Contour-Based Shape Representation for Image Compression and Analysis

Ciro D'Elia[1] and Giuseppe Scarpa[2]

[1] University of Cassino,
Department of Automation Electromagnetism Information Eng.
and Industrial Mathematics DAEIMI, (FR) Italy
delia@unicas.it
http://webuser.unicas.it/delia
[2] University Federico II of Naples,
Dept of Electronic and Telecommunication Eng., via Claudio 21, (NA) Italy

Abstract. With the rapid growth of computing power, many concepts and tools of image analysis are becoming more and more popular in other data processing fields, such as image and video compression. Image segmentation, in particular, has a central role in the object-based video coding standard MPEG-4, as well as in various region-based coding schemes used for remote-sensing imagery. A region-based image description, however, is only useful if it has a limited representation cost, which calls for accurate and efficient tools for the description of region boundaries.
A very promising approach relies on the *extended boundary* concept, first discussed in [6] and [7] and later used by Liow [5] to develop a contour tracing algorithm. In this work, we extend Liow's algorithm and introduce the corresponding reconstruction technique needed for coding purposes. In addition, we define an algebraic semi-group structure that allows us to formally prove the algorithm properties, to extend it to other boundary definitions, and to introduce a fast contour tracing algorithm which only requires a raster scan of the image.

1 Introduction

Thanks to the increasing availability of computing resources, segmentation-based compression algorithms, in which the classical transform coding scheme is preceded by a segmentation preprocessing, are becoming more and more popular. A well-know example is the MPEG-4 video coding standard [8] [9] [10], but segmentation-based compression is of interest for many other applications [4]. This approach guarantees two main results: on one hand, segmentation divides the image in statistically homogeneous regions, so that image encoding can be optimized locally for each region, with excellent encoding performance; on the other hand, the end user is provided with an high-quality segmentation, obtained on the original data, and embedded at no cost in the output stream. This by-product can be extremely useful in many applicative fields, *e.g.*, remote-sensing and biomedicine, where such an "high level" image descriptions can be used

I. Nyström et al. (Eds.): DGCI 2003, LNCS 2886, pp. 204–213, 2003.

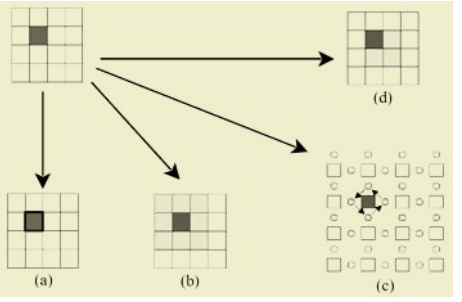

Fig. 1. Boundary Definitions: (a) Boundary in $\mathcal{R}^2$, (b) Trivial application of $\mathcal{R}^2$ definition to the space discrete space $\mathcal{N}^2$, (c) Boundary lattice, (d) Extended Boundary.

for subsequent automated analysis, such as classification, object detection, automatic diagnosis, and data mining.

Obviously, the quality of the segmentation map is very important for this scheme to succeed. It should be faithful to the image features, to enable an efficient coding of the image regions but should also present smooth region boundaries, to limit the encoding cost of the map itself. Once obtained a smooth map, a promising way to encode it is to extract the map contours and represent them efficiently. To this aim it is necessary to choose a boundary definition that allows for:

1. an efficient contour representation, *i.e.* a contour representation that is intrinsically suited for coding applications;
2. a memory efficient algorithm, considering that in some applications, like remote sensing, images of 8000x8000 pixels and more are commonplace;
3. a simple contour tracing algorithm;
4. a lossless (namely, perfect) image reconstruction.

In figure 1 several definitions of boundaries are illustrated: in fig.1.a there is the well known topological definition of boundary in the continuous ($\mathcal{R}^2$), where a point x in a closed set $\mathbf{R}$ of $\mathcal{R}^2$ belongs to the boundary ($\partial\mathbf{R}$)[1] if and only if in each neighborhood of x there is at least an external point of $\mathbf{R}$. However, since our segmentation map is defined on the discrete set ($\mathcal{N}^2$) we have to extend this definition. In fig.1.b there is the application of the previous definition to the discrete case $\mathcal{N}^2$ with the neighborhood of fig.2. It is clear that this definition is unable to provide a common boundary between two regions, namely, the boundary of region $\mathbf{R}$ is different from the boundary of the complementary region $\overline{\mathbf{R}}$, while in $\mathcal{R}^2$ $\partial\mathbf{R} \equiv \partial\overline{\mathbf{R}}$. Therefore, this extension to discrete sets is not suitable for coding purposes, because $\partial\mathbf{R}$ and $\partial\overline{\mathbf{R}}$ should be both encoded although they carry the same information. Furthermore, this definition contradicts the physical meaning of boundary and makes it difficult to carry out a geometrical analysis of the image. The definition of fig.1.c, where a Boolean lattice of boundaries

