribute to $[\boldsymbol{W}_{fg}(\boldsymbol{\alpha})\boldsymbol{u}]_j$. This region will be the intersection of a $\boldsymbol{\alpha}_X$ dependent strip and a $\boldsymbol{\alpha}_Y$ dependent strip as shown in Figure 3. The width of the strip is the width of f or g, as appropriate. Henceforth, we shall call such a region an *interpolation rectangle*. Next, one accesses the associated grid values $\boldsymbol{u}_{mn}$ in this region from memory. As this is done for every j, many non-contiguous memory accesses are required. Therefore, this step incurs appreciable CPU time. Thirdly, the interpolation weights applied to the different $\boldsymbol{u}_{mn}$ is computed. Typically, this involves the repeated evaluation of non-linear or transcendental functions. Therefore, this step also incurs appreciable CPU time. Finally, the weights are applied to the $\boldsymbol{u}_{mn}$ and summed to give $[\boldsymbol{W}_{fg}(\boldsymbol{\alpha})\boldsymbol{u}]_j$. The computational effort in all of these steps is proportional to the number of grid points $\boldsymbol{u}_{mn}$ in each interpolation rectangle.

2.3. The Case of the Least-Squares Problem

In this section, we look at the form of the derivatives of the least squares cost function Φ_{LS} and see more concretely why, for this problem, discrete warpings are the computational bottleneck in derivative-based minimization methods. We assume

throughout that the interpolator h is as many times differentiable as the context requires.

Any first order method (e.g., steepest descent) will rely on the gradient, which has the form

$$\nabla\Phi_{\mathrm{LS}}(\boldsymbol{\alpha}) = \boldsymbol{B}^T \begin{bmatrix} \boldsymbol{g}_X(\boldsymbol{\alpha}) \\ \boldsymbol{g}_Y(\boldsymbol{\alpha}) \end{bmatrix}, \tag{2.1}$$

where,

$$\begin{aligned} \boldsymbol{e}(\boldsymbol{\alpha}) &= \boldsymbol{W}_{hh}(\boldsymbol{\alpha})\boldsymbol{u} - \boldsymbol{v} \\ \boldsymbol{g}_X(\boldsymbol{\alpha}) &= \mathcal{D}\left\{\boldsymbol{e}(\boldsymbol{\alpha})\right\} \boldsymbol{W}_{\dot{h}h}(\boldsymbol{\alpha})\boldsymbol{u} \\ \boldsymbol{g}_Y(\boldsymbol{\alpha}) &= \mathcal{D}\left\{\boldsymbol{e}(\boldsymbol{\alpha})\right\} \boldsymbol{W}_{h\dot{h}}(\boldsymbol{\alpha})\boldsymbol{u}. \end{aligned}$$

Furthermore, any second order (e.g., Newton's) method will require not only the gradient, but also the Hessian, given by

$$\nabla^2\Phi_{\mathrm{LS}}(\boldsymbol{\alpha}) = \boldsymbol{B}^T \begin{bmatrix} \boldsymbol{H}_{XX}(\boldsymbol{\alpha}) & \boldsymbol{H}_{XY}(\boldsymbol{\alpha}) \\ \boldsymbol{H}_{XY}^T(\boldsymbol{\alpha}) & \boldsymbol{H}_{YY}(\boldsymbol{\alpha}) \end{bmatrix} \boldsymbol{B}, \tag{2.2}$$

where,

$$\begin{aligned} \boldsymbol{H}_{XX} &= \mathcal{D}\left\{\boldsymbol{W}_{\ddot{h}h}\boldsymbol{u}\right\}\mathcal{D}\left\{\boldsymbol{e}\right\} + \mathcal{D}\left\{\boldsymbol{W}_{\dot{h}h}\boldsymbol{u}\right\}^2 \\ \boldsymbol{H}_{YY} &= \mathcal{D}\left\{\boldsymbol{W}_{h\ddot{h}}\boldsymbol{u}\right\}\mathcal{D}\left\{\boldsymbol{e}\right\} + \mathcal{D}\left\{\boldsymbol{W}_{h\dot{h}}\boldsymbol{u}\right\}^2 \\ \boldsymbol{H}_{XY} &= \mathcal{D}\left\{\boldsymbol{W}_{\dot{h}\dot{h}}\boldsymbol{u}\right\}\mathcal{D}\left\{\boldsymbol{e}\right\} + \mathcal{D}\left\{\boldsymbol{W}_{h\dot{h}}\boldsymbol{u}\right\}\mathcal{D}\left\{\boldsymbol{W}_{\dot{h}h}\boldsymbol{u}\right\}. \end{aligned}$$

