Chapter 2

General Framework

This chapter introduces the framework of this work. First of all we are going to introduce some mathematical notions which are necessary to define precisely concepts such as region, partition or partition hierarchy. Then, the basic terminology associated to trees is introduced and the main issues concerning the processing of images with trees is discussed. The current available techniques in the field in which this thesis is enclosed are reviewed and finally, the framework used to process images using trees is presented.

2.1 Basic definitions

In our work, we are going to consider the image or *function* f as a mapping from a finite rectangular subset E, the underlying grid, of the discrete plane $\mathbb{Z}^2$ into a discrete set $\{0, 1, \ldots, NG\}$ of gray-levels, where NG represents the maximum possible gray-value (usually $NG = 255$). A *binary* image can only take values 0 or 1 and is often regarded as the set $X \subset E$ whose values at the points of the grid E are 1. On the other hand, *multicomponent* images are viewed as a vector of functions, $(f^A, f^B, \ldots)$, where f^A, f^B, ... make reference to the A, B, ... component of the image.

Let $p = (p_x, p_y)$ a point of the grid E, also called a pixel of the image f, where p_x (resp. p_y) is the horizonal (resp. vertical) coordinate of the point. We denote by $f(p)$ the value of the function f at position p. Similarly, $f^A(p)$, $f^B(p)$, ... denotes the A, B, ... component of the image evaluated at point p.

We denote by $\mathcal{N}_E(p)$ the set of the neighbors of pixel p for subset E. We assume that $\mathcal{N}_E(p)$ is translation invariant and symmetrical versus the point p. Useful neighborhood are those defining 4, 8 and 6 *connectivity* relationship (see Fig. 2.1).

In the squared grid 4 or 8 connectivity is usually used, whereas the 6 connectivity neighboorhood is used in the case of a hexagonal grid (see Fig. 2.2). The hexagonal grid is preferred from an algorithmic point of view due to its good symmetry properties, whereas the squared

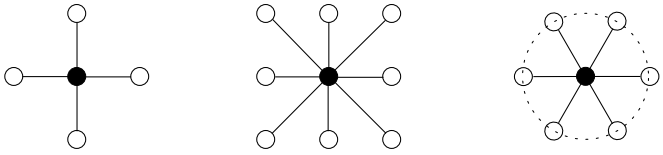

Figure 2.1: Most commonly used connectivity types in digital image processing.

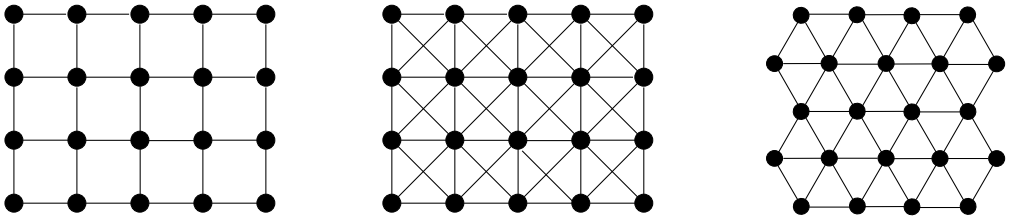

Figure 2.2: Squared grid (in 4 and 8 connectivity) and hexagonal grid (with 6 connectivity).

grid is preferred in practice since it models the way scanners and cameras digitalize an image.

By using the notion of connectivity, we define the following notions.

Definition 2.1 (Path) *Let x and y be two pixels on E. We say that there exists a path connecting x and y if, and only if, there exists an n-tuple of pixels $(p_0, p_1, \ldots, p_n)$ such that $p_0 = x$ and $p_n = y$ and*

$$p_i \in \mathcal{N}_E(p_{i-1}) \quad \forall i = 1, \ldots, n$$

Following this definition, we consider now the notion of connected component.

Definition 2.2 (Connected component) *Let X be a set of pixels included in E, and let p be a pixel of X. The connected component of X that contains p, $CC_p(X)$, is the union of all the paths included in X with origin in p.*

It is now possible to define the flat zone associated to point p as the largest connected component of f which includes p and where the function is constant. We may also use the notion of path to define it as follows:

Definition 2.3 (Flat zone) *Two pixels x and y belong to the same flat zone of function f if, and only if, there exists a path between x and y, $(p_0, p_1, \ldots, p_n)$, such that $p_0 = x$ and $p_n = y$ and that $f(p_i) = f(p_{i+1})$ for all values of i.*

A flat zone X such that $f(p) = h$ for all $p \in X$ is called flat zone of gray-level h.

The notion of *level set* is also needed in this work:

Definition 2.4 (Level set) *Given an image f, we call upper level set X_h of value h and lower level set X^h of value h the subsets:*

$$X_h(f) = \{p \in E, f(p) \geq h\} \quad X^h(f) = \{p \in E, f(p) \leq h\}$$

The level sets are nested; the family of upper (resp. lower) level sets is decreasing (resp. increasing):

$$\forall h_1 \leq h_2, \quad X_{h_1}(f) \supseteq X_{h_2}(f), \quad X^{h_1}(f) \subseteq X^{h_2}(f)$$

It is also possible to establish a nesting relationship between the connected components of $X_h(f)$: let p be a pixel of the upper level set $X_{h_2}(f)$, $p \in X_{h_2}(f)$, and $CC_p(X_{h_2}(f))$ its associated connected component. Then:

$$\forall h_1 \leq h_2, \quad CC_p(X_{h_1}(f)) \supseteq CC_p(X_{h_2}(f))$$

Similarly, if $p \in X^{h_1}(f)$ and $CC_p(X^{h_1}(f))$ is its associated component,

$$\forall h_1 \leq h_2, \quad CC_p(X^{h_1}(f)) \subseteq CC_p(X^{h_2}(f))$$

Furthermore, the level sets and the flat zones of a function are related. First, note that the set $\{p \in E, f(p) = h\}$ is the set of flat zones of gray-level h of the function f. The following relation holds:

$$\forall h, \quad X_h(f) = \{p \in E, f(p) = h\} \cup X_{h+1}(f) = \bigcup_{j \geq h} \{p \in E, f(p) = j\}$$

The previous equation states that the upper level set $X_h(f)$ is the union of the flat zones of gray-level $j \geq h$. Similarly, for the lower level sets,

$$\forall h, \quad X^h(f) = \{p \in E, f(p) = h\} \cup X^{h-1}(f) = \bigcup_{j \leq h} \{p \in E, f(p) = j\}$$

An equivalent relationship can be established between the connected components of the level sets and its associated flat zones.

It is necessary to define precisely the notion of partition and partition hierarchy, since they are commonly used in this work.

Definition 2.5 (Partition) *A partition $\mathcal{P}$ of the space E is a set of connected components $\{R_i\}$ that satisfy*

$$\forall i, j \quad i \neq j \;\Rightarrow\; R_i \cap R_j = \emptyset \;\; \textit{and} \bigcup_i R_i = E$$

where each connected component R_i is called a region.

