The representation of images using scale trees

Javier Ruiz Hidalgo

School of Information Systems,
University of East Anglia
Norwich, NR4 7TJ, UK.
Tel: +44 1603 593257; fax: +44 1603 593345

October 22, 1999

Abstract

This thesis presents a new tree structure that codes the grey scale information of an image. Based on a scale-space processor called the *sieve*, a *scale tree* represents the image in a hierarchical manner in which nodes of the tree describe features of the image at a specific scales.

This representation can be used to perform different image processing operations. Filtering, segmentation or motion detection can be accomplished by parsing the tree using different attributes associated with the nodes.

Contents

Acknowledgements

I would like to thank my supervisor Prof. J. Andrew Bangham for his guidance throughout my MSc and the interest he has taken in all my work. Also thanks to Dr. Richard Harvey for all his comments and ideas.

My thanks go to Shaun McCullagh and SYS support for their assistance, without it this thesis would have never been printed. My thanks also go to all the people in the image and signal processing labs for their friendship and their help.

I would also like to thank all SYS staff for all their competence and their assistance.

Chapter 1

Introduction

A long-standing goal for a computer vision system is to extract a simple description in terms of meaningful objects from an image. Ideas of this type are currently being embodied in the proposed MPEG-4 and MPEG-7 standards where images are represented and conveyed in the form of independent audio-video objects.

Coding an image according to its visual objects should be an extremely useful step in the process of recognising objects in the image. An effective representation of an image would be an *object tree* in which nodes describe meaningful items of the scene and where nodes are represented in a hierarchical structure. To achieve this, systems capable of robust and fast segmentation must be created.

Of course, an optimal segmentation of an image cannot be achieved. The different regions of interest and their locations may vary depending on the application goals. The criteria to identify objects in the scene is also task dependent and the definition of an *object of interest* may differ from one user to another.

For instance, while coding an image, only the psycho visual quality of the segmentation and the available bandwidth are significant. A segmentation based on multi scale processors may be important as they segment the image into different scales. The best segmentation for a specific purpose can then be extracted given a quality threshold.

In this thesis, a new tree structure, based on a scale-space processor called the

sieve, is presented. A series of segmentations going from large to small scale are organised under a hierarchical tree structure, a *scale tree*.

The resulting scale tree, obtained from a grey scale image, forms a pyramid of increasing size objects where the nodes correspond to features of a particular scale. The tree structure itself is fairly insensitive to geometrical transformations of the original image. Different operations such as filtering, segmentation or motion detection can be performed by parsing the tree and using information associated with the nodes.

It is shown that scale trees can approximate *object trees* and that scale trees may be modified using other attributes to more closely approximate object trees.

The organisation of this thesis is as follows. Firstly, chapter 2 gives a general review of different image processing segmentation techniques. A brief background of scale-space processors and mathematical morphology operators is also presented. The following chapter gives a description of the sieve algorithm used to build the scale tree structure.

Chapter 4 presents some other hierarchical structures that have been described in literature. Scales trees are defined in chapter 5. Chapter 6 gives an overview of some applications using scale trees. Different methods are used to simplify the trees and new algorithms are described to refine scale trees in order to obtain closer representations to object trees. In the last chapter, conclusions are drawn.

Chapter 2

Review

This chapter introduces some general image processing algorithms from the literature that are related to the transformation is studied here. The *sieve* algorithm used to build the scale trees performs a *segmentation* of the original image by removing maxima or minima of the input function at specific scale. This chapter discusses other segmentation techniques that have appeared in literature.

2.1 Segmentation

One of the most common and difficult problems in image processing has been image segmentation and many image processing algorithms require a previous filtering of the original image or an initial segmentation to locate the interesting features of the image.

Here we distinguish image segmentation and recognition. If an object is recognised then it may be segmented. Segmentation, however, does not necessarily imply recognition. Recognition implies some image understanding since it associates semantic labels with part of the image but semantically meaningful labels are not needed for segmentation. For instance, one can segment a white cloud over a blue sky using colour information without knowing that the resulting segmentation is a cloud.

In the following sections, different segmentation techniques are reviewed.

2.1.1 Previous segmentation techniques

For the segmentation of intensity images, there are four main approaches that appear in the literature [1, 44], namely, threshold techniques, boundary based methods, region based methods and hybrid techniques that combine some of the above.

Threshold techniques

Threshold techniques [62, 83] are based on the postulate that pixels with similar values belong to the same region. The value used can be grey level information, colour or any other pixel attribute. A *threshold* is then applied to the image so different regions are segmented.

Threshold methods do not use any spatial or position information of the pixels within the image and therefore they do not perform very well in noise or blurred images.

Boundary based methods

Boundary based methods [29, 38] rely on a rapid change in the *boundary* between two regions. These techniques use the assumption that pixel values exhibit a fast transition between regions [81]. Examples of filters that use this property are the well known Sobel or Roberts edge detection filter [4]. The outputs of these filters provide good candidates for region boundaries.

Converting the filter output into closed region boundaries is, however, a difficult problem for these methods.

Region based methods

Region based techniques [77, 107] use *regions* of the image as their initial partition. They use the postulate that regions of the original image are, in a sense,

homogeneous. A homogeneity criterion [18, 19] is chosen taking into account that pixels within regions are more similar than pixels in different regions.

Some region based techniques include the *split-and-merge* [51] procedure or some standard region-growing techniques [3, 17, 106]. The general method is to compare every pixel with its neighbours. If the homogeneity criterion is satisfied, the pixels are assigned to the same class.

Using these techniques, the final result depends on the choice of the homogeneity criterion and such region based methods do not perform well with rich textured images.

Hybrid techniques

Alternative segmentation methods are the hybrid techniques, which combine region and boundary information [1, 17] and include variable order surface fitting [12], watershed segmentation [13, 103] or seeded region growing techniques [1, 71]. For example, the watershed approach considers the gradient of the image as a starting point for a *flooding* algorithm so edge and region techniques are combined.

These techniques have problems with blurred edges in the image. Obtaining markers for the regions of interest is also a difficult task to resolve.

2.1.2 Recent segmentation techniques

More recently, improved segmentation techniques have appeared in literature. In [96], for instance, the whole image is treated as a weighted connected graph. In this framework, the segmentation task is to find a partition in the graph such that similarities within subgraphs are high and similarities between subgraphs are low.

In this case, for an image of $n \times n$ pixels, the computational cost of the algorithm is mainly finding the eigenvectors of an $n \times n$ matrix. However, this method only uses a few eigenvectors to perform the segmentation so the computational cost is significantly reduced [97].

More recently operators, such as Active Shape Models [23] can also be interpreted as new high order segmentation methods. In this case, the final goal of the method is to locate a deformable shape in the original image. Shape information is used to train a probabilistic model that constrains the fitting to the target image. Shape, content information and a probabilistic background (a set of training images are used to build the model) are applied together to perform the recognition.

In a recent extension, grey scale information inside the distributed model, called *active appearance model*, has been added with promising results in segmentation, tracking and recognition tasks [22]. However, we note that recognition is implicit in the ASM approach and so, as with all recognition algorithms, segmentation should follow automatically.

2.2 Scale-space

A sieve is a method for scale-space analysis of an input signal. Scale-space methods process the image with scale as a parameter [33, 34, 104]. In a conventional scheme, scale may be associated with resolution desired in the output function. Figure 2.1 shows the scale-space representation of an image.

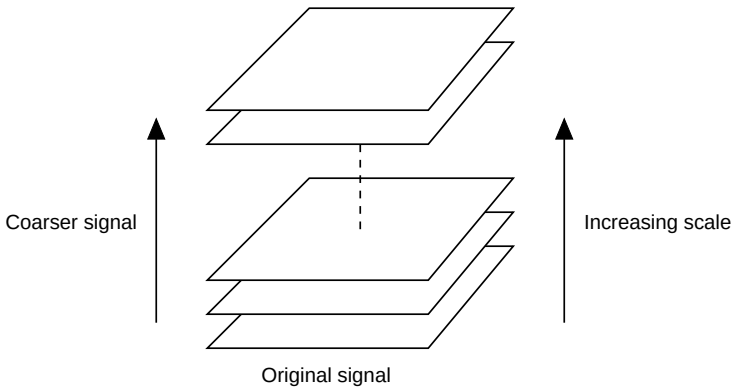


Figure 2.1: Scale-space filtering of a signal [66].

Scale-space processors simplify the image as the scale parameter increases, in a manner that demands that no features are allowed to be introduced by the scale-

space processor itself. This is accomplished by the *causality* property. If further constrains, such as *causality* and *isotropy*, are applied, the diffusion equation presented by Koenderink in [61] is satisfied.

$$\nabla (c \nabla f) = \frac{\partial f}{\partial s} \tag{2.1}$$

where s is scale. If the diffusivity parameter, c, is constant the equation above becomes the linear diffusion equation,

$$\nabla^2 f = f_s \tag{2.2}$$

The resulting scale-space processor, however, may introduce new extrema on the resulting signal (section 3.4) and edges of the original image will be blurred on the simplified output.

2.3 Morphological methods

The sieve algorithm used in this thesis does not rely on any linear filtering. Mathematical morphology [45, 46, 68, 93, 99], as a group of nonlinear filters, provides the theoretical framework and is the analysis of signals by shape, and has been developed by Serra [92–94] from work by Matheron [69] and Blum [15].

One of the most important difference between the linear filters described before and nonlinear morphological filters is in *edge preservation*. Morphological filters remove features of the original image while leaving the edges untouched.

2.3.1 Flat zones

The sieve algorithm studied in this thesis uses *connected sets* or *flat zones* [25,28]. It works by removing (i.e. merging) connected sets of the original image. A connected set or a flat zone is a group of *connected* pixels with the same intensity.

There are two different types of neighbourhood connectivity [57], 4-connected and 8-connected pixels. Figure 2.2 shows these.

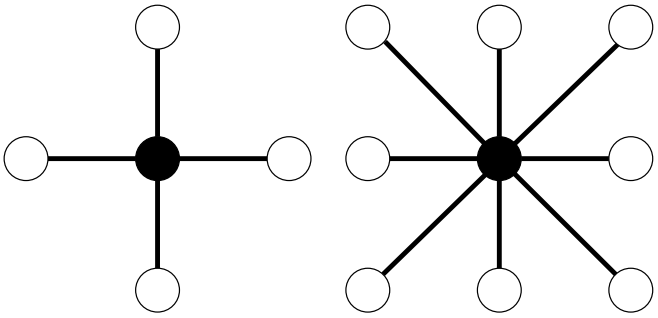


Figure 2.2: The left image shows the 4 connected pixels (in white) of the centre pixel (in black). The right image shows the 8-connected case.

A flat zone set represents a group of connected pixels (4 or 8-connectivity) with the same grey scale value. Figure 2.3 shows an example of different flat zones or connected sets of an image. Notice that there are two connected sets with value 3 as they form two separated regions.

3	3	3	3
3	3	4	4
2	2	4	4
2	2	3	3

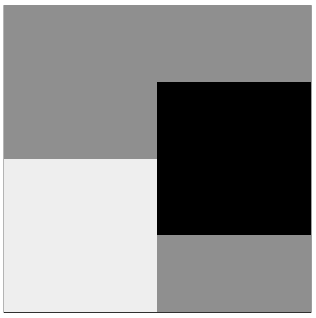


Figure 2.3: A figure showing the *flat zone* idea.

2.3.2 Connected operators

A *connected operator* [26, 75, 91, 95] is an operator that works with connected sets of the original image. In this sense, the sieve may be defined as a connected operator. Connected operators have been introduced in literature as *morphological filters by reconstruction* [24, 28, 102].

Salembier reported [91] that the first reference to connected operators, known as *openings by reconstruction*, appeared in literature in 1976 [56]. Initially, they eroded a binary image by a connected structuring element and *reconstructed* all connected components that were not removed completely by the erosion. These '*by reconstruction*' operators involved not only openings but also closing, alternating filters or alternating sequential filters.

Later, they were extended to grey scale images [101, 102] and different simplification criteria were obtained from the use of these operators, such as size-sensitive multiresolution decompositions [85, 90], area openings [101], geodesic operators [64], size [94] and complexity [75].

To define a connected operator, the notion of an *associated partition* must be introduced [95]. A partition of the image I is a set of disjoint connected components of I, $\{P_i\}$, the union of which is the entire image (each P_i is called a partition class). A partition $\{P_i\}$ is *finer* than another partition $\{Q_i\}$ if any pair of points belonging to the same class P_i also belong to an unique class in Q_i. The associated partition of an binary image X is made of all connected sets of the image and their complement, X^C.

DEFINITION 1 *A binary operator ψ is connected if, and only if, for any binary image X, the associated partition of X is finer than the associated partition of $\psi(X)$.*

In the case of grey level connected operators, the associated partition is defined using the flat zones of the image. The *partition of flat zones* of a grey function f is defined as the set of the largest connected components of the space where f is constant. Using this, the definition of a grey level connected operator is identical to the binary operator.

DEFINITION 2 *An operator Ψ is connected if, and only if, for any grey level function f, the partition of flat zones of f is finer than the partition of flat zones of $\Psi(f)$.*

The sieve operator only *removes* connected components of the image (maxima or minima at specific scale). As any connected operator, it simplifies the image while preserving the remaining contours. These connected operators have relations with structured representations such as region adjacency graphs [79] or trees [87]. In this thesis, a tree structure, called *scale tree*, derived from the sieve algorithm will be analysed.

Chapter 3

Sieves

A *sieve* is a nonlinear scale-space decomposition algorithm [5–9] that shares similarities with some mathematical morphology operators (section 2.3). It is a filter that removes extrema from an input signal at a specific scale s. At every stage in the filtering process, extrema of specific scale are removed from the original function. So, for instance, the filter stage $\mathcal{S}_1$ removes extrema of scale 1, $\mathcal{S}_2$ scale 2 and so on until the maximum scale of the original signal m is reached and the resulting filtered image is flat.

The removed extrema form *granules* that are stored in a new domain, (called *granularity*). This domain has been shown to preserve scale-space causality (Theorem 6.36 in [7]) and it is also invertible (Theorem 6.49 in [7]).

There are different types of filtering element, $\mathcal{S}$. Section 3.2 defines the different kind of operators used in this thesis together with some properties of these filters (section 3.4). The following section provides a more formal description of the sieve algorithm.

3.1 Sieve algorithm

The basis of the sieve algorithm considers an input function as a graph [49, 100]. Let $G = (V, E)$ be a graph, where V represents the set of vertices and E the edges

between those vertices. Figure 3.1 shows this concept for a one-dimensional case. The vertices V are the samples of the function and the set of edges E correspond to the adjacencies between points. In this example, the samples of the function are labelled with numbers so $V = \{1, 2, 3, \ldots, 11\}$ and the edges represent the connections between the points $E = \{\{1, 2\}, \{2, 3\}, \{(3, 4\}, \ldots, \{10, 11\}\}$ where $\{1, 2\}$, or $\{2, 1\}$, represents the edge or connection between vertices 1 and 2 in the graph.

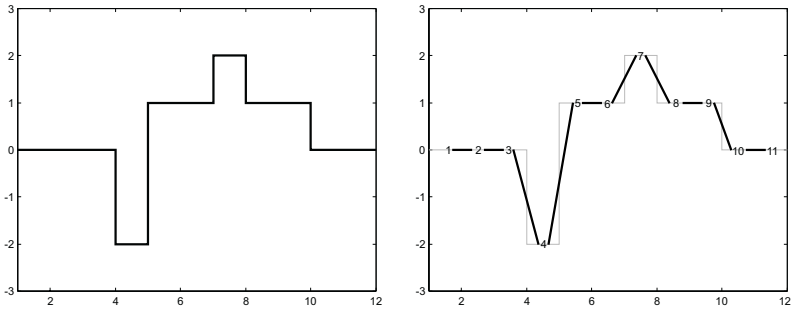


Figure 3.1: A 1D signal represented as a graph (G_{1D}).

Figure 3.2 shows the same concept for an image, in this case, the vertices of the graph correspond to all the pixels of the image. The set of edges E define the neighbourhood or connectivity of those pixels. In this example, a set of 4-connected edges are used (section 2.3). The graph of this image would be $G_{2D} = (V, E)$ with,

$$V = \{1, 2, 3, \ldots, 16\}$$

and the edges,

$$E = \{\{1, 2\}, \{1, 5\}, \{2, 3\}, \{2, 6\}, \ldots\}$$

The algorithm works by defining subsets, or connected regions, inside the graph representation. If $C_r(G)$ is defined as the set of connected subsets of G with r elements. Then, the region $C_r(G, v)$ over the graph G which encloses the pixel (vertex) v can be defined as:

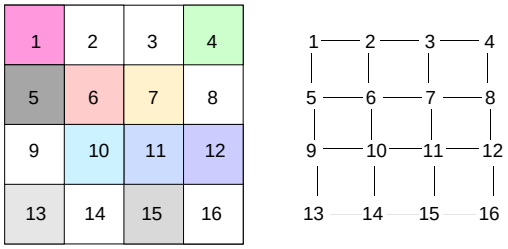


Figure 3.2: Image represented as a four-connected graph (G_{2D}).

$$C_r(G, v) = \{\xi \in C_r(G) \mid v \in \xi\} \tag{3.1}$$

where $C_r(G, v)$ is the set of connected subsets of r elements that contain v, or, simply, the connected regions of r vertices that contain the vertex v. For instance, in the 1D example of figure 3.1, the set of regions with 2 elements containing vertex 7 is ($r = 2$,$v = 7$),

$$C_2(G_{1D}, 7) = \{\{6, 7\}, \{7, 8\}\}$$

for the two-dimensional, four-connected, example (figure 3.2),

$$C_2(G_{2D}, 7) = \{\{7, 3\}, \{6, 7\}, \{7, 8\}, \{7, 11\}\}$$

Using the 2D example again, the regions of 3 elements containing pixel 1 are ($r = 3$,$v = 1$),

$$C_3(G_{2D}, 1) = \{\{1, 2, 3\}, \{1, 5, 9\}, \{1, 2, 6\}, \{1, 5, 6\}\}$$

For each integer $r \geq 1$, the sieve filter $\mathcal{S} : \mathbb{Z}^V \mapsto \mathbb{Z}^V$ can be defined to operate over the connected regions $C_r(G)$ of the graph. As mentioned earlier, the sieve algorithm removes *extrema* of a specific *scale* from an input signal f, so *length* is

used in the 1D case or *area* in 2D and so on for higher dimensional signals.

The structure of a sieve decomposition is shown in figure 3.3. Each stage $\mathcal{S}_s$ removes extrema of increasing scale and the output function of every stage $f_s = \mathcal{S}_s(f)$ is used as the input signal of the next stage at scale $s+1$ (equation 3.2).

$$f_{s+1} = \mathcal{S}_{s+1}(f_s) \tag{3.2}$$

where the initial function is,

$$f = f_0 = \mathcal{S}_0(f) \tag{3.3}$$

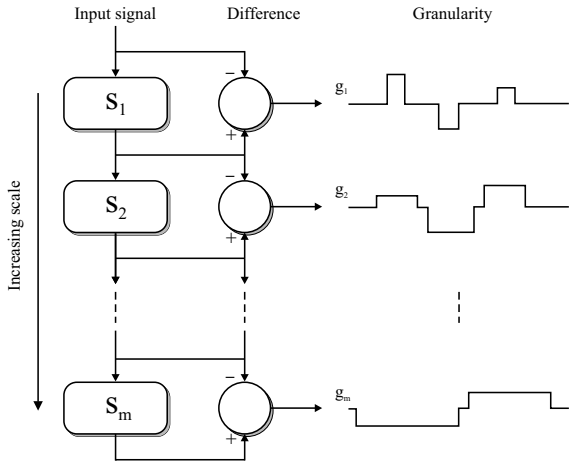


Figure 3.3: The complete sieve decomposition of a signal f.

Equation 3.4 shows the definition of a complete sieve algorithm over a signal $\mathcal{CS}(f)$. A cascade of increasing scale sieve algorithms are applied until no new maxima or minima are found. If m represents the scale of the largest granule in the input signal f, the complete sieve sequence can be defined as,

$$\mathcal{CS}(f) = \mathcal{S}_m\mathcal{S}_{m-1}\ldots\mathcal{S}_2\mathcal{S}_1(f) \tag{3.4}$$

The *granule function* is defined as the difference between two stages of the sieve filter (equation 3.5). 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.

$$g_s = \mathcal{S}_{s-1}(f) - \mathcal{S}_s(f) = f_{s-1} - f_s \tag{3.5}$$

The morphological operators defined in section 2.3 can be re-defined to operate on connected regions using the standard graph notation. The sieve operator, $\mathcal{S}_s$, can then be any of the morphological filters described before. Re-defining these filters to use connected regions implies that the morphological operators no longer use rigid structuring elements. Instead they use all sets of connected subsets defined by the function $C_r(G)$ as *flat* structuring elements.

For instance, in the 1D case, equation (3.1) becomes the set of intervals containing r elements,

$$C_r(G_{1D}, x) = \{[x,\ x + r - 1] \mid x \in \mathbb{Z}\} \qquad r \geq 1 \tag{3.6}$$

which is identical to filtering using a flat structuring element of length r with output at the middle pixel.

The following section discusses the new notation of these morphological filters together with some examples of their operation.

3.2 Types of sieves

The sieve algorithm can use any morphological operators (section 2.3). The only restriction imposed to these filters is that they must be idempotent (see section 3.4). As the sieve operator removes extrema of scale s, no smaller features are allowed to remain in the original signal so $S_s(S_s(f)) = S_s(f)$. This means that morphological opening, closing, $\mathcal{M}$-filters and $\mathcal{N}$-filters can be used but not

operations such as dilations or erosions as they are not idempotent.

Opening

The morphological opening is defined over a graph as,

$$\gamma_r f(x) = \max_{\xi \in C_r(G,x)} \min_{u \in \xi} f(u) \tag{3.7}$$

As γ_r operates on a set of r elements $C_r(G)$, it can only remove *maxima* of size $r-1$ and, for instance, γ_2 will remove all maxima of scale 1. γ_1 has no effect on the original signal $f = \gamma_1(f)$. To keep our notation, we will still describe a opening sieve (o-sieve) of scale one as S_1 so scale of size 1 is removed, remembering that an operation of γ_2 needs to be applied.

Figure 3.4 shows a 1D example of a opening sieve of scale 1. Maxima of scale 1 (positions $8-9$ and $11-12$) are removed.

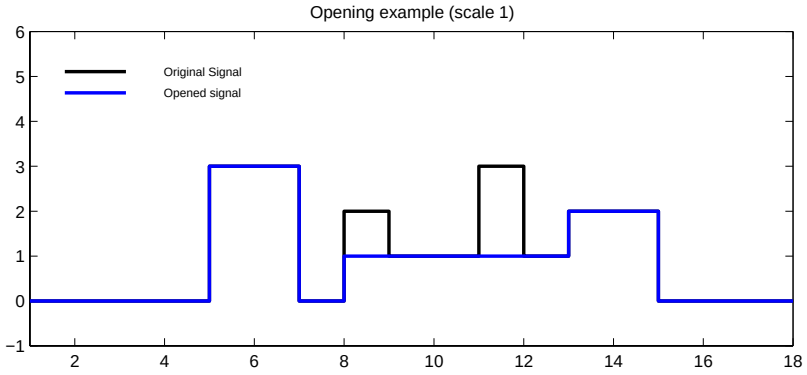


Figure 3.4: Example of a opening sieve (o-sieve) of scale 1.

Closing

As the opening filter is defined to operate over subsets of a graph G, the morphological closing is defined as its dual operation,

$$\varphi_r f(x) = \min_{\xi \in C_r(G,x)} \max_{u \in \xi} f(u) \tag{3.8}$$

and it removes *minima* in the subsets of $C_r(G)$ so φ_r can only remove minima of scale $r-1$ (as the opening filter). The closing sieve, or c-sieve, removes minima (and only minima) of scale s from the input signal (positions $7-8$ and $12-13$ in figure 3.5)

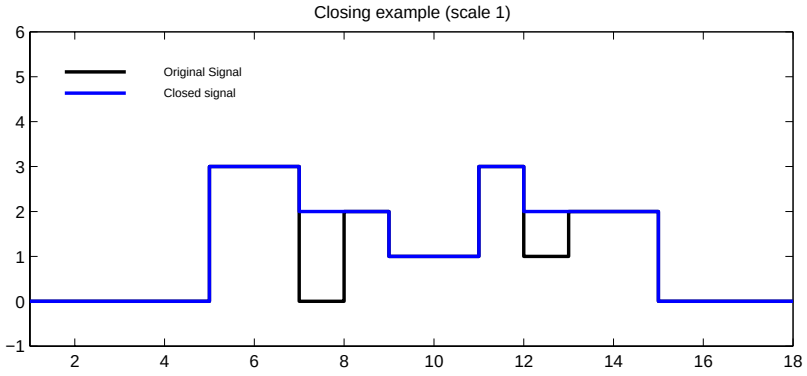


Figure 3.5: Example of a sieve closing (c-sieve) of scale 1.

