Accounting for Uncertainty in Medical Data: A CUDA Implementation of Normalized Convolution

S. Lindholm and J. Kronander

Department of Science and Technology, Linköping University

Abstract

The domain of medical imaging is naturally moving towards methods that can represent, and account for, local uncertainties in the image data. Even so, fast and efficient solutions that take uncertainty into account are not readily available even for common problems such as gradient estimation. In this work we present a CUDA implementation of Normalized Convolution, an uncertainty-aware image processing technique, well established in the signal processing domain. Our results show that up to 100X speedups are possible, which enables full resolution CT images to be processed at interactive processing speeds, fulfilling demands of both efficiency and interactivity that exist in the medical domain.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

An uncertainty-aware visualization pipeline has previously been acknowledged as a necessary development to provide reliable visualization tools in critical application areas such as medical diagnosis [Joh04, KS08]. Nevertheless, uncertainty-aware visualizations are still the exception rather than the norm. In visualization, uncertainty arise and propagate in each step of the pipeline, having significant impact on the final image [LLPY07]. A particularly pertinent sub-domain is low-dose medical imaging. Serious concerns about the dose levels in current clinical practice has recently been raised [BA10] and methods to retain diagnostic image quality at increasingly lower dose levels are highly requested. The work in this paper comprises one step towards the development of a fully uncertainty-aware pipeline.

Image processing frameworks for dealing with uncertainty have previously been presented in the Signal Processing community [Kay01], and the idea of separating values of the signal from the certainty of the measurements has a long history [Gra78, GK83, Knu89]. However, for achieving widespread medical use, algorithms must not only be robust and accurate but also fast. User evaluations [LP11] have shown that efficiency is one of the most challenging aspects of the medical work flow. In this work we focus on the Normalized Convolution (NC) [KW93a] framework for which we present a CUDA based implementation. Previous approaches using NC have been limited in practice by their exceedingly slow runtime speeds, an effect of the local adaptivity of the filters. In contrast, our parallelized CUDA implementation yields speedups in the order of 100X. To highlight the usefulness of our approach, we have chosen to exemplify the impact of uncertainty in medical image processing by focusing on estimating gradients from noisy data. Local image gradients are used in a multiple of common mid-level processing tasks such as edge-detection [Can86], shading [HKRs*06] and transfer function design [KKH02], and are therefore of high importance in any visualization pipeline. The main contributions of this work are

1. Based on the existing theory of Normalized Convolution, we present a framework for fast processing of uncertain medical image data, including discussions on hardware optimizations and numerical considerations.
2. Using a highly optimized CUDA implementation of Normalized Averaging and Normalized Convolution we achieve a 100X speedup compared to previous work.

It should be noted that although this paper focuses on

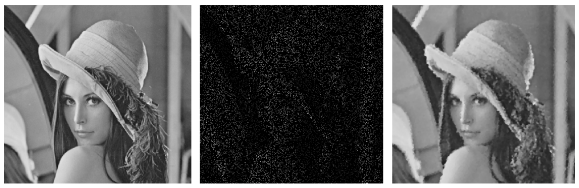

Figure 1: *Demonstration of the power of normalized averaging for restoring an image from only 10% of the original pixel values.* Left: *The original image.* Middle: *Image with 90% of the pixels removed.* Right: *The result of applying normalized averaging to the lossy image (a special case of Normalized Convolution with a single constant basis function).*

Figure 2: *Demonstration of the limitations when adapting normalized averaging for gradient estimation of lossy images (30%).* Left: *Original image.* Middle: *Lossy image.* Right: *Gradient estimation using filter weight normalization. It is evident that normalization of filter weights, which works well for smoothing filters, fails to produce sufficient results for gradient estimation of the lossy image.*

gradient estimation to highlight the effects of uncertainty, the method is independent with respect to the chosen basis/estiamtion desired. Other common convolution processes, such as curvature estimations, bilateral or anisotropic filtering would be affected in similar manner.

2. A Motivational Example

Consider a constant image [1 1 1] on which we apply a box filter [1 1 1]. The expected response is $(1+1+1)/3 = 1$ where the division by 3 is the natural filter normalization. Now consider the case where we have a dropped sample such that the same filter is now applied to the image [* 1 1] with a certainty of [0 1 1]. There are two naive ways to do this. Option one is to discard the uncertainty information (effectively interpreting * as 0) and filter the image as $(0+1+1)/3 = 0.67$. Option two is to discard the uncertain sample and calculate a new filter normalization $(1+1)/2 = 1$. This is called normalized averaging, and works well for the smoothing filters as demonstrated in Figure 1 where a decent image reconstruction has been achieved even after 90% of the pixels were dropped.

