Construction of Combinatorial Pyramids

Luc Brun[†] and Walter Kropatsch[‡]

† Laboratoire d'Études et de Recherche en Informatique(EA 2618)
Université de Reims - France
and
‡ Institute for Computer-aided Automation
Pattern Recognition and Image Processing Group
Vienna Univ. of Technology- Austria

† `luc.brun@univ-reims.fr`, ‡ `krw@prip.tuwien.ac.at`

Abstract. Irregular pyramids are made of a stack of successively reduced graphs embedded in the plane. Each vertex of a reduced graph corresponds to a connected set of vertices in the level below. One connected set of vertices reduced into a single vertex at the above level is called the reduction window of this vertex. In the same way, a connected set of vertices in the base level graph reduced to a single vertex at a given level is called the receptive field of this vertex. The graphs used in the pyramid may be region adjacency graphs, dual graphs or combinatorial maps. This last type of pyramids are called Combinatorial Pyramids. Compared to usual graph data structures, combinatorial maps encode one graph and its dual within a same formalism and offer an explicit encoding of the orientation of edges around vertices. This paper describes the construction scheme of a Combinatorial Pyramid. We also provide a constructive definition of the notions of reduction windows and receptive fields within the Combinatorial Pyramid framework.

1 Introduction

Graphs play an important role in different fields of computer vision and pattern recognition. However, many graph algorithms still suffer from a high computational complexity. Irregular pyramids allow coarse to fine strategies by encoding a hierarchy of graphs. The efficiency of an irregular data structure is strongly influenced by two closely related features: The decimation process used to build one graph from the graph below and the data structure used to encode each graph of the pyramid. The decimation process determines the height of the pyramid and the properties that may arise from the decimation process. The model used to encode the graphs of the pyramid determines the properties that may be encoded in each graph.

We present in this paper an implementation of Combinatorial Pyramids [1]. Such pyramids are defined as a stack of successively reduced combinatorial maps. A combinatorial map data structure may be intuitively understood as an encoding of one planar graph and its dual into a same data structure. Combinatorial maps encode additionally the orientation of edges around each vertex.

The remaining of this paper is organized as follows: We present in section 2 the decimation process used within the irregular pyramid framework and two graph encodings. The combinatorial map model is presented in Section 3. The construction scheme of a Combinatorial Pyramid is then presented in Section 4. We also describe in this section the notions of reduction window and receptive fields within the Combinatorial Pyramid framework. This section concludes with an application of Combinatorial Pyramids to the segmentation framework.

2 Irregular Pyramids

The irregular pyramids are defined as a stack of successively reduced graphs, each graph being built from the graph below by selecting a set of vertices named surviving vertices and mapping each non-surviving vertex to a surviving one [8]. This reduction operation was first introduced by Meer [7, 8] as a stochastic process. Using such framework, the graph $G_{l+1} = (V_{l+1}, E_{l+1})$ defined at level $l + 1$ is deduced from the graph defined at level l by the following steps:

1. The selection of the vertices of G_{l+1} among V_l. These vertices are the surviving vertices of the decimation process.
2. A link of each non surviving vertex to a surviving one. This step defines a partition of V_l.
3. A definition of the adjacency relationships between the vertices of G_{l+1} in order to define E_{l+1}.

2.1 Simple graph pyramids

In order to obtain a fixed decimation ratio between each level, Meer [7] imposes the following constraints on the set of surviving vertices:

$$\forall v \in V_l - V_{l+1}\ \exists v' \in V_{l+1} :\ (v, v') \in E_l \tag{1}$$

$$\forall (v, v') \in V_{l+1}^2 :\ (v, v') \notin E_l \tag{2}$$

Constraint (1) insures that each non-surviving vertex is adjacent to at least a surviving one. Constraint (2) insures that two adjacent vertices cannot both survive. These constraints define a *maximal independent set* (MIS).

