CONSTRUCTING A SPARSE CONVOLUTION MATRIX FOR SHIFT VARYING IMAGE RESTORATION PROBLEMS

Stanley H. Chan

ECE Dept, UCSD, La Jolla, CA 92092-0407
http://videoprocessing.ucsd.edu

ABSTRACT

Convolution operator is a linear operator characterized by a point spread functions (PSF). In classical image restoration problems, the blur is usually shift invariant and so the convolution operator can be characterized by one single PSF. This assumption allows one to use fast operations such as Fast Fourier Transform (FFT) to perform a matrix-vector computation efficiently. However, as in most of the video motion deblurring problems, the blur is shift variant and so the matrix-vector multiplication can be difficult to perform. In this paper, we propose an efficient method to construct the convolution matrix explicitly. We exploit the submatrix structure of the convolution matrix and systematically assigning values to the nonzero locations. For small to medium sized images, the convolution matrix gives superior speed than some state-of-art convolution operators.

Index Terms— sparse matrix, convolution matrix, shift variant, blur, image deblurring.

1. INTRODUCTION

It is well known that image blur can be modeled as a convolution between the image and the point spread function (PSF). Due to the linearity of convolution, we can express the observed image as

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \eta, \tag{1}$$

where $\mathbf{A}$ is the convolution matrix, $\mathbf{x}$ is a vector representing the object (usually as an unknown variable), $\mathbf{y}$ is a vector representing the observed image, and η is a noise vector.

In case where the PSF is *shift invariant*, i.e., all pixels in the image are blurred by the same PSF, the forward operation $\mathbf{A}\mathbf{x}$ can be done via fast Fourier Transforms (FFT) [1]. However, if the PSF is *shift variant*, the forward operation cannot be done using FFT.

Shift varying blur is common in local motion blur problems. For example, a video having a stationary background with fast moving objects is a shift varying problem, because stationary background is usually sharp whereas the front ground moving object is blurred.

The effectiveness of computing a forward operation $\mathbf{A}\mathbf{x}$ is essential for iterative methods such as PDCO [2], LSQR [3], RestoreTools [4], projected gradient [5] and many others. However, there are not many attempts to construct a fast convolution operator, especially for shift varying blur.

There are two major categories of methods that construct the convolution operator. Popular image restoration tools such as RestoreTools [6] and HNO [4] use object oriented programming techniques. These methods partition the image into blocks and each block is blurred by a different PSF, using FFT. So if the motion is so complicated that every pixel has to be blurred by a different PSF, then these methods fail (see section III for experiments).

Another method is to construct the convolution matrix explicitly. In [2], the author mentioned a block circular with circular block (BCCB) matrix but he did not discuss the methods to construct the matrix. [1] provides codes to construct a small sized BCCB matrix, yet they do not consider shift varying situations. Also, the method in [1] is slow and memory demanding. MATLAB's built-in command `convmtx2` is an efficient algorithm to construct a convolution matrix. However `convmtx2` does not handle shift variant blur.

In this paper we propose a method to construct a shift variant operator efficiently. For small to medium sized images, constructing the convolution matrix explicitly has the following advantages:

1. Having the sparse convolution matrix allows us to apply many powerful linear algebraic tools such as LU/ QR/ Cholesky/ SVD. Consequently, advanced algorithms such as truncated singular value decomposition (TSVD [4]) can be applied.
2. If the kernel is shift invariant and its support is small, the forward operation $\mathbf{A}\mathbf{x}$ using the convolution matrix is much more effective than FFT. Consider an image of n pixels and kernel of l entries. Cost for matrix-vector multiplication is $O(nl)$, but cost for FFT is $O(n \log n)$.
3. If the kernel is shift variant, FFT methods cannot be used. Although one can approximate the situation using locally shift invariant PSFs, this will fail for large number of PSFs and small block size. Convolution matrix can resolve this issue because once the matrix is constructed, cost of matrix vector multiplication is $O(nl)$, regardless the number of PSFs.

The primary application of this paper is for video motion deblurring problems, especially for videos captured by mobile phones or surveillance cameras. These videos are relatively small in size, but the motion can be complicated. Therefore the image image we consider is from small (SQCIF 128×96) to medium (CIF 352×288).

