Chapter 5

Filtering & Segmentation

5.1 Objectives

Our purpose in this chapter is to show that hierarchical region based representations are suitable for filtering and segmentation purposes.

To that end, first, a review on segmentation techniques and connected operators is done. It is discussed that both techniques are based on merging techniques and that a hierarchical region based representation can be useful for the implementation of either segmentation or connected operators. Both strategies can be implemented using pruning techniques, and thus we may take advantage of the fixed tree structure to design complex and at the same time efficient analysis techniques.

The approach taken to analyze and prune the tree is then discussed. The problem of the lack of robustness of non-increasing criteria is analyzed and a solution based on dynamic programming is proposed. Futhermore, the particularities associated to pruning each of the presented trees are also studied.

Finally, several pruning examples are presented. They range from the classical area filtering to motion filtering or rate-distortion based segmentation.

5.1.1 Segmentation Algorithms

The process of spatial partitioning of an image into mutually exclusive connected image regions is known as image segmentation. Each region is expected to be homogeneous with respect to a defined property. Typically, image segmentation is carried out in the early stages of a vision system to facilitate image representation and interpretation. A broad range of approaches to segment an image are actually available, each one has its advantages and drawbacks. A review of the available techniques may be found in [26] land [51]. The first article reviews segmentation techniques based on thresholding, classification and region growing techniques. The second one reviews the available techniques based on probabilistic approaches: relaxation techniques, fuzzy logic, classification with Markov models and neuron networks.

In this work we focus on region based segmentation techniques. The classical approach consists in, first, defining a set of initial regions and, second, in progressively merging regions to create a partition of the image. For instance, in the *Split & Merge* [28] algorithm, the set of initial regions is defined by the split process and the merging is then performed between the initial regions depending on a homogeneity criterion. The *Region Growing* [11] is another example: it is based on the merging of the set of initial regions with individual neighboring pixels that belong to an uncertainty area. Finally, the classical morphological tool for segmentation is the *watershed* [37, 95, 17]. It is also based on a merging strategy: the initial regions are the regional minima of the gradient of the image which are progressively expanded by merging of neighboring pixels in an order defined by the gradient value. Finally, the coding-oriented segmentation presented in [32, 31] also follows this basic merging strategy taking into account the coding cost of the resulting segmentation.

In [21] these merging techniques are studied in a general framework. The associated merging algorithm is studied in Sec. 4.2: to completely specify a region merging algorithm one has to specify three notions: the merging order (the order in which links are processed), the merging criterion (each time a link is processed, the merging criterion decides if the merging has to be done or not), and the region model (when two regions are merged, the model defines how to represent their union). In the case of a region growing algorithm, the merging order is defined by a similarity measure between two regions (for example color or gray-level distance), the merging criterion states that the pair of most similar regions have to be merged until a termination criterion is reached (for instance a given number of regions) and the region model is usually the mean of the pixels gray-levels or color values of the associated region.

Note that in the case of region growing the merging order (similarity between neighboring regions) is quite flexible and allows the definition of complex homogeneity models. By contrast, the merging criterion is very simple and crude: it states that the pair of most similar regions have always to be merged until the termination criterion is reached. As will be seen in the sequel, the use of trees allows us to increase the flexibility of the merging criterion. That is, more complex techniques may be applied to the nodes of the tree to select which nodes have to be pruned. By contrast, the tree structure is fixed and therefore the merging order cannot be changed "on the fly" just as one can do with a region merging algorithm.

5.1.2 Connected operators

Connected operators [76, 81, 16] are filtering techniques derived from mathematical morphology that eliminate part of the image content while preserving the contours of the remaining parts of the image. In short, these operators do not remove frequency components like linear filters, or some shapes like median filters or morphological opening and closing with a structuring element. They remove and merge flat zones.

Connected operators for sets

The definition of connected operator for a set is done via the notion of connected component. We assume that the classical connectivity (see Sec. 2.1) is used for that purpose.

Definition 5.1 (Connected operators for sets) *An operator ψ working on sets is said to be connected when for any set A, the symmetrical difference $A \setminus \psi(A)$ is exclusively composed of connected components of A or its complement A^C.*

This means that the operator acts only by preserving or removing connected components. In other words, a connected operator can only remove connected components of the sets or fill connected components of the background.

The operator may satisfy the following properties: the operator ψ is said to be *increasing* if $\forall X, Y \in E$ such that $X \subseteq Y \Rightarrow \psi(X) \subseteq \psi(Y)$, and *idempotent* if $\psi(\psi(X)) = \psi(X)$ (that is, applying the operator twice or more does not change the result).

Connected operators for functions

The extension of the definition of connected operator for sets to functions is done via the notion of flat zones and their associated partitions (see Sec. 2.1). The set of flat zones of an image f creates a partition $\mathcal{P}^{Fz}(f) = \cup_i Fz_i$, being Fz_i a flat zone of the image f.

Definition 5.2 (Connected operator) *An operator Ψ applied on a function f, $g = \Psi(f)$, is connected if the partition of flat zones of its input, $\mathcal{P}^{Fz}(f)$, and the partition of flat zones of its output, $\mathcal{P}^{Fz}(\Psi(f))$, satisfy $\mathcal{P}^{Fz}(\Psi(f)) \subseteq \mathcal{P}^{Fz}(f)$.*

If p and q are neighbors such that $f(p) = f(q)$ (that is, they belong to the same flat zone), a connected operator acts on f such that $g(p) = g(q)$. The latter condition can be reformulated in an equivalent manner as: if $g(p) \neq g(q)$, then $f(p) \neq f(q)$. Thus, a connected operator, also called a *planning* in [35], operates on the partition of flat zones by merging its flat zones. Connected operators are attractive because they simplify an image without introducing any new contour or blurring as would be the case of low pass filters. This property make these filters attractive in segmentation applications.

The dual operator, Ψ^*, is defined via the complementary of f, $-f$: $\Psi^*(f) = -\Psi(-f)$. In the digital image processing framework, the complementary of f is usually obtained as $NG - f$, where NG is the highest possible gray-level value of an image (usually $NG = 255$). Thus, $\Psi^*(f) = NG - \Psi(NG - f)$. Moreover, an operator is said to be self-dual if $\Psi(f) = \Psi^*(f)$.

There are several ways of creating connected operators for functions. The simplest one consists in extending a connected operator acting on sets [76, 27]: any operator ψ acting on sets can generate an operator Ψ acting on functions. For that purpose, first, the upper level sets $X_h(f)$ are generated for each possible gray-level value h $(0 \leq h \leq NG)$. Then, each binary image $X_h(f)$ is processed (independently) with the operator ψ acting on sets. Finally, the gray-level image $g = \Psi(f)$ is reconstructed using the filtered binary images $\psi(X_h(f))$ by means of a technique called "stacking". The classical approach consist in

$$g(p) = \Psi(f)(p) = \bigvee_h \{\forall j, h \geq j \geq 0 | p \in \psi(X_j)\} \tag{5.1}$$

If the binary connected operator ψ is increasing, the previous equation can be simplified:

$$g(p) = \Psi(f)(p) = \bigvee_h \{h | p \in \psi(X_h)\}$$

In the case of non-increasing binary operators, the use of Eq. 5.1 leads to non robust operators. This issue is further described and analyzed in Sec. 5.4.

The gray-level operator is said to be *increasing* if $\Psi(f) \leq \Psi(g)$ when $f \leq g$, *idempotent* if $\Psi(\Psi(f)) = \Psi(f)$, *anti-extensive* if $\Psi(f) \leq f$ and *extensive* if $\Psi(f) \geq f$. An *opening* is an increasing, idempotent and anti-extensive operator, whereas a *closing* is an increasing, idempotent and extensive operator.

Note that the approach of the extension of binary to gray-level connected operator presented previously is closely related to the Max-Tree representation. In the Max-Tree, each node is associated to a connected component that results from thresholding the image at all possible gray-levels (see Sec. 3.1). These nodes are structured according to their inclusion relationship between different gray-levels. Thus, the Max-Tree enables us to implement connected operators on functions by means of this extension mechanism.

As examples, let us briefly recall the *opening by reconstruction*, the *area opening* and the *λ-max* operator. The first two operators can deal with binary as well as gray-level images, whereas the last one is devoted to gray-level images only.

- *Opening by reconstruction* [94, 76, 16]: On a set, the binary opening by reconstruction ψ preserves all connected components that are not totally removed by a binary erosion by a structuring element of size $N \times N$. The extension to the case of gray-level images is straightforward. The resulting opening has a size simplification effect: it removes the bright components that are completely included in the structuring element. By duality, a closing by reconstruction can be defined. It simplification effect is similar to that of the opening but on dark components.

- *Gray-level area opening* [93]: This filter is similar to the previous one except that it preserves the connected components that have a number of pixels larger that a limit λ.

It also is an opening which has a size-oriented simplification effect, but the notion of size is different from the one used in opening by reconstruction [93]. By duality an *area closing* can be defined.

- λ-*max* operator: The criterion here is to preserve a connected component of the level set X_j if and only if this connected component hits a connected component of the level set $X_{j+\lambda}$. This is an example where the criterion involves two sets obtained at two different threshold values. The simplification effect of this operator is contrast oriented in the sense that it eliminates image components with a contrast lower than λ.

In [9, 10] an attribute based approach to openings is presented. It is intended to be an extension of the area opening presented in [93] to a more general framework. The approach they present is based on the extension of binary connected operators to gray-level operators by means of stacking. The extension includes increasing and non-increasing criteria. However, the problem of lack of robustness of the non-increasing criteria is not discussed.

In [54] it is discussed that region adjacency graphs may be used to implement connected operators. Connected operators act on the function by merging its associated flat zones and thus merging techniques on region adjacency graphs are suitable to create connected operators. The reader may also refer to [21, 70] for a discussion of the implementation of connected operators using the General Merging Algorithm presented in Sec. 4.2. In particular, it discusses the differences (from a region based merging strategy point of view) of segmentation algorithms and connected operators. Moreover, the paper analyzes how self-dual connected operators are created using the region based merging approach.

In [35] a classification of the connected operators is proposed. Classical connected operators have been implemented by means of the opening by reconstruction. These filters are a subclass of connected operators and are characterized by the fact that they never create a regional minimum and thus are suitable to create a *segmentation pyramid*. However, in a general case, a connected operator may create new regional extrema. The paper [35] studies two subclasses of connected operators which do not create new extrema: the *flattenings* and the *monotone planings*.

Definition 5.3 (Flattening) *An image g is a flattening of the image f if and only if for any pair of neighboring pixels (p, q)*

$$g(p) > g(q) \quad \Rightarrow \quad \begin{cases} f(p) \geq g(p) \ \textit{and} \ g(q) \geq f(q) \\ \qquad\qquad \textit{or} \\ f(q) \geq g(p) \ \textit{and} \ g(q) \geq f(p) \end{cases}$$

Basically the previous definition means that any transition in the destination image g is characterized by a larger variation in the source image f. Flattenings introduce a coupling between the values of the function f and the function g, whereas simple planings have no such coupling. The monotone planings introduce a coupling in the sense of variation and ignore the coupling of values.

Definition 5.4 (Monotone planning) *An image g is a monotone planning of the image f if and only if for any pair of neighboring pixels*

$$g(q) > g(q) \quad \Rightarrow f(p) > f(q)$$

Note that the dynamic range between $g(p)$ and $g(p)$ may be greater than between $f(p)$ and $f(q)$. The intersection of the two previous subclasses form the *levelings* [35, 36].

Definition 5.5 (Leveling) *An image g is a leveling of the image f if and only if for any pair of neighboring pixels* (p, q)

$$g(p) > g(q) \Rightarrow f(p) \geq g(p) \text{ and } g(p) \geq f(q)$$

The properties of levelings are studied in [36]. A construction technique based on the combination of geodesic dilation and erosion is proposed. The degree of simplification at each region of the original image f can be controlled by means of the marker function. Moreover, if the marker is self-dual the resulting leveling is self-dual. The extension to a color image is discussed in [23].

However, levelings are not able to deal with the *transition zones*, that is, the set of pixels that are between the regional extremum. The same issue appears on classical connected operators, and as a consequence for the Max-Tree, since it only acts on the regional maximum of the image. It will be seen that Binary Partition Trees allow to act on all the flat zones of the image.

This chapter demonstrates that hierarchical region based representations, and in particular, the Max-Tree and the Binary Partition Tree, may be used to implement connected operators. The work presented in this chapter has been partially published in [74, 69, 68, 67].

5.1.3 Discussion

As has been discussed, both segmentation algorithms and connected operators act on the image by merging regions. As will be seen, both techniques may be implemented using pruning techniques. Thus, the question that may arise now is what are the differences between a segmentation and a connected operator implemented using pruning techniques on trees. The purpose of a filter is usually to remove some details of the image which do not fulfill a certain criterion, whereas segmentation algorithms are usually used to partition the image into a set of meaningful regions by means of a homogeneity criterion.

The difference between both approaches is rather subtle. Consider, for instance, a filter that removes from a sequence of frames those objects that do not follow a certain motion.

The output of the filtering would show us the sequence of images from which some objects have been removed, whereas the residue between the original and filtered sequence would show us the objects that have been removed. The binary masks associated to the region of support of the residue may be interpreted as a segmentation: we may say that objects have been classified according to a motion criterion. Thus, filtering and segmentation seem to be closely related. In our work, we will talk of "segmentation" if the pixel based representation of the output resulting from pruning the tree is a partition (that is, a label image). On the other hand, we will talk of "filtering" if the output is the original image from which some elements have been removed.

5.2 Tree processing strategy

In this section we discuss the particular issues related to processing the Max-Tree and the Binary Partition Tree. We assume that the analysis algorithm applied on the tree defines a valid pruning strategy (see Sec. 2.2.3 on page 15).

5.2.1 Pruning

Max-Tree

In Sec. 2.2.3 we emphasized on the fact that the pruning of the tree depends on what the tree represents itself. The Max-Tree nodes represent the set of connected components that arise from thresholding the image at all possible gray-levels (see Sec. 3.1). Assuming that a valid pruning strategy has been defined, pruning a subtree of the Max-Tree is associated to merge the nodes to be pruned with its first non pruned ancestor. The pruning operation can be seen in an equivalent manner as to remove from the image the connected components of the nodes to be pruned (see Fig. 5.1). Thus, it can be seen that no particular restriction has to be applied when pruning the tree.

Binary Partition Tree

In this case more attention should be paid when pruning the nodes, since the tree represents a set of hierarchical partitions. A pruning operation on this tree should lead to a tree representing a partition hierarchy. As discussed in Sec. 2.2.3 "pruning" a subtree is done by merging the corresponding nodes into a node which is the child of the first non-pruned ancestor.

Fig. 5.2 shows an example of Binary Partition Tree pruning. The set of original hierarchical partitions and its associated tree are shown on the left. In this particular case, the analysis algorithm decides to removed nodes R_1, R_2 and R_6. The pruning consists in merging the corresponding nodes and creating a new node at the position of R_6, representing the region

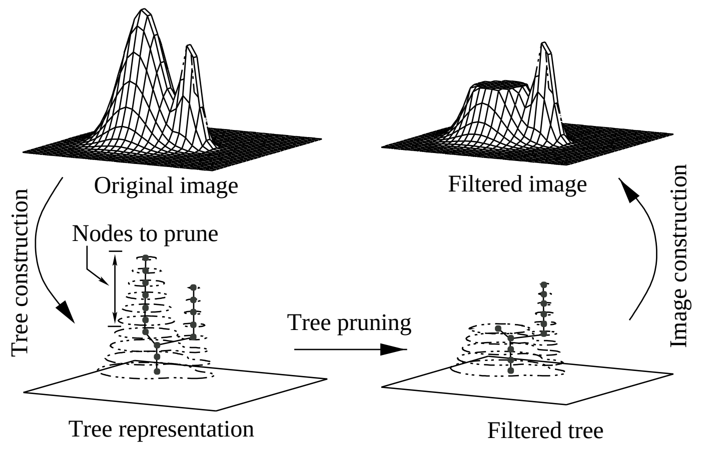


Figure 5.1: Example of Max-Tree processing.

of support associated to R_6. This procedure ensures us that pruning the tree results in a tree that represents a partition hierarchy. The resulting tree and its associated partition hierarchy is shown on the right of Fig. 5.2. Note that if R' is not created partition $\mathcal{P}^3_{pruned}$ would not be a partition since pixels of R' would not be classified as belonging to a determined region.

5.2.2 Reconstruction

Once the tree has been pruned, a pixel based representation of the associated tree is obtained.

Max-Tree

The Max-Tree represents the connected components resulting from thresholding the image at all possible gray-levels. The pruned tree can therefore be considered to be a structured representation of the connected components resulting from thresholding the filtered image at all possible gray-levels. The reconstruction is therefore done by stacking the connected components that make up the pruned tree.

Mathematically, the reconstructed image (see Fig. 5.1) is

$$g(p) = \bigvee_h \left\{ h : \exists k \,|\, p \in UpCC^k_h \right\} \tag{5.2}$$

where g represents the resulting image and $UpCC^k_h$ denotes a node of the Max-Tree we would like to reconstruct whose associated gray-level is h (see Sec. 3.1).

In the case of the Min-Tree, we may reconstruct the associated image with

$$g(p) = \bigwedge_h \left\{ h : \exists k \,|\, p \in LowCC^h_k \right\}$$

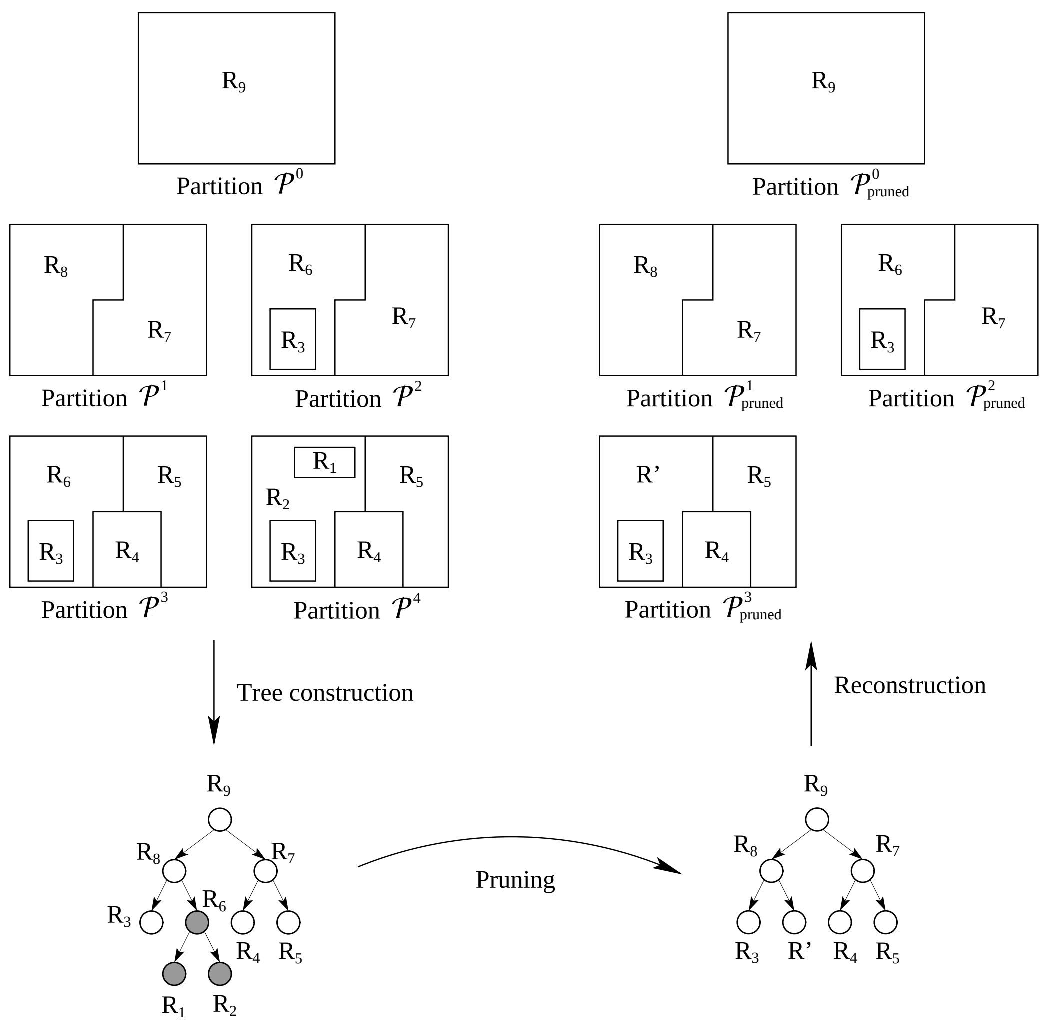

R_9
Partition $\mathcal{P}^0$
R_9
Partition $\mathcal{P}^0_{\text{pruned}}$
R_8
R_7
Partition $\mathcal{P}^1$
R_6
R_3
R_7
Partition $\mathcal{P}^2$
R_8
R_7
Partition $\mathcal{P}^1_{\text{pruned}}$
R_6
R_3
R_7
Partition $\mathcal{P}^2_{\text{pruned}}$
R_6
R_5
R_3
R_4
Partition $\mathcal{P}^3$
R_1
R_2
R_5
R_3
R_4
Partition $\mathcal{P}^4$
R'
R_5
R_3
R_4
Partition $\mathcal{P}^3_{\text{pruned}}$
Tree construction
Reconstruction
Pruning
R_9
R_8
R_7
R_3
R_6
R_4
R_5
R_1
R_2
R_9
R_8
R_7
R_3
R'
R_4
R_5

Figure 5.2: Example of Binary Partition Tree processing.

where $DownCC_k^h$ represents a node of the Min-Tree.

In the case of the Max-Tree the resulting operator is *anti-extensive*, that is, $g \leq f$. Moreover, the reconstructed function g is a leveling of f since $g \leq f$ and for each pair of neighboring pixels (p, q) the following condition holds: $g(p) > g(q) \Rightarrow g(q) = f(q)$.

On the other hand, processing with the Min-Tree results in extensive operators, $g \geq f$. The resulting function g is a leveling of f, since $g \geq f$ and for each pair of neighboring pixels (p, q) if $g(p) > g(q) \Rightarrow g(p) = f(p)$.

Binary Partition Tree

The partition associated to the leaves of the pruned tree may be constructed by simply labelling the region of support of each leaf node on the image. In some cases we may be interested in obtaining a color representation of the image. For that purpose, several approaches may be taken. One may fill each region R_i of the obtained partition with a certain color value. This color value may be extracted, for instance, by computing the mean color value associated to the region support R_i on the original image. If not stated otherwise, the latter approach is the one that it is used to present results in this chapter. The resulting operator is self-dual, $\Psi(f) = \Psi^*(f)$, if the tree has been created in a self-dual manner (see Sec. 4.4) and a self-dual model is used to fill each of the regions of the image.

5.3 Pruning strategy

Once the tree representation has been created, the analysis step analyzes each node N_k by measuring a specific criterion, $\mathcal{M}(N_k)$, on each of the associated regions and takes a decision on the elimination or preservation of the node.

In this work the usual approach that we have taken is, first, to measure a numerical value $\mathcal{M}(N_k)$ at the region R_k of each node N_k usually based on some region descriptor and second, on using a threshold λ to decide if the node N_k should be removed. If the criterion value $\mathcal{M}(N_k)$ is above (resp. strictly below) the threshold, the node is preserved (resp. removed).

$$\text{Node is removed if } \mathcal{M}(N_k) < \lambda \tag{5.3}$$

The tree together with the set of decisions taken on each node is called *decision tree* (the term should not be mistaken with the decision tree used for classification). Note that Eq. 5.3 does not necessarily define a valid pruning strategy. As will be seen, this depends on the properties of the criterion applied on the nodes of the tree. This issue is discussed in Sec. 5.4 and a robust solution based on dynamic programming is proposed in Sec. 5.5.