[1] Where $\partial\mathbf{R}$ is the boundary of $\mathbf{R}$.

between pixels is introduced, preserves the common boundary between regions but requires this supplementary lattice, which could not be memory efficient and, contrarily to the definition in the continuous[2], $\partial\mathbf{R}$ is not included in $\mathcal{N}^2$. Finally the definition of fig.1.d, based on the concept of *extended boundary* of the region $\mathbf{R}$, preserves both the common boundary between regions and the desirable property that $\partial\mathbf{R}$ belongs to $\mathcal{N}^2$.

3	2	1
4	P	0
5	6	7

Fig. 2. Neighbors of point P and Freeman codes.

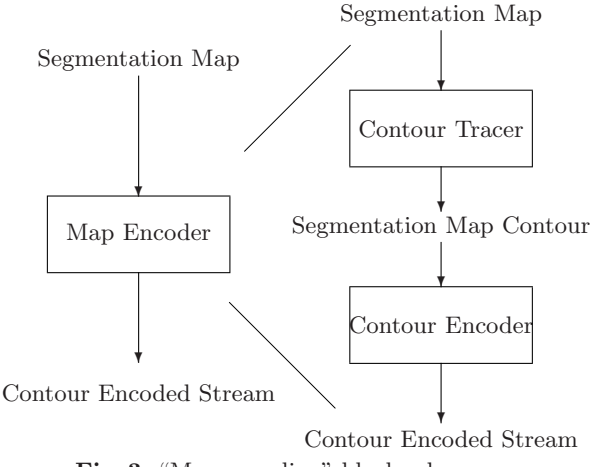

Fig. 3. "Map encoding" block scheme.

For these reasons, to develop our "Map encoding" algorithm (see fig.3), we have chosen the *extended boundary* definition which meets our major requirement about the efficiency of the contour representation. Of course, we have yet to shown how this boundary definition leads to a simple contour tracing algorithm and, above all, how the original segmentation map can be recovered, without loss of information, starting from the contours. In other words, we must prove that the extended boundary definition can be used to represent the map, and that the contours have the same information content of the map itself. In Section 2 we will show how to trace the image contours using the previous definition, while in the Section 3 we will show how to reconstruct the Image starting from the Contours or the Shapes.

2 Contour and Shape Tracing

In the following, we will show how to trace the contours using the extended boundary concept, how to organize them in shapes, for a multi label image

[2] Where $\partial\mathbf{R} \subseteq \mathcal{R}^2$.

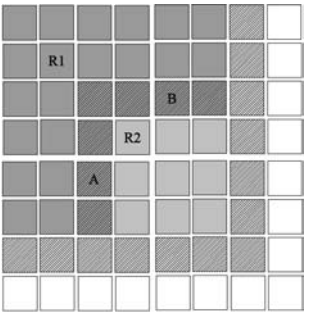

Fig. 4. In this sample there are two regions dark gray R_1 and light gray R_2 together with their extended boundary, depicted with ///. We can observe that the common boundary between regions R_1 and R_2 is preserved, indeed for instance, Pixel (A) belong either to $S_A(\mathbf{R_2})$ and $S_C(\mathbf{R_1})$, while Pixel (B) belong either to $S_A(\mathbf{R_2})$ and $S_B(\mathbf{R_1})$.

like our segmentation map, and how to reconstruct the image starting from the shapes or contours.

