Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research Report
RR-91-22

Self-Adapting Structuring and Representation of Space

Andreas Dengel

September 1991

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
D-6750 Kaiserslautern
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern und Saarbrücken is a non-profit organization which was founded in 1988 by the shareholder companies ADV/Orga, AEG, IBM, Insiders, Fraunhofer Gesellschaft, GMD, Krupp-Atlas, Mannesmann-Kienzle, Siemens-Nixdorf, Philips and Siemens. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct *systems with technical knowledge and common sense* which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- Intelligent Engineering Systems
- Intelligent User Interfaces
- Intelligent Communication Networks
- Intelligent Cooperative Systems.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Prof. Dr. Gerhard Barth
Director

Self-Adapting Structuring and Representation of Spaces

Andreas Dengel

DFKI-RR-91-22

This report is an extented version of papers [De91a; De91b] that are published in the proceedings of the Fourth Int´l Symposium on Artificial Intelligence, Cancún, Mexico, and the proceedings of the Int´l Conference on Visual Communication and Image Processing (VCIP-91), Boston, MA, November 1991.

This work has been supported by a grant from The Federal Ministry for Research and Technology (FKZ ITW-9003 0).

Self-Adapting Structuring and Representation of Space

ANDREAS DENGEL

Author´s abstract

The objective of this report is to propose a syntactic formalism for space representation. Beside the well known advantages of hierarchical data structure, the underlying approach has the additional strength of self-adapting to a spatial structure at hand. The formalism is called puzzletree because its generation results in a number of blocks which in a certain order — like a puzzle — reconstruct the original space. The strength of the approach does not lie only in providing a compact representation of space (e.g. high compression), but also in attaining an ideal basis for further knowledge-based modeling and recognition of objects. The approach may be applied to any higher-dimensioned space (e.g. images, volumes). The report concentrates on the principles of puzzletrees by explaining the underlying heuristic for their generation with respect to 2D spaces, i.e. images, but also schemes their application to volume data. Furthermore, the paper outlines the use of puzzletrees to facilitate higher-level operations like image segmentation or object recognition. Finally, results are shown and a comparison to conventional region quadtrees is done.

CONTENTS:

1 INTRODUCTION

Over the last years, much research has been done to develop formalisms for representing spatial information. In the domain of image processing, most of the techniques have addressed the problem from either the database view or the perspective of computer vision. Aspects traditionally associated with databases focus on efficient methods for image compression and accessing techniques (pixels from an image, images from a database). Computer vision, in contrast, concentrates on image analysis, pattern matching, or object recognition problems. All of these aspects are of vital significance for domains like geographic information systems, computer graphics, computational geometry, medical imaging, or robotics. For both research fields, databases and computer vision, it is necessary to generate a representation derived from raster-digitized images. For compression techniques on the one hand, the aim is to reach satisfactory high compression factors rather than to represent human perceptible image structures. Formalisms for spatial knowledge representation on the other hand, should capture concepts for abstraction, be understood by people, and be easy modifiable [SX89]. Thereby, the type of operations to be performed on the data heavily influences the formalism ultimately chosen for a specific task.

In many applications of artificial intelligence, data structures for spatial representations serve as a basis for numerous operations. Various publications in this field offer a rich set of proposals for object shape representations, but most of them are limited to two-dimensional spatial information rather than to 3D description. Another weakness is the incompatibility to compressed image data and the stipulated expensiveness of operations on that data. Due to these facts and in order to allow further significant advances in image compression and analysis, there is a pressing need to adopt radical new approaches for representing spatial structure.

Typical image data has several contigious regions that correspond to a finite set of entities (image objects) which are significant for a certain problem space. They are described by adjacent nonzero data points having specific locations, several geometric characteristics, and specific arrangements within the image.

To describe such a spatial structure, we propose an approach which transforms digitized image data into a hierarchical object-oriented representation called *puzzletree*. Following the classification made by [Sa84], puzzletrees may be considered as region quadtrees [KD76]. They are based on recursive decomposition and thus are able to describe geometric structures in an image space. Puzzletrees have the property to consider the spatial structure of a particular image to direct their generation. Therefore, the homogeneity of image regions serves as a basis to decide where a decomposition has to take place. For that purpose, we have developed a heuristic for image space decomposition in puzzletrees allowing for a flexible and compact representation.

In this report a description of the fundamentals of puzzletrees, their generation and representation is given. Section 2 first outlines the starting point and basic intention of the approach and gives a brief overview of the main characteristics of quadtrees,

describing their strengths and weaknesses. In addition, alternatives for hierarchical spatial representation by following heuristics are discussed and examined with respect to operationality. In Section 3 the puzzletree and its principles are introduced. In Section 4 the general strategy for puzzletree generation is illustrated and explained, while Section 5 proposes the entire generation algorithm and underlying heuristic. In Section 6, the internal object-oriented representation of puzzletrees is presented. Finally, Section 7 shows some results as well as comparisons to conventional region quadtree and points out to future work.

2 STARTING POINT & BACKGROUND

Starting point for this approach has been the intention to develop a description formalism of images allowing for both, image analysis and efficient image storage. Another aim has been the capability to provide information about a given spatial structure. In other words, the decomposition should follow the human-like capability of recognizing dominant structural features in images, i.e. groups of objects, single objects, or object parts. Most of existing representation techniques may be split into two classes, depending on the application domains for which they are used.

For compression techniques, the goal is to reach satisfactory high compression factors rather than to represent perceptual image structures. Generally, spatial knowledge may either be represented proportional or analogical [BB82]. Proportional formalisms comprise statements that have Boolean values to express propositions like “A <is-right-of> B” (e.g. predicate calculus, semantic nets, or programming languages) [SX89]. Analogical formalisms are such ones that are strongly analogous to the objects they represent. These are geometric data structures such as hierarchical ones, or scale models for image space representations and volumetric 3D representations. In general, they allow to make measurements to derive spatial relations of or between objects, like extensions, object sizes, distances, etc.

Hierarchical data structures for image representation are based on successive decomposition of rectangular space into adjacent subrectangles of same height or width. The subrectangles are recursively divided in the same way. As soon as a rectangle contains data points that have same color, an appropriate color-label may be assigned.

In many domains of computer vision it is necessary to split an image space into smaller parts to allow for partial processing of interesting subsets of data. In this sense, hierarchical data structures are useful because they represent image data in cells of spatial grids having different granularity. Thereby, each of the cells is labeled with the color filling that part of space. Such grids representing space by regions or volumes up to some grain are also designated as *occupancy arrays* [McD87]. Hierarchical data structures are easy to implement and provide a compact representation that allows an authentic image reconstruction. The most studied hierarchical data structure is the quadtree.

2.1 Overview of Quadtrees

The term quadtree refers to a set of hierachical data structures which have the common characteristic of hierarchical decomposition of a problem space. Taking the characterization of [Sa84], quadtrees may be categorized according to the following criteria:

- the type of data they represent;
- the principle of decomposition in which they divide a problem space;
- the degree of decomposition (variable or not variable) - also called as resolution.

Currently, quadtrees are used for the representation of point data, regions, curves, surfaces, and volumes. Thereby, the decomposition of original space may be either in equal-sized parts (regular decomposition), e.g. polygons, or may be governed by the input data and therefore results in parts of arbitrary size. The degree of decomposition is fix or predefined, or is determined according to the properties of the input data. The main advantages of quadtrees may be summarized as follows (see also [Sa90]):

- conceptual clear and uniform representation of dimensions and size of a space and its parts;
- straightforward representation of geometric features like position, extension, size and spatial relationships (i.e. location, adjacency, arrangements and distance) of image space objects and subsets of them;
- easy conversion in and from raster images;
- variable resolution representation;
- provision of a focus on interesting subsets of data.

The class of quadtrees mostly considered is the so called *region quadtree*. It is based on a stepwise and recursive decomposition of an image array of size $2^n \times 2^n$ into four quadrants of size $2^{n-1} \times 2^{n-1}$ [Sa81a]. If a quadrant not captures image points which are entirely black or white, the image is quartered in the same fashion until this criterion is fulfilled. Figure 1 shows the example of an image array (a), the quadrants resulting from quadtree generation (b) and the corresponding quadtree (c).

The quadtree for the example of Figure 1 (c) consists of 53 nodes (40 terminals). The root node corresponds to the entire input image. The subblocks obtained by subdivision are represented by respective child nodes. For region quadtree generation, the image object of Figure 1 (a) has first to be mapped into the minimal square of size $2^n \times 2^n$ that embodies the object. Consequently, resulting space is succesively subdivided into quadrants in order NW, NE, SW, SE, and represented by an aggregation hierarchy. Each successor of a specific node represents a quadrant obtained by subdivision, and terminals correspond to those blocks that capture pixels of same color. Each terminal node is said to be BLACK or WHITE depending on the color of data points of which its corresponding block consists. The nonterminals are said to be GRAY.

[SW88a, SW88b] give a thorough survey of variations of these data structures and moreover focus on advanced applications.

Concerning to this strategy, a very compact representation of images is provided. The storage requirements are directly potential to the resolution of the image, and doubling the resolution doubles the number of blocks whereas the number of pixels is quadrupled [Sa90]. Furthermore, the corresponding algorithms are easy to implement. Also measurements of properties such as area, perimeter, etc. are easy to maintain [Sa81b; Sa81c; Sa82]. However, the main weakness of this data structure is its lack in representing spatial structures. This is because spatial properties, i.e. data point colors, are considered just after a decomposition.

