Image Anal Stereol 2009;28:45-61
Original Research Paper

LABELING OF N-DIMENSIONAL IMAGES WITH CHOOSABLE ADJACENCY OF THE PIXELS

KAI SANDFORT[1] AND JOACHIM OHSER[2]

[1]University of Karlsruhe, IWRMM, Engesserstraße 6, 76131 Karlsruhe, Germany; [2]University of Applied Sciences, Darmstadt, Schöfferstraße 3, 46295 Darmstadt, Germany
e-mail: sandfort@math.uni-karlsruhe.de, jo@h-da.de
(Accepted February 18, 2009)

ABSTRACT

The labeling of discretized image data is one of the most essential operations in digital image processing. The notions of an adjacency system of pixels and the complementarity of two such systems are crucial to guarantee consistency of any labeling routine. In to date's publications, this complementarity usually is defined using discrete versions of the Jordan-Veblen curve theorem and the Jordan-Brouwer surface theorem for 2D and 3D images, respectively. In contrast, we follow here an alternative concept, which relies on a consistency relation for the Euler number. This relation and all necessary definitions are easily stated in a uniform manner for the *n*-dimensional case. For this, we present identification and convergence results for complementary adjacency systems, supplemented by examples for the 3D case. Next, we develop a pseudo-code for a general labeling algorithm. The application of such an algorithm should be assessed with regard to our preceding considerations. A benchmark and a thorough discussion finish our article.

Keywords: adjacency of lattice points, complementarity, connectivity, labeling, run length encoding.

INTRODUCTION

Labeling of connected components ('objects') is one of the most important tools of image processing. It is the basis for the generation of object features as well as of some kind of filtering, *i.e.*, removing of noisy objects or holes in objects. The criteria for removing an object or a hole can be chosen extremely flexible based on the object features. The task of labeling (object filling, region detection) is to assign labels (mostly unsigned integers) to the pixels in such a way that all pixels belonging to a connected component of the image are assigned the same label, and pixels belonging to different components get different labels. Due to its importance in image processing, there is plenty of literature about labeling and techniques to control (and improve) the processing and memory demands. These can be tremendous for large images, which frequently arise in practice. Once again, the challenge lies in finding a compromise between usability, flexibility, and efficiency.

The prototype of labeling algorithms is the simple and well-known Rosenfeld-Pfaltz method (Rosenfeld and Pfaltz, 1966; Klette and Rosenfeld, 2004). Here, the image is scanned until a pixel x_k is found that has not yet been labeled. If there is no neighboring pixel labeled w. r. t. the chosen adjacency system, a new label is chosen for x_k. Otherwise, if there is a neighboring pixel x_j with the label ℓ_j, the label ℓ_j is assigned also to x_k. In the case of more than one neighboring pixels which have different labels, this is noted in a table of pairs of equivalent labels. A connected component is finally identified as the set of pixels belonging to the same equivalence class of labels. The table of pairs can become very large and may lead to problems in finding the equivalence classes; the time complexity of a corresponding algorithm is $\mathcal{O}(m \log m)$ with m being the number of pairs. Thus, several variants of the Rosenfeld-Pfaltz method have been developed which employ techniques to keep m as small as possible.

However, a reduction in complexity is achieved also by other strategies. For example, in Dillencourt *et al.* (1992) the authors start their analysis with describing a combination of the *Union-Find* method with weight-balancing and path-compression, which yields $\mathcal{O}(\tilde{m}\,\alpha(\tilde{m}))$; here, α is the inverse of Ackermann's function and $\tilde{m}$ denotes the number of *Find* and *Union* operations and is related to the variable m above. On the one hand, the function α grows extremely slowly, and on the other hand, the quantity $\tilde{m}$ does not necessarily evolve on pixel level (as m) since the *Union-Find* method can be equally applied to other image representations with *e.g.*, bintrees or run lengths (instead of pixels) as modules. The same holds for extensions of *Union-Find* (Dillencourt *et al.*, 1992) and related algorithms. The fact that some image representations, in particular run length encodings, are compatible with and facilitate the labeling w. r. t. an adjacency system provides the basis for our algorithm. Given the input image as a pixel array, a conversion into a different representation might be done as a

preprocessing in a labeling scheme. Its purpose could be a decomposition (Aguilera *et al.*, 2002) or a more compact representation (our method) of the image to allow an efficient data access and/or to reduce memory demands.