First of all, we need a formal definition of *extended boundary*. Let $P_i(P)$ be the i-th 8-connected neighbor of pixel P, with i given by the code of fig.2, and let the following definitions hold:

1) $\text{LEFT}(\mathbf{R}) \equiv \{P : P \in \mathbf{R} \text{ and } P_4(P) \notin \mathbf{R}\}$;
2) $\text{UPPER}(\mathbf{R}) \equiv \{P : P \in \mathbf{R} \text{ and } P_2(P) \notin \mathbf{R}\}$;
3) $\text{RIGHT}(\mathbf{R}) \equiv \{P : P \in \mathbf{R} \text{ and } P_0(P) \notin \mathbf{R}\}$;
4) $\text{LOWER}(\mathbf{R}) \equiv \{P : P \in \mathbf{R} \text{ and } P_6(P) \notin \mathbf{R}\}$;

Then, the extended boundary of region $\mathbf{R}$ is the union of the sets:

$S_A(\mathbf{R}) = \{P : P \in \text{LEFT}(\mathbf{R}) \text{ or } P \in \text{UPPER}(\mathbf{R})\}$;
$S_B(\mathbf{R}) = \{P_6(P) : P \in \text{LOWER}(\mathbf{R}) \text{ or } P \in \text{LEFT}(\mathbf{R})\}$;
$S_C(\mathbf{R}) = \{P_0(P), P_7(P) : P \in \text{RIGHT}(\mathbf{R})\}$;

Looking at the definition, we can observe that $S_A(\mathbf{R})$ contains boundary points that are all *internal* to $\mathbf{R}$, while $S_B(\mathbf{R})$ and $S_C(\mathbf{R})$ add essentially *external* points[3]. Indeed for example, $S_B(\mathbf{R})$ is composed by points that are "under" internal points of $\mathbf{R}$. More important, note that points in $S_B(\mathbf{R})$ and $S_C(\mathbf{R})$, that are *external* boundary points for $\mathbf{R}$, are *internal* points for $\overline{\mathbf{R}}$, and in fact belong to the set $S_A(\overline{\mathbf{R}})$. That is the reason why this boundary definition preserves the common boundary between regions (a sample is provided in fig.4).

2.1 Single Region Contour Tracing

To trace the boundary $\partial\mathbf{R}$ of a region $\mathbf{R}$, it is sufficient to select a starting point $x_0 \in \partial\mathbf{R}$, and to follow the contour step by step using some suitable rules (for instance like in fig.7). Indeed during the contour tracing algorithm the boundary

[3] Note that $S_B(\mathbf{R})$ contains also same points already included in $S_A(\mathbf{R})$.

is described by a sequence of boundaries steps represented with the codes of fig.2 (Freeman Codes). In Section 2.2 we will show how to select the starting point x_0, while here we derive the rules to follow the contours. To this end, we first define the rules for an elementary region, and then show how to combine them in order to trace the contours of more complex regions. In fig.5.a a single-pixel region is shown (in gray), together with its extended boundary, depicted with ///, and its visiting order, indicated by the vector arrows. Fig.5.c shows the same items for a two-pixel region. In fig.5.b, instead, we can see how the boundary and visiting order of the two-pixel region can be obtained by combining the visiting orders of each component pixel. In fact, by considering the visiting order of each pixel and adding (and hence, erasing) overlapping vectors, we obtain exactly the result of figure 5.c. By using this composition rule, it is possible to derive the visiting order of an arbitrary region by adding one pixel at time. Furthermore, the same rule can be used not only to the so called 4-connected boundary of fig.1.d, but also for the 6-connected boundary of fig.1.c, as is evident in figure 6. With reference to this figure, the composition rule can be easily defined in form of an internal addition "+" operation over the set of the contours $\mathbf{C}$[4]. To define this operation in $\mathbf{C}$, for 4- or 6-connected contours, we need to define in turn the *boundaries cliques* as the subsets of the boundary in which each pair of pixels are connected. In fig6.a, for example, one such clique is shown with ///. We can note that for the 6-connected boundary definition the cliques comprise 4 elements, while for the 4-connected elements they comprise just 2 elements.

With the notion of boundary clique it is easy to define the addition of the boundaries of two disjoint regions $\mathbf{R}_1$ and $\mathbf{R}_2$.

