Fast Algorithms for Digital Signal Processing

Richard E. Blahut

Fast Algorithms for Digital Signal Processing

Richard E. Blahut

ADDISON-WESLEY PUBLISHING COMPANY
Reading, Massachusetts • Menlo Park, California
Wokingham, Berkshire • Amsterdam • Don Mills, Ontario • Sydney

Richard E. Blahut
IBM Corporation
Owego, New York

Library of Congress Cataloging in Publication Data

Blahut, Richard E.
Fast algorithms for digital signal processing.

Includes bibliographical references and index.
1. Signal processing--Digital techniques.
2. Algorithms. 1. Title
TK5102.5.B535 1984 621.38'043 83-25702
ISBN 0-201-10155-6

ABCDEFGHIJ-MA-8987654

Preface

Digital signal processing is now enjoying an explosive growth. Applications are found everywhere, including radar, sonar, seismic processing, communications, radio astronomy, and medical electronics. Digital signal processors—digital computers specializing in signal processing—are in development and on the market. All of this growth creates demand for even more digital signal processing—massive amounts in some applications.

One way to satisfy part of the demand is to choose cleverly designed algorithms. Instead of expanding the processor performance from one million multiplications per second to five million multiplications per second for some new task, a way might be found to organize the computations of the new task so that one million multiplications per second suffice. There is now a large body of theory available that can be used to attack a problem in this way. This theory is well understood by the theoreticians of the field, but the day-to-day design engineers often ignore it because it is not yet organized into an integrated course of study. It requires considerable familiarity on the part of the designer to select the best algorithm for an application from the bewildering assortment of algorithms that are now known for fast convolution and for the fast Fourier transform.

This book is the product of a course entitled "Fast Algorithms for Digital Signal Processing" that the author has taught both at Cornell University and for IBM. The course was designed for electrical engineers in their senior year or first year of graduate study. The goal of the course was to produce engineers who can comfortably apply the techniques. This is the first goal of the book.

The second goal is to provide a broad view of the state of development of the field of fast signal processing algorithms that can stimulate fresh

developments in the future. By integrating all the threads, many things become much more evident. For example, by posing the treatment of the Cooley-Tukey, Good-Thomas, and Winograd fast Fourier transforms in an arbitrary field, relationships between many later ideas are easy to understand.

I believe that it is important to distinguish between a fast algorithm, the function it computes, and the application in which it is used. These are distinct elements and, when they are allowed to run together, can lose their clarity. Therefore I insist on the distinction between a discrete Fourier transform and a fast Fourier transform. The first is a transform, and the second is an algorithm for computing the first. Likewise, the Viterbi algorithm is *not* a maximum-likelihood sequence estimator; it is an algorithm for computing a minimum distance path through a trellis, a path that may be a maximum-likelihood path for some application but need not be. Even then, the definition of the maximum likelihood should not be confused with an algorithm for computing its path. In keeping with this philosophy, throughout the book there is little discussion of applications. A computational task is stated, and then full attention is given to the problem of finding a good computational algorithm. Discussions of applications of digital signal processing must be found elsewhere.

The idea for this book came while I was writing my earlier book, *Theory and Practice of Error-Control Codes*. Many parts of that book dealt with fast algorithms for computation in a finite field, but the algorithms actually did not depend on the field. I felt that it would be worth the effort to put the algorithms into one book, divorced from their application and presented in a broader setting that includes the many other algorithms of importance in digital signal processing. The treatment touches on many areas of computer science and the theory of algorithms. However, the emphasis is on the engineering goals of finding the best algorithms for signal processing. Asymptotic analyses are of secondary interest.

This book uses branches of mathematics that the typical reader with an engineering education will not know. Therefore all of these mathematics topics are developed here, and all theorems are rigorously proved. I feel that if the subject is to mature and stand on its own, the necessary mathematics must be a part of such a book; appeal to a distant authority will not do. Engineers cannot confidently advance through the subject if they are frequently asked to accept an assertion or to visit their mathematics library. The mathematics needed for the study of fast algorithms is contained in Chapters 2 and 5. These chapters can be read quickly at first but returned to frequently as needed.