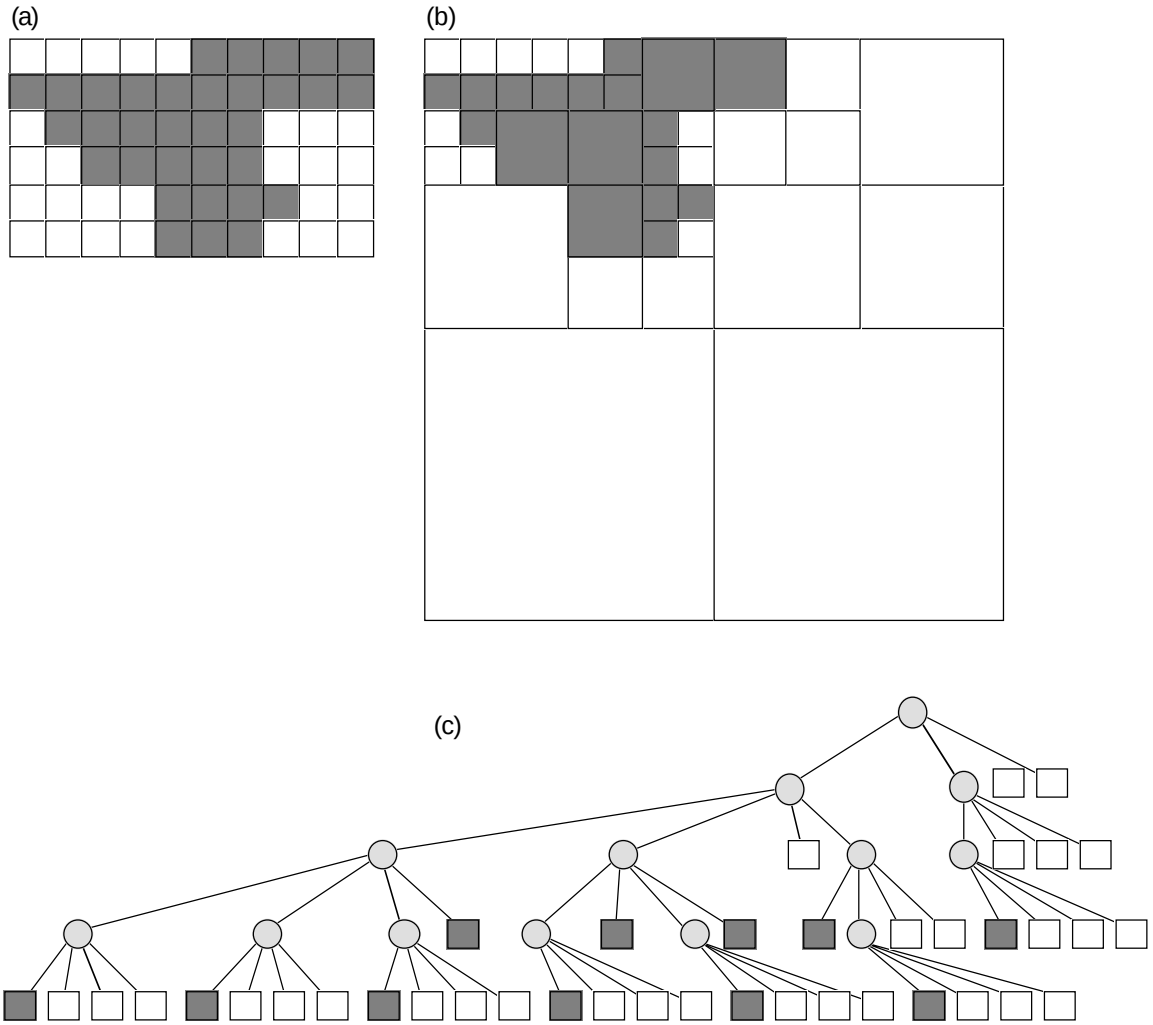


Figure 1: An image array (a), block decomposition (b) and the corresponding quadtree (c).

2.2 Heuristic Decomposition of Space

Every day, we spend much of our time treating with spatial problems, such as designing physical objects, recognize physical objects and geometric relations among them. To handle such problems, humans generally follow perceptual characteristics of

the physics of objects in space, i.e. they consider object structures to guide their behaviour. For example, having the task to describe an object, dominant structural features of the object are recognized first. The example in Figure 2 (a) shows a 2D object that represents obviously nothing that is well known by the reader. However, several associations come in mind and what ever they are, different possibilities for a syntactic verbal description exist. If the primitives of the description would be restricted to the conditions to follow the procedure of a hierarchical decomposition of the object in homogeneous rectangular regions of same color, several alternatives exist.

To limit the number of decompositions (number of resulting regions), alternative heuristics could be used:

[1] Use only vertical decomposition!

[2] Use only horizontal decomposition!

[3] Search first for dominant linear structures!

[4] Search first for rectangular regions having maximal sizes!

[5] Decompose in a binary fashion and alternate strictly!

Applying these heuristics to the object in Figure 2 (a) results in puzzles of rectangular regions which, combined in a certain order, describe the corresponding object. The applications are shown in Figure 2 (b) to (f), starting with heuristic [1] in Figure (b) up to heuristic [5] in Figure (f).

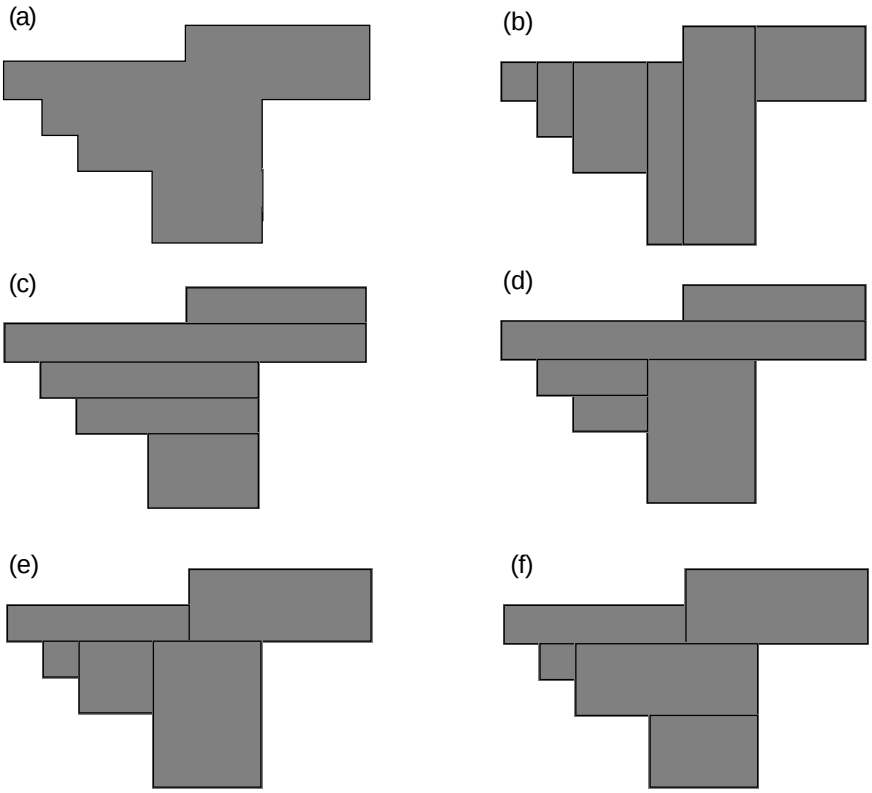


Figure 2: 2D object (a) and different representations after applying heuristics [1] to [5].

According to the costs (i.e. number of decompositions, number of nodes), the heuristics provide different results. While for heuristics [2] to [5] only 4 decompositions are necessary to describe the object by 5 rectangles (i.e. minimal costs), heuristic [1] needs 5 decompositions and 6 rectangles. There also exist various other alternatives to decompose the object in the example, in especially combinations of the solutions shown in Figure 2 (d) to (f). But they either do not follow any heuristic, or they need more costs, producing a higher number of homogeneous regions.

If the object in Figure 2 (a) is shown to a test person to verbally describe it, she or he is recognizing a set of rectangles which are clustered in a certain order. If subsequently the task is defined more precisely and the person has to describe the object by its composing rectangles, she or he follows the strategy to search for dominant linear structures in combination with their intensity (width). For the example of Figure 2 (a), tests have shown that most people intuitively would first select the vertical rectangle in the center of the object, like it is done in Figure 2 (b). But just after that, mostly they would like to reset their selection because of the structured object part on the left hand side of this rectangle which complicates a further decomposition. Moreover, most test persons do not follow any strategy or heuristic, but rather follow their intuition. If the task is altered and the person has to decide, which of the alternatives (b) to (f) of Figure 1 mostly reflects her or his verbal description of the object, 57 % of the persons (12 out of 21) selected alternative (d), while alternative (e) was selected 6 times and alternative (f) 3 times. Note that none of the persons has selected neither alternative (b) nor (c).

To somehow transfer behaviour of humans in spatial object structuring to an automate, heuristics that are proceeding in a similar way have to be defined by means of operations. Concerning the heuristics described above, strategy [1] and [2] seem easy to implement, but they are only suboptimal with respect to the describtion of structural object features. In addition, problems arise when considering a 2D object in the context of an entire image. Here at first, the objects have to be localized. For that reason, background data have also to be taken into consideration. Because heuristics [1] and [2] only use one orientation for decomposition, they are not expressive enough to describe objects in entire spaces. Moreover, a spatial consideration may hardly influence behaviour of a heuristic, producing sometimes exponential increasing of regions. Concerning the heuristics described above, heuristics [3] to [5] may directly be transfered to an entire image space (see Figure 3).

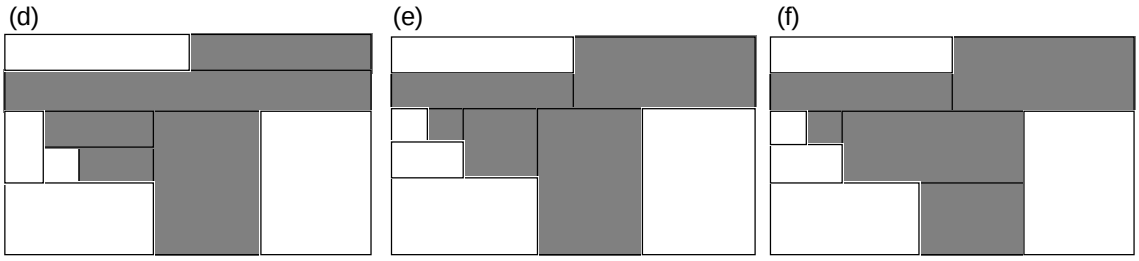


Figure 3: Results of applying heuristics [3] to [5] to 2D space describing the object of Figure 2 (a).

In all cases, 9 decompositions are necessary to represent this object space by 10 rectangles. These costs also reflect the minimum of the example considered. According to an automation, all of the heuristics seem to be adequate strategies, but following humans, heuristic [3] seems to be best. Trying to develop an algorithm, two of the three heuristics will fail.

For heuristic [4] it is necessary to find a method for determining maximal rectangular regions in a global image space. But this is a problem that is not tractable by machine, because it is necessary to establish and compare every combination of rectangular decomposition. Heuristic [5] also fails, because it is not decidable where to place a decomposition without having a global image view. Fortunately, heuristic [3] which is most adequate can be simulated by computer, and moreover, is easy to implement. This method is called puzzletree generation and will be described in the rest of this paper.

In the next section, some fundamental properties of puzzletrees are briefly introduced. Thereby, we are primarily concerned with images that only contain black or white regions to simplify our explanations. In this way, some definitions with respect to binary images have to be given. The term *image* refers to an original array of image points that is obtained by a camera or an optical scanner. Image points are the basic elements of an image. They often are designated by *pixels*, having a different value according to their color or grey level. If pixels of an image are either black or white, then the image is said to be *binary*. Moreover, pixels are *of the same color*, if their grey level or color is defined by the same value, i.e., in the case of binary images, they are entirely black (defined by value “1”) or entirely white (defined by value “0”). The term *block* determines a rectangular image region that may describe the entire input image as well as subsets of it.

3 PRINCIPLES OF PUZZLETREES

The puzzletree is a technique for image space structuring and representation. When generating puzzle-trees, it is not necessary to consider images of size 2^n x 2^n, such as the conventional quadtree approach does. Instead, rectangular images of any size and shape can serve as input. Based on successive subdivision, an input image is decomposed in rectangular adjacent blocks of same heigth or same width. Subdivision can be applied in either vertical or horizontal direction. If a block does not consist of pixels of the same color, it is subdivided into subblocks until the criterion is fulfilled. Thus, the resulting block puzzle of an image space represented by a puzzletree consists of blocks having arbitrary size and extensions rather than squares.

Figure 4 illustrates the principles of puzzletrees. The figure shows the example of two image objects (a) — a simplified chair and table —, the corresponding block puzzle resulting from spatial puzzletree consideration (b) and the internal representation of the corresponding puzzletree (c). The order of numbers designating the terminal nodes of the trees is determined by a breadth-first generation.