In order to allow a fast implementation of the analysis step the criterion should be computed in a recursive manner. In this work, “recursive” means that the criterion associated

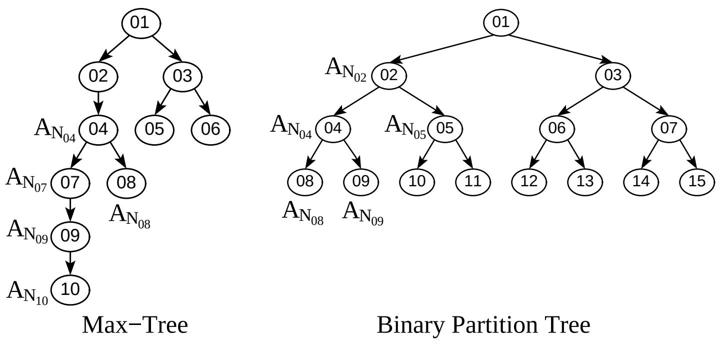


Figure 5.3: Example of recursive implementation of a criterion: size. See text.

to a node should be computed from the criteria measured on the children nodes. Fig. 5.3 shows an example for the size criterion (number of pixels). The size of a region $R = \cup R_i$ is equal to the sum of the sizes of the regions R_i it composes. In the case of the Binary Partition Tree, each node represents the union of its two children nodes. Thus, the size of a region R_k of a Binary Partition Tree is equal to the sum of the sizes of its two children nodes: for instance, $A_{N_{04}} = A_{N_{08}} + A_{N_{09}}$ or $A_{N_{02}} = A_{N_{04}} + A_{N_{05}}$. A similar approach may be taken for the case of the Max-Tree: the purpose in this case is to measure the size in pixels A_{N_k} of the connected component associated to the node N_k of the tree. The latter criterion can be computed in a recursive way using the relationship between the flat zones and connected components discussed in Sec. 3.1 (and Sec. 2.1). In particular, we discussed that the Max-Tree may be seen as a structured representation of the flat zones the image is composed of. Let us denote with $A^{Fz}_{N_k}$ the number of pixels associated to the flat zones of the connected component of N_k (see Sec. 2.1). The size associated to a connected component R_k of a Max-Tree, A_{N_k}, can be assessed by summing the contribution of the sizes associated to the connected components of its children and the term $A^{Fz}_{N_k}$. For instance, $A_{N_{09}} = A_{N_{10}} + A^{Fz}_{N_{09}}$ and $A_{N_{04}} = A_{N_{07}} + A_{N_{08}} + A^{Fz}_{N_{04}}$.

It can be seen that a recursive implementation of the criterion reduces the computational complexity of the analysis algorithm. With a non-recursive implementation of a criterion, the nodes of the tree are analyzed, for instance, by accessing for each node to the list of pixels associated to it. This results in a higher computational complexity.

5.4 Increasing and Non-Increasing Criteria

Mathematically, a criterion $\mathcal{M}$ assessed on a region R is said to be increasing if the following property holds:

$$\forall R_1 \subseteq R_2 \Rightarrow \mathcal{M}(R_1) \leq \mathcal{M}(R_2) \tag{5.4}$$

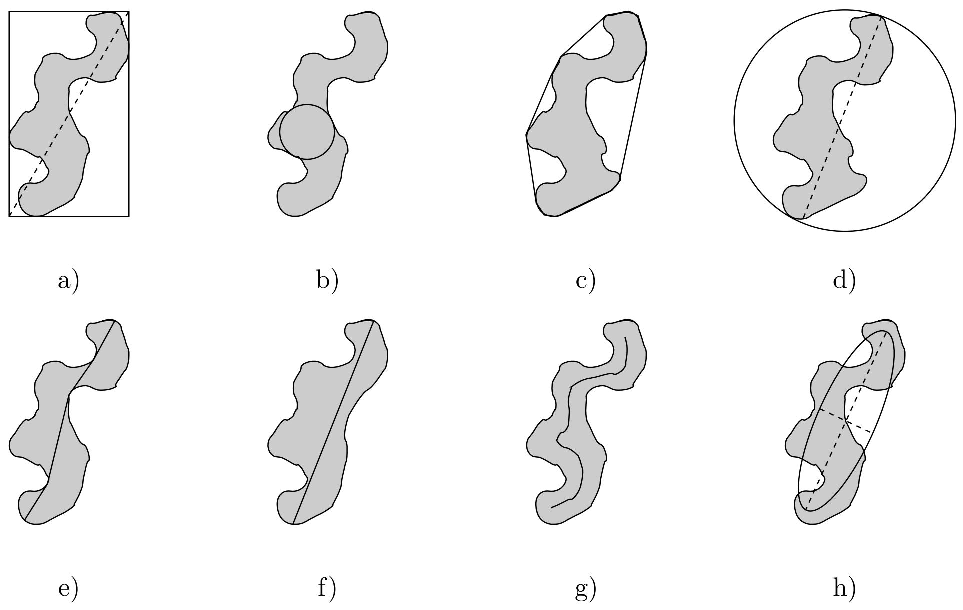


Figure 5.4: Examples illustrating increasing (a–d) and non-increasing (e–h) criteria [10]. a) The length of the diagonal of the minimum rectangle in a given direction that encloses the region R. b) The radius, diameter, or area of the largest circle that can fit into the region. c) The area or perimeter of the convex hull of the region. d) The radius, diameter or area of the smallest circle that encloses the region. e) The maximum geodesic distance in the region. f) The geodesic distance for a superset of the previous region. g) The length of the minimal skeleton of the connected region. h) The major and minor axes of the ellipse that best fits in the region.

In Fig. 5.4a–d several examples of increasing criteria that may be measured on a region R are shown. If region R satisfies $\mathcal{M}(R) \geq \lambda$ then all supersets of R also satisfy this criterion.

On the tree, an increasing criterion is interpreted as follows: assume that all nodes corresponding to regions where the criterion value is lower than a given threshold should be removed. The increasingness of the criterion and the hierarchical structure of the tree guarantees that if a node has to be removed all its descendants have also to be removed. Thus, an increasing criterion defines a valid pruning strategy. An example of a decision tree resulting from applying an increasing criterion is shown in Fig. 5.5.

However, there are many criteria that are non-increasing. Fig. 5.4e–f illustrates some examples. In Fig. 5.4e the maximum geodesic distance [87] of the connected region is depicted. The geodesic distance may decrease if the connected region increases in size, as shown in Fig. 5.4f. There are many more non-increasing criteria associated which a region. Indeed, on closer inspection, its seems that any attribute which is associated to shape is non-increasing [10]. Shape is inherently non-dimensional and hence independent of size. Examples of strict shape attributes include compactness and eccentricity. In contrast, the majority of

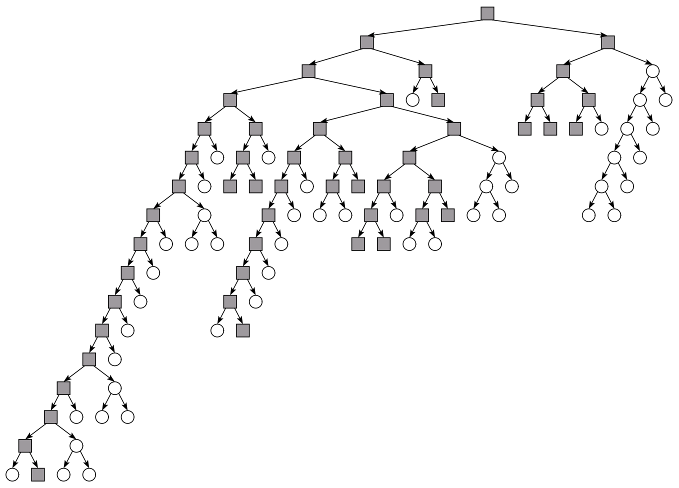


Figure 5.5: Example of increasing criterion. The pruning strategy is straightforward since the increasingness of the criterion guarantees that if a node has to be removed all its descendants have also to be removed. Gray squares: nodes to be preserved, white circles: nodes to be removed.

sizing criteria, such as the ones illustrated in Fig. 5.4a–d, are increasing.

If the criterion is not increasing, the pruning strategy is not straightforward since the descendants of a node to be removed have not necessarily to be removed. Fig. 5.6 illustrates this case. As before, a criterion has been assessed on each of the nodes of the tree and then a decision, remove or preserve, has been taken based on a threshold. The nodes to be preserved (resp. removed) are depicted with a gray square (resp. white circle). Note that if we follow either Path A or Path B in Fig. 5.6, we see that there are some oscillations of the remove/preserve decisions.

Let us analyze several approaches that may be taken in the case of a non-increasing criterion on the decision process. For that purpose, consider a leaf node of the tree and the set of nodes constituting the path from this leaf to the root node. In the example of Fig. 5.7, if we select N_1, the path to the root N_3 is $\{N_1, N_2, N_3\}$. Consider now the sequence of the criterion values $\mathcal{M}(N_k)$ associated to the previous path. In the example of Fig. 5.7, the *criterion sequence* is $\mathcal{M}(k) = \{\mathcal{M}(N_1), \mathcal{M}(N_2), \mathcal{M}(N_3)]\}$ and is represented as a curve (function of k) on the right side of Fig. 5.7.

If the criterion is increasing, the criterion sequence $\mathcal{M}(k)$ is a monotonic function and thus the pruning strategy is straightforward independently of the threshold value λ that is used. If the criterion is non-increasing, the criterion sequence $\mathcal{M}(k)$ may fluctuate around the λ value and the definition of the set of nodes to remove is less straightforward. Several different rules have been reported in the literature [27, 76, 74] for connected operators in order to deal with the non-increasing case. These rules impose additional conditions to Eq. 5.3 so that the

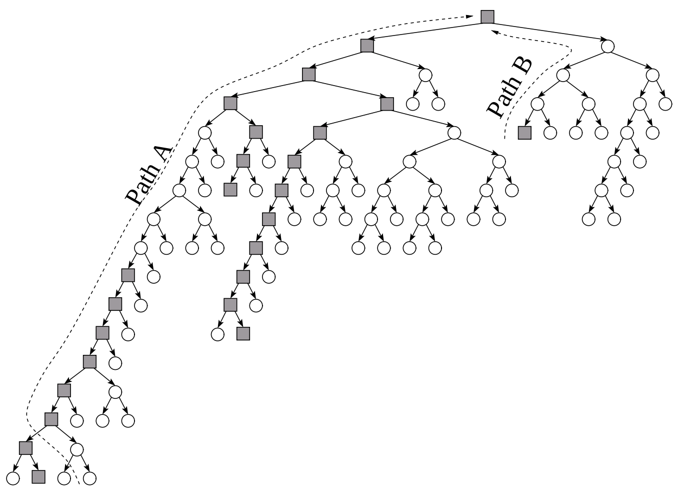


Figure 5.6: Example of non-increasing criterion. The pruning strategy is not straightforward since the criterion does not guarantee that if a node has to be removed all of its descendants have also to be removed (see decisions along path A or B). Gray squares: nodes to be preserved, white circles: nodes to be removed.

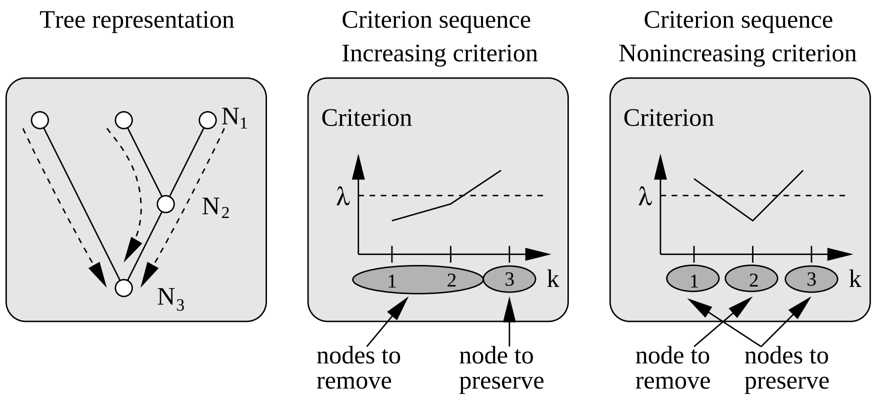


Figure 5.7: Criterion sequence for each local maximum.

resulting tree results in a decision tree which defines valid pruning strategies.

1. *Minimum decision*: A node N_k is preserved if $\mathcal{M}(N_k) \geq \lambda$ and if all its ancestors also satisfy this condition. The node is removed otherwise. This rule is illustrated in Fig. 5.8a. The original decision tree, shown in Fig. 5.6, has been transformed into a decision tree in which a valid pruning strategy is defined.

2. *Maximum decision*: This rule is the dual of the previous one: a node is removed if $\mathcal{M}(N_k) < \lambda$ and if all its descendant nodes satisfy the same relation. The node N_k is preserved otherwise (see Fig. 5.8b).

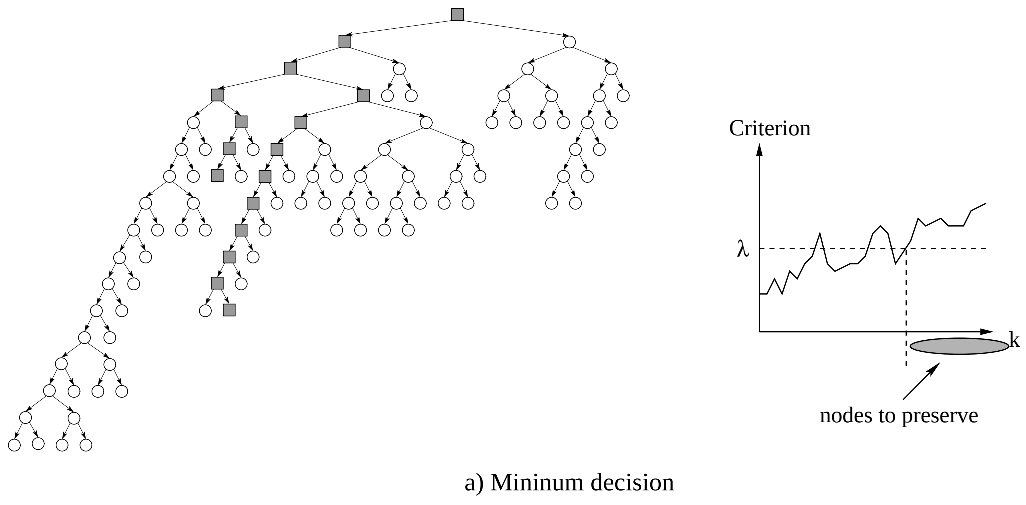
The previous minimum and maximum rules define a valid pruning strategy. However, from our practical experience, the minimum and maximum rules are not robust enough. For instance, similar images may produce quite different results, or small modifications of the criterion threshold involves drastic changes on the output. The effect is particular noticeable in video sequences, in which the threshold is fixed and a particular object should be removed through the video sequence. In [73], an improvement of the decision robustness is proposed. It consists in filtering the decision sequence, for example, with a median filter. This solution actually provides more robustness, however, in the following, a different solution is proposed. It turns out to be much more robust than any of the previous ones. It relies on a formulation of the decision process as an optimization problem. The approach has been published in [74, 67].

5.5 Optimization for Non-Increasing Criteria

The proposed solution consists in the formulation of the decision process as an optimization problem. The optimization should create a valid pruning strategy while preserving as much as possible the decisions defined by the criterion (see Eq. 5.3). This problem may be viewed as a dynamic programming issue that can be efficiently solved with a Viterbi algorithm [96].

The objective of the optimization algorithm is to change the minimum number of decisions taken with Eq. 5.3 so that a valid pruning strategy is defined (see Sec. 2.2.3). For that purpose a trellis is used. By applying the Viterbi algorithm on such trellis the set of nodes whose decisions have to be changed can be obtained.

An example of the trellis on which the Viterbi algorithm is applied is illustrated in Fig. 5.9. It has the same structure as the tree that has to be pruned except that each node N_k of the tree has associated two trellis states: *preserve* N_k^P and *remove* N_k^R. The two states of each child node are connected to the two states of its parent. However, in order to avoid non-increasing decisions, the preserve state of a child is not connected to the remove state of its parent. As a result, the trellis structure guarantees that if a node has to be removed its children have also to be removed. Furthermore, a cost is assigned to each state of N_k: this



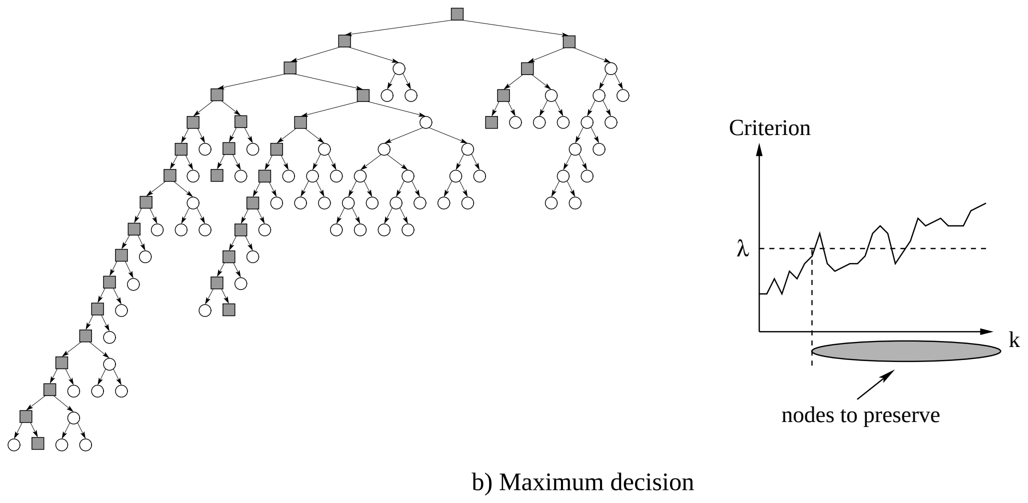


Figure 5.8: Illustration of various decision rules in the case of non-increasing criterion for the Fig. 5.6. a) Minimum decision. b) Maximum decision.

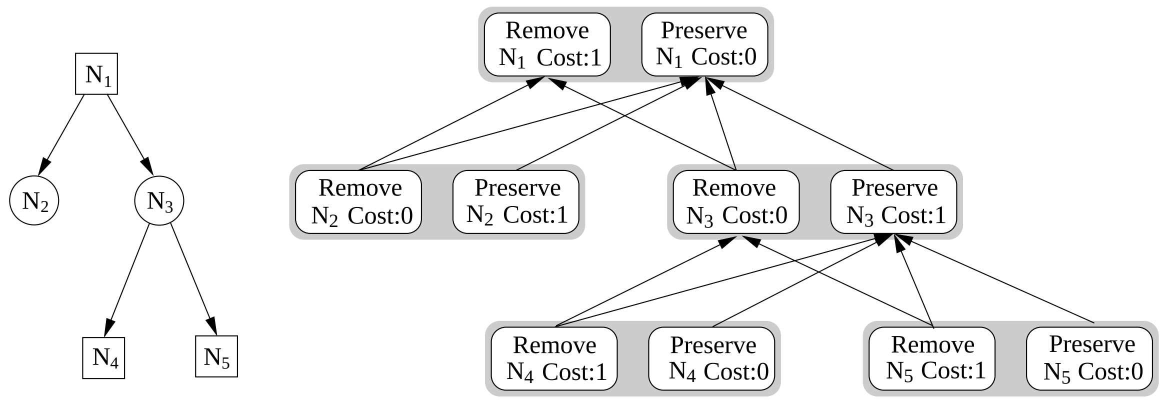


Figure 5.9: Example of non-increasing decision tree and associated trellis structure for the Viterbi algorithm. On the left, nodes to be preserved (resp. removed) are depicted with a square (resp. circle). In each of the possible states of the trellis state (right), the associated cost is indicated.

cost should be associated to the cost of taking a preserve or remove decision for the node N_k. In this work we use the following approach to compute the cost associated to a state: if the criterion value states that the node of the tree has to be removed (according to Eq. 5.3), the cost associated to the remove state is equal to zero (no modification) and the cost associated to the preserve state is equal to one (one modification). Similarly, if the criterion value states that the node has to be preserved, the cost of the remove state is equal to one and the cost of the preserve state is equal to zero. Although some modifications may be much more severe than others, the cost choice has no strong effect on the final result. This issue of cost selection is similar to the hard versus soft decision of the Viterbi algorithm in the context of digital communications [96].

The cost associated to each state of the trellis is used to take a decision on each node of the tree, remove or preserve, such that a valid pruning strategy is defined. The goal of the Viterbi algorithm is to define the set of decisions such that:

$$\text{Min} \sum_k \text{Cost}(N_k) \text{ such that a valid pruning strategy is defined} \tag{5.5}$$

To find the optimum set of decisions the associated trellis is used. Note that the trellis defines a set of paths going from all leaf nodes to the root node. For each node, the path can go through either the preserve or the remove state of the trellis. The cost of a path is equal to the sum of the costs of its individual state nodes it goes through. The Viterbi algorithm is used to find the paths that minimize the global cost at the root node. The optimization is achieved in a bottom-up iterative fashion. For each node, it is possible to define the optimum paths ending at the preserve state and at the remove state.

Let us first discuss the case of a node of the tree having one child node, see example of

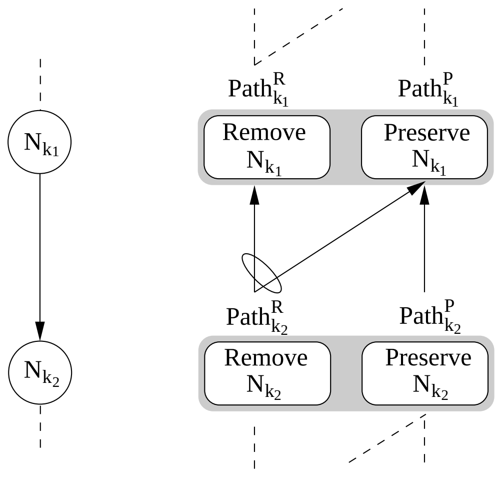


Figure 5.10: Trellis construction for the decision in the case of a single branch tree.

Fig. 5.10. Between nodes N_{k_1} and N_{k_2} there are three allowed transitions in its associated trellis: $N^P_{k_2} \rightarrow N^P_{k_1}$, $N^R_{k_2} \rightarrow N^P_{k_1}$ and $N^R_{k_2} \rightarrow N^R_{k_1}$. Assume now that the two optimum paths (lowest cost) starting from a leaf node and ending at $N^P_{k_2}$ and $N^R_{k_2}$ are known. Let us call $Path^P_{k_2}$ and $Path^R_{k_2}$, respectively, these two optimum paths. The definition of the optimum paths ending at $N^P_{k_1}$ and $N^R_{k_1}$ can be easily defined by a local decision. The optimum path ending in $N^P_{k_1}$, that is $Path^P_{k_1}$, is defined by the following rule:

```
If $\big(Cost(Path^P_{k_2}) + Cost(N^P_{k_1})\big) \leq \big(Cost(Path^R_{k_2}) + Cost(N^P_{k_1})\big)$
  then
    $Path^P_{k_1} = Path^P_{k_2} \bigcup (N^P_{k_2} \rightarrow N^P_{k_1})$
    $Cost(Path^P_{k_1}) = Cost(Path^P_{k_2}) + Cost(N^P_{k_2} \rightarrow N^P_{k_1})$
  else
    $Path^P_{k_1} = Path^R_{k_2} \bigcup (N^R_{k_2} \rightarrow N^P_{k_1})$
    $Cost(Path^P_{k_1}) = Cost(Path^R_{k_2}) + Cost(N^R_{k_2} \rightarrow N^P_{k_1})$
```

This rule simply states that the optimum path ending at state $N^P_{k_1}$ has to go through either state $N^P_{k_2}$ or $N^R_{k_2}$ and that the best path is the one leading to the lowest additive cost.