Now, lets see what happens in the case of a gradient filter, such as the central difference operator [-1 0 1]. Dropping a sample as in the example above leads to an effective adjustment of the filter from [-1 0 1] to [* 0 1]. It is easy to see that any constant offset in the image will now affect the filter response, which is not what was intended. It is important to note here that calculating a new filter normalization does not fix the problem. This is evident in Figure 2 where only 30% of the pixels have been removed but the gradient estimation is very poor.

In a more formal way, the operator [-1 0 1] can be interpreted as a basis vector used to analyze the signal. It is clear that getting a correct response is dependent upon its orthogonality to the constant vector [1 1 1]. Under the influence of uncertainty, this orthogonality cannot be guaranteed. As we shall see in the next section, normalized convolution provides a general framework to handle this problem. See Figure 3.

Figure 3: *Demonstration of the power of normalized convolution in gradient estimation of lossy images (30%).* Left: *Original image.* Middle: *Lossy image.* Right: *Gradient estimation using normalized convolution. Normalized convolution ensures the best possible estimations based on the given set of uncertainties for each pixel neighborhood in the image.*

3. Normalized Convolution Framework

The concept of *Normalized Convolution* (NC) was first introduced by Knutsson and Westin [KW93b,KWW93] to enable analysis of signals with locally varying sample certainty. Uncertainty in signal values can for example stem from known sensor dropouts (such as overexposed pixels in a camera), transmission errors, filtering of signal borders etc. Generally, the certainty of a signal element is modeled in the range $[0..1]$, where 1 corresponds to a fully known element and 0 to a missing sample. In this section we will introduce the basic notions of normalized convolution as a local subspace approximation of a signal using a weighted metric, where weights are proportional to the signal certainty.

Normalized convolution have been used and extended from the original derivation in a number of works, notable is the thesis by Farnebäck [Far02], providing a comprehensive study on the use of NC for local polynomial expansions. The thesis also provides connections between NC and weighted least squares. Mühlich and Mester [MM04] also showed that NC can be interpreted in a statistical signal processing framework. More recently, connections has been found between NC and several modern filtering paradigms [Mil], such as non-local means [BCM05], the

bilateral filter [TM98] and moving least squares [LS81]. Specifically, the difference between the methods can be shown to dependent only on the metric chosen in each local neighborhood.

3.1. Local Subspace Approximation Through Convolutions

In this section the basic signal processing framework used in NC is presented. Due to the limited scope of this paper, the reader is assumed to be familiar with basic concepts such as biorthogonal systems, dual coordinates, the metric tensor and inner product spaces (l^2). More details can be found in [Kre89, GK95].

Given a discrete signal $s(k)$ and a finite filter $f(l)$ with N taps, we can use convolution to compute a filter response,

$$h = s \star f \tag{1}$$

which can also be interpreted as an inner product,

$$h(k) = \langle s(k+l), f^*(-l) \rangle \tag{2}$$

This allows us to interpret convolution as a projection. Convolving $s(k)$ with a series of filters $f_m(l)$, where $m = 1...M$, gives M filter responses $h_m(k)$. These filter responses can be interpreted as the dual coordinates $\tilde{c}$ of the signal in the local *filter basis*. For M filters of length N we get a basis matrix B of size $N \times M$. This gives the local metric (or metric tensor) as

$$G = B^* G_0 B \tag{3}$$

where G_0 is the metric for the orthonormal cartesian coordinate frame. In the trivial case, this metric is defined by the identity matrix. From the dual dual coordinates and the metric we get

$$c = G^{-1} h_m(k) \tag{4}$$

with the coordinates c providing an expression for the local signal relative the basis given by the filters. For example, choosing the filters to be local polynomials $\{1, x, y, x^2, y^2\}$ we can approximate the Taylor expansion of a local signal region.

We can also interpret this subspace projection operation using a familiar least squares terminology. Consider a vector, $\mathbf{v} \in V$ and a subspace $U \subseteq V$ such that $\mathbf{v} = v_{\|} + v_{\perp}$ with $v_{\|} \in U$ and $v_{\perp} \notin U$. Least squares methods are then concerned with minimizing $|| \mathbf{v} - v_{\|} ||$. If B is a base in U and c a set of coordinates describing $v_{\|}$ in B, then this is the same as minimizing $|| \mathbf{v} - Bc ||= (\mathbf{v} - Bc)^* G_0 (\mathbf{v} - Bc)$. In classical linear algebra, the solution to this problem is given by the normal equations, stating that $c = (B^* G_0 B)^{-1} B^* G_0 \mathbf{v}$. Comparing this with Equation 4 we can identify the dual coordinates $\tilde{c}$ as $B^* G_0 \mathbf{v}$ and the metric G as $B^* G_0 B$.