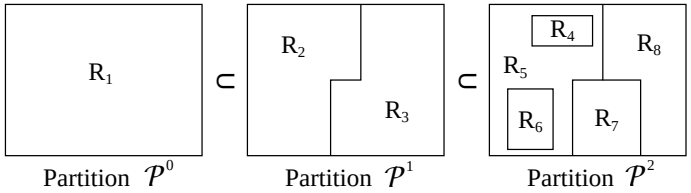

Figure 2.3: Example of partition hierarchy. Each of the partitions proposes a complete decomposition of the image and it is possible to establish an inclusion order between them, $\{\mathcal{P}^0, \mathcal{P}^1, \mathcal{P}^2\}$.

The partition is used to classify each pixel p of the function f as belonging to a certain region R_i. As such, the partition associated to an image f is usually interpreted as a label image g, of the same size of f, such that $p \in R_{g(p)}$. Note that the partition of the space is not unique.

From the above definitions we may conclude that a flat zone is a special kind of region, and the *partition of flat zones* of a function f is the partition of the function f where each R_i is a flat zone. The partition of flat zones of a function f is unique.

By using the definition of partition, we say that a partition $\mathcal{P}^0$ is included in partition $\mathcal{P}^1$, $\mathcal{P}^0 \subset \mathcal{P}^1$, if any region $R_j^1 \in \mathcal{P}^1$ is completely included in a region $R_i^0 \in \mathcal{P}^0$. This allows us to define a partition hierarchy as

Definition 2.6 (Partition hierarchy) *Let $\mathcal{H}$ be a set of partitions $\{\mathcal{P}^i\}$. We say that $\mathcal{H}$ is a partition hierarchy if it is possible to define an inclusion order between each pair of elements of $\mathcal{H}$, that is, $\mathcal{P}^i \subseteq \mathcal{P}^{i+1}$.*

Fig. 2.3 shows an example of partition hierarchy $\{\mathcal{P}^0, \mathcal{P}^1, \mathcal{P}^2\}$. Each of the partitions proposes a complete decomposition of the image and it is possible to establish an inclusion order between them. In this particular case, $\mathcal{P}^0 \subset \mathcal{P}^1 \subset \mathcal{P}^2$. Partition $\mathcal{P}^2$ includes more detail than $\mathcal{P}^1$, or in other words, all the contours of $\mathcal{P}^1$ are present in $\mathcal{P}^2$. On the contrary, partition $\mathcal{P}^0$ includes less detail than $\mathcal{P}^1$ or $\mathcal{P}^2$.

2.2 Hierarchical region based processing using trees

This section first introduces the basic terminology associated with the concept of tree and some properties that have to be satisfied in the case the tree represents an image. The issues associated with the tree pruning are discussed. It is shown that the way the pruning is done depends on what the tree represents.

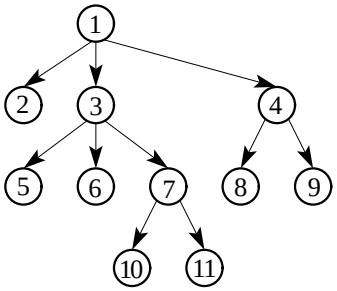

Figure 2.4: Example of tree structure.

2.2.1 Base terminology

A tree [1, 2] is a collection of elements called *nodes*, one of which is distinguished as *root*, along with a relation ("parenthood") that establishes a hierarchical structure on the nodes. Formally, a tree is defined to be [1]

Definition 2.7 (Tree) *A directed graph with no cycles is called a* directed acyclic graph. *A (*directed*)* tree *(sometimes called* rooted tree*) is a directed aciclyc graph satisfying the following properties:*

1. *There is exactly on vertex, called the* root*, to which no edges enter.*
2. *Every vertex except the root has exactly one entering edge*
3. *There is a path (which can be shown to be unique) from the root node to each vertex*

Consider the example of Fig. 2.4. Each node has been identified with a label k, $1 \leq k \leq 11$. The root node with label 1 has three *subtrees* with roots corresponding to nodes 2, 3 and 4. The node 1 is the parent of 2, 3 and 4, and these three nodes are the children of 1. Moreover, nodes 2 and 3 are the *siblings* of node 4. The first subtree, with root 2, is a tree of a single node, while the other two subtrees have a non trivial structure. For example, the subtree with root 3 has three subtrees with root node 5, 6 and 7. The first two ones correspond to one-node trees, while the last has two subtrees with root node 10 and 11.

If n_1, n_2, ..., n_k is a sequence of nodes in a tree such that n_i is the parent of n_{i+1} for $1 \leq i < k$, then this sequence is called a *path* from node n_1 to node n_k. The *length* of the path is one less than the number of nodes in the path. Thus, there is a path of length zero from every node to itself. Let us consider again the Fig. 2.4: between nodes 3 and 10 there is a path, namely $(3, 7, 10)$, of length 2.

If there is a path from node a to node b, then node a is an *ancestor* of node b, and node b is a *descendant* of node a. For example, in Fig. 2.4 the ancestors of 7 are itself, 3 and 1,

Figure 2.5: Two distinct (ordered) trees.

while its descendants are itself, 10 and 11. Notice that any node is both an ancestor and a descendant of itself.

An ancestor or descendant of a node, other than the node itself, is called a *proper ancestor* or *proper descendant*, respectively. In a tree, the root node is the only node with no proper ancestors. A node with no proper descendants if called a *leaf*. A subtree of a tree is a node, together will all its descendants.

The *height* of a node in a tree is the length of the longest path from the node to a (descendant) leaf node. In Fig. 2.4 node 2 has height 0, node 3 has height 2, and node 4 has height 1. The *height of a tree* is the height of the root node. The *depth* of a node is the length of the unique path from the root to that node.

The children of a node are usually ordered from left-to-right. Thus, the two trees of Fig. 2.5 are different because the two children of node 1 appear in a different order in the two trees. If we wish explicitly ignore the order of the children, we shall refer to a tree as an *unordered* tree. In our work we focus only on unordered trees, and we call them simply "trees".

2.2.2 Notation and properties

For notation purposes, let us denote with N_k a node of the tree and with R_k the region of support associated to N_k. With $N_k \rightarrow N_j$ we indicate that node N_k is parent of node N_j, or inversely, N_j is a child of N_k. Note that the sense of the arrow agrees with the sense of the edges on a tree, see for instance Fig. 2.4 or Fig. 2.5. Let us denote with $Child(N_k)$ the set of child nodes associated to a node N_k. Thus, a leaf node has no child and therefore $Child(N_j) = \emptyset$.

In our work we focus on hierarchical region based representations using trees. In particular, we focus on representations that decompose the image into a set of non overlapping regions.

$$\forall i, j, k \quad i \neq j \text{ such that } N_i, N_j \in Child(N_k) \Rightarrow R_i \cap R_j = \emptyset$$

where N_k is a non leaf node of the tree. The root node of the tree may represent the whole image support or a single region.