The organization of this paper is as follows: In section II we discuss the proposed algorithm, and then handle of boundary conditions. In section III we show experimental results with comparisons to several above mentioned methods. Due to limited space, full experimental results are available on our website. Last in section IV we discuss the pros and cons of the proposed methods, and give a conclusion.

2. PROPOSED ALGORITHM

2.1. Structure of A

The proposed algorithm is a systematic way of allocating blur kernel values into the convolution matrix. Suppose the image is of size $M \times N$, and the blur kernel is of size $P \times Q$. We start by ignoring the circular boundary conditions. So the convolution matrix has size

of $(M+P-1)(N+Q-1) \times MN$. Partition the convolution matrix into submatrices $A_{i,j}$ each of size $(M+P-1) \times M$:

$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1l} \\ A_{21} & A_{22} & \cdots & A_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ A_{k1} & A_{k2} & \cdots & A_{kl} \end{pmatrix} = \begin{pmatrix} \times & & & \\ \times & \times & & \\ & \ddots & \ddots & \\ & & \times & \times \\ & & & \times \end{pmatrix} \quad (2)$$

Effectively, the (i,j)-th submatrix $A_{i,j}$ can be interpreted as the *action from* the j-th column of the input *to* the i-th column of the output. Since the blur kernel has a small support relative to the image size, most of the off-diagonal submatrices are zero. The nonzero submatrices will be located along the diagonal, forming a banded diagonal of width Q submatrices, where Q is the number of rows of the blur kernel. The right hand side of Eq. 2 shows an example of $Q = 2$.

2.2. Indexing

The novelty of the proposed method is the usage of a 4-dimensional cube to allocate kernel values in $\mathbf{A}$, regardless if it is shift varying or not. The 4-dimensional cube has size $M \times P \times N \times Q$. The (i,p,j,q)-th entry of the cube is the (p,q)-th kernel value at pixel (i,j). Stacking the entries of the cube into a column vector, we denote it as T_{val}.

Corresponding to T_{val} there is a column index and a row index vector T_{col} and T_{row} respectively. The column and row indices run along the principle diagonal submatrices, then run along the submatrice on one-diagonal off the principal, two-diagonal off the principal and so on. Within each submatrix, the column and row indices also run in the similar manner: first run along the principal diagonal, then run along one diagonal off the principal and so on.

Although at a first glance a 4-dimensional cube seems to be a big array, the cube is actually not very big as it contains only the *nonzero* elements of the kernel. As compared to [6] where a kernel is first padded with zeros to the size of image and then stored as an entry of a cell structure, the proposed 4-dimensional cube requires much less memory. Besides, it is possible to further reduce the memory of the proposed method by considering repeating kernels. This is subjected to our future work.

2.3. Boundary Conditions

Suppose an image is partitioned into several regions of different blurs. If one uses the above algorithm to construct $\mathbf{A}$, then not only the boundaries of the entire image, but also boundaries of each partition will show serious problems of zero padding.

To resolve this issue, the 4-dimensional cube plays an important role - by simply shuffling the values in the cube in a stair case manner, the boundary conditions for each blur kernel can be satisfied: for a fixed q-th column of the kernel, and fixed j-th column of the image, the (i,p)-th entry can be reallocated at $(i-p+1,p)$. Pictorially the (i,p)-th entry can be visualized as

$$\begin{pmatrix} & \cdots & & \cdots & \\ & \cdots & & \cdots & \\ & \cdots & & \cdots & \\ \times & \times & \times & \times & \times \\ \triangle & \triangle & \triangle & \triangle & \triangle \\ \circ & \circ & \circ & \circ & \circ \end{pmatrix} \rightarrow \begin{pmatrix} & \cdots & & \cdots & \\ & & \times & \triangle & \circ \\ & \times & \triangle & \circ & \circ \\ \times & \triangle & \circ & \circ & \circ \\ \triangle & \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ & \circ \end{pmatrix}.$$

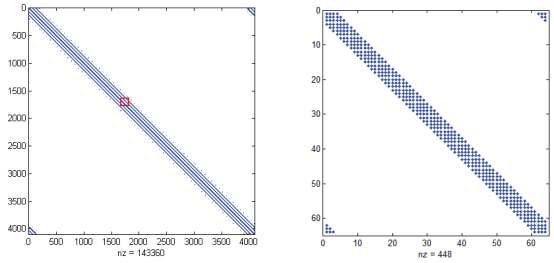
The reallocation of the j,q-th entry can be done similarly as now (j,q)-th entry should be located at $(j-q+1,q)$.