A similar decision rule can be defined for the best path ending at state $N^R_{k_1}$. As already discussed, the transition $N^P_{k_2} \rightarrow N^R_{k_1}$ is not allowed since it would lead to a non-increasing decision tree. Therefore, the optimum path (path of lowest cost) ending at $N^R_{k_1}$ comes always from the remove state of N_{k_2}, that is $N^R_{k_2}$:

$$Path^R_{k_1} = Path^R_{k_2} \bigcup (N^R_{k_2} \rightarrow N^R_{k_1})$$
$$Cost(Path^R_{k_1}) = Cost(Path^R_{k_2}) + Cost(N^R_{k_2} \rightarrow N^R_{k_1})$$

This rather simple procedure has to be extended to deal with trees with various branches. The extension is depicted in Fig. 5.11 in the case of the junction of two branches, but the

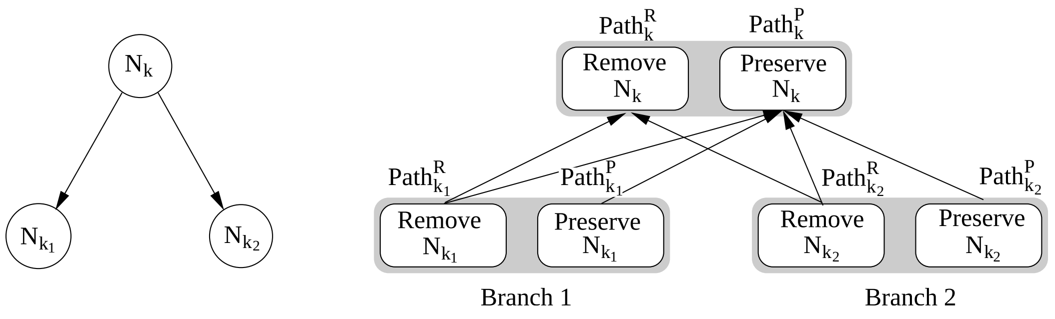


Figure 5.11: Trellis construction for the decision tree in the case of multiple branches.

procedure is general and can deal with an arbitrary number of branches. In our case there is not one but two optimum paths ending at each node N_k. One branch comes from branch 1 whereas another one comes from branch 2. These paths are independent from each other. As a result, we have to define independently these two paths. Let us first analyze the case of the trellis node N_k^P.

The path starting from a leaf node and ending at state N_k^P, $Path_k^P$, is composed of two sub-paths: the first one, $Path_k^{P,1}$, comes from the branch 1 and the second one, $Path_k^{P,2}$, from the branch 2 (see Fig. 5.11). In both cases, the path can emerge either from the preserve or from the *remove* state of the child nodes. Assume that the optimum paths ending at nodes N_{k_1} and nodes N_{k_2} are known. The possible paths ending at state N_k^P coming from the left and right branch are:

$$\begin{array}{lcl} Path_k^{P,1} & = & Path_{k_1}^R \bigcup (N_{k_1}^R \rightarrow N_k^P) \quad \text{or} \quad Path_{k_1}^P \bigcup (N_{k_1}^P \rightarrow N_k^P) \\ Path_k^{P,2} & = & Path_{k_2}^R \bigcup (N_{k_2}^R \rightarrow N_k^P) \quad \text{or} \quad Path_{k_2}^P \bigcup (N_{k_2}^P \rightarrow N_k^P) \end{array}$$

The Viterbi algorithm should select, among each of the two possible paths of $Path_k^{P,1}$ and $Path_k^{P,2}$ the one of lowest cost. The path ending at node N_k is the union of the optimum paths coming from each of the branches

$$Path_k^P = Path_k^{P,1} \bigcup Path_k^{P,2}$$

Therefore, the optimum path (path of lower cost) for each child can be easily selected. For the first branch the optimum path is

```
If $Cost(Path_{k_1}^P) \leq Cost(Path_{k_1}^R)$
  then
    $Path_k^{P,1} = Path_{k_1}^P \bigcup (N_{k_1}^P \rightarrow N_k^P)$
    $Cost(Path_k^{P,1}) = Cost(Path_{k_1}^P)$
  else
    $Path_k^{P,1} = Path_{k_1}^R \bigcup (N_{k_1}^R \rightarrow N_k^P)$
    $Cost(Path_k^{P,1}) = Cost(Path_{k_1}^R)$
```

For the second branch the optimum path is

```
If $Cost(Path^P_{k_2}) \leq Cost(Path^R_{k_2})$
    then
        $Path^{P,2}_k = Path^P_{k_2} \bigcup (N^P_{k_2} \to N^P_k)$
        $Cost(Path^{P,2}_k) = Cost(Path^P_{k_2})$
    else
        $Path^{P,2}_k = Path^R_{k_2} \bigcup (N^R_{k_2} \to N^P_k)$
        $Cost(Path^{P,2}_k) = Cost(Path^R_{k_2})$
```

The total cost of the path ending at N^P_k is

$$Cost(Path^P_k) = Cost(Path^{P,1}_k) + Cost(Path^{P,2}_k) + Cost(N^P_k)$$

In the case of the remove state, N^R_k, the two sub-paths can only come from the remove states of the children. So, no selection has to be done. The path and its cost are constructed as follows:

$$\begin{aligned}
Path^{R,1}_k &= Path^R_{k_1} \bigcup (N^R_{k_1} \to N^R_k) \\
Path^{R,2}_k &= Path^R_{k_2} \bigcup (N^R_{k_2} \to N^R_k) \\
Path^R_k &= Path^{R,1}_k \bigcup Path^{R,2}_k \\
Cost(Path^R_k) &= Cost(Path^R_{k_1}) + Cost(Path^R_{k_2}) + Cost(N^R_k)
\end{aligned}$$

The presented optimization algorithm is initialized at the leaf nodes N_k by setting the cost of its associated path to the cost of its associated trellis node, that is, $Cost(Path^P_k) = Cost(N^P_k)$ and $Cost(Path^R_k) = Cost(N^R_k)$. The optimization procedure is then iterated in a bottom-up fashion until the root node is reached. One path of minimum cost ends at the preserve state of the root node and another path ends at the remove state of the root node. Among these two paths, the one of minimum cost is selected. This path connects the root node to all leaves and the states it goes through define the final decisions. By construction, these decisions define a valid pruning strategy and they are as close as possible to the original decisions with Eq. 5.3.

A complete example of optimization is shown in Fig. 5.12. The original tree involves 5 nodes. As before, the preserve decisions are depicted by a square whereas the remove decisions are indicated by a circle. As can be seen, the original tree does not correspond to a set of increasing decisions because N_3 should be removed but N_4 and N_5 should be preserved. The algorithm is initialized by creating the trellis (see Fig. 5.12) and by populating the states by their respective cost. The first step of the algorithm consists in selecting the lowest cost paths that go from states N^R_4, N^P_4, N^R_5, N^P_5 to states N^R_3, N^P_3. The corresponding trellis is shown in the upper part of Fig. 5.12 together with the corresponding costs of the four surviving paths which are depicted with a solid line. Paths to be removed are depicted with a dotted line. The second step iterates the procedure between states N^R_2, N^P_2, N^R_3, N^P_3 and states

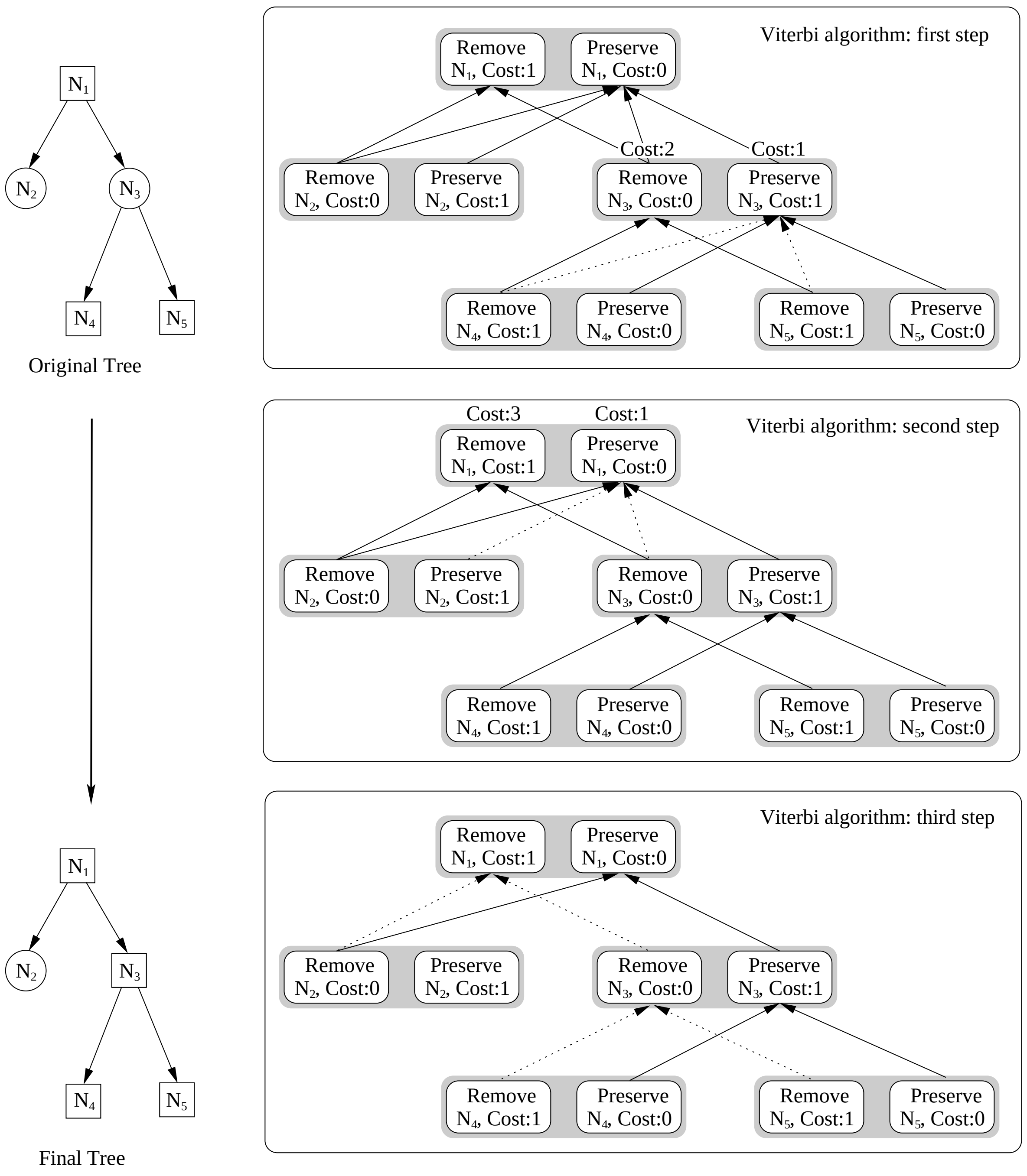


Figure 5.12: Definition of the optimum decisions by the Viterbi algorithm. On the right, the different steps for the Viterbi algorithm on the trellis structure of Fig. 5.9 are shown. At each iteration, the paths to be discarded are depicted with a dotted line, whereas the optimum paths are depicted with a solid line. On the left, the initial and final decision tree are shown.

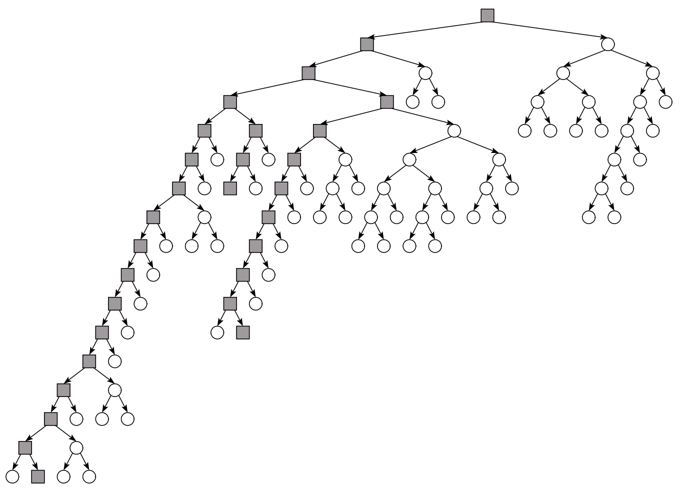


Figure 5.13: Set of increasing decisions resulting from the use of the Viterbi algorithm on the original tree of Fig. 5.6. Five decisions along path A and one decision along path B have been modified. Gray squares: nodes to be preserved, white circles: nodes to be removed.

N_1^R, N_1^P. Here again, only four paths survive. They are indicated in the central diagram of Fig. 5.12. Finally, the last step consists in selecting the path of lowest cost that terminates at the root states. In the example of Fig. 5.12, the path ending at the remove state of the root node (N_1^R) has a cost of 3, whereas the path ending at the preserve state (N_1^P) has a cost of 1. This last path is taken since it corresponds to the lowest cost path. In order to find the optimum increasing decisions, one has to track back the selected path from the root to all leaves. In our example, we see that the paths hit the following states: N_1^P, N_2^R, N_3^P, N_4^P and N_5^P. Note that a valid pruning strategy is defined and just one modification of the original decisions as given by Eq. 5.3 is involved. The diagram at the bottom of Fig. 5.12 shows the final path together with the modified tree. As can be seen, the only modification has been to change the decision of node N_3 and the resulting set of decisions is increasing.

A complete example of decisions modification is shown in Fig. 5.13. The original decision tree is shown in Fig. 5.6. The Viterbi algorithm has to modify 5 decisions along path A and one decision along path B (see Fig. 5.13) to get the optimum pruning strategy.

To summarize this section, let us say that the pruning strategy can be applied directly on the tree if the decision criterion is increasing (size is a typical example). In the case of a non-increasing criterion such as the perimeter, the Viterbi algorithm can be used to modify the smallest number of decisions so that increasingness is obtained. These modifications define a valid pruning strategy. If not stated otherwise, it is assumed that the optimization algorithm is applied on the tree when a non-increasing criterion is used.

5.6 Discussion

Let us discuss now the relationship of the optimization algorithm with the minimum and maximum restitution presented previously. For that purpose, we are going to analyze the number of nodes that are preserved and removed for a fixed branch of the tree for the three types of decision.

Let us denote with $PN_{Min}(\mathcal{M}(k),\lambda)$ the number of nodes that are preserved when a minimum decision is applied on a fixed criterion sequence $\mathcal{M}(k)$ of a tree (see page 85) using threshold λ. Similarly, we denote with $PN_{Max}(\mathcal{M}(k),\lambda)$ (resp. $PN_{Opt}(\mathcal{M}(k),\lambda)$) the number of nodes that are preserved using a maximum (resp. optimum) decision. The following relationship is fulfilled

$$PN_{Min}(\mathcal{M}(k),\lambda) \leq PN_{Opt}(\mathcal{M}(k),\lambda) \leq PN_{Max}(\mathcal{M}(k),\lambda) \tag{5.6}$$

This is a rather interesting result, since it states that the optimization algorithm performs a decision that is "between" the minimum and the maximum decision. Compare, for instance, Fig. 5.6, Fig. 5.8 and Fig. 5.13.

The reason is rather intuitive. Given a criterion sequence $\mathcal{M}(k)$, denote with $N_{k_{Min}}$ (resp. $N_{k_{Max}}$) the node associated to the criterion sequence such that its proper ancestors are preserved and its descendants removed when a minimum (resp. maximum) decision is applied on the tree. When the optimization algorithm is applied to the branch associated to the criterion sequence $\mathcal{M}(R)$, for the set of paths ending at a descendant node N_k of $N_{k_{Max}}$, it is satisfied that $Cost(Path_k^R) < Cost(Path_k^P)$. Thus, the optimization algorithm states that at least all descendants of N_{Max} are removed. On the other hand, for any proper ancestor N_k of $N_{k_{Min}}$, it is satisfied that $Cost(Path_k^P) < Cost(Path_k^R)$. Thus, all the proper ancestors of $N_{k_{Min}}$ have to be preserved.

On the Max-Tree, Eq. 5.6 is translated to the following property

$$f_{Min} \leq f_{Opt} \leq f_{Max}$$

where f_{Min}, f_{Opt} and f_{Max} represent, respectively, the reconstructed function (see Eq. 5.2 on page 80) when a minimum, optimum or maximum decision is used, respectively. Furthermore, the partition of flat zones associated to the minimum ($\mathcal{P}_{Min}^{Fz}$), optimum ($\mathcal{P}_{Opt}^{Fz}$) and maximum ($\mathcal{P}_{Max}^{Fz}$) decision are related by

$$\mathcal{P}_{Min}^{Fz} \subseteq \mathcal{P}_{Opt}^{Fz} \subseteq \mathcal{P}_{Max}^{Fz}$$

whereas for the Binary Partition Tree

$$\mathcal{P}_{Min} \subseteq \mathcal{P}_{Opt} \subseteq \mathcal{P}_{Max}$$

where $\mathcal{P}_{Min}$, $\mathcal{P}_{Opt}$ and $\mathcal{P}_{Max}$ correspond to the partition associated to the leaves of the tree when a minimum, optimum or maximum decision is used.

The previous equations state that applying a minimum decision results in an image with the smallest number of regions (with respect to the optimum or maximum decision). Thus, the minimum decision is the decision that most "simplifies" the image. On the other hand, the maximum decision results in an image with the highest number of regions (with respect to the minimum or optimum decision), being thus the method that least "simplifies" the image. The optimum decision results in a simplification of the image that is in between of both minimum and maximum decision.

Finally, note that the equality is held in the previous equations if the criterion is increasing.

5.7 Examples

This section shows several filtering and segmentation examples that have been applied in our work to the two types of trees we have discussed: the Max-Tree and the Binary Partition Tree.

5.7.1 Marker & Propagation Segmentation

The watershed [37] is the classical tool that has been used in Mathematical Morphology for image segmentation (see also Sec. 2.3.3). However, except for a few simple cases the watershed cannot be applied directly since it usually produces a severe over-segmentation, which is difficult to overcome. The classical reguralization process uses markers: the strategy consists, first, in "marking" (defining with markers) the interior of the regions to be segmented and, second, in performing a propagation of these markers to eventually define the regions contours. This second step can be viewed as the definition of the zone of influence of each marker. Let us mention, that depending on the application, the markers can be computed automatically [37, 66] or manually.

Propagation processes based on similarity between neighboring regions can be easily implemented using tree structures. In this case, the descriptors to attach to the nodes are the markers, and the pruning is done according to the zone of influence of each marker. The zone of influence of each marker is obtained using the links defined by the tree structure.

Max-Tree

Let us first describe this propagation on a simple example using the Max-Tree structure. Fig. 5.14 shows on the left a simple image made of six flat zones. The associated Max-Tree is made up of six regions. Consider now two markers, A and B, that have to be propagated by merging with neighboring regions. Let us first mark the two corresponding nodes on the tree (see Fig. 5.14b). Propagation is performed on the tree using a bottom-up approach. By construction of the Max-Tree, the most similar neighboring region with respect to a given node

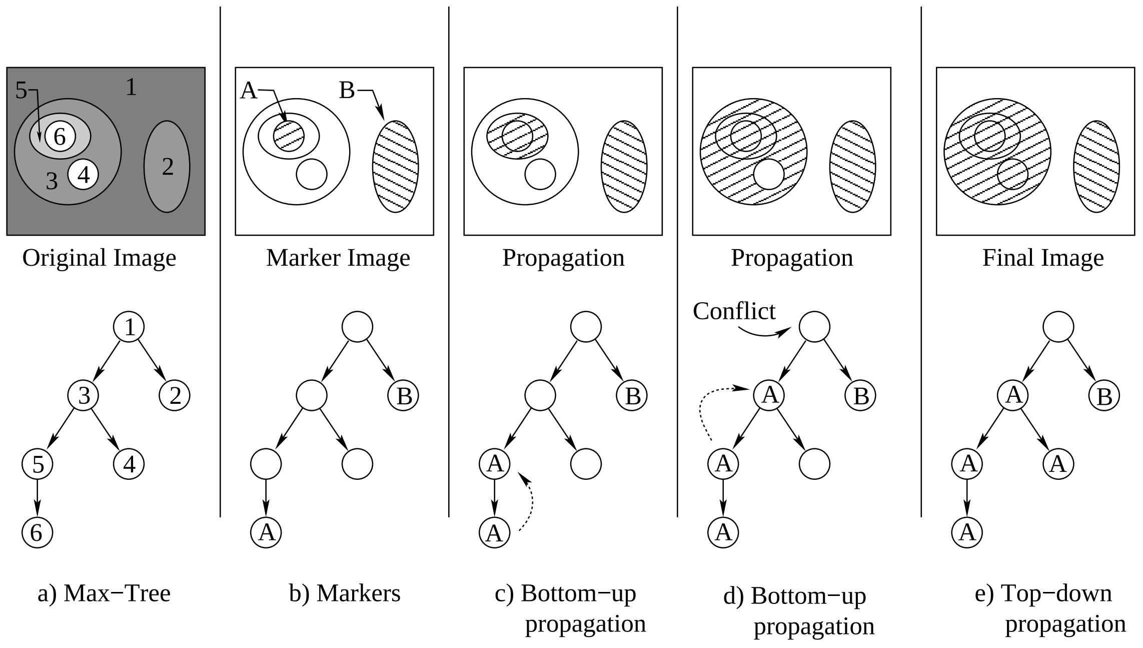


Figure 5.14: Propagation process on the Max-Tree. a) Original image and its associated Max-Tree are shown, b) Marked image and tree, c-d) Bottom-Up propagation through Max-Tree, e) Top-Down propagation and final segmentation.

is represented by its parent. Therefore, the marker A associated to node N_6 is propagated to its parent node N_5. The result is depicted in Fig. 5.14c. The propagation then continues by marking the neighboring region which is most similar to region 5, that is node N_3. The result is shown in Fig. 5.14d. The next step would be based on propagating both markers A (node N_3) and B (node N_2). Of course this propagation can only be done if the siblings of each of the marked nodes is not in conflict with the marker itself, that is if none of the siblings' descendants has been assigned to a different marker. In the example of Fig. 5.14d, the node N_1 is in conflict with the markers A and B and thus propagation stops here. Finally, a top-down propagation of markers is done so that children nodes have the same label as their parents. The resulting segmentation is shown in Fig. 5.14e.