Many classical labeling methods, including the one by Rosenfeld and Pfaltz, are two-pass-techniques. Here, in the first pass preliminary labels are assigned and equivalences between labels are registered, and in the second pass these correspondences are resolved into equivalence classes. Either representatives of these classes (*e.g.*, their smallest elements) or consecutive integers assigned to them are then set as the final labels. The resolving step is a critical issue, and several methods have been proposed to handle it. An efficient solution is explained in Thurfjell *et al.* (1992). However, the basic idea to maintain and merge equivalences in a 1D array originated in a work by Hoshen and Koppelman (1976). Hoshen (1998) later on examined its use in image processing. An alternative and general solution, with special consideration of dismissing and reusing preliminary labels, is given in Dillencourt *et al.* (1992) (integrated in a *Union-Find* method), along with a detailed proof of correctness. While in the latter the final label is determined only in a second run, the methods cited before can be used to merge label classes on-the-fly, *i.e.*, as soon as an equivalence is detected, and by that allow a direct mapping of a preliminary label to the corresponding final one right after the first run through the input image.

A very different concept is followed by recursive labeling methods. Such methods completely avoid the setup of a mapping between preliminary and final labels and identify a complete object (connected component) at once. The information about connectedness is usually carried in the algorithms by design, rather than provided as a parameter or data structure. An early account of recursive detection of connected components is given in Hopcroft and Tarjan (1973). The underlying principle resembles that of filling algorithms. Recursive techniques can achieve an excellent performance as they often are accessible to the optimization potential of a compiler and a CPU. However, since the recursion depth heavily depends on the contents of the input image and so can not be predicted, they have to tackle the risk of a stack overflow. A smart approach to this is described by Martín-Herrero in Martín-Herrero (2004), see also Martín-Herrero (2007). It relies on a quasi-run length encoding in one dimension and recursion in the remaining dimensions of the input image, which gains a significant decrease of the recursion depth compared to recursion in all dimensions. The resulting algorithm yields performance values which seem to be (among) the best of today's labeling algorithms and will serve as a benchmark in our paper.

We now explain what we mean by quasi-run length encoding above and put the bridge to our labeling method. We assume a binary input image and consider the labeling of its foreground pixels. In the code by Martín-Herrero, starting with a single pixel *P*, all foreground pixels which are consecutive in a given direction and trivially connected to *P* are found by iterative scanning and assigned a label differing from the foreground value. Afterwards, the same is recursively performed for all foreground neighbors (according to the notion of connectedness) of all pixels of the run. Each neighbor acts as the starting pixel *P* of the next stage of the recursion. In this way, all pixels of a connected component are reached and every pixel is assigned a label only once. However, in searching for the foreground neighbors of all members of a run, a lot of pixels are accessed more than once. Although the query of a label is a simple and fast operation, this could be seen as a (small) shortcoming. It is not possible here to restrict the check of the neighbors to a few of the pixels of a run since otherwise not all pixels of an object might be captured by recursion. Anyway, this hybrid method is a single-pass-technique in the sense that the input image as a whole has to be passed only once in order to labelc orrectly all connected components.