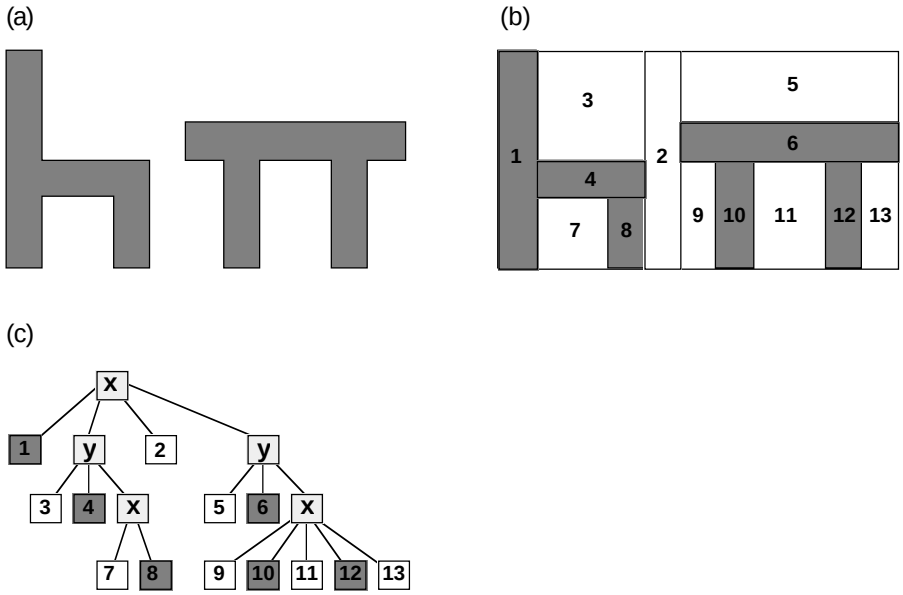


Figure 4: A 2D object (a), its block puzzle representation (b) and the corresponding puzzletree (c).

The root node of a puzzletree corresponds to the entire input space (image) by considering the enclosing rectangle of the objects of interest. The subblocks obtained during subdivision are represented by respective child nodes. Concerning the example of Figure 4, in a first step, image space is divided in four subblocks by vertical decomposition. Each of the four child nodes are in order from left to right. In case of a horizontal subdivision, the order would be from top to bottom. Nonterminal nodes are denoted as *division nodes*. A division node is said to be horizontal if it is designated by a label "y" (decomposes an image space in horizontal direction at y-positions), and vertical if it is designated by a label "x" (decomposes an image space in vertical direction at x-positions). All child nodes of a node have either the same height or the same width depending of the direction label designating their parent node. All terminal nodes of the puzzletree represent those blocks for which all containing pixels are of same color. In the example, they are either black or white. Note that unlike quadtrees, the degree of the tree depends on the number of subdividions of a rectangle in either horizontal or vertical direction.

4 GENERAL STRATEGY

To generate puzzletrees from an image space, several processing steps are necessary. An image is input into a computer through an optical scanner tracking row by row to obtain digitized image data. Then, the raster image is tracked row by row and column by column to determine runs which consist of connected pixels of same colour. The resulting run-length encoding (RLE) [Ru68] for image rows and columns is the basis for a heuristic that is used to evaluate linear homogeneity within image space to direct recursive decomposition of blocks. For all blocks resulting from a decomposition,

frame-like objects are generated to represent their geometric properties and spatial relations. As a result, a puzzletree is obtained. It may be interpreted as an aggregation hierarchy containing blocks as terminals that capture pixels of same color and higher-level blocks which correspond to division nodes. Figure 5 summarizes the different processing steps for puzzletree generation.

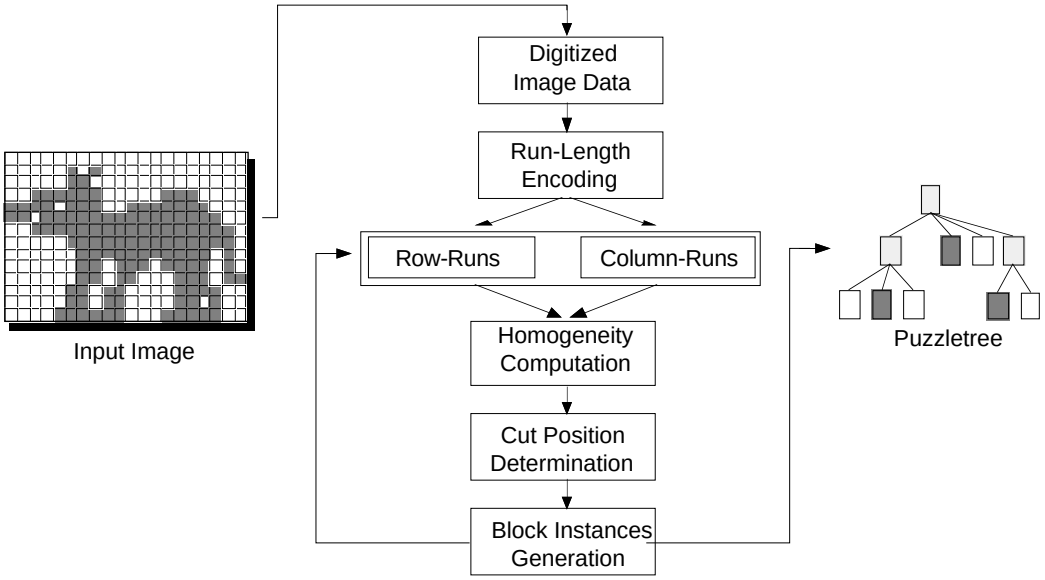


Figure 5: General strategy for puzzletree generation.

5 GENERATION OF PUZZLETREES

During puzzletree generation an image is not decomposed at fixed, predefined points of the array. Rather, it depends on the spatial structure and position of image objects which are described by connected nonzero data points. This strategy includes that the degree of decomposition (i.e. number of resulting subblocks) is neither regular nor in equal-sized blocks, but rather may be governed by the input data. Therefore, the spatial image structure is first examined with respect to dominant linear features of both, objects and background.

5.1 Linear Homogeneity

The structure of an image space is characterized by a certain complexity. This complexity is mainly influenced by number and size of homogeneous image regions having same color. For a one-dimensional decomposition of space, it is only necessary to consider linear homogeneity, which depends on the frequency and the length of pixel sequences in a row or a column having different colors. In Figure 6, image rows of different degree of linear homogeneity are shown. The figure shows rows having high (a), medium (b), and low (c) homogeneity degrees.

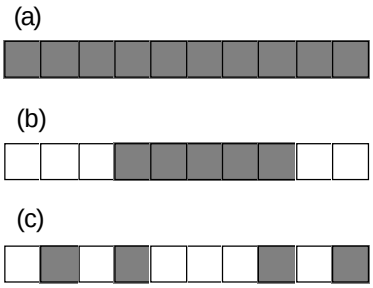


Figure 6: Binary image rows having high (a), medium (b), and low homogeneity degrees.

To validate the complexity of images, rows and columns are entities for which their degree of linear homogeneity could be measured. Therefore, the following criteria are considered:

- The degree of linear homogeneity of an image row or column decreases with the number of changes in color.
- The degree of linear homogeneity increases with the length of a pixel sequence in a row or column that are of the same color.

Taking these criteria, the set of pixels in an image row or column define the corresponding degree of linear homogeneity, no matter which color they have. To define linear homogeneity precisely, the digitized image data is first encoded by run-length encoding (RLE) [Ru68]. This is done for both, rows as well as columns, whereas rows are tracked from left to right and columns from top to bottom. The resulting code consists of a set of pairs (c_i, x_i). These pairs correspond to pixel sequences within a row or column, whereby x_i denotes the number of pixels following each other by having the same color c_i. Considering the examples in Figure 5, the first row can be encoded by {(1, 10)}, whereas examples (b) and (c) have respective RLEs of {(0, 3), (1, 5), (0, 2)} and {(0, 1), (1, 1), (0, 1), (1, 1), (0, 3), (1, 1), (0, 1), (1, 1)}. RLE for rows and columns of an image space is applied once at the beginning of puzzletree generation. To obtain a value that expresses the criteria for linear homogeneity, the following definition is made:

Definition: **Linear Homogeneity**

Assume, the length of an image row or column is given by L. Furthermore, the corresponding RLE is described by the set of pairs $\{(c_1, x_1), (c_2, x_2), \ldots, (c_n, x_n)\}$, whereby $1 \le n \le L$. In the case of binary images, $c_i \in \{0, 1\}$, whereas "0" corresponds to color white and "1" to color black. All values of x_i are defined by $1 \le x_i \le L$. Then the degree of linear homogeneity H of a row or column is defined by the quotient of the sum of the squared x_i and the square of the length of the row or column L

$$H = \frac{\sum_{i=1}^{n} x_i^2}{L^2} \quad , \qquad 1/L \le H \le 1 \, . \qquad (1)$$

The following example illustrates the maximal and minimal degree H_{max} and H_{min} of linear homogeneity for a row having a length L of 10. The corresponding RLEs are {(1, 10)} and {(1, 1), (0, 1), (1, 1), (0, 1),(1, 1), (0, 1),(1, 1), (0, 1),(1, 1), (0, 1)}.

Example:

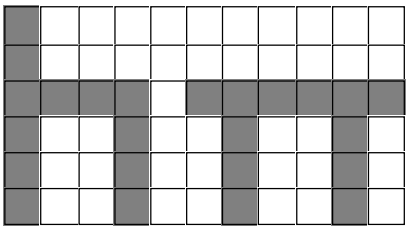
$H = H_{max} = 1$

$$H = H_{min} = \frac{\sum_{i=1}^{10} 1^2}{10^2} = \frac{1}{10}$$

Applying Formula (1) to the examples of Figure 6, degrees of linear homogeneity of H_a = 1, H_b = 0,38 and H_c = 0,16 can be calculated.

5.2 Orientation and Position of Decomposition

In Figure 7, the image space of Figure 4 (including object and background data) and corresponding RLEs of rows and columns are shown.



RLE - rows (1 .. 6)

{(1, 1), (0, 10)}
{(1, 1), (0, 10)}
{(1, 4), (0, 1), (1, 6)}
{(1, 1}, {0, 2), (1, 1}, {0, 2), (1, 1}, {0, 2), (1, 1}, {0, 1)}
{(1, 1}, {0, 2), (1, 1}, {0, 2), (1, 1}, {0, 2), (1, 1}, {0, 1)}
{(1, 1}, {0, 2), (1, 1}, {0, 2), (1, 1}, {0, 2), (1, 1}, {0, 1)}

RLE - columns (1 .. 11)

{(1, 6)}
{(0, 2), (1, 1), (0, 3)}
{(0, 2), (1, 1), (0, 3)}
{(0, 2), (1, 4)}
{(0, 6)}
{(0, 2), (1, 1), (0, 3)}
{(0, 2), (1, 4)}
{(0, 2), (1, 1), (0, 3)}
{(0, 2), (1, 1), (0, 3)}
{(0, 2), (1, 4)}
{(0, 2), (1, 1), (0, 3)}

Figure 7: A binary image and corresponding runs for rows and columns after RLE.

Calculating the degrees of linear homogeneity by Formula (1), *homogeneity vectors* R_H (for rows) and C_H (for columns) can be determined that capture the respective values.

R_H = (0.83, 0.83, 0.44, 0.14, 0.14, 0.14);

C_H = (**1**, 0,39, 0,39, 0,56, **1**, 0,39, 0,56, 0,39, 0,39, 0,56, 0,39);

This vectors are used to determine the orientation and position for a decomposition.

Linear homogeneity may also be interpreted as a measurement about how often and in what distances a row or column has to be decomposed to obtain sequences of pixels

having the same color. To obtain an almost minimal number of decompositions, rows or columns having a high degree of linear homogeneity should be selected. Ideal are those rows and columns having the maximal linear homogeneity of “1”, because no further decomposition is necessary.

