Available online at www.sciencedirect.com

ScienceDirect

Computerized Medical Imaging and Graphics 30 (2006) 279–289

Computerized Medical Imaging and Graphics

www.elsevier.com/locate/compmedimag

Motion compensation in digital subtraction angiography using graphics hardware

Yu Deuerling-Zheng[a,*], Michael Lell[b], Adam Galant[c], Joachim Hornegger[a]

[a] Pattern Recognition, Friedrich-Alexander University, Martensstrasse 3, D-91058 Erlangen, Germany
[b] Diagnostic Radiology, Friedrich-Alexander University, Maximiliansplatz 1, D-91054 Erlangen, Germany
[c] Siemens Medical Solutions, Hoffmann Estates, IL 60195, USA

Received 23 October 2005; received in revised form 7 May 2006; accepted 23 May 2006

Abstract

An inherent disadvantage of digital subtraction angiography (DSA) is its sensitivity to patient motion which causes artifacts in the subtraction images. These artifacts could often reduce the diagnostic value of this technique. Automated, fast and accurate motion compensation is therefore required. To cope with this requirement, we first examine a method explicitly designed to detect local motions in DSA. Then, we implement a motion compensation algorithm by means of block matching on modern graphics hardware. Both methods search for maximal local similarity by evaluating a histogram-based measure. In this context, we are the first who have mapped an optimizing search strategy on graphics hardware while paralleling block matching. Moreover, we provide an innovative method for creating histograms on graphics hardware with vertex texturing and frame buffer blending. It turns out that both methods can effectively correct the artifacts in most case, as the hardware implementation of block matching performs much faster: the displacements of two 1024×1024 images can be calculated at 3 frames/s with integer precision or 2 frames/s with sub-pixel precision. Preliminary clinical evaluation indicates that the computation with integer precision could already be sufficient.

Keywords: Motion compensation; Image registration; Digital subtraction angiography; Graphics hardware; Block matching

1. Introduction

Digital subtraction angiography (DSA) is the standard modality for visualizing human vasculature. This is achieved by acquiring a sequence of 2D digital X-ray projection images, accompanied by the injection of contrast medium into the vessels of interest. This sequence consists of a few contrast-free image (*mask images*) and mainly contrast-enhanced images (*contrast images*). In a successive step, one selected mask image is subtracted from each of the contrast images, so that the background structures are ideally masked out and only the vessels of interest are displayed.

Obviously the background structures can only be removed completely when they are aligned perfectly and have equal gray-level distributions. However, differences between images of a time sequence are often unavoidable, with patient motion being the most important reason. Consequently, the subtraction images contain artifacts which could impact their diagnostic value considerably. Therefore, retrospective solutions [1–4] such as image registration are required, which retrieve a geometrical transformation accounting for the mismatches caused by patient motion. The aim is to bring the mask and the contrast image in optimal spatial alignment prior to the subtraction.

In order to correct complex motions, motion compensation approaches must adapt to local deformations. Moreover, DSA is being increasingly applied in intervention, which requires fast and automatic methods. However, due to the computational complexity of related techniques, this requirement has been hardly satisfied. In fact, on many today's DSA systems, motion artifacts are still corrected by a simple global translation in x- or y-direction (known as *pixel shift*), which yields seldom satisfactory results.

The purpose of this paper is to find an automated, fast and accurate solution to correct motion artifacts in DSA. At first, we examine a method explicitly designed to correct local deformations in DSA images. In the remainder of this paper, this method is referred as *flexible pixel shift*. Then, in search of a faster computation, we implement a motion compensation algorithm based on block matching on modern graphics hardware. The comparison of these two implementations shows that both

* Corresponding author. Tel.: +49 9131 403043.
E-mail address: deuerling.zheng@t-online.de (Yu Deuerling-Zheng).

0895-6111/$ – see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compmedimag.2006.05.008

implementations are capable of detecting delicate local displacements, as the hardware implementation performs much faster so that it can be used as an interactive tool in clinical routine.

The utilization of graphics hardware (Graphics Processing Units, GPU) for motion compensation and image processing has become a field of active research in recent years. We explore in this paper the possibility and applicability of GPUs for a block matching algorithm with a histogram-based similarity measure. In this context, we are the first one who have mapped an optimizing search strategy for block matching on GPUs. More than that, we propose an innovative method to create histograms with vertex texture and frame buffer alpha blending, which has solved one of the most difficult tasks in general-purposed computation on GPUs.

In the following section, we characterize flexible pixel shift and block matching at the algorithmic level. In Section 3, we describe the GPU-implementation in detail. The application results on several data sets acquired from various anatomical regions are presented in Section 4. The results of a preliminary clinical evaluation are included as well. We finally end up with discussion of the results and the conclusion.

2. Motion compensation techniques for DSA

As mentioned in Section 1, both flexible pixel shift and block matching are capable of detecting local motions in DSA. To this end, the displacement of a point is determined by moving it in its local neighborhood and searching for the maximal similarity. The difference between these two approaches is, flexible pixel shift moves only the point of interest with the given displacement vector while the location of the neighborhood with respect to the entire image is fixed; the displacements of the remaining points within the neighborhood are obtained via bilinear interpolation (Fig. 1(a)). Block matching assumes uniform displacement of all the points within the neighborhood, thus the whole block is moved rigidly with the given displacement vector (Fig. 1(b)). Block matching, also referred as template matching, has been widely applied to detect motion in video images. Its application in DSA can be found in [1,4,2].

Theoretically, flexible pixel shift is expected to yield more accurate estimates for the displacement of a single point, because the best match is calculated exactly by moving that point alone. With block matching, the displacement of a control point is approximately determined by the displacement of the entire block. In cases of delicate motions, this approximation works only when the block is small enough. However, smaller blocks contain little statistical information and may lead to unreliable matching scores. On the other hand, patient motion involving exactly one point as flexible pixel shift assumes, occurs seldom in the real world. Instead, more pixels in the neighborhood are often involved.

In practice, block matching performs much faster than flexible pixel shift. The reason is the large number of bilinear interpolations which are necessary in flexible pixel shift even when the similarity is evaluated with integer displacements. On the contrary, bilinear interpolation is only computed in block matching in two cases: (1) when searching the displacements with subpixel precision (will be illustrated in Section 2.3), and (2) as displacements are only explicitly computed on selected control points, the displacement vector field of the entire image is obtained via interpolation, otherwise artifacts could occur at the block edges. In spite of the differences mentioned above, these two approaches share common properties in many aspects which are discussed in the following.