For our algorithm, we take a perspective which is somehow complementary to the one just described. We start with an iterative run length encoding of all foreground pixels in the input image. This preprocessing has been mentioned before as a conversion from one image representation into another. It guarantees that all foreground pixels are registered a priori and makes it possible to detect all correspondences by checking only the boundaries, *i.e.*, the start and end point of each run. The overall labeling algorithm permits a proper control of the stack and memory demands. We also believe that the representation of the image as an array of run lengths is advantageous for a further image processing as well as analysis, *e.g.* a fast extraction of object features. On the flipside, however, it lacks some of the parallel processing and real time features of the method by Martín-Herrero if the image is given as a pixel array. Both methods exploit a property of any reasonable definition of adjacency, namely that pixels which are consecutive in a direction are connected. This is reflected below in the inclusion Eq. 3. Here, all admissible directions are implicitly defined by the lattice or the pixel array, respectively.

Finally, we remark that the introduction in Thurfjell *et al.* (1992) contains a short survey of different labeling algorithms, and Chapter 6 in Ritter and Wilson (2001) formalizes labeling operations on the basis of image algebra.

The outline of the paper is as follows. First we give a short introduction to lattices (point lattices, grids). For a general definition of pixel neighborhood, we follow the approaches of Ohser *et al.* (2002; 2003); Schladitz *et al.* (2006) and state the concept of adjacency on *n*-dimensional lattices and pairs of complementary adjacency systems in. An important insight is that, in 3D, complementarity according to a consistency relation for the Euler number differs from the complementarity as conventionally defined by the discrete Jordan-Brouwer theorem. Furthermore, we recall the notion of connectedness in the continuous case and give a definition of connected components in a lattice (discrete case) w. r. t. to a given adjacency system. This finishes our theoretical investigation. Before explaining the labeling algorithm, we introduce some basic mathematical objects to handle images and intend to clarify the link between the background from the previous sections and a practical implementation. Afterwards, we explain a run length encoding and a labeling procedure by means of pseudo-codes. For a convenient illustration, important and frequently used functionality is encapsulated here in auxiliary routines. A further section is devoted to a benchmark and a test of our algorithm on application data from material testing. We finally address some applications and possible extensions of the algorithm in a short conclusion.

ADJACENCY OF HOMOGENEOUS LATTICES AND EULER NUMBER

An *n*-dimensional homogeneous lattice is a subset $\mathbb{L}^n$ of the *n*-dimensional Euclidean space $\mathbb{R}^n$ with

$$\mathbb{L}^n = \{x \in \mathbb{R}^n : x = \sum_{i=1}^{n} \lambda_i u_i,\ \lambda_i \in \mathbb{Z}\} = U\mathbb{Z}^n\,, \quad (1)$$

where $u_1,\dots,u_n \in \mathbb{R}^n$ form a basis of $\mathbb{R}^n$, $U = (u_1,\dots,u_n)$ is the matrix of column vectors, and $\mathbb{Z}$ ist the set of integers. By $C = U\cdot[0,1]^n$ we denote the unit cell of $\mathbb{L}^n$. Let $\mathscr{F}^0$ be the set of vertices of a polyhedron, in particular $\mathscr{F}^0(C) = U\cdot\{0,1\}^n$.

In the literature, the adjacency of lattice points (or pixels) is usually characterized by a *neighborhood graph* Γ, and the complementarity of adjacencies is defined via the Jordan-Brouwer surface theorem (Lachaud and Montanvert, 2000). Here we use an alternative concept of adjacency systems based on the Euler number of a discretization, thoroughly introduced in Nagel *et al.* (2000) and Ohser *et al.* (2002; 2003).

DISCRETIZATION WITH RESPECT TO AN ADJACENCY SYSTEM

Let $\mathbb{L}^n$ be a lattice with the basis $\{u_1,\dots,u_n\}$ and the unit cell C. The vertices of C are indexed, and we write $x_j = \sum_{i=1}^n \lambda_i u_i$, $\lambda_i \in \{0,1\}$, with the index $j = \sum_{i=1}^n 2^{i-1}\lambda_i$. Clearly, the unit cell C has 2^n vertices, $x_i \in \mathscr{F}^0(C)$, $i = 0,\dots,2^n-1$. In a similar way we introduce the index of a subset $\xi \subseteq \mathscr{F}^0(C)$. Let 1 denote the indicator function of a set, *i. e.* $1(x\in\xi) = 1$ if $x \in \xi$ and $1(x\in\xi) = 0$ otherwise. The index ℓ is assigned, and we write ξ_ℓ if