Binary Partition Tree

The same approach can be taken for the Binary Partition Tree. An example is shown in Fig. 5.15. The image is made up of four flat zones. The Binary Partition Tree shown in Fig. 5.15a indicates that regions R_1 and R_2 are the most similar. Once merged, their closest region is R_3. Finally, region R_4 is the most dissimilar. As can be seen in Fig. 5.15, the gray-level value associated to region R_4 is quite different from the values of other regions. Let us consider, as before, two markers A and B that have to be propagated by merging with

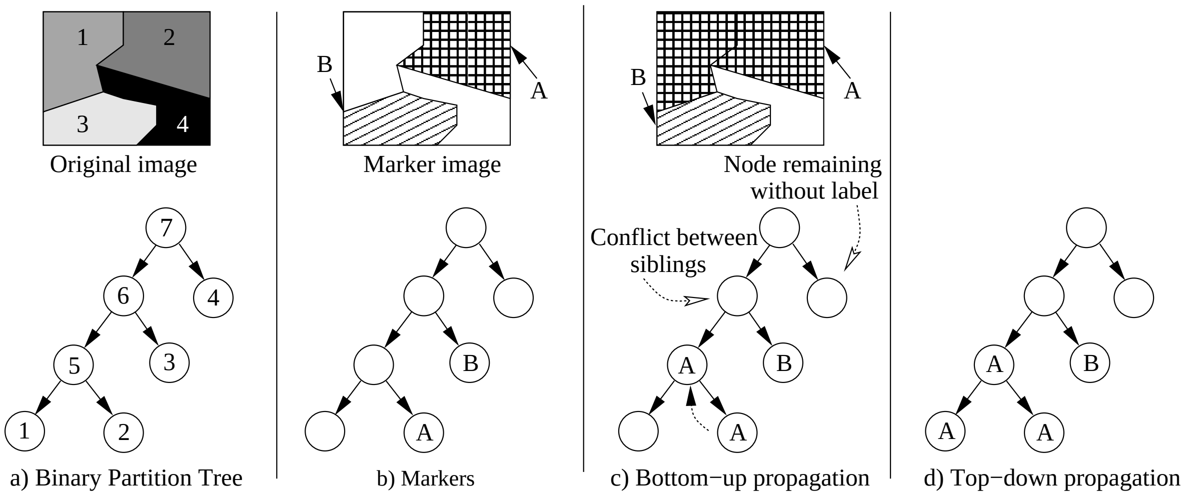


Figure 5.15: Propagation process on the Binary Partition Tree.

neighboring regions. Let us mark the two corresponding nodes on the tree (see Fig. 5.15b). By construction of the Binary Partition Tree, the most similar neighboring region with respect to a given marker is represented by its sibling and the result of the merging is represented by the parent. Therefore, the marker associated to a node is propagated to its parent. As before, this propagation can only be done if the sibling is not in conflict with the marker, that is if none of the siblings' descendants has been assigned to a different marker. In our case propagation stops at node N_6 because there is a conflict between the marker of node N_5 (marker A) and the marker of node N_3 (marker B).

Discussion

The propagation approach presented here for the Max-Tree is very similar to the watershed algorithm with the marker and propagation strategy. In the latter case the image is interpreted as a topographic surface and once the markers have been defined, these are flooded. As already pointed out in Sec. 2.3.3, a practical way of defining such flooding is based on the so known "immersion": assume that we pierce holes at each of the markers, and then slowly immerse the surface into a lake. The water will progressively fill up the different catchment basins of the image f. A "dam" is constructed at each pixel where the water coming from different markers would merge. Fig. 5.16 shows an example that compares the approach taken on the Min-Tree with the watershed algorithm[1]. Note that the Min-Tree associated to the $1D$ signal has the same structure as the one of Fig. 5.14. The evolution of the immersion algorithm as it floods the topographic surface is depicted in Fig. 5.16a through Fig. 5.16d. Note that

[1]In this particular case we take the Min-Tree since this tree is oriented towards the minima (as shown in Fig. 5.16) and thus it is easier to explain the behavior of the watershed algorithm.

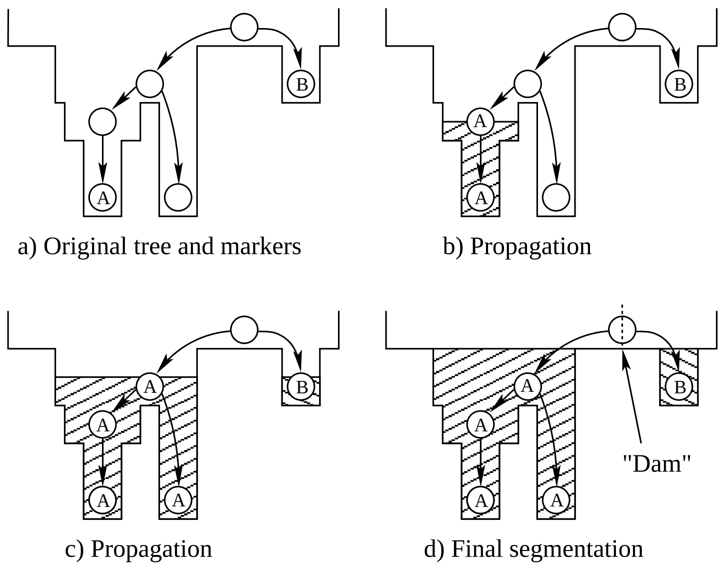


Figure 5.16: Comparison of the classical watershed algorithm with the propagation approach presented in Sec. 5.7.1.

propagation stops on the Min-Tree as soon as the corresponding watershed flooding algorithm reaches the gray-level where both lakes would meet, that is, the gray-level where a "dam" would begin to be constructed. On the Min-Tree this is indicated by marking the associated node as being in "conflict" with its children. In conclusion, we may say that the difference between a marker & propagation strategy with the watershed and the Min-Tree is rather subtle: the only difference resides in the fact that those regions that are in "conflict" in the Min-Tree correspond to the regions where the "dam" would be constructed.

Algorithm and examples

Fig. 5.17 precisely describes the algorithm performing the propagation on the tree structure. For that purpose, let ND_d be the set of nodes whose depth is d (see Sec. 2.2.1). We denote with Parent(N_k) the function that returns the parent of node N_k, whereas Label(N_k) denotes the label (marker) that is assigned to the node N_k. The possible values that may take Label(N_k) are: "none", if no label is assigned to the node, "conflict" if the node is in conflict with several labels, or a positive value indicating the label number associated to the node. The algorithm works in three main steps: first, assignment of markers to leaf nodes (lines 01–04), second, bottom-up propagation of the markers to parents locating conflicts between labels (lines 05–18) and third, top-down propagation of labels so that children nodes have the same label as their parents (lines 19–24).

A complete set of examples are shown in Fig. 5.18– 5.20. In all cases we assume that

```
    // Initialization of markers
for each node $N_k$ in the tree
    Label($N_k$)←none
assign a label to the nodes that overlap with a marker
    // Bottom up propagation of markers
for(d←max-depth; d $\geq$ 0; d←d-1)
    for each node $N_k$ in $ND_d$
        $N_j$ ← Parent($N_k$)
        if (Label($N_k$) = none)
            continue
        if (Label($N_k$) = conflict)
            Label($N_j$)←conflict
            continue
        if (Label($N_j$) = none)
            Label($N_j$)←Label($N_k$)
        else
            if not (Label($N_j$) = Label($N_k$))
                Label($N_j$)←conflict
    // Propagate label to children
for(d←0; d$\leq$ max-depth; d←d+1)
    for each node $N_k$ in $ND_d$
        $N_j$ ←Parent($N_k$)
        if (Label($N_k$) = none)
            Label($N_k$)←Label($N_j$)
```

Figure 5.17: Algorithm for the marker propagation in the tree structure.

a user has defined two markers (dark and gray). The first step is to assign a marker to the nodes of the tree if it overlaps to a certain degree with its associated pixels. Then, the propagation process creates three connected components. Rather than pruning the tree, we show the region of support of the subtrees that have to be pruned.

Propagation processes on the Max-Tree can be used to extract bright objects with respect the background. In the case of Fig. 5.18, the head, shoulders and the screen on its right constitute bright objects and have been extracted by the propagation process. Note that dark objects can be segmented from the image by using the Min-Tree representation. Fig. 5.19 shows a second example of propagation on a Max-Tree: the gray marker is a connected component but the result of the propagation shows several connected components as region of influence of the gray marker. This is due to the fact that several markers (in this case gray and dark) have been set on the original image, and that the region of influence of some of the initially marked result as being in conflict with two (or more) markers. That is, some of the markers nodes have descendants that have been marked with a different marker, see Fig. 5.19a. The application of algorithm of Fig. 5.17 sets these nodes as in conflict removing thus the marker that was previously assigned to it, see Fig. 5.19.

In Fig. 5.20, an example using the Binary Partition Tree is shown. As before, a user has defined two markers (dark and gray). The first step consists in assigning the markers to the nodes in the tree. Note that in this case, by construction of the Binary Partition Tree, only the corresponding leaf nodes have to be marked (Fig. 5.20 on top). Then, the propagation process creates three connected components (Fig. 5.20 on bottom). The two first ones correspond to the zones of influence of the markers whereas the last one remains without label because it is judged as being "too different". As can be seen, the face and shoulder regions defined by the markers have been properly segmented and the background has not been merged with any of these regions.

Conflictive nodes

The propagation strategy presented in this section process does not assign a label to all regions of the image. In the example of Fig. 5.15, node N_4 remains without label. This situation means that the similarity between regions defined by markers A and B is higher than any combination with region R_4. As said above, region R_4 is indeed the most dissimilar. The propagation process is controlled in the sense that the algorithm does not blindly assign all regions to a marker. This type of control is attractive in most cases. However, for some specific applications, one would like to use a propagation algorithm that actually creates as many regions as markers.

In the case of the Max-Tree, there seems to be no solution leading to valid pruning strategy. The problem is due to the fact that the region of support of a node N_k in a Max-Tree is the union of the region of support of its children nodes plus a set of flat zones associated to the connected component of node N_k (see Sec. 2.1). In order to assign a label to all regions (i.e. flat zones) of the image the flooding algorithm should perform the propagation until the root node. A valid pruning strategy can be obtained only if one marker is initially placed on the tree.

In the case of the Binary Partition Tree a similar approach is taken. In this case the problem is to merge unassigned regions to one of the closest neighboring region that has been reached by a marker during the propagation. This task is easily solved with the help of the Binary Partition Tree. Indeed, consider an unassigned region R_k that has a sibling in conflict (regionn R_4 in Fig. 5.15). Its closest (in terms of the connectivity defined by the Binary Partition Tree) neighboring region that has been reached by a marker is one of the descendants of its sibling. Indeed, the set of sibling descendants is the set of closest homogeneous regions with respect to R_k. Furthermore, at least one of the descendants has

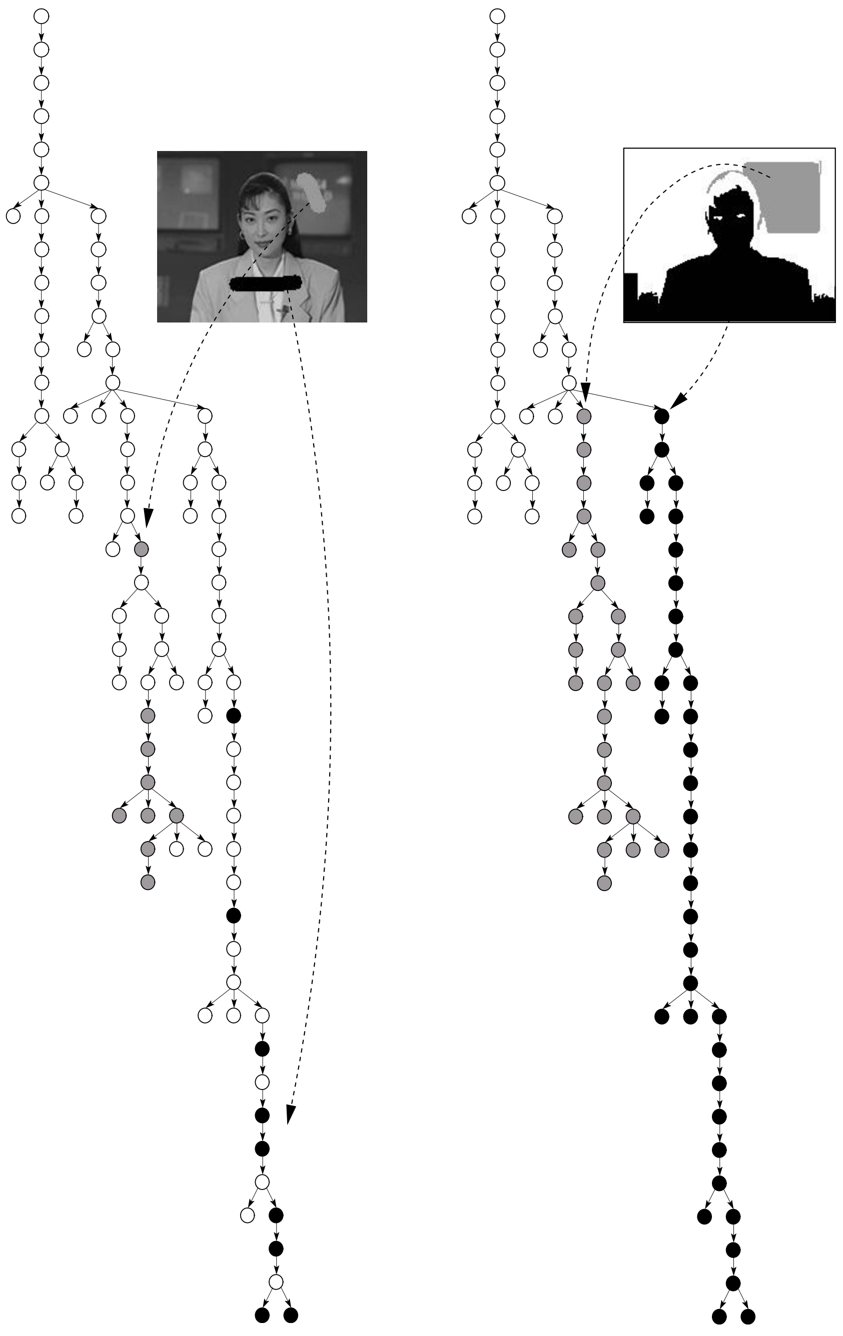


Figure 5.18: Example of a propagation process on the Max-Tree. Left, the nodes of the tree are marked according to the region the user has marked on the image. Right, result of propagation.

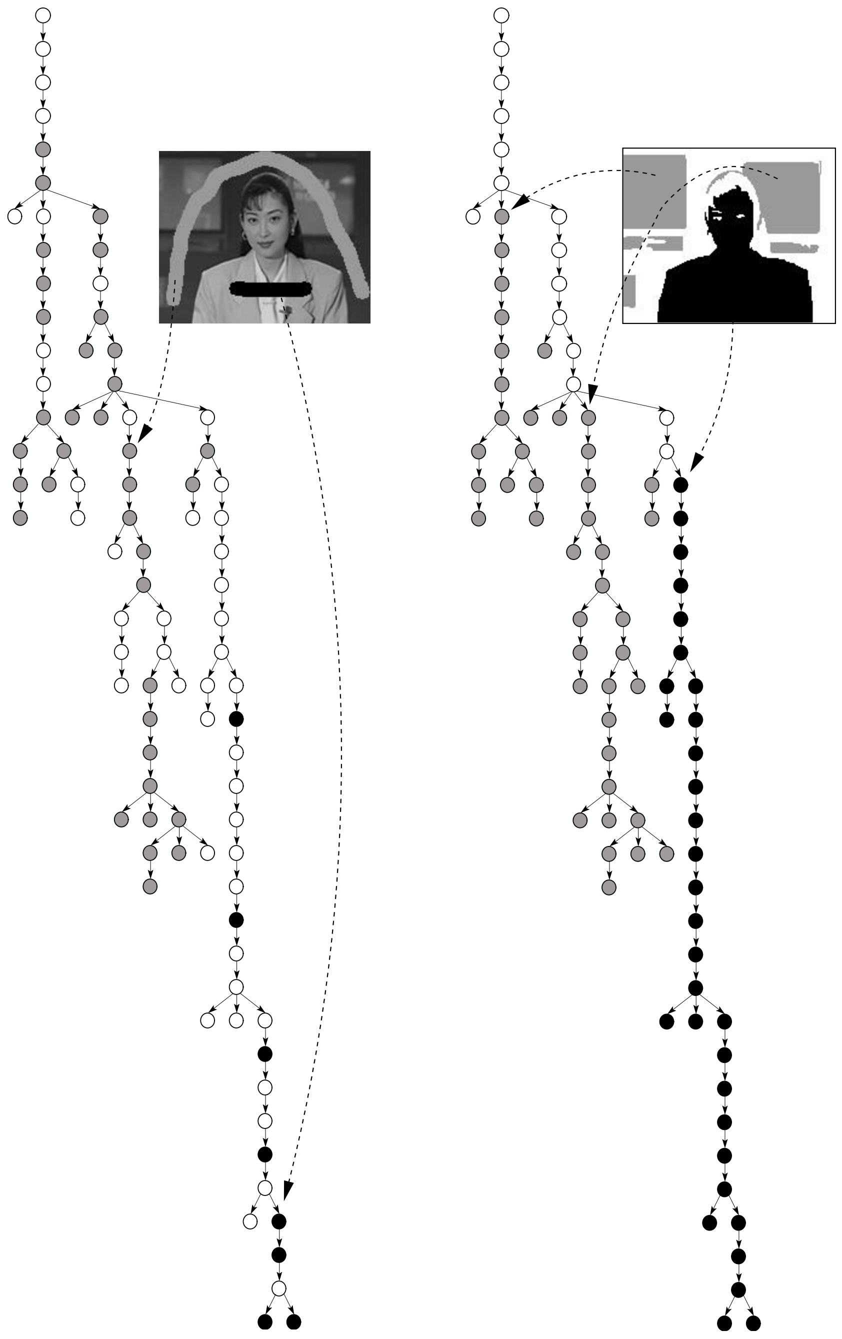


Figure 5.19: Example of a propagation process on the Max-Tree. Left, the nodes of the tree are marked according to the region the user has marked on the image. Right, result of propagation. Several disconnected components may be obtained even if the original marker is connected. See text for discussion.

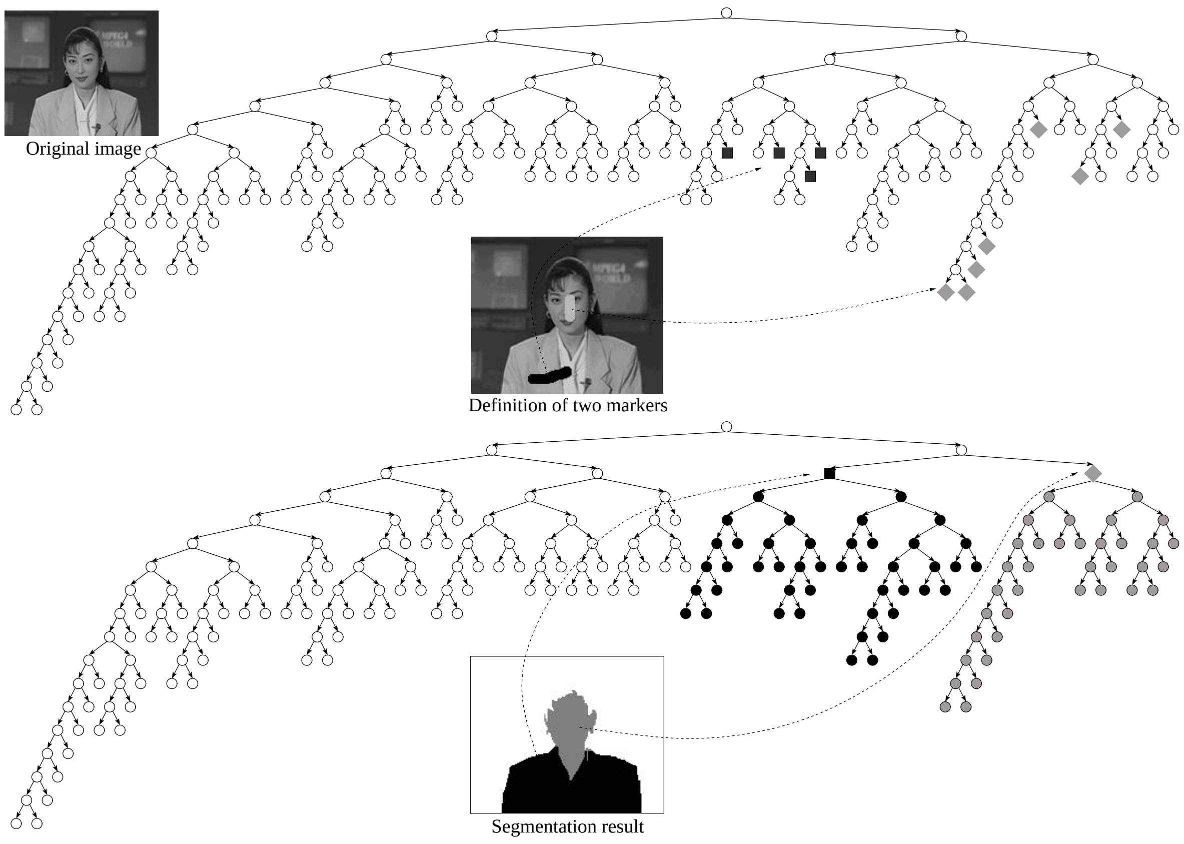


Figure 5.20: Example of a propagation process on the Binary Partition Tree. Top, the (leaf) nodes of the tree are marked according to the regions the user has marked. Bottom, result of propagation.

been assigned to a marker. Otherwise the propagation process could not have been stopped before reaching the sibling of R_k. Therefore, starting from the sibling of R_k, one simply has to scan all the descendants until one region that is, at the same time, neighbor of R_k and assigned to a marker, is found. Note that in some cases, several regions fulfill this criterion. This situation is illustrated in Fig. 5.15 where region R_4 was unassigned and two regions are at the same time neighbor of region R_4 and assigned to a marker: $R' = R_1 \cup R_2$ (assigned to marker A) and $R'' = R_3$ (assigned to marker B). In this situation, the most simple solution consists in arbitrarily selecting one of these regions. Of course, if necessary, specific rules based on similarity or geometrical criteria can be designed. In this work, the unassigned region is assigned to the region that shares with it the highest perimeter.

5.7.2 Area Filtering

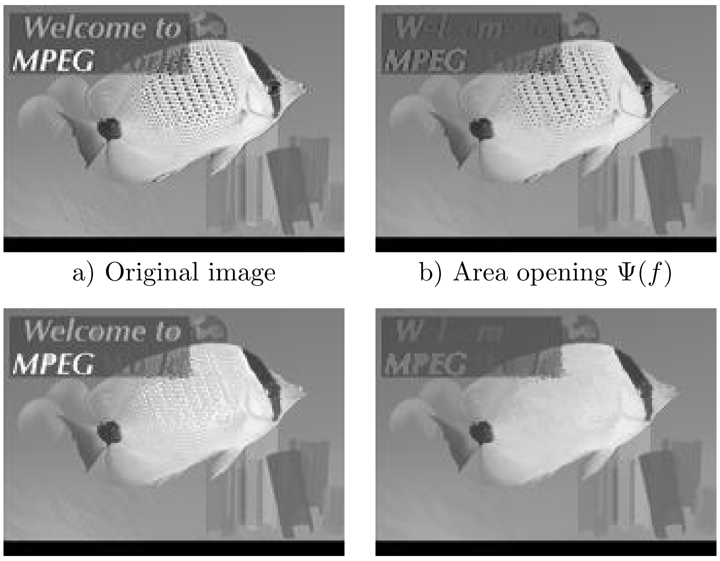
A simple criterion that can be used to prune the tree is based on measuring the size in pixels associated to each node. The criterion is defined to be $\mathcal{M}(N_k) = A_{R_k}$, where A_{R_k} is the area (in pixels) of the region R_k. The pruning then removes all nodes whose size is (strictly) below a threshold λ. The resulting filter has a size-oriented simplification effect. Note that the proposed criterion is increasing, and thus a valid pruning strategy is defined when applying the threshold.

Max-Tree

In the case of the Max-Tree, the effect of the filter is to remove all bright connected components whose size is below a threshold λ. The resulting filter is the well known area opening with parameter λ [93]. Filtering based on an area criterion results in an increasing, idempotent and anti-extensive filter.