Meer [7] defines a set of surviving vertices fulfilling the maximal independent set requirements thanks to a stochastic process: A variable x_i is associated to each vertex v_i and each vertex associated to a local maximum of this variable survives. Since two adjacent vertices can not both represent a local maximum of the variable, the constraint (2) is satisfied. However, some vertices may not represent a local maximum of the random variable while having only non surviving neighbors. The constraint (1) is in this case violated. The determination of the MIS is thus performed thanks to an iterative process [8, 6]. Jolion [5] and Haxhimusa et al. [4] have recently proposed two improvements of this iterative decimation process.

Meer [7] uses as variable x_i a random variable uniformly distributed between $[0, 1]$. This purely random process has been adapted by Montanvert [8] to the connected component analysis framework by restricting the decimation process to a set of subgraphs of the graph G_l. This restriction of the decimation process is performed by a function $\lambda(v, v')$ defined on each couple of adjacent vertices v and v' and such that $\lambda(v, v') = 1$ if v and v' belong to the same subgraph and 0 otherwise. Jolion [6] proposed to use an interest operator such as the variance computed on a neighborhood of each vertex rather the random variable. Using such a reduction scheme, surviving vertices are located on homogeneous parts of the image.

Given the set of surviving vertices, Meer and Montanvert [7, 8] connect each non surviving vertex to its surviving neighbor whose random variable is maximum. Jolion [6] uses a contrast operator such as the difference of gray levels to connect each non surviving vertex to its surviving neighbor with the closest gray level.

Note that within the segmentation framework, the use of the function λ defined by Montanvert [8] supposes to define first an implicit partitioning of the current graph in order to set the value of $\lambda(v, v')$ for each couple of vertices. Using the interest and contrast operators defined by Jolion the partition of the graph is built during the decimation process. The set of non surviving vertices connected to a surviving vertex defines its reduction window of the surviving vertex and thus the parent child relationship between two consecutive levels.

The final set of surviving vertices defined on V_l corresponds to the set of vertices V_{l+1} of the reduced graph $G_{l+1} = (V_{l+1}, E_{l+1})$. The set of edges E_{l+1} of G_{l+1} is defined by connecting by an edge in G_{l+1} any couple of surviving vertices having adjacent children.

2.2 Dual Graph pyramids

Using the reduction scheme defined in Section 2.1, two surviving vertices in G_l are connected in G_{l+1} if they have at least two adjacent children. Two reduction windows adjacent by more than a couple of children will thus be connected by a single edge in the reduced graph. The stack of graphs produced by the above decimation process is thus a stack of simple graphs (i.e. graphs without self-loops nor double edges). Such graphs does not allow to encode multiple boundaries between regions nor inclusions relationships. This drawback may be overcome by using the dual graph pyramids introduced by Willersin and Kropatsch [10]. Within the dual graph pyramid framework the reduction process is performed by a set of edge contractions. The edge contraction operation collapses into one vertex two vertices adjacent by an edge and remove the edge. In order to keep the number of connected components of the initial graph, the set of edges to be contracted in a graph G_l of the pyramid must forms a forest of G_l. The resulting set of edges is called a contraction kernel. Using the decimation process described in Section 2.1, each non surviving vertex v is adjacent to its father v'. If we select one edge between v and v' for each non surviving vertex v, the resulting set of darts K forms a forest of the initial graph composed of trees of depth 1. This set encodes thus a contraction kernel.

The contraction of a graph reduces the number of vertices while maintaining the connections to other vertices. As a consequence some redundant edges such as self-loops or double edges may occur, some encode relevant topological relations (e.g. an island in a lake) others can be removed without any harm to the involved topology. These redundant edges may be characterized in the dual of the contracted graph. The removal of such edges is called a dual decimation step and the set of removed edges is called a removal kernel.