In the example the homogeneity vector C_H captures two maxima of linear homogeneity having a value of 1. Thus, the algorithm prefers a vertical subdivision of the original image block. Note that both, object data as well as background data are taken into account. In the case of the example, three positions of decompositions are determined after columns 1, 4 and 5.

5.3 Start Orientation

In many cases, the problem of determining the start orientation (i.e. vertical or horizontal) of a first decomposition is not trivial because both, R_H and C_H could have same maxima. In Figure 8, an example of a raster image is shown. The example describes two adjacent image columns and one row having maximal degree of linear homogeneity. For the rest of the image, no further description is made.

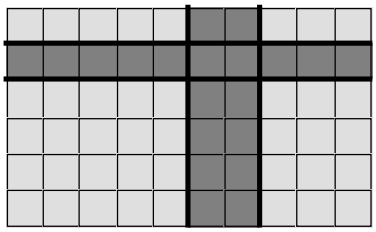


Figure 8: Example of an image having the characteristic to show maximal degree of linear homogeneity along horizontal and vertical dimensions.

Taking the above described heuristic for puzzletree generation, there exist several alternatives to decide about the orientation of a first image decomposition. For making a descision, there exist several possibilities:

- Length of a row or a column.
- Number of rows or columns having maximal degree of linear homogeneity.
- Number of adjacent maxima.
- Number of pixels of directly resulting terminal blocks.

Here, either the length of rows and columns (A) or the size of a block (B) are considered as criterion. Respecting the example, the result is either horizontal (criterion A) or vertical(criterion B).

Tests have shown that a satisfying decision can not be made without examining the global image structure. According the achieve optimal compression of an image, i.e minimal number of puzzletree nodes, it is necessary to consider the color of all pixels — and this seems to be an unsolvable problem. Figure 9 illustrates this fact for an example in which adjacent pixels influence the results obtained by puzzletree

generation. Here, the four bottom-most pixels in column 5 — Figure 9 (a) — and the 5 left-most pixels in row 3 — Figure 9 (b) — are set to black. Depending on the choice of the first cut orientation, puzzletrees having different numbers of nodes are obtained. In Figure 9 (a) the better choice has been first to horizontally decompose, while in Figure 9 (b) the better alternative has been a vertical decomposition.

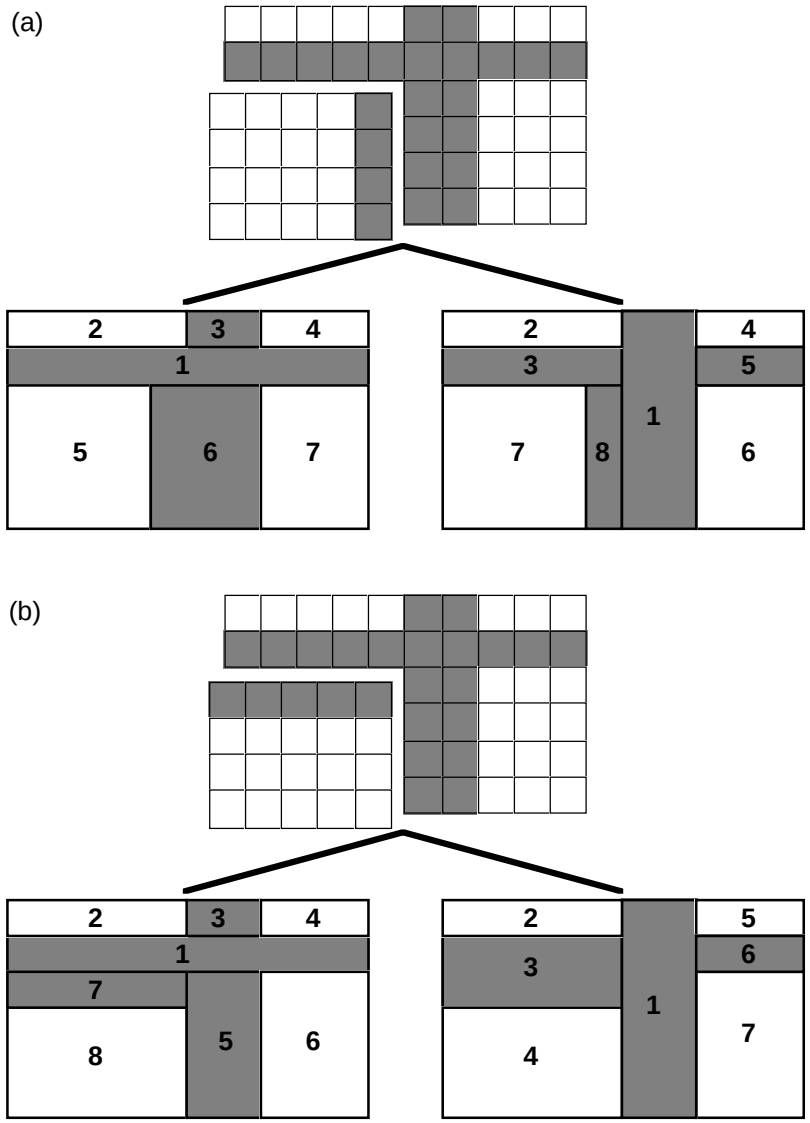


Figure 9: Influence of adjacent pixel colors to the results obtained by of puzzletree generation.

To resolve this conflict much more information is necessary. The part of image that is relevant for determining a first cut orientation can be given by a set of pixels which are adjacent to the rows and columns under consideration or furthermore can be hidden anywhere in the image. Figure 10 illustrates such an example on which a final descision about the first cut orientation can be made very late during puzzletree generation.

If it would become possible to examine and furthermore validate a global image space, it is a short step to obtain optimal compression (i.e. minimal number of blocks). However, an image space is two-dimensional and linear homogeneity is one-dimensional and therefore expresses only linear features that can be used for a local decision for the actual subdivision step.

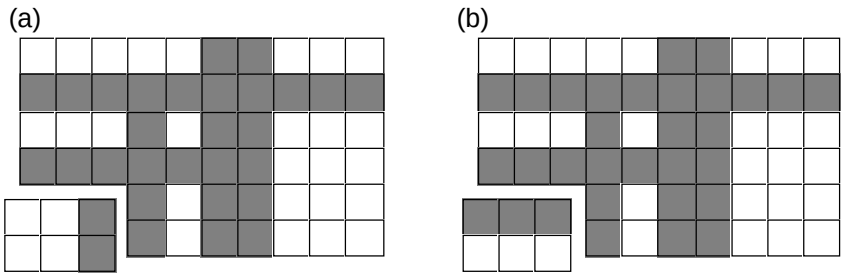


Figure 10: Influence of pixel colors to the results obtained by of puzzletree generation.

Due to this fact, several alternatives exist that may be used to prefer either vertical or horizontal subdivision. In the puzzletree approach, the first criterion is the number of adjacent rows or columns in which the maximum occurs. If there are single rows or columns showing same maximum, the length of them is considered. In the case vectors R_H and C_H capture the same number of adjacent maxima having moreover the same degree of linear homogeneity, the length of rows and columns is taken as a second criterion. Nevertheless, if the first orientation for a decomposition is chosen, subsequent orientations follow by a strictly alternation.

Applying these criteria in stepwise and recursive manner to a certain image space, heuristic [3] like shown in Section 2, can be fully transferred to a computer. In Figure 4, the result is shown of applying this strategy to an image.

5.4 Alternation of Cut Orientation

If images have a higher complexity, the intermediate results of the respective puzzletree needs few minor corrections to fulfil the criterion of a strictly alternation in cut orientation. This criterion is desirable for subsequent processing of the data structure, because it simplifies the operations to be initiated. The example in Figure 11 illustrates the correction. It is rather a minor than a fundamental problem and is easy to solve; the respective nodes are deleted and their successors are added to the successors of their predecessor (ref. Figure 11, step (2)).

Considering the final result of image space decomposition, a puzzletree can also be interpreted as a rule of how to combine a large number of blocks of different size, extensions and color to reproduce a 2D object. This property may be of significant importance in object recognition problems and is different from conventional region quadtrees.

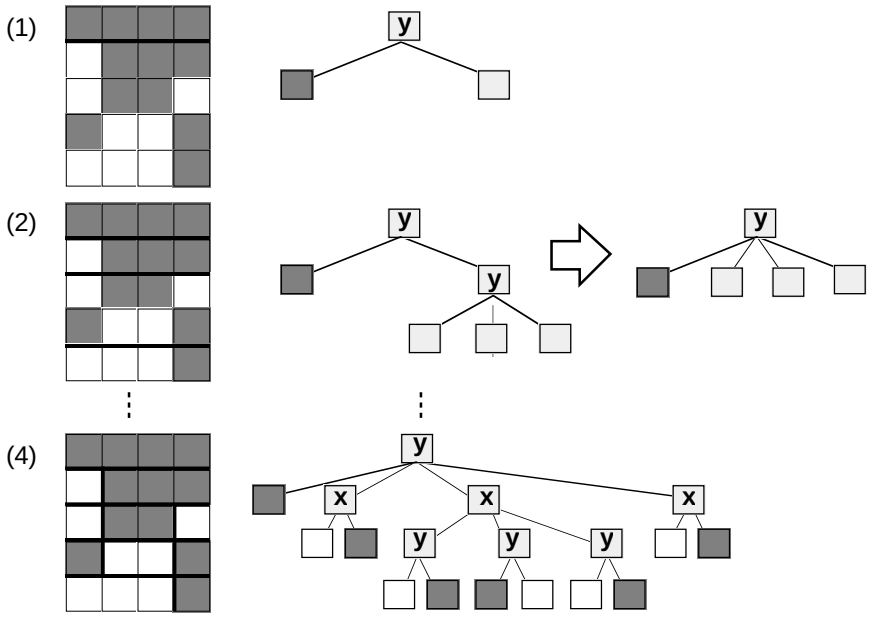


Figure 11: Providing a strictly alternation of cut orientation in puzzletrees.

6 REPRESENTATION OF PUZZLETREES

For an internal representation of puzzletrees, at least two possibilities come to mind. The first and most obvious encoding is a frame or record-like representation of the puzzletree nodes that are referenced by pointers. An alternative possibility for encoding is given by a special notation called puzzlecode [De91b].

6.1 Object-Oriented Representation

For our work in image analysis, we need an internal representation of a puzzletree that should be easy to generate, easy to modify and easy to understand. Since the philosophy of object-oriented programming and data models address our requirements, we use frame-like objects to represent nodes of a puzzletree and to relate them.

Thus, every image block, whether it is a division node or a terminal, is represented by a frame-like object. In general, an application domain may be handled in terms of such objects (see also [DM91]). Each object is an instance of the particular type which defines the object´s structure. The structure of an object is a description that is specified by declarative as well as procedural parts. Declarative parts are properties and relationships, while procedural parts are methods that can be applied in any processing state.

In our domain, an object-oriented spatial representation of an image consists of an aggregation of image blocks having different grain. With the type *block* certain methods (i.e. comp-area, comp-perimeter, etc.) are associated as well as a specific template of properties (i.e. x-origin, y-origin, height, etc.) and relations (i.e., part-of, has-parts, etc.). Figure 12 shows the structure of the type block.