Definition 1. *Let $\partial\mathbf{R}_1$ and $\partial\mathbf{R}_2$ be the oriented boundaries of the two generic disjoint*[5] *regions $\mathbf{R}_1$ and $\mathbf{R}_2$, respectively: the operation $(\partial\mathbf{R}_1 + \partial\mathbf{R}_2)$ is defined as the vectorial sum of the boundaries steps in each common boundary clique of $\mathbf{R}_1$ and $\mathbf{R}_2$.*

To gain insight on this simple definition we can apply it to the 6-connected example of fig.6.a, where $\mathbf{R}_1$ and $\mathbf{R}_2$ are two single-pixel regions. We have already observed that $\partial\mathbf{R}_1$ and $\partial\mathbf{R}_2$ have two cliques in common, and for each one of these we have to add vectorially the "boundaries steps" as depicted in fig.6.b. Note that this addition operation allows us to define also the Abelian semi-group $\mathcal{C}(\mathbf{C}, +)$[6], which allows us to derive the contour tracing rules of fig.7 (as presented in [5]) by applying the previous operation to the depicted configurations. In particular, to decide about the next boundary step, it is sufficient to observe the value of the current point P and of $P_2(P), P_3(P)$ and $P_4(P)$ (see fig.2), because only these points can affect the clique for the next boundary step. This is also true for 6-connected boundary definition, and hence a similar look-up table can be derived for this case.

[4] This sets contains the oriented contours of all the subsets of the image.

[5] If $\mathbf{R}_1$ and $\mathbf{R}_2$ are not disjoint we can consider $\mathbf{R}'_1 = \mathbf{R}_1 - \left(\mathbf{R}_1 \bigcap \mathbf{R}_2\right)$ instead of $\mathbf{R}_1$.

[6] It is also interesting to note that the semi-group is generated by the singleton regions. Trivially any region and hence any contour can be obtained by combining single-pixel regions (singleton).

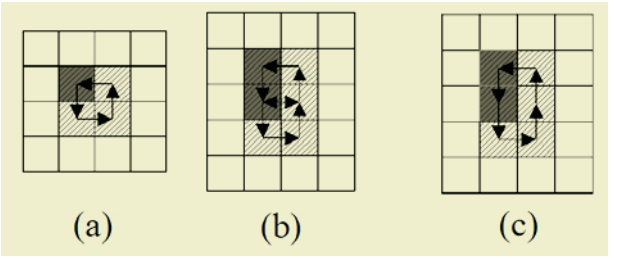

Fig. 5. Contour Tracing Composition Rule.

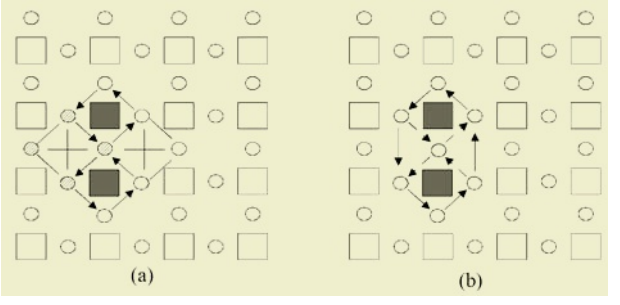

Fig. 6. 6 connected contour definition. (a) boundary cliques, (b) Composition Rule (dashed arrows are summed in continous line arrows).

In conclusion, using the rules of fig.7, we can move step by step along the boundary starting from a generic point; therefore, to implement the contour tracer, we can store this look-up table or else compute the next step using the $\mathcal{C}$ operations on $P, P_2(P), P_3(P), P_4(P)$.

In addition, the properties of $\mathcal{C}(\mathbf{C}, +)$ can be also used to implement the contour tracing of a region $\mathbf{R}$ with a single raster scan. In fact, during the raster scan, it is possible to update the contour of the object adding pixel by pixel. This can be convenient when dealing with a very large image that exceeds the available memory resources, or also to have a more local algorithm on the data to speed up the contour tracing procedure, and finally also to implement a parallel version of the contour tracing algorithm, if needed.