We see that the expressions for the various derivatives are replete with discrete warping operations. Moreover, the only other operations of any significance are the multiplications with $\boldsymbol{B}$ and its constituents. As mentioned, this was to be expected based on the simple form of $\boldsymbol{\psi}(\cdot)$ for the problem.

What is noteworthy beyond this, however, is that the the constituents of the Hessian $\boldsymbol{B}_X^T\boldsymbol{H}_{XX}\boldsymbol{B}_X, \boldsymbol{B}_Y^T\boldsymbol{H}_{YY}\boldsymbol{B}_Y, \ldots$ have the multi-banded form of Figure 2, since $\boldsymbol{H}_{XX}$, $\boldsymbol{H}_{YY}$, and $\boldsymbol{H}_{XY}$ are all diagonal. Consequently, the total Hessian $\nabla^2\Phi_{\mathrm{LS}}$ also has this structure. This means that, not only do the manipulations of $\boldsymbol{B}$ not contribute significantly to computational overhead, but Hessian inversions, when used in Newton-like methods, should also not contribute significantly. The inversions may be done rapidly with Gaussian elimination.

3. ACCELERATION TECHNIQUES

3.1. State Variable Hold-Over

In this section, we discuss an acceleration technique that we call *state variable hold-over*. The technique is applicable to algorithms that alternatingly update $\boldsymbol{\alpha}_X$ and $\boldsymbol{\alpha}_Y$ as follows:

ALGORITHM 3.1. *[Skeleton of a block alternating algorithm]*

Initialization. Choose a number of iterations $L > 0$ and sub-iteration parameters $L_X, L_Y > 0$.

For $i = 0, \ldots, L$

  For $k := 0, \ldots, L_X - 1$

    Update $\boldsymbol{\alpha}_X$ only: $(\boldsymbol{\alpha}_X^{\mathrm{old}}, \boldsymbol{\alpha}_Y^{\mathrm{old}}) \mapsto (\boldsymbol{\alpha}_X^{\mathrm{new}}, \boldsymbol{\alpha}_Y^{\mathrm{old}})$

  end

  For $k := 0, \ldots, L_Y - 1$

    Update $\boldsymbol{\alpha}_Y$ only: $(\boldsymbol{\alpha}_X^{\mathrm{old}}, \boldsymbol{\alpha}_Y^{\mathrm{old}}) \mapsto (\boldsymbol{\alpha}_X^{\mathrm{old}}, \boldsymbol{\alpha}_Y^{\mathrm{new}})$