So far, nothing apart from traditional image filtering has been introduced. All we have done is to interpret filtering at any given point in the image as a projection of a local patch of the image into a subspace spanned by a set of basis vectors.

3.2. Incorporating Filter Applicability

For our problem setting, we are interested in weighting the subspace projection (which can be seen as a least squares estimate) by a suitable *applicability function*, $a(l)$, describing the influence of neighbor pixels on the center pixel. To achieve this, instead of convolving the signal, $s(k)$, with the filters directly, we first multiply them with the applicability function to obtain a suitable *localization* of the filter basis. Thus we use filters defined as

$$g_m(l) = a(-l) f_m^*(-l) \tag{5}$$

where $a(l)$ is chosen to be a localizing function such that $a(l) > 0$, such as a Gaussian function.

Convolving $s(k)$ with $g_m(l)$ gives

$$h_m(k) = \sum_l s(k+l) g_m(-l) \tag{6}$$

$$= \sum_l s(k+l) a(l) f_m^*(l) \tag{7}$$

which describes a generalized inner product between $s(k)$ and $f_m^*(l)$, where the weighting is provided by $a(l)$. This generalized inner product redefines the metric in the trivial basis as

$$G_0 = diag(a(l)) \tag{8}$$

which leads to an expression of the metric in B as

$$G = B^* G_0 B \tag{9}$$

$$= B^* diag(a(l)) B \tag{10}$$

It is worth to note here that the matric G is still invariant as to where in the image the convolution is applied. In other words, the metric we use for our projection is always the same. With the introduction of uncertainty, this will no longer hold.

3.3. Incorporating Uncertainty

Given a signal $s(k)$ and accompanying signal certainty $r(k)$, we can allow the inner product to adapt to the local uncertainty of the signal. Incorporating the signal certainty in the convolution gives

$$h_m(k) = \sum_l s(k+l) r(k+l) a(l) f_m^*(l) \tag{11}$$

corresponding to a weighted inner product between $s(k)$ and $f_m^*(l)$, where the weights are set according to $r(k+l)a(l)$. This ensures that low certainty entries in the signal are given less weight when estimating neighborhood pixel regions. As

$r(k)$ varies with position, we thus get a position dependent metric $G_0(k)$.

$$G_0(k) = diag(a(l)r(k+l)) \tag{12}$$

In the filter basis, B, we can write the metric as

$$G(k) = B^* G_0(k) B \tag{13}$$
$$= B^* diag(a(l)r(k+l))B \tag{14}$$

and the local coordinates of the filter basis is given by inserting this expression into Equation (4)

$$c = (B^* diag(a(l)r(k+l))B)^{-1} h_m(k) \tag{15}$$

It is important to note that the metric is no longer spatially invariant, meaning we have to recompute G^{-1} at each point in the image. The spatial invariance also has the implication that the convolution will always be non-separable, independently of which filters that are used. Both of these things will impact the optimizations described in Section 4.

3.4. NC Optimization by Pre-computation

Computing the coefficients for every point using Equation (15) is a costly, as both a matrix inverse and several matrix products need to be evaluated. Explicitly expressing the computations with inner products and vector multiplications, denoting the columns of the base matrix B as b_m where $m = 1..M$, we can write Equation (15) as

$$c = \begin{pmatrix} \langle a \cdot r \cdot b_1, b_1 \rangle & \dots & \langle a \cdot r \cdot b_1, b_M \rangle \\ \vdots & \ddots & \vdots \\ \langle a \cdot r \cdot b_M, b_1 \rangle & \dots & \langle a \cdot r \cdot b_M, b_M \rangle \end{pmatrix}^{-1} \begin{pmatrix} \langle a \cdot r \cdot b_1, s \rangle \\ \vdots \\ \langle a \cdot r \cdot b_m, s \rangle \end{pmatrix} \tag{16}$$

Using the properties of the inner products, this can be rewritten as

$$c = \begin{pmatrix} \langle a \cdot b_1 \cdot b_1^*, r \rangle & \dots & \langle a \cdot b_1 \cdot b_M^*, r \rangle \\ \vdots & \ddots & \vdots \\ \langle a \cdot b_M \cdot b_1^*, r \rangle & \dots & \langle a \cdot b_M \cdot b_M^*, r \rangle \end{pmatrix}^{-1} \begin{pmatrix} \langle a \cdot b_1, r \cdot s \rangle \\ \vdots \\ \langle a \cdot b_m, r \cdot s \rangle \end{pmatrix} \tag{17}$$

Pre-computing the quantities $(a \cdot b_i \cdot b_j^*)$, $(a \cdot b_i)$ and $r \cdot s$ for all $i, j = 1...M$, significantly decreases the total number of multiplications necessary for each point/pixel in the signal. See [Far02] for more details.