Fig. 5.21 shows an example of area filtering. The area opening removes bright and small sized components of the image whereas the area closing removes dark and small sized components.

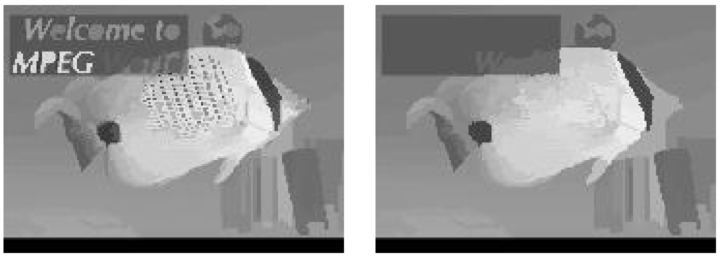
Note that the operator $\Psi(f)$ only acts on the regional maxima of the image. Once the regional maxima have been modified to fulfill the merging criterion, the operator does not modify the flat zones below this level. In the case of the area opening, all regional maxima of the filtered image have an area larger than the threshold. However, regional minima or transition areas can be of any size. Fig. 5.21b illustrates this situation: in the filtered image, a large number of flat zones (minima or transition regions) have a size smaller than 50. We will find a similar issue with the Binary Partition Tree in the next section.



a) Original image b) Area opening $\Psi(f)$

c) Area closing $\Psi^*(f)$ d) Area open-close $\Psi^*(\Psi(f))$

Figure 5.21: Example of area opening and closing of size $\lambda = 50$ applied on the Max and Min-Tree structure.



a) b)

Figure 5.22: Example of size-oriented simplification (size threshold to 50 pixels). a) Simple size simplification, b) Size simplification with propagation strategy

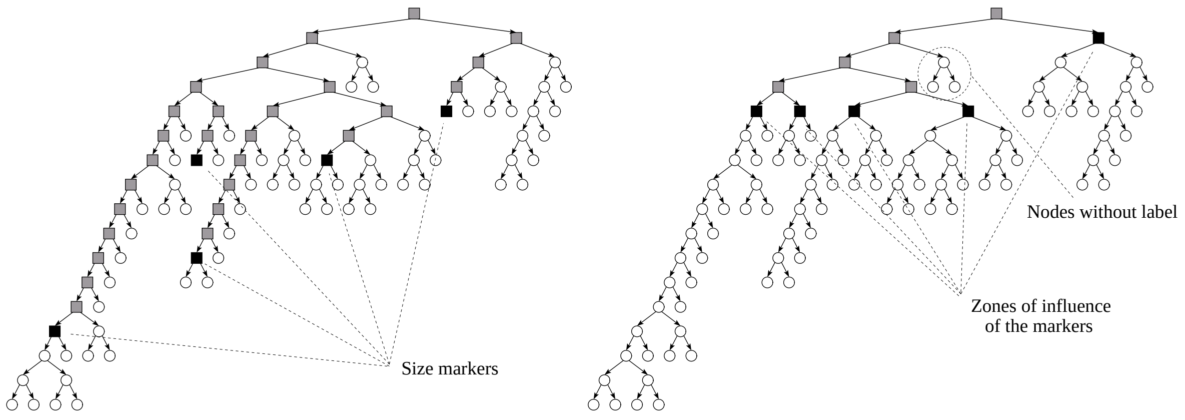


Figure 5.23: Size-oriented simplification. Left, Binary Partition Tree with size criterion. The black squares indicate the size markers. Right, definition of the zones of influence of the size markers.

Binary Partition Tree

A first example of size-oriented simplification is shown in Fig. 5.22a. The size threshold has been set to $\lambda = 50$ pixels. This result may be surprising because a large number of regions smaller than 50 pixels are still visible in the filtered image (the texture of the fish for example). To understand this result, let us analyze the example of Binary Partition Tree shown on the left side of Fig. 5.23 (note that this tree is presented here as a simple illustration. It is not the tree used to generate the example of Fig. 5.22). In this tree, one can see a large number of configurations where one node has to be removed whereas its sibling has to be preserved. Note that since the criterion is increasing, the parent of these two nodes has to be preserved. In terms of regions, this configuration means that one of the siblings as well as the parent correspond to large regions whereas the other sibling is of small size. Fig. 5.24 illustrates this issue on a very simple example: nodes N_1 and N_2 should be preserved because its associated area is large, whereas the size associated to node N_3 is small. As it has already been discussed, pruning a subtree (in our case N_3) on a Binary Partition Tree is done by merging its corresponding descendants. Pruning node N_3 results in a tree in which a new node appears at the position of N_3 and whose region of support is R_3. Thus, when reconstructing its associated partition a region R_3 is made visible on the partition image (see Fig. 5.24).

For certain applications, it may be necessary to force the operator to produce an output image where all flat zones are guaranteed to fulfill the simplification criterion. This modification can easily be implemented using the propagation process explained in Sec. 5.7.1 (page 96). The idea is explained in Fig. 5.23. The first step consists in defining the markers. These markers are all preserve leaves as well as preserve nodes that have two remove children.

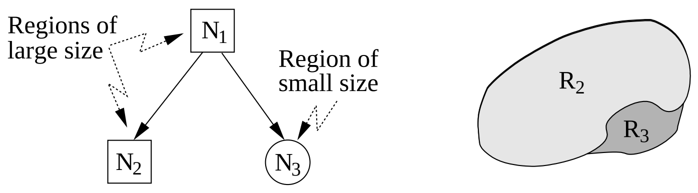


Figure 5.24: Illustration of decisions where a node has to be preserved whereas it's sibling has to be removed.

In the example of Fig. 5.23, there are five markers. The second step defines the filtered partition by propagating these markers as in the case of the segmentation described in Sec. 5.7.1. The result of the propagation is shown on the right of Fig. 5.23.

Note however, that the present maker and propagation strategy may lead to regions that remain without label (see Fig. 5.23). The associated nodes are in conflict: nodes for which the descendants of it sibling are assigned to different markers. In this kind of applications (filtering) we are interested in obtaining a partition with the same number of regions as the number of markers that have been set. As explained in Sec. 5.7.1, the approach that has been taken in our work is based on scanning the siblings descendants of the unassigned node for regions that have a label and are at the same time neighbor of the unassigned regions.

Fig. 5.22 right shows the result of this strategy on the Binary Partition Tree. A size-oriented simplification of the Bream image using this strategy is presented in Fig. 5.22b. All regions of size smaller than 50 pixels have been removed.

If not stated otherwise, the present technique (marker & propagation with resolution of possible nodes in conflict) is used in our work to filter a Binary Partition Tree after the analysis and decision process.

The present marker and propagation strategy is not useful for the Max-Tree. The reason is due the way the tree is defined and interpreted. Although each node represents a connected component from a level set, the problem has its origin in the flat zones. When the tree is reconstructed to a pixel based representation, the stacking of the connected components shows up the individual flat zones the image is made of (see Sec. 2.1). Flat zones are usually of small size (in most cases made up of one or two pixels). And by definition of the Max-Tree we are not able to control independently the gray-level value of each of the flat zones a connected component is made of. Thus, if we want all the flat zones to have a size greater than, for instance, 50 pixels, the whole tree – until the root – would be pruned in most cases.

5.7.3 Contrast Filtering

An operator widely used in the field of mathematical morphology is the so called contrast filter, also known as λ-max extraction. It is based on taking the image f, subtract a constant

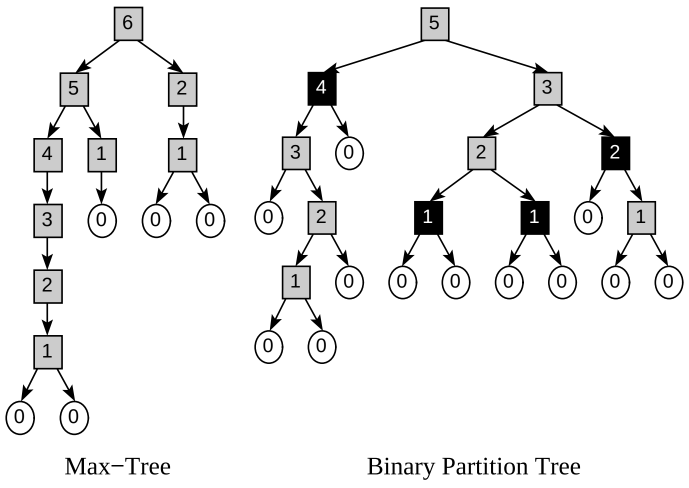


Figure 5.25: Example of contrast filtering applied on the Max-Tree (left) and Binary Partition Tree (right). At each node its associated height is indicated. Threshold is set to $\lambda = 1$. Nodes to be preserved (resp. removed) are depicted with squares (resp. circles). For the Binary Partition Tree, the black squares indicate the zone of influence of markers, see text.

λ to f, and proceed to reconstruct f by using $f - \lambda$ as marker [94]. The idea behind this operator is to extract the domes of high λ of the function f. A dome R of high λ is a connected component of pixels such that each neighboring pixel q of R (with $q \notin R$) satisfies $f(q) < \bigwedge \{f(p)|p \in R\}$; and $\bigvee \{f(p)|p \in R\} - \bigwedge \{f(p)|p \in R\} < \lambda$.

Max-Tree

By relating the structure of the Max-Tree and the implementation of the contrast filter via reconstruction we see that connected component R_k of the level set associated to gray-level h should be preserved if $\bigwedge \{f(p)|p \in R_k\} - h \geq \lambda$, and removed otherwise. The criterion that is to be used on the Max-Tree for a node N_k is

$$\mathcal{M}(N_k) = \bigwedge \{f(p)|p \in R_k\} - h \tag{5.7}$$

where R is the region of support of N_k, and h is the gray-level of the connected component associated to the node N_k. The resulting operator is anti-extensive, increasing but not idempotent. In this case the “empty nodes”, discussed in Sec. 3.1, have to be taken into account if we want the result of the Max-Tree filter to be equal to the λ-max filter. Note that the criterion of Eq. 5.7 is equivalent to measure the height of each node: leaf nodes have zero height, its parents have height one, and so on. The maximum height is associated to the root node N_{root}, $\mathcal{M}(N_{root}) = \bigvee \{f(p)|p \in E\} - \bigwedge \{f(p)|p \in E\}$.

An example of contrast filter is shown in Fig. 5.25 (left) for the Max-Tree. At each node, its associated height is indicated. Nodes to be removed are depicted with a circle, whereas

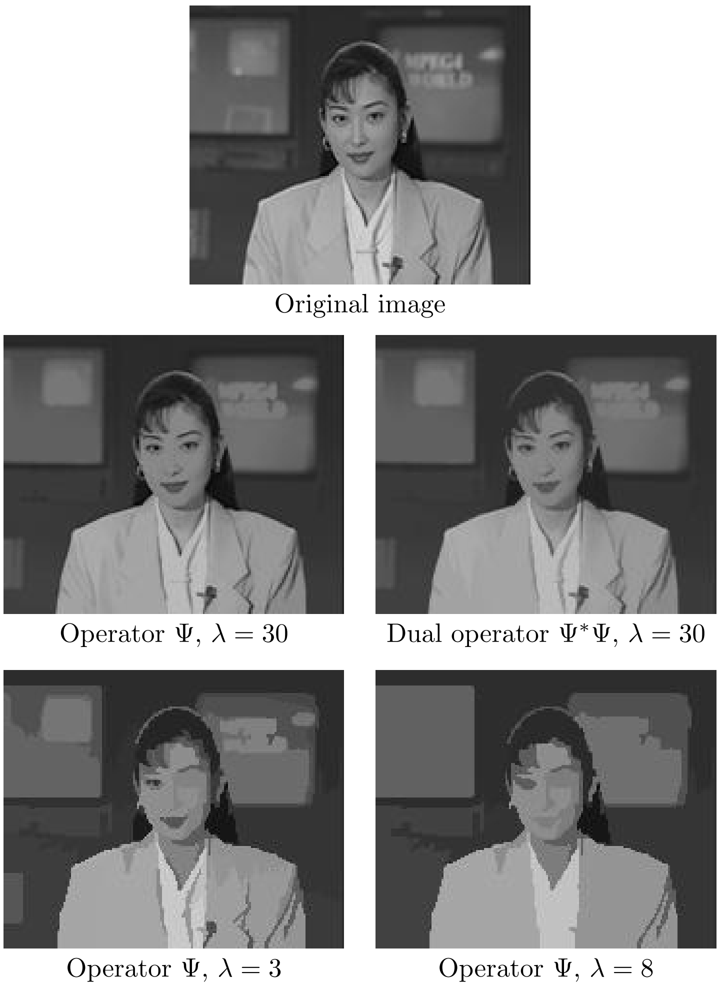

Original image
Operator Ψ, $\lambda = 30$
Dual operator $\Psi^*\Psi$, $\lambda = 30$
Operator Ψ, $\lambda = 3$
Operator Ψ, $\lambda = 8$

Figure 5.26: Example of contrast filtering. Original image shown on top, filtering with the Max-Tree and Binary Partition Tree are shown in the middle and bottom respectively.

those to be preserved are shown with a square. When pruning the tree, the nodes to be removed are merged with the first non removed ancestor. In Fig. 5.26 a second example is shown. The original image is shown on top. The result of the filtering is shown in the middle of Fig. 5.26. The Ψ operator extracts all bright domes of height 30: the effect is visible in homogeneous regions such as the face or the jacket of the woman. The dual operator Ψ^* extracts from the previous filtered image all dark domes of height 30. The contrast has been modified at the hair or the eyes of the woman. The results that are obtained via Max-Tree contrast filtering are exactly the same as the ones obtained with the reconstruction of f using $f - \lambda$ as marker.

Binary Partition Tree

The previous idea can be taken to the Binary Partition Tree structure. The criterion to be measured on each of the nodes is its associated height (see Sec. 2.2.1), and we then decide the pruning based on a threshold λ. The effect of the filter is equivalent to discard nodes associated to the mergings performed between the initial regions of the partition as the Binary Partition Tree is created. That is, if the Binary Partition Tree has been created using a color homogeneity criterion this filter discards some of the mergings performed by the merging algorithm. The resulting effect is thus to keep high contrasted (with respect its neighbors) regions. As before, the resulting filter is not idempotent.

Fig. 5.25 (right) shows the corresponding example of contrast filtering on a Binary Partition Tree. After deciding which nodes have to be removed or preserved, the marker & propagation strategy discussed in the previous section is applied. The resulting zones of influence are depicted with black squares. Pruning consists in merging its associated descendants. Fig. 5.26 (bottom) a second example. The original image is shown on top. This image is first partitioned into 500 regions. The Binary Partition Tree is created using a color homogeneity criterion (see Sec. 4.3.2). The result of applying the filter to the Binary Partition Tree is shown on the bottom for thresholds $\lambda = 3$ and $\lambda = 8$.

Discussion

By comparing the result with the one obtained with the Max-Tree, we see that – as for the area filtering – the Binary Partition Tree acts on bright and dark regions, whereas the Max-Tree acts only on bright objects: all other objects remain unchanged. As a result, the contours of the objects present in the filtered image with the Max-Tree are smooth in comparison to those of the image filtered with the Binary Partition Tree.

It should be noted that similar numerical values for the threshold do not necessarily provide similar simplification effects for the filter applied on the Max-Tree and Binary Partition Tree, respectively. In fact, the reason is twofold: first, the Binary Partition Tree is created using an initial partition to define the support of its leaf nodes. Thus, depending on the number of regions of the initial partition the resulting number of nodes of the Binary Partition Tree is different. The height of the tree (see Sec. 2.2.1) depends on the initial partition. Second, the Binary Partition Tree and Max-Tree have different definitions and interpretations: for the same image, both structures are generally dissimilar. Even if the Binary partition Tree is created from the partition of flat zones of the original image (which is the level of detail used to create the Max-Tree, see Sec. 3.1), both structures are not comparable.

Moreover, for the case of the Max-Tree one can distinguish small details in the resulting filtered image even if the threshold is rather high. For instance, the eyes or the microphone of the woman are visible in the filtered images of Fig. 5.26 (middle). This is due to the fact that the eyes (or the microphone) are a high contrasted region with respect to its neighboring regions. Formally, we may say that the numerical value of the dynamics [34] of these minima is high and thus a high threshold is needed to remove the latter regions from the image ($\lambda \approx 120$).

On the other hand, in the case of the Binary Partition Tree, the small high contrasted details are removed for low λ. In fact, when the Binary Partition Tree is created using the merging algorithm, the region of support associated to the microphone is rapidly defined since it is small and its associated regions of the initial partition are homogeneous in color. The height associated to the subtree associated to the region of support of the microphone is rather low and thus it can be removed with a low threshold. Notice that the eyes of the woman are smaller in size than the microphone: in the Binary Partition Tree, the height associated to the subtree of the eye of the woman is smaller than the one associated to the microphone. As a result, the minimum threshold λ needed to remove the eyes is lower than the threshold needed to remove the microphone (see Fig. 5.26). This is not the case for the Max-Tree: the eyes and the microphone have different sizes, but its dynamics [34] is similar (both regions are completely removed from the image for $\lambda \approx 120$).

As can be seen, the contrast filter applied on the Max-Tree has a contrast simplification effect which is "independent" of the size of of the regions. On the other hand, the contrast filter applied on the Binary Partition Tree also leads to a contrast simplification effect but is highly dependent on the area: small regions are always removed first "independently" of its contrast with its neighboring regions.

5.7.4 Complexity Filtering

The complexity of an object is an example of non increasing geometrical criterion combining size and shape criterion. The idea behind this operator is to remove complex objects. Intuitively, it can be seen that if a region R has a small area but a very long perimeter, it corresponds to a complex object. To this end, simplification criteria relying on the ratio between the area A_R and the perimeter ∂R can be used. For a region R

$$\text{Complexity}(R) = \frac{\partial R}{A_R} \qquad \text{Simplicity}(R) = \frac{A_R}{\partial R} \tag{5.8}$$

Note that this criterion is not the compactness [24] defined as the ratio between the area and the square of the perimeter. Interest in the simplicity criterion can be seen in segmentation based coding applications (for which it was designed). Indeed, in segmentation based coding, one has often to decide if a specific area of the image has to be segmented or not. In the first case, the contours of the region are sent to the receiver, and part of the coding cost is proportional to the length of the contour to code, that is the perimeter. In the second case, the area is considered as texture information, and its coding cost is generally proportional to

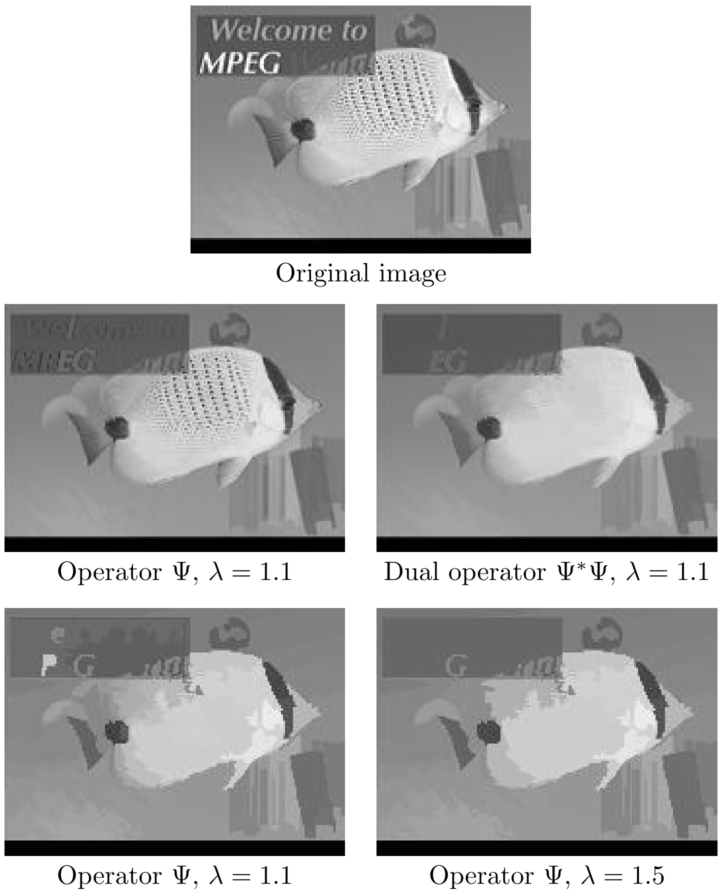

Welcome to
MPEG
Original image
Operator Ψ, $\lambda = 1.1$
Dual operator $\Psi^*\Psi$, $\lambda = 1.1$
Operator Ψ, $\lambda = 1.1$
Operator Ψ, $\lambda = 1.5$

Figure 5.27: Example of filtering with the simplicity operator. Top, original image. Middle, filtering using the Max-Tree. Bottom, filtering using the Binary Partition Tree.

its area. As can be seen, the simplicity operator allows the classification of objects following a contour/texture cost.

In the following, we will use the simplicity criterion, $\mathcal{M}(N_k) = \text{Simplicity}(R_k)$, because it agrees with the node removal criterion of Eq. 5.3 (page 82): remove all the nodes whose simplicity is low (i.e. high complexity). After deciding which nodes are removed and which preserved, the optimization algorithm described in Sec. 5.5 is applied to obtain a valid pruning strategy.

Max-Tree

On the Max-Tree, the simplicity operator removes complex and bright objects from the original image. As usual, a dual operator dealing with dark objects can be defined. An example of processing can be seen in Fig. 5.27. The original image is composed of various objects with different complexity. In particular, the text and the texture of the fish can be considered as being complex in comparison with the shape of the fish and the books on the lower right corner. Fig. 5.27 (middle) shows the output of simplicity operator and its dual. The global processing can be considered as an alternated operator. As illustrated on this example, the simplicity operators efficiently remove complex image components (text and texture of fish) while preserving the contours of the objects that have not been eliminated. Note that the simplification effect is not size-oriented, because the filters have removed large objects (the letters “MPEG”) as well as small objects (the texture of the fish). The simplification is not contrast-oriented as can be seen by the difference in contrast between “Welcome to” and “MPEG” which have been jointly removed.

Binary Partition Tree

An example of processing with the Binary Partition Tree can be found in Fig. 5.27. The Binary Partition Tree has been created using a color homogeneity criterion and an initial partition made up of 500 regions. As in the case of the Max-Tree, the operator is able to remove efficiently complex objects of the image, such as the texture of the fish and the text on top. The “G” has not been completely removed since it is connected to the background.

Discussion

For both tree structures, the filter removes complex objects independent of their size. The criterion is, in fact, a non-increasing criterion. However, note that the application of the optimization criterion does not ensure that all preserved nodes have a simplicity higher than the specified threshold.

As for the contrast filter, note that the result obtained with the Binary Partition Tree leads to images with highly contrasted contours between regions: each region of the associated partition has been filled with its mean gray-value. On the other hand, filtering with the Max-Tree produces smooth contour transitions between the objects present in the image. This produces smooth textures in the resulting filtered image.

As opposed to the case of the contrast filter, in this case the threshold value can be used to compare the simplification effect for the Max-Tree and the Binary Partition Tree. This is due to the fact that the criterion depends on the support of the region rather than on the structure of the tree. From our experimental results, similar thresholds produce similar simplification effects in both cases.

The resulting operator is (for both Max-Tree and Binary Partition Tree) not increasing, due to the non-increasing criterion, and not idempotent, due to the optimization algorithm: the result of the optimization depends on the decisions taken for all nodes in the branch that is being analyzed. An image and its filtered version have different tree structures. Thus, it is not possible to ensure that $\Psi(\Psi(f)) = \Psi(f)$.