2.2 Multi-label Map Contour Tracing

The last problem to solve, for the contour tracing, is the selection of a suitable starting point x_0 for each connected region in the map. In [5], Liow suggests to select x_0 as the first point, during the raster scan, that meets the the conditions:

a) P has the selected label;
b) $P_2(P) \neq P$ and $P_4(P) \neq P$;

So for each label we have just one starting point, and hence one region traced. In classification applications, however, the same label can be used for a large

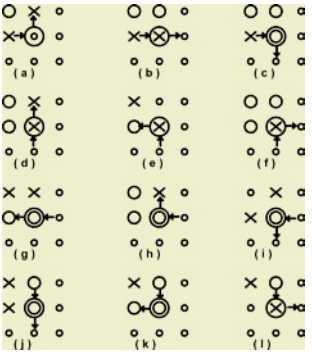

Fig. 7. Contour Tracing Rule: The circle is the traced region, X is any other region, small circle is don't care, while Big Circle is the contour point that is being traced. Observing the figure it is clear that, knowing the incoming direction, the outgoing direction of the current boundary point depends at most on the configuration of P, $P_2(P), P_3(P)$ and $P_4(P)$ because the other are don't care.

number of non-connected regions. By selecting all the points that meet condition (b) we end up with too many starting points, and each region could be traced twice. To solve this problems we can use two solutions, one memory efficient and the other computationally efficient. The first solution detects a starting points using condition (b) and then, while the region is traced, a list of forbidden starting point is filled with the other points of the traced region that also respect condition (b). This solution is memory efficient but, for large images could be quite slow, because once detected a candidate starting point we have to search in a possibly large list if it is forbidden or not. The other solution, that we use, requires a boolean bitmap in which each forbidden point is marked, so that it is straightforward to see whether a candidate is forbidden or not.

Up to now, the presented algorithm detects one starting point per connected region and then, using the rule of fig.7, traces the contours of each region of the map. The map is then represented by a list of region contours (shapes). In this way, however, no information is retained on the region adjacency, which could turn out to be useful both for the map compression and for subsequent applications based on image analysis. Therefore, the implemented algorithm is slightly different: when the region is traced, like in [5], the contour is cut in *chains*, contour segments shared by only two regions, connected by *vertices* (see fig.8), contour points shared by three or more regions. So, after the contour tracing algorithm, the image is represented by a list of vertices and a list of chains[7]. The shape-based image representation can be derived easily by building a list of shapes and grouping, for each shape, all the chains that belong to it. The shape-based representation is quite convenient because only one label per shape must be saved (rather than two labels per chain), and because coding together

[7] As will clear in the paragraph 3, to have a lossless image representation it is also necessary to store the internal and external label for each chain.

statistically homogeneous chains is usually more efficient. In addition, with the organization in shapes, it is quite easy to build the region adjacency graph, to compute geometrical features like the region perimeter and, more in general, any feature obtained as an integral along the shape boundary.

We can, finally, summarize the entire shape extraction algorithm with the following steps:

1) detection of the region starting point;
2) extraction of the contour segments for each region (like in [5]), following the rule of fig.7 (contour image representation);
3) grouping of contour-segments in shapes (shape image representation);
4) building of the region adjacency graph using both contour and shape representation.

3 Image Reconstruction from Shapes

In this section we will describe how to recover a segmentation map from its contour or shape representation without any loss of information.

First of all, we have to prove that it is indeed possible to recover the original map from its extended boundaries. As a matter of fact, by simple inspection of fig.1.d or fig.4 it is clear that the the extended boundary does not contain by itself all the information needed to recover the image, because it is not possible to understand whether a point is internal or external to the region, observing, for instance, the light gray region (R_2) of fig.4 we can argue that there are same boundary pixel that are internal and same other that are external, so it is not possible to reconstruct the boundary label value. This is the main problem to solve, because once the right label has been assigned to the boundary points it is quite simple to reconstruct the whole image by using, for instance, a flood fill procedure.

Even if the extended boundary itself does not contain the required information, by observing fig.7 we can argue that, except for the case of fig.7.a[8] the *oriented* extended boundary tells us directly if a point is internal or not. For instance, in the case of fig.7.b, we can say that the current point is external if we enter and leave both from the left, while in the case of fig.7.c, we can say that it is internal if we enter from the left and leave on the bottom.

Hence, the reconstruction procedure can be summarized as follow

1) reconstruction of the boundaries chain by chain, i.e. painting with original labels the boundaries (wire-frame image);
2) reconstruction of the internal points (original map);

It is worth underlining that we can exploit the extended boundary properties also to implement the reconstruction of internal points. Indeed, we can argue that, moving from left to right and top to bottom of the image, the label change only

[8] This case have to be treated in a special way, anyway it is possible reconstruct the right value also in this case.