4. Implementing Normalized Convolution in CUDA

This section will highlight and explain strengths and difficulties of performing normalized convolution on the Nvidia CUDA platform. In short, CUDA [Nvi09, Nvi10] is a parallel computing architecture developed to provide a C, or even C++, like interface to modern GPUs. Hierarchial layers of computational resources and memory provide opportunities to exploit parallel computation, coalesced access to global memory, utilization of fast shared memory, loop unrolling and latency hiding.

This section will begin by summarizing some of the key optimizations that can be used to speed up any non-separable convolution and will then proceed to difficulties related to normalized convolution. The impact of the optimizations on system performance is presented in Section 5.

A note on the word *kernel*: CUDA programs are called kernels, so to avoid the ambiguity between filter kernels and CUDA kernels we will henceforth simply refer to filters as filters while a kernel always refer to a CUDA program.

4.1. Optimizing Non-separable Convolution

Non-separable filtering is typically performed on patches of $4^2 - 16^2$ pixels on a version of the image that has been padded to the nearest multiple of 16. With one thread per pixel, this pattern allows fast coalesced reads over both image dimensions, copying one image patch to the shared memory of each available multiprocessor. Any additional boundary padding necessary for filtering (half the size of the filter) is typically copied as a secondary step or by additional threads. Highest performance is typically achieved when the patch size is as large as possible without overflowing the shared memory. Another important optimization used in this work is loop unrolling which, as explained later, is available 'for free' as the implementation already utilizes template functions. The next section will discuss optimizations specific for Normalized Convolution.

4.2. Optimizing Normalized Convolution

In Section 3.1, we saw that the main mathematical difference between standard and normalized convolution is that the subspace into which the signal is projected changes spatially in the image (due to the uncertainty). What this means in terms of computation is that we need to build, and invert, the metric matrix at each pixel.

There is plenty of documentation available for large matrix inversion using CUDA. There are, however, very few examples where matrices have be inverted on a per-thread basis (which is the case in normalized convolution). Applying standard, serial methods for matrix inversion is often problematic as these rely on recursion, a feature not present in the CUDA programming language.

The matrices that need to be inverted are $M \times M$ where M is the number of bases. These matrices are typically small, i.e. $M < 7$, but still require iterative solutions if $M > 3$. The iterative matrix inversion used in this work is based on the classic formula of scaling the adjoint matrix, Equation 18:

$$A^{-1} = \frac{1}{det(A)} * Adj(A) \tag{18}$$

Although this method is known to be computationally expensive and suffer from numerical inaccuracies for larger

matrices it was deemed sufficient for the scope of this work. The computation of the determinant typically uses a recursive scheme with one dimensionality reduction per recursion. This is also true for the computation of the co-factor matrix as it relies on computation of determinants for a range of sub-matrices. This recursive process is performed through template instantiation using the following syntax.

```
template<int M> __device__ float Determinant(...)
    { ... Determinant<M-1>(...) ... }
template<> __device__ float Determinant<3>(...);
template<> __device__ float Determinant<2>(...);
template<> __device__ float Determinant<1>(...);
```

This effectively instantiates a unique function per recursion step, and thus circumvents the problem that no function may call itself. It is important to note, however, that the manual instantiation of $M = 1$ is necessary or an infinite number of functions will be instantiated, $[M, -\infty]$. The manual instantiations of $M = 2$ and $M = 3$ are provided for optimization purposes.

While function instantiation allows us to execute recursive processes, it also comes with a memory cost. The determinant function, for example, need to allocate a small array to hold a sub-matrix. If the 'recursive' call to the next determinant function is done within the scope of this allocation, the result is that subsequent allocations will be performed on each level of the recursive process. Now, remember that this is performed on a per-thread basis and that the amount of available registers and shared memory is small. The effect of this is that inversion of large matrices ($M \gtrsim 5$) will be significantly slowed as the shared memory is overflowed. Exactly when this happens depend on multiple factors, including filter and patch sizes.

4.3. Implementation

This section will provide a few additional notes on the CUDA implementation of normalized convolution. First, the allocation and utilization of the various memory types in CUDA will be detailed.

```
__shared__ float s_data[...];
__shared__ float r_data[...];
__constant__ float precomp_abb[...];
__constant__ float precomp_ab[...];
```

The two shared memory buffers, *s_data* and *r_data*, are allocated on a per-block basis and are initiated with the appropriate patches of the image and its uncertainty information during kernel launch. The two constant arrays are allocated on a per-grid basis and are initiated from the host code. *precomp_abb* and *precomp_ab* hold the applicability and precomputed tables described in Section 3.4.