Using the dual graph pyramid framework each receptive field is a connected set of vertices in the base level. Moreover, each edge between two vertices encodes an unique connected boundary between the associated receptive fields. Finally, the use of self-loops within the hierarchy allows to differentiate adjacency relationships between receptive fields from inclusion relations.

3 Combinatorial maps

A combinatorial map may be seen as a planar graph $G = (V, E)$ encoding explicitly the orientation of edges around a given vertex. First the edges of E are split into two half edges called *darts*, each dart having its origin at the vertex it is attached to. The fact that two half-edges (darts) stem from the same edge is recorded in the reverse permutation α. A second permutation σ encodes the set of darts encountered when turning counterclockwise around a vertex.

A combinatorial map is thus defined by a triplet $G = (\mathcal{D}, \sigma, \alpha)$, where $\mathcal{D}$ is the set of darts and σ, α are two permutations defined on $\mathcal{D}$ such that α is an involution:

$$\forall d \in \mathcal{D} \quad \alpha^2(d) = d \tag{3}$$

Given a dart d and a permutation π, the π-orbit of d denoted by $\pi^*(d)$ is the series of darts $(\pi^i(d))_{i \in \mathbb{N}}$ defined by the successive applications of π on the dart d. The σ and α orbits of a dart d will be respectively denoted by $\sigma^*(d)$ and $\alpha^*(d)$.

Each vertex of a combinatorial map $G = (\mathcal{D}, \sigma, \alpha)$ is implicitly encoded by its σ orbit. This implicit encoding may be transformed into an explicit one by using a vertex's labeling function [1]. Such a function, denoted by μ associates to each dart d a label which identifies its σ-orbit $\sigma^*(d)$. More precisely a vertex's labeling function should satisfy:

$$\forall (d, d')^2 \in \mathcal{D} \; \mu(d) = \mu(d') \Leftrightarrow \sigma^*(d) = \sigma^*(d')$$

For example, if darts are encoded by integers, the function $\mu(d) = min_{d' \in \sigma^*(d)}\{d'\}$ defines a valid vertex's labeling function.

Given a combinatorial map $G = (\mathcal{D}, \sigma, \alpha)$, its dual is defined by $\overline{G} = (\mathcal{D}, \varphi, \alpha)$ with $\varphi = \sigma \circ \alpha$. The orbits of the permutation φ encode the set of darts encountered when turning around a face of G More details about combinatorial maps may be found in [2, 1].

4 Combinatorial Pyramids

As in the dual graph pyramid scheme [10] (Section 2), a combinatorial pyramid is defined by an initial combinatorial map successively reduced by a sequence of contraction or removal operations respectively encoded by contraction and removal kernels.

4.1 Constructing Contraction Kernels using Union-Find

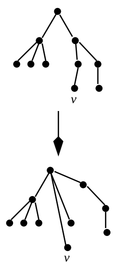
As mentioned in Section 2, Jolion [6] and Montanvert [8] have proposed two different methods to decimate a graph. These methods may be adapted to build a contraction kernel by selecting one edge between each non surviving vertex and its father(Section 2.2).

We have encoded the decimation process defined by Jolion [6]. Our implementation uses conjointly the interest and contrast operators defined by Jolion, with the function λ defined by Montanvert. This last function is used to forbid the attachment of some non surviving vertices to surviving ones. A similar result may be obtained by setting the contrast operator to an infinite value for such couple of vertices.

The method of Montanvert [8] is based on an implicit partitioning of the graph. Such a partitioning may be obtained by using, for example, a clustering algorithm on the value attached to each vertex. Within this context the design of a contraction kernel is equivalent to the design to a spanning forest of the graph. This problem may be solved efficiently using union and find operations [3] on sequential or on parallel machines with few processors using.