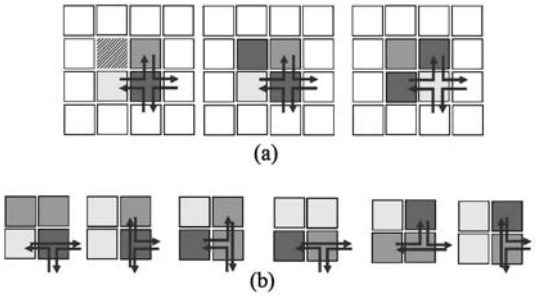

Fig. 8. Vertexes condition (a) Vertexes of 4 regions (b) Vertexes of 3 regions.

on the boundary points, and the internal points on the right of the boundary have the same label of its boundary. This property makes possible to implement step (2) of the reconstruction algorithm during the raster scan, when the internal points are filled with the value of the current brush, and the brush changes value on the boundaries. The procedure described above reconstructs the image from a representation in terms of vertices, chains, and their internal and external labels. To improve the encoding efficiency, of particular importance in compression applications, we can switch to a representation of the image in terms of shapes, so that just one label per shape is needed rather two labels per each chain of the shape. On the other hand, the reconstruction procedure from the shape representation is more complex, because we know only the internal label for the chains of each shape and the missing information (external label) must be recovered in some way. This can be accomplished, however, by observing that the external points are internal to other shapes. There are two possible cases, because an external point of a shape belongs either to an adjacent shape or to a surrounding shape. In the first case, the boundary label is painted with the right value when we reconstruct the adjacent shape, while in the case of a surrounding shape we have to retrieve same way the the its label. This can be accomplished during step (2), using a stack of brushes for each image line instead of a single brush. Indeed in each line, we can build, during the paint step, the stack of the labels of the surrounding regions in a such a way that it is extremely easy to access the needed information.

4 Conclusions

The proposed representation is based on the *extended boundary concept* first discussed in [6] and [7] and then used for a Contour tracing algorithm in [5]. This work extends the contour tracing algorithm proposed in [5] and introduces a reconstruction algorithm needed for coding purpose. Furthermore it has been defined an algebraic semi-group structure that allow the formal demonstration of the algorithm [5], his extension to other definition of boundaries, and the introduction of a raster contour tracing algorithm.

Acknowledgements

The Authors are grateful to Prof. Francesco Tortorella for his precious suggestions and encouragements.

References

1. Giacinto Gelli, Giovanni Poggi: "Compression of Multispectral images by Spectal Classification and Transform Coding", *IEEE Transaction on Image Processing* , volume.8, numero.4, pp.476-489, April 1999.
2. Gelli, G.; Poggi,G.; Ragozini, A.R.P.: "Multispectral-image compression based on tree-structured Markov random field segmentation and transform coding", *Geoscience and Remote Sensing Symposium, 1999. IGARSS '99 Proceedings. IEEE 1999 International*, Volume: 2 , 1999 Page(s): 1167 -1170 vol.2
3. D'Elia, Poggi, Scarpa: "An Adaptive MRF model for boundary preserving segmentation of multispectral images", *Eusipco 2002*, Sept 2002.
4. C.D'elia, G.Poggi, G.Scarpa: "Advances in segmentation and compression of multispectral images", *Proc. IEEE IGARSS 01*, vol.*, pp.*, Sidney, July 2001.
5. Yuh-Tay Liow: "A contour tracng algorithm that preserves common boundaries between regions", *Image Understanding* , volume.3, numero.3, pp.313-321, May 1991.
6. H.F. Feng and T. Pavlidis: "The generation of polynomial outlines of objects from gray level pictures", *IEEE Trans. on Circuit and Systems*, **CAS-22**, pp.427-439, 1975.
7. T. Pavlidis: "Structure Pattern Recongnition", *Springer-Verlang*, Berlin, New York, 1977.
8. A. Puri and T. Chen: "Multimedia Systems, Standards, and Networks", Signal Processing and Communications Series, Marcel Dekker Inc., March 2000.
9. MPEG-4 System Group: "Coding of audio-visual objects: video", ISO/IEC JTC1/SC29/WG11 N2202, March 1998.
10. A.K. Katsaggelos, L.P. Kondi, F.W. Meier, J.Ostermann, and G.M. Schuster: "MPEG-4 and rate-distorsion-based shape-coding techniques", *Proc. IEEE* **86**, pp.1029-1051, 1998.