Futhermore, the region of support associated to each of the child nodes $N_j \in Child(N_k)$

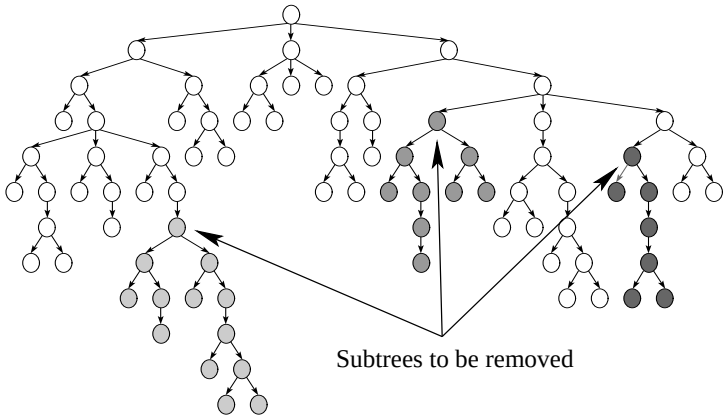

Figure 2.6: Example of tree pruning. If a node is to be removed, all of its descendants are also removed.

is contained in the region of support associated to its parent N_k.

$$\forall j,k \text{ such that } N_j \in Child(N_k) \Rightarrow R_j \subseteq R_k$$

From the previous two statements one infers that the union of the region of support of the children of a node N_k is included in the region of support of N_k.

$$\forall k \quad \bigcup_i R_i \subset R_k \text{ such that } N_i \in Child(N_k)$$

Note that the region of support of a descendant or the union of a set of descendants of a node N_k is included in the region of support of N_k.

2.2.3 Pruning

The processing techniques developed in this work are based on pruning strategies. In our work, "pruning" means that if a node N_k is to be removed all of its descendants should be also removed. In other words, the analysis algorithm applied on the tree representation should lead to a set of decisions (remove or preserve) on each node that satisfies this property. It the latter case we will say that the analysis algorithm defines a *valid pruning strategy*. Fig. 2.6 shows an example of tree pruning.

It is important to understand that the way the pruning is done depends on what the tree represents itself. One may think that pruning the tree of Fig. 2.6 is simply to remove the corresponding subtrees. When pruning a tree one has to make sure that the pruning itself results in a valid tree representation.

Let us analyze the case in which the tree represents a partition hierarchy. Its definition on page 11 states that a partition is a complete decomposition of an image into regions. That is,

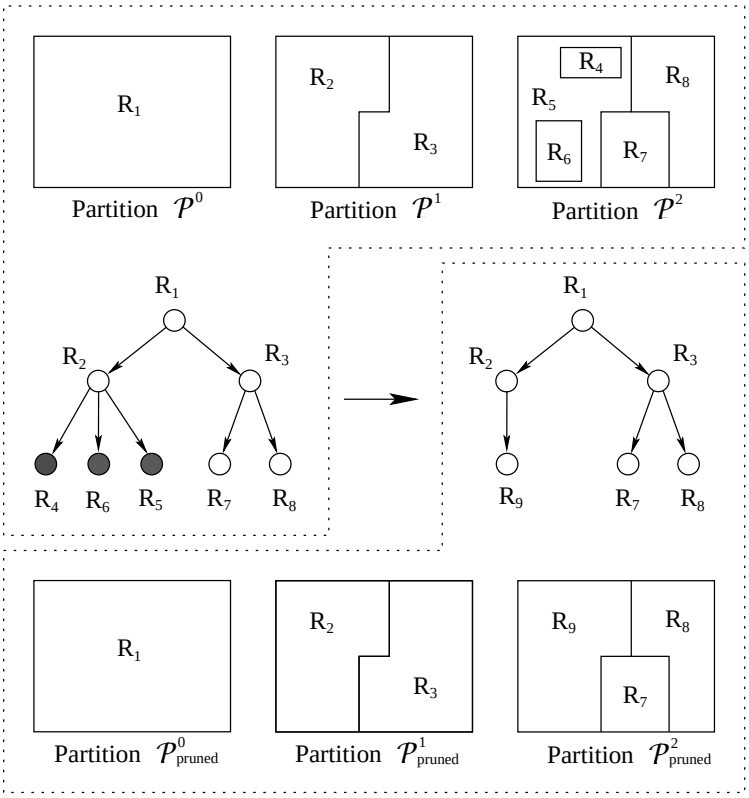

Figure 2.7: Example of tree pruning when the tree represents a partition hierarchy. On top, the original set of hierarchical partitions is shown together with its associated tree representation. The result of pruning this tree has to lead to a tree representing a set of hierarchical partitions, see bottom.

no pixel should be left without being assigned to a particular class. Therefore, the result of pruning a tree that represents a partition hierarchy should be a tree representing a partition hierarchy.

Fig. 2.7 shows an example of tree pruning illustrating the mentioned case. On top the original partition hierarchy, $\mathcal{P}^1 \subset \mathcal{P}^2 \subset \mathcal{P}^3$, is shown. Its associated tree representation is depicted on the left. Assume that the analysis algorithm applied on the tree decides to remove the nodes marked in gray. Pruning the tree should lead to a tree that is associated to a partition hierarchy. In our example regions R_4, R_5 and R_6 are the nodes to be pruned. On the partition, this means that we would like to remove the detail that is contributed by these regions. Regions R_4, R_5 and R_6 have to be merged together on the partition $\mathcal{P}^2$ giving as result $\mathcal{P}^2_{pruned}$. On the pruned tree shown on the right of Fig. 2.7, this is represented with a new node R_9 that is associated to the merging of regions R_4, R_5 and R_6. The associated partition hierarchy, $\mathcal{P}^0_{pruned}$, $\mathcal{P}^1_{pruned}$ and $\mathcal{P}^2_{pruned}$ is shown on the bottom of Fig. 2.7. Note that with the previous merging operation we ensure that the tree resulting from pruning

represents a partition hierarchy. In conclusion, in the case in which the tree represents a partition hierarchy, pruning nodes should be associated to merge the corresponding regions.

Fig. 1.3 on page 5 shows a second example of tree pruning. In this case, the tree represents a decomposition of the image but not a partition hierarchy, since the image on the top-right corner of Fig. 1.3 is not a partition (the image does not assign a label to all the pixels of the image). In this case, no particular restriction has to be imposed to the pruning. In this case "pruning" is simply associated to remove the corresponding nodes. Futhermore, the operation of removing nodes equivalently as the process of merging the nodes to remove with the first preserved ancestor node. In the example of Fig. 1.3, removing nodes R_4 and R_5 is equivalent to merge them with R_2.

2.3 State of the Art

Our purpose in this section is to review the work that has been done in the field of tree processing. We restrict ourselves to approaches in which the tree is explicitly used using pruning or other similar techniques. Note that some of the presented work has been developed parallel to the work that is presented here.