$\mathcal{M}$-filter

The $\mathcal{M}$-filter consists of an opening followed by a closing. Equation (3.9) shows the definition of an $\mathcal{M}$-filter operating on a graph.

$$\mathcal{M}_r f(x) = \varphi_r(\gamma_r f(x)) = \min_{\xi \in} \max_{C_r(G,x)} (\max_{u \in \xi} \min f(u)) \tag{3.9}$$

An M-sieve filter removes extrema of scale s by removing maxima (opening) and then minima (closing) at that scale. Figure 3.6 shows an example of the

operator removing extrema of scale 1 (M-sieve$_1(f)$).

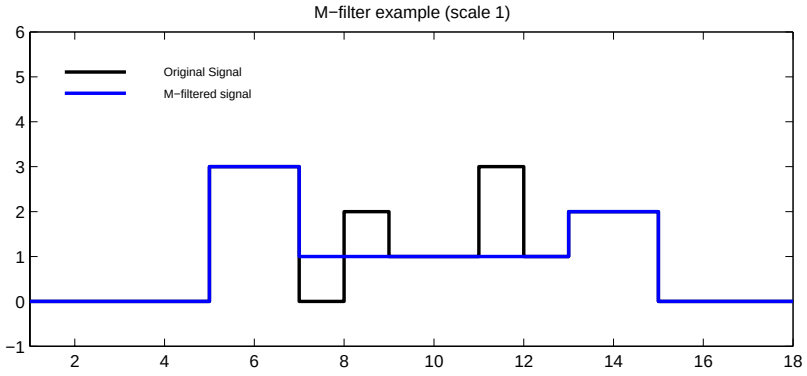


Figure 3.6: Example of a $\mathcal{M}$-filter sieve (M-sieve) of scale 1.

$\mathcal{N}$-filter

The morphological $\mathcal{N}$ filter is defined as a closing followed by an opening,

$$\mathcal{N}_r f(x) = \gamma_r(\varphi_r f(x)) = \max_{\xi \in \ C_r(G,x)} \min (\min_{u \in \xi} \max f(u)) \tag{3.10}$$

and removes minima and then maxima at a scale s. The resulting sieved signal may differ from the M-sieved output if extrema of different sign (maxima / minima) are found in the same connected set $C_r(G)$. Figure 3.7 shows the same 1D input signal filtered this time with an N-sieve. As minima are processed first, positions $7-9$ go to a higher level than when the M-sieve is used.

As opening and closing sieves, M and N-sieves remove extrema of scale $s = 1$ when a filter of size $r = 2$ is used.

Recursive median filter

The recursive median filter [70] is a one-pass approximation to the root median filter [74], a median filter applied to the input signal until no further change occurs. Equation 3.11 shows the definition of a 1D recursive median filter using a window of r samples, where r is odd.

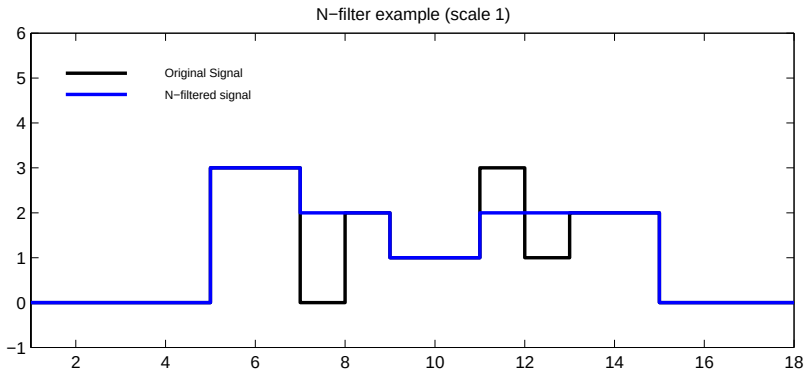


Figure 3.7: Example of a $\mathcal{N}$-filter sieve (N-sieve) of scale 1.

$$\rho_r f(x) = \begin{cases} median\{ \quad \rho_r f(x - \frac{r-1}{2}), \rho_r f(x - \frac{r-1}{2} + 1), \ldots , \\ \qquad \ldots , \rho_r f(x-1), f(x), f(x+1), \ldots , & x \geq 0 \\ \qquad \ldots , f(x + \frac{r-1}{2} - 1), f(x + \frac{r-1}{2})\} \\ \\ \quad 0 & x < 0 \end{cases} \tag{3.11}$$

so, for example, when $r = 3$,

$$\rho_3 f(x) = median\{\rho_3 f(x-1), f(x), f(x+1)\} \tag{3.12}$$

the filter is the median of $f(x), f(x+1)$ and the previous output.

The recursive median filter retains all of the properties of the multiple pass root median filter such as noise rejection and idempotency [9], and therefore, is a valid operator to use with the sieve.

The recursive median filter of r samples can only remove extrema of $s = (r-1)/2$ (r must be odd). So, the filter, ρ_r, removes only extrema of size $s = 1$ when a window of size $r = 3$ is applied (a window of $r = 3$ samples is needed to achieve the same results as a 2 pixel window in o-, c-, M- or N-sieve filters). The recursive median sieve operator r-sieve will remove scale of size s, so it will use

a window of $r = 2s + 1$ samples.

Figure 3.8 shows an example of a recursive median sieve removing extrema of size 1. The median filter scans the image using the 'implicit' left to right order in 1D. For higher dimensional signals, an order has to be imposed. In the 2D case, a left-to-right, top-to-bottom scan leads to a very similar results to the M- and N-sieves.

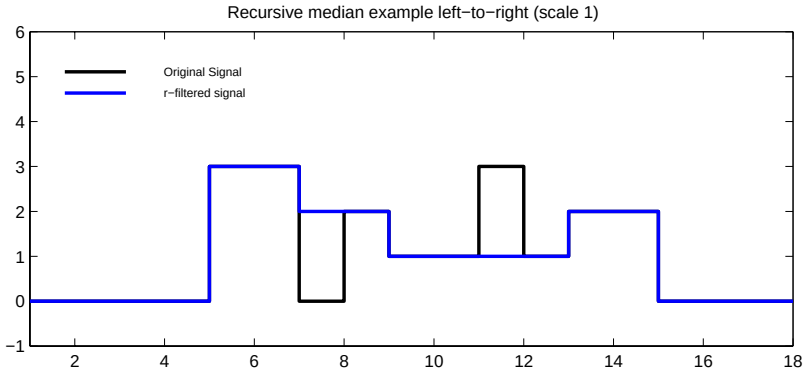


Figure 3.8: Example of a recursive median sieve (r-sieve) processing extrema from left to right.

Note that the output of the recursive median filter depends on how the extrema are positioned in the input signal and, if the original signal f is scanned in any other order, different outputs may occur. Figure 3.9, for instance, shows a 1D r-sieve using a right-to-left scan, positions $7 - 9$ and $11 - 13$ differ from the results using a left-to-right scan.

The different sieves defined are summarised in table 3.1.

Filter	Symbol	Sieve	Extrema Processing
opening	γ	o-sieve	maxima
closing	φ	c-sieve	minima
$\mathcal{M}$-filter	$\mathcal{M}$	M-sieve	bipolar $\pm$
$\mathcal{N}$-filter	$\mathcal{N}$	N-sieve	bipolar $\mp$
recursive median	ρ	r-sieve	bipolar random

Table 3.1: Sieve types.

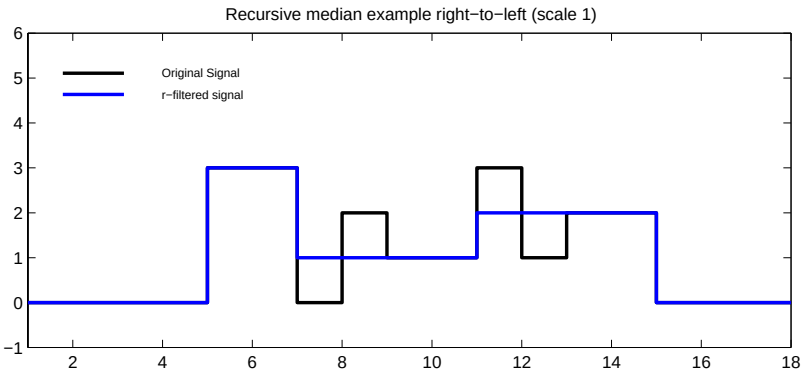


Figure 3.9: Example of a recursive median sieve (r-sieve) processing extrema from right to left.

3.3 Examples

In this section, more advanced examples of the sieve operator will be presented. As defined in equation (3.4) and (3.5), a complete sieve decomposition transforms the input signal into a new different domain called *granularity*, $\mathcal{G}$, and it is composed of all removed granule functions g_s at all scales.

$$\mathcal{G} = \{g_1, g_2, \ldots, g_m\} \tag{3.13}$$

Figure 3.10 shows the granularity decomposition of the same signal used in the previous examples. An M-sieve operator is used to filter the input signal until no more extrema are found, in this case, at scale 5. At every y coordinate the granule function g_y is represented, for scale y.

As the morphological filters have been defined in any number of dimensions using the graph notation, sieves can be extended to operate with N-dimensional signals. The left of figure 3.11 shows a very simple image. On the right is a 3D visualisation of the same image where height is grey scale value so extrema values form mountains or valleys in the surface.

A complete sieve decomposition of this image is shown in figure 3.12. An

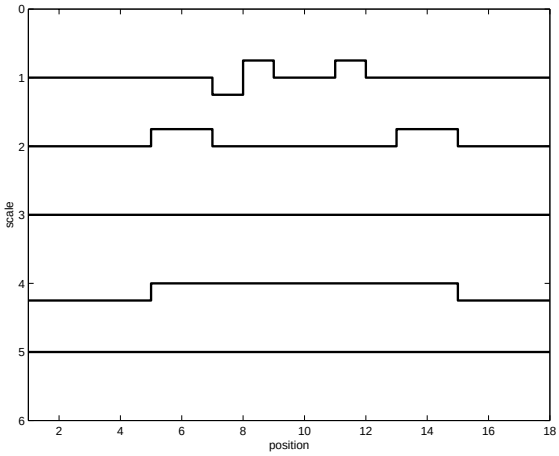


Figure 3.10: Granularity decomposition of a 1D signal using the M-sieve.

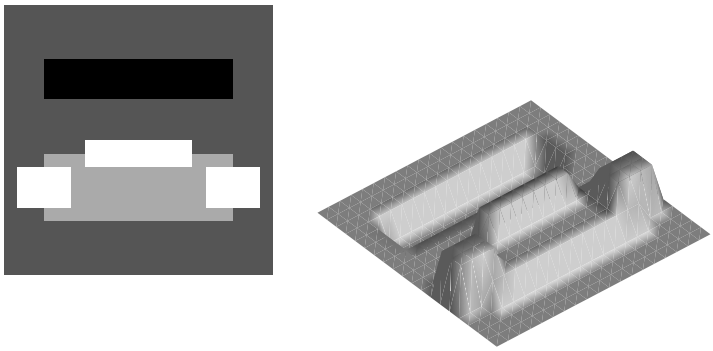


Figure 3.11: Original 2D image for the 2D sieve decomposition. On the left, the grey scale value of the pixels. On the right, a 'mesh' representation of the original.

M-sieve filter is used to segment the image. For clarity, only scales where the granule functions, g_s, are different than zero are shown. The left column shows the filtered image while right column shows the granule functions.

To conclude this section, a last example with a real image will be analysed. In this thesis, the sieve algorithm is used to build a tree representation, called a *scale tree*, of an image. This new structure keeps information about the scale of the image by removing maxima and minima. As extrema of both signs (maxima and minima) are removed, opening and closing sieves cannot be used.

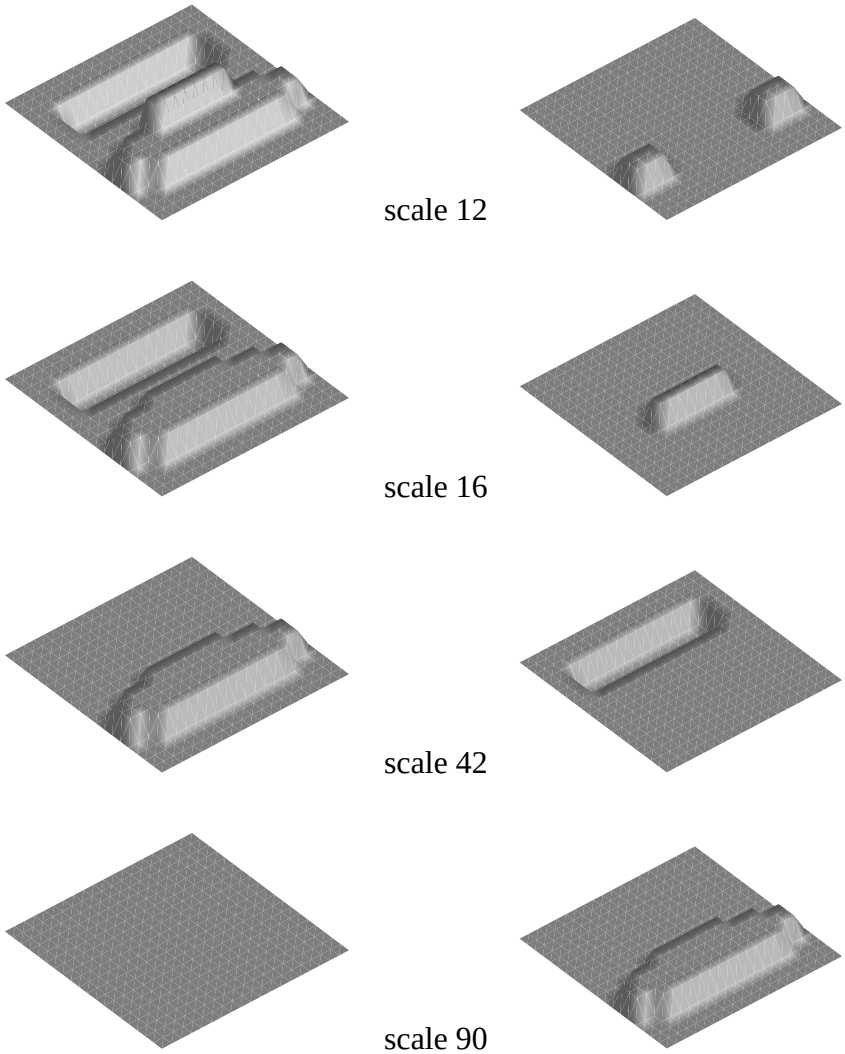


Figure 3.12: Example of a sieve decomposition in 2D. Left column shows the filtered images M-sieve(I). Right column shows the granules removed at each stage.

The r-sieves operators were not chosen to build the tree structure as they exhibit some variations under rotations or other transformations of the original image. Since there are few differences between using M and N operators, M-sieves were chosen arbitrarily. A 4-connected 2D sieve was used (the results are rather similar to using 8-connected sets (see section 2.3)). Figure 3.13 shows an example

of the M-sieve operator at different scales.

Figure 3.13: M-sieve decomposition of the image on the left. Middle and right images sieved up to scale 500 and 2500 pixels.

3.4 Properties

It has been shown that the sieve is a nonlinear scale-space selector that removes extrema of increasing scale from the input signal. The sieve is therefore a good candidate to perform an initial segmentation of the image based on scale. The sieve filters also have properties that make them interesting for some image processing tasks. This section will discuss some of the more important properties together with relevant example images. For a formal description and a proof of these properties see [7] and [9].

Idempotency

The sieve algorithm removes extrema of size s from an input signal. Since no smaller features are allowed to exist after the current scale filter has been applied, the sieve operator must be idempotent,

$$f_s = \mathcal{S}_s(f) = \mathcal{S}_s(\mathcal{S}_s(f)) \tag{3.14}$$

or, in other words, applying the sieve filter at scale s implies filtering the signal at all lower scales until scale 1.

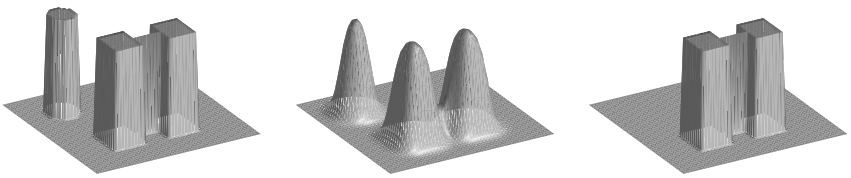
$$f_s = f_{s-1} f_{s-2} \ldots f_2 f_1 = \mathcal{S}_s \mathcal{S}_{s-1} \ldots \mathcal{S}_2 \mathcal{S}_1(f) \tag{3.15}$$

Scale-space causality

N-dimensional opening and closing-sieve operators preserve scale-space causality [61] as they do not introduce any new extrema to the original signal. Using the graph notation again, if $\mathcal{S}_s$ is a sieve filter operating over a signal $f \in \mathbb{Z}^V$ and $\{x, y\} \in E$ is a particular edge. As s increases, the difference $\delta_s = f_s(y) - f_s(x)$ does not change sign and its absolute value decreases.

$$\delta_1 \leq \delta_2 \leq \ldots \leq 0, \quad or \quad \delta_1 \geq \delta_2 \geq \ldots \geq 0 \tag{3.16}$$

Figure 3.14 shows a well known example due to Lifshitz and Pizer [65] that illustrates scale-space causality. The original image is, on the left hand side, with 2 extrema, the circle on the top and the two squares connected by a thin isthmus. The standard linear diffusion system [2, 66] does not enhance *local* extrema but reduces the amplitude of the isthmus and hence creates three maxima where there were two. The M-sieve (on the right) treats the squares as one object and so, preserves scale-space causality.



A) *B)* *C)*

Figure 3.14: *A)* shows the original image with two extrema. *B)* Linear diffusion systems introducing new extrema (now the two squares form two different extrema) . In *C)*, The sieve algorithm leaves the two squares untouched so no new extrema are introduced.

Invertibility

One of the most powerful properties of scale sieves is that they are invertible and the original image can be reconstructed from the granularity domain. Using the example of figure 3.12, the original image can be reconstructed by adding all granules (right column) over all scales,

$$f = \sum_{s=1}^{m} g_s = g_1 + g_2 + \ldots + g_m \tag{3.17}$$

where m is the maximum scale of the signal.

However, in the reconstruction of the original signal, the DC component of the signal is not retained as the maximum intensity of the image is lost (only the difference is stored). There may be an offset between the original image and the reconstructed image. To solve the problem, the signal can be padded with zeros, so no more extrema are introduced and the DC component is eliminated. 1D padding with zeros implies adding zeros to the start and end of the signal. In 2D padding a border of zeros is placed around the image with area greater than the size of the image.

Figure 3.15 shows a 1D example of this problem. On the left, the original signal with an offset of 4, on the right, the 'zero-padded' signal. The bottom row shows the reconstructed signals using the M-sieve. Note that the reconstructed 'unpadded' signal does not preserve the original offset. The reconstructed 'padded' signal, however, is identical to the original, zero-padded signal.

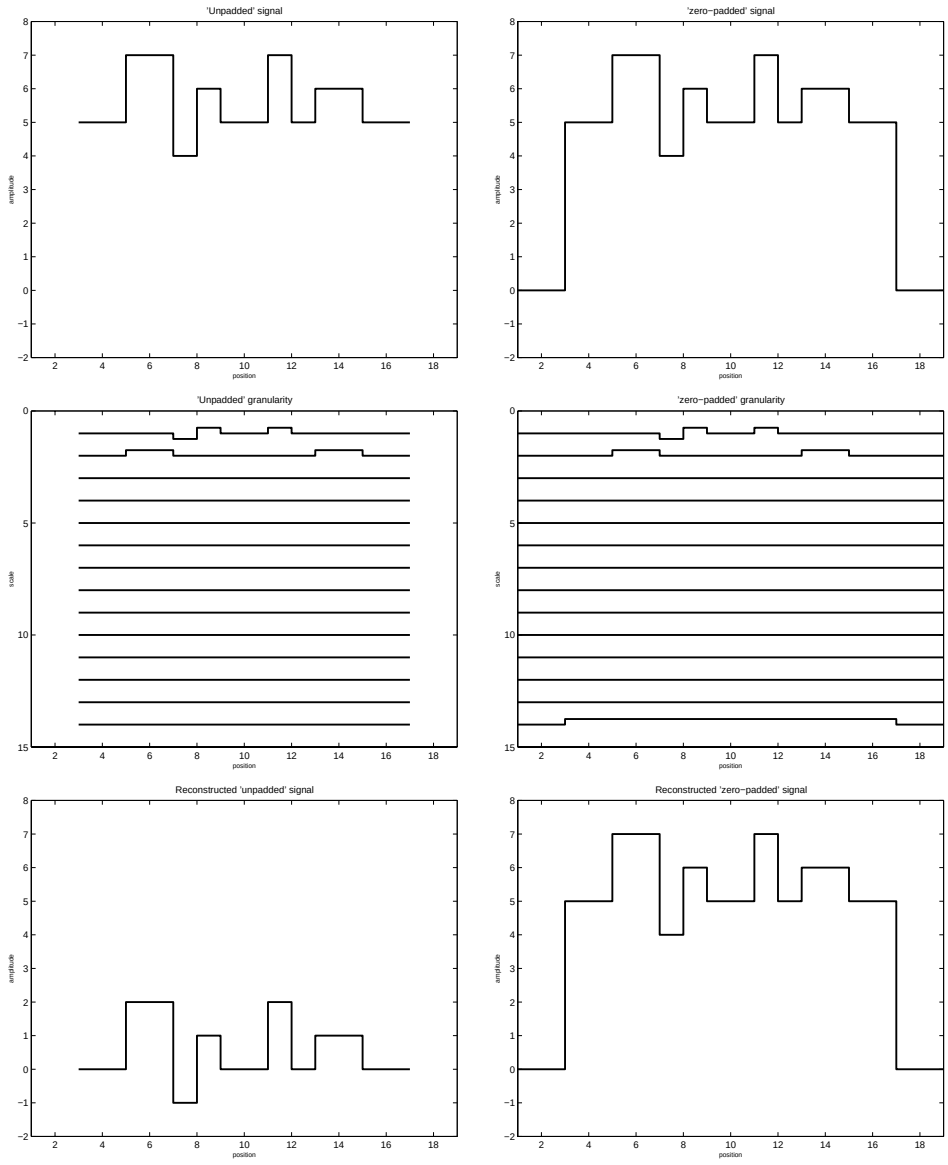


Figure 3.15: Demonstration of the invertibility of sieves. On the left the original image with an offset, on the right the zero-padded signal. Middle row shows the granularity decomposition of the two signals. Bottom row shows the reconstructed signals. Note that the ‘unpadded’ reconstruction does not preserve the DC component of the original.

Chapter 4

Trees in image processing

Following the description of the sieve algorithm on chapter 3, this chapter introduces the idea of a tree structure representing an image. Descriptions of some useful trees currently used in image processing will be added. These descriptions will provide a background to the use of sieves in building the new scale tree structure described in this thesis.

Trees are a commonly used data structure in computer graphics where Binary Space Partitioning trees [20], decision trees [43] and other relational structures have been used to describe all the objects within a scene (such as VRML formats or BIFS planes on MPEG-4). In the case of image processing, these structures have not been applied as extensively but, in the last few years, these have become more popular.

The following sections will present some trees currently being used in image processing. As will be seen, some of them share similarities with the tree structure developed in this thesis in the sense that they store the same information or that the tree building algorithm is similar. In each case, the advantages and disadvantages of using such structures will be analysed and some applications showing where and how the trees are used will be presented.

4.1 General image processing trees

Several trees have appeared in literature relating to different image processing tasks. This section briefly discusses some of them by comparing the algorithm used to build the trees as well as the properties that make them interesting for some applications in image processing.

4.1.1 Directed trees

Directed trees first appeared in the image processing literature in 1976 in a paper by Koontz and Narendra [63] and have since been applied to different segmentation techniques [73]. Every node in a directed tree represents one pixel of the original image. Figure 4.1 shows an example of two directed trees, T_1 and T_2, completely segmenting an image.

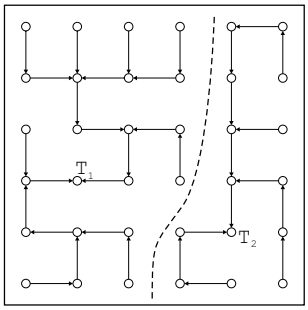


Figure 4.1: An illustration of a directed tree with image points as nodes. Two different directed trees, T_1 and T_2, segment the image (dotted line) in two disjointed regions.