  end

end

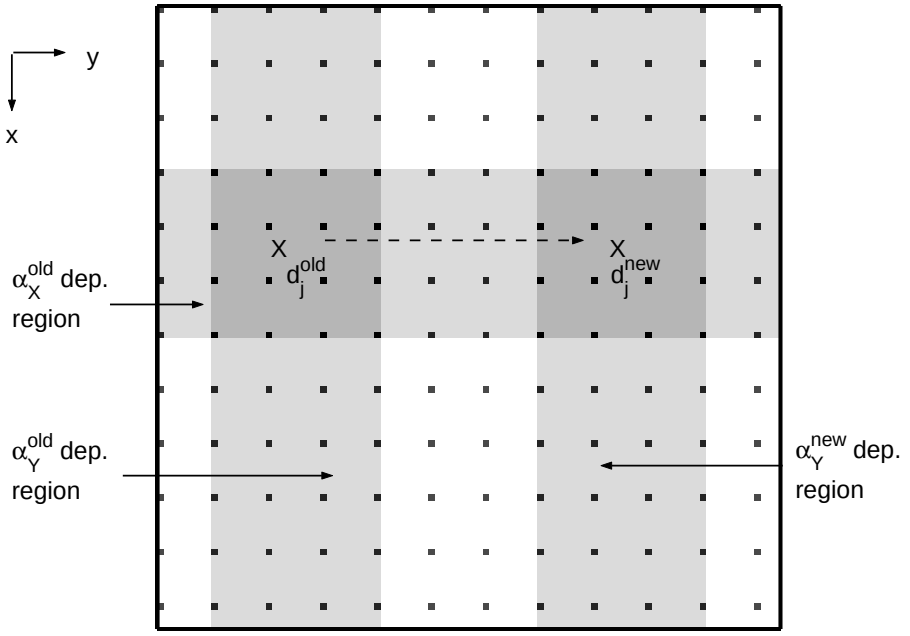

Figure 3. Locating relevant grid neighborhoods in interpolation operations relevant to discrete warpings. Also, a depiction of how state variable hold-over facilitates this.

When certain variables are held fixed, quantities that are functions of the fixed variables can be held in memory and re-used each time the non-fixed variables are updated. Thus, additional, consecutive updates of the non-fixed variable are more cheaply obtained than if all quantities were freshly computed. How this concept applies to the interpolation operations presently discussed is illustrated in Figure 3 for the case where $\boldsymbol{\alpha}_X$ is the fixed variable. There, we see that, when $\boldsymbol{\alpha}_Y$ is updated, the x-coordinate of $\boldsymbol{d}_j^{\text{new}}$ is the same as of $\boldsymbol{d}_j^{\text{old}}$. Thus, the local interpolation rectangle has the same x-coordinates as for $\boldsymbol{d}_j^{\text{old}}$. Locating the new interpolation rectangle only requires computing the new y-coordinates. Similarly, the x-direction interpolation weights applied to the new rectangle is the same as for the old. Hence, only the y-direction weights need to be recomputed.

3.2. Table-Lookup Elimination

In this section, we propose a table-lookup technique that eliminates the need to interpolate along at least one axis. The idea is to pre-interpolate along that axis at a small sampling interval, compared to the original grid spacing of $\boldsymbol{u}$, and tabulate the results. During the iterations of the minimization algorithm, interpolation along this axis may then be replaced with nearest-neighbor table-lookup. Figure 4 illustrates this for when the x-axis (but not the y-axis) is finely sampled. As shown in the figure, the number of grid values in each local interpolation rectangle reduces as compared to Figure 3 when no pre-inteprolation is done. The reduction factor is the support width of the interpolator for the finely sampled direction.[†] Accordingly, the overall time to compute $\boldsymbol{W}_{fg}(\boldsymbol{\alpha})\boldsymbol{u}$ reduces by the same factor.

3.2.1. Incorporating State Variable Hold-Over

A further advantage to the table-lookup scheme becomes apparent when one considers incorporating state variable hold-over (see Section 3.1). Suppose, as in Figure 4, that $\boldsymbol{\alpha}_X$, but not $\boldsymbol{\alpha}_Y$, was being updated as in Algorithm 3.1. If state variable hold-over is used, the weights for the y-direction interpolation (which is the only direction in which interpolation occurs) are held in memory throughout the consecutive updates of $\boldsymbol{\alpha}_X$. Thus, updating $\boldsymbol{W}_{fg}(\boldsymbol{\alpha})\boldsymbol{u}$ simply involves multiplying the values in the new interpolation rectangle by these already available weights and summing. No new weight calculations are required.

[†] The effect of this reduction will be particularly prominent if the interpolator for the finely sampled direction is very wide compared to those of other directions.