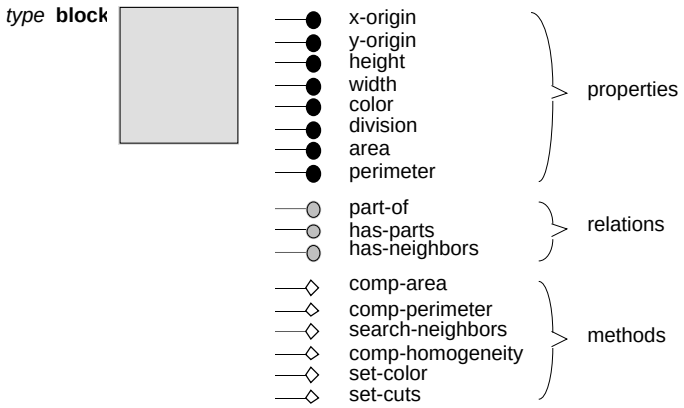


Figure 12: Structure of type block.

Every instance of the type block is an object that matches the template defined for a block having specific values instantiated for each property and relationship. Figure 13 illustrates an example for object-oriented image representation. Properties of block instances are illustrated explicitly, while relations are shown by links between type and instances as well as between blocks and subblocks (part-of- and has-parts). Neighborhood-relations are not explicitly shown.

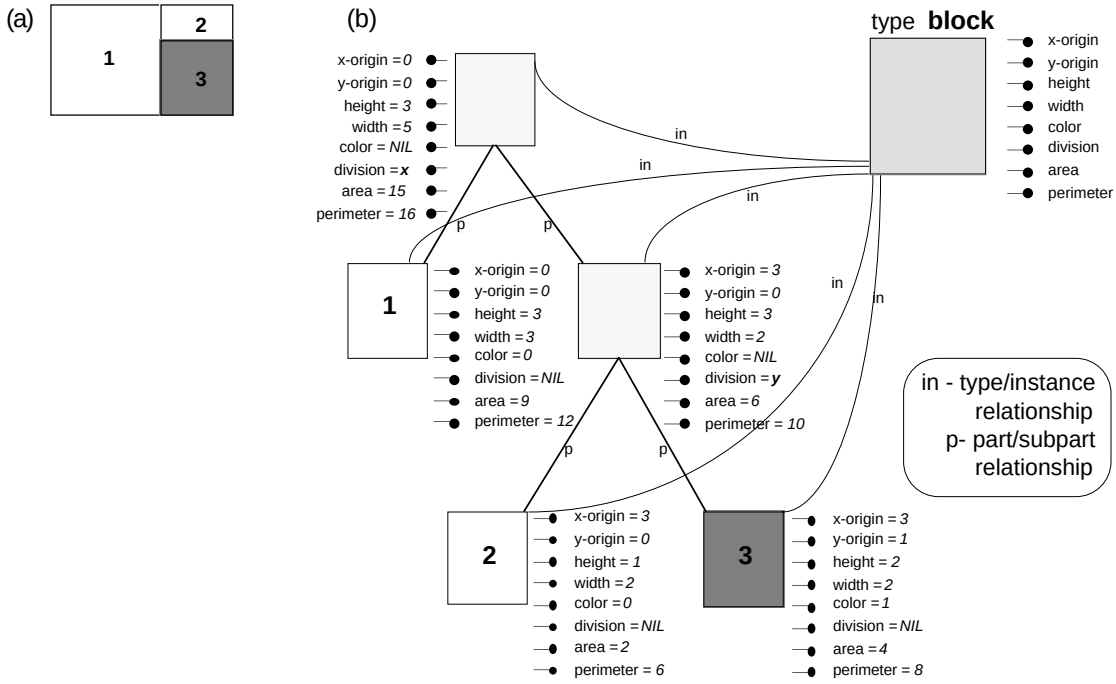


Figure 13: Example of an image (a) and corresponding hierarchy of block instances (b).

In such object-oriented aggregation hierarchies the value for a particular property can be stated explicitly, or obtained implicitly by evaluating the corresponding methods, or obtained by inheritance. For instance, the area of a block is implicitly determined by

a function multiplying its dimensions. Another example is the value of the width for the subblocks obtained by horizontal decomposition of a block that can be directly inherited from this block. Using such methods to derive higher-level information is a primary task in computer vision.

6.2 Puzzletree Encoding

An alternative possibility for encoding is given by a special notation called puzzlecode (also used in a similar way in another approach [DB89]). While the record or frame representation of puzzletrees is a basic medium for subsequent knowledge-based operations on image space (e.g., neighbor finding, connected component labeling, object detection) [De91a], the puzzlecode is a direct description of the image having primarily the intention of compression. But, it also allows a discussion of geometric concepts in terms of coded images themselves, similarily proposed for quadcodes [LL87].

The puzzlecode is a restricted notation based on lists and primitive symbols, describing subdivision and characteristics of an image space. This is done by nested sublists consisting of three kinds of designators. These are:

- letters "x", "y" denoting orientations for a subdivision,
- position labels,
- values which indicate the color of a block.

The position labels are represented by fractions. The numerator indicates the row or column of pixels within a block at which a decomposition has to take place, whereas the denominator indicates the length of a row or column. For better distinction, color values are quoted.

Using these primitives, the puzzletree in Figure 4 (c) can be expressed by:

```
(x 1/11 4/11 5/11      ´1
                 (y 3/6 4/6  ´0
                            ´1
                            (x 2/3 ´0 ´1))
                 ´0)
                 (y 2/6 3/6  ´0
                            ´1
                            (x 1/6 2/6 4/6 5/6  ´0  ´1 ´0 ´1 ´0)))
```

The above example of puzzlecode can be read as follows: First, the input image is subdivided in vertically direction at positions 1/11, 4/11 and 5/11. The first of the resulting blocks contains pixels that are entirely black. The second of the resulting blocks is further divided in horizontal direction in three subblocks at position 3/6 and 4/6, etc.

7 PUZZLECODE COMPRESSION

In several cases, if the structure of a given image space is very complex, a non-optimal subdivision of a block can be initiated. Applying the puzzletree generation heuristic to the example shown in Figure 14 (a), an appropriate puzzletree (b) is

generated. The respecting vertical decompositions at positions 1/5 and 2/5 provide two subblocks which both are further subdivided at position 1/3 into subblocks of same color — see Figure 14 (b), shaded area.

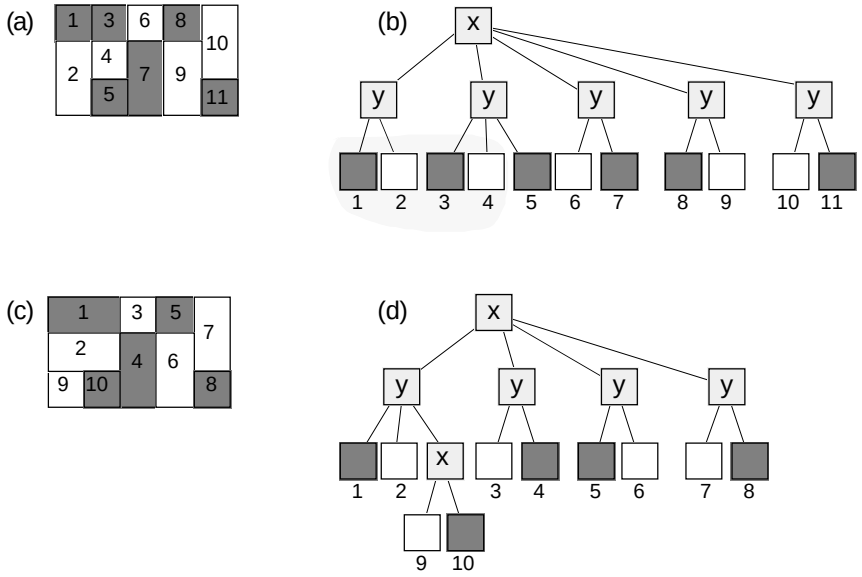


Figure 14: Example of an image (a) and corresponding hierarchy of block instances (b).

Assume these blocks are designated as b_1 and b_2, then they may be represented by the following puzzlecode:

b_1 = (**y 1/3 ´1 ´0**)

b_2 = (**y 1/3** 2/3 **´1 ´0** ´1)

The fact that both blocks b_1 and b_2 have same parts with respect to cut position and color can be used to reduce the number of nodes needed for puzzletree representation. Therefore, the first cut position 1/5 of the image in Figure 14 (a) can be deleted and the blocks b_1 and b_2 are combined to a single block. This combination of blocks is called *compression*.

Definition: **Puzzlecode compression**

Assume, o, $ø$ designate orientations for a possible decomposition; $o, ø \in \{x, y\}$ and $o \neq ø$. In addition blocks b_1 and b_2 are described by the puzzlecodes

$$b_1 = (o \;\; p_{1,1} \;\; p_{1,2} \;\; \ldots \;\; p_{1,n} \;\; c_{1,1} \;\; c_{1,2} \;\; \ldots \;\; c_{1,n} \;\; c_{1,n+1})$$
$$b_2 = (o \;\; p_{2,1} \;\; p_{2,2} \;\; \ldots \;\; p_{2,m} \;\; c_{2,1} \;\; c_{2,2} \;\; \ldots \;\; c_{2,m} \;\; c_{2,m+1})$$

whereby $p_{i,j}$ denotes the cut position and $c_{i,j}$ the color of resulting blocks. Assume also, boundaries of block b_1 are defined by $p_{1,0}$ and $p_{1,n+1}$ and those of b_2 by $p_{2,0}$ and $p_{2,m+1}$. Furthermore, b_1 and b_2 are decomposed at position p_0.

Then, the puzzlecode compression of blocks b_1 and b_2

$$b_3 = b_2 \pm b_1 = (o\ \ p_{3,1}\ \ p_{3,2}\ \ \ldots\ \ p_{3,l}\ \ c_{3,1}\ \ c_{3,2}\ \ \ldots\ \ c_{3,l}\ \ c_{3,l+1})$$

is defined by:

$$c_{3,k} = \begin{cases} c_{1,i} & if\ \ c_{1,i} = c_{2,j} \\ (\text{ø}\ p_0\ c_{1,i}\ c_{2,j}) & else \end{cases} \qquad (2)$$

$$p_{3,k} = \begin{cases} p_{2,j} & if\ \ p_{1,j} > p_{2,j} \\ p_{1,j} & else \end{cases} \qquad (3)$$

for $c_{1,i-1} < c_{2,j} \le c_{1,i}$ or $c_{2,j-1} < c_{1,i} \le c_{2,j}$ and $k = i + j - \sum_{i,j}(c_{1,i} = c_{2,j})$.

Note, that resulting $c_{3,k}$ may represent both, block colors or further decomposition in subblocks. Taking the same assumption as for the definition, the following algorithm for puzzlecode compression may be given:

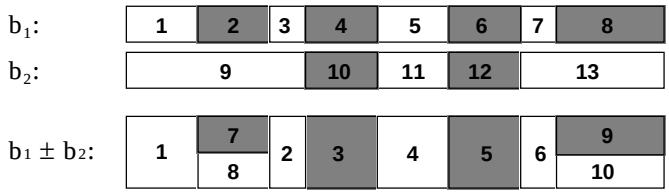
Algorithm: *compression*:

```
(1)   i := 1; j := 1; k := 1;
(2)   while not (i ≥ n and j ≥ m) do begin
(3)         if (c1,j = c2,j )                                  then c3,k := c1,i
(4)                              else c3,k := (ø  p0  c1,i  c2,j )
(5)         if (i ≤ n) and (j ≤ m) then
(6)               case   (p1,i < p2,j) or (j = m+1):  p3,k :=  p1,i ;  i := i+1 ;
(7)                      (p1,i > p2,j) or (i = n+1):  p3,k :=  p2,j ;  j := j+1
(8)                      else:                        p3,k :=  p1,i ;  i := i+1 ;  j := j+1 ;
(9)         k := k+1 ;
(10)  end;
```

Applying puzzletree compression to b_1 and b_2 of the example in Figure 13, the compressed notation $b_1 \pm b_2$ = (**y 1 / 3** 2 / 3 **´1 ´0** (x ´0 ´1)) is obtained. This compression results in reducing the number of nodes needed for the puzzletree representation of the image. The result of this compression is shown in Figure 14 (c-d).