2.1. Similarity measure

Both flexible pixel shift and block matching evaluate a similarity measure which determines the degree of correspondence between region of interests. A distinct property of DSA is the inflow of contrast medium into the vessels of interest, which causes significant local variation of the

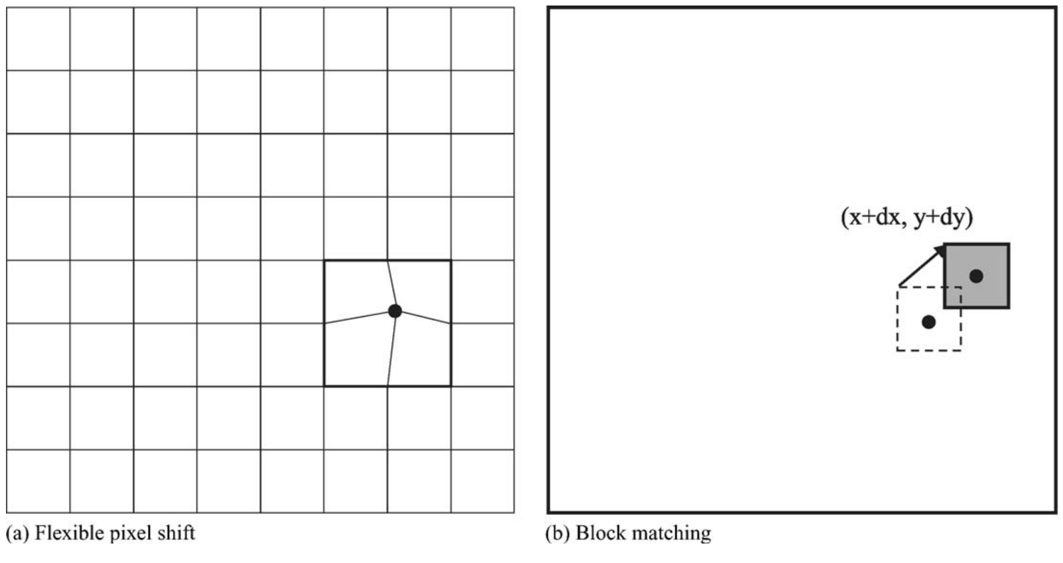

Fig. 1. (a) Only the point of interest is moved, the position of its neighborhood with respect to the entire image is fixed. The displacements of the remaining points in the neighborhood are obtained by bilinear interpolation. (b) The displacement of a point is estimated by moving a surrounding window rigidly in the local search area and optimizing a certain similarity function.

gray level distribution. Most similarity measures proposed so far, assume constant gray level distribution and are consequently not suited for DSA. A robust similarity measure against this gray level variation must take statistical information into account.

If the structures in two images are perfectly aligned, the subtraction image contains ideally only two different groups of intensity values, one for the background and one for the contrast enhanced vessels. This leads to two peaks in the probability density function of the gray values in the subtraction image. Accordingly, the gray value distribution of a block, which is a subset of the subtraction image, will contain either two peaks if vessels are present, or one peak if it consists only of background structures. The amount of the distribution dispersion correlates thus tightly with the structural misalignment.

Generally, a measure quantifying the dispersion should satisfy the following two requirements: (1) it assumes its extremum when the distribution is totally “flat”, i.e., all the entries give the equal contribution; (2) it weights more clustering than dispersion. A well-known example is the Shannon entropy

$$M_{\mathrm{ENT}}(\mathbf{d}) = -\sum_{i} p(i)\,\log\,p(i), \tag{1}$$

where $\mathbf{d} \in \mathbb{Z}^2$ denotes a given displacement vector and $p(i)$ denotes the probability distribution of the intensities in the subtraction image. The most straightforward way to resemble the probability density function is to create a normalized histogram $H(i)$ in which each entry is divided by the total number of entries ($\sum_{i=-I}^{I} H(i) = 1$).

A generalized form of the Shannon entropy is the weighted sum of the normalized histogram entries

$$M(\mathbf{d}) = \sum_{i=-I}^{I} f(H(i)), \tag{2}$$

where $f : \mathbb{R}_+ \to \mathbb{R}$ is a weighting function. Buzug and Weese [1] have proven that any differentiable, strictly convex or strictly concave function can be a suitable weighting function such that $M(\mathbf{d})$ satisfies the above requirements. Further, they suggested that the energy of the histogram

$$M_{\mathrm{EHD}}(\mathbf{d}) = \sum_{i=-I}^{I} H^2(i) \tag{3}$$

is the most appropriate measure due to its relatively lower computational cost while the computation accuracy is retained. *Note that M_{EHD} is to be maximized, whereas, M_{ENT} is to be minimized.*

2.2. Search strategy

The similarity measure $M(\mathbf{d})$ in Eqs. (1)–(3) is a function of the displacement $\mathbf{d}$. The optimal alignment is found by searching for the extremal $M(\mathbf{d})$. The robustness of the *full search* is self-evident, because all the $(2w + 1)^2$ positions within the search area are examined, w denoting the number of pixels in both search directions (Fig. 2(a)). It is however not a feasible approach due to high computational cost. A much more efficient strategy is the *conjugate direction search* which is carried out successively in n linearly independent conjugate directions, n being the number of variables in the objective function [5]. Fig. 2(b) shows how the search is carried out in case $n = 2$.

The complexity of conjugate direction search with a step size of one pixel can be easily derived: in the best case where the displacement is minimal, $\mathbf{d} = (0, 0)$, only five positions (the starting position itself and its four neighbors) must be examined. In the worst case where the displacement is maximal, $\mathbf{d} = (w, w)$, $(2w + 3)$ positions must be examined. Thus, the complexity is reduced from $\mathcal{O}(x^2)$ by full search to $\mathcal{O}(x)$ and $\mathcal{O}(1)$ by conjugate direction search.

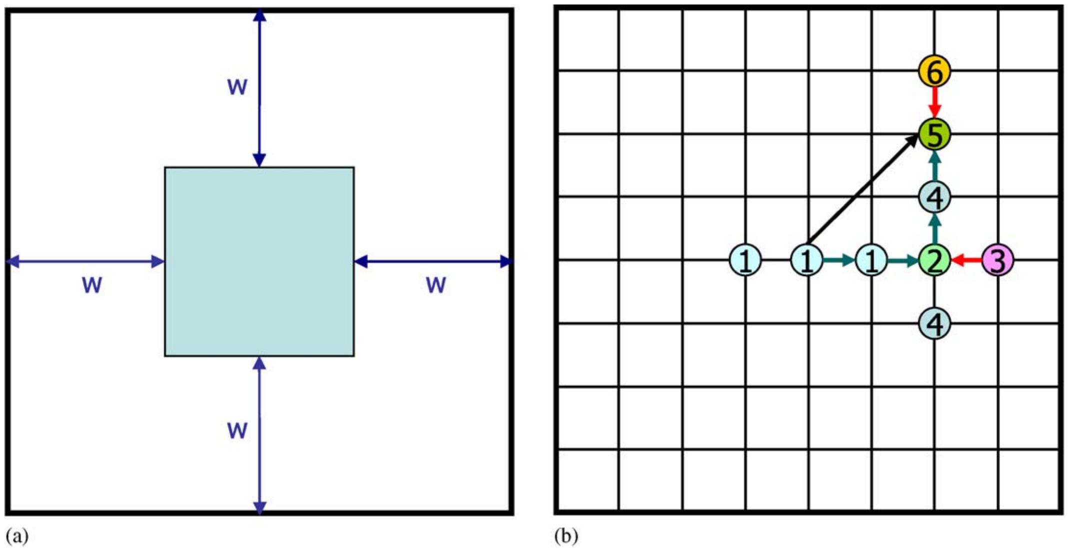

Fig. 2. (a) Full search: w denotes the search range. (b) Conjugate direction search: search starts along the x-axis. The starting point is compared with its two horizontal neighbors. Then go to the direction of the point which has the best match. Search continues along the x-axis until no improvement of the match score can be achieved. Then repeat the search along the y-axis. The optimum found along the y-axis is the global optimum.