2.3.1 Quadtree

The quadtree decomposition is a simple technique for image representation at different resolution levels. Several approaches may be taken to construct the tree, from which we mention the top-down approach. With the top-down approach, the image is successively subdivided into squared blocs according to some homogeneity criterion.

Assume that the algorithm is analyzing a squared block. A criterion assessed on that block is used to decide if the pixels within the block are homogeneous. If the block being analyzed satisfies the homogeneity criterion, the block is kept and the algorithm does not subdivide it. If it is not homogeneous, the block is divided into 4 squared blocks over which the previous described analysis decomposition is applied again. The algorithm starts by considering the whole image as a single squared block.

Classical ways to assess the homogeneity inside a block may be based, for instance, in measuring the difference between the maximum and minimum grey level pixel value in the block, or in measuring the standard variance of the pixel values the block contains. Then, a simple threshold is used to decide if the block has to be further subdivided or not.

As can be seen, the decomposition relies on a simple threshold. This results in introduction of geometrical features not present in the image. In [82] an improved quadtree decomposition is presented. The threshold that decides if a block is to be further subdivided is set adaptively according to the resolution level of the block. For coding purposes, the goal is to choose

threshold values that minimize a distortion under a constraint of a fixed number of bits. Rate distortion theory is used for that purpose. A similar approach is taken in [88]. In this case the approach that has been taken is to apply rate distortion theory using the distortion and the number of bits assigned as parameters for constructing the tree.

Binary Space Partitioning Trees [57] where developed to overcome some of the problems posed by the Quadtree representation. In fact, in rigid geometric descriptions such as the quadtree representation, a very large number of regions is needed to generate a piecewise smooth signal. The Binary Space Partition Tree is based on successively dividing each region into two regions using a straight line. In [56], an approach based on the Hough transform is presented to contruct the tree. Two steps are perfomed at each iteration, and a rough idea is given here: first, the boundaries of the objects present in the image are detected through and edge detection process, and second, the Hough transform is used to determine the linear characteristic of such boundaries. The line corresponding to the one that passes through the maximum number of connected edge points is selected to partition the image into two sub-images. The whole process is then interated on each of the subimages.

The resulting tree is then pruned using a rate distortion approach for coding purposes. As before, the purpose is to guarantee a minimum distortion in the signal when rate is reduced to a determined budget. The approach used to prune the tree is presented in [57].

2.3.2 Partition Trees

The *Partition Tree* is a structured representation of a set of hierarchical partitions obtained usually by means of a segmentation procedure. As illustrated in Fig. 2.8, the *Partition Tree* [52] is created from an initial partition of an image, plus a set of hierarchical partitions that are "above" and "below" the initial partition. The latter are created by using, respectively, merging and region splitting algorithms by means of a homogeneity criterion. The regions can be homogeneous either spatially (lower levels of the tree) or in motion (upper levels of tree). Thus, the Partition Tree defines a universe of possible regions the image is made of.

In [72], the Partition Tree representation is used for coding purposes. Based on a Rate-Distortion approach [59], the Partition Tree is analyzed in order to select the best coding strategy among the set of regions that are represented by the Partition Tree and a list of coding techniques [72]. The Partition Tree defines the set of regions out of which the algorithm should create the final partition. The list of coding techniques deals with the coding of these regions. In practice, each region of the Partition Tree is coded by all coding techniques and the corresponding rate distortion values are stored in a *Decision Tree* [59, 77]. In the formulation of the Rate-Distortion problem, it is assumed that the budget (in bits) is given for each frame. Based on this budged, the algorithm finds the coding strategy that minimizes the distortion.

Fig. 2.9 summarizes the proposed strategy. Once the Partition Tree has been constructed,

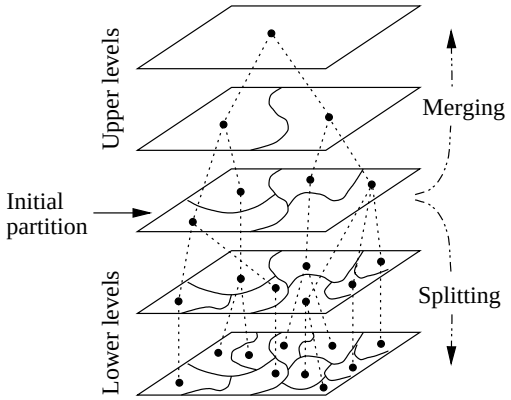

Figure 2.8: Example of Partition Tree [72].

several region coding techniques (defined by the set of coding techniques $\{CT_1, \ldots, CT_n\}$) are considered for each region. By means of the rate-distortion approach the best partition is defined. The present approach is used in [72] for a generic video coding algorithm.

2.3.3 Critical Lake Tree

The *Critical Lake Tree* [34] is a tree that represents in a compact and structured way the set of (hierarchical) partitions that are obtained when a flooding algorithm is applied on a gradient image. Similarly to the watershed [37], imagine the gradient image as a topographic surface where holes are pierced at each minima. Each minimum of the gradient image is associated to a leaf node. This surface is plunged into the water with a constant speed. Then each catchment basin is flooded creating a lake. As the level gets higher, adjacent lakes may meet. Two neighboring catchment basins will always merge at the lowest path-point [37] on the border separating them. Such point is called *critical lake* and its region of support is the union of both lakes. Over the tree structure this is represented with a node whose two children nodes are associated to the lakes that have met. From this "merging" only one of the two absorbing lakes survives. For that purpose a criterion value is assessed on both lakes at the point where they meet. The lake whose associated criterion value is greatest continues to be flooded. The resulting effect is that one of the lakes "absorbs" the other. The flooding then continues, creating new critical lakes and stops with one region at the root of the tree. Fig. 2.10 shows an example of tree of critical lakes. Note that in contrast with the classical watershed algorithm [37], in this case one lake absorbs the other, whereas in the watershed a dam is constructed between two lakes at the point where they meet [22, 91].

By proceeding with this approach the lakes separated by a low crest merge first, whereas those separated by a high watershed line are the last to merge. Thsu, the tree of critical lake allows us to classify the regions associated to the catchment basins of a gradient image