Last, to satisfy the boundary condition for the entire image, the nonzero submatrices in the first $Q/2$ rows of $\mathbf{A}$ has to be moved to the last $Q/2$ rows, and vice versa for the non-zero submatrices in the last $Q/2$ rows. This is illustrated in Eq. 3. Note that after moving the rows, size of $\mathbf{A}$ becomes $(M+P-1)N \times MN$.

$$\begin{pmatrix} \circ & & & & \\ \times & \times & & & \\ & \ddots & \ddots & & \\ & & \times & \times & \\ & & & \triangle & \end{pmatrix} \rightarrow \begin{pmatrix} \times & \times & & & \triangle \\ \times & \times & & & \\ & & \ddots & & \\ & & & \times & \times \\ \circ & & & \times & \times \end{pmatrix} \quad (3)$$

Since a submatrix has the same as the structure as that of $\mathbf{A}$, applying the above method to each submatrix the boundary conditions are satisfied. Note that size of $\mathbf{A}$ is reduced to $MN \times MN$.

Fig. 1 is an example showing the location of non-zero entries of the sparse matrix. The kernel used here has size 5×7.



(a) sparse matrix (b) zoom in of (a)

Fig. 1. Structure of the constructed sparse matrix.

3. EXPERIMENTS

The following experiments are the comparisons with some state-of-art approaches, namely `bccb` [1], `psfMatrix` [6], [4], and a straight forward for-loop approach. In [1] the author did not explicitly mention about the construction of the convolution matrix, but codes of constructing the matrix is freely available online. `psfMatrix` is an FFT based approach, and the operator is defined through objects. The for-loop approach is a straight forward implementation of shift variant PSFs. Suppose every pixel of an $M \times N$ image f has a different PSF $h_{x,y}$. Then the blurred pixel value at position (x,y) is calculated using the following algorithm.

Algorithm 1 For-Loop implementation

```
for x = 1 : M do
  for y = 1 : N do
    g(x,y) = Σ_{i,j} h_{x,y}(x − i, y − j) f(i,j)
  end for
end for
```

3.1. Motion blur by proposed method

We first show some motion blurred images using the proposed algorithm. In Fig. 2, the images are blurred according to the motion vector fields specified. Although we show coarse motion vector fields, the actual ones are dense, i.e., every pixel has a different

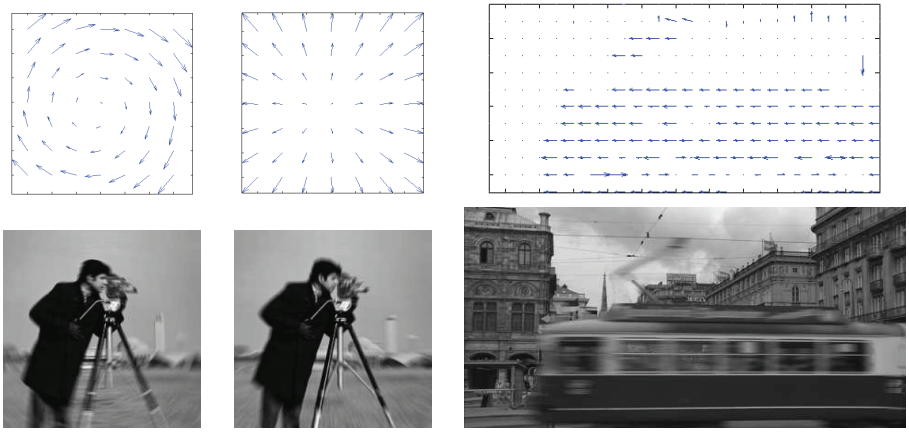


Fig. 2. Some results of the blur operations performed using the proposed algorithm. The blur is calculated based on motion vectors specified in the top row. No knowledge about the motion structure is required.

PSF. The effectiveness of the proposed algorithm is that the operator can be constructed without knowing the structure of the motion, as opposed to many motion blur modeling algorithms where the underlying model (such as rotation, translation) has to be specified before the estimation.