Within this technique each tree of a contraction kernel is encoded by storing in each vertex a reference to its father. The father of each vertex is initialized by $father(v) = v$. The find operation applied on a vertex v determines the root of the tree containing v. The union operation merge two trees into one. Given two vertices v_1 and v_2 whose father must be merged ($\lambda(father(v_1), father(v_2)) = 1$) the union of both trees is performed by setting the “grand-father”' of one of the two vertices, say v_1 to the father of v_2. We have thus $father(father(v_1)) = father(v_2)$. Using such a construction scheme for contraction kernels, the time complexity of the algorithm is dominated by the number of indirections required to find the father of each vertex. Indeed, if one vertex v_1 is merged to v_2 and then v_2 to v_3, we have $father(v_1) = v_2$ and $father(v_2) = v_3$. The determination of the father of v_1 requires thus 2 indirections. These indirections may be suppressed by using the algorithm `find_compress` (Algorithm 1) which connects directly to the root all the nodes in the branch of the tree which connect one vertex to its root.

The construction of the contraction kernel is performed by Algorithm 2. The test performed on line 12 of Algorithm 2 merges two trees, if their associated roots are different and if the regions encoded by these trees should be merged according to the function λ. The function μ used on lines 10 and 11 encodes the vertices and is defined in Section 3.



```
find_compress(array **father**,vertex $v$)
{
    if (**father**($v$) $==v$ )
        return $v$;
    else
    {
        **father**($v$) = find_compress(**father**,**father**($v$))
        return **father**($v$)
    }
}
```

Algorithm 1: The `find_compress` algorithm.

execution times	Union & Find	Iterative Decimation
64×64	0.01 s	0.39 s
128×128	0.04 s	3.13 s
256×256	0.19 s	24.65 s
512×512	0.78 s	3 min 14s
1024×1024	3.5 s	25 min 33 s

Table 1. Execution times required by the iterative decimation process and the one based on union and find operations.

Let us consider a combinatorial map $G_l = (\mathcal{SD}_l, \sigma_l, \alpha)$ reduced to $G_{l+1} = (\mathcal{SD}_{l+1} = \mathcal{SD}_l - K_{l+1}, \sigma_{l+1}, \alpha)$ by K_{l+1}. Since an union operation is performed for each edge added to the contraction kernel K_{l+1}, the number of union performed by the algorithm is equal to $|K_{l+1}|$. In the same way, we perform two `find_compress` operations for each dart, the total number of `find_compress` operations performed by the algorithm is thus equal to $2|\mathcal{SD}_l|$. Based on the previous considerations it can be shown [9] that the sequential complexity of Algorithm 2 is equal to $\mathcal{O}(2\beta(2|\mathcal{SD}_l|, |K|)|\mathcal{SD}_l|))$ where β is a very low growing function.

Tables 1 compares the execution times on a sequential machine of the decimation processes using the iterative definition of survivors (Section 2) and the union and find operations. Both decimation processes reduce a $2^n \times 2^n$ image to a single vertex. Note that the number $|\mathcal{SD}_l|$ of initial darts is, in this case a linear function of the size 2^{2n} of the input image. This size is multiplied by 4 between each line of Table 1. The execution time of the Union & Find algorithm is also multiplied by 4 between each line. The complexity of the Union & Find algorithm is thus found to be linear according to the size of the image on this experiment. On the other hand, the execution times of the iterative decimation process is approximately multiplied by 8 between each line. The complexity of this algorithm is thus not linear on this experiment.

```
define_kernel(pyramide P, function lambda)
{
    contraction kernel $K = \emptyset$
    top of the pyramid $G_l = (\mathcal{SD}_l, \sigma_l, \alpha)$

    For any vertex $v$ in $G_l$
            $\mathbf{father}(v) = v$
    For any dart $b \in \boldsymbol{\mathcal{D}}_l$
    {
        $f_1$= find_compress($\mathbf{father}, \mu(b)$)
        $f_2$= find_compress($\mathbf{father}, \mu(\alpha(b))$)
        if ($f_1$ != $f_2$ and lambda($f_1, f_2$))
        {
            $K = K \cup \{\alpha^*(b)\}$
            $\mathbf{father}(f_1) = f_2$
        }
    }
    return $K$
}
```