With respect to this example, the effectiveness of puzzletree compression does not become obvious. But there may be examples for which corresponding rows and/or columns are much longer than in the example of Figure 14. In praxis, such examples exist very often. The following example illustrates such a case.

Example: Two adjacent blocks b_1 and b_2 before and after compression.



After compression, the number of terminals is reduced from 13 to 10. As a side effect, the number of nonterminals increases by 2 nodes (can be recognized at these positions of the example, where horizontal cuts are added).

The effective saving of nodes can be calculated by taking the difference between *direct matches* (corresponds to the number of saved terminals) and *direct mismatches* (corresponds to the number of new nonterminals). Direct matches belong to blocks which describe same intervals with same color along one and the same orientation. like block 4 of b_1 and block 10 in b_2 or block 5 in b_1 and block 11 in b_2 in the above example. Direct mismatches describe blocks which have different colors along one orientation and the boundaries of one block are within the boundaries of the other, like block 2 of b_1 and block 9 in b_2 in the example.

8 EXTENSION TO 3D DATA

While octrees [Hu78, JT80] are the equivalent of quadtrees in 3D space representation, the puzzletree approach may also be extended to describe volume data. Similar to puzzletrees, objects may be described by blocks (in the sense of 3D-blocks) rather than by cubes. This is done by successive decomposition of 3D space in block elements of same color but of arbitrary size, extensions, and volumes. Without delving in details, the extension of puzzletrees to 3D data is straightforward. Figure 15 shows the example of a chair (a), its blockpuzzle (b) as well as the generated 3D puzzletree.

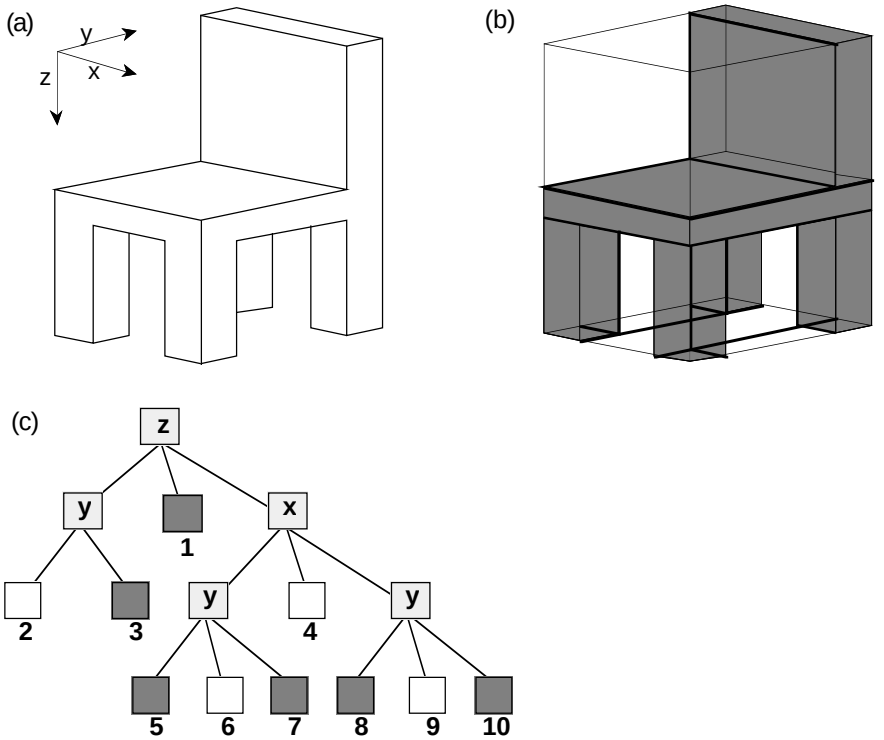


Figure 15: 3D-blockpuzzle (b) and puzzletree (c) of an chair-object (a).

As illustrated, orientations for a decomposition are extended to x, y and z axes. Note that the node No.1 describes the seating of the chair, node No. 3 its back, and nodes No. 5, 7, 8, and 10 represent the legs.

While 2D puzzletrees describe rules of how to combine blocks to a 2D puzzle, 3D puzzletrees are equivalent rules to define the creation of 3D objects, or spaces by blocks. This property is important for object location and recognition, because such descriptions can be used as reference patterns of how objects are structured, and therefore, may serve as a basis to classify objects at hand. Thereby, it is not important to ensure a detailed description of an object (i.e., in the case of rounded or circular parts of an object), but rather to be able to represent objects by rules that offer descriptions of different degrees of abstraction. Considering the example in Figure 15, that means, there may be cylindrical legs of the chair, but for the puzzletree representation, this fact only leads to a deeper tree having additional descriptions in the subtrees at nodes No. 5, 7, 8, and 10.

9 RESULTS AND DISCUSSION

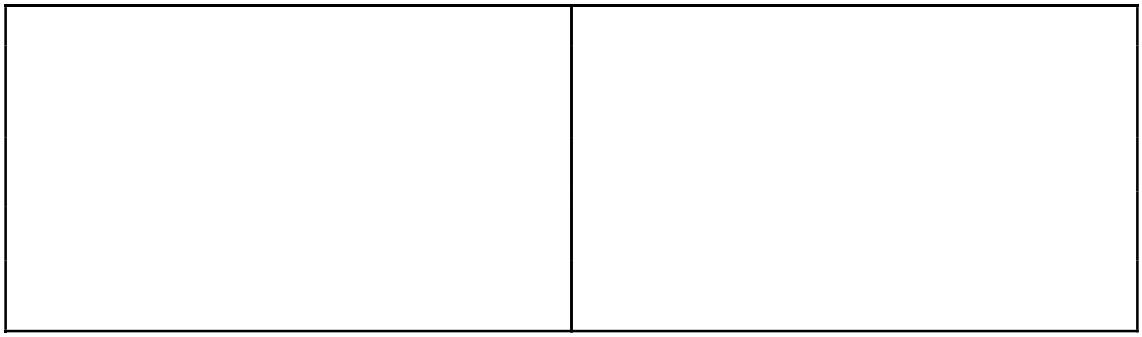
Several comparisons of the puzzletree approach to conventional region quadtrees can be made. While a decomposition in quadtree generation hardly is influenced by the position of a grid over an image, puzzletrees, by contrast, are established independent of this fact, because of an adaption to the spatial structures at hand. Nevertheless, some comparisons to "classical" region quadtrees can be made.

In the puzzletree approach, an image space is only decomposed into one direction obtaining rectangular blocks of same height or length, whereas a quadtree subdivides an image into four disjoint congruent square blocks. A respective tree structure expressing quadtree properties and moreover strictly alternating between x and y axes is the bintree [Kn80, Sa84]. But here, a space is always subdivided into two parts of equal size. In contrast to region quadtrees, the blocks obtained during puzzletree generation have no regular sizes and extensions, nor positions but depend on the complexity of the input image, like the degree of subdivision. Thus, size and shape of a puzzletree are extremely sensitive to the structure of an input image. However, puzzletrees are very flexible because of their consideration of spatial structures and therefore, provide a very compact representation of space. In addition, the complexity of parts of the puzzletree, i.e., subtrees of it, allows a derivation of the spatial structure in different image parts.

Another comparison is possible by considering the number of nodes (terminal nodes) and the generation time. Like quadtrees, the worst case for a puzzletree of a given depth is given when the image array corresponds to a chessboard pattern. In this case the number of nodes that are needed is equivalent to the quadtree approach. In all other cases, the puzzletree provides results having a lower number of nodes (terminal nodes) than obtained by quadtrees. For comparison, we have implemented both approaches in Common-Lisp on a Mac II fx, being equipped with 8 MBytes main memory. Concerning cpu time, puzzletree generation needs about 0.8 to 4 times of the time that is needed for quadtree generation. Considering a set of 50 images, having different size and structure, the averaged factor is about 2.6. Concerning the number of nodes which are needed for conventional binary image representation, puzzletrees need about 4 to 40 times less nodes than quadtrees need.

In Figure 16, examples (a-c) of binary images are shown. The rectangles in the examples describe the space that is considered for puzzletree and quadtree generation. The examples are: an image of a single word (a), a text image (b), an image of a fifty austrian schilling bill (c), and an image of a globe (d).

(a) (b)



(c) (d)

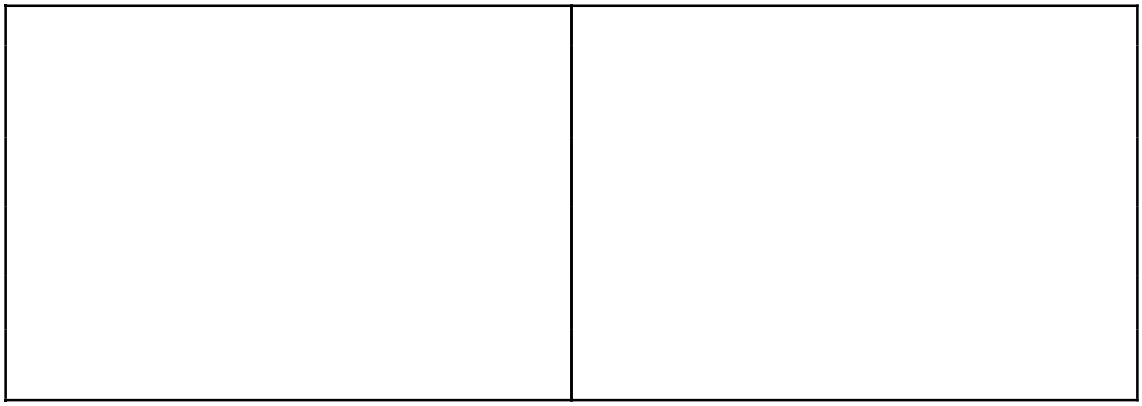


Figure 16: Examples of image spaces.

Note, that for a respective quadtree generation, the arrays have to be extended to the minimal square of size 2^n x 2^n that embodies the selected image space.

The subsequent table captures the results obtained by puzzletree as well as by quadtree generation. The differences in generation time are mainly caused by calculating the homogeneity criterion. Note also the large differences in number of nodes (leaf nodes) needed for image space representation. This is important for efficiency of all subsequent operations on the data structure.

		nodes		leaf nodes		gen. time (cpu-sec.)	
expl	image size	QT	PT	QT	PT	QT	PT
(a)	42 x 96	6420	221	4816	151	4.83	5.43
(b)	126 x 116	8340	2055	6256	1373	5.40	14.45
(c)	94 x128	10252	4199	7690	2929	9.47	28.15
(d)	140 x 147	29372	2129	21290	1396	25.57	35.92

Table 1: Comparison of quadtree (QT) and puzzletree (PT) approach.

In Figure 17 a binary image of text scanned by a resolution of 200 dpi is shown. In the upper right part, of the image, a rectangular region is described for being input for both, puzzletree generation as well as quadtree generation. In Figure 18 the resulting decomposition of the image block is illustrated, i.e. puzzletree decomposition on the left hand and quadtree decomposition on the right.