The algorithm used to build directed trees operates on an edge enhanced image, computed from the original. This edge function establishes the *directed* links between pixels (arrows of the figure). If the corresponding edge image computed from the original, of $N \times M$ pixels, is,

$$E = \{e(x, y), \quad x = 1, \ldots, N; y = 1, \ldots, M\} \tag{4.1}$$

the algorithm computes an 'edge gradient' function $G(x, y)$ used to identify the

'closer' adjacent pixel to establish a tree link.

$$G(x,y) = \max[e(x',y') - e(x,y)] \quad (x',y') \in n(x,y) \tag{4.2}$$

with $n(x,y)$ being the neighbourhood pixels of a point (x,y) of the image.

The algorithm segments the original image by expanding the directed tree over uniform regions in the edge image. Each resulting directed tree T_i forms a different region in the segmented image (figure 4.1).

4.1.2 Containment trees

Containment trees are a different approach to building a tree structure of an image. They represent the image by storing *containment* information. They have been used in some shape based recognition in pictorial databases [59]. Each node in a containment tree stores information about a contour, C_i, of the original image. The tree structure itself is built by creating child nodes for contours *contained* or enclosed in a bigger contour (it's father).

Figure 4.2 shows three examples of these trees. As it can be seen, the containment tree is a very simple representation of a black and white image where containment information is stored in a tree structure. Such a simple tree leads to the same representation for a wide variety of different images, see for example the two first columns in figure 4.2. At the same time, the use of this structure is invariant to strong transformations of the original images such as zooms, rotations or small occlusions.

It is obvious that this representation cannot be used for a complete picture system retrieval. The containment tree gives good results if it is used to make a preliminary discrimination of the search space [59] for other, higher order, and usually computational more expensive systems.

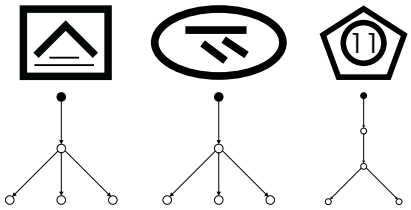


Figure 4.2: Three containment trees. Left and middle columns having the same tree. Right column illustrating a different containment tree with the same number of contours. The black node corresponds to a 'dummy' contour including all the curves and acting as *root* of the tree.

4.1.3 Shock trees

Shock trees appeared in literature in [98] as a graph derived from a set of *shock* measures [55] in the curvatures of the image. The shock tree is built by analysing the singularities of a curve evolution process acting on simple closed curves of a binary image.

The singularities of a curve are measured by the variation of the radius function associated with each curve in the image. The variations are classified into four types (figure 4.3). These shock variations are then transformed into a hierarchical structure. A third order shock, for instance, has two children formed by first order shocks.

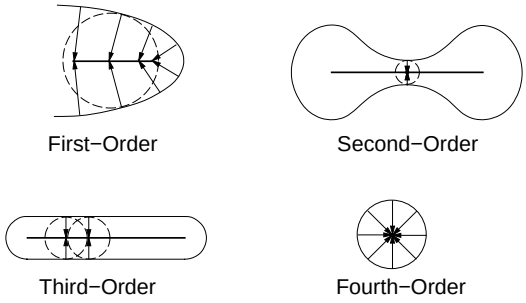


Figure 4.3: Four different types of variations of closed curves. A first order shock appears on curve segments. Second order shocks arises at necks, followed by first order shocks in each direction. A third order appears on straight lines. Four order shocks appear in circles [98].

Shock trees have been applied to object classification systems based on sil-

houettes images. The tree matching algorithm uses topology alone [78] but other techniques using label information has been also studied [98].

4.2 Scale-space trees

Trees relating to scale-space processors (see section 2.2) have appeared in literature in structures, such as *quad trees* or *oct trees*. They share the use of a scale decomposition as a main factor to build the trees. The following sections briefly discuss these representations.

4.2.1 Quad and Oct trees

Quad trees are one of the earliest multi-scale representation of an image and were introduced by Klinger in 1971 [58]. The algorithm to build a quad tree is similar to the *split-and-merge* algorithm [18, 19, 51, 77] and it works recursively by dividing the original image I into blocks (B_i) [53, 54].

Firstly, the whole image is treated as a single block of size $2^n \times 2^n$ pixels, a formal criterion is used to decide if the pixels within a block are similar enough. If the block being analysed satisfies the homogeneity criterion, the block is kept and the algorithm stops. If it is not, the block is divided into m similar regions and the algorithm is applied again on every created subregion.

To satisfy the criterion, a measure of pixel value variation is taken and if it is greater than a certain threshold u then the block is subdivided. Usually, the measure used is proportional to the grey scale variance of the pixels $g(x, y)$ in all regions. Generally measures like the difference between maximum and minimum pixel value in that block (maximum intensity),

$$|g_{max}(x, y) - g_{min}(x, y)| \quad \forall (x, y) \in B_i \tag{4.3}$$

or a measure of the standard variance of the regions pixels (equation 4.4) will be used.

$$\sigma^2 = \frac{\sum_{(x,y)\in B_i} (g(x,y) - \mu)^2}{B_i} \tag{4.4}$$

For simplicity, the original image is divided in sub quarters ($m = 4$). Figure 4.4 shows an example of a quad tree encoding of regions over a simple binary image X. The variance measure (4.4) is used with a threshold equal to zero (dividing white from black pixels). The resulting segmentation can be seen on the left. On the right is the quad tree encoding of the regions.

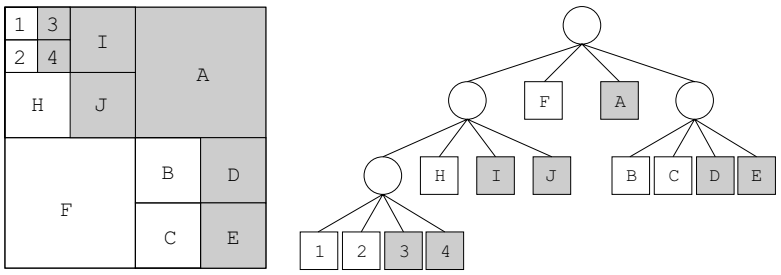


Figure 4.4: Quad tree representation of regions.

If there is strong intensity variations in the original image and a small threshold u is set, then the image will be over-segmented. Figure 4.5 [1] shows an example of a MATLAB implementation with a real image of 128×128 pixels and a dynamic range of $[0 - 255]$ in pixel intensity. As it can be seen, regions with high contrast are segmented into smaller regions. However, some regions of low contrast, like the grass, can be over segmented into different blocks.

Oct trees are the 3-D generalisation of quad trees [52, 80]. In this case, a cubic block is divided into eight sub-cubes, or octants, of equal volume. The algorithm to build oct trees is the same as quad trees. A homogeneity criterion is used to further subdivide the cube or to stop the algorithm. The resulting segmentation can be represented with a tree structure of degree 8 (referred to as an oct tree).

[1] Due to [16].

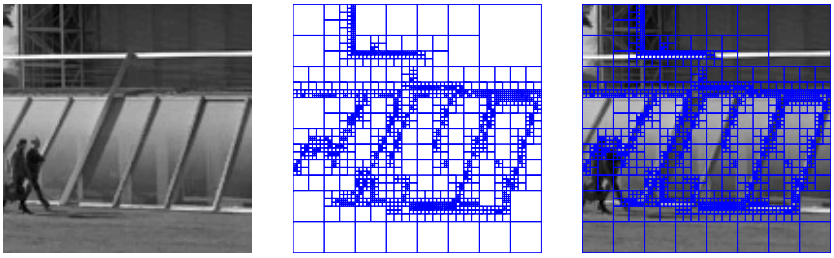


Figure 4.5: Quad tree decomposition of an image. On the left the original, middle figure the quad tree blocks resulting of applying the maximum intensity measure with a threshold $u = 15$. On the right, the superimposed blocks onto the original.

4.3 Morphological trees

In this section, three different types of *morphological* trees will be presented. Morphological trees treat the image as a group of connected regions, i.e. connected pixels with the same intensity (see chapter 2). As they work with only connected sets [95] they preserve all contour information of the image.

4.3.1 Critical lake trees

A *critical lake tree* is a simple tree introduced by Meyer et al. in [21]. The algorithm used to build the tree structure is the watershed segmentation introduced by Digabel et al. in 1977 [30].

Figure 4.6 shows an example of these trees. A watershed plus markers [13, 103] approach is used with minima as the seed. Starting from the minima markers, a *flooding* algorithm [31, 72] is then applied to completely segment the image. Every 'lake' or minima of the signal forms a node in the critical *lake* tree. After the whole topographic surface is flooded all path points through which two lakes merged are labelled by a given measure, such as *depth* of the lake, *area* or *volume* of the particle.

The resulting tree structure provides an useful representation of an image. Using the information stored in every *critical* node a multi-scale segmentation of the

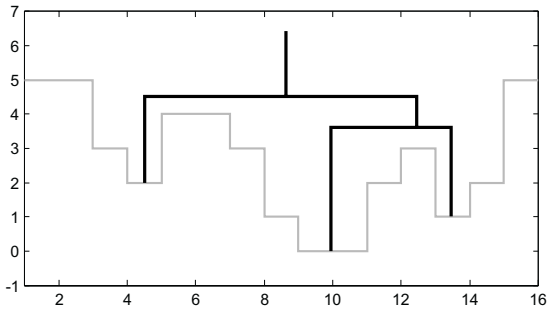


Figure 4.6: Critical lake tree example. The original function in light gray and the tree in black.

original image can be made [21].

4.3.2 Max and Min trees

Max trees are a relatively new representation of images. They appeared for the first time in literature in 1996 by Salembier and Oliveras as an extension of connected operators [87]. In a max tree structure, every node of the tree represents a connected set, or flat zone (section 2.3) in the original image.

The use of connected operators can be significantly simplified by constructing the max tree and computing the operator for each node in the tree [36]. Using the binary image of Figure 4.7, each component can be represented by a node inside a max tree structure. Firstly, all background pixels are assigned to the root node N_{root} (in this case white pixels). Secondly, three new nodes are created $\{N_{1_k}\}_{1\leq k\leq 3}$ representing the connected sets A, B and C of the binary image in the top row.

The removal of a node is accomplished by moving this specific node to its father in the max tree structure. The top row, right hand image of figure 4.7 shows the result of using a binary opening on the left hand image. The bottom row shows the same opening operator using the max tree. Note that the output max tree matches perfectly with the output binary image (top right image in figure 4.7).

The simple 1D signal, f, of figure 4.8 is used to illustrate the algorithm for building a max tree. A threshold h is used to binarise the input function f and

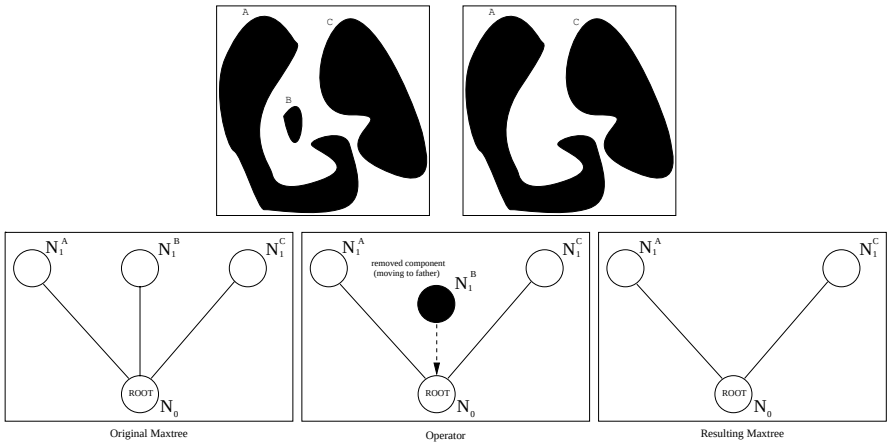


Figure 4.7: Example of a binary connected operator using a max tree.

to store the different connected sets found in the binary function as nodes of the tree. At every step, the threshold is increased by one and all nodes created in the previous step are re-analysed. Pixels with values lower than the threshold remain in that node, pixels with values greater or equal than the threshold h form new nodes (children of the node being processed).

In the 1D example, the threshold is set to 0 (the lowest value of the function) and it is used to quantise the signal. In this case, all points are greater or equal to the threshold h so all the points are assigned to one node, N_0, called the root. In the second step, the threshold is increased, $h = 1$ and the root node is analysed. Two connected sets are found, N_1^1 and N_1^2, and so these pixels are removed from node N_0 and assigned to two new nodes which are children of the root.

The second and last step, as the threshold reaches the greatest amplitude of the signal, finds two new connected sets and so nodes N_2^1 and N_2^2, children of nodes N_1^1 and N_1^2 respectively, are formed. While the algorithm creates new nodes some old nodes may become empty and, therefore, these nodes must be removed. For example, the empty node N_1^2 has to be upgraded to node N_2^2.

The extension of this algorithm to 2D is shown in Figure 4.9. The original image has 6 connected flat zones (A, B, C,D, E and F), the numbers represent the gray scale level of the connected set.

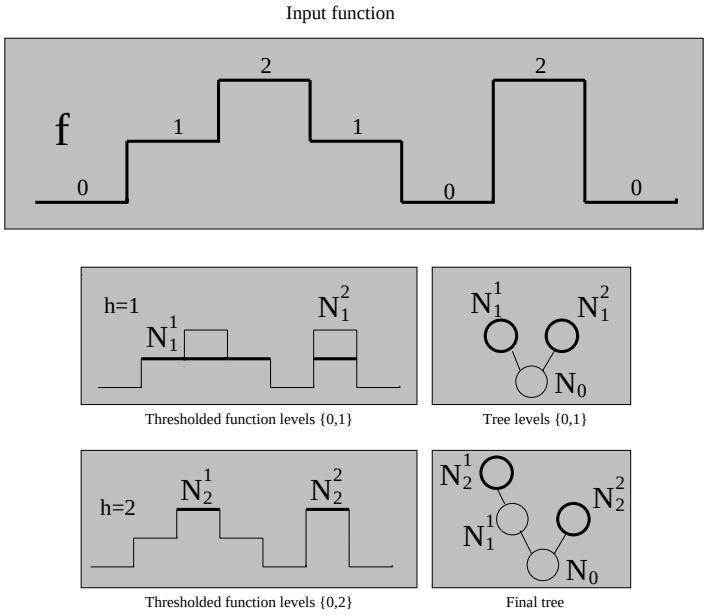


Figure 4.8: Example of max tree creation in $1D$.

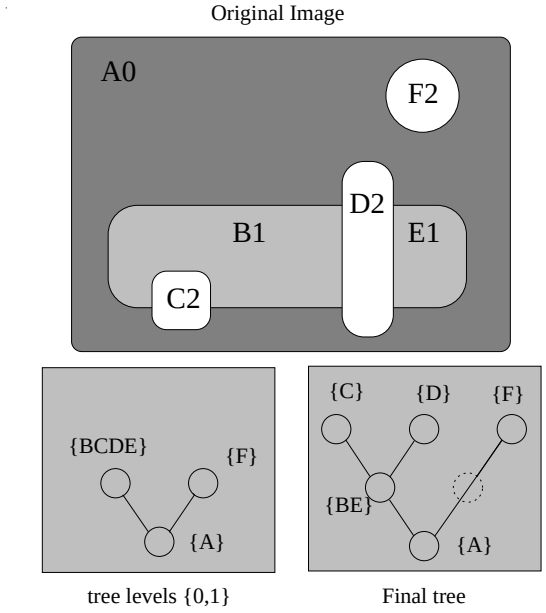


Figure 4.9: Max tree creation ($2D$).

The steps to build the $2D$ version of the max tree are the same as those used to make the $1D$ tree representation. The complete max tree construction can be

summarised as follows. Starting from the lowest value in the image, each node N_g^k (g stores the information about the gray scale level represented by that node) is processed. Within that node, pixels equal to the threshold h remain in the node and the different connected components of the remaining pixels (gray level value higher or equal than h) create the different children of that specific node.

Three important properties of the resulting max tree are listed below:

1. No contour information has been lost, the original image may be accurately reconstructed from the final tree. Even if one node is removed, the remaining nodes retain their contour information (figure 4.7).

2. The final tree is oriented towards the maximum of the image (*max tree*) so, the leaves of the tree are the maxima of the original function. To obtain the same representation orientated to the minima of the image (*min tree*) the original image has to be inverted.

3. The nodes in the final tree structure are not necessary formed by connected pixels, as can be observed of node $\{BE\}$ in Figure 4.9. This node represents the flat zones $B1$ and $E1$ which are two different connected components in the original image. Figure 4.10 shows a 'real' example where a node of the max tree (white pixels) is not formed by local connected pixels.

Once the max tree structure is built, and the different connected sets or flat zones of the image are assigned to all nodes, many criteria can be applied to each node to perform a variety of different tasks. For example a disparity criterion can be used for a stereo analysis [76], or a motion criterion for motion recognition [88, 89].

As max trees share many similarities with scale trees, section 6.2.2 will focus on the motion recognition application discussing the advantages and problems of both max and scale trees.

Figure 4.10: Example of a node in the max tree with non local connected pixels. White pixels represent one only node of the tree. Image taken from [67].

4.3.3 Binary partition trees

Binary partition trees are an extension to the max trees described above [84]. A binary partition tree is a shape oriented image representation in the sense that it is a structure representation of regions that can be obtained from an initial partition of an image.

There is no fixed algorithm to build binary partition trees, different solutions can be adopted depending on the application. In general, the binary partition tree retains information of the merging steps performed by the segmentation algorithm [37], this is called the *merging sequence*.

The tree is built using an initial partition of flat zones in the image, the algorithm merges neighbouring regions following the homogeneity criterion until a single region is obtained which is assigned to the root node. For example, a colour homogeneity criterion can be used to merge regions so every original flat zone is merged with the next 'closest colour' neighbouring region.

Binary partition trees can be used in a large number of processing goals such as detection or recognition, visual browsing, segmentation or information retrieval [86].

Chapter 5

Scale Trees

This chapter introduces the notion of a *scale tree*. First of all, a general graph-theoretical framework will be introduced. Some properties of scale trees will be presented. Finally, some examples will illustrate the building algorithm.

5.1 Building scale trees

A scale tree is a tree representation of the sieve algorithm. In chapter 3 was shown that the sieve algorithm removes extrema of any scale s from an input signal. If a cascade of increasing scale sieve filters is used, extrema of increasing size are removed. The complete sieving filter $\mathcal{CS}(f)$ (equation 3.4) can easily be transformed into a tree representation. Granules of different sizes are represented as nodes in the tree. Bigger granules are represented as parent nodes of small granules.

5.1.1 Graph notations

Before going into details of the proposed scale trees, some graph-theoretical notations and definitions need to be introduced [47]. Let $G = (V, E)$ be a graph as in section 3.1, where V is the set of nodes and E the set of edges. The *order* of G is defined as the number of nodes in V and its *size* as the number of edges in

E. Two nodes $n, m \in G$ are said to be *adjacent* ($n \sim m$) if they are connected by and edge.

A *path* in the graph can be defined as a sequence of nodes $n_0 n_1 n_2 ... n_p$ where $\forall i = 1...p, n_{i-1}\ n_i$. In other words, a path is a list of nodes that tells you how to go from node n_0 to n_p. In this case, the length of the path is p. A graph G is *connected* when there is always a path to join any pair of nodes. If $n_0 = n_p$ then the path is called a *cycle*.

Given the notion of a path, the *distance* between two nodes, $d(n, m)$, is the shortest path between those two nodes, with $d(n, m) = \infty$ if no path can be made (non connected graph). A connected graph with no cycles is called a *tree*, and a *rooted* tree is a tree with a central node, called the *root*. Two nodes in a tree, or rooted tree, are always connected by a unique path. The *level* of node n, $lev(n)$ in a tree is the length of the path connecting the root to n. So if $n \sim m$ and $lev(m) - lev(n) = +1$ then n is the *father* of node m, $\mathcal{F}(m) = n$ (equation 5.1) and, therefore, node m is the *child* of n.

$$\mathcal{F}(m) = n \quad \text{iff} \quad \begin{cases} n \sim m \\ lev(m) - lev(n) = +1 \end{cases} \tag{5.1}$$

5.1.2 Scale tree definitions

Scale trees are rooted trees, where every node N_{s_i} represents a granule of size s which has been merged into the next larger scale granule in one step of the sieve. After every sieve at scale s, the granule function g_s (equation 5.2) describes the removed granules, which form nodes in the scale tree. The granule function can consist of more than one granule (non zero regions in g_s) so each every granule function may generate more than one node in the scale tree (equation 5.3). The node N_{s_i} represents the granule i at the scale s.

$$g_s = \mathcal{S}_{s-1}(f) - \mathcal{S}_s(f) \tag{5.2}$$

$$N_{s_i} = g_{s_i} \quad \text{where} \quad \bigcup g_{s_i} = g_s \tag{5.3}$$

The algorithm used to build a scale tree is as follows. At every step of the complete sieve operator, the granule function g_s is computed and every granule g_{s_i} is labelled by its corresponding node N_{s_i}. As a new node is created, a link is established between previous nodes which have been already merged and that specific node N_{s_i}. This link is defined as a father relation ($\mathcal{F}$) in the scale tree structure (equation 5.4).

$$\mathcal{F}(N_{s'_i}) = N_{s''_j} \quad \text{iff} \quad \begin{cases} s' < s'' \\ g_{s'_i} \subset g_{s''_j} \\ \forall s,\ min|g_{s'_i} - g_{s_i}| = |g_{s'_i} - g_{s''_j}| \end{cases} \tag{5.4}$$

Equation 5.4 shows the father relation between two different granules at scale s' and at the larger scale s''. Node $N_{s''_j}$ is the father of $N_{s'_i}$ if and only if the three conditions stated above are fulfilled[1].

As the sieve can be defined in 1D or 2D, scale trees representations can be derived for 1D or 2D signals. In this thesis, the 2D sieve is been using extensively, but, for clarity, section 5.1.3 explains the tree building algorithm with a 1D example. Note that, in any case, the tree representation of a function is the same in 1D, 2D or in any dimension as it is using a scale decomposition (sieve) of the original image. Therefore, *scale* is used generally for any dimension but specifically refers to *length* in the 1D case, *area* in 2D case or *volume* in 3D.

[1]The last condition in (5.4) is intended to eliminate grandfathers, great grandfathers and so on.

5.1.3 1D scale trees

As a simple example, the function shown in figure 5.1, will be used to build, in this case, a 1D scale tree.

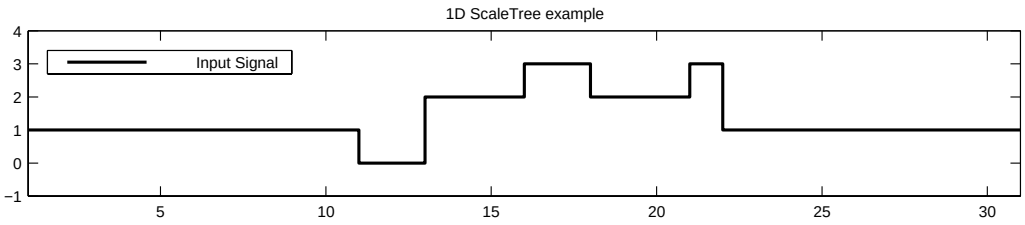


Figure 5.1: Input signal f for 1D scale tree example.

In the first step (figure 5.2), a scale 1 sieve operator $\mathcal{S}_1(f)$ is applied over the input signal f. The granule function g_s is computed, in this case g_1 as we are sieving at scale 1, and the node N_{1_1} is created labelling the corresponding granule. Note that there is only one granule in g_1, and so only one node N_{1_1} is created. The graph on the left of figure 5.2 shows the granule function (with the granule g_{1_1} in red) and the resulting sieve function at scale 1 $\mathcal{S}_1(f)$ which will be used as input function in the next scale sieve is shown in blue.

The right side of figure 5.2 shows the new node built (in red) named N_{1_1} as it represents the first, and only, granule at scale 1. The ellipse represents the remaining points in the input signal f which have not yet been assigned to any node of the scale tree.

Note that the link between node N_{1_1} and the ellipse is not a father relation. It is just a temporary link between the new node and the granule it merges to.

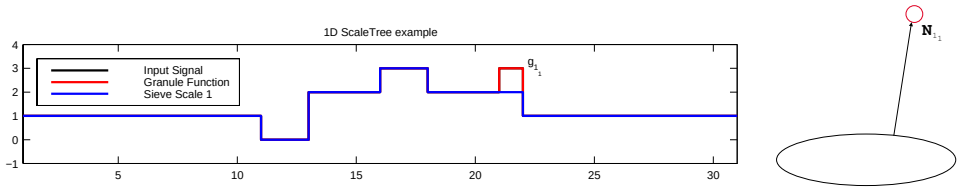


Figure 5.2: Sieve up to scale 1, granule g_{1_1} labelled in node N_{1_1} (step 1).