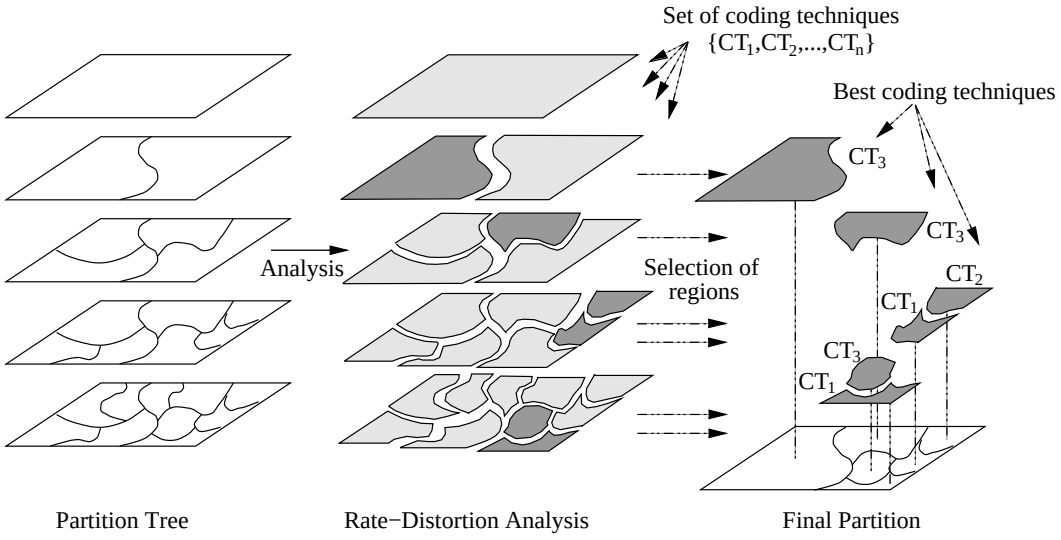

Figure 2.9: Rate-distortion approach applied on the Partition Tree.

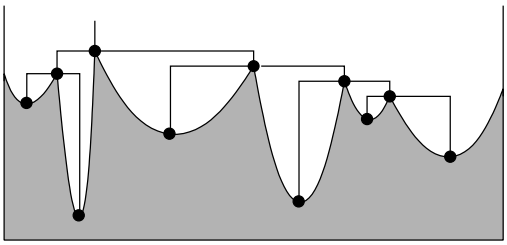

Figure 2.10: Construction of the tree of critical lake during flooding.

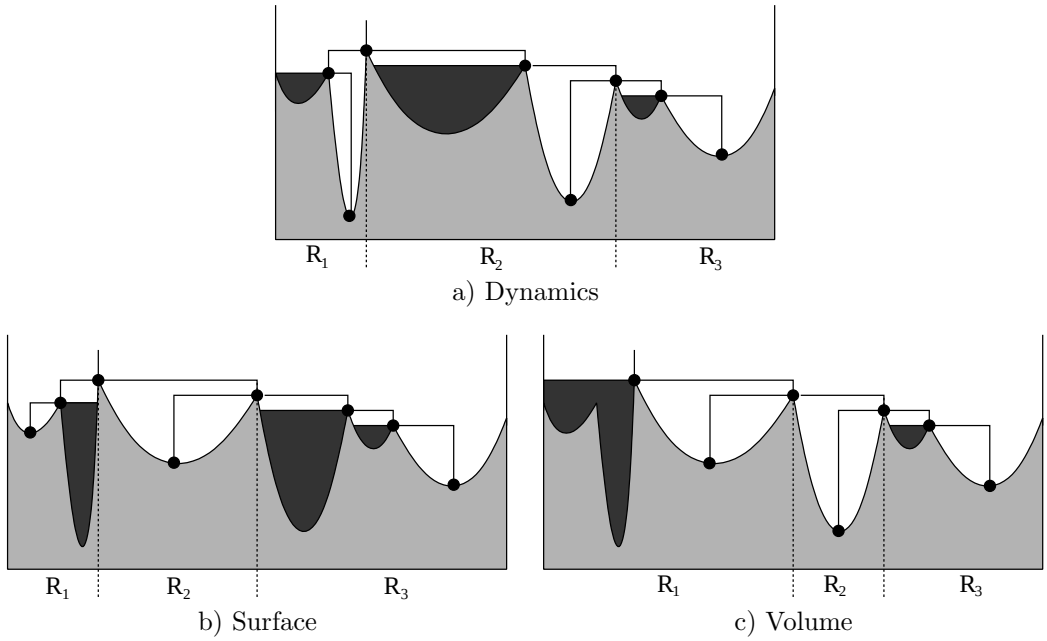

Figure 2.11: Critical lake trees pruned according to (a) Dynamics (depth), (b) Surface and (c) Volume criteria. Only three of the most important branches (minima) are preserved.

in a hierarchical way. Several approaches may be taken to establish the way a lake absorbs another. Among the possible criteria, one may find criteria based on dynamics (depth), surface or volume [15, 22].

Based on the critical lake tree, where each branch is labelled according to the selected absorbing criterion, a family of critical connected filters can be constructed [15]. The branches whose associated criterion value is below a given threshold is pruned: for that purpose, the corresponding minimum are flooded with the classical watershed. This allows to filter the regions which do not fulfill a given condition for a given criterion. For instance, it removes slightly contrasted regions when the depth measure is considered or small areas when the surface is taken into account. The result of such pruning is a segmented image. Fig. 2.11 shows several examples of critical lake pruning. The associated segmentation, made up of three regions $\{R_1, R_2, R_3\}$ is indicated in each figure.

As a result, a set of hierarchical segmentations can be obtained by the pruning operation on a given critical lake tree if several thresholds are considered. Thus, all edges of the tree may be labelled and ranked in increasing order. Cutting some edges from this tree creates subtrees and produces a partition of the image. However, if we deal with complex objects, it is possible that the resulting partition does not properly adapt to the objects present in the scene. This is usually due to the fact that certain regions have not merged correctly, or that we would like to have a partition with higher precision (i.e. more regions) on one object

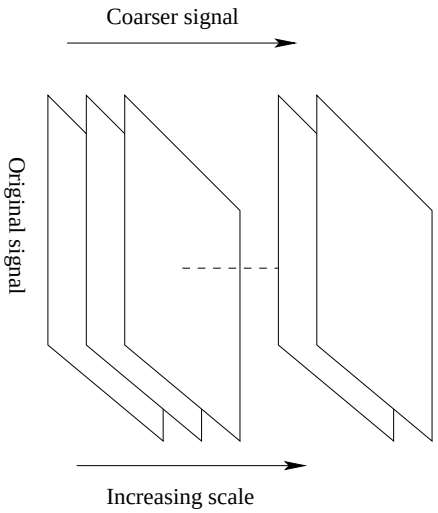

Figure 2.12: Scale Space Filtering Scheme.

and less on others. For that purpose, interactive segmentation tools try to make use of the user input to obtain the desired result [33, 101]. The approach taken in [101] allows the user to manipulate the hierarchy by merging regions that result from partitions using different thresholds until the result is satisfactory. It is only necessary to begin with the partition that best fits the desired result, and then *descend in the pyramid* in a determined region to query a higher number of regions in it, or to *climb in the pyramid* to allow merging of non interesting regions.

2.3.4 Area Tree

The *Area Tree* [64, 5, 6] is a structured representation of the objects present in the image[1]. The sieve [3, 4] is the mathematical tool used to construct the tree. A *sieve* is a method for scale-space analysis of an input signal. Scale-space methods process the image with a scale as parameter (see Fig. 2.12). In a conventional scheme, scale is usually associated with resolution desired in the output function. A sieve is a non-linear decomposition algorithm [3, 4] that is based on the theory of mathematical morphology [79, 80]. Let Ψ_s be a morphological filter that removes extrema from an input signal at a specific scale s. So, for instance, Ψ_1 removes extrema of scale 1, Ψ_2 of scale 2 and so on. The filter Ψ_s is applied on the image with increasing s until the resulting image is flat (or, in other words, the maximum scale of the original signal has been reached).