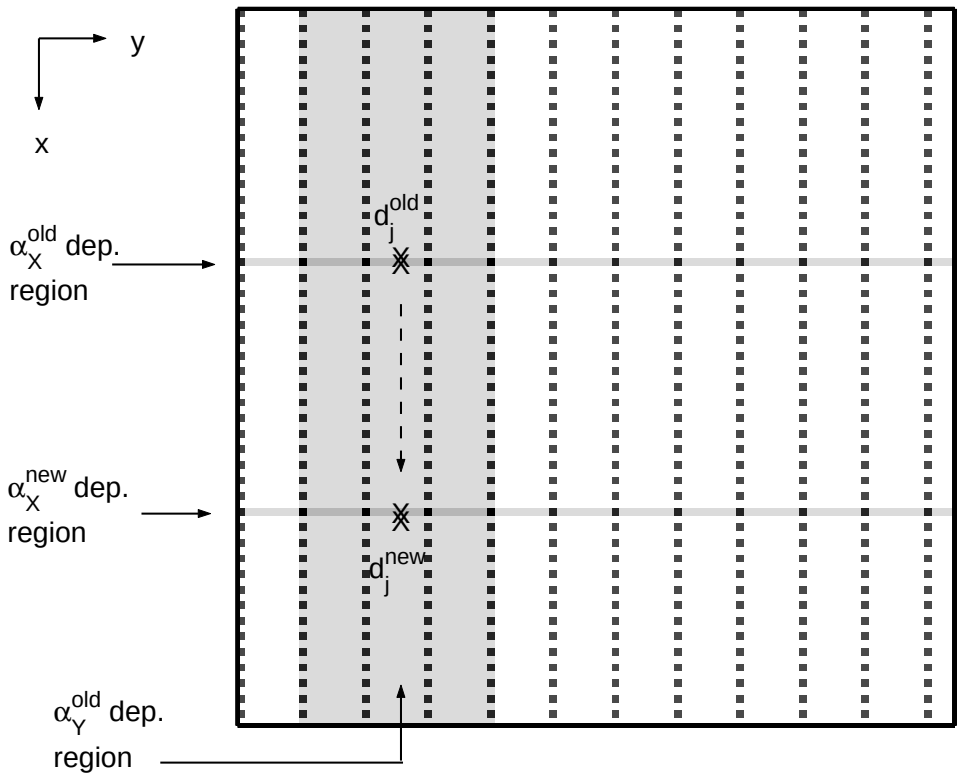

Figure 4. Illustration of table-lookup elimination.

3.2.2. Generating the Table

The table-assisted interpolations in Figure 4 approximate each $[\boldsymbol{W}_{fg}(\boldsymbol{\alpha})\boldsymbol{u}]_j$ as follows (cf. Equation (1.5)),

$$[\boldsymbol{W}_{fg}(\boldsymbol{\alpha})\boldsymbol{u}]_j \approx \sum_{n=1}^{N} g(y-n) \left[\sum_{p=1}^{M} f(x-p)\boldsymbol{u}_{pn} \right] \Bigg|_{\substack{x=\langle \boldsymbol{d}_j^X(\boldsymbol{\alpha}_X)\rangle \\ y=\boldsymbol{d}_j^Y(\boldsymbol{\alpha}_Y)}} . \tag{3.1}$$

Here $\langle \boldsymbol{d}_j^X \rangle$ is $\boldsymbol{d}_j^X$ rounded to the nearest in a set of finely spaced x-values $\{x_{ms}\}$ where

$$x_{ms} = m + s/S, \qquad 1 \le m \le M-1, 0 \le s \le S-1. \tag{3.2}$$

The parameter $S \in \mathbb{N}$ is a pre-selected value that controls the fineness of the sampling. This indexing scheme is illustrated in Figure 5.

The table-lookup technique assumes that the quantity $\sum_{p=1}^{M} f(x-p)\boldsymbol{u}_{pn}$ has been pre-tabulated at all x_{ms}. A naive way of constructing such a table would be to loop over the values in $\{x_{ms}\}$ in ascending order and, for each one, to compute the table entries

$$T_{mns} \triangleq \sum_{p=1}^{M} f(x_{ms}-p)\boldsymbol{u}_{pn} \tag{3.3}$$

directly. It proves more efficient, however, to process $\{x_{ms}\}$ in integer-spaced groups, each corresponding to a fixed s