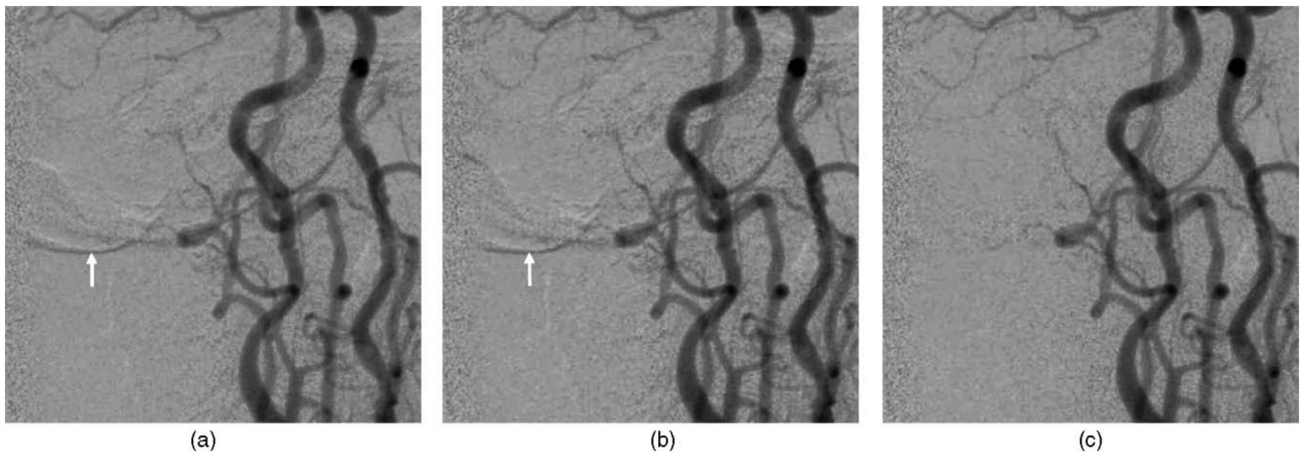

Fig. 3. Sub-pixel precision: (a) original subtraction image, showing artifacts (highlighted with arrows); (b) block matching with integer precision, most artifacts remain unchanged; (c) block matching with sub-pixel precision, artifacts are removed.

2.3. Sub-pixel accuracy

Even sub-pixel misalignment can cause significant artifacts in subtraction images. Therefore, the motion artifacts should be corrected with sub-pixel precision. An example is shown in Fig. 3. As digital images contain only information about the pixels locating at integer positions, interpolation is necessary. There exist mainly two approaches to perform the interpolation: (1) interpolating the similarity values calculated at integer displacements, and (2) computing the similarity values directly while shifting an image for non-integer displacements. The first approach requires the construction and analytical solution of a continuous bivariate function, whereas, the latter approach can be achieved by bilinear interpolation which is a built-in operation on the GPU and can be carried out in real time. Clearly, the latter approach is preferred in our case.

It has been reported that a sub-pixel precision of 1/10 pixel is sufficient for angiography images [6,2]. For the sake of efficiency, the search is usually carried out *hierarchically*, i.e., at first with integer precision, then with sub-pixel precision only in the neighborhood where the optimal integer displacement was found. The complexity can be derived likewise as with integer precision: nine positions in the best case (the starting point itself, its four neighbors with integer precision and four neighbors with sub-pixel precision) and $2(w_{\mathrm{int}} + w_{\mathrm{sub}} + 2) + 1$ positions in worst case, w_{int} and w_{sub} being the search range for integer and sub-pixel precision respectively.

3. Hardware implementation

3.1. Programmable graphics hardware

The data parallelism in typical image processing algorithms makes them well suited for the parallel pipelined architecture of the GPUs. An efficient mapping of such algorithms onto graphics hardware can thus lead to considerable performance gains [7–11].

A crucial aspect concerning image processing on the graphics hardware is the texture memory and its associated access operations. Textures are bitmaps of pixel colors, which are designed to give 3D objects realistic visual complexity with little geometric data. For the purpose of image processing the input images are usually loaded as textures onto the graphics card and the results, the manipulated images, are written into textures as well.

Originally, textures were only bound to the pixel shader stage and random reading access to the texture memory (*gathering*) is allowed, which encourages the use of textures as a general read-only memory, with texture coordinates as the memory address. In fact, textures are widely used as general lookup tables to store information other than pixel colors.

As gathering in the pixel shader is rather fast, the random writing access to the texture memory (*scattering*) was not supported, because the pixel positions are determined by the hard-wired rasterizer and can not be changed in the later stages on the pipeline any more. The realization of scattering in the pixel shader would change the pipeline architecture substantially. This inflexibility had limited GPUs to be used more generally.

Fortunately, a new feature has been recently added to the graphics processor, *vertex texturing*, which enables texture fetch in the vertex shader stage and therewith allows the scattering. This feature is incorporated in Shader Model 3.0 and supported on GeForce 6 Series. It is possible now to implement algorithms where gathering and scattering are both required, e.g., sorting and histogram computation. This is also the key feature we utilize to implement the histogram-based similarity measures.

3.2. Implementation

Block matching is an algorithm which fits the architecture of parallel processors very well: the blocks are processed with the same operations and independently from each other, potential performance gains can thus be achieved by parallelizing the processing. As Kelly and Kokaram [12] have shown, it is more efficient to shift all the blocks within the entire frame simultaneously and calculate the correspondence on a block basis. They mapped a full search block matching on the graphics hardware by passing the displacement vector as a global parameter to the GPU.

However, full search is computationally very intensive and usually not feasible for interactive application. In order to take advantage of optimizing search strategies, each block must be moved with an individual displacement vector while all the blocks are moved simultaneously. Our solution is to store the individual displacement vectors in a texture which is used as a lookup table. In order to avoid data transfer between CPU and GPU, this texture is stored on the GPU.

To further enhance the efficiency, we determine the displacements (explicitly by means of block matching) only on selected control points defined by a regular grid. As shown in [2], a block of 51×51 pixels yields a good compromise between computational cost and statistical reliability. In order to take all the pixels into account, we define the blocks size as 64×64 pixels. The DSA images have usually the dimension of 512×512 or 1024×1024, which ensures an exact decomposition of the input image into blocks.

As illustrated in Section 2, the displacement of a control point is found by moving the respective block rigidly within the search space, i.e., all the pixels within the block are assumed to have the same displacement. When this displacement is then mapped uniformly to all the pixels within that block to construct the final displacement vector field, artifacts at the edges between the blocks are likely introduced. To avoid this problem, the displacements of the remaining points (non-control points) are obtained by bilinear interpolation. A problem hereby is, while the blocks are an exact partition of the image, the resulting displacement vector field does not cover the entire image, because the control points locate within the blocks. This problem is solved by extending the current grid by one cell in both x- and y-directions and setting the displacement of the points on the borders to zero; the points between are then interpolated.