As constant memory cannot be allocated dynamically, an upper boundary must be set for the number of bases and filter size. The simplest way to do so is to use pre-processor macros (i.e, *#define*). Similar to the template situation, a fairly straightforward way to achieve flexibility even when using 'fixed' values is to compile the same code multiple times with different macro values and then select the appropriate one during runtime. This is another example of a tradeoff between speed and flexibility in CUDA.

With the maximum number of bases, maximum kernel size and patch size denoted K_m, M_m and T respectively. Then, the two patches of shared memory requires $(T + 2 * K_m) * (T + 2 * K_m)$ floats each, and the two constant arrays ${M_m}^2 * {K_m}^2$ and $M_m * {K_m}^2$ floats respectively. In our implementation, patch size is set as a pre-processor macro but could potentially be passed as an argument as CUDA does support dynamic allocation of shared memory.

In the next section, the CUDA implementation is compared to a C++ implementation. The C++ implementation is to a large degree identical to the CUDA code, but use a series of *for*-loops to account for the lack of parallelism on the CPU.

5. Results

To evaluate how normalized convolution performs in CUDA, different implementation options were timed separately. These are presented here, followed by a general discussion in Section 6. All performance tests were performed at an image resolution of 512×512 pixels using a 2.67GHz CPU paired with a Nvidia GTX560Ti GPU.

First, the CUDA implementation is compared to a C++ implementation. Results can be found in Figure 4. As expected, the CUDA implementation is significantly faster over the entire parameter space. The CUDA implementation also displays little dependency on filter size thanks to the utilization of fast shared memory cache and registers. At the same time, however, a steeper increase in computation time is evident for cases with more than 4 bases. This steep performance increase is likely an effect of the increased memory consumption, forcing more samples to be taken outside of the GPUs fast local registers.

Second, the CUDA specific effects of patch size, memory layout and template usage were investigated. Figure 5 displays the performance impact when using template instantiations instead of relying on function parameters for configuring the number of bases and filter size. The template instantiation is nearly twice as fast as a result of loop unrolling and compiler optimizations. Figure 6 shows the benefit of utilizing shared memory in favor of the slower global memory. Naturally, utilizing shared memory is increasingly important the more memory that is read, such as for increasing filter sizes. Figure 7 shows the impact of computational speed when varying the patch size. As expected, larger patch sizes results in faster computation times. One reason for this is the increased occupancy of the various levels of computation on the GPU. Larger patch sizes also reduce the rela-

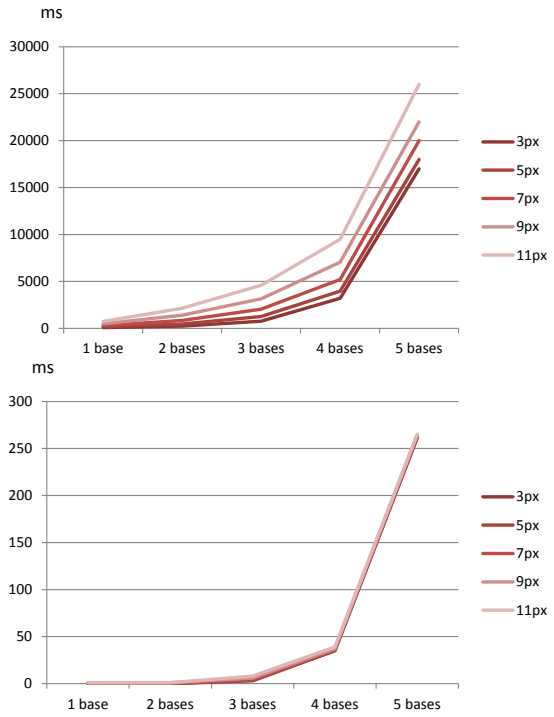

Figure 4: *C++ vs. CUDA.* Top: *C++ timings, in milliseconds, for varying number of bases and filter size.* Bottom: *CUDA timings, in milliseconds, for varying number of bases and filter size. The CUDA implementation is approximately 100 times faster but also displays more drastic increase in time consumption for higher number of bases, $M > 4$. It is also worth to note that the filter size has less impact in the CUDA implementation thanks to the utilization of shared memory.*