Algorithm 2: *Computation of a contraction kernel*

4.2 Removals Kernels

Given a combinatorial map $G = (\boldsymbol{\mathcal{D}}, \sigma, \alpha)$ and a contraction kernel K, the contracted combinatorial map $G' = (\mathcal{SD} = \boldsymbol{\mathcal{D}} - K, \sigma', \alpha)$ may contain redundant edges corresponding to double edges or empty-self-loops (Section 2.2). An empty self loop $\alpha^*(d)$ is characterized [1] in G' by $\varphi'(d) = d$. Note that the removal of an edge $\alpha^*(d)$ corresponding to and empty self loop may create a new empty self-loop $\alpha^*(\sigma(d))$ if $\alpha^*(d)$ and $\alpha^*(\sigma(d))$ define nested loops. The removal operation is thus performed by an iterative process which first removes the empty self-loop $\alpha^*(d)$ characterized by $\varphi'(d) = d$ and then removes each dart $\sigma^n(d)$ which become an empty self loop after the removal of $\sigma^{n-1}(d)$. In the same way, double edges are characterized [1] in G' by $\varphi'^2(\sigma(d)) = \alpha(d)$. Such darts are also removed iteratively by checking the double edge condition for the sequence of σ successors of the dart d defining the first double edge.

4.3 Computing the reduced combinatorial maps

The creation of the reduced combinatorial map from a contraction or a removal kernel is performed in parallel by using dart's reduction window [1]. Given a combinatorial map $G = (\boldsymbol{\mathcal{D}}, \sigma, \alpha)$, a kernel K and a surviving dart $d \in \mathcal{SD} = \boldsymbol{\mathcal{D}} - K$, the reduction window of d is either equal to:

$$RW(d) = d, \sigma(d), \dots, \sigma^{n-1}(d)$$

with $n = Min\{p \in \mathbb{N}^* \mid \sigma^p(d) \in \mathcal{SD}\}$ if K is a removal kernel or

$$RW(d) = d, \varphi(\alpha(d)), \dots, \varphi^{n-1}(\alpha(d))$$

with $n = Min\{p \in \mathbb{N}^* \mid \varphi^p(\alpha(d)) \in \mathcal{SD}\}$, if K is a contraction kernel.

Given a kernel K and a surviving dart $d \in \mathcal{SD}$, such that $RW(d) = d.d_1 \ldots d_p$, the successor of d within the reduced combinatorial map $G' = G/K = (\mathcal{SD}, \sigma', \alpha)$ is retrieved from $RW(d) = d.d_1 \ldots .d_p$ by [1]:

$$\sigma'(d) = \begin{cases} \sigma(d_p) \text{ if } K \text{ is a removal kernel} \\ \varphi(d_p) \text{ if } K \text{ is a contraction kernel} \end{cases} \quad (4)$$

Note that the reduction window of a surviving dart d connects $d \in G'$ to a sequence of darts in the initial combinatorial map G. The notion of dart's reduction window connects thus two successive levels of the pyramid and corresponds to the usual notion of reduction window [8].

Within the combinatorial pyramid framework each vertex defined at level i is encoded by its σ_i-orbit. The reduction window of such a vertex is defined as the concatenation of the darts belonging to its σ_i-orbit:

$$RW_i(\sigma_i^*(d)) = \bigodot_{j=1}^{n} RW_i(d_j)$$

with $\sigma_i^*(d) = (d_1, \ldots, d_p)$. The symbol $\bigodot$ denotes the concatenation operator.

4.4 Receptive Fields

Dart's reduction windows allow us to reduce a combinatorial map using either contraction or removal kernels. Starting from an initial combinatorial map G_0 and given a sequence of kernels $K_1, \ldots, K_n$ we can thus build the sequence of reduced combinatorial maps $G_0, G_1, \ldots, G_n$.