One shortcoming that some readers will see in the book is the lack of commentary on practical applications of the algorithms. Issues such as wordlength requirements, roundoff error, and running time of algorithm A on computer B are treated hardly at all. This is a conscious decision because I am not wise enough to make broad statements along these lines. The few

studies I have seen of such issues always treat narrowly defined problems, and I distrust any general conclusions based on the available data. I feel that the designer needs to assess these items in the context of the problem and should study the literature directly to see how other designers fared.

Of the algorithms discussed in this book, many are already important in practice. Others will become important in the future as the applications of digital signal processing become more diverse and more massive. Other methods, perhaps, will never be useful. The intent of this book is to provide a broad survey. It is the design engineers of the next several decades who will decide which of the techniques are important.

The heart of the book is in the cyclic convolution algorithms of Chapters 3 and 7 and the Fourier transform algorithms of Chapters 4 and 8. Chapters 7 and 8 are the multidimensional continuations of Chapters 3 and 4, respectively, and can be read immediately thereafter if desired. The study of one-dimensional convolutions and Fourier transforms is only completed in the context of the multidimensional problems. Chapters 2 and 5 are mathematical interludes; some readers may prefer to treat them as appendices, consulting them only as needed. Chapters 6 and 9 can be read individually following Chapters 3 and 4. The remainder, Chapters 10, 11, and 12, are in large part independent of the rest of the book. Each can be read independently with little difficulty.

Acknowledgments

My major debt in writing this book is to Shmuel Winograd. Without his many contributions to the subject the book would be shapeless and much shorter. He was also generous with his time in clarifying many points to me, and in reviewing early drafts. The papers of Winograd and also the book of Nussbaumer were a source for much of the material discussed in this book.

The book could not have reached its present level without being tested and critiqued repeatedly. I am particularly indebted to Professor B. L. Dickinson for several very detailed reviews which greatly contributed to the quality of the book. I am also indebted to Professor Toby Berger, Professor C. S. Burrus, Professor J. Gibson, Professor J. G. Proakis, Professor T. W. Parks, Dr. B. Rice, Professor Y. Sugiyama, Dr. W. Vanderkulk and Professor G. Verghese for their gracious criticisms. The book could not have been written without the support that was given by the International Business Machines Corporation. I am deeply grateful to IBM for this support and also to Cornell University for giving me the opportunity to teach from the preliminary manuscript. And the most important support came from Barbara, my wife, without whose help the book would not have been written.

Contents

1. Introduction 1

1.1 Introduction to Fast Algorithms 1
1.2 Applications of Fast Algorithms 6
1.3 Number Systems for Computation 8
1.4 Digital Signal Processing 9
1.5 History of Fast Signal Processing Algorithms 17
Problems 18
Notes 19

2. Introduction to Abstract Algebra 20

2.1 Groups 21
2.2 Rings 25
2.3 Fields 29
2.4 Vector Spaces 33
2.5 Matrix Algebra 36
2.6 The Integer Ring 43
2.7 Polynomial Rings 47
2.8 The Chinese Remainder Theorems 56
Problems 62
Notes 64

3. Fast Algorithms for Short Convolutions **65**

3.1 Cyclic Convolution and Linear Convolution 66
3.2 The Cook-Toom Algorithm 68
3.3 Winograd Short Convolution Algorithms 76
3.4 Design of Short Linear Convolution Algorithms 84
3.5 Polynomial Products Modulo a Polynomial 89
3.6 Design of Short Cyclic Convolution Algorithms 91
3.7 Convolution in General Fields and Rings 97
3.8 Complexity of Convolution Algorithms 99
Problems 109
Notes 113

4. Fast Algorithms for the Discrete Fourier Transform **114**

4.1 The Cooley-Tukey Fast Fourier Transform 115
4.2 Small-Radix Cooley-Tukey Algorithms 118
4.3 The Good-Thomas Fast Fourier Transform 127
4.4 The Goertzel Algorithm 131
4.5 Fourier Transforms Computed by Using Convolutions 133
4.6 The Winograd Small Fast Fourier Transform 142
Problems 152
Notes 153

5. Number Theory and Algebraic Field Theory **155**

5.1 Elementary Number Theory 155
5.2 Finite Fields Based on the Integer Ring 162
5.3 Fields Based on Polynomial Rings 165
5.4 Minimal Polynomials and Conjugates 167
5.5 Cyclotomic Polynomials 169
5.6 Primitive Elements 172
Problems 174
Notes 175