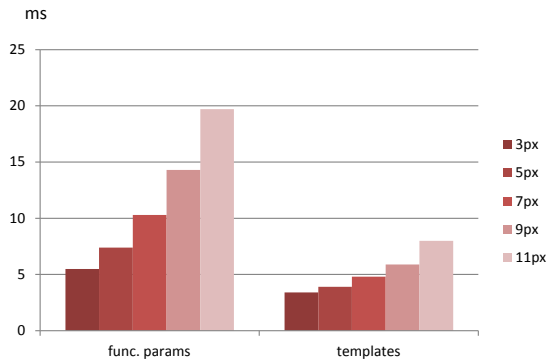

Figure 5: *Templates are used as they offer a decent trade-off between hardware optimization and program flexibility. Two variables are considered here; the number of bases and the filter size. Passing these variables as function arguments nearly doubles the computation time relative the case where the values are 'fixed' as template arguments. The downside to templates is that a large number of functions must be instantiated in order to cover all combinations of values.*

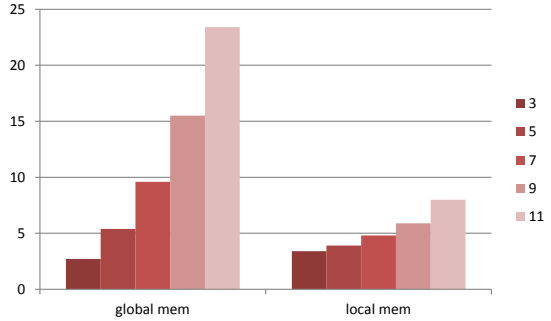

Figure 6: *Utilization of shared memory is essential in convolution as any given memory position is samples multiple times when applying a filter over a series of neighboring pixels. The larger the filter, the more important the shared memory becomes. The timings were performed using template functions with a three base setup.*

tive size of the filter skirt area (the pixels outside the patch but inside the filter radii). As a side note, this is a problem with non-separable filtering in general if the dimensionality of the images increases, as the relative area of the necessary boundary padding for each patch increases.

Figure 8 shows normalized gradient estimation on a real world data set where the *top row* contains (float left ro right); the reference image, lossy image and the uncertainty map. The *middle row* shows three types of gradient estimation on the reference image and the *bottom rows* shows the same gradient estimations on the lossy image. The three types of gradient estimation are (from left to right); no uncertainty compensation, re-weighted filters, and normalized convolution. The example is taken from a series of Computed Tomography images acquired at varying levels of dosage. the reference image was acquired at a dosage of 180mAs while the low-dose image was acquired at 12mAs. The uncertainty image was extracted by taking the absolute difference of the two images. It should be noted that the scans were taken consecutively for the purpose of post-mortem imaging, so no registration was necessary to align the images. This is obviously a fabricated case, as we naturally would not have access to the high dose image when scanning live patients or would work directly on the high dose image in post mortem cases. It does, however, serve the purpose of highlighting how large of an effect uncertainty can have on gradient estimation in medical imaging.

6. Conclusions

In this work we have taken one step towards an uncertainty aware pipeline for medical imaging. Normalized con-