$$\ell = \sum_{j=0}^{2^n-1} 2^j \cdot 1(x_j \in \xi)\,, \quad (2)$$

for all $\xi \subseteq \mathscr{F}^0(C)$, where ℓ takes the integer values between 0 and $\nu = 2^{2^n}-1$. The ξ_ℓ are local pixel configuration of the foreground of a binary image, and $\nu+1$ is the number of different configurations.

We introduce the convex hulls $F_\ell = \operatorname{conv}\xi_\ell$ forming convex polytopes with $F_\ell \subseteq C$ and $\mathscr{F}^0(F_\ell) \subseteq \mathscr{F}^0(C)$, $\ell = 1,\dots,\nu$. Let $\mathscr{F}^j(F)$ denote the set of all j-dimensional faces of a convex polytope F. For a set $\mathbb{F}$ of convex polytopes we set $\mathscr{F}^j(\mathbb{F}) = \bigcup\{\mathscr{F}^j(F) : F \in \mathbb{F}\}$. Now we are able to equip the lattice $\mathbb{L}^n$ with a (homogeneous) adjacency system defining the neighborhood of lattice points.

Definition 1 *Let $\mathbb{F}_0 \subseteq \{F_0,\dots,F_\nu\}$ be a set of convex polytopes $F_\ell = \operatorname{conv}\xi_\ell$, and let $\mathbb{F}$ be the union over all lattice translations of $\mathbb{F}_0$, that is $\mathbb{F} = \bigcup_{x\in\mathbb{L}^n}\mathbb{F}_0 + x$. If*

(i) $\emptyset \in \mathbb{F}_0, C \in \mathbb{F}_0$,

(ii) if $F \in \mathbb{F}_0$ then $\mathscr{F}^i(F) \subset \mathbb{F}_0$ for $i = 0,\dots,\dim F$,

(iii) if $F_i, F_j \in \mathbb{F}_0$ and $\operatorname{conv}(F_i \cup F_j) \notin \mathbb{F}_0$ then $F_i \cap F_j$, $\overline{F_i \setminus F_j}$, $\overline{F_j \setminus F_i} \in \mathbb{F}_0$ (where the bar denotes the topological closure),

(iv) if $F_{i_1},\dots,F_{i_m} \in \mathbb{F}_0$ and $F = \bigcup_{j=1}^m F_{i_j}$ is convex then $F \in \mathbb{F}_0$, $m = 2,\dots,\nu$,

then the system $\mathbb{F}_0$ is called a local adjacency system *and $\mathbb{F}$ is said to be an* adjacency system *of the lattice $\mathbb{L}^n$.*

The pair $\Gamma = (\mathscr{F}^0(\mathbb{F}), \mathscr{F}^1(\mathbb{F}))$ is the *neighborhood graph* of $\mathbb{F}$ consisting of the set $\mathscr{F}^0(\mathbb{F})$ of nodes and the set $\mathscr{F}^1(\mathbb{F})$ of edges. The order of the nodes is called the *connectivity* of $\mathbb{L}^n$.

In the simplest case, where the adjacency system is generated from the unit cell C, the order of the nodes is $2n$, and we write $\mathbb{F}_{2n} = \bigcup_{x\in\mathbb{L}^n}\bigcup_{j=0}^{n}\mathscr{F}^j(C+x)$. The maximum adjacency system consisting of the convex hulls of all point configurations provides a κ-adjacency with $\kappa = 3^n - 1$, $\mathbb{F}_\kappa = \bigcup_{x\in\mathbb{L}^n}\{F_0 + x, \ldots, F_\nu + x\}$. Notice that for all adjacency systems $\mathbb{F}$ on $\mathbb{L}^n$ the inclusion

$$\mathbb{F}_{2n} \subseteq \mathbb{F} \subseteq \mathbb{F}_\kappa \tag{3}$$

holds. Now we recall the adjacency systems on $\mathbb{L}^3$ considered in detail in Ohser *et al.* (2002; 2003).