6. Computation in Surrogate Fields **176**

6.1 Convolution in Surrogate Fields 177
6.2 Fermat Number Transforms 180
6.3 Mersenne Number Transforms 182
6.4 Convolution Algorithms for Finite Fields 185
6.5 Complex Convolution in Surrogate Fields 188
6.6 Integer Ring Transforms 193
6.7 Chevillat Number Transforms 196

6.8 The Preparata-Sarwate Algorithm 196
Problems 200
Notes 201

7. Fast Algorithms and Multidimensional Convolutions 202

7.1 Nested Convolution Algorithms 203
7.2 The Agarwal-Cooley Convolution Algorithm 208
7.3 Splitting Algorithms 215
7.4 Iterated Algorithms 219
7.5 Polynomial Representation of Extension Fields 225
7.6 Convolution with Polynomial Extension Fields 228
7.7 The Nussbaumer Polynomial Transforms 230
7.8 Fast Convolution of Polynomials 233
Problems 237
Notes 239

8. Fast Algorithms and Multidimensional Transforms 240

8.1 Small-Radix Cooley-Tukey Algorithms 241
8.2 Nested Transform Algorithms 246
8.3 The Winograd Large Fast Fourier Transform 251
8.4 The Johnson-Burrus Fast Fourier Transform 258
8.5 Splitting Algorithms 261
8.6 An Improved Winograd Fast Fourier Transform 266
8.7 The Nussbaumer-Quandalle Permutation Algorithm 268
Problems 280
Notes 282

9. Architecture of Filters and Transforms 283

9.1 Convolution by Sections 284
9.2 Algorithms for Short Filter Sections 289
9.3 Iterated Filter Sections 292
9.4 Symmetric and Skew-Symmetric Filters 297
9.5 Decimating and Interpolating Filters 303
9.6 Autocorrelation and Crosscorrelation 306
9.7 Construction of Transform Computers 313
9.8 Limited-Range Fourier Transforms 318
Problems 319
Notes 320

10. Fast Algorithms Based on Doubling Strategies 322

10.1 Halving and Doubling Strategies 323
10.2 Data Structures 327
10.3 Fast Algorithms for Sorting 328
10.4 A Recursive Radix-Two Fast Fourier Transform 330
10.5 Fast Transposition 333
10.6 Matrix Multiplication 336
10.7 A Recursive Euclidean Algorithm 339
10.8 Computation of Trigonometric Functions 344
Problems 350
Notes 351

11. Fast Algorithms for Solving Toeplitz Systems 352

11.1 The Levinson and Durbin Algorithms 353
11.2 The Trench Algorithm 362
11.3 The Berlekamp-Massey Algorithm 368
11.4 A Recursive Berlekamp-Massey Algorithm 376
11.5 Methods Based on the Euclidean Algorithm 379
Problems 385
Notes 386

12. Fast Algorithms for Trellis and Tree Search 387

12.1 Trellis and Tree Searching 388
12.2 The Viterbi Algorithm 392
12.3 The Stack Algorithm 395
12.4 The Fano Algorithm 399
Problems 404
Notes 405

Appendix A. A Collection of Cyclic Convolution Algorithms 407

Appendix B. A Collection of Winograd Small FFT Algorithms 418

References 427

Index 435

CHAPTER 1

Introduction

Algorithms for computation are found everywhere, and efficient versions of these algorithms are highly valued by those who use them. We are mainly concerned with certain types of computations, namely, those related to digital signal processing, including tasks such as digital filters, discrete Fourier transforms, correlation, and spectral analysis. Our purpose is to present the modern techniques for digital implementation of these computations. We are not concerned with the design of the tap weights of a digital filter; our concern is only with the computational organization of its implementation. Nor are we concerned with why one should want, for example, a discrete Fourier transform; our concern is only with how it can be computed efficiently. Surprisingly, there is an extensive body of theory dealing with this highly specialized topic.

1.1 INTRODUCTION TO FAST ALGORITHMS

An algorithm, like most other engineering devices, can be described either by an input/output relationship or by a detailed explanation of its internal construction. When one applies the techniques of digital signal processing to a new problem, one is concerned only with the input/output aspects of the algorithm. Given a signal, or a data record of some kind, one is concerned with what should be done to this data, that is, with what the output of the algorithm should be when such and such a data record is the input. Perhaps the output is a filtered version of the input or its Fourier transform. These input/output relationships for an algorithm can be expressed

mathematically without prescribing in detail all of the steps by which the calculation is to be performed.