The whole sequence is processed by fixing the mask image and registering the contrast image one after another with respect to the mask image. Essentially, the motion compensation for an image pair is carried out with the following steps:

1. Initialize a set of displacement vectors and similarity scores $R = \{(\mathbf{d}_i, s_i)\}$, $i = 1, 2, \ldots, N$, N being the number of blocks.
2. Find the optimal displacement vectors following conjugate direction search.
   a. Calculate the difference image from the mask and contrast images, whereby the i-th block (of the mask image) is shifted with the corresponding displacement vector $\mathbf{d}_i$.
   b. Maximize the similarity score $s_i = M_{\mathrm{ENT}}(\mathbf{d}_i)$ (Eq. (1)).
3. Construct the final displacement vector field $F(\mathbf{d}_k)$, $k = 1, 2, \ldots, M$, M being the number of pixels. Obtain the motion compensated subtraction image by warping the mask image with F.

3.2.1. Histogram of differences

A histogram of an image or a portion of the image is usually implemented as an array, with the pixel intensities as the array indices. The input image is read pixel by pixel, and the corresponding array element is incremented. Thereby both random reading access and random writing access to the memory are necessary within one shader program. As illustrated in Section 3.1, this was not realized until the appearance of vertex texturing. Therefore, filling histogram was one of the toughest tasks of the general-purpose computation using graphics hardware.

Our solution is illustrated as following: by projecting a 3D scene onto a 2D plane, more vertices may be mapped to the same position, with both color and depth information contained in a fragment. The final color of a pixel on the screen is the mixture of all the fragments which are supposed to display at the same position (*alpha-blending*). If all the input fragments have the same intensity and the alpha-blending function is set as additive, the frequency of the fragments can be retrieved from the final intensity of the output pixel.

To this end, the difference image $\mathbf{P}$ is bound as a texture to the vertex shader in order to scatter the pixels in $\mathbf{P}$ by their intensities. At first, a set $\mathbf{V}$ of vertices is firstly defined, with the cardinality of $\mathbf{V}$ being equal to the number of pixels in $\mathbf{P}$. Then, we fetch for each vertex $\mathbf{v}$ in $\mathbf{V}$ the corresponding pixel[1] $\mathbf{p}$ from $\mathbf{P}$ and change the position of $\mathbf{v}$ according to the intensity of $\mathbf{p}$. In particular, the intensity value and the block index of $\mathbf{p}$ are combined to build the x- and y-coordinates of $\mathbf{v}$. As a result, the pixels in $\mathbf{P}$ are scattered not only with their intensity values, but also with their block indices. Herewith the histograms for all the blocks are filled in parallel within one rendering pass.

3.2.2. Similarity function

As illustrated in Section 2, the histogram based similarity function is a weighted sum over the histogram entries (Eqs. (1)–(3)), which are stored in a texture. To sum up the values in a texture, the so called *sum-reduce-operation* is applied. Sum-reduction is similar as progressive down-sampling of the texture over multiple passes, where in each pass a local sum of $n \times n$ pixels of the source texture is written into one pixel of the target texture. The image is "reduced" by a factor of n along each axis with each pass. For a image of $N \times N$ pixels, $\log_n N$ passes are required. The global sum is finally written into a single pixel. As reading from and writing into the same texture is not allowed within one rendering pass, at lease two textures are necessary. After each pass these two textures are alternated (ping-pong buffering).

3.2.3. Precision of the computation

As graphics hardware of earlier generation supports only fixed-point arithmetic, the new superscalar architecture of GeForce 6 Series provides full 32-bit floating point accuracy, which enables computation with high precision in a much broader range. The texture formats used in the implementation are listed in Table 1. The input images are given as 16-bit `unsigned short`, which are loaded into a 16-bit fixed-point format directly. To store the difference image, a 32-bit floating-point format is used, because this texture will be bound to the

[1] More accurately, an element of a texture is referred to as a texel.

Table 1
Texture formats used in the implementation

Usage	Size	Precision	Dimension
Input image	1024 × 1024	16-bit fixed-point	1
Difference image	1024 × 1024	32-bit floating-point	1
Histogram	1024 × 256	16-bit floating-point	4
Displacement vector	16 × 16	32-bit floating-point	4

vertex shader. Vertex textures are required at a minimum to support 32-bit floats, in order to handle large world and view spaces. The histogram is stored in a 16-bit floating-point format (`half`) which is the only available floating format supporting alpha-blending. The texture storing the displacement vectors and similarity scores is of 32-bit floating point format, so that the highest precision for the evaluation is guaranteed.

The 32-bit floating-point formats on GPUs follow the IEEE standard which is known as m23e8 format. This precision is usually sufficient for processing medical images which use 10–14 bits for a pixel. The 16-bit `half` format on GPUs has 10 bits for mantissa and 5 bits for exponent (m10e5). The number of mantissa bits dictates the precision. With 10 mantissa bits and 1 hidden bit, the `half`-format can only precisely represent 2^{11} equidistant numbers, e.g., integers. The blocks in our implementation have the size of 64 × 64 pixels. In extreme cases that all the pixels within a block have the same intensity value, the histogram entry of this value is 2^{12} which can not be represented exactly with the `half`-format. However, this could only occur at the image corners where no structural information is available, hence the overall precision is not affected. By creating histogram for a larger block, more consideration concerning the precision is necessary.

The pixel depth of our input images is 12 bits, i.e., theoretically there exist 2^{13} different intensity values after the subtraction. This necessitates 8192 bins to hold the possible histogram entries (by a bin size of 1). Luckily, the differences between the mask and the contrast images are mostly in a very narrow range around zero. In fact, we found out that 1024 bins are already sufficient to compute the histogram in our case. The texture for the histogram is thus of size 1024 × 256, storing histograms for 256 blocks with 1024 bins for each histogram.

4. Experimental results

We tested our implementation on two graphics cards of the Nvidia GeForce 6800 Series: 6800 and 6800 GT. The main features of these two cards are listed in Table 2.

We applied our implementation on four data sets: shoulder, head, abdomen and hand. All data sets were acquired with an angiography unit (AXIOM-Artis, Siemens Medical Solutions). Each data set consists of a sequence of 1024 × 1024 images with the first image as the mask image and the remainder as contrast images. The pixel depth is 12 bits while 16 bits are allocated. The block size is 64 × 64 pixels, accordingly an image is partitioned into 16 × 16 blocks. The displacement is computed with both integer (search step = 1 pixel) and sub-pixel (search step = 1/10 pixel) precision, where the search range is ±10 and ±0.5 pixels, respectively.

Figs. 4–7 show the results of applying our implementations on the four given data sets. Each of these figures consists of four images, labeled with (a)–(d) and arranged as following:

(a) original subtraction image,
(b) block matching with integer precision (implemented on GPU),
(c) block matching with sub-pixel precision (implemented on GPU),
(d) flexible pixel shift (implemented on CPU).

The number of images in each sequence, the corresponding computation time required to process the whole sequence are listed in Table 3. The computation time includes reading the image sequence from the main memory and transferring it to the GPU.