In step 2, the sieve is applied to the new input signal, $\mathcal{S}_1(f)$. Now at scale 2. This time, two different extrema are found, g_{2_1} and g_{2_2}, in the granule function g_2. Following the algorithm these two granules are removed and labelled via two new nodes of the tree, N_{2_1} and N_{2_2}. The tree representation shows the two new nodes in red. A temporary link is made to keep information about where the two granules are going to be merged to.

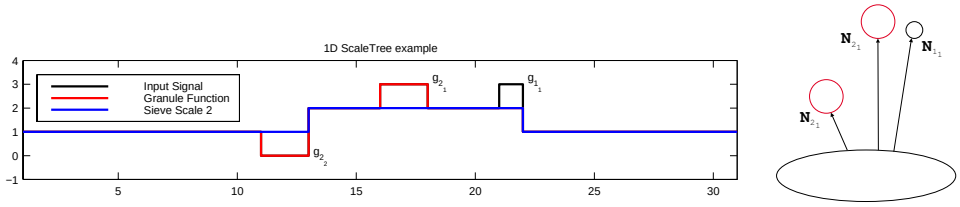


Figure 5.3: Sieve up to scale 2, granules g_{2_1} and g_{2_2} removed and stored in nodes N_{2_1} and N_{2_2} respectively (step 2).

Finally, a sieve to the next scale is applied. In this case, the next scale is 9 which removes the big step g_{9_1}. Once the granule is removed, a node is created in the tree N_{9_1}. At this point, the tree building algorithm links the two already created nodes (N_{1_1},N_{2_1}) to the current N_{9_1} node, since granules g_{1_1} and g_{2_1} were attached to granule g_{9_1} in a previous step of the complete sieve (step 3). The node N_{9_1} is now linked to nodes N_{1_1} and N_{2_1} with a father child relation (equation 5.4). In other words, nodes N_{1_1} and N_{2_1} become siblings and children of node N_{9_1}.

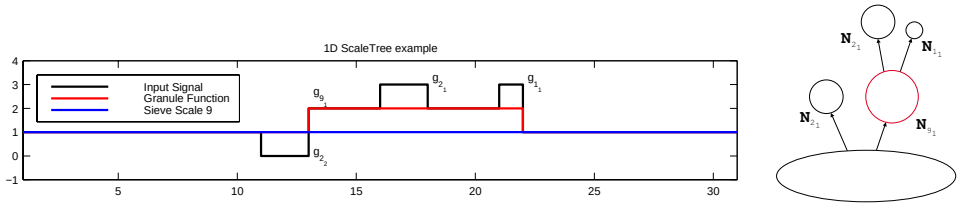


Figure 5.4: Sieve up to scale 9, last extremum g_{9_1} removed and stored in node N_{9_1} of the scale tree (step 3).

The output of the sieve operator at this stage, $\mathcal{S}_9(f)$, is a constant signal and so no new extrema can be found. Scale 9 was the maximum granule size (m) in the input signal f and so the complete sieve operator stops. However, to finish

building the tree, a final step needs to be completed. All remained points are assigned to the root node N_{root} and all nodes without father at this stage, are linked to this root node. The figure 5.5 shows the final tree. Five different nodes result from the sieve process, each of which corresponds to a granule of the input function f.

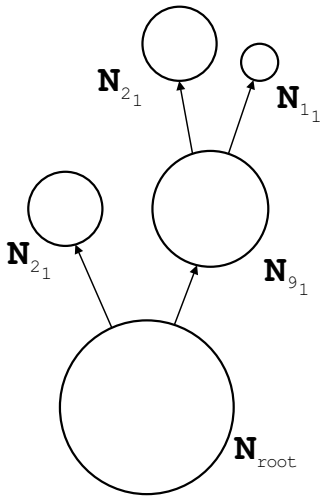


Figure 5.5: Final scale tree of signal f in figure 5.1.

5.1.4 2D scale trees

We now revise an example of building a 2D scale tree of an image I. As section 3.1 described, the sieve algorithm can be defined in any number of dimensions. Now, the 2D sieve algorithm will be used to build the scale tree representation of an image.

The steps needed to build a scale tree representation of an image I are the same as those used to build the scale tree of a 1D function (section 5.1.3). As a simple example, the image shown in figure 5.6 is used to create a 2D scale tree. The image is of size 20x20 pixels with 5 different regions; the grey rectangle at the bottom, partially occluded with the three white rectangles, and the black rectangle at the top of the image.

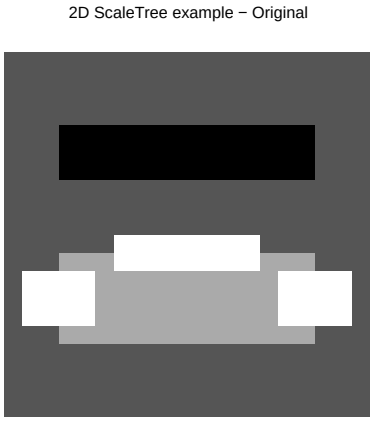


Figure 5.6: Input image I for the 2D scale tree example.

The only difference between the algorithm used to build a 2D scale tree and that used for the 1D is the definition of scale. In 2D scale is a measure of *area* instead of length. The first step involves sieving the input image I to increasing scales and halting the algorithm when the granule function g_s contains non zero elements. For this example, that scale is area 12 pixels. Two extrema are found with this sieve operator (the two white squares at the sides of the grey rectangle).

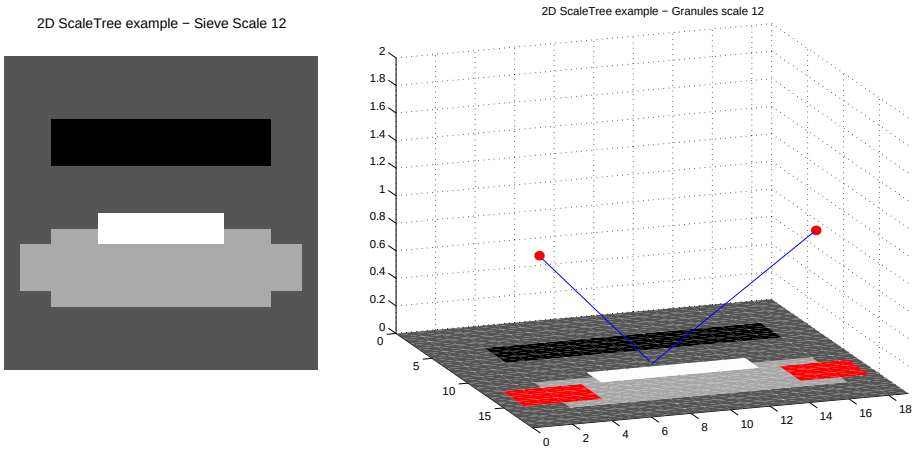


Figure 5.7: Sieve up to scale 12 pixels, two granules removed and stored in nodes of the scale tree (step 1).

Figure 5.7 shows, on the left, the resulting sieved image $\mathcal{S}_{12}(I)$ where the two white squares have been merged to the grey rectangle. The two new nodes created can be seen on the right (red dots). At the base of that figure, the original image is represented with the granules found in this step coloured in red.

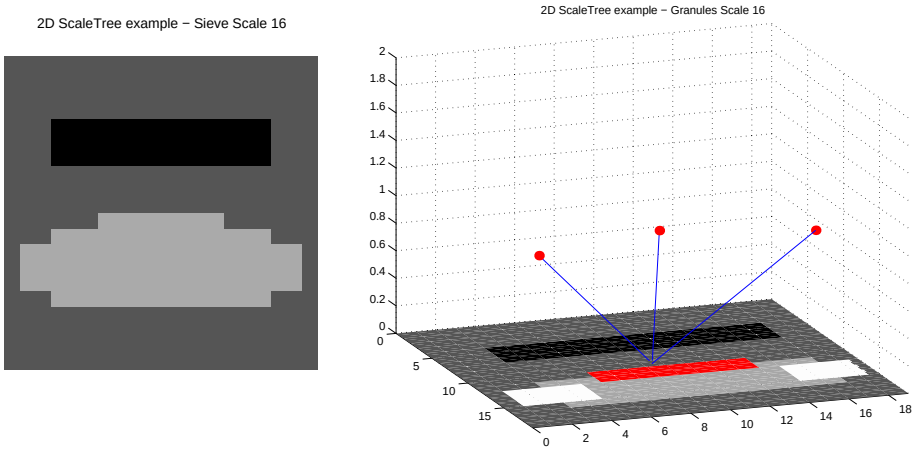


Figure 5.8: Sieve up to 16 pixels, granule g_{16} stored in the scale tree (step 2).

The x and y coordinates of the two new nodes N_{12_1} and N_{12_2} are positioned at the centroid [14] of the respective granule. The z coordinate is the level or height of the tree (root node is level 0, its children 1, its grandchildren 2 ... etc). The blue link to the middle of the figure represents the same undetermined nodes that were given by the ellipse in the 1D example. The link keeps information about the grey region the white squares have been merged to, so when that granule is removed by the complete sieve a father link can be established between them.

The next important scale is 16 pixels, the area of the remaining white. In this step, 2, another node is created N_{16_1} to label the new granule. The sieved result and the temporary scale tree is shown in figure 5.8.

In step 3, at scale 42, The black rectangle at the top of the image is found. The sieve output (left of figure 5.9) shows how the granule is removed. Again, the right side shows the new node in the scale tree (N_{42_1}).

In the last step, 4, the output of $\mathcal{S}_{90}(I)$ is zero, so no more extrema can be

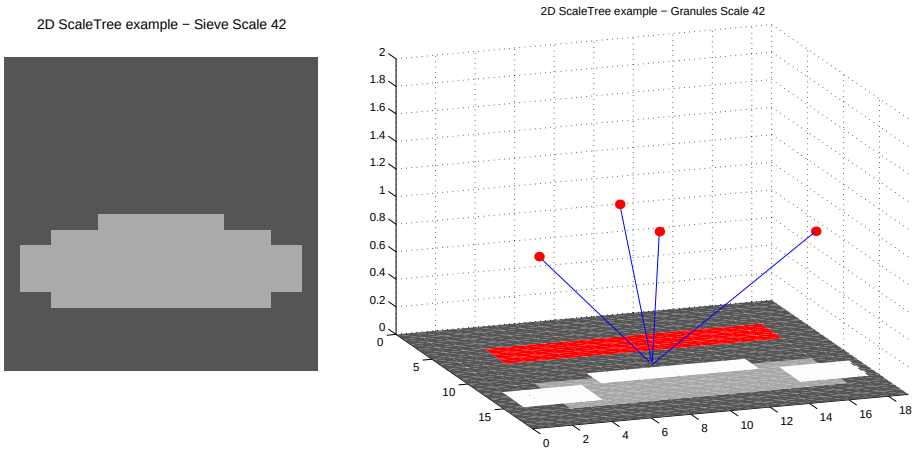


Figure 5.9: (Step 3) Sieve up to scale 42.Left shows the sieve output, $\mathcal{S}_{42}(I)$. On the right, the temporary scale tree.

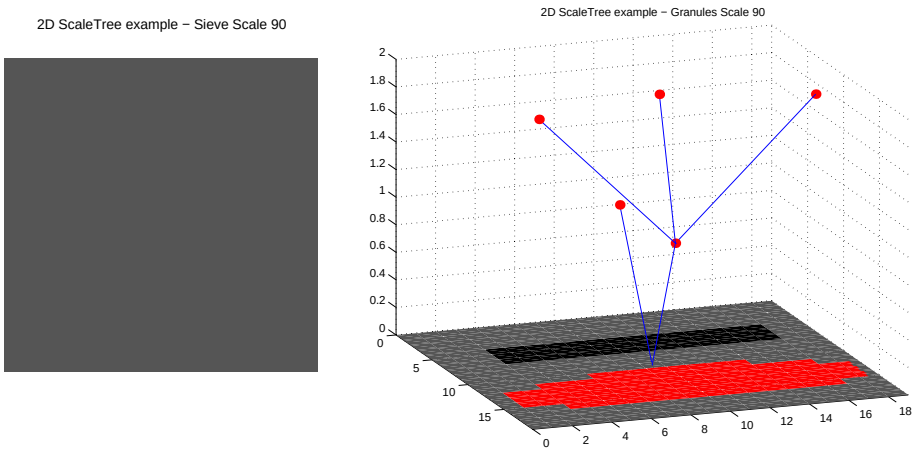


Figure 5.10: Step 4 of the 2D scale tree building algorithm.

found and the complete sieve finishes. The granule function g_{90} contains the last remaining granule. The right-hand of figure 5.9 shows the corresponding node created N_{90_1}. Note that this implies father links between node N_{90_1} and nodes N_{12_1}, N_{12_2} and N_{16_1} as these regions were merged to granule g_{90} in steps 1 and 2.

In the final step, all remaining pixels are associated with the root node of the scale tree. Nodes N_{42_1} and N_{90_1} are linked to N_{root} following the equation 5.4. Figure 5.11 shows the final tree with the original image at the bottom. Six nodes are created to completely represent the image in figure 5.6.

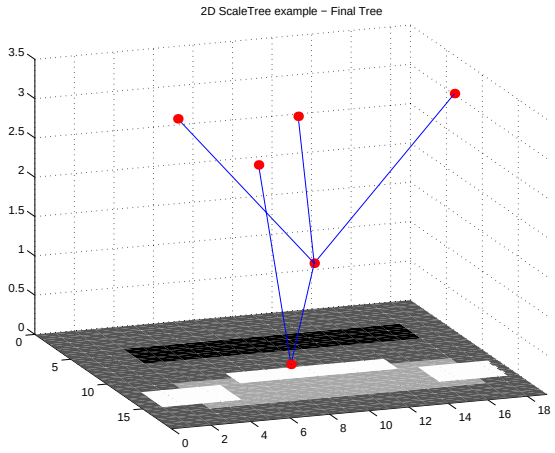


Figure 5.11: Final scale tree of input image in figure 5.6.

Again, the x and y coordinates of every node are the centroid of the granule. The z coordinate represents the level or height in the scale tree. Note that node N_{90_1} has three children. Generally, scale tree nodes can have any number of children. However, the internal representation chosen to store the scale tree data was a binary tree [60, 105], for details about the internal representation of scale trees see appendix A.

Until now, scale trees have been built out of images or functions. However, given that a scale tree is a complete representation of the original image, it is possible to recreate the original image from the scale tree without losing any information.

The algorithm to reconstruct the original image from the scale tree can be explained in two slightly different ways. The first implies visiting all nodes, adding the value of the granule function at that point g_{s_i}, or simply, adding all granule function g_s for every scale (equation 5.5).

$$I = \sum_s g_s = \sum_s \mathcal{S}_{s-1}(f) - \mathcal{S}_s(f) \tag{5.5}$$

The second method is a simple variation of the first. In the scale tree, instead of

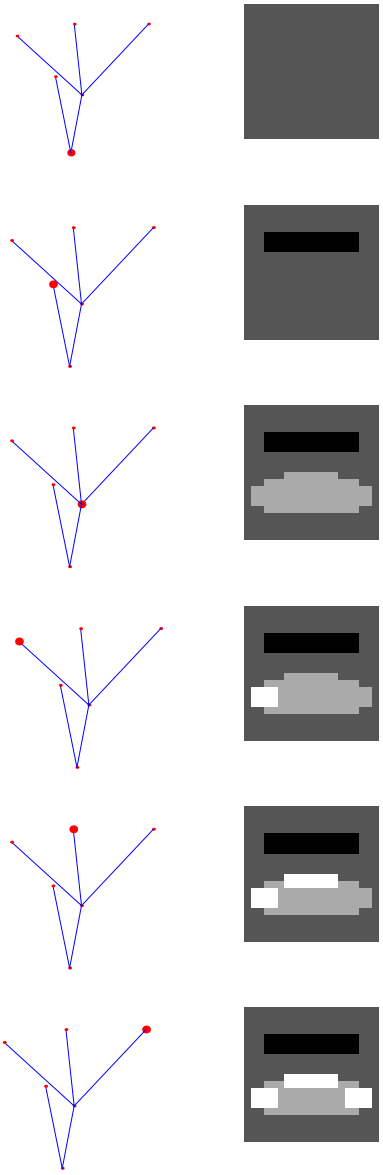


Figure 5.12: Traversing the tree in preorder and image obtained at the left.

storing the difference between two sieve functions, g_s, the actual grey scale pixel value in the image is stored. Reconstructing the original image I implies visiting the tree from the root to the leaves *replacing* the grey scale values of every node visited.

This method solves the 'padding' problem as replacing the values preserve the DC component of the original image. As going from the root to the leaves (greater to less scale) is the same as visiting the tree in preorder [60], we can easily get back to the original image doing a preorder traversal of the scale tree and replacing all grey scale values of all nodes N_{g_i} into the original image.

The preorder traversal algorithm is defined as follows:

ALGORITHM 1 *Preorder traversal algorithm:*

1. *Visit the root*
2. *Traverse the next-sibling sub-tree*
3. *Traverse the next-child sub-tree*

The scale tree of the last example, figure 5.11, can be now used to reconstruct the original image I. As figure 5.12 shows, visiting the scale tree in preorder involves visiting the root node first N_{root} (which produces the image on the right). The next step of the algorithm visits the next brother of the root node, which doesn't exist so the following child is traversed. In this case, node N_{42_1} and its granule is replaced in the image. Following the algorithm, all nodes are visited and the final image is obtained (bottom right panel of figure 5.12).

Until now, very simple examples have been presented to allow us to follow the scale tree algorithm. Real images lead to very complicated trees with an order of thousands of nodes, depending on the size and contents of the image. Appendix B shows some experimental results of the performance of building scale trees using different sets of images.

Figure 5.13 shows an example with a computer generated table. On the top row, two different views of a table are shown. On the bottom row, the corresponding scale trees of the originals are shown. With simple images like these, the trees are also simple, with 110 and 75 nodes respectively.

This example also shows one useful feature of scale trees. As we have seen, scale trees are an abstraction of the original image, so the tree structure itself may be fairly insensitive to geometrical changes in the image, such as rotation, translation, or some changes in the view point.

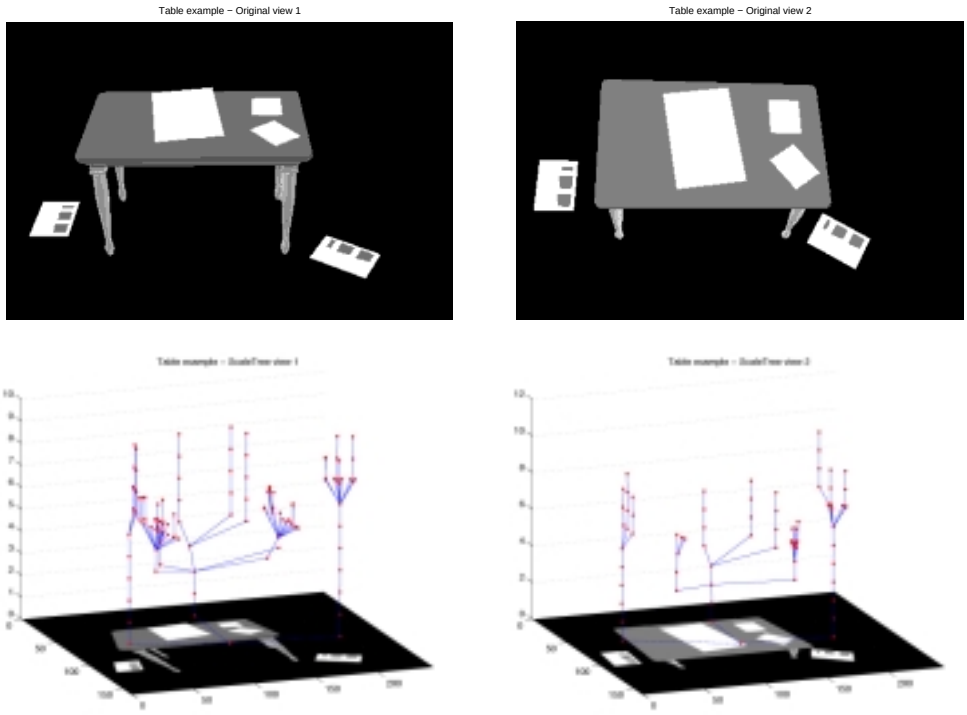


Figure 5.13: Two different views of a computer generated table and its respective scale trees.

The two scale trees of figure 5.13 look very similar, both having three main branches corresponding to the two sheets of paper and the table. Of course, some differences are present in the trees, like the nodes corresponding to the table legs. This property can be beneficial for pattern or object recognition tasks as the tree is relatively insensitive to geometric transformations of the original signal.

To conclude this section, the scale tree algorithm is tested on real images. Figure 5.14 shows the image of a popular doll. On the right, the scale tree with 1509 nodes. The nodes at the bottom of the tree correspond to noise in the background and can be removed easily applying a normal sieve at scale of 4 or 5 pixels (depending of the actual size of the image) before creating the scale tree. Columns of the tree with many nodes correspond to shading effects on flat objects in the original image.

As a final example, figure 5.15 will be used to illustrate some of the problems

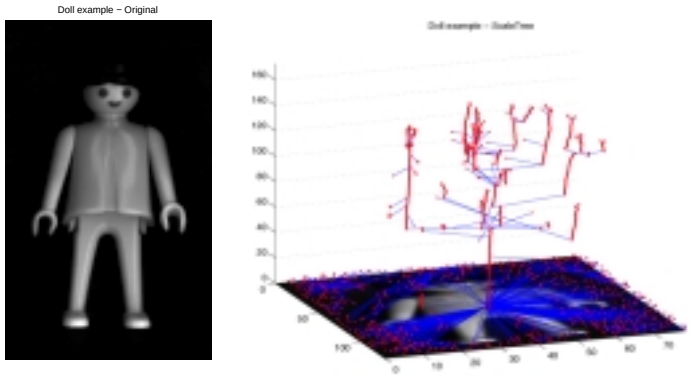


Figure 5.14: The scale tree of a real image.

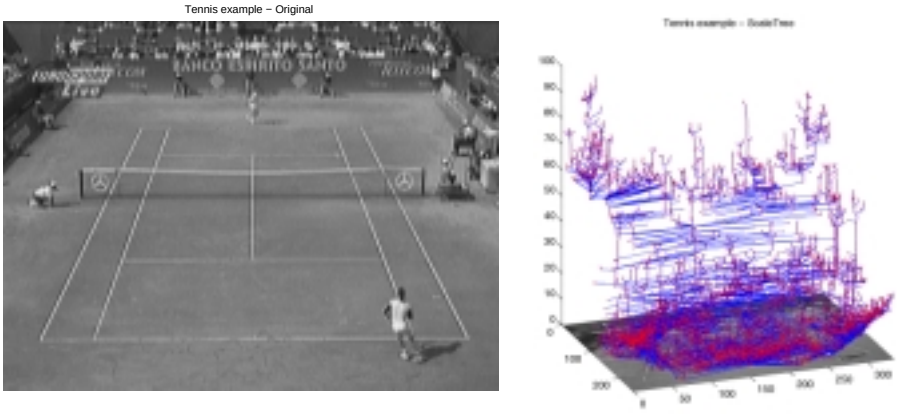


Figure 5.15: A scale tree of a tennis match.

of complicated images. As images increase in size or complexity, the number of nodes in the scale trees become too large. Different techniques that reduce the complexity of scale trees are used and studied in this thesis as it is obvious from figure 5.15 that 'real' trees may be too complicated for some computationally intensive operations. Operations for pruning, trimming, and collapsing branches will be introduced later in this thesis (section 6.1.1).

The algorithm to build a scale tree is straight forward. As a brief description, the complete sieve, $\mathcal{CS}(f)$, is applied to the original function until no new extrema are found. All granules are stored as nodes of the scale tree and father links are established between them as defined in equation 5.4.

5.2 Scale tree properties

This section will discuss some interesting properties of the scale trees. Following subsections will list some of the most important properties.

5.2.1 Scale tree advantages

First of all, notice that scale trees share all the properties and features of 'normal' trees [60]. Trees are a standard data structure, well known and studied in literature [32, 50, 105]. As well as having these characteristics, scale trees have some properties of their own that make them very attractive to some image processing operations. Some of these properties are listed below:

1. Any image I has a *unique* corresponding scale tree T.

2. Scale trees are invertible. We can go from the original image to the final tree and vice-versa without losing any content information.

3. The complete sieve operators, and thus scale trees, are based on connected operators derived from mathematical morphology (section 2.3). As connected operators interact with flat zones of the image (regions where the image has constant intensity) and only merge these flat zones into other ones, they can not introduce any extrema or any new contour on the image. Even if nodes are removed from the scale tree, the remaining nodes retain all their contour information.