5.7.5 Motion filtering

In this section, the criterion deals with the motion information in image sequences. Denote by $f_t(p)$ an image sequence where $p = (p_x, p_y)$ represents the coordinate of a pixel and t the time instant. Our objective now is to define a filter able to eliminate the image regions R that do not undergo a given motion. The first step is therefore to define the motion model giving, for example, the displacement field at each position $\Delta(p) = \{\Delta_x(p_x, p_y), \Delta_y(p_x, p_y)\}$. The field can be constant $\Delta(p) = \{\Delta_x, \Delta_y\}$ if one wants to extract all objects following a translation, but in general the displacement can depend on the spatial position (p_x, p_y) to deal with more complex motion models such as affine or quadratic models.

The sequence processing is performed as follows: each frame is transformed into its corresponding tree representation and each node N_k is analyzed. To check whether or not the pixels contained in a given node N_k are moving in accordance to the motion field $\Delta(p) = \{\Delta_x(p_x, p_y), \Delta_y(p_x, p_y)\}$ a simple solution consists in considering the region associated to N_k, R_k, and to compute (the opposite of) the Mean Displaced Frame Difference, $\overline{DFD}$, of this region with the previous frame. Note that the opposite of the mean DFD is used so that the criterion value for a region that has to be preserved is higher than the corresponding value when the region has to be removed (see Eq. 5.3). The criterion can be expressed as [73]:

$$\overline{DFD}_{f_t}^{f_{t-1}}(R) = \frac{-\sum_{p \in R} |f_t(p) - f_{t-1}(p - \Delta(p))|}{A_R} \tag{5.9}$$

In practice, however, it is not very reliable to state on the motion of part of the image on the basis of only two frames. The criterion should have a reasonable memory of the past decisions. This idea can be easily introduced in the criterion by adding a recursive term. Two mean DFDs are measured: one between the current frame f_t and the previous frame f_{t-1} and a second one between the current frame and the previous filtered frame $\Psi(f_{t-1})$, where Ψ denotes the filter. The motion criterion is finally defined as:

$$\mathcal{M}(R) = \alpha\, \overline{DFD}_{f_t}^{f_{t-1}}(R) + (1-\alpha)\overline{DFD}_{f_t}^{\Psi(f_{t-1})}(R) \tag{5.10}$$

where $0 \leq \alpha \leq 1$. If α is equal to 1, the criterion is memoryless, whereas low values of α allow the introduction of an important recursive component in the decision process. In a way similar to all recursive filtering schemes, the selection of a proper value for α depends on the application: if one wants to detect very rapidly any changes in motion, the criterion should be mainly memoryless ($\alpha \approx 1$), whereas if a more reliable decision involving the observation of a larger number of frames is necessary, then the system should rely heavily on the recursive part ($0 \leq \alpha \ll 1$).

The motion criterion described by Eqs. 5.9 and 5.10 deals with a particular set of motion parameters. Objects that do not follow the given motion are removed. For some applications, it may be useful to preserve objects that are within a given range of motion (notion of

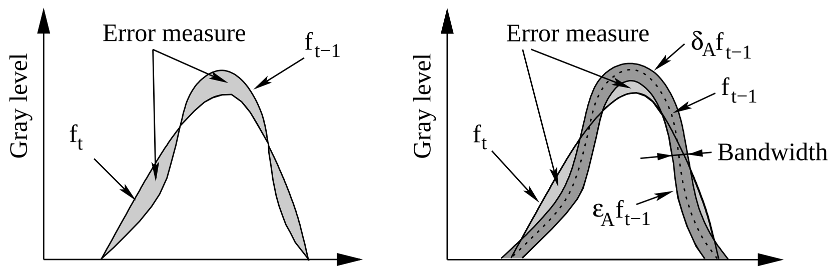


Figure 5.28: Motion bandwidth concept illustration. Left, criterion for one motion parameter (zero bandwidth). Right, criterion for a range of motion (bandwidth > 0). The error measure associated to the use of a motion bandwidth is lower than the error associated to the null motion bandwidth.

"motion bandwidth"). To this end, the criterion of Eq. 5.9 can be modified by introducing an erosion ϵ_A and a dilation δ_A of the previous frame, where A represents the structuring function. The difference $|f_t(p) - f_{t-1}(p - \Delta(p))|$ in the $\overline{DFD}$ of Eq. 5.9 is replaced at each point p either by $f_t - \delta_A(f_{t-1})$ if $f_t > \delta_A(f_{t-1})$, or by $\epsilon_A(f_{t-1}) - f_t$ if $f_t < \epsilon_A(f_{t-1})$, or by 0 if $\delta_A(f_{t-1}) \leq f_t \leq \epsilon_A(f_{t-1})$. This approach is illustrated in Fig. 5.28. As can be seen, the erosion and the dilation of f_{t-1} create a "tube" in which the function f_t can remain without contributing to the $\overline{DFD}$. The size of the structuring function used in the dilation and the erosion defines the motion "bandwidth". Note that in Eq. 5.10 the motion "bandwidth" is created for f_{t-1} and $\Psi(f_{t-1})$.

Max-Tree

A first motion filtering example with the Max-Tree is shown in Fig. 5.29–5.32. The objective of the operator is to remove all moving objects. The motion model is defined by: $(\Delta_x, \Delta_y) = (0, 0)$. In this sequence, all objects are still except the ballerina behind the two speakers and the speaker on the left side who is speaking. The application of the connected operator $\Psi(f)$ described previously removes all bright moving objects (Fig. 5.30). The application of the dual operator $\Psi^*(f)$ removes all dark moving objects (Fig. 5.31). The residue (that is the difference with the original image) presented in Fig. 5.32 shows what has been removed by the operator. As can be seen, the operator has very precisely extracted the ballerina and the (moving) details of the speaker's face. Moreover, the results are robust in front of slight changes of the threshold.

The residue for the case from filtering the sequence using the maximum decision using the same filtering parameters is very similar to the result obtained with the optimum decision. This means that the Viterbi algorithm has resulted in most of the cases in a decision tree similar to the decision obtained with the maximum decision (see also Sec. 5.6).

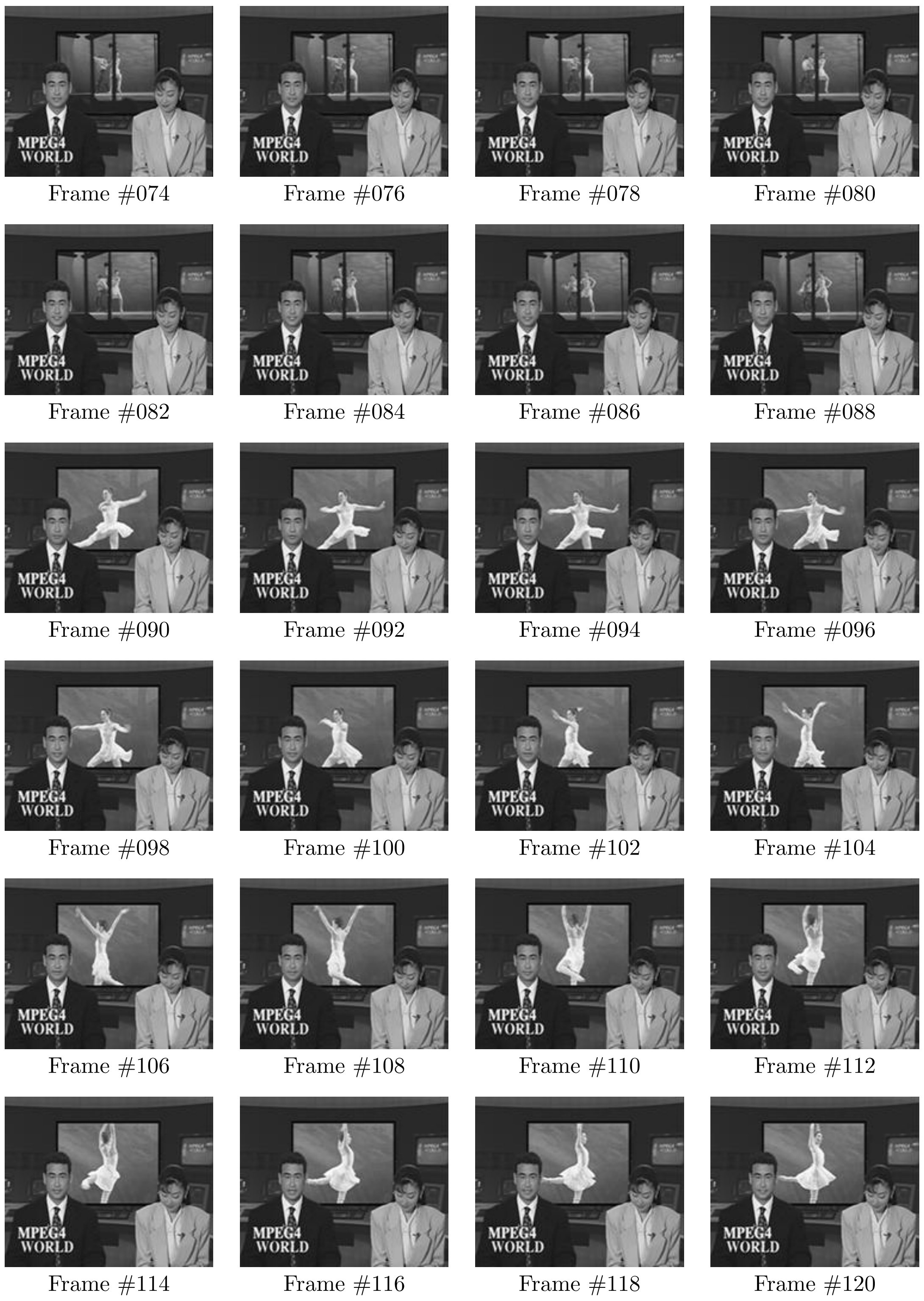


Figure 5.29: Original sequence for the motion filtering.

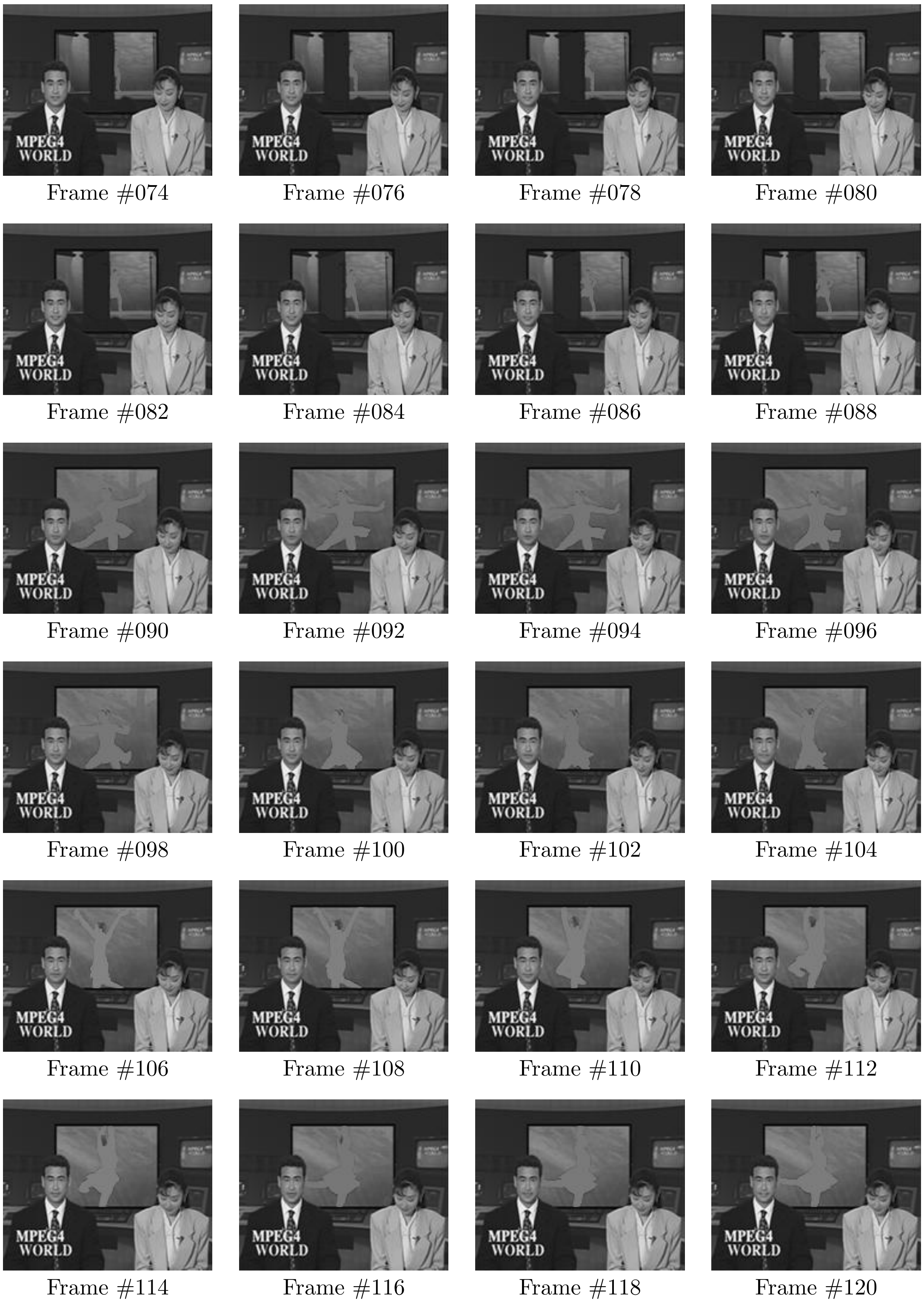


Figure 5.30: Example of motion filtering preserving fixed objects applied on the Max-Tree. Motion filter $\Psi(f)$.

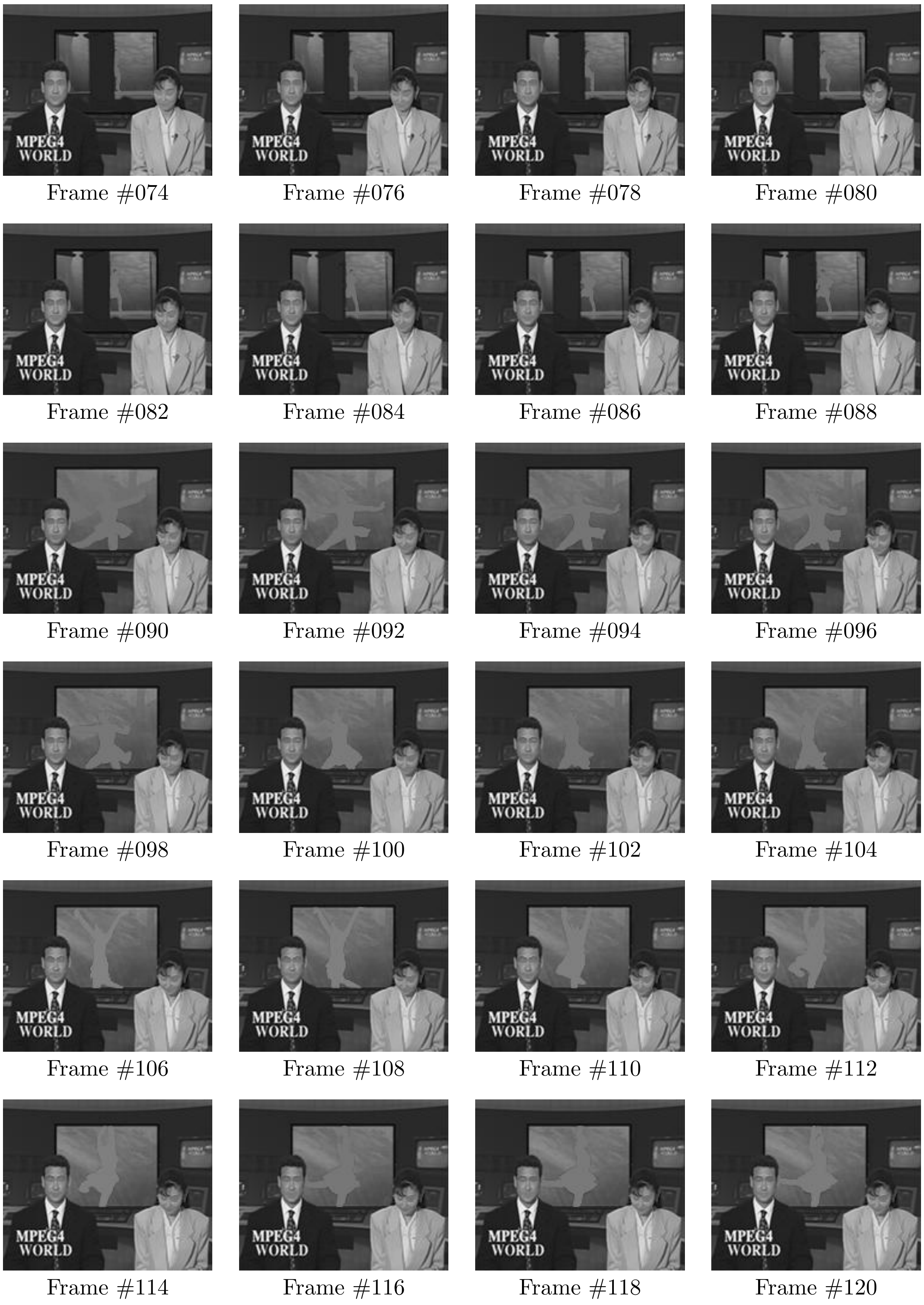


Figure 5.31: Example of motion filtering preserving fixed objects applied on the Min-Tree. Motion dual operator $\Psi^*(\Psi(f))$.

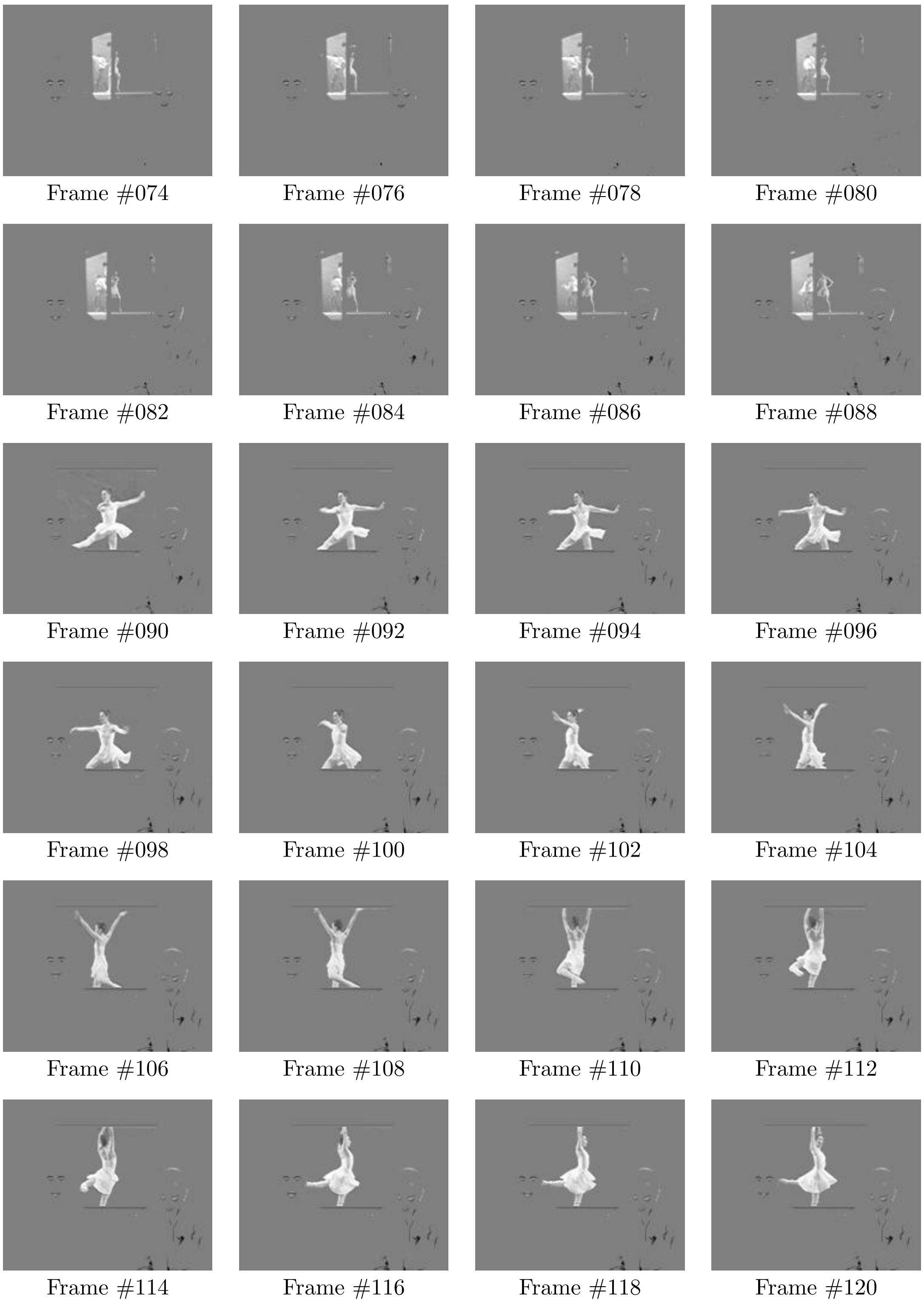


Figure 5.32: Example of motion filtering preserving fixed objects, see Fig. 5.31. Residue $f - \Psi^*(\Psi(f))$.

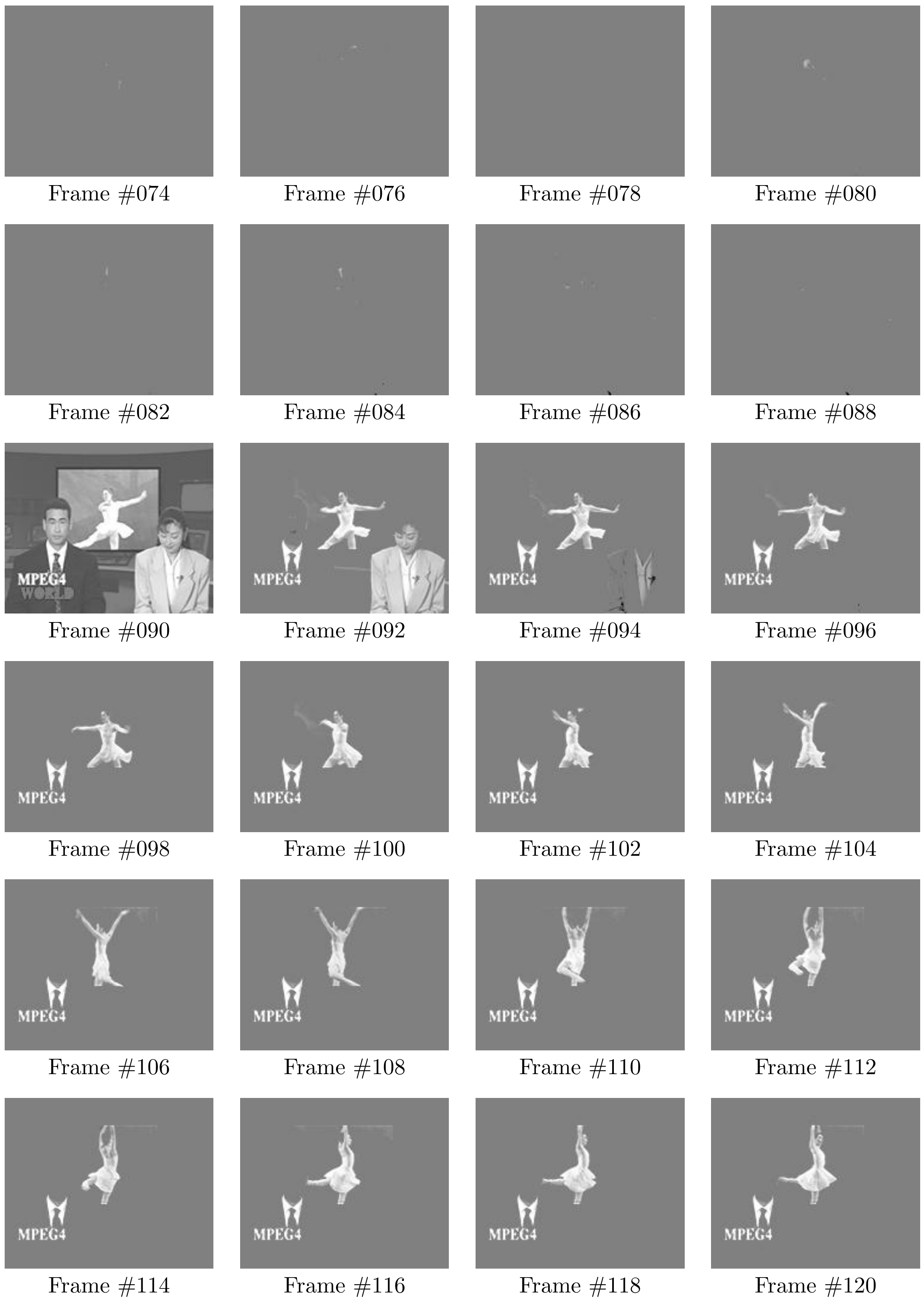


Figure 5.33: Example of motion connected operator preserving fixed objects. Residue $f - \Psi^*(\Psi(f))$ if minimum decision is used for operators Ψ and Ψ^*.