3.2. Shift Invariant Experiment

The purpose of this experiment is to compare performance under shift invariant conditions. The image size ranges from SQCIF (176×144) to CIF (352×288). The PSF is Gaussian with fixed variance $\sigma = 3$. Sizes of the PSF varies from 3×3 to 7×7. Partial list of the results are shown in Table 1, and the complete list can be found on our website. As one can expect, explicitly constructing a convolution matrix takes time. So the construction time of `bccb` and the proposed method are longer than that of `psfMatrix` and for-loop, especially when image size increases. However, as the forward computation time is concerned, the proposed method clearly outperforms the other methods as shown.

Table 1. Invariant Blur Kernel. Time to construct the operator, and time to perform one matrix-vector multiplication.

Construction Time (sec)					
image size	kernel size	`bccb` [1]	`psfMat -rix` [6]	for-loop	Propose
128x96	5x5	38.6572	0.0423	0.0014	0.5196
	7x7	166.9316	0.0419	0.0014	3.3319
352x288	5x5	fail	0.1072	0.0015	30.742
	7x7	fail	0.1002	0.0015	90.235
Matrix-vector multiplication Time (sec)					
image size	kernel size	`bccb` [1]	`psfMat -rix` [6]	for-loop	Propose
128x96	5x5	0.0021	0.043	0.0754	0.0031
	7x7	0.0032	0.0424	0.0773	0.0045
352x288	5x5	fail	0.1839	0.6357	0.0139
	7x7	fail	0.1767	0.6357	0.0238

3.3. Shift Variant Experiment

This experiment concerns about shift varying PSFs. First, an image is partitioned into blocks and each block has a PSF. Therefore, smaller block size implies more blocks and hence more PSFs. The PSFs in this experiments are Gaussian PSFs with size 5×5. Its variance is a function of the position of the image. Pixels around the center of the image have smaller variance, whereas pixels near the edge of the image have larger variance.

As shown in Table 2, `bccb` fails to construct the convolution matrix due to lack of memory. `psfMatrix` defines an object easily for large block sizes, but fails for small block sizes. This is because `psfMatrix` has to store a large number of PSFs, and each PSF has to be padded with zeros to make itself the same size as the image.

The time needed to construct a convolution matrix using the proposed method is 10 seconds on average, and the computing time for a matrix-vector multiplication is as short as 0.001 seconds. Compared to `psfMatrix` where the computing time can be as long as 20 seconds (image size 256×256, block size 16×16), the proposed method is clearly better performed.

Table 2. Variant Blur Kernel. Time to construct the operator, and time to perform one matrix-vector multiplication.

Construction Time (sec)					
image size	block size	`bccb` [1]	`psfMat -rix` [6]	for-loop	Proposed
256x256	64x64	fail	0.2227	0.0015	10.3789
	32x32	fail	0.7129	0.0015	10.6477
	16x16	fail	2.6279	0.0014	10.8041
	8x8	fail	fail	0.0014	11.0712
	4x4	fail	fail	0.0014	11.7754
Matrix-vector multiplication Time (sec)					
image size	block size	`bccb` [1]	`psfMat -rix`[6]	for-loop	Proposed
256x256	64x64	fail	1.1633	0.4112	0.0291
	32x32	fail	4.3206	0.4102	0.0287
	16x16	fail	23.439	0.4105	0.0424
	8x8	fail	fail	0.4137	0.0345
	4x4	fail	fail	0.4103	0.0320

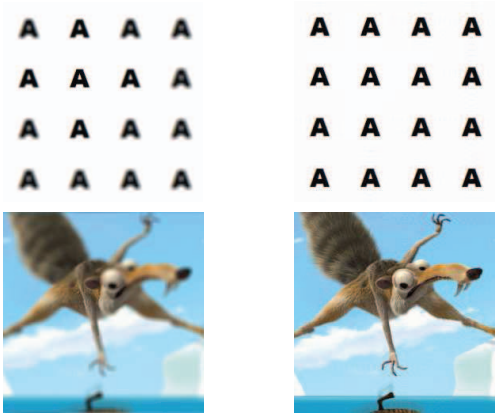
3.4. Deblurring Experiment

Our last experiment compares the performance of the methods for solving image restoration problems. Since this paper is independent to a specific image restoration algorithm, we illustrate the speed improvement by solving a Tikhonov regularized least square problem using LSQR [3]: minimize $\|\mathbf{A}\mathbf{x}-\mathbf{y}\|^2+\lambda\|\mathbf{x}\|^2$, with $\mathbf{A}$ being a shift (in)variant convolution matrix, $\mathbf{x}$ a 256×256 image arranged in lexicographic order, and $\mathbf{y}$ the observed image. Here we set $\lambda = 10^{-2}$, tolerance $\texttt{atol} = \texttt{btol} = 10^{-6}$.