4. Scale trees perform an initial segmentation of the image into flat zones [25, 27, 91]. Together with this segmentation, the relation between these flat zones is found using the complete sieve algorithm.

5. The final tree is oriented towards the minimum scale of the image. Leaves of the tree represent smaller granules than nodes closer to the root. So, in that sense, the tree represents a set of regions at different scales of resolution.

6. Scale trees are a scale structure with information about containment (stored in father links). The tree establishes a relation of inclusion (or adjacency in some cases) between all regions (flat zones) of the original image.

7. Scale trees are invariant to some geometric transformations of the input signal, such as rotations, translations or any distortions which preserve the topology of the original image.

In the following sections, some examples of these properties will be analysed.

5.2.2 Seeding with extrema

As we have seen, scale trees are a tree representation of the complete sieve algorithm, $\mathcal{CS}$. Thus, they are oriented towards scale. They also imply a relation of inclusion between nodes and so, as the tree is traversed from lower nodes to the leaves, all visited nodes are included in their parents nodes.

It is important to note that the sieve algorithm (section 3.1) is seeded with extrema. There may be flat zones in the original image that are smaller than some of the scale tree's leaves nodes but, because they are not extrema, they do not appear as leaves of the tree. The next example will show this property.

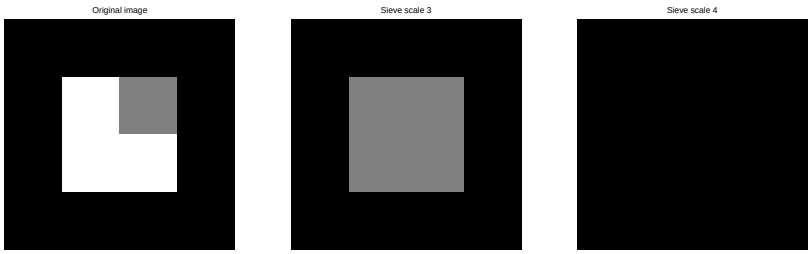


Figure 5.16: Complete sieve operator applied to the image of size 4×4 pixels on the left, $\mathcal{CS}(I)$. First sieve (up to 3 pixels) removes the white region and merges it to the grey pixel (middle image). Last sieve, $\mathcal{S}_4(I)$, merges that grey granule with the black one (on the right).

Figure 5.16 shows the complete sieve sequence applied to the original image on the left. There are three different connected sets in the image, corresponding to

the black, grey and white regions. As a first approach, one should build the scale tree starting with the grey pixel as one leaf of the tree, because it is the smallest granule of the image. However, because the grey pixel is not a extremum, it is not used to start the algorithm.

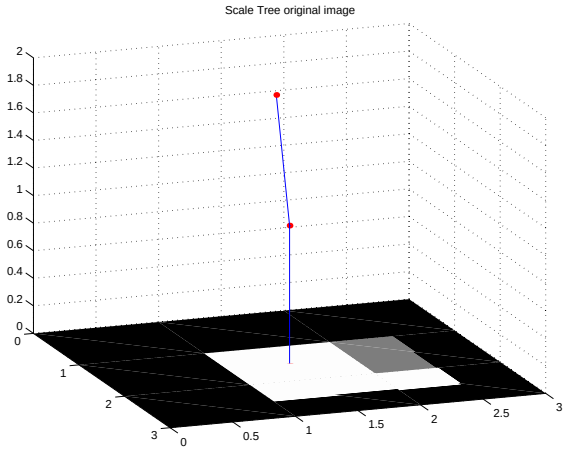
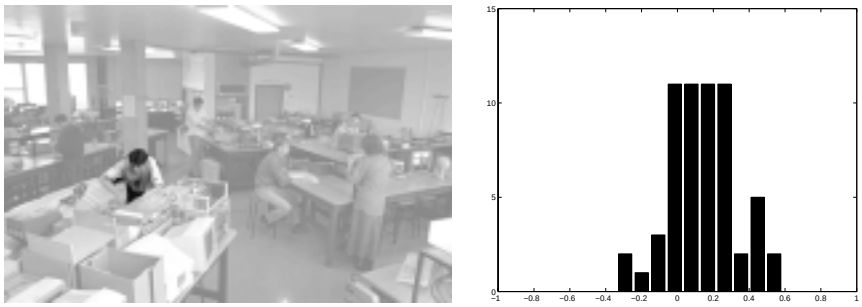


Figure 5.17: Final scale tree of figure 5.16.

Using the complete sieve to build the scale trees implies seeding with extrema. In this example, two extrema are found. The white pixels and the black around them. The sieve algorithm merges them to the next closest (in amplitude value) flat zone. Obviously, in this example, the smaller region is the white one, and it merges to the grey granule. In the next step, the grey granule becomes extrema and it can be merged then with the black region. The final scale tree looks like figure 5.17. It has three different nodes, the leaf node corresponding to the three white pixels.

Is it beneficial then to seed in extrema? As we have seen in the last example, seeding in extrema can sometimes lead to strange partitions of the original image. In fact, seeding with extrema can have two main advantages. First of all, it gives a preliminary clue about where to start merging granules. Looking at figure 5.16 again, if no seed is given, one can not be sure if the grey region (smallest) should be merged with the white pixels or the blacks. Seeding with extrema gives a good starting point to merge granules.

Another advantage of this kind of seeding is that the final scale tree is oriented towards extrema as well as scale and extrema may be associated with objects. To test this, it is appropriate to obtain evidence that clusters seeded from extrema are associated with objects in a different variety of images. Some studies show that real objects in real images tend to be extrema in respect with the background [11].



A) *B)*

Figure 5.18: *A)* a photograph with a region that has been segmented manually (highlighted for the illustration). *B)* a histogram that represents the proportion of the manually selected object that is represented by scale-tree branches. The abscissa, centred on zero, is a difference of two ratios where positive values reflect objects that form branches. The ordinate is the number of observations.

Volunteers were asked to draw around, what they chose to identify as, objects. Figure 5.18*A)* shows an example. As extrema are localised as leaves of the tree, if this selected region includes regional extrema it will be associated with whole branches of the scale tree. One way to quantify this is to find the fraction of area (scale) that is locally more extreme than the region outside, $r_{obj} = \max_{s_{in}} \frac{s_{in}}{s_{all}}$, where s_{all} is the scale (area) of the selected region identified as an object manually and s_{in} is the area included in a branch of the scale tree.

This relation shows the ratio of the maximum area which is represented as a whole branch in the scale tree. If the region is entirely represented by complete branches, then the fraction r_{obj} would be 1. The fraction is then compared with a control segment obtained by randomly translating the region shape to another position in the image and again finding the ratio, r_{rand}.

Figure 5.18*B)* shows the distribution of $r = r_{obj} - r_{rand}$ obtained from 60 ob-

jects selected by 5 people from 6 images. The majority of differences are positive, showing that the manually segmented objects are more often associated with extrema than random segments. This supports the view that scale-trees, obtained from sieves (section 3.1), are likely to be useful for representing objects in a wide variety of images.

5.2.3 Invariant representation

It was shown in section 5.2.1, that scale trees are invariant to some geometrical transformations of the original image. As the scale tree is built using extrema and scale information, some simple rotations, translations or distortions which preserve the topology are unlikely to produce any substantial change to the scale tree.

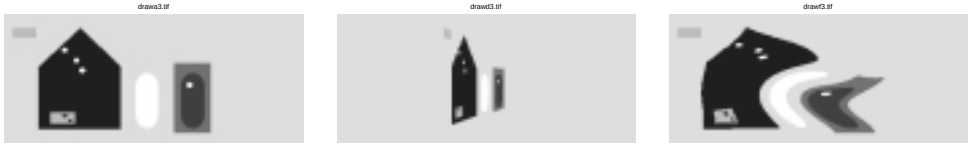


Figure 5.19: A stylised grey scale image (left) and, to the right, two distorted versions. Each image has size 383 by 165 pixels.

Such a scale tree represents a considerable abstraction of the original image. This is illustrated in the figures 5.19 and 5.20. In each case distortion of the original image causes the x and y co-ordinates of the image to change but the tree topology is invariant.

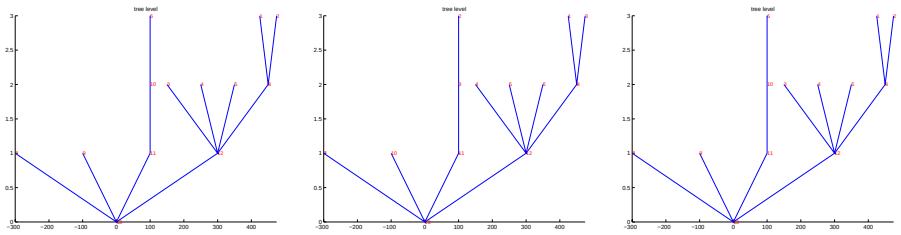


Figure 5.20: Scale trees for the images in Figure 5.19. The y-axis represents tree depth.

In this case, the scale tree has been printed in 2D where now the y coordinate is the height or tree level and x is the distance along the previous axis $\alpha x + \beta y$ where α and β have been chosen to avoid tree branches occluding. Figure 5.20 shows the scale tree representation of the three images in figure 5.19. As the image topology is preserved in all transformations, the scale trees remain the same in all three cases and even "strong" transformations in the original image do not change the tree structure. This property is the key for some simple recognition tasks invariant to rotations, zooms or small movements in the camera angle.

Chapter 6

Applications

Previous chapters have introduced the notion of a *scale tree*, a new representation of an image based on the sieve algorithm (chapter 3). In a first approximation, a scale tree can be seen as an attempt to define an *object tree*, a structure where every node of the tree represents a meaningful object of the scene. This chapter will discuss some applications of the scale trees together with some ideas and algorithms to refine the scale tree of an image in order to 'transform' it into a real object tree.

To do so, this chapter is divided into two different parts. The first section will overview some 'lossy' algorithms to reduce the information of scale trees. The next sections will introduce some different techniques to convert scale trees into object trees.

6.1 Parsing scale trees

One of the most difficult goals in image processing and computer vision is to define a scene by terms of meaningful objects. For humans, all the information of an image can be decomposed in, maybe, four or five objects of interest. When we recall a photograph, a simple descriptor such as 'a photograph of my son on the beach' is often sufficient. Just two descriptors are used to identify the image, *son*

and *beach*. Reducing the quantity of information is the key.

These ideas of simplifying an image in terms of meaningful objects are currently being adopted by 'new' standards in image processing and computer vision, such as MPEG-4 with its visual object planes or VOP's. Furthermore, the upcoming standard MPEG-7 tries to adopt the 'photograph' concept described before by using two or three significant descriptors of the image for a later indexing [40–42].

The following section describes some algorithms used in this thesis to reduce the complexity of scale trees.

6.1.1 Simpler trees

As described in section 5.1.4, scale trees of real images may be highly complicated (see figure 5.15). The tree structure itself needs to be simplified. In order to be able to reduce the complexity of scale trees, two ways of decreasing the number of nodes have be considered: collapsing long unbranched chains and pruning low contrast children [10].

Collapsing scale trees

The conventional approach [7] to simplify an image using the sieve algorithm has been to describe the image via *channels* which divide the granularity functions into different bins. A channel is a sum of the granules within a fixed range of scales. Figure 6.1 shows an example of a channel decomposition of an image. Since particular features exhibit some area variation, due to shading of objects or blur caused by imperfections in the imaging system, different objects may exist in more than one channel. In the example, the windmill is decomposed into 2 different channels (16 and 17).

A solution to this problem is to track the object through scale and look for a peak in the scale selection surface [66]. In terms of the scale tree representation, this tracking means visiting every node going from the leaves to the root. The

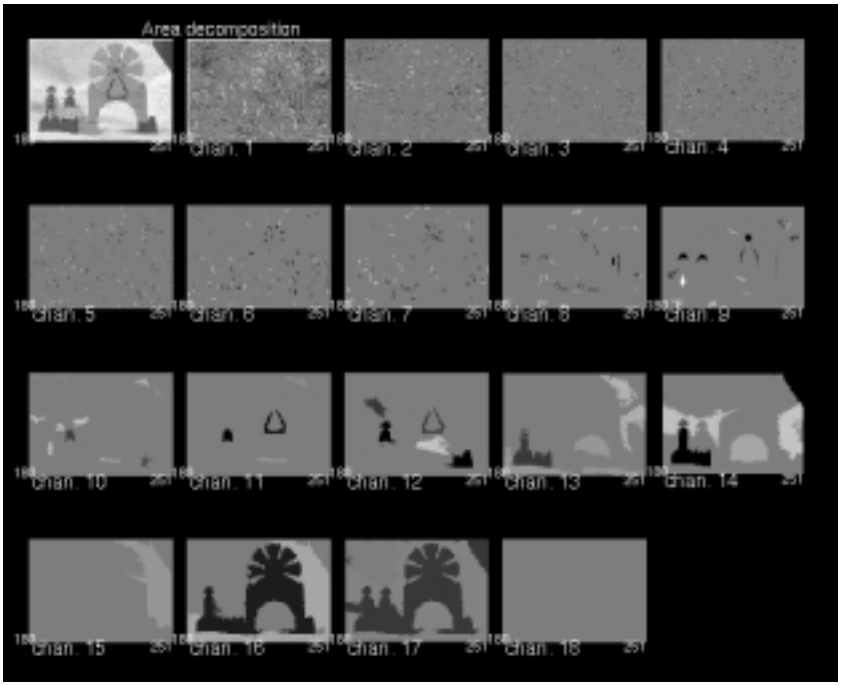


Figure 6.1: An example of a channel decomposition. The windmill on the figure is represented in 2 different channels.

sequence of nodes from leaf N to the root can be defined as,

$$S(N) = [N, \mathcal{F}(N), \mathcal{F}(\mathcal{F}(N), \quad \dots \quad , \mathcal{F}(\dots \mathcal{F}(N) \dots), N_{root}] \tag{6.1}$$

Using, for instance, the node N_{2_1} of figure 6.2 the sequence associated with it would be,

$$S(N_{2_1}) = [N_{2_1}, N_{9_1}, N_{31_1}] \tag{6.2}$$

For every node N of the scale tree, a possible measure of the scale selection surface can be defined as the relation between node N and its father $\mathcal{F}(N)$ of their difference in amplitude value over their difference in scale.

$$\Delta(N) = \frac{|g(\mathcal{F}(N)) - g(N)|}{s(\mathcal{F}(N)) - s(N)} \tag{6.3}$$

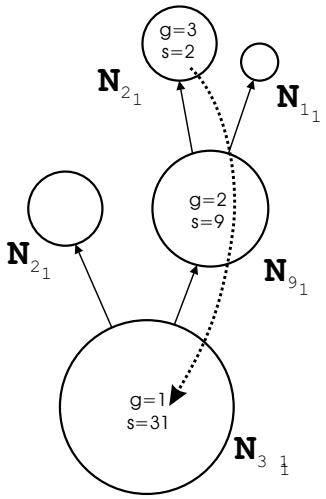


Figure 6.2: Scale selection sequence.

where $g(N)$ is the grey scale value of the connected set represented by node N and $s(N)$ is the scale, or area, of the node.

This scale selection measure can be applied to every sequence of the tree defined earlier, so, for every branch of the scale tree, a *scale selection sequence* can be computed as,

$$SSS(N) = [\Delta(N), \Delta(\mathcal{F}(N)), \Delta(\mathcal{F}(\mathcal{F}(N))), \ldots, \Delta(\mathcal{F}(\ldots \mathcal{F}(N) \ldots))] \quad (6.4)$$

note that, in this case, the scale selection function cannot be applied to the root as its father node its not defined.

The peak in the function $SSS(N)$ is, for an object, the node at which its rate of change of intensity with respect to scale is maximised. For example, a perfect disc of area s would yield a sequence of $SSS(N)$ that is all zero except for one value at its true scale.

The *collapsing* algorithm, $\kappa(T)$, works by computing the maximum value of the scale selection sequence of every branch of the scale tree and by removing all nodes of that branch to the node holding the peak in the sequence. In terms of the image, removing nodes in the scale tree structure implies merging connected sets. The effect on the scale tree representation is that long, unbranched chains of nodes

are *collapsed* into a single node. As has been seen, long chains usually correspond to shading or blurring effects, therefore, the collapsing algorithm simplifies the image by *sharpening* the original.

Figure 6.3 shows the collapsing algorithm on the scale tree of the computer generated table, T_o, shown in figure 5.13. The original image is slightly blurred due to dithering during the rendering process and so, some chains of unbranched nodes can be observed in the scale tree (like the sheet of paper on the very left). The resulting collapsed tree, $\kappa(T_o)$, on the left hand side of figure 6.3, is a simpler version of the original, where unbranched chains with more than one node have been collapsed into only one (the maximum in the scale sequence). The right hand side of figure 6.3 shows the image associated with the collapsed scale tree.

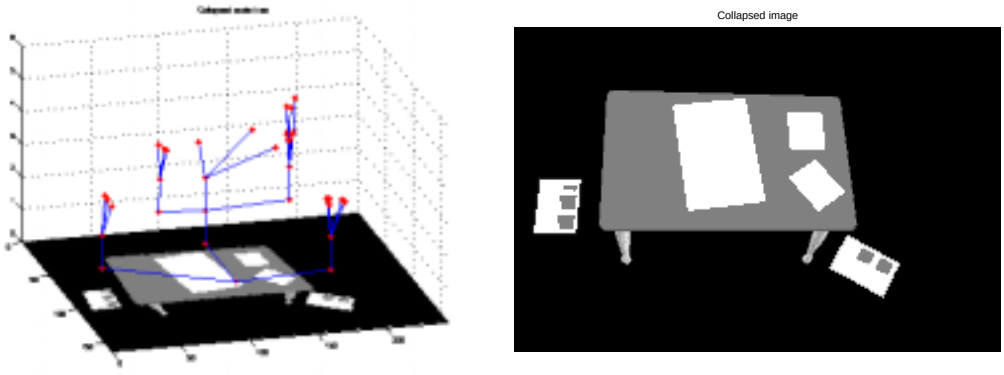


Figure 6.3: An example of the collapsing algorithm κ. On the left, the *collapsed* scale tree of figure 5.13 (note the simpler branches in the tree). On the right hand side, the resulting image from the collapsed tree.

In order to have a better understanding on how the algorithm works, figures 6.4 and 6.5 show another example. Figure 6.4 shows a Gaussian filtered version of the original computer generated table. The filter smooths the image leaving the overall structure of the original image intact. As a result, the scale tree of the Gaussian filtered version, T_g, looks similar to the original except for the long chains of nodes due to the smoothing effect of the filter.

Figure 6.5 shows the result of applying the same collapsing algorithm in the previous filtered image. The resulting tree, $\kappa(T_g)$, is simplified again. The image

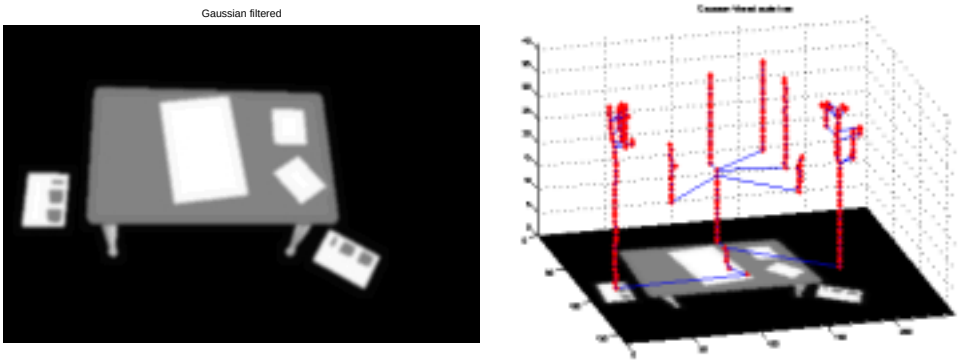


Figure 6.4: On the left, the original image after a gaussian smoothing. On the right, the corresponding scale tree T_g.

associated with the tree is sharpened and is similar to the collapsed version of the original.

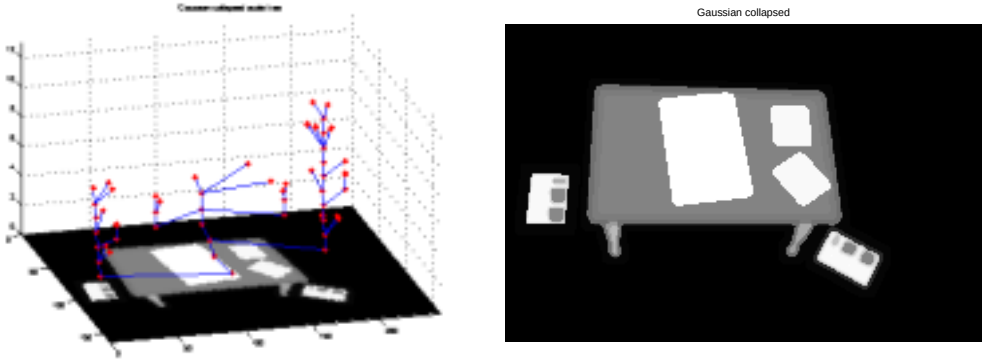


Figure 6.5: Collapsed tree of the gaussian version, $\kappa(T_g)$. On the right, the corresponding image. The original image is restored, and the effect of the smoothing filter is almost removed.

As a last example, a real image will be used. Figure 6.6 shows a small, simple image of a pair of snooker balls extracted from the sequence in figure 6.16. Even a simple image like this is associated with a complicated scale tree. The right hand side of figure 6.6 shows the associated tree, T_b, of the balls. The three long chains of the tree correspond to the two balls and the shadow of the white one.

Again, the collapsing operator, κ, can be applied to the scale tree of the balls. Figure 6.7 shows the scale selection sequence for all leaves of the scale tree of

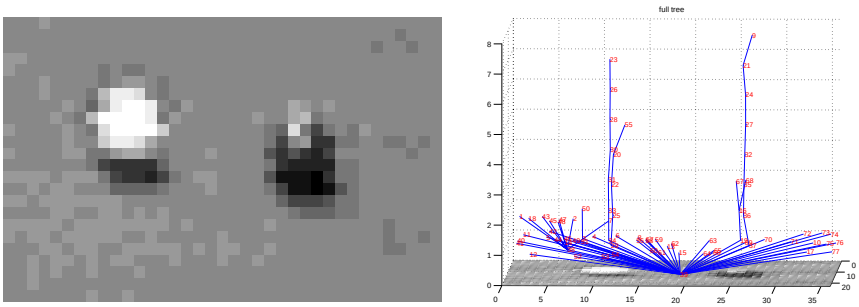


Figure 6.6: A 4-bit grey scale image extract from the snooker sequence and its scale tree.

the snooker balls. Figure 6.8 shows the result of collapsing all unbranched chains, $\kappa(T_b)$, into the maximum of the selection sequence, SSS, of all the leaves.

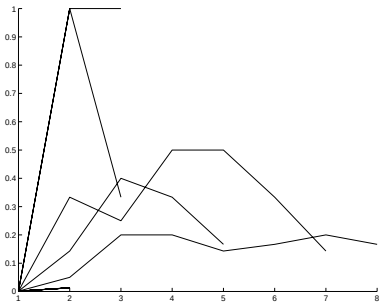


Figure 6.7: Scale selection sequence, SSS, of all leaves of the tree T_g.

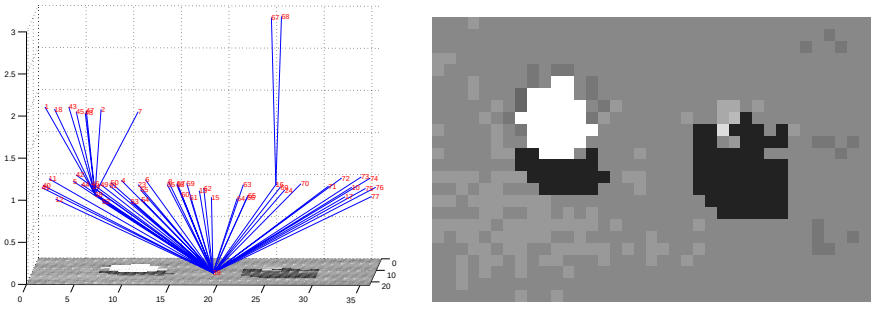


Figure 6.8: Scale tree after collapsing the unbranched chains. On the right, the associated image.

As it has been seen, the collapsing operator, κ, searches for a maximum in the scale space of the image. As a result, all unbranched chains of the tree (note that

nodes with children cannot be removed) are *collapsed* into one node. In terms of the image, all connected sets corresponding to those branches are merged into the maximum of the scale selection sequence. Therefore, the resulting image is simplified by sharpening the edges and flattening the original image.

Pruning scale trees

The second operator considered in this thesis is the *pruning* operator, $\pi(T)$. The pruning operator simplifies the original scale tree by removing irrelevant children of the tree. In any case, *irrelevant* nodes in the tree can be found by applying different criteria. In this thesis, two different criteria have been studied, using either grey scale value or scale information of the nodes.