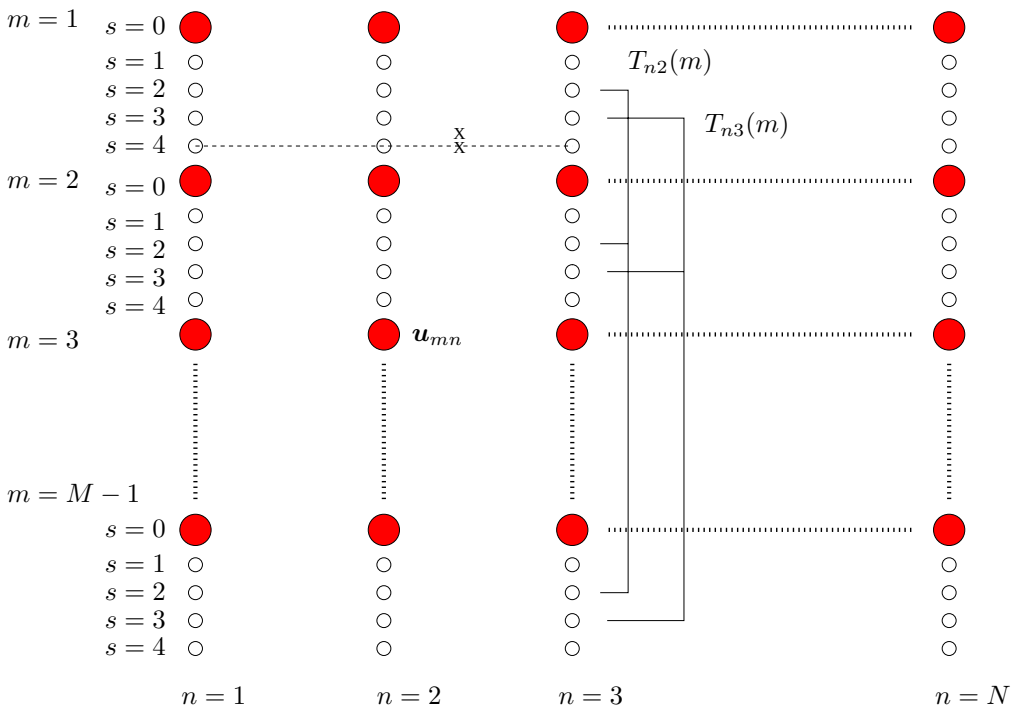

Figure 5. By taking integer-spaced sets of samples, the lookup table can be rapidly constructed via FFTs.

value. Defining $\kappa_{f,s}(m) \triangleq f(m + s/S)$ and incorporating (3.2) into (3.3),

$$\begin{aligned} T_{ns}(m) &= \sum_p f(m + s/S - p)\boldsymbol{u}_{pn} \\ &= \sum_p \kappa_{f,s}(m - p)\boldsymbol{u}_n(p) \\ &= [\kappa_{f,s} \otimes_p \boldsymbol{u}_n](m) \end{aligned} \tag{3.4}$$

where $\otimes_p$ denotes 1D discrete convolution with respect to p. The discrete convolution in (3.4) can be done rapidly via FFTs. Moreover, the FFT of each $\kappa_{f,s}$ depends only on f and can be archived in permanent storage.

The choice of S reflects a compromise between memory consumption and accuracy. Obviously, for larger S, the spacing of the $\{x_{ms}\}$ is smaller and the approximation accuracy of (3.1) increases. However, the storage requirements of the implementation is also magnified by S. The total number of stored grid parameters $\boldsymbol{u}_{mn}$ effectivley increases to SMN and the number of data in the FFT of the $\kappa_{f,s}$ is $O(SM)$. Since f and g are typically gradually varying functions, however, a moderate S value (≤ 10) is likely to be sufficient.

4. COMPUTATIONAL TESTS

We applied the acceleration techniques of Section 3 to the registration of the 128×128 torso image shown in Figure 6(a) with the image shown in Figure 6(b). The absolute difference between these images appears in Figure 6(d). The array of pixel values in Figure 6(a) played the role of the reference image data, $\boldsymbol{u}$. By applying a certain warping $\boldsymbol{W}_{hh}(\boldsymbol{\alpha}^{\text{true}})\boldsymbol{u}$ to $\boldsymbol{u}$, we generated the target image data vector $\boldsymbol{v} \in \mathbb{R}^{128^2}$ which, when rearranged as a 128×128 image, gives the image displayed in Figure 6(b). The test points $\{(x_j, y_j)\}_{j=1}^{J}, J = 128^2$, coincided with the grid points $\{(m, n) : 1 \leq m \leq 128, 1 \leq n \leq 128\}$. The interpolator h was the cubic B-spline functions, each 4 pixels wide. For deformation basis functions, we used 2D tensor products of cubic B-splines, also of width 4. The basis functions $\{b_k^X(x, y)\}_{k=1}^{120}$ were each centered at a pixel in the shaded region about the left lung boundary in Figure 6(c). Similarly, deformation basis functions $\{b_k^Y(x, y)\}_{k=1}^{283}$ were situated at pixels in the shaded region by the right lung. The true parameter values $\alpha_{X,k}^{\text{true}}$ and $\boldsymbol{\alpha}_{Y,k}^{\text{true}}$ were all equal to -10 pixel lengths.