Table 2
NVIDIA GeForce 6800 product lineup specifications

Key Features	GeForce 6800 GT	GeForce 6800
No. of pixel processors	16	12
No. of vertex processors	6	5
Memory type/amount	GDDR3/256 MB	GDDR/128 MB
GPU speed	350 MHz	325 MHz
RAM speed	1000 MHz	700 MHz

Table 3
The image sequences used in the experiment and the corresponding computation time to process the sequences

Type	Number of images (*N*)	Computation time (s)			
		Integer		Sub-pixel	
		6800 GT	6800	6800 GT	6800
Head	21	7.20	8.86	10.23	13.26
Hand	20	6.89	8.76	8.98	11.10
Shoulder	18	6.28	9.17	8.74	11.95
Abdomen	39	12.74	15.12	17.79	22.06

Note that in each sequence the first image is the mask image and the remainder are the contrast images, thus $(N - 1)$ image pairs are processed, *N* being the number of images in each sequence.

5. Clinical evaluation

To evaluate the effectiveness of our implementations, two radiologists were asked to rating the images in Figs. 4–7. To avoid any bias in the ratings, the images were presented to the radiologists in a completely blinded manner: the type of the correction was kept unknown to the observers and the evaluation was carried out independently by the two radiologists. The ratings are listed in Table 4.

A reliable statistical analysis was not carried out due to the limited sample size. However, the total score in Table 4 indicates that by given a larger sample, the block matching with both integer and sub-pixel precision could improve the image quality significantly and their effects are comparable with those of flexible pixel shift. With sub-pixel precision we have shown that our approach can correct weak artifacts accurately (Fig. 3).

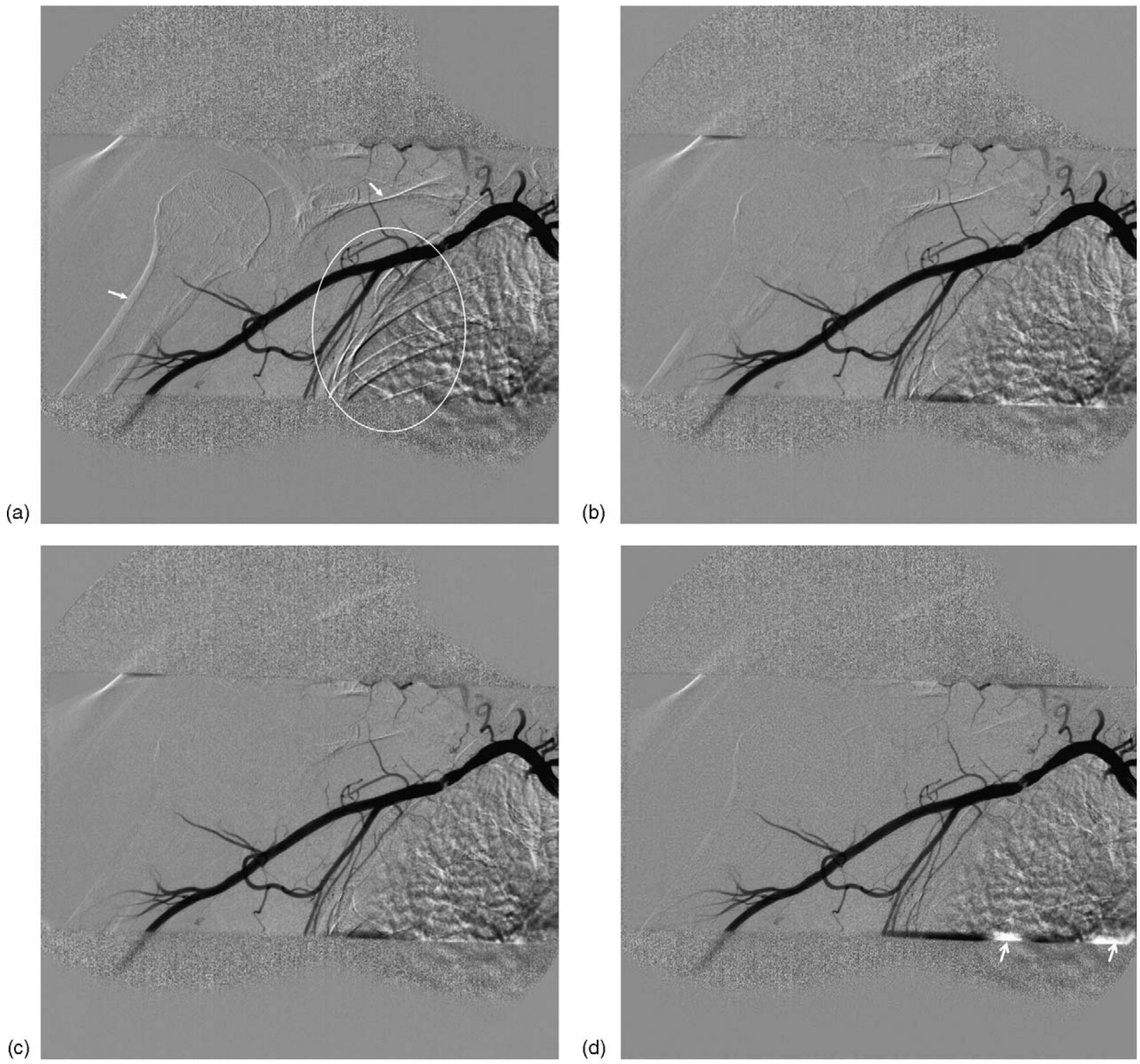

Fig. 4. Digital subtraction angiogram of the right upper arm with injection in the subclavian artery. The original subtraction image (a) contains strong artifacts (highlighted with solid arrows and ellipse). These are reduced considerably by block matching with integer precision (b) and further with sub-pixel precision (c). Flexible pixel shift (d) yields comparable results. The arrows in (d) show the artifacts at the boundary between the actual exposure area and the homogeneous area.

On the other hand, these artifacts do not impact the diagnostic value of the subtraction images that much. Sometimes, radiologists prefer weak artifacts so that they can be used as landmarks. It is worth noting that processing with only integer precision is much faster (3 frames/s) than that with sub-pixel precision (2 frames/s), which encourages its wide application as an interactive tool in the clinical routine.

6. Discussion

6.1. Motion compensation accuracy

The relatively stronger artifacts (Figs. 4 and 5) caused mainly by the displacements of bones are removed considerably by applying all the implementations, whereas, the relatively weaker

Table 4
Result of clinical evaluation

	Shoulder (Fig. 4)	Head (Fig. 5)	Abdomen (Fig. 6)	Hand (Fig. 7)	Total
Original subtraction	(4, 3)	(4, 4)	(3, 2)	(4, 3)	27
CPU	(1, 1)	(1, 1)	(4, 4)	(1, 2)	15
GPU integer	(3, 2)	(3, 3)	(2, 1)	(3, 1)	18
GPU subpixel	(2, 4)	(2, 2)	(1, 3)	(2, 4)	20

The ratings of two independent radiologists are presented as (r, s), where r and s stand for the first and the second radiologist, respectively. The images are rated with scores 1–4, with 1 being the best and 4 being the worst image quality.