Devising such a good algorithm for an information processing problem, from this input/output point of view, may be a formidable and sophisticated task, but this is not our concern in this book. We will assume that we are given an input/output algorithm described in terms of filters, Fourier transforms, interpolations, decimations, correlations, modulations, histograms, matrix operations, and so forth. All of these can be expressed with mathematical formulas and so can be computed just as written. This will be called the obvious implementation.

One may be content with the obvious implementation; for many years most were content, and even today some are still content. But once people began to compute such things, other people began to look for more efficient ways to compute them. This is the story we aim to tell, the story of fast algorithms. By a fast algorithm, we mean a detailed description of a computational procedure that is not the obvious way to compute the required output from the input. A fast algorithm usually gives up a conceptually clear computation in favor of one that is computationally efficient.

Suppose we need to compute a number A given by

$$A = ac + ad + bc + bd$$

As written, this requires four multiplications and three additions to compute. If we need to compute A many times with different sets of data, we will quickly notice that

$$A = (a + b)(c + d)$$

is an equivalent form that requires only one multiplication and two additions, and so is to be preferred. This simple example is quite obvious but really illustrates most of what we shall talk about. Everything we do can be thought of in terms of the clever insertion of parentheses in a computational problem. But in a big problem the fast algorithms cannot be found by inspection. It will require a considerable amount of theory to find them.

A nontrivial yet simple example of a fast algorithm is complex multiplication. The complex product

$$(e + jf) = (a + jb) \cdot (c + jd)$$

can be written in terms of real multiplications and real additions

$$e = (ac - bd)$$

$$f = (ad + bc)$$

We see that these formulas require four real multiplications and two real additions. A more efficient "algorithm" is

$$e = (a - b)d + a(c - d)$$

$$f = (a - b)d + b(c + d)$$

whenever multiplication is harder than addition. This form requires three real multiplications and five real additions. If c and d are constants for a series of complex multiplications, then the terms $c + d$ and $c - d$ are constants also and can be computed off-line. It then requires three real multiplications and three real additions to do one complex multiplication.

We have traded one multiplication for an addition. This can be a worthwhile savings, but only if the signal processor is designed to take advantage of it. Some signal processors have been designed with a prejudice for a complex multiplication that uses four multiplications. Then the advantage of the improved algorithm is wasted.

We can dwell further on this example as a foretaste of things to come. The complex multiplication above can be rewritten as a matrix product

$$\begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} c & -d \\ d & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

where the vector $\begin{bmatrix} a \\ b \end{bmatrix}$ represents the complex number $a + jb$, the matrix $\begin{bmatrix} c & -d \\ d & c \end{bmatrix}$ represents the complex number $c + jd$, and the vector $\begin{bmatrix} e \\ f \end{bmatrix}$ represents the complex number $e + jf$. The matrix-vector product is a way to represent complex multiplication.

The alternative algorithm can be written in matrix form as

$$\begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} (c-d) & 0 & 0 \\ 0 & (c+d) & 0 \\ 0 & 0 & d \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

The algorithm can be thought of as nothing more than the unusual matrix factorization:

$$\begin{bmatrix} c & -d \\ d & c \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} (c-d) & 0 & 0 \\ 0 & (c+d) & 0 \\ 0 & 0 & d \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & -1 \end{bmatrix}$$

We can abbreviate the algorithm as

$$\begin{bmatrix} e \\ f \end{bmatrix} = \mathbf{BDA} \begin{bmatrix} a \\ b \end{bmatrix}$$

where **A** is a 3 by 2 matrix that we call a matrix of preadditions; **D** is a 3 by 3 diagonal matrix that is responsible for all of the general multiplications; and **B** is a 2 by 3 matrix that we call a matrix of postadditions.

We shall find that many of the best computational procedures for convolution and for the discrete Fourier transform can be put into this factored form of a diagonal matrix in the center on each side of which is a matrix whose elements are 1, 0, and -1. These fast algorithms will have the structure of a batch of additions followed by a batch of multiplications followed by another batch of additions.