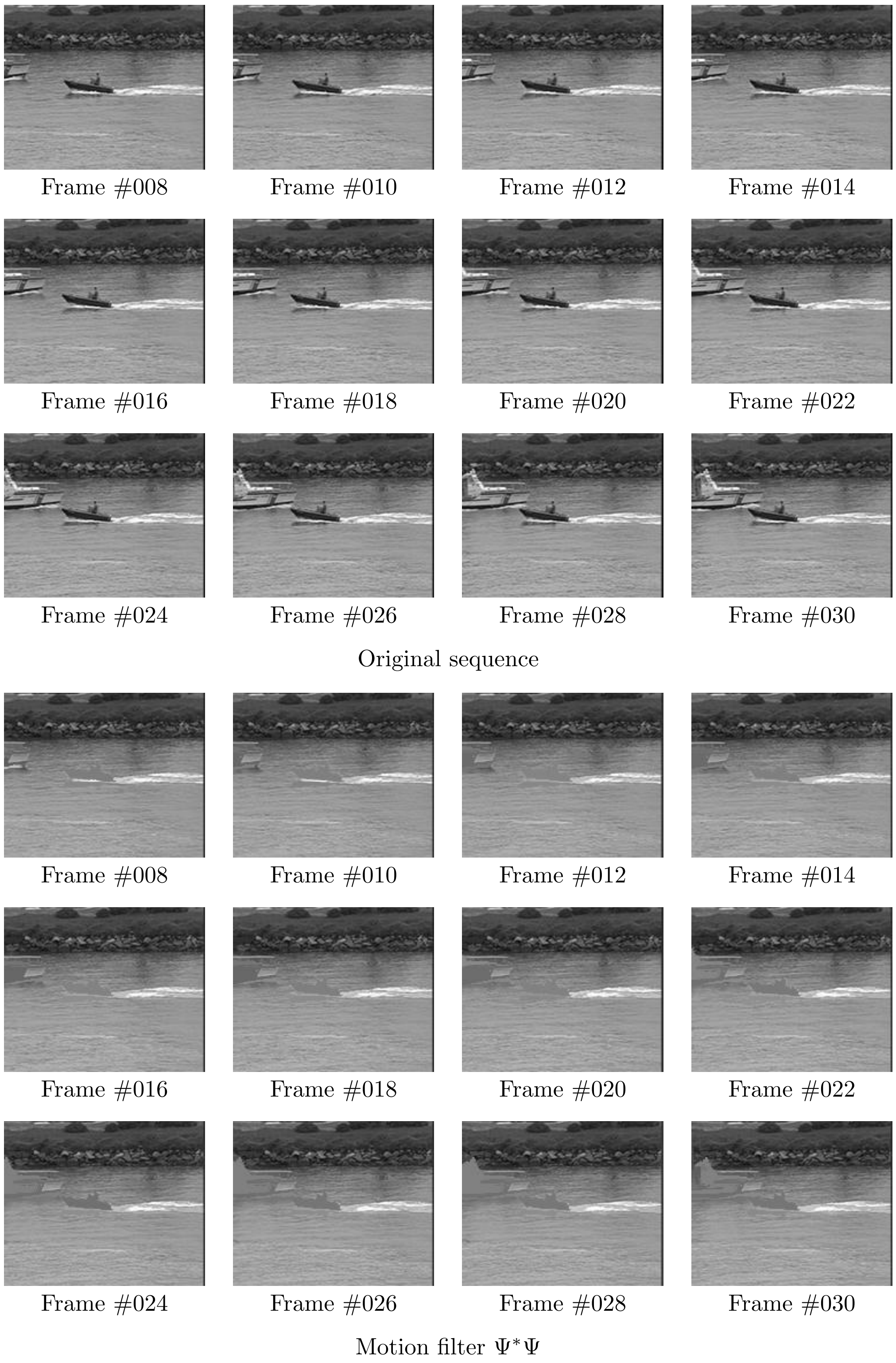


Figure 5.34: Example of a decomposition of a sequence into three sequences. Top, original sequence. Bottom, objects following the dominant motion $(\Delta_x, \Delta_y) = (2, 0)$ are extracted. See also Fig. 5.35.

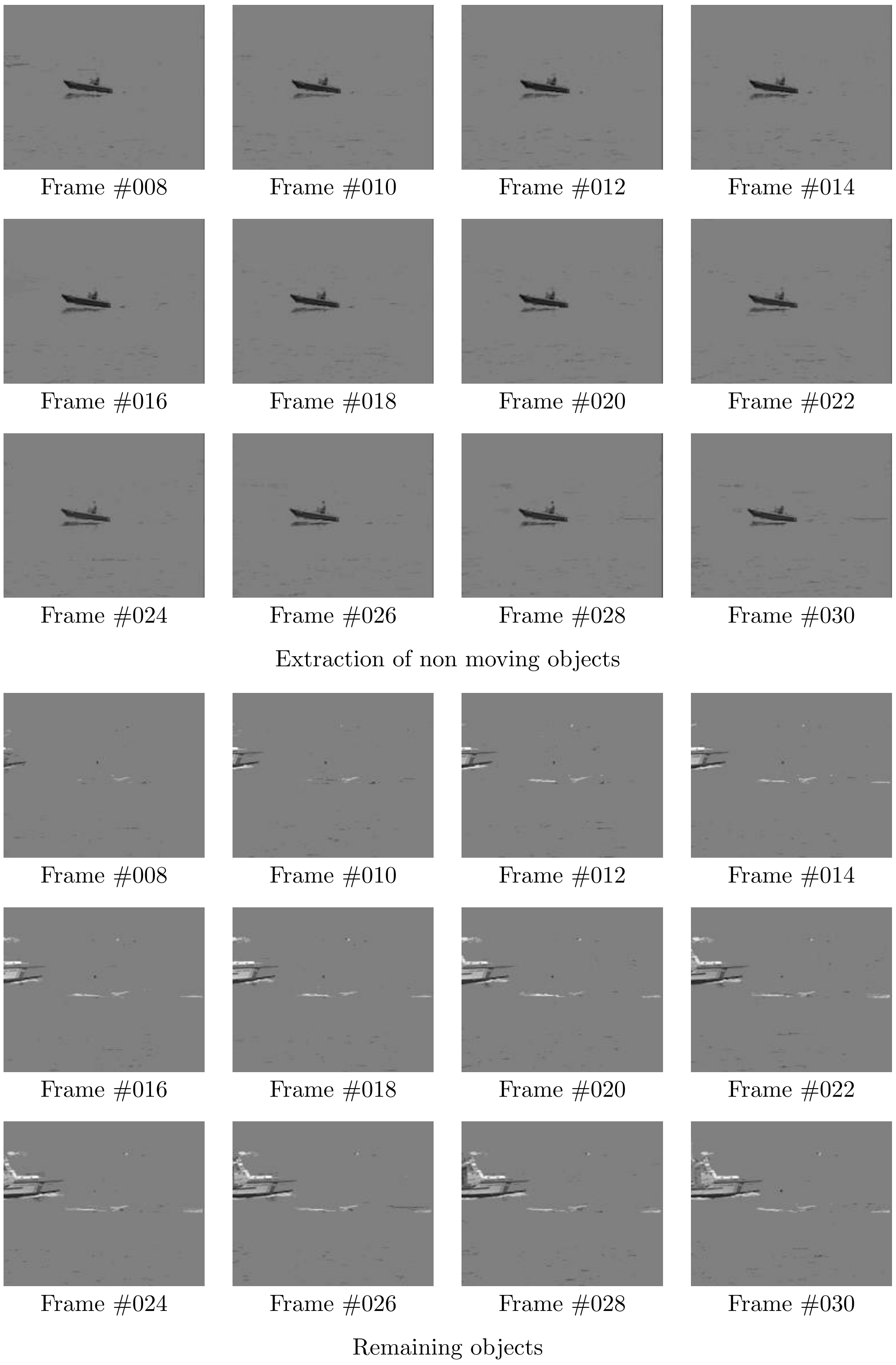


Figure 5.35: Example of a decomposition of a sequence into three sequences. Top, extraction of non moving objects from the residue of the sequences shown in Fig. 5.34. Bottom, remaining objects.

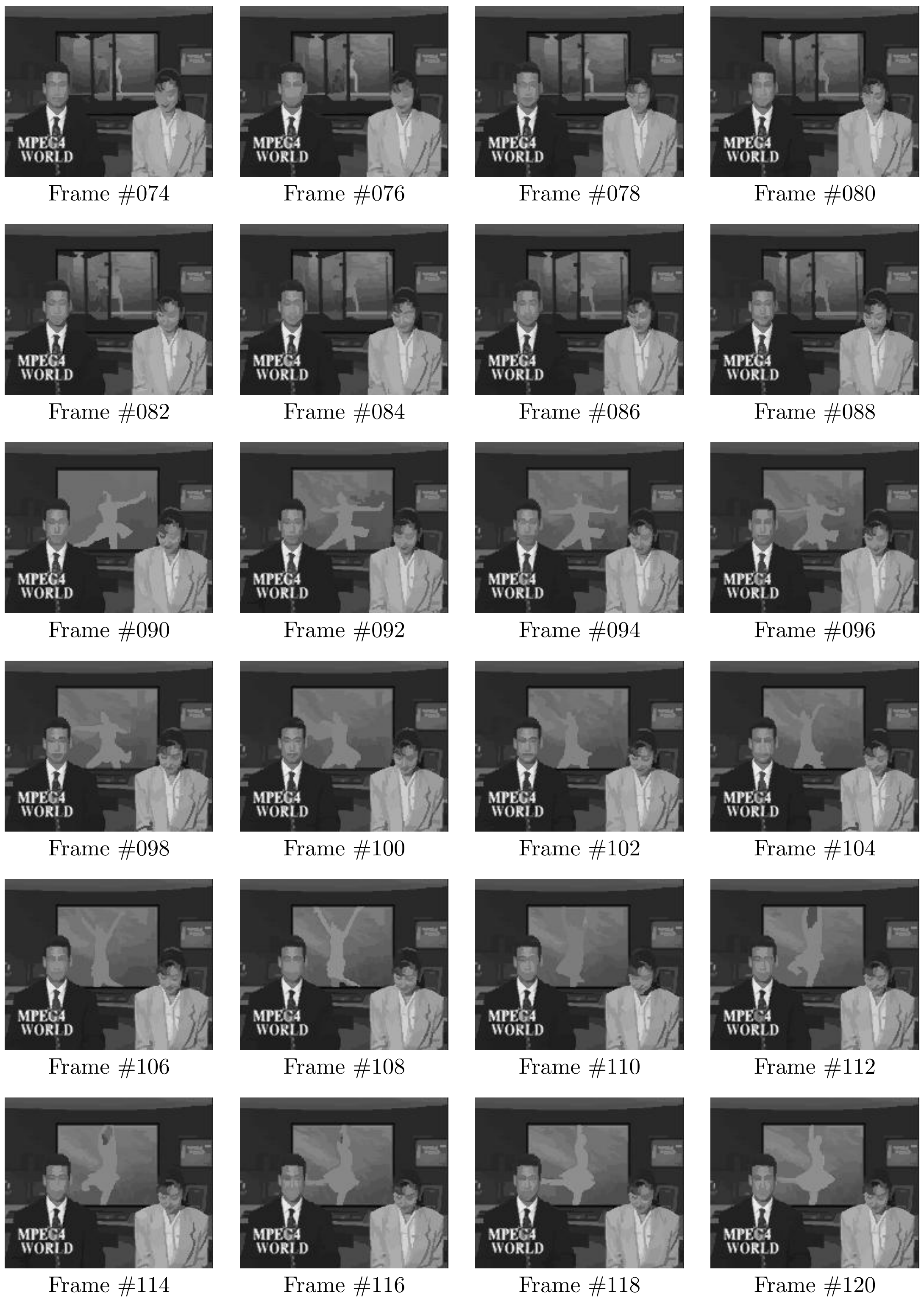


Figure 5.36: Example of motion connected operator preserving fixed objects. Filter using Binary Partition Tree.

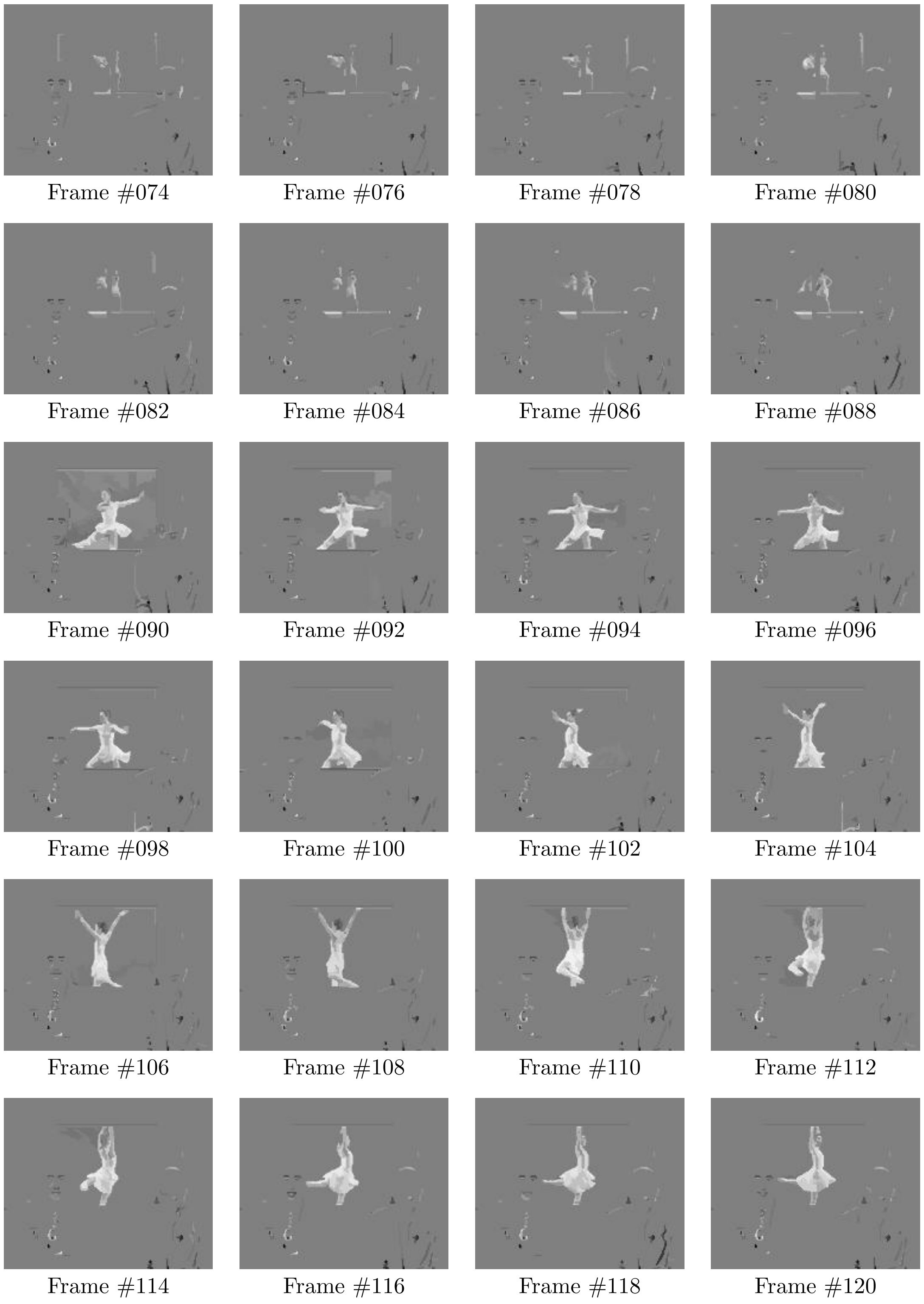


Figure 5.37: Example of motion connected operator preserving fixed objects. Residue between the original (Fig. 5.36) and filtered (Fig. 5.29) sequence.

For the case of the minimum decision the resulting images are black if filtering parameters remain unchanged with respect to the ones used for the optimum case. This result is due to the fact that some nodes near the root have a motion criterion below the threshold: in this case, all the descendants of such nodes are removed producing thus black images. Additionally, slight changes in the threshold result in drastic changes on the output. Thus, in order to use the minimum decision the filtering parameters have to be adequately set. In Fig. 5.33 the residue from filtering the sequence using a minimum decision is shown. As before, first the motion operator is applied and on the result the dual motion operator is computed. Note that a lot of regions appear in frames #90-#94. The reason is due to the shot transition that the sequence performs from frame #89 to frame #90. As a result, when the motion filter is applied, some nodes near the root are set to be removed and thus the residue shows the whole image. Note that the letters "MPEG4" and the shirt of the news reader appear on the residue through the remaining frames. This is due to the memory of the filter: these regions are removed since they were removed in previous images. The memory of the filter may also produce this type of effects in the case of the optimum decision. In fact, this problem may appear with a sudden movement or scene change in the sequence. A simple solution to avoid the mentioned effects could be based, for instance, on an adaptive value of α: when a scene change or sudden movement is detected, its value is set to a low value. Otherwise, α is set to the desired value.

The example illustrated in Fig. 5.34 and 5.35 shows a decomposition of the original sequence into three sequences. The original sequence shows two boats on a river. The camera is following the black boat in the center. Therefore, the river and the background have an apparent motion (called the dominant motion), whereas the black boat is still. First the dominant translation is estimated giving the following motion model $(\Delta_x, \Delta_y) = (2, 0)$. Objects following this translation are obtained by application of the motion operator followed by its dual. As can be seen in Fig. 5.34 bottom, the background and the river regions are obtained. Then, the difference between the original frame and the filtered frame is computed. This difference involves only the two boats. On this residue still objects $(\Delta_x, \Delta_y) = (0, 0)$ are extracted. As shown in Fig. 5.35 top, the black boat has been extracted. Finally, the remaining components are shown in Fig. 5.35 bottom. This is a decomposition in the sense that the sum of the three sequences restores the original sequence. As can be seen, the filtering has separated the background and the two boats moving in two different directions.

Binary Partition Tree

Fig. 5.36 shows a motion filtering example using the Binary Partition Tree. The original sequence is shown in Fig. 5.29: each image has been partitioned into 500 regions and the Binary Partition Tree has been created with a color homogeneity criterion. The regions associated to the leaves of the tree provide fine details of the image and thus the operator

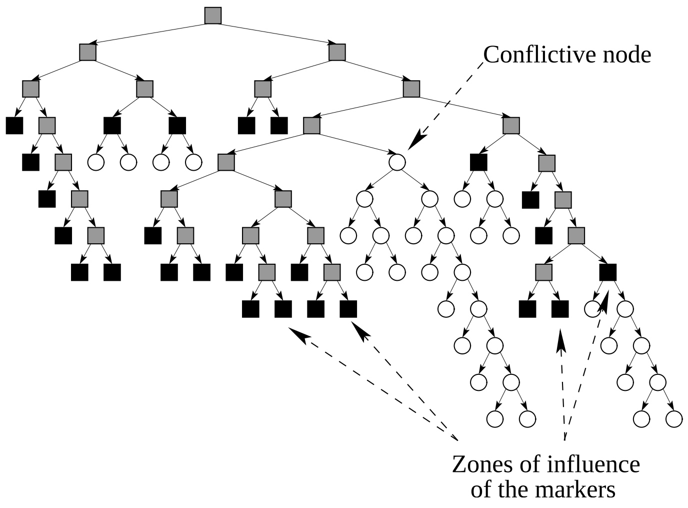


Figure 5.38: Example to show the strategy used to reconstruct the pixel based representation associated to the Binary Partition Tree for the motion filter. The figure shows the definition of the zones of influence after applying the maker & propagation strategy for the motion criterion. Nodes to be preserved (resp. removed) are depicted with a gray square (resp. white circle). Black squares indicate zones of influence of markers. See text for discussion.

will be able to detect small moving details.

As in the case of the Max-Tree, the purpose is to extract all moving objects from the sequence. For that purpose, the criterion is assessed on each node and the optimization algorithm is applied to obtain a valid pruning strategy. For the case of the motion filter, in this case the marker & propagation strategy is applied without resolving the conflicts (see Sec. 5.7.2). The pruning is then applied (see Sec. 5.2.1), and the color pixel based representation is reconstructed as follows (see Fig. 5.38): the regions associated to the zones of influence of the markers are filled up with the mean color of its region of support, otherwise the region is reconstructed using the mean color associated to the region of support of its parent. In the latter case the regions correspond to nodes that are in conflict.

This technique has been used since, from our experimental results, the use of the marker & propagation strategy with conflict resolution does not provide good visual results. Our purpose is to obtain an image in which the filtered regions are clearly visible. If conflict resolution is used, regions associated to conflictive nodes would be merged with neighboring regions and thus it would not be possible to distinguish exactly which are the moving regions. An example is shown in Fig. 5.39. On top, reconstruction using conflict resolution is shown. In this case, the ballerina corresponds to a conflictive node. With the conflict resolution, this region would be merged with a neighboring region. Note that the contours of the removed object (the ballerina) are not visible.