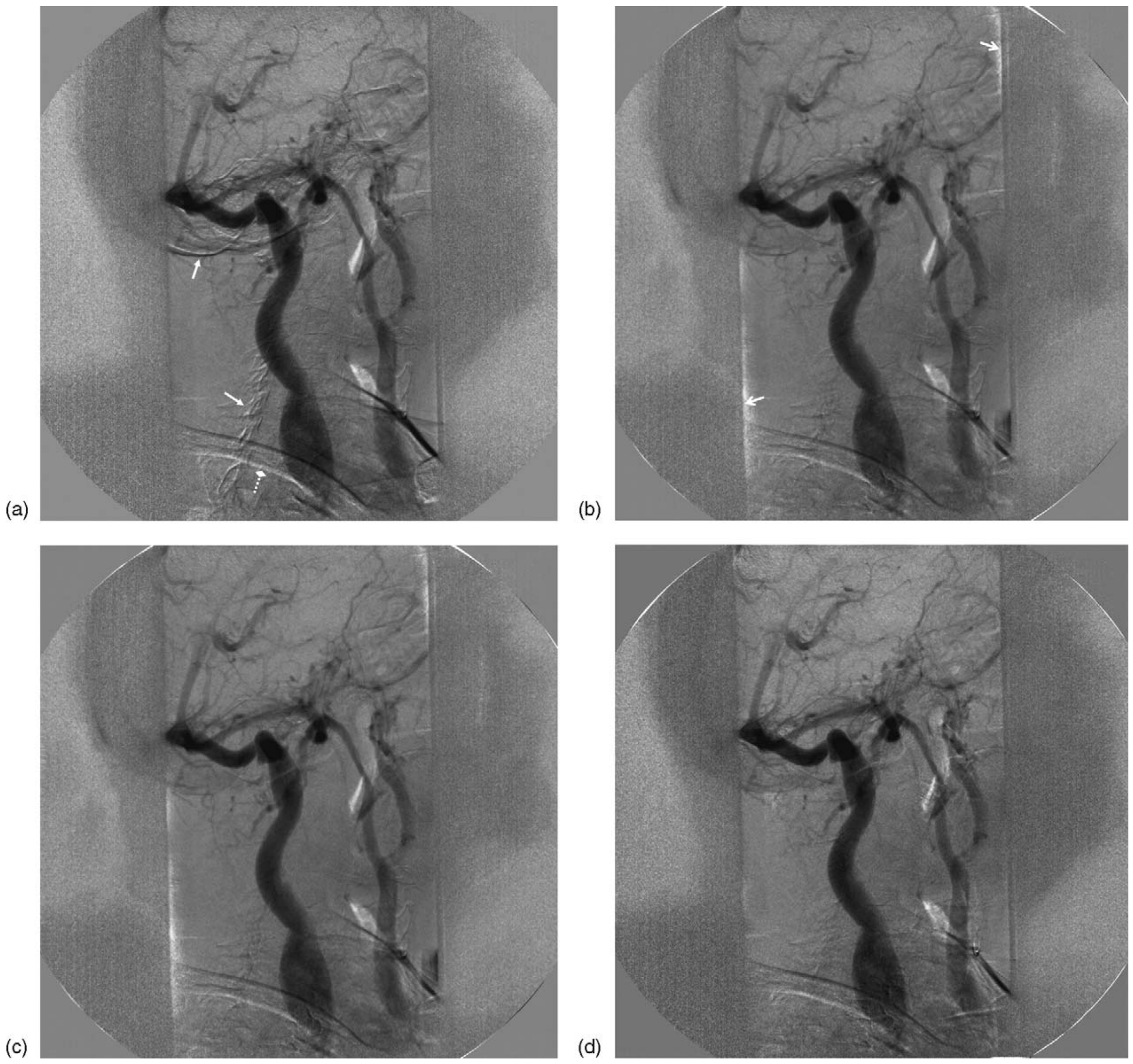

Fig. 5. Cerebral DSA, venous phase. The original subtraction image (a) contains strong artifacts caused by the displacements of independent moving structures: the solid arrows point to the artifacts caused by movement of the skull and the cervical spine, the stippled arrow point to the artifacts caused by movement of the clavicula. Block matching with both integer (b) and sub-pixel precision (c) removes the artifacts caused by the skull and the spine considerably, but those caused by the clavicula are still present. Flexible pixel shift (d) yields similar results. The arrows in (b) show the artifacts at the boundary between the actual exposure area and the homogeneous area.

artifacts (Fig. 3) remain by block matching with integer precision. In Figs. 5 and 6, the artifacts are only partly removed, with both block matching and flexible pixel shift. This is because the artifacts are caused by the displacement of structures which are overlapped to each other, e.g., the clavicula and the cervical spine in Fig. 5 and the intestines and the lumbar spine in Fig. 6. The correction of the displacement of one structure could possibly introduce misalignment of other structures, which was reported in [4,2] as well. This is also the major limitation of a 2D motion compensation algorithm for projection images.

The images in the data sets head, hand and shoulder are acquired with using blinds[2] that are attached directly in front of the X-ray tube to reduce the radiation dose. Accordingly, the contrast at these areas is very low in both the input and the subtraction images. The displacements found in these flat areas are often rather large, because there is not enough structural information to yield an unique match. This is a well-known problem of block matching and was observed by Kelly and Kokaram [12] as well. Consequently, artifacts can be observed at the border between the actual exposure area and the areas where the blinds are placed. These artifacts are also observed in the results of the flexible pixel shift (Fig. 4). Our suggestion to solve this problem is to incorporate an user defined region of interest, such that the homogeneous regions, where no vessels of interests are present, can be excluded. Note that these regions make up to 50% area of the entire image, which indicates further reduction of the actual necessary computation time.

[2] These are referred as a virtual collimator by some vendors.