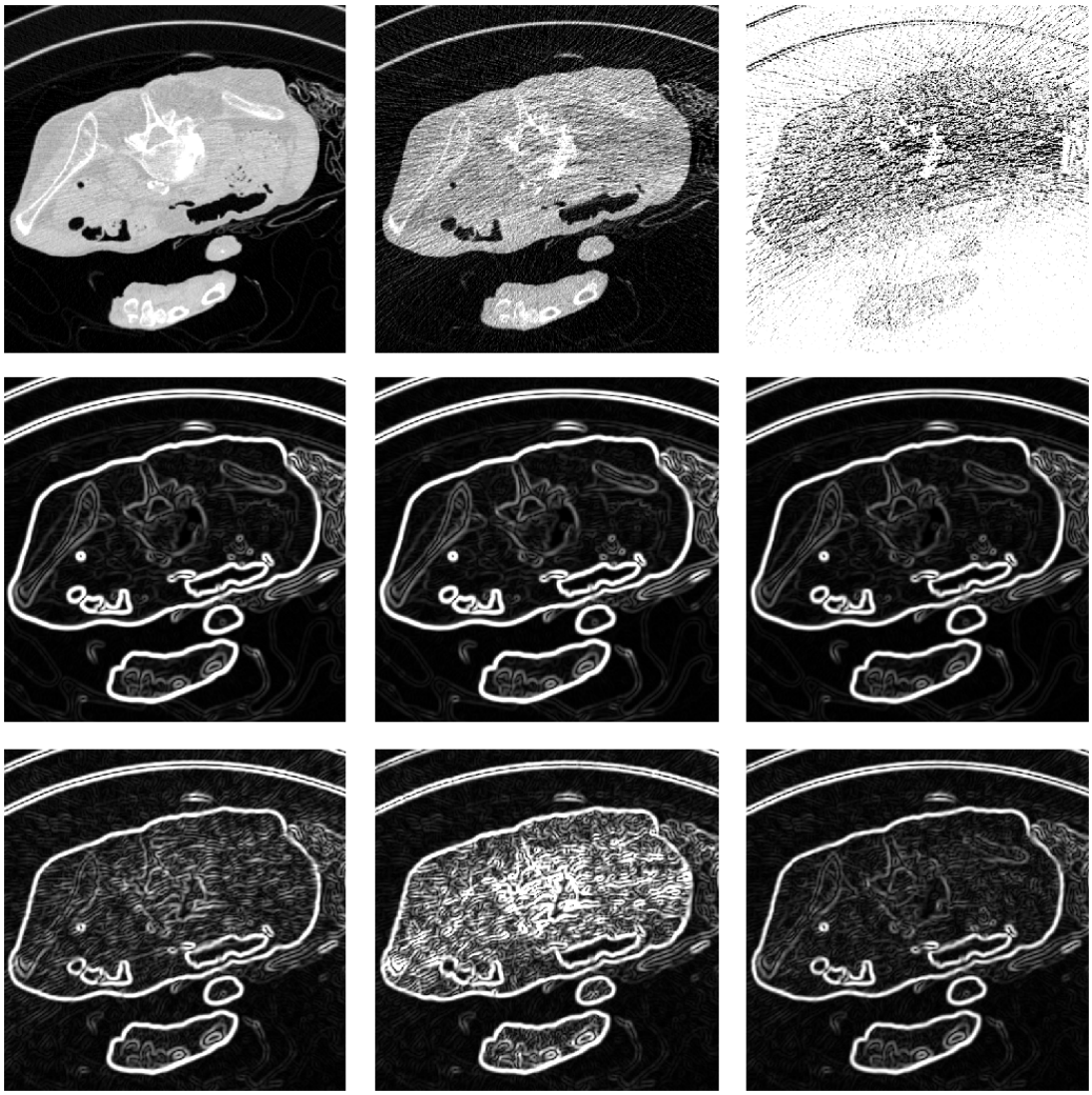

Figure 8: *Noise reduction in gradient computation on low-dose Compute Tomography (ldCT) images.* Top row: *High dose image (reference), low-dose image, difference image depicting noise in the* $[0, 1]$ *range. The* middle *and* bottom *rows show gradient computation on the reference and low-dose image respectively.* Left: *Uncertainty has been discarded.* Center: *Using normalized averaging.* Right: *Using Normalized convolution. Normalized convolution clearly provides a better approximation of the gradients under the presence of uncertainty. The images were generated using a symmetric Gaussian applicability function with* $\sigma^2 = 2$ *and a filter size of* 11×11 *pixels. The CT data resolution is* 512×512 *pixels.*

volution, an established uncertainty aware image processing technique, has been implemented in the CUDA programming language to meet the high efficiency demands of the medical domain. We have demonstrated the importance of maintaining an uncertainty aware pipeline by showing the effect uncertain samples can have on gradient estimation.

However great potential CUDA has for speeding up computation of normalized convolution, it is not without limitations. As expected, computation time and storage requirements grow drastically with increased number of bases. This is, to a large degree, dependent on the chosen scheme to solve the linear system. Since the primary focus for this work was gradient estimation, it was deemed sufficient to use the scaled adjoint matrix and solve the system by using the inverse. For larger systems, however, more efficient and numerically robust approaches would be necessary. Memory consumption in particular will become a major concern as the number of bases grows and local/shared memory is over-

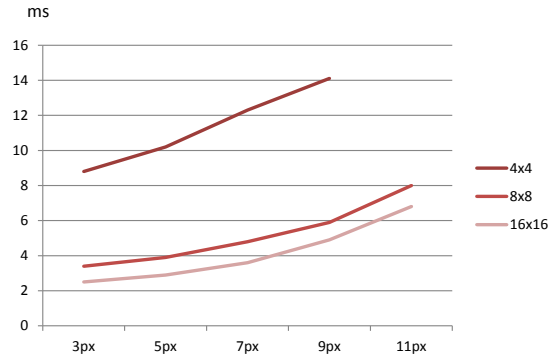

Figure 7: *The effect on CUDA performance when varying the patch size. While the patch size should be as large as possible it is limited by the memory consumption of the threads it triggers. The timings were performed using template functions with a three base setup.*

flowed. We did experience numerical errors in the matrix inversion process when dealing with larger matrices which further indicates that finding the inverse by the adjoint is only valid solution when dealing with low number of bases. Future work will naturally go towards alternative solutions to solve higher dimensional linear systems under the constraint of limited fast memory.

Acknowledgements

The authors would like to thank Hans Knutsson of the Department of Biomedical Engineering, Linköping University and Claes Lundström of the Department of Science and Technology, Linköping University.