The proposed method explicitly constructs the sparse convolution matrices. Therefore, the computation time for a matrix-vector multiplication is significantly less than its counterparts. Especially for shift varying blurs, the proposed method is the only one that enables the computation. Table 3 shows the timings, and Fig. 3 shows some of the reconstructed images. More results can be found on our web.

Table 3. Computating time using LSQR.

A. Shift Invariant			
Method	Construction Time (sec)	Computation Time (sec)	Total Time (sec)
`psfMatrix`	0.1254	21.477	21.6024
Proposed	16.418	3.4811	19.8991
B. Shift Variant (block size 16×16)			
`psfMatrix`	3.5319	>1000	>1000
Proposed	60.092	5.5031	65.5955



(a) Shift varying blurred (b) Reconstructed using LSQR

Fig. 3. Image reconstruction of shift varying blur using proposed method with LSQR. The average time for LSQR to converge is 60 seconds (gray scale), and 180 seconds (color).

3.5. Discussion

An immediate consequence of the results of this paper is a way of constructing an efficient convolution operator. It can be deduced from the above experiments that constructing an explicit convolution matrix is the best choice for small to medium sized problems because the forward computing time is the shortest. The upper limit of using a convolution operator is bounded by the total number of non-zero entries of the matrix, which is $MNPQ$, and the memory of the machine. If $MNPQ$ exceeds the limit, then one has to use either an object based method if the number of PSF is small, or a for-loop method if the number of PSF is large (e.g., motion blur). This observation is illustrated in Fig. 4.

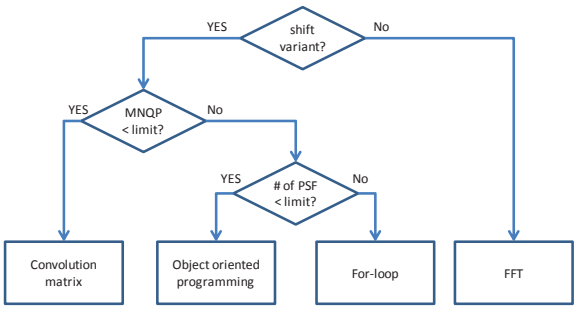


Fig. 4. Hierarchy of constructing a convolution operator

4. CONCLUSION

We propose an efficient algorithm to construct a sparse convolution matrix for shift variant blur. Experimental results show that for large number of shift varying PSFs, the proposed method is the only method that provides a solution. For small number of shift varying PSFs, or a shift invariant PSF, conventional object based methods perform slightly better. When used in standard image restoration algorithms, the proposed method shows huge speed improvement.

5. ACKNOWLEDGMENT

The author thanks James G. Nagy and Truong Q. Nguyen for their feedback and proof reading of this paper.

6. REFERENCES

[1] C. R. Vogel, *Computational Methods for Inverse Problems*, SIAM, Philadelphia, PA, 2002, MATLAB code available at `www.math.montana.edu/~vogel/Book`.

[2] Byunggyoo Kim, *Numerical Optimization Methods for Image Restoration*, Ph.D. thesis, Stanford University, Dec 2002.

[3] Christopher C. Paige and Michael A. Saunders, "Lsqr: An algorithm for sparse linear equations and sparse least squares," *ACM Transactions on Mathematmal Software*, vol. Vol 8, no. 1, pp. 43–71, March 1982.

[4] Per Christian Hansen, James G. Nagy, and Dianne P. O' Leary, *Deblurring Images: Matrices, Spectra, and Filtering*, Fundamentals of Algorithms 3. SIAM, 2006, MATLAB code available at `http://www2.imm.dtu.dk/~pch/HNO`.

[5] Antonin Chambolle, "An algorithm for total variation minimization and applications," *Journal of Mathematical Imaging and Vision*, vol. 20, no. 1-2, pp. 89–97, 2004.

[6] Katrina Palmer Lee, James G. Nagy, and Lisa Perrone, "Iterative methods for image restoration: A matlab object oriented approach," Tech. Rep., Emory University, 2002, available at `http://www.mathcs.emory.edu/~nagy/RestoreTools/index.html`.