[1]The tree is called originally by the authors as "Scale-Space Tree". However, in this work we use the term "Area Tree" to refer to it in order to avoid misunderstanding with the general term "scale-space representations".

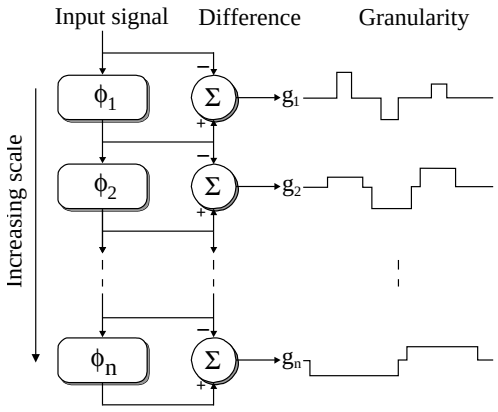

Figure 2.13: Sieve decomposition scheme.

The removed extrema at each step of the filtering form *granules* that are stored in a new domain, called *granularity*. Fig. 2.13 shows an example of sieve decomposition of a signal f. Note that the scheme is similar to the Laplacian pyramid [12]. At each step, the signal is filtered using a filter Ψ_s that uses the previous filtering result as input. The *granule function* g_s is defined as the difference between two stages of the sieve filter. The granule function g_s contains all connected subsets of the original function that have been removed from the original signal for scale s. The *granules* g_{s_i} are the non-zero connected regions in the granule function g_s.

The set of granules g_s that are obtained at each stage of the decomposition are included in each other. Thus, they may be represented by means of a tree structure. In such tree, the nodes are associated to the granules g_{s_i} that are removed by the sieve Ψ_s. The parenthood relationship indicate to which granule (or region) the granules are "merged" as the scale of Ψ_s increases. The sieve algorithm can use any morphological filter to perform the decomposition of the image. Useful examples for Ψ_s are the area opening [93], area closing, $\mathcal{M}$-filter (area opening followed by an area closing) or the $\mathcal{N}$-filter (area closing followed by an area opening). In this case the scale parameter s is the size in pixels of the granules that are removed by the filter.

Fig. 2.14 shows an example of Area Tree representation. The original image is made up of seven flat zones. On this image, a sieve Ψ_s is applied. Note that the first regions to be removed by the filter are regions R_4, R_3 and R_5 since they have the least scale (or area). When the filter at the appropriate scale is applied, these regions are merged to region R_6. This is represented in the tree with a set of nodes associated to regions R_4, R_3 and R_5 whose parent is region $R'_6 = R_6 \cup R_3 \cup R_4 \cup R_5$. The next regions to be removed by the sieve, as the scale s increases, are regions R_2 and R_1 which are merged to region R_7. Finally, the last region to be removed is region R'_6.

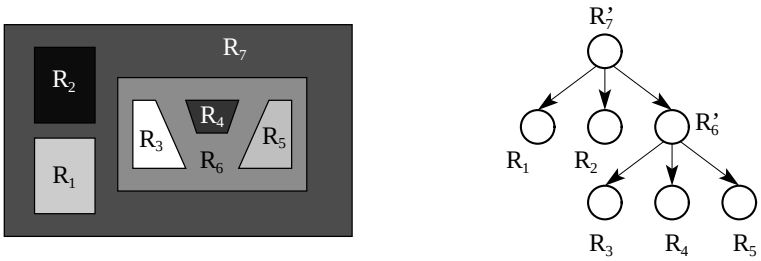

Figure 2.14: Example of Area Tree representation. The original image is made up of seven flat zones, $\{R_1, R_2, \ldots, R_7\}$. On the left, the Area Tree representation is shown, where $R'_6 = R_6 \cup R_3 \cup R_4 \cup R_5$ and $R'_7 = R_7 \cup R_1 \cup R_2 \cup R'_6$.

The tree representation can be used for several applications [64, 6, 44]. Based on pruning strategies one may, for instance, assess a criterion on each node of the tree and select by means of a threshold which nodes should be removed. Possible applications include motion filter [5] and stereo matching [44].

2.3.5 Inclusion Tree

The Inclusion Tree [43] is a scale-space representation of the image. The image is decomposed into a tree of "shapes" based on connected components of level sets. As it has already been pointed out in Sec. 2.1 the connected components of the level sets are nested, enabling thus to represent the level sets connected components with a tree, as shown in Fig. 2.15. As we can see, the connected components associated to the upper and lower level sets differ.

The idea behind the Inclusion Tree is to obtain a representation of the image in which the information associated to the upper and lower level sets has been merged together. Note, for instance, that region F appears as a connected component in the lower level set but is a hole in the upper level set. On the contrary, region D appears as such in the upper level set but is a hole in the lower level set. Thus, in [43] a different approach is proposed to create the tree: the author considers the level set connected components of $X_h(f)$ and $X^h(f)$ where its associated holes have been filled.

The tree creation is done as follows. First, build the tree of connected components of lower level sets and the tree of connected components of upper level sets, taking into account the holes in each connected component. Second, find for each hole in a connected component the connected component in the other tree corresponding to it. Third, merge both trees, putting connected components corresponding to holes as descendants of the ones containing them. Fig. 2.16 shows an example in which the information of the level sets of Fig. 2.15 has been merged into one tree.

The Inclusion Tree is a scale-space representation of the image. Due to inclusion, the

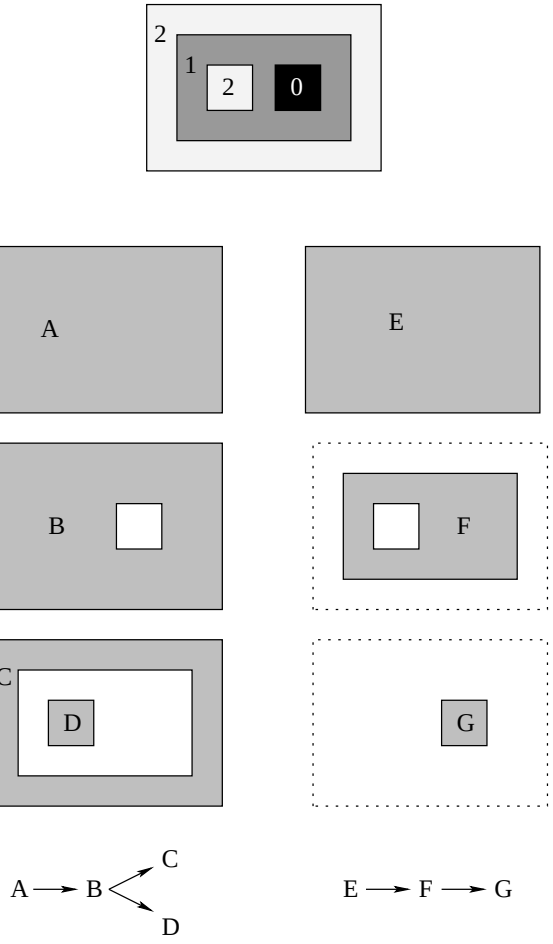

Figure 2.15: The trees of connected components of upper and lower level sets of a simple image. On top, the original image, made up of 4 flat zones is shown. The number in each region is associated to its gray level value.