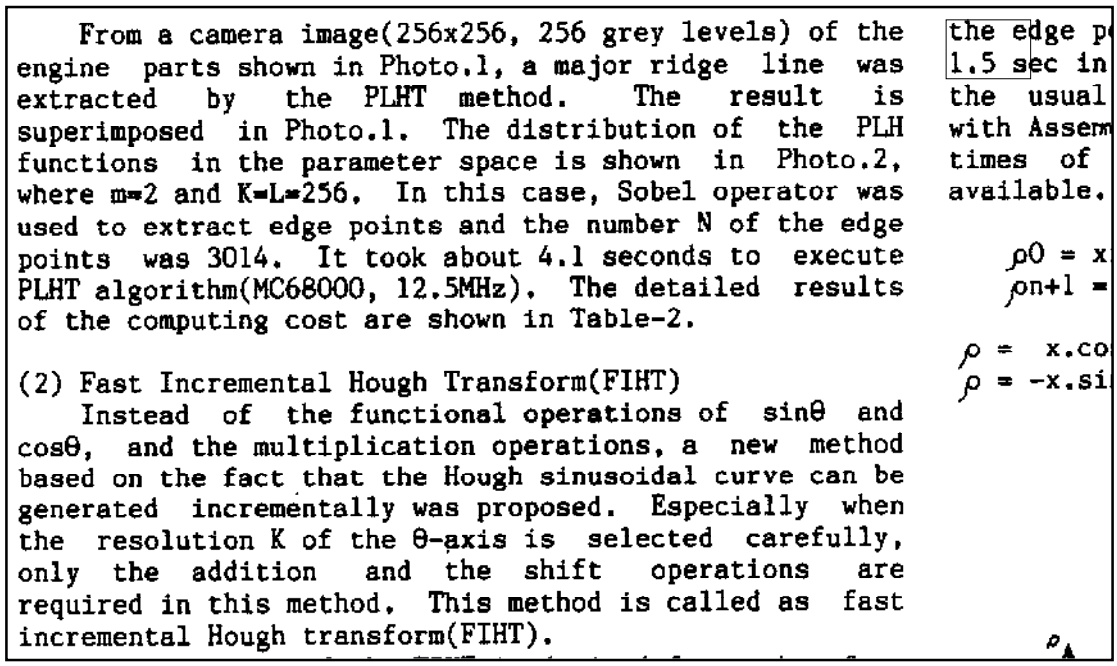


Figure 17: Binary image of a text showing a block being input for puzzletree and quadtree generation.

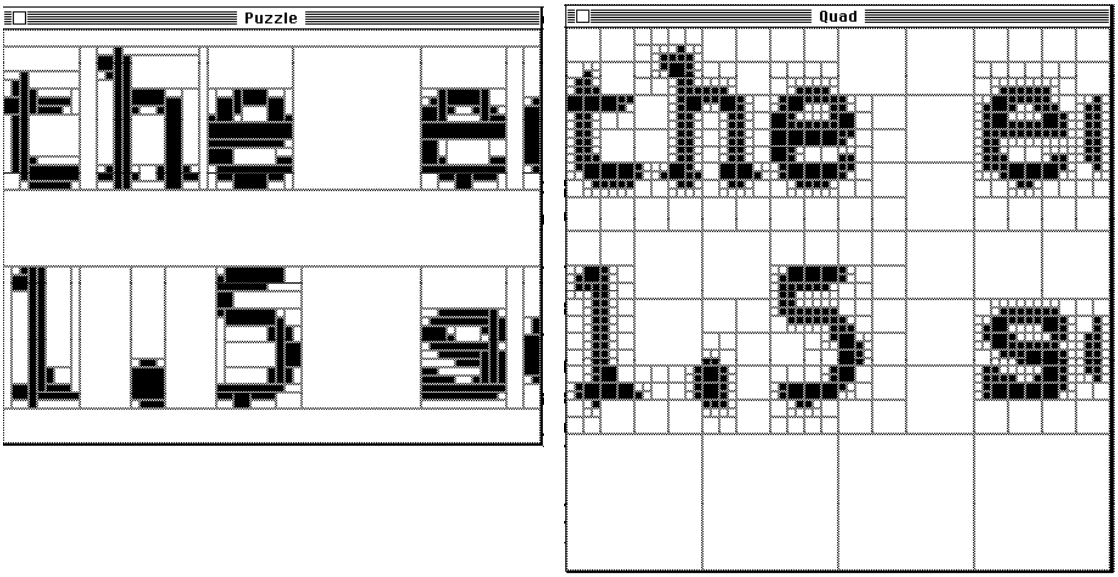


Figure 18: Puzzletree decomposition and quadtree decomposition of the image block shown in Figure 10.

The original image block size is 63 x 49 dots, while for quadtree generation, it is expanded to 64 x 64 dots. Generation time for puzzletree generation is 3.5 seconds and for quadtree generation 1.45 seconds. Considering the number of nodes, the puzzletree approach needs 332 nodes (225 leaves), while quadtree generation needs 1260 nodes (946 leaves) .

We are currently implementing a procedure for determining neighborship relations as well as connected components. In addition, we examine the approach with respect to recognition tasks in document images, i.e. text and graphics.

10 CONCLUSIONS

The algorithm presented in this report provides a method for efficient physically structuring of space. The procedure may be used to generate object-oriented representation of space [De91b] or for its puzzletree encoding [De91a]. It may be applied either for representing images or volumes, or any n-dimensional space.

Advantages of the approach are its sensitivity to a spatial structure at hand and its capability of self-adaptation. For this reason, a very compact aggregation of space is attained which allows for an effective application of subsequent operations (neighbor finding, connected component labeling, segmentation, etc.). However, for a broader application of the puzzletree in object recognition tasks, some additional work is necessary. Primarily, the approach is also very useful for object recognition tasks, but for a general application, the puzzle representation is not immediately usable because it does not have properties like rotation invariance.

Puzzletrees of space are generated in a way that the first subdivision may be arbitrarily set to be horizontal or vertical. Consequently, horizontal and vertical subdivision alternate strictly. Thus, space can be represented by dividing it into nested blocks by a certain order, position, and orientation of decomposition and by coloring terminals. The final result of a decomposition is an almost minimal set [De91a] of blocks that divide a given space into homogeneous regions, i.e in the case of images into blocks occupied by pixels of same color. In contrast to quadtrees, the blocks have no standard sizes, extensions, or locations. Additional advantages of puzzletrees are the saving of storage over an array representation (compression factor is much better than the one of quadtrees) and the provision of a global data structure in contrast to polygon representation.

Because our work is primarily concerned with recognition and classification tasks, puzzletrees are mainly considered as specifications for object structure descriptions. To this end, we concentrate on experiments that utilize and apply puzzletree descrip-tions for the creation of decision tree classifiers (similar to the approach proposed by [DB89]).

ACKNOWLEDGEMENT

Special thanks should be given to R. Bleisinger for the valuable hints to improve this paper. I am also grateful to R. Hoch, M. Malburg and T. Bayer for their suggestions.

REFERENCES

[BB82] Ballard, D.H. & Brown, C.M.: *Computer Vision*, Prentice-Hall (1982).

[DB89] Dengel , A. & Barth, G.: *ANASTASIL: A Hybrid Knowledge-based System for Document Layout Ana-lysis*, Proc. 11th IJCAI, Detroit, MI (1989), 1249-1254.

[De91a] Dengel, A.: *Puzzletrees: A Approach for Self-Adapting Structuring of Space Using Puzzletree-Encoding*, Proceedings Fourth International Symposium on AI, Canún, Mexico, November 1991.

[De91b] Dengel, A., *Object-oriented Representation of Image Space by Puzzletrees*, Proc. VCIP´91, Boston, MA, Nov. 1991.

[Hu78] Hunter, G.M.: *Efficient Computation and Data Struc-tures for Graphics*, Ph.D. dissertation, Dept. of Electr. Eng. & Comp. Science, Princeton University, Princeton, NJ.

[JT80] Janckins, C. & Tanimoto, S.K.: *Oct-trees and Their Use in Representing Three-Dimensional Objects*, Computer Graphics & Image Processing 14, 3 (Nov. 1980), 249-270.

[Kn80] Knowlton, K.: *Progressive Transmission of Grey-Scale and Binary Pictures by Simple, Efficient, and Lossless Encodimg Schemes*, Proc. IEEE 68, 7 (July 1980), 885-896.

[LL87] Li, S.-X. & Loew, M.H.: *The Quadcode and Its Aritmethic*, Com. ACM 30, 7 (July 1987), 621-626.

[McD87] McDermott, D.V.: *Spatial Reasoning*, in: S.C. Shapiro (ed.) Encyclopedia of Artificial Intelligence, John Wiley & Sons, Inc. (1987), pp. 863-870.

[KD76] Klinger, A. & Dyer, C.R.: *Experiments in picture representation using regular decomposition*, Computer Graphics & Image Processing 5 (1976), pp. 68-105.

[MD91] Mattos, N.M. & Dengel, A.: *The Role of Abstraction Concepts and Their Built-In Reasoning in Document Representation and Structuring*, submitted to IJCAI-91, Sidney, Australia, August 1991.

[Ru68] Rudovitz, D.: *Data Structures for Operations on Digital Images*, in: G.C. Cheng et al (eds.): Pictoral Pattern Recognition Thompson Book Co., Washington D.C. (1968), pp. 105-133.

[Sa81a] Samet, H.: *An Algorithm for Converting Rasters to Quadtrees*, IEEE PAMI 3, 1 (1981), pp. 93-95.

[Sa81b] Samet, H.: *Connected Component Labeling Using Quadtrees*, Journal of the ACM 28, 3 (July 1981), 487-501.

[Sa81c] Samet, H.: *Computing perimeters of images represented by quadtrees*, IEEE PAMI 3, 6 (Nov. 1981), 683-687

[Sa82] Samet, H.: *Neighbor finding techniques for image represented by quadtrees*, Computer Graph. & Image Proc. 18, 1 (Jan. 1982), 37-57

[Sa84] Samet, H.: *The Quadtree and Related Hierarchical Data Structures*, Computing Surveys, Vol. 16, No. 2 (June 1984), pp. 187-260.

[SW88a] Samet, H. & Webber, R.E.: *Hierarchical Data Structures and Algorithms for Computer Graphics, Part I: Fundamentals*, IEEE Computer Graphics & Applications (Mai 1988), pp. 48-68.

[SW88b] Samet, H. & Webber, R.E.: *Hierarchical Data Structures and Algorithms for Computer Graphics, Part II: Fundamentals*, IEEE Computer Graphics & Applications (July1988), pp. 59-75.

[Sa90] Samet, H.: *Hierarchical Spatial Data Structures for Image Databases*, Tutorial Manuscript, 10th ICPR, , Atlantic City (June 1990).

[SX89] Srihari, S.N. & Xiang, Z.: *Spatial Knowledge Representation*, IJPRAI, Vol. 3, No. 1 (1989), pp. 67-84.