The first criterion used is a slight variation of an standard quantisation. The difference in grey scale value between every node, N, of the scale tree and its father, $\mathcal{F}(N)$, is computed. If the absolute value of the difference is greater than a certain threshold, ε, the child is removed (the connected set is merged with the connected set associated with the father node). If less or equal, the node is retained.

$\emptyset(N)$ denotes the action of removing node N in the scale tree. $g(N)$ represents the grey scale value of the granule associated with node N. The pruning algorithm with the grey scale amplitude criterion can be defined as follows,

$$\pi_q(T,\varepsilon) = \begin{cases} N, & |g(\mathcal{F}(N)) - g(N)| > \varepsilon \\ \emptyset(N), & |g(\mathcal{F}(N)) - g(N)| \leq \varepsilon \end{cases} \quad \forall\, N \in T \tag{6.5}$$

An example of this algorithm is shown in figure 6.9. However, as the contrast of the image is already high, the pruned scale tree looks very similar to its original. Even so, 24 nodes out of 75 were removed applying a prune with $\varepsilon = 1$ (note, for example, the removed branch associated with the bottom right paper on the table).

A better example of the 'pruning by amplitude' algorithm is shown using the image in figure 6.10. A 'wood-grain' texture is added to the table in order to increase the low contrast nodes in the scale tree. A prune operator is applied to

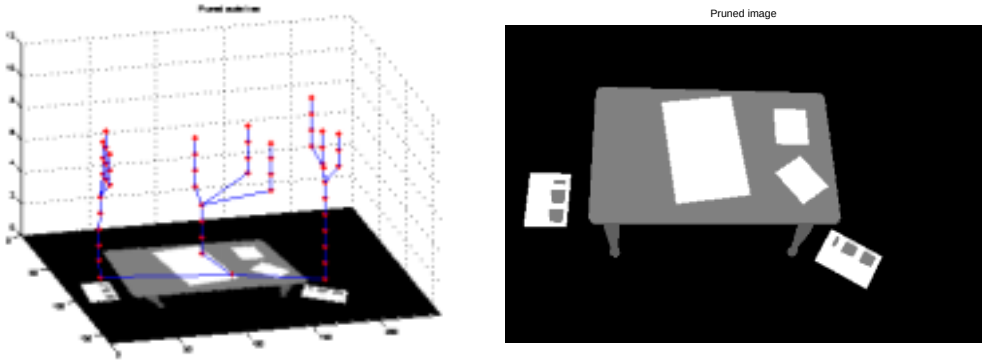


Figure 6.9: Pruned scale tree of the original table, $\pi(T_o)$. On the right, the corresponding image.

the new textured image. This time, as the nodes of the tree corresponding with the texture map have a low contrast between them, they are removed by the operator (see figure 6.11). The associated image, on the right, looks similar to the original. The simplified pruned tree can be seen on the left hand side.

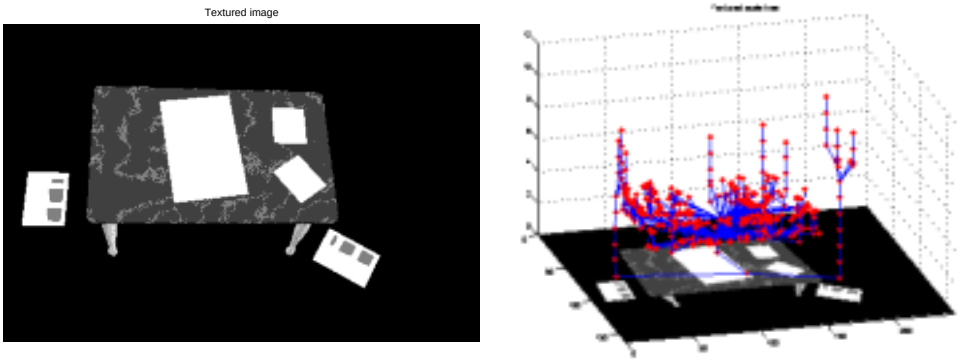


Figure 6.10: A textured version of the computer generated table (left). The corresponding scale tree on the right.

The prune operator with the grey amplitude difference criterion, $\pi_q(T)$, makes a 'local' quantisation of the original image. Note that the operator is different to a standard quantisation, as every node is analysed using only its 'local' father of the tree. So, for instance, low contrast edges may be preserved if they are not associated with a father link in the scale tree.

The second version of the pruning algorithm is defined using scale information

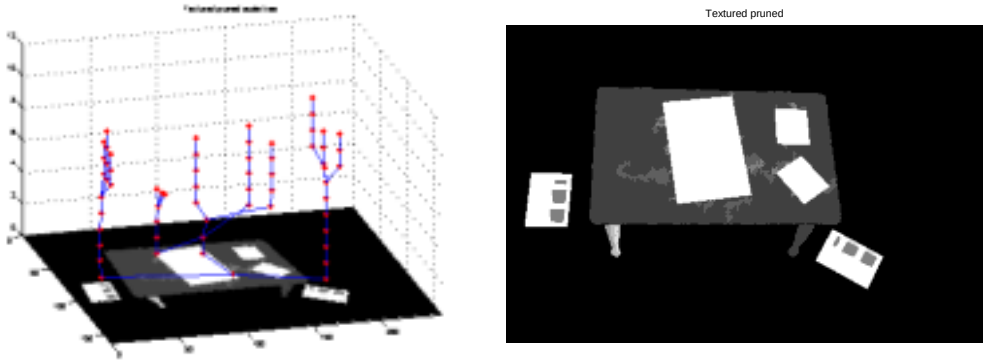


Figure 6.11: Tree and image after pruning using a grey scale amplitude criterion.

instead of grey value. In this case, the criterion is based in the relation between the scale of a node, $s(N)$, and the scale, or area, of its father, $s(\mathcal{F}(N))$. Again, a threshold is used to decide if the irrelevant node is 'small' enough to be removed. Note that, in this case, the threshold ϵ must be between $[0, 1]$, as by definition in a scale tree, the scale of a father node is always greater than any of its children. Note that, an $\epsilon = 0$ will leave the scale tree intact.

The new criterion can be written as,

$$\pi_s(T, \epsilon) = \begin{cases} N, & \frac{s(N)}{s(\mathcal{F}(N))} > \epsilon \\ \emptyset(N), & \frac{s(N)}{s(\mathcal{F}(N))} \leq \epsilon \end{cases} \quad \forall\ N \in T \tag{6.6}$$

Returning to the snooker balls example, a further simplification of the resulting collapsed tree can be made by using now a prune algorithm. As the tree of figure 6.8 shows, the collapsing operator is not able to remove the nodes at the bottom of the tree. These nodes are children of the root (their *level* of the tree is 1) and therefore, their corresponding scale sequence is not long enough (length 1) for the collapsing operator to be able to simplify them. The pruning algorithm, however, removes them as the scale difference between their father (the root) is large enough.

Figure 6.12 shows the resulting tree and associated image after the pruning operator is applied to the collapse tree in figure 6.8. At the end of this second

step, the simplification obtained has preserved the two objects of interest. In this case, the snooker balls.

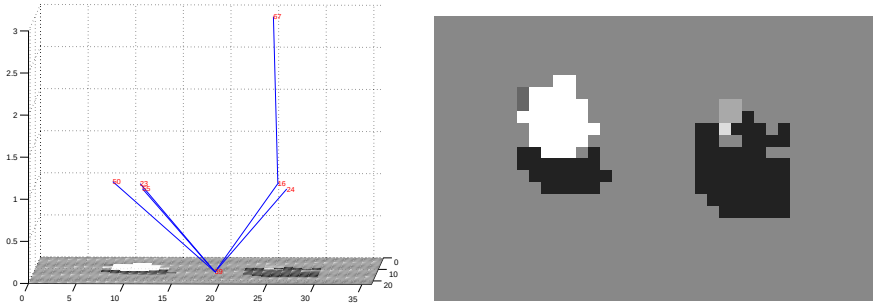


Figure 6.12: Pruning version of the collapsed tree obtained in figure 6.8. Both criterion, grey scale difference and scale relation are used in the example ($\varepsilon = 2$, $\epsilon = 0.01$).

To end this section, a final example of combining collapsing and pruning is given. A sequence of a moving hand is analysed by constructing the scale tree of every frame and applying a collapsing following by a pruning. The middle row in figure 6.13 shows the collapsed and pruned versions of the scale trees of the hand images in the first row, $\pi_s(\pi_q(T_t))$. This example illustrates again that the scale tree is, in practice, somewhat invariant to scale, rotation or minor shape changes. Moreover, the two algorithms developed so far can successfully simplify the images (bottom row of figure 6.13). In this case, for instance, the sequence size is reduced from 120Kb to 12Kb when stored as raw binary data.

In this section, two different algorithms to reduce the complexity of scale trees have been introduced, collapsing and pruning, (κ,π). In the case of the pruning operator, the use of thresholds require previous knowledge of the original image and it would be desirable to replace them with a more principled step. Based on, for instance, a probabilistic decision. As future work, other criteria for pruning scale trees can be investigated.

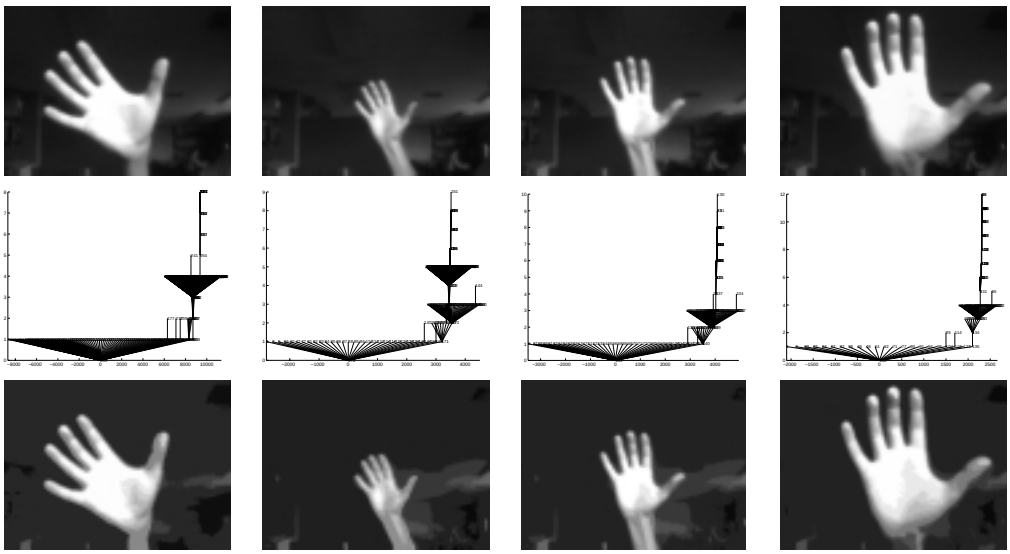


Figure 6.13: Top row, some frames from a 8 bit grey scale movie sequence. The second row shows collapsed and pruned scale trees side on. The third row shows the images corresponding to the reduced trees.

6.2 Towards object trees

Scale trees on their own can be a good approximation to *object trees*. An object tree is a structure where every node represents a meaningful object of the image. This section will discuss the use of scale trees as an useful representation of an image. Different applications will use the nodes of the scale tree as handles to 'real' objects of the scene. Obviously, there is still a long way to go before we are able to 'compare' scale trees with object trees. The following sections will show some examples of where scale trees fail to represent the topology of the image. As a consequence of this, new algorithms to refine the scale tree in order to get closer to an object tree will be analysed.

In any case, scale trees perform a *segmentation* of the original image. In order to segment an image, all the pixels that define the image must be grouped together forming different *regions* of interest or *meaningful objects*.

6.2.1 The segmentation problem

In general, different segmentation algorithms (see chapter 2) have been applied to solve this problem. To group two different pixels, (i, j), of an image in the same cluster or region, a function, f, of some measure taken of the pixels, $f(M_i, M_j)$, must be computed. This measure, M, can be composed from a set of different features of the pixel. For every pixel, i, of the image, a vector of measures, $\underline{\mathrm{M}}_i$, can be defined with information about hue, saturation, grey scale value, x and y position in the image, motion vector, stereo disparity, texture, etc.

$$\underline{\mathrm{M}} = [h, s, v, x, y, m, st, t \ldots] \tag{6.7}$$

An optimal solution to the problem would be to compute the function f for all the pixels of the image and all the possible grouping of those pixels in different regions. If the size of the original image is relatively big, the computation cost required to solve the problem makes the solution impractical.

Section 2.1 presented a general review of different suboptimal solutions appearing in literature. As defined in chapter 5, scale trees are a *suboptimal* approach to successfully segment images and, as a suboptimal solution, some assumptions have been made. As a first step, scale trees use flat zones or connected sets of the original image to perform the segmentation. The use of connected sets implies that regions of the same pixel intensity cannot be split into two different regions of interest.

As shown in chapter 3, the sieve algorithm that is used to build the tree simplifies the image by merging neighbouring extrema granules, or flat zones, of increasing scale. Therefore, the sieve uses three features, intensity information and the (x, y) coordinates of the pixels. As some examples have shown, the resulting scale tree from the complete sieving process can successfully segment objects in real images.

In the first instance, there can be some nodes in a scale tree that approximate

the ‘idea’ of a node in an object tree. A node in the scale tree that completely corresponds to a meaningful object in the scene can be called a *grandmother node*. It will be desirable for an scale tree to have a number of grandmother nodes that represent the different objects in the image. Obviously, that will not always be the case as the scale selection of the sieve algorithm can fail detecting an object and it can be represented in different branches or subtrees of the scale tree.

A) *B)* *C)*

Figure 6.14: Image editing using a grandmother node. *A)* The original image. *B)* A screenshot of our tree based image editor showing a collapsed and pruned version of the original scale tree of the left figure. A node that approximates the grandmother node, N_g, and its associated segment of the right hand face object are highlighted. *C)* shows the effect of changing the spatial position elements of N_g.

Figure 6.14 shows an example where a grandmother node can be found in the scale tree. Figure 6.14*A)* shows an image of two faces. Figure 6.14*B)* shows the graphical user interface of a tree based image editor used to find the grandmother node representing one of the faces. First of all, the original scale tree is collapsed and pruned to simplify the searching for a grandmother node. A node that approximates the *object face* is then manually found, N_g. The segment associated with the node N_g is highlighted in green on figure 6.14*B)*. The located node, N_g, can be used as a ‘handle’ pointing to the pixels representing the face. Figure 6.14*C)* shows, for instance, the result of moving the spatial coordinates of all the handles and so, the original face is moved.

This approach uses a heuristic decision to find the node N_g. Automatic meth-

ods to identify grandmother nodes in scale trees, and therefore objects in the original image, would be more desirable.

6.2.2 Scale tree applications

As introduced in the previous example, scale trees can already be used in different applications. This section will introduce two different applications where the scale tree was applied. First of all, the tree structure will be used to interpret motion in image sequences. The second application will use the scale tree to interpret the depth position of different granules in the scene.

Motion estimation

One of the applications where scale trees were applied was motion estimation. A similar technique applied by Salembier et al. in matching motion with max trees was used [82, 87, 88]. The motion estimation is based on a new operator defined over the scale tree. In this case, a *motion operator* is defined to recognise nodes that are moving in an image sequence.

The motion operator is based on a chosen *motion model*. There are many possibilities to choose for the motion model. For instance, in a 2D case, translations, rotations, scale or affine models can be used [35, 39]. Here, it is assumed that the translation followed by any pixel (x, y) going from image f_{t-1} to image f_t is (d_x, d_y). In that case, a measure of the *mean frame displaced difference* can be computed for every node, N, of the scale tree, T_t.

$$D^{f_t}_{f_{t-1}}(N) = \frac{\sum_{(x,y)\in N} |f_{t-1}(x - d_x, y - d_y) - f_t(x, y)|}{s(N)} \tag{6.8}$$

This measure computes the normalised difference between the pixels of node N in the image f_t and the same pixels (taking into account the motion model) in the previous image sequence f_{t-1}. The mean displaced difference, $D(N)$, of

nodes following the chosen translation (d_x, d_y) will be lower than the stationary nodes.

In practice, however, obtaining movement from image sequences using only two frames is not very robust. To solve this problem, a recursive term is added into the equation. The mean displaced difference, D, is measured between the current image, f_t, and the previous image, f_{t-1} and between the current image and the previous processed image, denoted by $\mu(f_{t-1})$.

The *motion criterion* used to calculate the motion of all nodes of the tree can be defined as follows,

$$C = aD^{f_t}_{f_{t-1}}(N) + (1-a)D^{f_t}_{\mu(f_{t-1})}(N) \tag{6.9}$$

where the parameter a $(0 \leq a \leq 1)$ defines the *memory* of the motion criterion. If a is near 1 the criterion is memoryless but it is able to detect faster new changes in the image sequence. On the other hand, if a is set near 0, the estimations are based in the observation of a large number of frames and new changes are going to be detected slowly.

The final motion operator, $\mu(T_t, \lambda)$, will work on the scale tree by removing the nodes that do not undergo the specified displacement (d_x, d_y). A threshold, λ, can be used to determine if the node is following the given motion.

$$\mu(T_t, \lambda) = \begin{cases} N, & C(N) \leq \lambda \\ \emptyset(N), & C(N) > \lambda \end{cases} \quad \forall\, N \in T_t \tag{6.10}$$

Figure 6.15 shows a diagram of this motion defined in equation (6.10). In summary, the motion connected operator follows these steps:

- The scale tree, T_t, associated with the input image, f_t, is created.
- The motion criterion, $C(N)$, is computed for every node in the scale tree.
- A threshold λ is used to remove nodes that do not undergo the specific

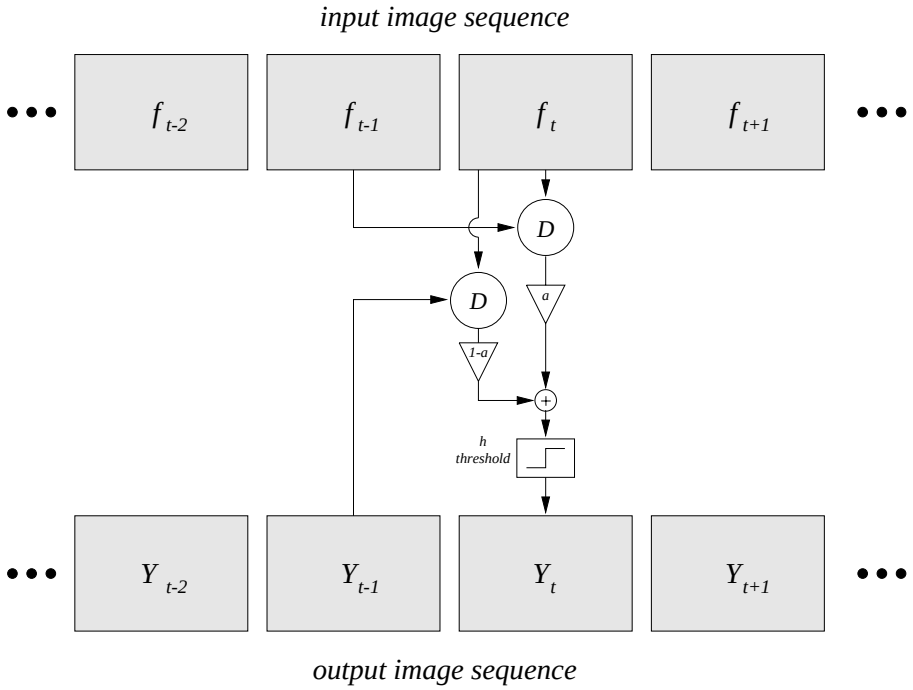


Figure 6.15: Diagram of the motion operator.

motion defined by the translation model (d_x, d_y). If the motion measured in the previous step is greater than the threshold, $\mu(N) > \lambda$ the node is removed.

- The associated image of the processed tree is computed, $\mu(f_t)$.

The figures 6.16 and 6.17 show the use of the motion operator on 'real' sequences. In this case, a translation of $(d_x, d_y) = (0, 0)$ is chosen, so, the operator will remove moving nodes of the scale tree and will preserve static ones. The left column of figure 6.16 shows the original sequence taken from the Snooker World Championship of 1997. The white ball is moving to the top of the pool and it is about to hit the red ball on the right.

The second column shows the resulting output images after applying the motion operator, $\mu(f_t)$. The right column shows the difference between the input sequence and the output, $f_t - \mu(f_t)$. The motion criterion operating in the scale trees of the originals completely removes the moving balls and the moving cue.

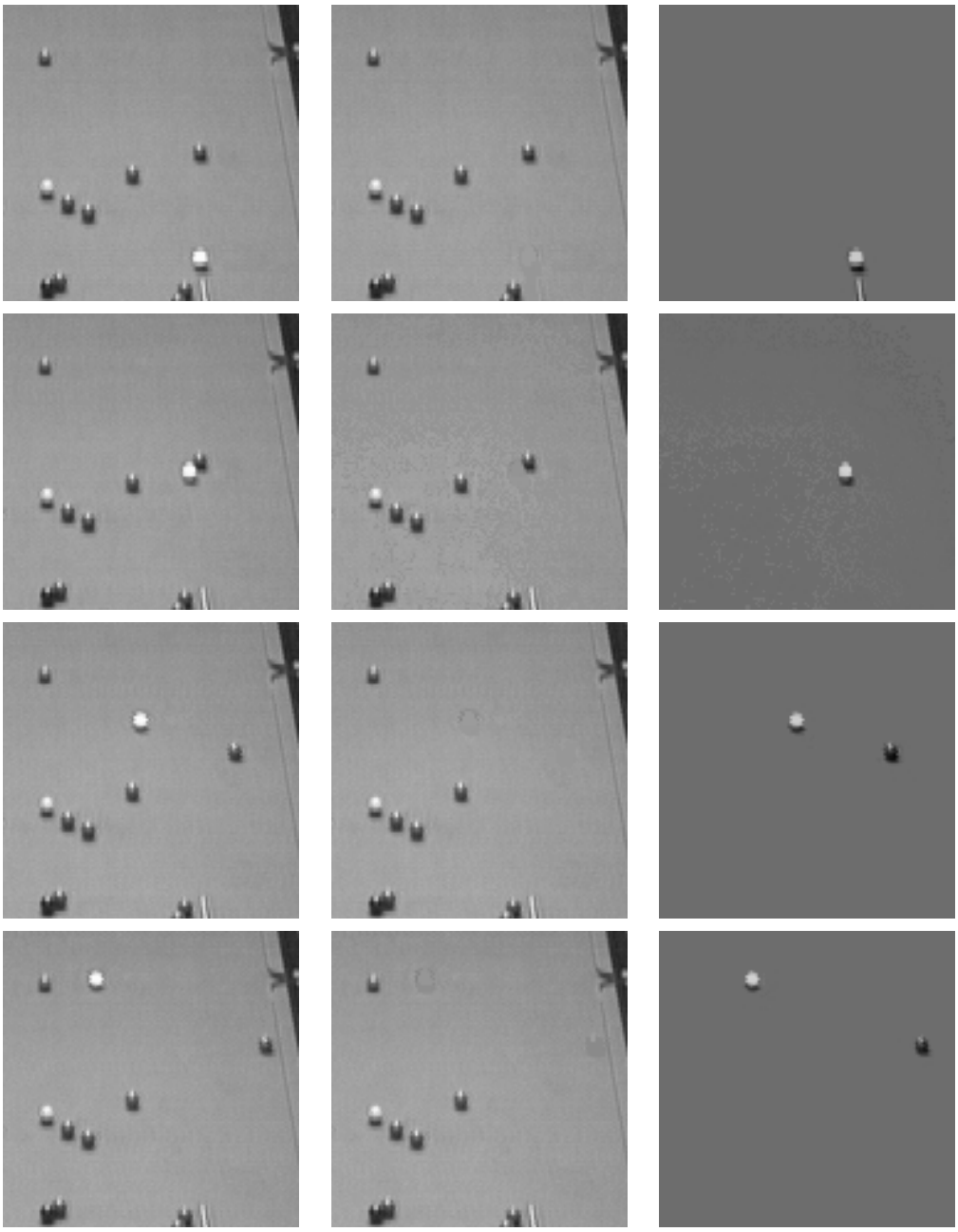


Figure 6.16: Motion operator over a real snooker sequence.

Figure 6.17 shows the same operator, this time working on a tennis sequence. At the right hand side column, the players are successfully segmented. However, parts of the background are removed as well due to changes in the illumination.