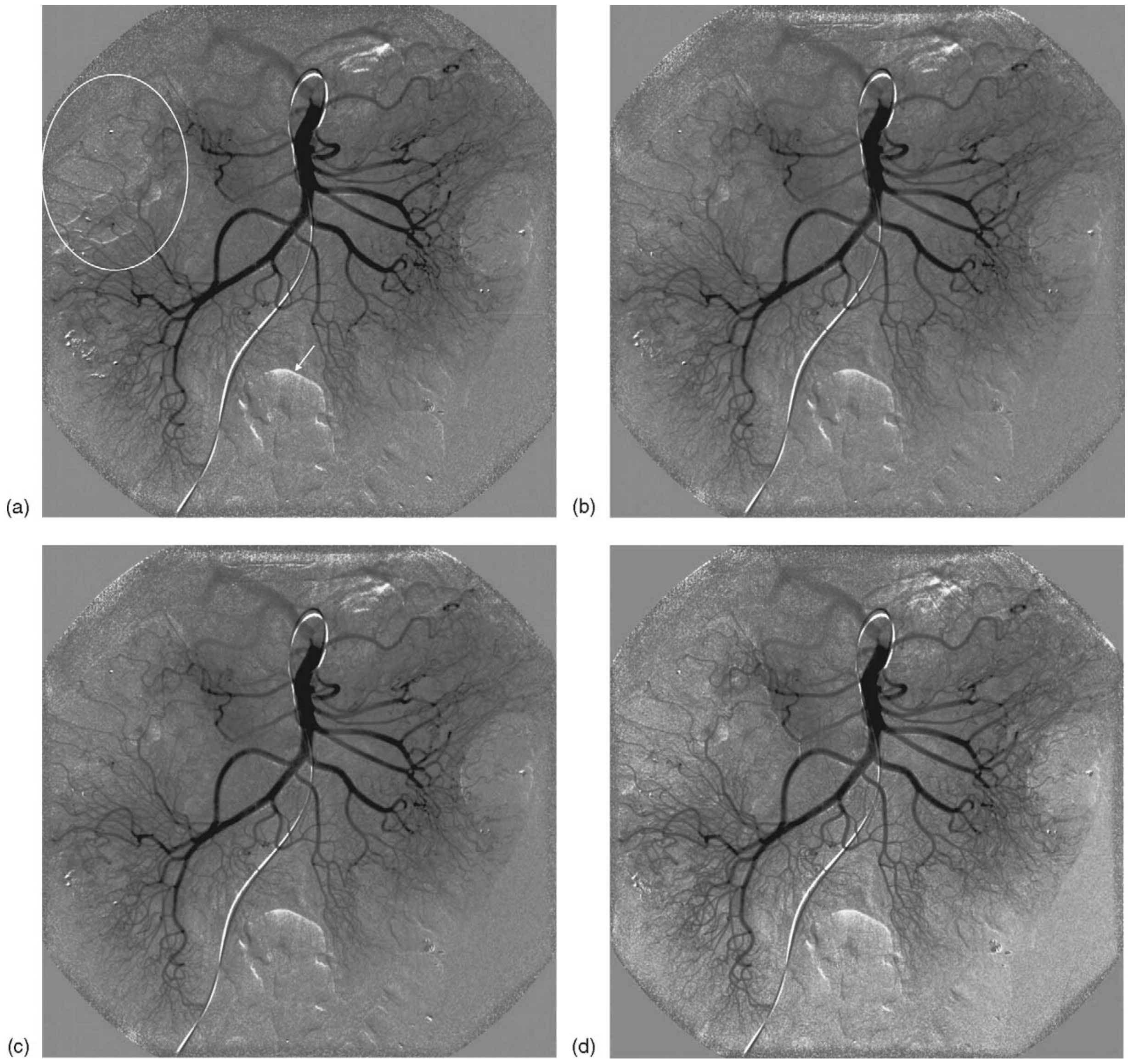

Fig. 6. Abdominal DSA with injection in superior mesenteric artery. The original subtraction image (a) contains moderate artifacts (highlighted with the ellipse). These are reduced partly by block matching with both integer (b) and sub-pixel precision (c). Flexible pixel shift (d) removes less artifacts than GPU-registration.

6.2. *Performance analysis*

As flexible pixel shift needs 6–8 s to process two 1024 × 1024 images, our GPU-implementation of block matching performs one magnitude faster. In the latter case, a speedup of approximately 15–30% can be observed between the results on 6800 GT and on 6800 (Table 3). In the following, we will only discuss the performance analysis concerning the GPU-implementation and refer to the results achieved on 6800 GT.

In average it takes us 0.47–0.51 s to process two images (Table 3). In order to determine the performance bottleneck, we first measured the time required to load an input image (1024 × 1024 × 16 bits) without any processing, which takes approximately 10 ms. A sharp dropdown of the frame rate can be observed after filling the histograms of the difference image, which implies a bottleneck at this stage. As illustrated in Sections 3.1 and 3.2.1, both gathering and scattering are required to fill a histogram and this is only possible in the vertex shader. On the other hand, in the previous step the difference image was generated in the pixel shader and then resides in the frame buffer. Before the appearance of vertex texturing, reading data from the frame buffer back to the vertex shader was not possible without going through the main memory, which could block the hardware and lead to serious performance degradation.

By allowing bidirectional data transfer between vertex shader and pixel shader, vertex texturing enables almost all kinds of general-purposed computations on the GPU and may change GPU-programming forever. However, texture fetch in the vertex shader still generates latency and is much slower than that in the pixel shader [13]. In our implementation, it takes approximately 35 ms to fill the histogram for a 1024 × 1024 image. No significant performance change could be observed whether only one histogram is filled for the entire image or all the histograms are filled on a block basis in parallel. We measured the

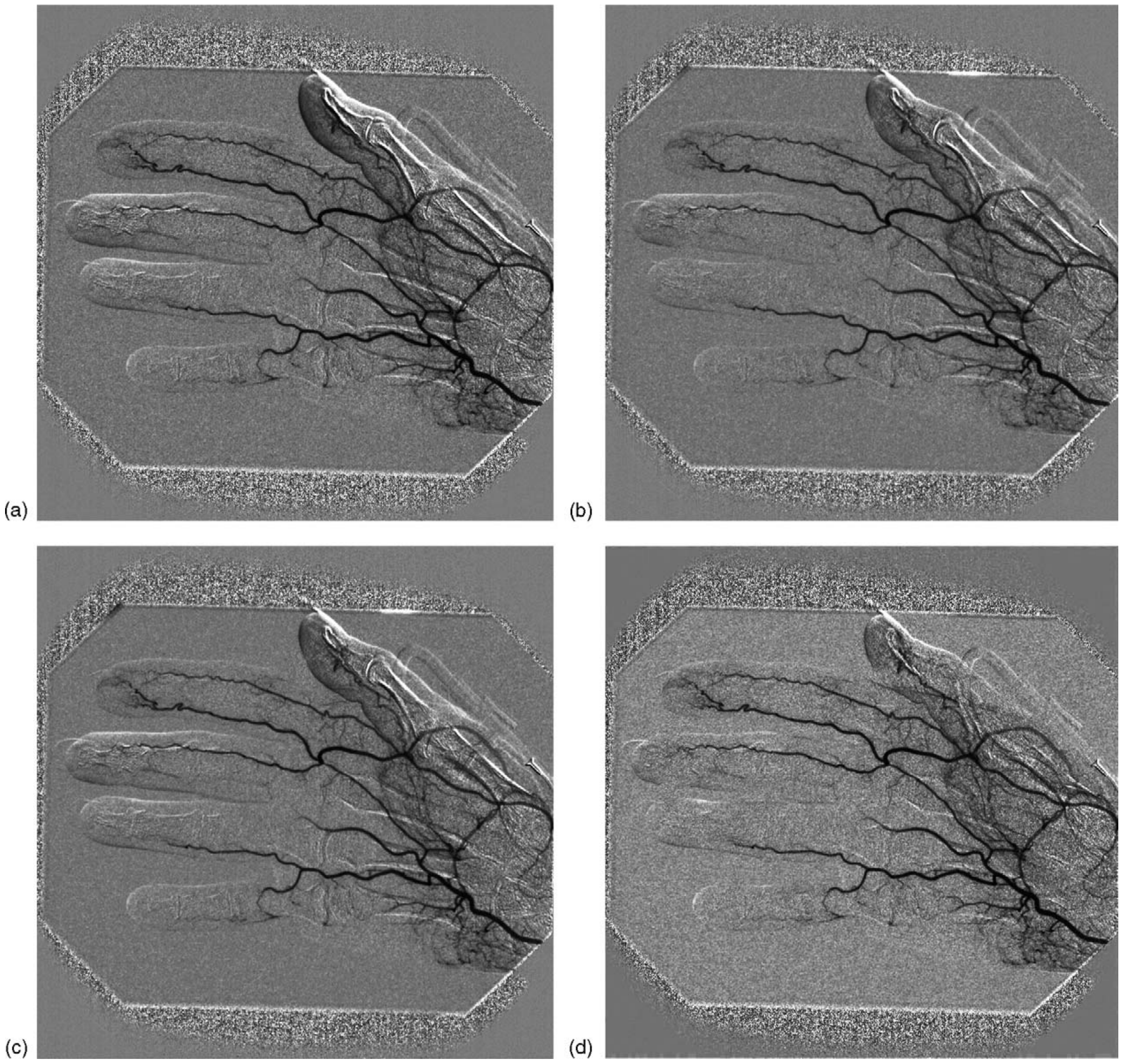

Fig. 7. DSA of the hand. The original subtraction image (a) contains strong artifacts. These are reduced only partly by both block matching (b and c) and flexible pixel shift (d). The difference between integer (b) and sub-pixel precision (c) is not evident.

time required to render the difference image using vertex texture, which turns out to be nearly the same as that required to fill the histograms. Compared to pixel texturing (∼11 ms at the comparable stage), vertex texturing (∼35 ms) is no doubt much slower.