To minimize Φ_{LS}, we employed the following special case of Algorithm 3.1:

ALGORITHM 4.1. *[A Block-Alternating Newton-like Method]*

For $i = 0, \ldots, L$

For $C \in \{X, Y\}$

For $k = 0, \ldots, L_C - 1$

$$\begin{aligned} \boldsymbol{\mathcal{H}} &:= \boldsymbol{B}_C^T \boldsymbol{H}_{CC}(\boldsymbol{\alpha}_X, \boldsymbol{\alpha}_Y) \boldsymbol{B}_C \\ \boldsymbol{g} &:= \boldsymbol{B}_C^T \boldsymbol{g}_C(\boldsymbol{\alpha}_X, \boldsymbol{\alpha}_Y) \\ \boldsymbol{\alpha}_C &:= \boldsymbol{\alpha}_C - \{\boldsymbol{\mathcal{H}}\}_+^{-1} \boldsymbol{g} \end{aligned} \tag{4.1}$$

end

end

end

In (4.1), the transformation $\{\cdot\}_+$ is some operation that makes $\boldsymbol{\mathcal{H}}$ positive definite. When $\{\cdot\}_+$ is simply the identity mapping, $\{\boldsymbol{\mathcal{H}}\}_+ = \boldsymbol{\mathcal{H}}$, these update equations reduce to Newton steps in $\boldsymbol{\alpha}_X$ and $\boldsymbol{\alpha}_Y$ respectively. However, Φ_{LS} is non-convex, so $\{\cdot\}_+$ would normally be something non-trivial.

In the present experiment, $\{\boldsymbol{\mathcal{H}}\}_+$ was the transformation that alters the eigendecomposition of $\boldsymbol{\mathcal{H}}$ by replacing each eigenvalue, λ_k, with $\max\{|\lambda_k|, 0.1\}$.[‡] This can be viewed as a refinement of the Levenberg-Marquardt method.[§] In the Levenberg-Marquardt method, the Hessian eigenvalues are all increased by a common constant. Conversely, in our method, eigenvalues that are sufficiently large to begin with are left alone, thus avoiding unnecessarily large curvatures and slower convergence. The computation of $\{\boldsymbol{\mathcal{H}}\}_+^{-1}$ is significantly more time-consuming than for the $\{\cdot\}_+$ transformation used in the Levenberg-Marquardt method. However, another advantage to the block alternation technique is that the dimensions of $\boldsymbol{\mathcal{H}}$ are smaller than the full Hessian $\nabla^2 \Phi_{\mathrm{LS}}(\boldsymbol{\alpha})$, making the transformation more manageable.

We implemented Algorithm 4.1 using all combinations of the acceleration techniques in Section 3. Each implementation was then tested on the reference and target image data. In all cases, the iteration/sub-iteration loop sizes were $L = 5$ and $L_X = L_Y = 10$. Also, the components of the initial $\boldsymbol{\alpha}$ were set, in each test, to -5 pixel lengths. The implementations employing the table-lookup technique did so with $S = 10$. Two sets of tables were maintained. One set was derived from pre-interpolation (with h and all relevant derivates) in the x-direction. The second set was similarly derived by pre-interpolating in the y-direction.[¶]

In Figures 7 and 8, the progress of each algorithm implementation is plotted versus CPU time. Figure 7 only accounts for the time spent on gradient and Hessian evaluations, which is where all of the interpolation operations occur. Figure 8 plots descent against the total CPU time expended up to that point in the program execution. The latter took into account overhead for generating the lookup tables before the start of the iterations and computing $\{\boldsymbol{\mathcal{H}}\}_+^{-1}$. The trials marked 'Plain' refer to the case where none of the acceleration techniques were used, i.e., interpolations were done entirely-on-the fly. The memory consumption, measured in millions of stored numbers, was also determined for each implementation (see Table 1). For 3D registration, one would require this amount of memory for each image slice.