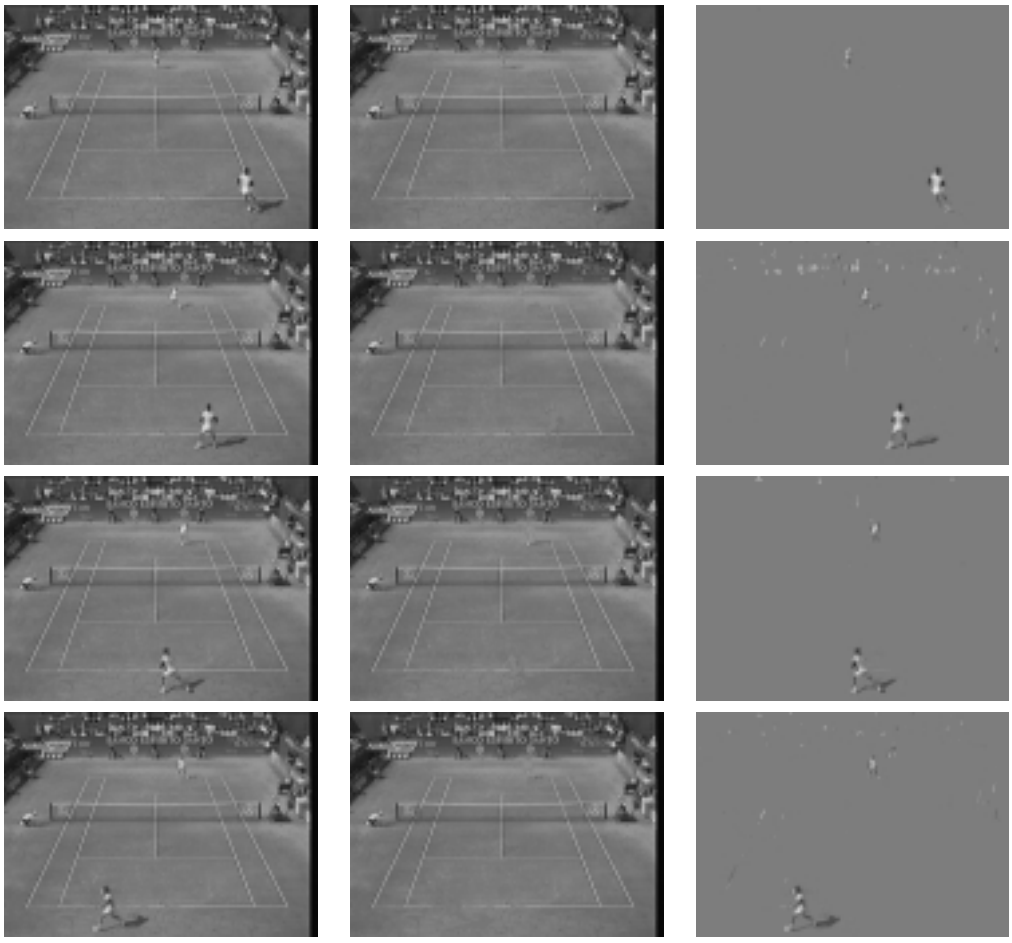
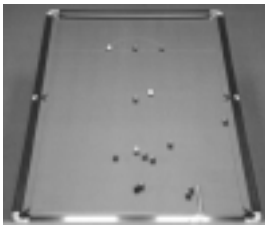


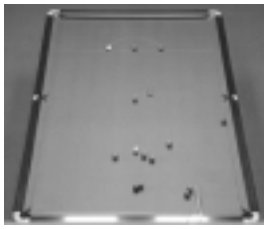
Figure 6.17: The motion operator using scale trees in a tennis sequence. Left column shows the original sequence, middle and right columns show the output of the motion operator and the its difference with the original.

The motion operator chosen in this section was based on the same operator used with max trees [87, 87]. It would be good idea, then, to compare the results of the operator working on the same sequences but using a max tree representation instead of the scale tree. Figures 6.18 and 6.19 show the results. The scale tree operator successfully segments the snooker balls and the tennis players out of the original images.

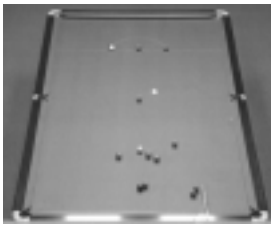
This time, however, two different operators may be applied. As the max trees are oriented towards the maxima of the images, only bright nodes can be detected



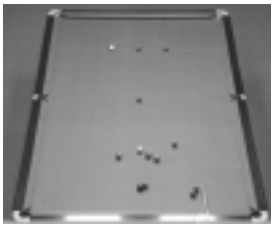
A) Original sequence f.



B) Motion Operator $\mu(f)$.



C) Dual operator $\mu^*(f)$.



D) Residue $\mu^*(f) - f + \mu(f)$.

Figure 6.18: The same motion operator working with max trees.

as moving nodes (dark nodes are 'hidden' in the bottom of the max tree structure) [87]. The dual operator, $\mu^*(f)$, operating on the min tree must be defined in order to detect the motion of dark objects.

In the case of scale trees, as extrema of both signs (maxima and minima) are used to seed the algorithm, the definition of a dual operator is not necessary. However, using scale trees, moving nodes still have to create extrema in the original so they will form leaves in the tree.

Stereo

The same operator applied to estimate motion can be used for analysing stereo pairs. The left and right images of the stereo pair can be considered as the images f_{t-1} and f_t of a sequence. This time, the initial motion will only consists of translations in the x coordinate, $(d_x, 0)$.

The *stereo operator* will compute the motion criterion for every node of the tree for a given range of displacements, $d_{min} \leq d_x \leq d_{max}$. The most likely displacement followed by the node N, (d_x that minimises the frame displaced

A) Original sequence f.

B) Motion Operator $\mu(f)$.

C) Dual operator $\mu^*(f)$.

D) Residue $\mu^*(f) - f + \mu(f)$.

Figure 6.19: Motion operator, μ, using max trees.

difference, $D(N)$), will be used as the depth value of the associated region. This technique has already been applied to max trees with success [48, 76, 82].

To finish this section, adding additional information to the scale tree will be studied. Figure 6.20 shows a sequence where added depth information has been used to resolve an occlusion problem. Until now, every node of the scale tree stored information of the (x, y) coordinates of the pixels of the associated granule and its grey scale value. Information about depth can be added to every node, saying how far from the camera it is located.

The examples on figures 6.20 and 6.21 use this information to solve the occlusion problem that occurs when one region overlaps another. For example, the added disparity information indicates that nodes associated with the finger are in front of the nodes associated with the doll. In this case, if the doll is moved over the finger, the new depth information will make the doll appear behind the finger.

Figure 6.21 shows a similar example. This time, information of colour (hue and saturation) is used as well to represent the image associated with the tree. In both examples, the added information is just used as a later clue. A better

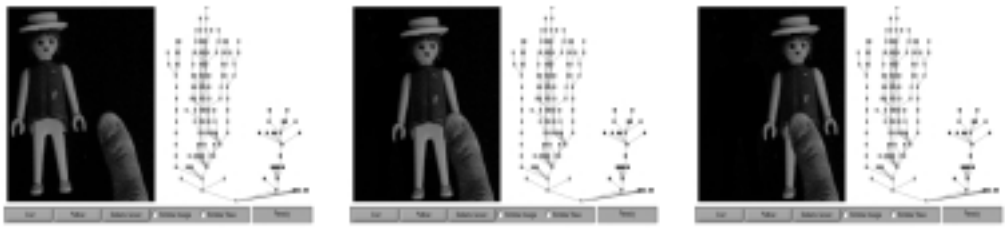


Figure 6.20: Three screenshots of a graphical user interface showing the use of additional information, in this case, disparity information to resolve occlusions of objects in the scene.

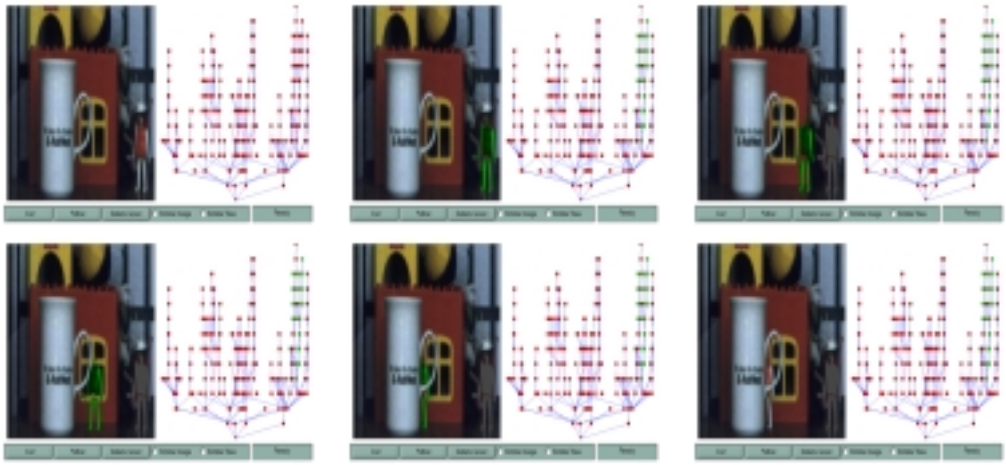


Figure 6.21: Similar example of using depth information to resolve occlusions of objects. Left side of the images show the doll being occluded behind the cup. Right side show the scale tree of the picture with the selected granule highlighted in green. Adding colour information (hue and saturation) to every node improves the results.

approach would be to use the additional 'knowledge' to readjust the actual scale tree structure.

6.2.3 Refining scale trees

The last example shows that scale trees, T_s can be an useful representation of the object tree, T_o. However, it will often be necessary to bring in other evidence and modify the T_s to make it a better candidate for the T_o. Figure 6.21 successfully overlapped the cup and the moving doll. However, it is impossible to find a better

grandmother node that represents the doll and, for instance, the hat is left behind.

It is obvious then that further evidence is still needed in order to transform the scale trees into meaningful object trees. It would be desirable, to *refine* the scale tree structure, by moving, merging or deleting nodes to simulate a better approximation of T_o or, at least, to get a greater number of grandmother nodes in the tree.

In a first approximation, one could say that regions of the image with the same motion vector are part of the same object. Using the same assumption, nodes of the tree with the same stereo depth, or the same colour will represent the same object. In that case, those nodes should be repositioned on the tree using the new motion, colour or stereo information.

Using the graph notation again, if different information is added to the scale tree, the 'new' tree can be represented as a *labelled* graph, $G = (V, E, L)$, where:

- **Vertices** or **nodes** $V = 1, 2, 3..., n$.
- **Edges** or **links** $(i, j) \in E \subseteq V \times V$.
- **Labels** $L : V \rightarrow l$, with $l \in \{h, s, v, x, y, m, st, \dots\}$.

Now, each node in the scale tree is represented with a set of *labels* or information attached to it. These labels should be used to rearrange the original topology of the scale tree in order to get the representation closer to that of an object tree. Consider the following example.

Figure 6.22 shows the image of a red teapot against a beige background. As the scale tree is, again, too complicated, a collapsed and pruned version of the original will be used (figure 6.23). As figure 6.24 shows, the hole in the handle is assigned as part of the teapot tree. The problem is to reassign that node to the background so a grandmother node representing only the teapot can be found.

In this case, only one static image is used so information about motion or stereo depth cannot be obtained. However, colour information is available so saturation, for instance, can be used to distinguish the teapot from the background

Figure 6.22: Original teapot used to 'refine' the scale tree.

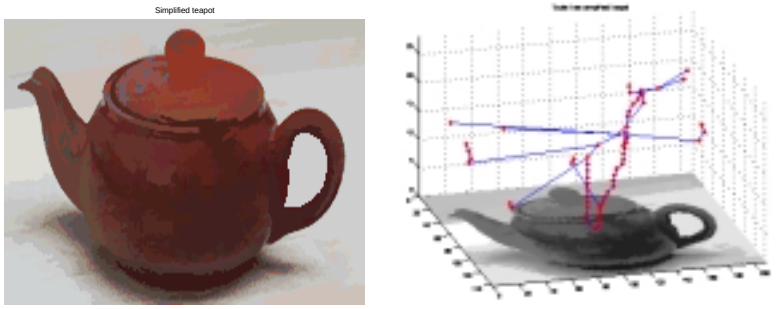


Figure 6.23: Simplified teapot and its associated scale tree.

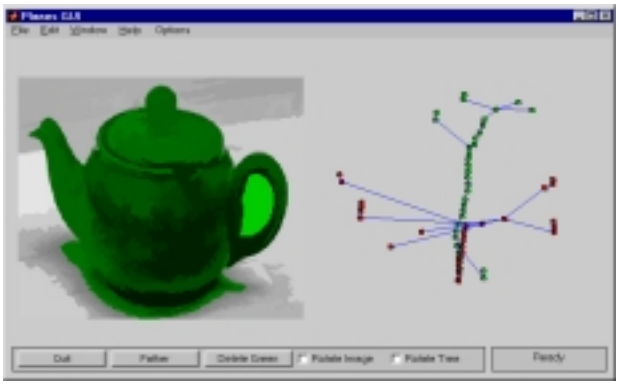
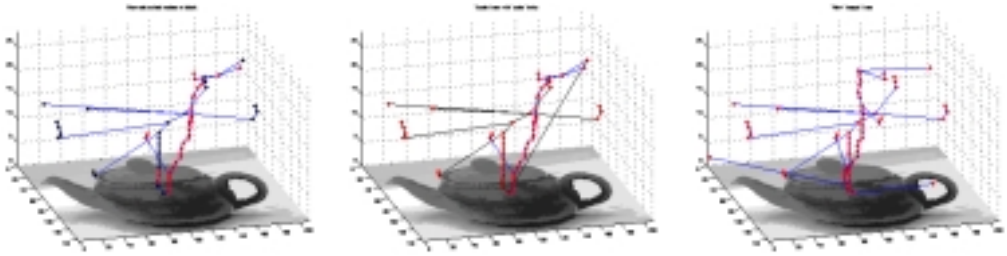


Figure 6.24: In green, the associated granule of the 'closer' grandmother node representing the teapot. The hole in the handle is included in the same subtree.

(beige contains enough red to make hue an unsatisfactory feature). Figure 6.25*A)* shows the nodes of the tree with saturation between $0 \leq s \leq 0.2$. These nodes

can be moved in the tree so there is always a father link relation connecting them.

This is done in two steps. Firstly, new links (*uncle* links) are added to the tree connecting the selected nodes. The algorithm that creates the new links is simple, starting from a selected node, it visits the tree going down to the root until another selected node is found and an uncle link is established. The second step involves removing the old scale tree link and preserving the new colour based relation. Figures 6.25*B)* and 6.25*C)* show these two steps. Note that in the final tree, the node corresponding to the hole is assigned to the background. The final tree structure now contains a grandmother node that represents only the teapot (figure 6.26).



A) *B)* *C)*

Figure 6.25: *A)* shows the nodes of the tree (in black) with saturation between $0 \leq s \leq 0.2$.*B)* new uncle links connecting last nodes added (in black).*C)* The 'old' scale links are replaced with the links in the last step.

In the future, this approach can be expanded. Instead of using just a new link, many 'uncle' links can be obtained from the labels of every node. From all links connecting one node, an improved algorithm should statistically *decide* which is the best candidate to obtain an object representation.

6.2.4 Object trees

Once the scale-tree is as close to an object tree as possible, the tree structure can be used in many different tasks. The object representation cannot only be used for filtering, but as a set of handles that allocate the different visual object planes in

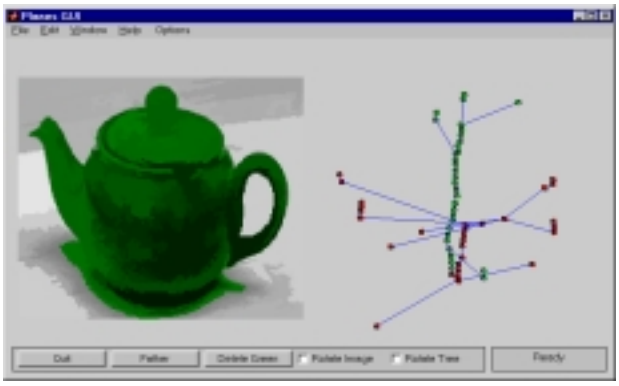


Figure 6.26: The new grandmother node now only represents the teapot.

the MPEG-4 standard.

It can also be used for object or shape recognition. This application can be done at two levels; (1) The tree structure itself codes the object topology that is, to a large extent, independent of geometrical scaling, rotations and distortions and (2) a more detailed matching can be performed by also using attributes of the nodes, such as shape or topology of the associated granule.

This chapter has introduced two different applications where scale trees were used with varying amount of success. It has introduced some algorithms to reduce the complexity of scale trees and new techniques have been introduced to modify the topology of scale trees in order to get a closer representation of an ideal object tree.

These new algorithms are in the first stage of development. There is a clearly opportunity to enhance this work using new and improved algorithms and techniques.

Chapter 7

Conclusion

In this thesis, a new scale tree representation of an image has been presented. Scale trees are derived from a scale-space, mathematical morphology based algorithm called a sieve. The scale tree structure represents the amplitude of the original image in a hierarchical structure in which every node of the tree is associated with features of specific scale.

As a brief summary of the contents of this thesis, chapter 2 gave a general view of different algorithms used in image processing and, more specifically, some techniques used in literature to solve different segmentation problems.

A relatively new scale-space segmentation algorithm, the *sieve*, was used to build scale trees in chapter 3. Chapter 4 presented some other image processing hierarchical structures from the literature. The scale tree structures are described in chapter 5.

Finally, chapter 6 described some applications where scale trees are already being used. Methods to reduce the complexity of the trees, such as pruning or collapsing were also shown.

7.1 Discussion

It has been demonstrated that scale trees are a useful representation of images. They are insensitive to geometrical transformations of the original image such as rotations, zooms, or any transformation that preserves the image topology. As scale trees also simplify the image by preserving all contour information, they are an interesting structure to be applied in different image processing tasks.

Scale trees have already been used in some applications such as segmentation or motion estimation. There is also a relatively fast algorithm to construct scale trees (see appendix B).

Scale trees are a reasonable approximation to objects trees. It has been shown that *grandmother nodes* (nodes that represents a meaningful object of the scene) can be found inside the scale tree structure. These nodes can be used, for instance, in object recognition. In this framework, the recognition could be done using the tree topology itself or the attributes associated with the grandmother node.

Grandmother nodes are also extremely close to what MPEG-4 defines Visual Object Planes. Segmentation techniques, such as this one, could be used to divide the original image sequence into significant 'planes' that will be coded and compressed separately.

7.2 Future work

Obviously, the scale tree is not the ideal representation for an image. There are still problems to resolve in order to get a true object representation of the image. Some examples have been already shown where the scale tree structure does not represent meaningful objects of the scene (such as the teapot example in section 6.2.3).

In any case, methods to modify and restructure the original tree topology using extra information of the nodes have been studied. Colour information has been used to adapt the tree structure to a closer object representation. However, using

different attributes associated with the nodes such as motion or stereo information together with robust methods to decide between them would be more desirable.

Finding the grandmother node associated with the object is not a trivial problem. Even when an exact representation of an object in the image can be associated with one node of the scale tree (a grandmother node), the search algorithm is a complicated problem.

Heuristic methods have been used in this thesis, however, automatic search methods to find grandmother nodes will be required in the future. Probabilistic descriptions could be used to analyse the most likely position of such nodes in the tree.

In summary, scale trees are a fast and a suitable starting point for a more refined, object based, segmentation of an image. Scale trees then are, a first approximation to real object trees. However, different methods to refine scale trees in order to get a closer object representation are still required.

Appendix A

Matlab scale trees

The algorithm to build scale trees was coded using MATLAB[1] version 5.2. The lack of pointers in MATLAB® makes the use of a standard tree representation where a set of pointers reference other nodes in the tree (children nodes) impossible. A binary tree was chosen as the data structure to store the scale tree information.

Scale trees are not necessarily binary trees. The example of figure 5.6 in page 46 shows, for instance, a node N_i of the scale tree representation having more than two children nodes. Any tree can be represented as a binary tree if the *father* links of the tree are transformed into *child* and *following brother* links as figure A.1 illustrates.

A standard MATLAB® structure class was used. A set of vector arrays stored the hierarchical information of the tree (stored as fields of the structure class). First of all, every node in the tree is numbered ($1 \ldots N$). This label is then used to store the child, the following brother, and the father information of every node in the tree.

The hierarchical information of the tree can be stored then in three vector arrays; *father[i]* denotes the father node of node i, *child[i]* its first children and *nextbrother[i]* the following sibling in the structure.

[1] © COPYRIGHT by The MathWorks, Inc.

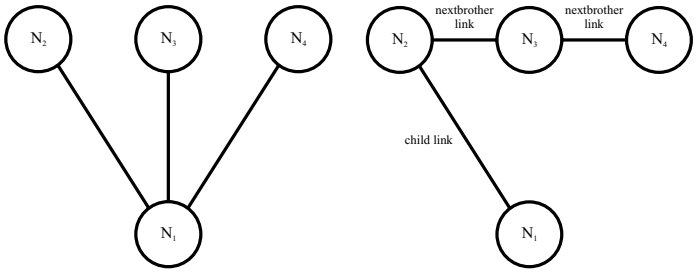


Figure A.1: A 'normal' tree represented as a binary tree. Right hand figure shows the binary representation of the left tree. Every node can only have two links, representing a child or a sibling in the tree.

Using, for example, the tree in figure A.1, the three arrays coding the scale tree can be seen in table A.1. If a link of the tree does not exist $\emptyset$ is used.

father[1]=$\emptyset$	*child[1]=2*	*nextbrother[1]=*$\emptyset$
father[2]=1	*child[2]=*$\emptyset$	*nextbrother[2]=3*
father[3]=1	*child[3]=*$\emptyset$	*nextbrother[3]=4*
father[4]=1	*child[4]=*$\emptyset$	*nextbrother[4]=*$\emptyset$

Table A.1: *father*, *child* and *nextbrother* arrays storing the tree in figure A.1.

In order to improve the speed when parsing the tree, new *pointers* (in this case, arrays) can be added. A *root* field with information of the root node in the tree. The *firstbrother* and *Nchildren* arrays contain information of the first sibling of a group of brothers and the number of children of the node.

The same technique (using array fields) is applied to store the information of each node. Fields such as *value*, *number*, *level* store the grey scale value of the granule, the number of pixels or the level in the tree.

The pixels of the associated granule are also stored. In the scale tree representation, each granule is stored as a node of the tree. As a scale processor is used, the granule associated with a *father* node includes the smaller granules of all its *children* nodes. Storing all granule pixels for every node of the tree would be very inefficient. To solve this, each node stores only the pixels that make it different from its children.

Using the examples in figure 5.16 and 5.17. If every granule is stored, it is then necessary to keep the three white pixels associated with the leaf node. The four grey pixels granule formed when the white connected set is merged to the grey pixel and finally, the whole image (16 pixels) is stored as the associated granule for the root node.

In that case, we need to store $3 + 4 + 16 = 23$ pixels for a 4×4 pixels image. However, only the different pixels that form the new granule need to be stored. In this example, only the black pixels are associated with the root node, the grey with the middle one and the three white pixels with the leaf node. The granule associated with a node as then represented in the whole branch, or subtree, formed from that node.

To store the pixel positions, two arrays are used, *nodelist* keeps a sequential list with the (x, y) coordinates of every pixel of all granules. The field *plist[i]* points to the start position in *nodelist* of the pixel list corresponding to node i. The array *pixels* keeps the actual number of pixels stored in that node (the difference between its associated granule and all its children). The algorithm to obtain all the pixels of the granule associated with node k can be done as follows:

ALGORITHM 2 *Granule extraction*

```
clear granule-list
for each node i in the subtree with k as root do:
    list = nodelist{from plist[i] to plist[i]+pixels[i]}
    Add list to granule-list
end for
```

Sometimes, however, it is desirable to know which node of the scale tree a certain pixel is represented, to do so, another field was added. A matrix of equal size to the original image is enough to store the node any pixel belongs to. In this case, using this *node* field, pixel (x, y) will be associated with node *node[*x,y*]*.

All these arrays were contained in a MATLAB® structure class variable. As a brief summary, table A.2 shows a list wit all the fields in the structure that represents the scale tree. The type size, together with a description of each field is

given.

Field	Size	Description
father	$1 \times N$	Father node
child	$1 \times N$	First children
nextbrother	$1 \times N$	Next sibling
firstbrother	$1 \times N$	First brother of all siblings
Nchildren	$1 \times N$	Number of children
Nnodes	$1 \times N$	Order of the tree
root	1×1	Root node of the tree
nodelist	$1 \times (P \times Q)$	Pixels list
plist	$1 \times N$	Pointer to the start of the granule in *nodelist*
node	$P \times Q$	Node label for every pixel in the image
value	$1 \times N$	Grey scale value of the corresponding granule
number	$1 \times N$	Number of pixels of the granule
pixels	$1 \times N$	Number of stored pixels in that node
level	$1 \times N$	Level in the scale tree

Table A.2: Summary of the different fields in the MATLAB *tree* structure using an image of $P \times Q$ pixels with N nodes.