– *6-adjacency.* The 6-adjacency is used as a standard in image processing. It is generated from the unit cell C, $\mathbb{F}_6 = \bigcup_{x\in\mathbb{L}^3}\bigcup_{j=0}^{3}\mathscr{F}^j(C+x)$.

– *14.1-adjacency.* This adjacency system is generated from the tessellation of C into the 6 tetrahedra F_{139}, F_{141}, F_{163}, F_{177}, F_{197}, and F_{209} being the convex hulls of the configurations

That is $\mathbb{F}_0$, consists of all j-faces of the tetrahedra, $j = 0,\ldots,3$, and their convex unions. The edges of the corresponding neighborhood graph Γ are the edges of C, the face diagonals of C containing the origin 0, the space diagonal of C containing 0, and all their lattice translations. The order of the nodes of Γ is 14.

– *14.2-adjacency.* The 14.2-adjacency system is generated from the tetrahedra F_{43}, F_{141}, F_{147}, F_{169}, F_{177}, and F_{212}, which are the convex hulls of

The corresponding neighborhood graph Γ differs from that one for 14.1 in the choice of one face diagonal of C such that it does not contain 0.

– *26-adjacency.* The system $\mathbb{F}_{26}$ is the maximum adjaceny system on $\mathbb{L}^3$, $\mathbb{F}_{26} = \mathbb{F}_\kappa$.

Based on the definition of adjacency we introduce the $\mathbb{F}$-discretization of a set.

Definition 2 *The discretization $X \sqcap \mathbb{F}$ of a compact subset $X \subset \mathbb{R}^n$ w. r. t. a given adjacency system $\mathbb{F}$ is defined as the union of all j-faces of the elements of $\mathbb{F}$ for which all the vertices hit X, i.e.,*

$$X \sqcap \mathbb{F} = \bigcup\{F \in \mathbb{F} : \mathscr{F}^0(F) \subseteq X\} . \tag{4}$$

It is important to realize that in particular for higher-dimensional images the connectivity of the pixels and, hence, the labeling can heavily depend on the choice of the adjacency system. The number of neighbors of a pixel in an n-dimensional image can range from $2n$ to $3^n - 1$. For the 2d case the conectivity ranges from 4 to 8 while in the 3d case the extremal choices are 6- and 26-connectivity. As a consequence of the wide range of the number of neighbors, the neighborhood of pixels should be chosen very carefully in dependence of the dimensionality of the image, the lateral resolution, the image data, and the aims of processing and analysis (Schladitz *et al.*, 2006).

EULER NUMBER AND COMPLEMENTARITY OF ADJACENCY SYSTEMS

Since the set $X \sqcap \mathbb{F}$ forms a (not necessarily convex) polyhedron, the number $\sharp\mathscr{F}^j(X \sqcap \mathbb{F})$ of elements of $\mathscr{F}^j(X \sqcap \mathbb{F})$ is finite and, therefore, the Euler number $\chi(X \sqcap \mathbb{F})$ can be computed via the Euler-Poincaré formula,

$$\chi(X \sqcap \mathbb{F}) = \sum_{j=0}^{n}(-1)^j \sharp\mathscr{F}^j(X \sqcap \mathbb{F}) . \tag{5}$$

A local version of Eq. 5 is given in Schladitz *et al.* (2006).

It is well-known from image processing that if one chooses an adjacency system $\mathbb{F}$ on the discretization of X, then there is implicitly chosen a system $\mathbb{F}_c$ on the discretization of the complementary set X^c. In other words, if the 'foreground' $X \cap \mathbb{L}^n$ is connected w. r. t. $\mathbb{F}$, then the 'background' $X^c \cap \mathbb{L}^n$ must be connected w. r. t. $\mathbb{F}_c$. For $n > 2$ it is not sufficient to consider connectivity, and further criteria have to be regarded. In the following we introduce 'complementarity' by means of the Euler number of the discretization $X \sqcap \mathbb{F}$.