The transitive closure of the father-child relationship defined by the dart's reduction window corresponds to the notion of dart's receptive field. A dart's receptive field connects one dart in a combinatorial map $G_i = (\mathcal{SD}_i, \sigma_i, \alpha)$ to a sequence of darts defined in the base level combinatorial map $G_0 = (\boldsymbol{\mathcal{D}}, \sigma, \alpha)$. These sequences are defined [1] recursively by $RF_0(d) = d$ for any $d \in \boldsymbol{\mathcal{D}}$ and:

$$\forall i \in \{1, \ldots, n\}, \forall d \in \mathcal{SD}_i$$
$$RF_i(d) = \begin{cases} \bigodot_{j=1}^{p} RF_{i-1}(d_j) & \text{if } K_i \text{ is a removal kernel} \\ \bigodot_{j=1}^{p} d_j RF_{i-1}^*(\alpha(d_j)) & \text{if } K_i \text{ is a contraction kernel} \end{cases} \quad (5)$$

with $RW_i(d) = d_1 \ldots, d_p$. The symbol $RF_{i-1}^*(\alpha(d_j))$, $j \in \{1, \ldots, p\}$ denotes the sequence $RF_{i-1}(\alpha(d_j))$ without its first dart.

The receptive field of a dart encodes its embedding in the base level combinatorial map. If this initial combinatorial map is associated to a planar sampling grid, each initial vertex corresponds to one pixel and a vertex defined at level i of the pyramid encodes a connected set of vertices, i.e. a region. Within the combinatorial map framework the embedding of a vertex $\sigma_i^*(d)$ in the initial combinatorial map is called a vertex's receptive field and is defined using the same construction scheme than the vertex's reduction window. Given one dart $d \in \mathcal{SD}_i$, the receptive field of its associated vertex $\sigma_i^*(d) = (d_1, \ldots, d_p)$ is thus defined as the concatenation of the dart's receptive fields belonging to $\sigma_i^*(d)$:

$$R_{\sigma_i^*(d)} = \bigodot_{j=1}^{p} RF_i(d_j)$$

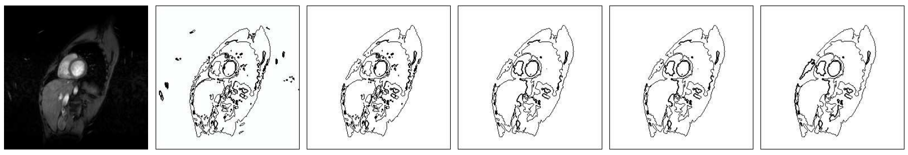

(a) original (b) level 1 (c)level 2 (d) level 3 (e) level 4 (f) level 5

Fig. 1. One Combinatorial Pyramid encoding a stack of partitions of a heart.

The receptive field of a vertex $\sigma_i^*(d)$ encodes the darts of the vertices reduced to the vertex $\sigma_i^*(d)$ at level i [1]. Using a labeling function μ in the base level combinatorial map, each label identifies a pixel in the associated image. This connected set of pixels may be retrieved by computing the set of reduced vertices: $\cup_{d' \in R_{\sigma_i^*(d)}} \mu(d')$.

4.5 Application to the segmentation framework

Fig. 1 shows an application of Combinatorial Pyramids to the segmentation framework. The original image (Fig. 1(a)) is quantized into K gray levels. The quantization algorithm used in this experiment [11] decomposes the initial range of image's gray levels into K intervals. This set of intervals is designed in order to minimize the sum of the K variances computed on each interval.