The final example of this introductory section is a fast algorithm for multiplying matrices. Let

$$\mathbf{C} = \mathbf{AB}$$

where **A** and **B** are l by n and n by m matrices, respectively. The standard method for computing the matrix **C** is

$$c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj} \qquad \begin{matrix} i = 1, \ldots, l \\ j = 1, \ldots, m \end{matrix}$$

which requires nlm multiplications and $(n - 1)lm$ additions as it is written. We shall give an algorithm that reduces the number of multiplications by almost a factor of two but increases the number of additions. The total number of operations increases slightly.

We use the identity

$$a_1b_1 + a_2b_2 = (a_1 + b_2)(a_2 + b_1) - a_1a_2 - b_1b_2$$

on the elements of **A** and **B**. Suppose that n is even (otherwise append a column of zeros to **A** and a row of zeros to **B**, which does not change the product **C**). Apply the above identity to pairs of columns of **A** and pairs of rows of **B** to write

$$c_{ij} = \sum_{k=1}^{n/2} (a_{i,2k-1} + b_{2k,j})(a_{i,2k} + b_{2k-1,j}) - \sum_{k=1}^{n/2} a_{i,2k-1}a_{i,2k} - \sum_{k=1}^{n/2} b_{2k-1,j}b_{2k,j} \qquad \begin{matrix} i = 1, \ldots, l \\ j = 1, \ldots, n \end{matrix}$$

The computational savings results because the second term depends only on i and need not be recomputed for each j and the third term depends only on j and need not be recomputed for each i. The total number of multiplications used to compute matrix **C** is $\frac{1}{2}nlm + \frac{1}{2}n(l + m)$, and the total number of additions is $\frac{3}{2}nlm + lm + (\frac{1}{2}n - 1)(l + m)$. For large matrices the number of multiplications is about half of the direct method.

This last example may be a good place for a word of caution about numerical accuracy. Although the number of multiplications is reduced, this algorithm is more sensitive to roundoff error unless it is used with care. By proper scaling of intermediate steps, however, one can obtain computational accuracy that is nearly the same as the direct method. Consideration of computational noise is always a practical factor in judging a fast algorithm, although we shall usually ignore it. Sometimes when the number of operations is reduced, the computational noise is reduced because there are fewer sources of noise. In other algorithms, though there are fewer sources

of computational noise, the answer may be very sensitive to one or more of them, and so the computational noise in the answer is increased.

Most of this book will be spent studying only a few problems: the problems of linear convolution, cyclic convolution, multidimensional linear and cyclic convolution, the discrete Fourier transform, multidimensional discrete Fourier transforms, the solution of Toeplitz systems, and finding paths in a trellis. Some of the techniques we shall study deserve to be more widely used; multidimensional Fourier transform algorithms can be especially good if one takes the pains to understand the most efficient ones. For example, Fig. 1.1 compares some methods of computing a two-dimensional Fourier transform. The improvements in performance come more slowly toward the end of the list. It may not seem very important to reduce the number of multiplications per output cell from six to four after one has already gone from 40 to six, but this can be a shortsighted view. It is an additional savings and may well be worth the design time in a big system.

There is another important lesson contained in Fig. 1.1. An entry, labeled the hybrid Cooley-Tukey/Winograd FFT, can be designed to compute a 1000- by 1000-point two-dimensional Fourier transform with 40 real multiplications per grid point. This example may help to dispel an unfortunate myth that the discrete Fourier transform is practical only if the blocklength is a power of two. In fact, the signal processor need not insist that the customer can only have such a blocklength; good algorithms are available for many values of the blocklength.

Algorithm	*Multiplications/Pixel**	*Additions/Pixel*
Direct Computation of Discrete Fourier Transform 1000×1000	8,000	4,000
Basic Cooley-Tukey/FFT 1024×1024	40	60
Hybrid Cooley-Tukey/ Winograd FFT 1000×1000	40	72.8
Winograd FFT 1008×1008	6.2	91.6
Nussbaumer-Quandalle FFT 1008×1008	4.1	79

*1 Pixel = 1 output grid point

FIGURE 1.1 Relative performance of some two-dimensional Fourier transform algorithms.