Appendix B

Sieve Results

This appendix will introduce some experimental results using the tree algorithm presented in this thesis. The order of scale trees and the complexity of the algorithm will be analysed for a different set of real images.

Figure B.1 shows the results of studying the scale tree complexity of 20 different real images. The images were sub-sampled from their original size of 768×512 pixels to $P \times Q = 700 \times 500$, 600×500, 500×500, 450×450, 390×390, 315×315, 225×225 and 100×100 pixels. The sub-sampling was achieved by taking a block of the desired size at a random position of the original image. The algorithm was executed as a MATLAB® .mex file on a PII 300Mhz.

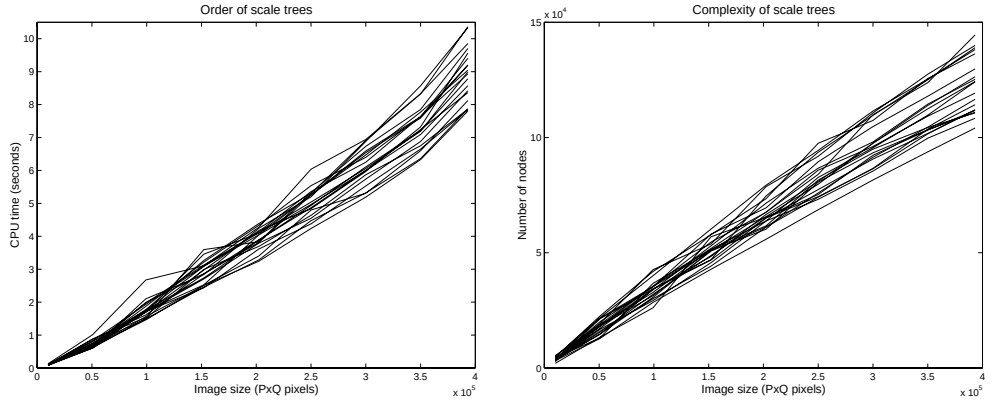


Figure B.1: Complexity of the scale tree algorithm.

The left hand side of figure B.1 shows the CPU time spent computing the scale tree of the images. The y coordinate shows the CPU time in seconds and the x coordinate shows the total number of pixels $N = P \times Q$ of the image. The right hand side shows the order (number of nodes) of the resulting scale trees. All 9 measures (9 image sizes) corresponding to the same image are joined together in the same line. Clearly, the CPU time increases with the number of pixels, N, of the image as the number of nodes of the resulting scale tree.

Figure B.2 shows the results (CPU time and number of functions calls) for each function in the scale tree implementation. The different functions of the implementation can be grouped together in three different processes. General initialisation and memory allocation, computing the sieve algorithm and the building of the scale tree structure.

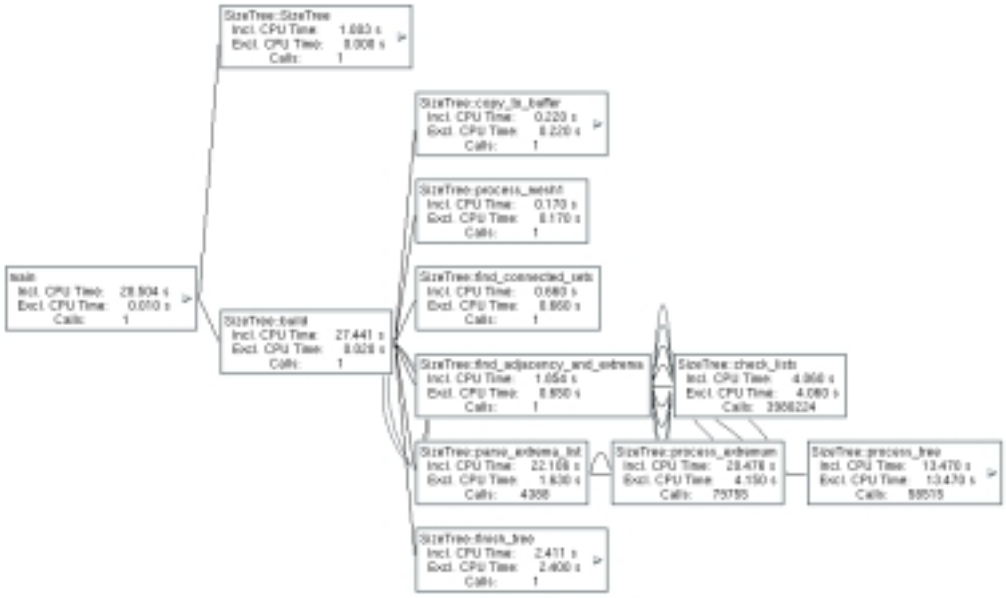


Figure B.2: CPU time and number of calls for some functions of the scale tree implementation. A 768×512 pixels image was used. In this case, the algorithm was executed on a O2 SGI workstation.

Figure B.3 shows the CPU time spent for each of the previous processes of the scale tree algorithm. It can be observed that for small images (small number

of nodes in the scale tree) the order complexity of building the tree is almost the same as the computing the sieve algorithm. As the image size increases the time spent building the scale tree structure is greater than the time spent searching and merging the extrema of the image (sieve).

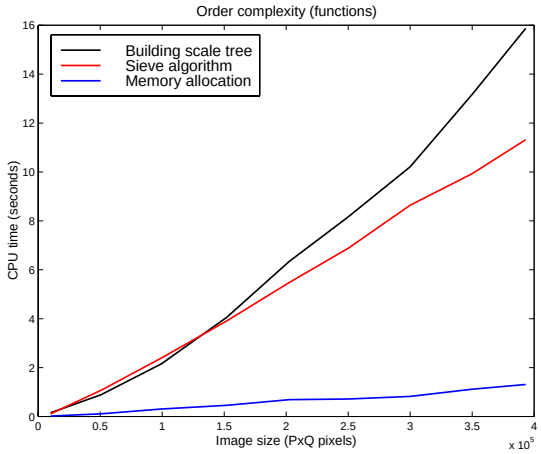


Figure B.3: CPU time of each process of the scale tree algorithm.

Bibliography

[1] Rolf Adams and Leanne Bischof. Seeded region growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):641–647, June 1994.

[2] J. Babaud, A. P. Witkin, M. Babaud, and R. O. Duda. Uniqueness of the Gaussian kernel for scale-space filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:26–33, 1986.

[3] Faris Badii and Janaka Jayawardena. Region growing and global labeling in image analysis. In *Proceedings of International Conference on Pattern Recognition*, pages 656–659, 1984.

[4] D. H. Ballard and C. M. Brown. *Computer Vision*. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1982.

[5] J. Andrew Bangham, T. G. Campbell, and R. V. Aldridge. Multiscale median and morphological filters used for 2D pattern recognition. *Signal Processing*, 38:387–415, 1994.

[6] J. Andrew Bangham, P. Chardaire, C.J. Pye, and P.D. Ling. Multiscale nonlinear decomposition: the sieve decomposition theorem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(5):529–539, 1996.

[7] J. Andrew Bangham, Richard Harvey, and P.D.Ling. Morphological scale-space preserving transforms in many dimensions. *Journal of Electronic Imaging*, 5(3):283–299, July 1996.

[8] J. Andrew Bangham, Paul Ling, and Richard Harvey. Scale-space from nonlinear filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(5):520–528, May 1996.

[9] J. Andrew Bangham, Paul Ling, and Robert Young. Multiscale recursive medians, scale-space and transforms with applications to image processing. *IEEE Transactions on Image Processing*, 5(6):1043–1048, June 1996.

[10] J. Andrew Bangham, Javier Ruiz Hidalgo, and Richard Harvey. Robust morphological scale-space trees. In Stephen Marshall, Neal harvey, and Druti Shah, editors, *Proceedings of Noblesse Workshop on Non-Linear Model Based Image Analysis*, pages 133–139, Glasgow, July 1998.

[11] J. Andrew Bangham, Javier Ruiz Hidalgo, Richard Harvey, and Gavin Cawley. The segmentation of images via scale-space trees. In Paul H. Lewis and Mark S. Nixon, editors, *Proceedings of British Machine Vision Conference*, volume 1, pages 33–43, Southampton, UK, September 1998.

[12] P. J. Besl and R. C. Jain. Segmentation through variable-order surface fitting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10:167–192, 1988.

[13] S. Beucher and Fernand Meyer. The morphological approach to segmentation: the watershed transformation. In E. Dougherty, editor, *Mathematical Morphology in Image Processing*, pages 433–481, Marcel Dekker, New York, USA, 1993.

[14] Andrew Blake and Alan Yuille. *Active Vision*. The MIT Press, Cambridge, Massachusetts (USA), 1992.

[15] H. Blum. Biological shape and visual science (part 1). *Journal of Theoretical Biology*, 38:205–287, 1973.

[16] Alison Bosson. *Experiments with scale-space vision systems*. PhD thesis, University of East Anglia (UEA), Norwich, UK, 1999. In progress.

[17] C.R. Brice and C.L. Fennema. Scene analysis using regions. *Artificial Intelligence*, 1(3):205–226, 1970.

[18] F. Cheevasuvit, H. Maitre, and D. Vidal-Madjar. A robust method for picture segmentation based on split-and-merge procedure. *Computer Vision, Graphics and Image Processing*, 34:268–281, 1986.

[19] S-Y. Chen, W-C. Lin, and C-T. Chen. Split-and-merge image segmentation based on localized feature analysis and statistical tests. *Graphical Models and Image Processing*, 53(5):457–475, 1991.

[20] N. Chin and S. Feiner. Near real-time shadow generation using bsp trees. In *ACM siggraph, Computer Graphics*, volume 23 of *3*, pages 99–106, 1989.

[21] Jacek Cichosz and Fernand Meyer. Morphological multiscale image segmentation. In *Proceedings of Workshop on Image Analysis for Multimedia*

Interactive Services, pages 161–166, Louvain-La-Neuve, Belgium, June 1997.

[22] T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active appearance models. In Hans Burkhardt and Bernd Neumann, editors, *Proceedings of European Conference on Computer Vision*, volume 2, pages 484–498, Freiburg, Germany, June 1998.

[23] T. F. Cootes and C. J. Taylor. Modelling object appearance using the grey-level surface. In E. Hancock, editor, *Proceedings of British Machine Vision Conference*, volume 1, pages 479–488, York, U.K., September 1994.

[24] José Crespo and Victor Maojo. New results on the theory of morphological filters by reconstruction. *Pattern Recognition*, 31(4):419–429, 1998.

[25] José Crespo, Ronald W. Schafer, Jean Serra, Cristophe Gratin, and Fernand Meyer. The flat zone approach: A general low-level region merging segmentation method. *Signal Processing*, 62:37–60, 1997.

[26] José Crespo and R.W. Schafer. Locality and adjacency stability constraints for morphological connected operators. *Journal of Mathematical Imaging and Vision*, 7(1):85–102, 1997.

[27] José Crespo and Jean Serra. Morphological pyramids for image coding. In *Proceedings of Visual Communication and Image Processing*, volume 2094, pages 159–170, Cambridge, UK, November 1993.

[28] José Crespo, Jean Serra, and R.W. Schafer. Theoretical aspects of morphological filters by reconstruction. In *Signal Processing*, volume 47, pages 157–180, 1995.

[29] L. S. Davis. A survey of edge detection techniques. *Computer Graphics and Image Processing*, 4:248–270, 1975.

[30] H. Digabel and C. Lantuéjoul. Iterative algorithms. In *Proceedings of the 2nd European Symposium on Quantitative Analysis of Microstructures in Material Science, Biology and Medicine*, October 1977.

[31] B. P. Dobrin, T. Viero, and M. Gabbouj. Fast watershed algorithms: analysis and extension. In *Proceedings of SPIE*, volume 2180, pages 209–220, 1994.

[32] Bryan Flamig. *Practical Data Structures in C++*. John Wiley and Sons, 1993.

[33] L. M. J. Florack, A. H. Salden, B. M. terr Harr Romeny, J. J. Koenderink, and M. A. Viergever. Non-linear scale-space. *Image and Vision Computing*, 13(4):279–294, 1994.

[34] L. M. J. Florack, B. M. terr Harr Romeny, J. J. Koenderink, and M. A. Viergever. Linear scale-space. *Journal of Mathematical Imaging and Vision*, 4(4):325–351, 1994.

[35] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics. Principles and practice.* Addison–Wesley, 1990.

[36] Luis Garrido, Albert Oliveras, and Philippe Salembier. Motion analysis of image sequences using connected operators. In *Proceedings of Visual Communication and Image Processing*, volume 3024, pages 546–557, San Jose (CA), USA, February 1997.

[37] Luis Garrido, Philippe Salembier, and D. Garcia. Extensive operators in partition lattices for image sequence analysis. *Signal Processing*, 66(2):157–180, April 1998.

[38] Donald Geman, Stuart Geman, Christine Graffigne, and Ping Dong. Boundary detection by constrained optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:609–628, 1990.

[39] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Addison-Wesley, 1992.

[40] MPEG Requirements Group. Applications for MPEG-7. Doc. ISO/MPEG N2328, July 1998.

[41] MPEG Requirements Group. MPEG-7 context and objectives. Doc. ISO/MPEG N2326, July 1998.

[42] MPEG Requirements Group. MPEG-7 requirements document. Doc. ISO/MPEG N2327, July 1998.

[43] Srinivas Gutta, Ibrahim F. Imam, and Harry Wechsler. Hand gesture recognition using ensembles of radial basis function (RBF) networks and decision trees. *International Journal of Pattern Recognition and Artificial Intelligence*, 11(6):845–872, 1997.

[44] R.M. Haralick and L.G. Shapiro. Image segmentation techniques. *Computer Vision, Graphics and Image Processing*, 29(1):100–132, January 1985.

[45] Robert M. Haralick, Philip L. Katz, and Edward R. Dougherty. Model-based morphology: the opening spectrum. *Graphical Models and Image Processing*, 57(1):1–12, January 1995.

[46] Robert M. Haralick, Stanley R. Sternberg, and Xinhua Zhuang. Image analysis using mathematical morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(4):532–550, July 1987.

[47] F. Harary. *Graph Theory*. Addison-Wesley, Reading, MA, 1969.

[48] Richard Harvey, Kimberly Moravec, and J. Andrew Bangham. Stereo vision via connected-set operators. In *Proceedings of European Signal Processing Conference*, volume 2, pages 613–616, 1998.

[49] H. J. A. M. Heijmans, P. Nacken, A. Toet, and Luc Vincent. Graph morphology. *Journal of Visual Computing and Image Representation*, 3(1):24–38, March 1992.

[50] Ellis Horowitz, Sartaj Sahno, and Susan Anderson-Freed. *Fundamentals of Data Structures in C*. Computer Science Press, 20 Beaumont Street, Oxford OX1 2NQ, UK, 1993.

[51] S. L. Horowitz and T. Pavlidis. Picture segmentation by a directed split and merge procedure. In *Proceedings of International Conference on Pattern Recognition*, pages 424–433, 1974.

[52] G. M. Hunter. *Efficient computation and data structures for graphics*. PhD thesis, Departament of Electrical Engineering and Computer Science, Princeton University, 1978.

[53] Anil K. Jain. Advances in mathematical models for image processing. *Proceedings of IEEE*, 69:502–528, March 1981.

[54] Anil K. Jain. *Fundamentals of digital image processing*. Prentice Hall, Englewood Cliffs, New Jersey 07632, USA, 1989.

[55] B. B. Kimia, A. Tannenbaum, and S. W. Zucker. Shape, shocks, and deformations I: the components of two-dimensional shape and the reaction-diffusion space. *International Journal of Computer Vision*, 15:189–224, 1995.

[56] J. C. Klein. *Conception et réalisation d'une unité logique pour l'analyse quantitative d'images*. PhD thesis, Nancy University, France, 1976.

[57] Reinhard Klette and Piero Zamperoni. *Handbook of Image Processing Operators*. John Wiley and Sons Ltd., Baffins Lane, Chichester, West Sussex PO19 1UD, England, 1994.

[58] A. Klinger. Pattern and search statistics. In J.S. Rustagi, editor, *Optimazing methods in statistics*, New York, 1971.

[59] M. Kliot and E. Rivlin. Invariant-based shape retrieval in pictorial databases. In Hans Burkhardt and Bern Neumann, editors, *Proceedings of European Conference on Computer Vision*, volume 1, pages 491–507, Freiburg, Germany, June 1998.

[60] Donald Ervin Knuth. *The art of computer programming: fundamentals algorithms*, volume 1. Addison Wesley Longman, 3 edition, May 1997.

[61] J. J. Koenderink. The structure of images. *Biological Cybernetics*, 50:363–370, 1984.

[62] R. R. Kohler. A segmentation system based on thresholding. *Computer Graphics and Image Processing*, 15(4):319–338, April 1981.

[63] W. L. Koontz, P. M. Narendra, and K. Fukunaga. A graph-theoretic approach to nonparametric cluster analysis. *IEEE Transactions on Communications*, 25:936–944, September 1976.

[64] C. Lantuejoul and F. Maisonneuve. Geodesic methods in image analysis. *Pattern Recognition*, 12(2):117–187, 1984.

[65] Lawrence M. Lifshitz and Stephen M. Pizer. A multiresolution hierarchical approach to image segmentation based on intensity extrema. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6):529–540, 1990.

[66] Tony Lindeberg. *Scale-Space Theory in Computer Vision*. Kluwer Academic Publishers, 1994.

[67] M. Maimone and S. Shafer. The CMU calibrated imaging stereo datasets. http://www.cs.cmu.edu:8001/usr0/anon/project/cil/html/cil-ster.html.

[68] Petros Maragos and Ronald W. Schafer. Morphological filters - part II: Their relations to median, order-statistic, and stack filters. *IEEE Trans. Acoustics, Speech, and Signal Processing*, 35(8):1170–1184, August 1987.

[69] G. Matheron. *Random Sets and Integral Geometry*. Wiley, New York, 1975.

[70] Michael P. McLoughlin and Gonzalo R. Arce. Deterministic properties of the recursive separable median filter. *IEEE Trans. Acoustics, Speech, and Signal Processing*, 35(1):98–106, January 1987.

[71] A. Mehnert and P. Jackway. An improved seeded region growing algorithm. *Pattern Recognition Letters*, 18(10):1065–1071, October 1997.

[72] Fernand Meyer and S. Beucher. Morphological segmentation. *Journal of Visual Computing and Image Representation*, 1(1):21–45, 1990.

[73] P. M. Narendra and M. Goldberg. Image segmentation with directed trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(2):185–190, March 1980.

[74] Thomas A. Nodes and Neal C. Gallagher. Two-dimensional root structures and convergence properties of the separable median filter. *IEEE Trans. Acoustics, Speech, and Signal Processing*, 31(6):1350–1365, December 1983.

[75] Albert Oliveras and Philippe Salembier. Generalized connected operators. In *Proceedings of Visual Communication and Image Processing*, volume 2727, pages 761–773, Orlando(FL),USA, March 1996.

[76] Albert Oliveras, Philippe Salembier, and Luis Garrido. Stereo image analysis using connected operators. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, pages 3169–3172, Munich, Germany, April 1997.

[77] T. Pavlidis and Y-T. Liow. Integrating region growing and edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:225–233, 1990.

[78] Marcello Pelillo, Kaleem Siddiqi, and Steven W. Zucker. Matching hierarchical structures using association graphs. In Hans Burkhart and Bernd Neumann, editors, *Proceedings of European Conference on Computer Vision*, volume 2, pages 3–16, Freiburg, Germany, June 1998.

[79] F. K. Potjer. Region adjacency graphs and connected morphological operators. In P. Maragos, R. W. Schafer, and M. A. Butt, editors, *Proceedings of Internacional Symposium on Mathematical Morphology*, pages 111–118, Atlanta (GA), USA, May 1996.

[80] D. R. Reddy and S. Rubin. Representation of three-dimensional objects. Technical report, Department of Computer Science, Carnegie-Mellon University, 1978. pages 78-113.

[81] A. Rosenfeld and L.S. Davis. Image segmentation and image models. *Proceedings of IEEE*, 67(5):764–772, May 1979.

[82] Javier Ruiz Hidalgo. Using max-trees and level connected sets to interpret stereo and moving images. 3rd year project, University of East Anglia, Norwich, NR47TJ, UK, June 1997.

[83] P. K. Sahoo, S. Soltani, A. K. C. Wong, and Y. C. Chen. A survey of thresholding techniques. *Computer Vision, Graphics and Image Processing*, 41(2):233–260, February 1988.

[84] Philippe Salembier and Luis Garrido. Binary partition tree as an efficient representation for filtering, segmentation and information retrieval. In *Proceedings of International Conference on Image Processing*, pages 4–7, Chicago (IL), USA, October 1998.

[85] Philippe Salembier and M. Kunt. Size sensitive multiresolution decomposition of images with rank order based filters. *Signal Processing*, 27:205–241, 1992.

[86] Philippe Salembier, F. Marqués, M. Pardàs, R. Morros, I. Corset, S. Jeannin, L. Bouchard, Fernand Meyer, and B. Marcotegui. Segmentation-based video coding system allowing the manipulation of objects. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(1):60–74, February 1997.

[87] Philippe Salembier, Albert Oliveras, and Luis Garrido. Motion connected operators for image sequences. In *Proceedings of European Signal Processing Conference*, volume 2, pages 1083–1086, Triestre, Italy, September 1996.

[88] Philippe Salembier, Albert Oliveras, and Luis Garrido. Antiextensive connected operators for image and sequence processing. *IEEE Transactions on Image Processing*, 7(4):555–570, April 1998.

[89] Philippe Salembier and H. Sanson. Robust motion estimation using connected operators. In *Proceedings of International Conference on Image Processing*, volume 1, pages 77–80, Santa Barbara (CA), USA, October 1997.

[90] Philippe Salembier and Jean Serra. Morphological multiscale decomposition of images with rank order based filters. In *Proceedings of Visual Communication and Image Processing*, volume 1818, pages 620–631, Boston (MA), USA, November 1992.

[91] Philippe Salembier and Jean Serra. Flat zones filtering, connected operators and filters by reconstruction. *IEEE Transactions on Image Processing*, 3(8):1153–1160, August 1995.

[92] Jean Serra. *Image analysis and mathematical morphology*, volume 1. Academic Press, London, 1982.

[93] Jean Serra. Introduction to mathematical morphology. *Computer Vision, Graphics and Image Processing*, 535:283–305, 1986.

[94] Jean Serra. *Image analysis and mathematical morphology: Theoretical Advances*, volume 2. Academic Press, London, 1988.

[95] Jean Serra and Philippe Salembier. Connected operators and pyramids. In *Proceedings of Image Algebra and Mathematical Morphology*, volume 2030, pages 65–76, San Diego (CA), USA, July 1993.

[96] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. In *Proceedings of Computer Vision and Pattern Recognition*, pages 731–737, San Juan, Puerto Rico, June 1997.

[97] Jianbo Shi and Jitendra Malik. Self inducing relational distance and its application to image segmentation. In *Proceedings of European Conference on Computer Vision*, volume 1, pages 528–543, Friburg, Germany, June 1998.

[98] K. Siddiqi, A. Shokoufandeh, S. J. Dickinson, and S. W. Zucker. Shock graphs and shape matching. In *Proceedings of International Conference on Computer Vision*, pages 222–229, Bombay, India, 1998.

[99] Stanley R. Sternberg. Grayscale morphology. *Computer Vision, Graphics and Image Processing*, 35:333–355, 1986.

[100] Luc Vincent. Graphs and mathematical morphology. *Signal Processing*, 16:365–388, 1989.

[101] Luc Vincent. Grayscale area openings and closings, their efficent implementation and applications. In Jean Serra and Philippe Salembier, editors, *Proceedings of International Workshop on Mathematical Morphology and its applications to Signal Processing*, pages 22–27, May 1993.

[102] Luc Vincent. Morphological grayscale reconstruction in image analysis: applications and efficient algorithms. *IEEE Transactions on Image Processing*, 2(2):176–201, April 1993.

[103] Luc Vincent and P. Soille. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:583–598, 1991.

[104] A. P. Witkin. Scale-space filtering. In *Proceedings of Internacional Joint Conference on Artificial Intelligence*, pages 1019–1022, 1983.

[105] Derick Wood. *Data Structures, Algorithms, and Performance*. Addison-Wesley, 1993.

[106] S.C. Zhu and A. Yuille. Region competition: Unifying snakes, region growing, and bayes/mdl for multi-band image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(9):884–900, September 1996.

[107] S. W. Zucker. Region growing: childhood and adolescence. *Computer Graphics and Image Processing*, 5:382–399, 1976.