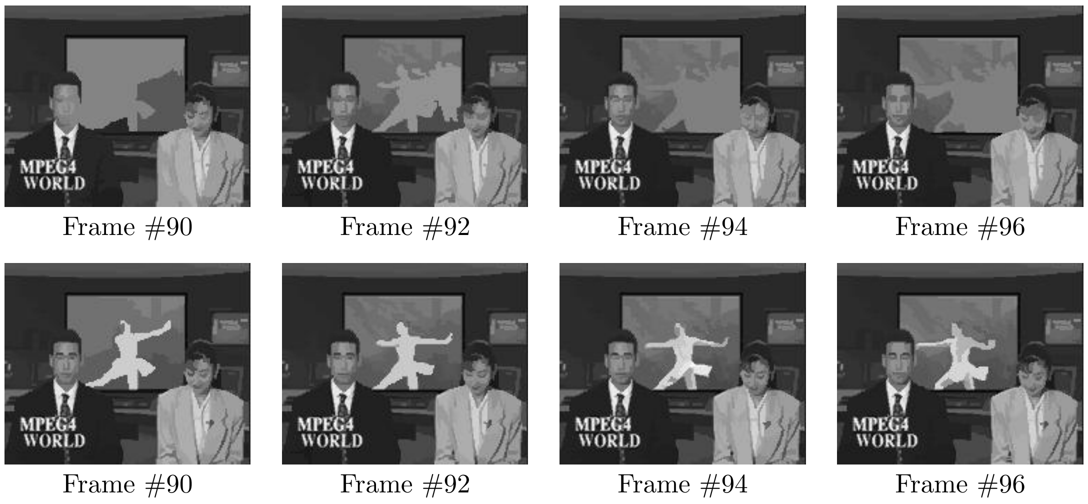


Figure 5.39: Example illustrating visually non desired results for different types of pixel based reconstruction strategies. Top: reconstruction with conflict resolution, each region is filled up with its mean color. Bottom: Reconstruction without conflict resolution and filling each region with its mean color (instead of using parent color for the conflictive regions).

The reason of using the mean color of the support of the parent to reconstruct some of the nodes is due to the fact that with this strategy the nodes that have been pruned are visually “more” visible. For instance, let us take the ballerina (which is a subtree to be pruned): if the region of support of the ballerina is reconstructed using its associated mean color, the visual effect would be a bright colored region. In the resulting image it would be difficult to recognize immediately that the ballerina has been removed (see Fig. 5.39 on bottom). Thus, in other to produce a better visual effect, the previously described approach has been used.

Fig. 5.37 shows the residue between the original and filtered sequence. It indicates the regions that have been removed by the operator. As in the case of the Max-Tree, moving objects have been precisely extracted. The results shown in Fig. 5.37 are quite similar to those presented in Fig. 5.32 for the Max-Tree. This is due to the fact that in this case the moving objects are mostly bright or dark with respect its background. These objects are represented as a node in both the Max-Tree and the Binary Partition Tree (since they are easily to segment using color homogeneity criterion). The application of the motion filter results thus in similar results.

Discussion

The motion operator can potentially be used for a large set of applications. In particular, it opens the door to different ways of handling the motion information. Indeed, generally, motion information is measured without knowing anything about the image structure. The filter takes a different viewpoint by making decisions on the basis of the analysis of a set of meaningful regions of the image. By using motion connected operators, we can "inverse" the classical approach to motion and, for example, analyze simplified sequences where objects are following a known motion. In [75], a robust background motion estimation strategy is presented. The approach is based on an iterative process. First, a region based motion estimation [18, 78] is performed between two frames using (at the beginning of iteration) the whole image support. An approximation of the dominant motion is obtained: the estimation is influenced by the motion of objects that do not follow the dominant motion. The motion filter is then used to filter out all objects that do not follow the estimated dominant motion. As a result, a mask of the outliers (regions not following dominant motion) is obtained. The whole process is iterated by estimating the motion taking into account the information of the outliers.

5.7.6 Rate & Distortion Browsing

The previous sections involves the evaluation and the optimization of a local criterion independently on each region of the tree. By contrast, the following browsing example discusses an approach where the optimization is global on the entire tree structure.

Browsing is an important functionality for information retrieval. Most of the time, the user would like to have a rough idea on the query results. The goal is not to visualize a high quality image, but simply to be able to discard or not the query result. This issue is not trivial if the transmission channel between the client and the server has a reduced bandwidth. The Binary Partition Tree is a very attractive representation to deal with such a functionality. Indeed, as shown in [72], partition trees in general are appropriate for defining optimum pruning strategies in the rate/distortion sense with restriction on the rate to be transmitted or the distortion of the coded image. Let us discuss this approach.

Assume that the visual information is transmitted by selecting some regions described by the Binary Partition Tree and by sending their contours plus a constant color value per region. The definition of the coding strategy consists in finding the best partition created by regions, R_i, contained in the tree such that the global distortion, $\mathcal{D}$, is minimized and the rate or coding cost, $\mathcal{R}$ (in bits), is lower than a given budget. Note that in this section, we assume that the goal is to minimize the distortion under a rate constraint. It is also possible to minimize the rate under a distortion constraint. The only modification would be to exchange the roles of $\mathcal{D}$ and $\mathcal{R}$.

As discussed in [72], the first step consists in analyzing the rate $\mathcal{R}(R_i)$ and distortion $\mathcal{D}(R_i)$ associated to each region R_i in the tree. The computation of the distortion is rather straightforward and the Squared Error between the original and coded frames can be used:

$$\mathcal{D}(R_i) = \sum_{p \in R_i} \left(f^Y(p) - M^Y_{R_i}\right)^2 + \alpha \left(\left(f^U(p) - M^U_{R_i}\right)^2 + \left(f^V(p) - M^V_{R_i}\right)^2 \right) \tag{5.11}$$

where $f^Y(p)$, $f^U(p)$ and $f^V(p)$ denote the luminance and chrominance components of the image f at pixel position p whereas $M^Y_{R_i}$, $M^U_{R_i}$ and $M^V_{R_i}$ represent the components of the region model. The parameter α is used to balance the luminance and the chrominance distortion. In order to simplify the approach, we assume that each color component of the region is modeled with a constant value.

The situation is more complex for the computation of the rate $\mathcal{R}(R_i)$. Indeed the rate associated to a region is composed of 24 bits for the color information (or 8 in the case of a gray-level image) plus a certain number of bits for the shape information. In this work we assume that the contour of each region is coded independently with respect to its neighboring regions. We have used the following approximation of the contour rate: an average number of bits necessary to encode a contour point has been estimated. This number is denoted by *BPCP* (Bits Per Contour Point). We have assumed that the contour rate assigned to a region is equal to this average figure multiplied by the region perimeter, ∂R_i. As a result, the rate per region is given by:

$$\mathcal{R}(R_i) = 24 + BPCP \times \partial R_i \tag{5.12}$$

The rate/distortion optimization itself relies on the technique discussed in [49, 58, 59]. The problem can be formulated as finding a partition $\mathcal{P} = \{R_i\}$ such that the distortion $\mathcal{D} = \sum_{R_i} \mathcal{D}(R_i)$ of the associated image is minimized with the restriction that the total rate $\mathcal{R} = \sum_{R_i} \mathcal{R}(R_i)$ is below a given budget $\mathcal{R}_0$. Note that both the rate and the distortion have to be additive over the regions. It is well known that this problem can be reformulated as the minimization of the Lagrangian: $\mathcal{D} + \beta\mathcal{R}$ where β is the so-called Lagrange parameter. Both problems have the same solution if we find β^* such that $\mathcal{R}$ is equal (or very close) to the budget. Therefore, the problem consists in using the Binary Partition Tree to find a set of regions creating a partition such that:

$$\text{Min } \{\mathcal{D} + \beta^*\mathcal{R}\} \text{ , with } \beta^* \text{ such that } \mathcal{R} \approx \mathcal{R}_0 \tag{5.13}$$

Assume, in a first step, that the optimum β^* is known. The definition of the best partition can be done by a bottom-up analysis of the Binary Partition Tree. To initialize the process, all the leaves of the Binary Partition Tree are assumed to belong to the optimum solution. Then, one checks if it is better to code the area represented by two sibling nodes as two independent regions $\{R_1, R_2\}$ or as a single region $R_{1\cup 2}$ (the common parent node of R_1 and R_2). The selection of the best choice is done by comparing the Lagrangian of $R_{1\cup 2}$ with the sum of the Lagrangians of R_1 and R_2:

$$\begin{array}{l} \text{If } \mathcal{D}(R_{1\cup 2}) + \beta^*\mathcal{R}(R_{1\cup 2}) \leq \sum_{i=1,2} \mathcal{D}(R_i) + \beta^*\mathcal{R}(R_i) \\ \quad \left\{ \begin{array}{ll} \text{then,} & \text{encode } R_{1\cup 2} \text{ as a single region} \\ \text{else,} & \text{encode } R_1 \text{ and } R_2 \text{ as two independent regions} \end{array} \right. \end{array} \tag{5.14}$$

The best encoding strategy (encode $R_{1\cup 2}$ as itself or as the union of its children) is stored in $R_{1\cup 2}$ together with the corresponding Lagrangian value. The procedure is iterated up to the

```
function BottomUpAnalysis(Input: $\beta$, Output: $\mathcal{R}, D$)
    for(d←max-depth; d ≥ 0; d←d-1)
        for each node $N_k$ in $ND_d$
            Lag($N_k$) ← $\mathcal{R}(N_k) + \beta^{*}\mathcal{D}(N_k)$
            if is-leaf-node($N_k$)
                Encode($N_k$) ← true
            else  // Selection of nodes to code
                if Lag($N_k$) ≤
                    Lag(Child1($N_k$)) + Lag(Child2($N_k$))
                then
                    Encode($N_k$) ← true
                    Encode(Child1($N_k$)) ← false
                    Encode(Child2($N_k$)) ← false
                else
                    Lag($N_k$) ← Lag(Child1($N_k$)) + Lag(Child2($N_k$))
    for(d←0; d ≤ max-depth; d←d+1)      // Clean up decisions
        for each node $N_k$ in $ND_d$
            if Encode($N_k$) = true
                for each proper descendant $N_j$ of $N_k$
                    Encode($N_j$) ← false
    $\mathcal{R}$ ← 0; $\mathcal{D}$ ← 0   // Assess overall distortion and coding rate
    for each node $N_k$ in the tree
        if Encode($N_k$) = true
            $\mathcal{R} \leftarrow \mathcal{R} + \mathcal{R}(N_k)$
            $\mathcal{D} \leftarrow \mathcal{D} + \mathcal{D}(N_k)$
function end
```

Figure 5.40: Pseudo-code of the bottom-up analysis performed used to find the best coding strategy given β.

root node and defines the best coding strategy. In Fig. 5.40 the associated algorithm used to perform the bottom-up analysis is shown. The algorithm finds the best coding strategy (according to Eq. 5.14) given a certain β. Lag(N_k) denotes the Lagrangian associated to node N_k, Encode(N_k) is a boolean variable that indicates if node N_k belongs to the best encoding strategy, and Child1(N_k) and Child2(N_k) denote the first and second child, respectively, of node N_k.

In practice, of course, the optimum β^* parameter is not known and the previous bottom-up analysis of the Binary Partition Tree is embedded in a loop that searches for the best β parameter. The computation of the optimum β parameter can be done with a gradient

```
$\beta_l \leftarrow 0$;                    // Compute $\mathcal{D}$ and $\mathcal{R}$ for a very low $\beta$
BottomUpAnalysis(Input: $\beta_l$, Output: $\mathcal{R}, D$);
if $\mathcal{R} < \mathcal{R}_0$ then { no solution; exit;}
$\mathcal{R}_l \leftarrow \mathcal{R}$; $\mathcal{D}_l \leftarrow \mathcal{D}$;
$\beta_h \leftarrow 10^{20}$;                 // Compute $\mathcal{D}$ and $\mathcal{R}$ for a very high $\beta$
BottomUpAnalysis(Input: $\beta_h$, Output: $\mathcal{R}, D$);
if $\mathcal{R} < \mathcal{R}_0$ then { no solution; exit;}
$\mathcal{R}_h \leftarrow \mathcal{R}$; $\mathcal{D}_h \leftarrow \mathcal{D}$;
do                                // Find the optimum $\beta$ value
    $\beta \leftarrow (\mathcal{D}_l - \mathcal{D}_h)/(\mathcal{R}_h - \mathcal{R}_l)$;
    BottomUpAnalysis(Input: $\beta$, Output: $\mathcal{R}, D$);
    if $\mathcal{R} < \mathcal{R}_0$ then
        $\mathcal{R}_h \leftarrow \mathcal{R}$;$\mathcal{D}_h \leftarrow \mathcal{D}$;
    else
        $\mathcal{R}_l \leftarrow \mathcal{R}$; $\mathcal{D}_l \leftarrow \mathcal{D}$;
until ($\mathcal{R} \approx \mathcal{R}_0$)
```

Figure 5.41: Pseudo-code for the rate-distortion optimization. The algorithm finds the optimum value of β such that $\mathcal{R} \approx \mathcal{R}_0$.

search algorithm. The algorithm starts with a very high value β_h (10^{20}) and a very low value β_l (0) of β. For each value of β, the bottom-up optimization procedure described above is performed. For β_h (resp. β_l), the optimum resulting partition should be the one associated to the root node (resp. leaf nodes). The partitions correspond to rates $\mathcal{R}_h$ and $\mathcal{R}_l$, and should be below and above the budget respectively. If none of these rates is close enough to the budget, a new Lagrange parameter is defined as $\beta = (\mathcal{D}_l - \mathcal{D}_h)/(\mathcal{R}_h - \mathcal{R}_l)$. The procedure is iterated until the rate gets close enough to the budget. The pseudo-code of the algorithm is described in Fig. 5.41. In practice, the optimum β^* parameter is found with few iterations, typically less than ten iterations. The bottom-up analysis itself is not expensive in terms of computation since the algorithm has simply to perform the comparison of equation 5.14 for all nodes of the tree.

Fig. 5.42 shows a Binary Partition Tree corresponding to an initial partition involving 100 regions. If this original image would have to be transmitted for browsing, and assuming that a coding strategy involving the coding of the contours with chain code and of a constant color value for each region is used, the cost in terms of bits would be approximately equal to 12898 bits (see Fig. 5.42). With respect to the original image in QCIF format, this strategy already provides a reasonable compression factor: the original image involves $176 \times 144 \times 8 = 202752$ bits and the corresponding compression factor is equal to 15.7. However, for visualization

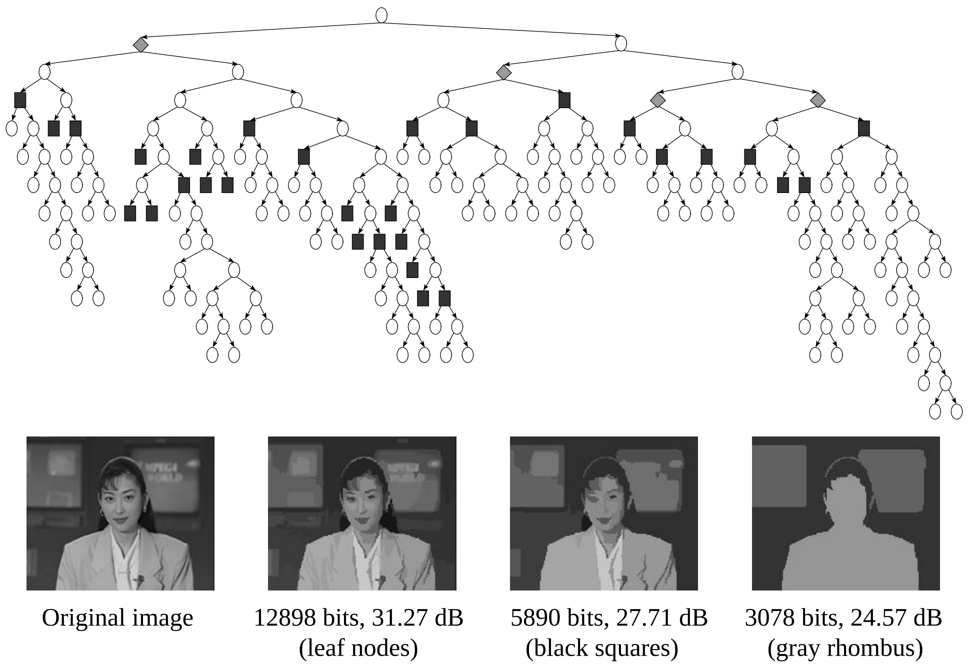


Figure 5.42: Examples of pruning for visualization: Black squares (gray rhombus) in the tree define the optimum solution in the rate-distortion sense for 11000 bits (3000 bits).

purposes, this strategy is not optimum. We show in Fig. 5.42 two more examples of coded images at 5890 and 3078 bits. These images have a higher compression factor and can be used in the case of a low transmission rate. At the same time, they allow the user to have an idea about the image content. The coding strategy with 12898 bits is associated to code the leaf nodes of the tree (the initial partition), whereas the one with 5890 bits (resp. 3078 bits) is associated to the nodes depicted with a black square (resp. gray rhombus). As can be seen, for low bit rates, the algorithm selects regions close to the root of the tree. For higher bit rates, a large number of small regions providing details about the image content can be transmitted. Finally, Fig. 5.43 gives the complete rate/distortion curve. One can see the evolution of the visual quality as a function of rate associated to the image.

5.8 Performance

Let us discuss the issues related to the memory and computational cost associated to the processing of images via Max-Tree or Binary Partition Tree. For simplicity purposes, we will restrict ourselves to compare both for the case of gray-level images.

The construction of the tree, as already discussed in Chap. 3 and Chap. 4, is generally much faster for the Max-Tree than the Binary Partition Tree. The Max-Tree construction

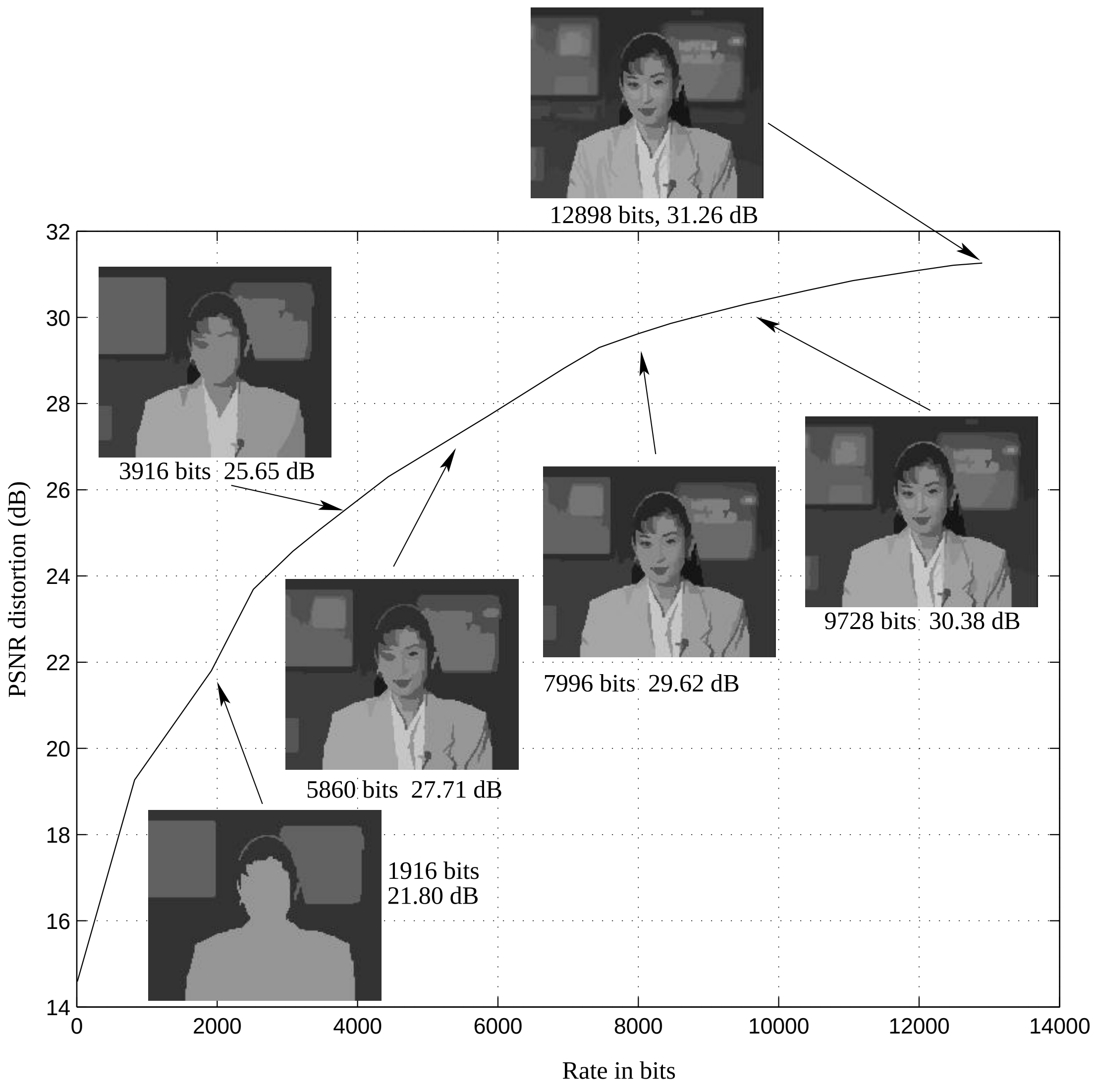


Figure 5.43: Rate/distortion curve for the partition tree of Fig. 5.42

algorithm has been designed for a particular type of input data: binary or gray-level images. The number of possible gray-levels is limited and thus its associated functions (such as the hierarchical queue) can be targeted for this particular set of data. As a result, the memory cost associated to the construction of the Max-Tree can be kept to reasonably low levels.

On the contrary, the construction of the Binary Partition Tree is much more expensive in terms of time. The bottleneck in the construction process is the computation of the initial partition that defines the leaf nodes of the tree. In our work the initial partition is constructed using the general merging algorithm starting from a graph made up of single pixels. If the initial partition is computed starting from a graph with a lower number of regions the computation time can be reduced considerably (see Sec. sec:BptPerformance).

For the case of a gray-level image, one may take the watershed of minima to reduce the number of regions of the graph. The construction of the tree structure, given the initial partition, depends mainly on the number of regions of the initial partition. However, the tree creation is generally fast in terms of time. From our experimental results, for an initial partition sized 176×144 pixels made up of 500 regions, the construction of the tree using color homogeneity takes about 0.1 seconds on a Pentium II 400MHz Linux based computer (this value includes graph construction from initial partition and merging algorithm).

In terms of memory cost, when constructing the Binary Partition Tree, the bottleneck is again at the computation of the initial partition. Performing the merging of regions starting from the pixel level results in a high memory consumption. As before, starting the merging algorithm from a lower number of regions reduces the memory cost: the lower the number of regions the lower the memory consumption.

The tree processing is very fast on both the Max-Tree and Binary Partition Tree: both are tree structures, and thus the tree analysis and pruning can be done in a fast way. Note that in the case of the Binary Partition Tree there may be an additional computational and memory cost in the analysis algorithm, with respect the Max-Tree, due to the marker & propagation with conflict resolution algorithm.

5.9 Conclusions

In this chapter the usefulness of region based representations for filtering and segmentation applications has been discussed. Both applications can be implemented using pruning algorithms. The pruning strategy is based on analyzing each node of the tree by measuring a specific criterion on each of the associated regions. Then, a decision on the elimination or preservation is taken on each node using usually a simple threshold on the criterion value. The non-increasingness of the criterion has been studied and a robust strategy based on an optimization algorithm has been presented.

The Max-Tree (resp. Min-Tree) is a tree oriented towards the maxima of the image. Thus, this representation is suitable for removing bright (resp. dark) objects. Furthermore, the so called transition zones remain unchanged. This produces smooth contours in the filtered image (as seen, for instance, in Fig. 5.21 and Fig. 5.26, pages 106 and 110 respectively).

On the contrary, the Binary Partition Tree is a structure that represents the regions that result from a region based merging algorithm. Thus, as opposed to the Max-Tree, the Binary Partition Tree is able to represent multi-component images. The resulting filtered images are characterized by having sharp contours. However, it should be noted that this effect is due to the simple model (mean color) used to fill up each of the regions of the partition. More complex models may result in smoother contours.

As a result, the image or video sequence associated to the pruned tree has been either filtered or segmented with respect the original image according to the selected criterion.