Each region in (Fig. 1(b)) encodes a connected component of the pixels whose gray value are mapped onto a same interval. The background of the image is determined by selecting the region with the greatest cardinal adjacent to the exterior of the image. Then, all regions included in the background whose size is lower than a given threshold T are merged with the background (Fig. 1(c)). The mean gray level of each region is then used to initialize a gray level histogram. The frequency $h(i)$ of one entry i of the histogram is set to the sum of region's cardinal whose mean gray level is equal to i. This histogram is then quantized into K values and we merge any couple of adjacent region whose mean gray levels are mapped onto a same interval . This process is iterated form Fig. 1(d) to Fig. 1(f) until no merge occurs. Note that, at each level of the pyramid, the quantization algorithm only provides a partition of the range of grays values. The encoding of the partition and the merge operations are performed using the combinatorial pyramid model. The partition which may be obtained by using only [11] is represented on Fig. 1(b). The segmentation results represented in Fig. 1 have been obtained with $K = 4$ and $T = 10$.

The mean gray level and the cardinal of each region are computed during the construction of the pyramid by updating the parameters of each surviving vertex from the ones of its reduction window. The borders of the partitions represented in Fig. 1 are deduced from the receptive fields (Section 4.4)

5 Conclusion

We have defined in this article the construction scheme of a Combinatorial Pyramid. Two methods to build a contraction kernel have been proposed: one based on the method initially defined by Meer, Montanvert and Jolion [7, 8, 6] and new one based on Union and Find operations. Our experiments show that this last method is more efficient than the first one when the contraction criteria may be determined before the construction of the contraction kernel. We have also presented the notions of Reduction Window and Receptive fields within the Combinatorial Pyramid framework. The receptive fields have been used in our experiments to retrieve the borders of the partitions.

References

1. L. Brun. *Traitement d'images couleur et pyramides combinatoires.* Habilitation à diriger des recherches, Université de Reims, 2002.
2. L. Brun and W. Kropatsch. Contraction kernels and combinatorial maps. In J. M. Jolion, W. Kropatsch, and M. Vento, editors, 3^{rd} *IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pages 12–21, Ischia Italy, May 2001. IAPR-TC15, CUEN.
3. C. Fiorio and J. Gustedt. Two linear time union-find strategies for image processing. Technical Report 375/1994, Technische Universitat Berlin, 1994.
4. Y. Haxhimusa, R. Glantz, M. Saib, G. Langs, and W. G. Kropatsch. Logarithmic Tapering Graph Pyramid. In L. Van Gool, editor, *Pattern Recognition, 24th DAGM Symposium*, volume 2449 of *Lecture Notes in Computer Science*, pages 117–124, Zurich, Switzerland, September 2002. Springer, Berlin Heidelberg.
5. J.-M. Jolion. Data driven decimation of graphs. In J.-M. Jolion, W. Kropatsch, and M. Vento, editors, *Proceedings of* 3^{rd} *IAPR-TC15 Workshop on Graph based Representation in Pattern Recognition*, pages 105–114, Ischia-Italy, May 2001.
6. J. M. Jolion and A. Montanvert. The adaptative pyramid: A framework for 2d image analysis. *Computer Vision, Graphics, and Image Processing*, 55(3):339–348, May 1992.
7. P. Meer. Stochastic image pyramids. *Computer Vision Graphics Image Processing*, 45:269–294, 1989.
8. A. Montanvert, P. Meer, and A. Rosenfeld. Hierarchical image analysis using irregular tessellations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):307–316, APRIL 1991.
9. R. E. Tarjan. Efficiency of a good but non linear set union algorithm. *J. Assoc. Comput. System Mach.*, 22:215–225, 1975.
10. D. Willersinn and W. G. Kropatsch. Dual graph contraction for irregular pyramids. In *International Conference on Pattern Recogntion D: Parallel Computing*, pages 251–256, Jerusalem, Israel, 1994. International Association for Pattern Recognition.
11. S. Wong, S. Wan, and P. Prusinkiewicz. Monochrome image quantization. In *Canadian conference on electrical and computer engineering*, pages 17–20, September 1989.