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
67608 Kaiserslautern
FRG

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse oder per anonymem ftp von ftp.dfki.uni-kl.de (131.246.241.100) unter pub/Publications bezogen werden.
Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far are obtainable from the above address or per anonymous ftp from ftp.dfki.uni-kl.de (131.246.241.100) under pub/Publications.
The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

RR-92-49
Christoph Klauck, Ralf Legleitner, Ansgar Bernardi:
Heuristic Classification for Automated CAPP
15 pages

RR-92-50
Stephan Busemann:
Generierung natürlicher Sprache
61 Seiten

RR-92-51
Hans-Jürgen Bürckert, Werner Nutt:
On Abduction and Answer Generation through Constrained Resolution
20 pages

RR-92-52
Mathias Bauer, Susanne Biundo, Dietmar Dengler, Jana Koehler, Gabriele Paul: PHI - A Logic-Based Tool for Intelligent Help Systems
14 pages

RR-92-53
Werner Stephan, Susanne Biundo:
A New Logical Framework for Deductive Planning
15 pages

RR-92-54
Harold Boley: A Direkt Semantic Characterization of RELFUN
30 pages

RR-92-55
John Nerbonne, Joachim Laubsch, Abdel Kader Diagne, Stephan Oepen: Natural Language Semantics and Compiler Technology
17 pages

RR-92-56
Armin Laux: Integrating a Modal Logic of Knowledge into Terminological Logics
34 pages

RR-92-58
Franz Baader, Bernhard Hollunder:
How to Prefer More Specific Defaults in Terminological Default Logic
31 pages

RR-92-59
Karl Schlechta and David Makinson: On Principles and Problems of Defeasible Inheritance
13 pages

RR-92-60
Karl Schlechta: Defaults, Preorder Semantics and Circumscription
19 pages

RR-93-02
Wolfgang Wahlster, Elisabeth André, Wolfgang Finkler, Hans-Jürgen Profitlich, Thomas Rist:
Plan-based Integration of Natural Language and Graphics Generation
50 pages

RR-93-03
Franz Baader, Berhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, Enrico Franconi:
An Empirical Analysis of Optimization Techniques for Terminological Representation Systems
28 pages

RR-93-04
Christoph Klauck, Johannes Schwagereit:
GGD: Graph Grammar Developer for features in CAD/CAM
13 pages

RR-93-05
Franz Baader, Klaus Schulz: Combination Techniques and Decision Problems for Disunification
29 pages

RR-93-06
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux: On Skolemization in Constrained Logics
40 pages

RR-93-07
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux: Concept Logics with Function Symbols
36 pages

RR-93-08
Harold Boley, Philipp Hanschke, Knut Hinkelmann, Manfred Meyer: COLAB: A Hybrid Knowledge Representation and Compilation Laboratory
64 pages

RR-93-09
Philipp Hanschke, Jörg Würtz: Satisfiability of the Smallest Binary Program
8 Seiten

RR-93-10
Martin Buchheit, Francesco M. Donini, Andrea Schaerf: Decidable Reasoning in Terminological Knowledge Representation Systems
35 pages

RR-93-11
Bernhard Nebel, Hans-Juergen Buerckert: Reasoning about Temporal Relations: A Maximal Tractable Subclass of Allen's Interval Algebra
28 pages

RR-93-12
Pierre Sablayrolles: A Two-Level Semantics for French Expressions of Motion
51 pages

RR-93-13
Franz Baader, Karl Schlechta: A Semantics for Open Normal Defaults via a Modified Preferential Approach
25 pages

RR-93-14
Joachim Niehren, Andreas Podelski,Ralf Treinen: Equational and Membership Constraints for Infinite Trees
33 pages

RR-93-15
Frank Berger, Thomas Fehrle, Kristof Klöckner, Volker Schölles, Markus A. Thies, Wolfgang Wahlster: PLUS - Plan-based User Support Final Project Report
33 pages

RR-93-16
Gert Smolka, Martin Henz, Jörg Würtz: Object-Oriented Concurrent Constraint Programming in Oz
17 pages

RR-93-17
Rolf Backofen: Regular Path Expressions in Feature Logic
37 pages

RR-93-18
Klaus Schild: Terminological Cycles and the Propositional μ-Calculus
32 pages

RR-93-20
Franz Baader, Bernhard Hollunder: Embedding Defaults into Terminological Knowledge Representation Formalisms
34 pages

RR-93-22
Manfred Meyer, Jörg Müller: Weak Looking-Ahead and its Application in Computer-Aided Process Planning
17 pages

RR-93-23
Andreas Dengel, Ottmar Lutzy: Comparative Study of Connectionist Simulators
20 pages

RR-93-24
Rainer Hoch, Andreas Dengel: Document Highlighting — Message Classification in Printed Business Letters
17 pages

RR-93-25
Klaus Fischer, Norbert Kuhn: A DAI Approach to Modeling the Transportation Domain
93 pages

RR-93-26
Jörg P. Müller, Markus Pischel: The Agent Architecture InteRRaP: Concept and Application
99 pages

RR-93-27
Hans-Ulrich Krieger: Derivation Without Lexical Rules
33 pages

RR-93-28
Hans-Ulrich Krieger, John Nerbonne, Hannes Pirker: Feature-Based Allomorphy
8 pages

RR-93-29
Armin Laux: Representing Belief in Multi-Agent Worlds viaTerminological Logics
35 pages

RR-93-33
Bernhard Nebel, Jana Koehler: Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis
33 pages

RR-93-34
Wolfgang Wahlster:
Verbmobil Translation of Face-To-Face Dialogs
10 pages

RR-93-35
Harold Boley, François Bry, Ulrich Geske (Eds.): Neuere Entwicklungen der deklarativen KI-Programmierung — *Proceedings*
150 Seiten
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

RR-93-36
Michael M. Richter, Bernd Bachmann, Ansgar Bernardi, Christoph Klauck, Ralf Legleitner, Gabriele Schmidt: Von IDA bis IMCOD: Expertensysteme im CIM-Umfeld
13 Seiten

RR-93-38
Stephan Baumann: Document Recognition of Printed Scores and Transformation into MIDI
24 pages

RR-93-40
Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, Andrea Schaerf: Queries, Rules and Definitions as Epistemic Statements in Concept Languages
23 pages

RR-93-41
Winfried H. Graf: LAYLAB: A Constraint-Based Layout Manager for Multimedia Presentations
9 pages

RR-93-42
Hubert Comon, Ralf Treinen:
The First-Order Theory of Lexicographic Path Orderings is Undecidable
9 pages

RR-93-44
Martin Buchheit, Manfred A. Jeusfeld, Werner Nutt, Martin Staudt: Subsumption between Queries to Object-Oriented Databases
36 pages

RR-93-45
Rainer Hoch: On Virtual Partitioning of Large Dictionaries for Contextual Post-Processing to Improve Character Recognition
21 pages

RR-93-46
Philipp Hanschke: A Declarative Integration of Terminological, Constraint-based, Data-driven, and Goal-directed Reasoning
81 pages

DFKI Technical Memos

TM-92-01
Lijuan Zhang: Entwurf und Implementierung eines Compilers zur Transformation von Werkstückrepräsentationen
34 Seiten

TM-92-02
Achim Schupeta: Organizing Communication and Introspection in a Multi-Agent Blocksworld
32 pages

TM-92-03
Mona Singh:
A Cognitiv Analysis of Event Structure
21 pages

TM-92-04
Jürgen Müller, Jörg Müller, Markus Pischel, Ralf Scheidhauer:
On the Representation of Temporal Knowledge
61 pages

TM-92-05
Franz Schmalhofer, Christoph Globig, Jörg Thoben:
The refitting of plans by a human expert
10 pages

TM-92-06
Otto Kühn, Franz Schmalhofer: Hierarchical skeletal plan refinement: Task- and inference structures
14 pages

TM-92-08
Anne Kilger: Realization of Tree Adjoining Grammars with Unification
27 pages

TM-93-01
Otto Kühn, Andreas Birk: Reconstructive Integrated Explanation of Lathe Production Plans
20 pages

TM-93-02
Pierre Sablayrolles, Achim Schupeta:
Conlfict Resolving Negotiation for COoperative Schedule Management
21 pages

TM-93-03
Harold Boley, Ulrich Buhrmann, Christof Kremer:
Konzeption einer deklarativen Wissensbasis über recyclingrelevante Materialien
11 pages

TM-93-04
Hans-Günther Hein: Propagation Techniques in WAM-based Architectures — The FIDO-III Approach
105 pages

DFKI Documents

D-92-24
Jürgen Müller, Donald Steiner (Hrsg.):
Kooperierende Agenten
78 Seiten

D-92-25
Martin Buchheit: Klassische Kommunikations- und Koordinationsmodelle
31 Seiten

D-92-26
Enno Tolzmann:
Realisierung eines Werkzeugauswahlmoduls mit Hilfe des Constraint-Systems CONTAX
28 Seiten

D-92-27
Martin Harm, Knut Hinkelmann, Thomas Labisch: Integrating Top-down and Bottom-up Reasoning in COLAB
40 pages

D-92-28
Klaus-Peter Gores, Rainer Bleisinger: Ein Modell zur Repräsentation von Nachrichtentypen
56 Seiten

D-93-01
Philipp Hanschke, Thom Frühwirth:
Terminological Reasoning with Constraint Handling Rules
12 pages

D-93-02
Gabriele Schmidt, Frank Peters, Gernod Laufkötter: User Manual of COKAM+
23 pages

D-93-03
Stephan Busemann, Karin Harbusch(Eds.):
DFKI Workshop on Natural Language Systems: Reusability and Modularity - Proceedings
74 pages

D-93-04
DFKI Wissenschaftlich-Technischer Jahresbericht 1992
194 Seiten

D-93-05
Elisabeth André, Winfried Graf, Jochen Heinsohn, Bernhard Nebel, Hans-Jürgen Profitlich, Thomas Rist, Wolfgang Wahlster:
PPP: Personalized Plan-Based Presenter
70 pages

D-93-06
Jürgen Müller (Hrsg.):
Beiträge zum Gründungsworkshop der Fachgruppe Verteilte Künstliche Intelligenz
Saarbrücken 29.-30. April 1993
235 Seiten
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-93-07
Klaus-Peter Gores, Rainer Bleisinger:
Ein erwartungsgesteuerter Koordinator zur partiellen Textanalyse
53 Seiten

D-93-08
Thomas Kieninger, Rainer Hoch: Ein Generator mit Anfragesystem für strukturierte Wörterbücher zur Unterstützung von Texterkennung und Textanalyse
125 Seiten

D-93-09
Hans-Ulrich Krieger, Ulrich Schäfer:
TDL ExtraLight User's Guide
35 pages

D-93-10
Elizabeth Hinkelman, Markus Vonerden,Christoph Jung: Natural Language Software Registry
(Second Edition)
174 pages

D-93-11
Knut Hinkelmann, Armin Laux (Eds.):
DFKI Workshop on Knowledge Representation Techniques — Proceedings
88 pages

D-93-12
Harold Boley, Klaus Elsbernd, Michael Herfert, Michael Sintek, Werner Stein:
RELFUN Guide: Programming with Relations and Functions Made Easy
86 pages

D-93-14
Manfred Meyer (Ed.): Constraint Processing – Proceedings of the International Workshop at CSAM'93, July 20-21, 1993
264 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-93-15
Robert Laux: Untersuchung maschineller Lernverfahren und heuristischer Methoden im Hinblick auf deren Kombination zur Unterstützung eines Chart-Parsers
86 Seiten

D-93-20
Bernhard Herbig:
Eine homogene Implementierungsebene für einen hybriden Wissensrepräsentationsformalismus
97 Seiten

D-93-21
Dennis Drollinger:
Intelligentes Backtracking in Inferenzsystemen am Beispiel Terminologischer Logiken
53 Seiten