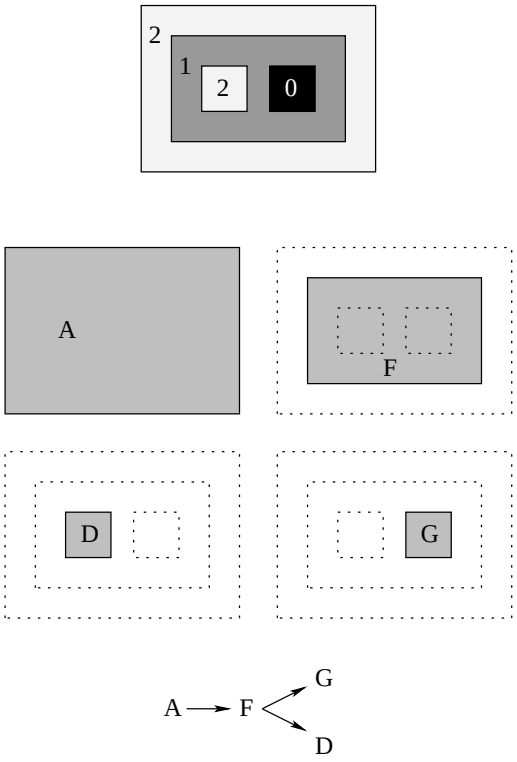

Figure 2.16: The Inclusion Tree for a simple image, see also Fig. 2.15.

shapes of the tree are sorted with respect to their sizes. A shape obviously contains only objects that have a smaller size. As for the Area Tree (see Sec. 2.3.4), the scale is here directly the size of the shapes in term of number of pixels. Large scale objects appear near to the root of the tree, whereas small scale objects appear near the leaf nodes. It should be noticed that the Inclusion Tree is self-dual: the tree structure that is obtained from image f is the same than the structure obtained from the image $-f$.

As it was done for the Area Tree, several approaches may be taken to process the tree [43, 42]. For instance, a measure can be taken on each node of the tree and then decide the removal based on a threshold. Area or motion filter can be found in [43].

2.3.6 Discussion

The Quadtree, Partition Tree and Critical Lake Tree are structures devoted to represent a set of hierarchical partitions, whereas the remaining structures are devoted to respresent a set of regions that can be extracted from the image at different resolution levels. Each of them has been developed for a set of different applications. These applications range from coding, to filtering - area, motion, stereo, ... - and segmentation.

Note that Area Tree and the Inclusion Tree are invertible: the original image can be obtained from the tree representation. For instance, the decomposition performed by the sieve in the Area Tree is invertible since it is based on the same scheme as the Laplacian pyramid [12]. The invertibility property is interesting, since it allows transforming from the image domain to the tree representation domain and vice-versa. In the case of the Quadtree, Partition Tree, and Critical Lake Tree only an approximation of the original image can be obtained: the detail of the reconstructed image corresponds to the level of detail of the leaf nodes of the tree representation.

The reader may have noticed that the Inclusion Tree and the Area Tree (presented previously in Sec. 2.3.5 and Sec. 2.3.4 respectively) are closely related. The Area Tree uses the area as parameter to structure the regions, whereas the Inclusion Tree uses shape containment relationship to construct the tree. It may seem that both representations are in fact the same representation. In Fig. 2.17 it is shown that this is not true. On top, the original image and the set of connected components (filling its holes) that can be extracted from the upper and lower level set are shown. The associated Inclusion Tree has three nodes hat indicate the inclusion relationship of the different shapes that make up the image. On bottom the Area Tree representation is shown. Assume, for that purpose, that the area (size in pixels) associated to region E is less than the one associated to F and that an $\mathcal{M}$-filter (opening followed by a closing) is used as sieve. In order to construct the tree representation, a set of sieves Ψ_s with increasing s is applied on the image. Notice that the white ring is removed when the scale parameter, s, is equal to the area associated to E. The resulting image is flat and therefore the decomposition can be stopped here. The associated Area Tree representation is depicted

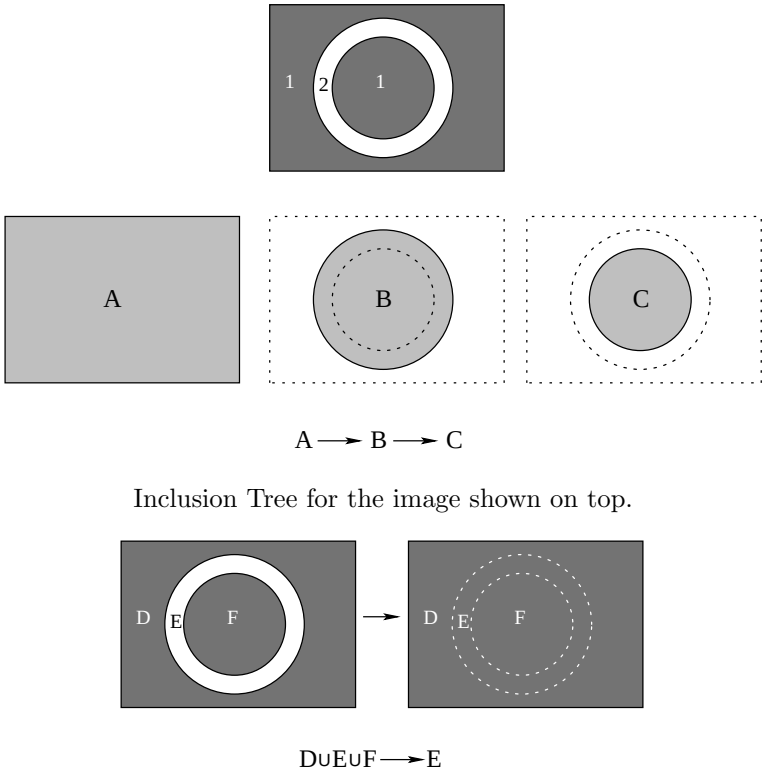

Figure 2.17: Example showing the Inclusion Tree and the Area Tree correspond to different representations.

on bottom of Fig. 2.17. Thus, it is clear that the Inclusion Tree structures regions according to its inclusion relationship with neighboring regions, whereas the Area Tree indexes regions according to its size.