But the latency generated by vertex texturing is not the only factor responsible for the bottleneck. The large amount of vertices (1024 × 1024) used to fetch the pixels is an important issue as well. To render a scene, usually only a small number of vertices are defined (e.g., the corner points of the triangles). The value of the pixels, for which no vertices are explicitly defined (e.g., the points within the triangles), is obtained by means of interpolation at the rasterizer. Therefore, the number of vertices is much smaller than that of pixels. Accordingly much fewer parallel vertex engines than pixel engines are available on a GPU. For instance, there are 12–16 pixel pipelines and only 5–6 vertex pipelines on a GeForce 6800 GPU (Table 2). We observed that processing 1024 × 1024 vertices without vertex texturing (i.e., only the conventional vertex computation such as geometrical transformation) is achieved at ∼75 frames/s, which is similar to processing 512 × 512 vertices with vertex texturing.

7. Summary

In summary, both flexible pixel shift and block matching can correct motion artifacts in DSA effectively. The best results are achieved in those cases where the moving structures are not superimposed in the original 3D scene. In other cases, the motion artifacts can be partly reduced.

The block matching implemented on the GPU turns out to be one magnitude faster than the CPU-version of flexible pixel shift. The processing of two 1024 × 1024 images with sub-pixel precision is achieved at 2 frames/s, including loading the images from the main memory and transferring them to the graphics hardware. From the point of view of the medical applicants, the registration with integer precision is sufficient in most cases. This can be carried out even faster (3 frames/s) and allows it to be used as an interactive tool in medical image analysis.

Acknowledgments

We thank Christian Eisenacher and Prof. Marc Stamminger at the chair of Computer Graphics, Friedrich-Alexander University, for helpful discussions regarding GPU implementation.

References

[1] Buzug TM, Weese J. Image registration for DSA quality enhancement. Comput Med Imag Graph 1998;22(2):103–13.

[2] Meijering EH, Zuiderveld KJ, Viergever MA. Image registration for digital subtraction angiography. Int J Comput Vis 1999;31(2/3):227–46.

[3] Taleb N, Bentoutou Y, Deforges O, Taleb M. A 3D space-time motion evaluation for image registration in digital subtraction angiography. Comput Med Imag Graph 2001;25(3):223–33.

[4] Bentoutou Y, Taleb N, Chikr El Mezouar M, Taleb M, Jetto L. An invariant approach for image registration in digital subtraction angiography. Pattern Recogn 2002;35(12):2853–65.

[5] Srinivasan R, Rao K. Predictive coding based on efficient motion estimation. IEEE Trans Commun 1985;33(8):888–96.

[6] Van Tran L, Sklansky J. Flexible mask subtraction for digital angiography. IEEE Trans Med Imag 1992;11(3):407–15.

[7] Cates JE, Lefohn AE, Whitaker RT. GIST: an interactive, GPU-based level set segmentation tool for 3D medical images. Med Image Anal 2004;8(3):217–31.

[8] Rumpf M, Strzodka R. Nonlinear diffusion in graphics hardware. In: Proceedings of EG/IEEE TCVG symposium on visualization (VisSym '01). 2001. p. 75–84.

[9] Rumpf M, Strzodka R. Level set segmentation in graphics hardware. In: Proceedings of the IEEE international conference on image processing (ICIP '01). 2001. p. 1103–06.

[10] Sherbondy A, Houston M, Napel S. Fast volume segmentation with simultaneous visualization using programmable graphics hardware. In: Proceedings of the IEEE visualization. 2003. p. 171–76.

[11] Strzodka R, Droske M, Rumpf M. Fast image registration in DX9 graphics hardware. J Med Inform Technol 2003;6:43–9.

[12] Kelly F, Kokaram A. Fast image interpolation for motion estimation using graphics hardware. In: Proceedings of the IS&T/SPIE electronic imaging–real-time imaging VIII, SPIE 5297. 2004. p. 184–94.

[13] NVIDIA GPU programming guide. 2004. http://developer.nvidia.com/object/gpu_programming_guide.html.

Yu Deuerling-Zheng has studied medicine (1989–1996) at the Tongji Medical University and worked (1996–1997) at the Radiological Department of Xiehe Hospital in Wuhan, China. She received her diploma in computer science (2000–2005) at the Friedrich-Alexander University (Erlangen, Germany), where her focus was on medical image processing and computer graphics. She has recently joined Siemens Medical Solutions in Forchheim, Germany, and works on innovative topics in the field of medical engineering.

Michael Lell has studied medicine from 1989 to 1991 at the University of Regensburg and from 1991 to 1996 at the Technical University Munich. He has specialized in radiology at the University of Erlangen and is assistant professor of radiology since 2005. In 2006 he started research at the Department of Radiological Sciences at University of California, Los Angeles. His current focus of research is cardiovascular imaging and image post-processing. He is member of German Roentgen Society (DRG), European Society of Radiology (ECR) and German Society of Radiology and Nuclear Medicine.

Adam Galant holds the Masters degree from AGH University of Science and Technology in Krakow, Poland. Residing in the US since 1981 worked as the consulting engineer specializing in designs of electronic systems for image processing and 3D computer simulation. Presently he is working as the researcher for Siemens Medical Solutions. Member of AAAS, IEEE Computer Society and ACM SigGraph and SigDA.

Joachim Hornegger graduated in Theoretical Computer Science/Mathematics (1992) and received his PhD degree in Applied Computer Science (1996) at the University of Erlangen-Nuremberg (Germany). His PhD thesis was on statistical learning, recognition and pose estimation of 3D objects. Joachim was an assistant professor at the University of Erlangen-Nuremberg (1996/97), a visiting scientist at the Technion (Haifa, Israel), a visiting scholar at the Massachusetts Institute of Technology (Cambridge, MA, USA), and a visiting scholar and lecturer at Stanford University (Stanford, CA, USA) in the academic year 1997/98. In 1998 he joined Siemens Medical Solutions Inc. where he was working on 3D angiography. In parallel to his responsibilities in industry he was a lecturer at the Universities of Erlangen (1998–1999), Eichstaett-Ingolstadt (2000), and Mannheim (2000–2003). In October 2003 Joachim became Professor of Medical Imaging Processing at the University of Erlangen-Nuremberg and since October 2005 he is a chaired professor heading the Institute of Pattern Recognition. His main research topics are currently pattern recognition methods in medicine and sports. He is a member of IEEE Computer Society and GI.