References

[BA10] BRINK J. A., AMIS JR E. S.: Image Wisely: a campaign to increase awareness about adult radiation protection. *Radiology 257*, 3 (2010), 601–602. 1

[BCM05] BUADES A., COLL B., MOREL J. M.: A review of image denoising algorithms, with a new one. *Simul 4* (2005), 490–530. 2

[Can86] CANNY J.: A computational approach to edge detection. *IEEE transactions on pattern analysis and machine intelligence* (1986). 1

[Far02] FARNEBCK G.: *Polynomial Expansion for Orientation and Motion Estimation*. PhD thesis, Linkping University, Sweden, SE-581 83 Linkoping, Sweden, 2002. Dissertation No 790, ISBN 91-7373-475-6. 2, 4

[GK83] GRANLUND G. H., KNUTSSON H.: Contrast of Structured and Homogenous Representations. In *Physical and Biological Processing of Images*, Braddick O. J., Sleigh A. C., (Eds.). Springer Verlag, Berlin, 1983, pp. 282–303. 1

[GK95] GRANLUND G. H., KNUTSSON H.: *Signal Processing for Computer Vision*. Kluwer Academic Publishers, 1995. ISBN 0-7923-9530-1. 3

[Gra78] GRANLUND G. H.: In search of a general picture processing operator. *Computer Graphics and Image Processing 8*, 2 (1978), 155–173. 1

[HKRs*06] HADWIGER M., KNISS J. M., REZK-SALAMA C., WEISKOPF D., ENGEL K.: *Real-time Volume Graphics*. A. K. Peters, Ltd., Natick, MA, USA, 2006. 1

[Joh04] JOHNSON C.: Top Scientific Visualization Research Problems. *IEEE Computer Graphics and Applications 24*, 4 (2004), 13–17. 1

[Kay01] KAY S. M.: *Fundamentals o Statistical Signal Processing, Volume 1*. Prentice Hall, 2001, 2001. 1

[KKH02] KNISS J., KINDLMANN G., HANSEN C.: Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics 8* (July 2002), 270–285. 1

[Knu89] KNUTSSON H.: Representing local structure using tensors. In *The 6th Scandinavian Conference on Image Analysis* (Oulu, Finland, June 1989), -, pp. 244–251. Report LiTH-ISY-I-1019, Computer Vision Laboratory, Linköping University, Sweden, 1989. 1

[Kre89] KREYSIG E.: *Introductory Functional Analysis with Applications*. Wiley, 1989. 3

[KS08] KIRBY R. M., SILVA C. T.: The need for verifiable visualization. *IEEE Comput. Graph. Appl. 28* (September 2008), 78–83. 1

[KW93a] KNUTSSON H., WESTIN C. F.: Normalized and differential convolution. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, x (1993), 515–523. 1

[KW93b] KNUTSSON H., WESTIN C.-F.: Normalized and differential convolution. In *Computer Vision and Pattern Recognition, 1993. Proceedings CVPR '93., 1993 IEEE Computer Society Conference on* (jun. 1993), IEEE, pp. 515 –523. 2

[KWW93] KNUTSSON H., WESTIN C.-F., WESTELIUS C.-J.: Filtering of Uncertain Irregularly Sampled Multidimensional Data. In *Twenty-seventh Asilomar Conf. on Signals, Systems & Computers* (Pacific Grove, California, USA, November 1993), IEEE, pp. 1301–1309. 2

[LLPY07] LUNDSTRÖM C., LJUNG P., PERSSON A., YNNERMAN A.: Uncertainty visualization in medical volume rendering using probabilistic animation. *IEEE transactions on visualization and computer graphics 13*, 6 (2007), 1648–1655. 1

[LP11] LUNDSTRÖM C., PERSSON A.: Characterizing visual analytics in diagnostic imaging. *International Workshop on Visual Analytics* (2011). 1

[LS81] LANCASTER P., SALKAUSKAS K.: Surfaces Generated by Moving Least Squares Methods. *Mathematics of Computation 37*, 155 (1981), 141–158. 3

[Mil] MILANFAR P.: A tour of modern image filtering. To appear in IEEE Signal Processing Magazine. 2

[MM04] MÜHLICH M., MESTER R.: A statistical extension of normalized convolution and its usage for image interpolation. *Proceedings of European Confereance on Signal Processing, (EuraSip)* (2004). 2

[Nvi09] NVIDIA CORPORATION: NVIDIA CUDA C Programming Best Practices Guide CUDA Toolkit 2.3. 4

[Nvi10] NVIDIA CORPORATION: Nvidia CUDA Programming Guide Version 2.3. 4

[TM98] TOMASI C., MANDUCHI R.: Bilateral filtering for gray and color images. *Computer Vision, IEEE International Conference on 0* (1998), 839. 3