[‡]The threshold value of 0.1 was empirically chosen. More generally, one desires a constant threshold that is small compared to the Hessian eigenvalues in the neighbourhood of the solution.

[§]The application of Levenberg-Marquardt, without coordinate block alternation, was considered for the least squares registration problem.[2,3]

[¶]Clearly, many variants of this scheme are possible.

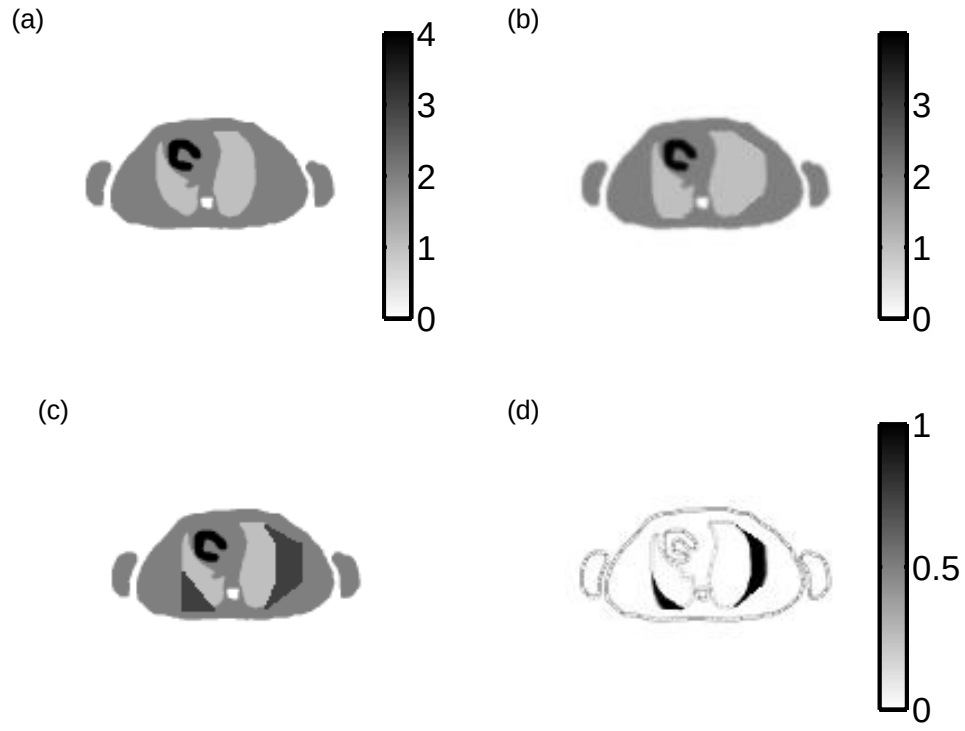

Figure 6. (a) The reference image. **(b)** The target image. **(c)** Locations of the deformation basis functions. Cubic B-splines were placed at all pixels in the darkened regions near the lung boundaries. Basis functions $\{b_k^X\}_{k=1}^{K_x}$ were placed in the region at the left lung and $\{b_k^Y\}_{k=1}^{K_y}$ were placed in the region at the right lung. **(d)** Absolute difference between the reference and target images.

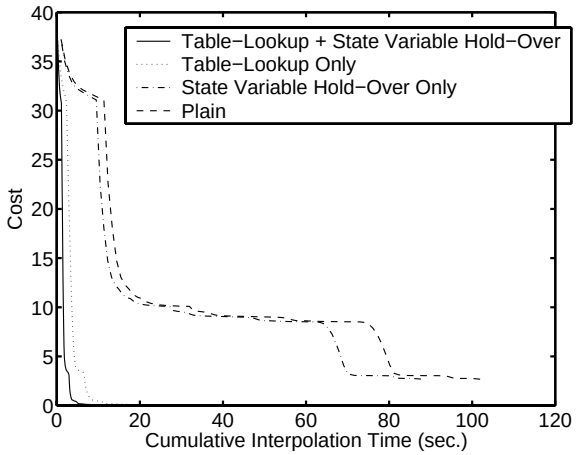

Figure 7. The progress of the algorithm implementations versus cumulative time spent on interpolation operations, i.e., on gradient and Hessian computations.