1.2 APPLICATIONS OF FAST ALGORITHMS

Very large scale integrated circuits called chips are now available. A chip can contain on the order of 100,000 logical gates, and it is not surprising that the theory of algorithms is looked to as a way of efficiently organizing these gates. Sometimes a considerable performance improvement can be realized by choice of algorithm. Of course, a performance improvement can also be realized by increasing the size of the chip or its speed. These latter kinds of improvements are more widely understood.

For example, suppose one devises an algorithm for a Fourier transform that has only one-fifth the computation of another Fourier transform algorithm. By using the new algorithm, one might realize an improvement that can be as real as if one increased the speed or the size of the chip by a factor of five. To realize this improvement, however, the chip designer must reflect the architecture of the algorithm in the architecture of the chip. A naive design can dissipate the advantages by increasing the complexity of indexing, for example, or of input/output flow. An understanding of the fast algorithms described in this book will be required to obtain the best system designs in the era of very large scale integrated circuits.

At first glance, it might appear that the two kinds of development—fast circuits and fast algorithms—are in competition. If one can build the chip big enough or fast enough, then it seemingly does not matter if one uses inefficient algorithms. No doubt this view is sound in some cases, but in other cases one can also make exactly the opposite argument. Large digital signal processors often create a need for fast algorithms. This is because one begins to deal with signal processing problems that are much larger than before. Whether competing algorithms for some problem have running times proportional to n^2 or n^3 may be of minor importance when n equals 3 or 4; but when n equals 1000, it becomes critical.

The fast algorithms we shall develop are concerned with digital signal processing, and the applications of the algorithms are as broad as the application of digital signal processing itself. Now that it is practical to build a sophisticated algorithm for digital signal processing onto a chip, we would like to be able to choose such an algorithm to maximize the performance of the chip. But to do this for a large chip involves a considerable amount of theory. In its totality the theory goes well beyond the material that will be discussed in this book. Advanced topics in logic design and computer architecture such as parallelism and pipelining must also be studied before one can determine all aspects of complexity.

We usually measure the performance of an algorithm by the number of multiplications and additions it uses. These measures are about as deep as one can go at the level of the computational algorithm. At a lower level, we would wish to know the area of the chip or the number of gates on it and the time required to complete a computation. Often one judges a circuit by the

area-time product. We will make no effort to give performance measures at this level because this is beyond the province of the algorithm designer.

The significance of the topics in this book cannot be appreciated without understanding the massive needs of digital processing applications of the future. It is not possible even to guess at the size of such systems. Suffice it to say that, at the present time, applications are easy to foresee that require orders of magnitude more digital signal processing than current technology can satisfy.

Over the last decade, sonar systems have become almost completely digital. Though they process only a few kilohertz of signal bandwidth, these systems can use tens of millions or even hundreds of millions of multiplications per second and even more additions. Extensive racks of digital equipment are needed for such systems, and yet reasons for even more processing are routinely conceived.

Radar systems also are becoming digital, but many of the front-end functions are still done by conventional microwave or analog circuitry. One needs to notice only that, in principle, radar and sonar are quite similar, but that radar has 1000 or more times as much bandwidth, to see the enormous potential for more digital signal processing in radar systems.

Seismic processing provides our principal method for exploration deep below the earth's surface. This is an important method of searching for petroleum reserves. Many computers are already busy full time processing the large stacks of data tapes, but there is no end to the computations remaining.

Computerized tomography is now widely used to synthetically form images of internal organs of the human body by using X-ray data from multiple projections. Algorithms are under study that will reduce considerably the X-ray dosage, but the signal processing requirements are far beyond anything that is practical today. Other forms of medical imaging are under study, such as those using ultrasonic data, nuclear magnetic resonance data, or particle decay data, that also use digital signal processing.

Nondestructive testing of manufactured articles such as castings is possible by means of computer-generated internal images based on the response to induced vibrations.

It is also possible in principle to enhance poor-quality photographs. Pictures blurred by camera motion or out-of-focus pictures can be corrected by signal processing. However, to do this digitally takes large amounts of signal processing computations.

Satellite photographs can be processed digitally to merge several images or to enhance features, or to combine information received on different wavelengths, or to create stereoscopic images synthetically. For example, for meteorological research, one can create a moving three-dimensional image of the cloud patterns moving above the earth's surface based on a sequence of satellite photographs from several aspects.