Additionally, it should be noticed that since the Inclusion Tree is self-dual the associated operators Ψ that are applied on it are also self-dual, that is, $\Psi(f) = -\Psi(-f)$. The self-duality property is not hold for the case of the Area Tree since neither the $\mathcal{M}$-filter nor the $\mathcal{N}$-filter are self-dual.

2.4 Objectives and contribution of the thesis

As outlined in Sec. 1.2, the main objectives of this thesis is the dicussion of hierarhical representations and processing for region based processing. Let us first discuss the objectives for the hierarchical representation.

Two different tree representations are studied in this work: the Max-Tree and the Binary

Partition Tree. The Max-Tree a simple, easy to contruct tree representation. Moreover, a very efficient algorithm has been developed for its construction. It was developed to implement anti-extensive connected operators. Not only classical operators can be applied on such representation, but taken advantage of the tree structure new and more complex operators can be implemented on it. Moreover, as will be studied later on (sec Sec. 3.3 and Sec. 5.7.1) the Max-Tree can be obtained via the approach of Sec. 2.3.4, whereas the Min-Tree (the dual representation of the Max-Tree) is a representation that includes the representation of Sec. 2.3.3.

On the other hand, the Binary Partition Tree was developed to overcome some of the drawbacks imposed by the Max-Tree, but has demonstrated to be a representation that may be used for a wider range of applications than the Max-Tree, such as connected operators, segmentation and information retrieval. In fact, the Binary Partition Tree, as opposed for instance to the Max-Tree or the Area Tree, can be constructed with a higher flexibility than the latter ones. Moreover, the Binary Partition Tree can be constructed in a self-dual manner. As a result, operators implemented on it are self-dual.

The second objective is this thesis is the hierarchical processing. As discussed previously, our purpose is to focus only processing techniques based on pruning. The approaches studied in Sec. 2.3 are all based on the following pruning approach: each node of the tree is first analysed by assessing a particular criterion on and second, pruning is perfomed according to a fixed threshold. In this work, the latter strategy has been adopted to process in some cases the tree. Furthermore, processing strategies in which the analysis is global on the entire tree structure are developed. The problem of the lack of robustness of non-increasing criteria is studied and a solution based on an optimization strategy is proposed. Furthermore, a content based image retrieval approach based on trees is developed.

2.5 General framework

In Fig. 2.18, we illustrate the general scheme that has been used in our work for processing images and videos. It is divided into two steps. In the first step, the tree representation is constructed using the image or video sequence. In the second step, the tree is processed in order to decide which nodes have to be pruned. Note that both input and output of the general scheme is a pixel based representation.

2.5.1 Tree construction (Part I)

The first block is devoted to the tree creation. It can be viewed as the process where the abstraction from pixel to region is performed. The input to this block is the pixel based image. An associated partition can be given. This partition establishes the maximum detail to have in the resulting tree representation. If such partition is given, the construction of

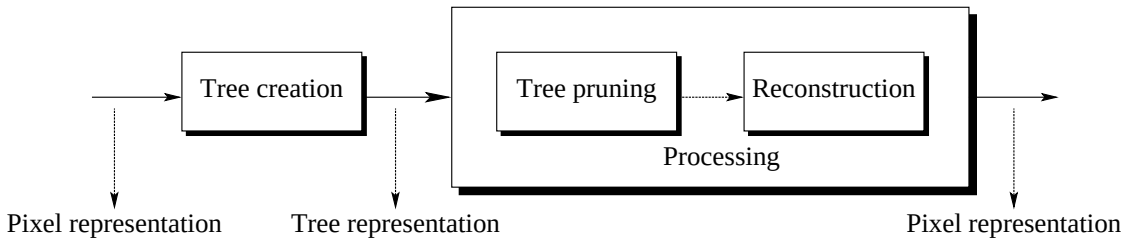

Figure 2.18: General scheme for processing images using trees.

the tree representation can be done for instance using the associated region adjacency graph as support. Note that in the latter case the tree construction is done in two stages: in a first stage, an abstraction from pixel to region is performed constructing the region adjacency graph of the partition, and in the second stage the hierarchical representation is created using the graph. If, however, the partition is not given, the tree can be constructed directly using the pixel information. In the latter case we may consider that the associated partition is composed of regions formed by individual pixels. The granularity of the partition is directly related (but not necessarily proportional) to the size of the tree. As the partition grows in number of regions, the number of nodes in the tree increases arising to a more complex tree. Therefore, when constructing the tree, a good compromise between fidelity and complexity should be taken.

From our practical experience, the tree creation is usually the computationally most expensive task from the scheme presented in Fig. 2.18. Thus, efficient algorithms for constructing the tree representation have to be developed. Note that the tree representation could be coded in order to store its associated coded stream on disc. Such approach may be useful if the tree should be processed several times as in the case of a database of trees. However, the efficient coding of the tree structure and associated information is out of the scope of this work.

As explained in the previous sections, in our work two types of tree representations have been developed: the Max-Tree and the Binary Partition Tree. Chap. 3 focuses on the creation of the Max-Tree, whereas Chap. 4 does it on the creation of the Binary Partition Tree.

2.5.2 Tree processing (Part II)

The second block is devoted to the processing of the tree. The processing stage can be subdivided into two stages. The first one is devoted to the analysis and pruning of the tree and the second is devoted to the construction of the pixel representation of the pruned tree.

Pruning

In this work we focus on processing techniques that lead to a pruning of the tree. For that purpose, the analysis algorithm makes use of descriptors that attached to the nodes of the tree. Such descriptors may include, for instance, area, perimeter or the geometry of the associated region. The strategy that is applied when pruning the tree depends on what the tree represents, see Sec. 2.2.3.

In Chap. 5 several tools are developed that enable applying filters and segmentation algorithms to the tree. It is shown that both filtering and segmentation can be implemented by using pruning algorithms. As it has been already mentioned in Sec. 2.2.3, pruning is characterized by the fact that if a node is to be removed, all of its descendants should be removed also. This issue is closely related to the notion of increasingness of the operator applied to analyze the nodes of the tree. This problem is discussed and a robust solution is proposed in Chap. 5. Finally, Chap. 6 discusses that tree based representations may be used to perform content based image retrieval.

Reconstruction

The objective of this block is rather simple and therefore no special chapter has been dedicated to it. Its objective is to restore a pixel based representation image from the tree(s) that have been output by the pruning algorithm. Depending on the application one may be interested in recovering a pixel based representation associated to the pruned tree (as in Chap. 5) or in recovering the image(s) associated to the pruned subtrees (as in Chap. 6).

Reconstruction is usually done by first constructing its associated partition. Most of the time we are going to be interested in reconstructing with the highest possible detail. For that purpose, the information associated to the leaf nodes of the pruned tree have to be used.

In other cases, one may be interested in obtaining a color image representing the pruned tree. For that purpose several approaches may be taken. Taking the pixel based partition, one may fill each region of the partition image with e.g. the mean color value associated to the region in the original image. Other possibilities are further described in this work.