5. CONCLUSIONS

From Figures 7 and 8, we see that the table-lookup strategy decreases the computation time, as predicted, by a factor of 4, the width of the B-spline interpolators. When table-lookup is used, adding the technique of state variable hold-over leads to additional acceleration by about a factor of 2. For the implementations that used table lookup, a delayed start of less than 2 seconds is seen in Figure 8. This delay is due to the time spent generating tables, but constitutes a small fraction of the overall iteration time.

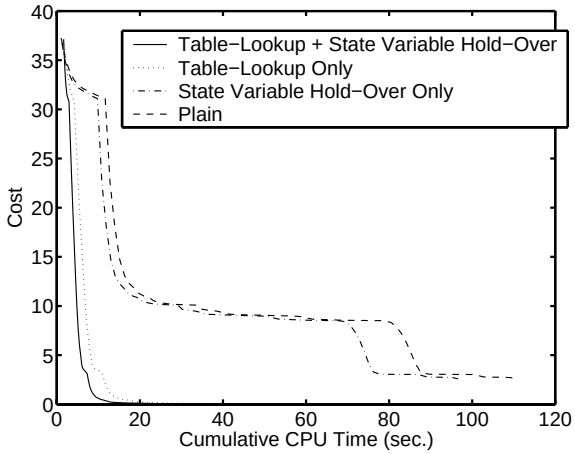

Figure 8. The progress of the algorithm implementations versus total CPU time.

Table 1. Memory consumption for the different implementations.

Method	Consumed RAM (millions of numbers/slice)
Table-Lookup + State Variable Hold-Over	1.65
Table-Lookup Only	1.38
State Variable Hold-Over Only	0.89
Plain	0.28

We can expect still faster computation per iteration when using the Levenberg-Marquardt method to perform the updates in (4.1). This is because Levenberg-Marquardt updates rely only on Gaussian elimination, and will be rapid since the transformation $\{\cdot\}_+$ employed by Levenberg-Marquardt preserves the banded structure of the partial Hessian $\boldsymbol{\mathcal{H}}$ (cf. Section 2.1). However, the descent may be slower with Levenberg-Marquardt than with the $\{\cdot\}_+$ that we have used here.

The acceleration comes at the expense of extra memory consumption. With 128×128 slices and $S = 10$, Table 1 shows that the memory requirement by no means strains the resources of modern processors. For larger data sets $\boldsymbol{u}$, it may be necessary to use smaller S. A variety of other memory-conserving compromises are also possible. For example, the tabulated data could be approximated as 2 byte short integers, as opposed to floating point. Or, instead of maintaining a separate table of finely spaced 1D grid data (as in Figure 4) for each Cartesian direction, one could do so for only one direction. In our future work, we plan to test the impact of these various compromises. Conversely, however, if memory resources are abundant and $\boldsymbol{u}$ is not too large, one could contemplate even more liberal use of memory than was done here. One might even generate a table that finely samples u in all dimensions at once (unlike in Figure 4). In that case, *all* interpolation operations would be eliminated, approximated instead by nearest-neighbor table-lookup.

Acknowledgements

The first author wishes to thank Michael Zibulevsky for inspiring the table lookup ideas discussed in this article.

REFERENCES

1. M. W. Jacobson and J. A. Fessler, "Joint estimation of image and deformation parameters in motion-corrected PET," in *Proc. IEEE Nuc. Sci. Symp. Med. Im. Conf.*, **5**, pp. 3290–4, 2003.
2. J. Kybic, P. Thevenaz, A. Nirkko, and M. Unser, "Unwarping of unidirectionally distorted EPI images," *IEEE Tr. Med. Imag.* **19**, pp. 80–93, Feb. 2000.
3. J. Kybic and M. Unser, "Fast parametric elastic image registration," *IEEE Tr. Im. Proc.* **12**, pp. 1427–42, Nov. 2003.