Notice that the complement X^c of X is unbounded. Thus, the Euler number $\chi\left(\overline{X^c}\right)$ of its topological closure may be defined by Hadwiger's recursive definition (Schneider, 1993, p. 175).

Definition 3 *The pair $(\mathbb{F}, \mathbb{F}_c)$ is called a pair of complementary adjacency systems if $(X \sqcap \mathbb{F}) \cap (X^c \sqcap \mathbb{F}_c) = \emptyset$ and*

$$\chi(X \sqcap \mathbb{F}) = (-1)^{n+1}\chi(X^c \sqcap \mathbb{F}_c) \tag{6}$$

holds for all compact $X \subset \mathbb{R}^n$. An adjacency system $\mathbb{F}$ is called self-complementary if $\chi(X \sqcap \mathbb{F}) = (-1)^{n+1}\chi(X^c \sqcap \mathbb{F})$ holds for all compact X.

For $n = 3$ there exist 3 pairs of complementary adjacency systems: $(\mathbb{F}_6, \mathbb{F}_{26})$, $(\mathbb{F}_{14.1}, \mathbb{F}_{14.1})$ and $(\mathbb{F}_{14.2}, \mathbb{F}_{14.2})$. That is, the 6-adjacency is complementary to the 26-adjacency and there are known two self-complementary adjacency systems, the 14.1-adjacency and the 14.2-adjacency, see Ohser *et al.* (2002; 2003).

Consider a lattice $\mathbb{L}^3$ equipped with a neighborhood graph $\Gamma' = (\mathbb{L}^3, \mathscr{F}^1)$ where the system of edges $\mathscr{F}^1$ may consist of all edges and face diagonals of the cells of $\mathbb{L}^3$. The order of the nodes of Γ' is 18 and, hence, the adjacency is called the 18-adjacency (which is widely used in image processing). The 18-adjacency is 'Jordan-Brouwer-complementary' to the 6-adjacency generated solely from the edges of the lattice cells (Lachaud and Montanvert, 2000). However, Eq. 6 does not hold for the pair $(\mathbb{F}_{18}, \mathbb{F}_6)$ and hence it is not complementary in the sense of Definition 3 (Schladitz *et al.*, 2006). This shows that in higher dimensions ($n > 2$) the 'Jordan-Brouwer-complementarity' differs from that of Definition 3. The criteria in Definition 3 seam to be stronger than that of the 'Jordan-Brouwer-complementarity'.

MULTIGRID CONVERGENCE

Now we consider the relationship between the Euler number of a compact set $X \subset \mathbb{R}^n$ and the Euler number of its discretization. It can not be expected that $\chi(X) = \chi(X \sqcap \mathbb{F})$ for all compact sets X, but if X has a sufficiently smooth surface, the Euler number of $X \sqcap \mathbb{F}$ converges to the Euler number of X for increasing lateral resolution. Here 'smooth' is defined by morphological opening and morphological closure. The set X is called morphologically open w. r. t. a set $A \subset \mathbb{R}^n$ if X is invariant w. r. t. opening with A, $X \circ A = X$. Here the opening is defined by $X \circ A = (X \ominus \check{A}) \oplus A$, and $X \ominus A = (X^c \oplus A)^c$ is the Minkowski subtraction. Analogously, the set X is called morphologically closed w. r. t. A if $X \bullet A = X$ where $X \bullet A = (X \oplus \check{A}) \ominus A$ is the morphological closure with A. Morphological regularity of X means that there is an $\varepsilon > 0$ such that X is morphologically open as well as morphologically closed w. r. t. a ball B_ε of radius ε.

Theorem 1 *Let $(\mathbb{F}, \mathbb{F}_c)$ be a pair of complementary adjacency systems on $\mathbb{L}^n$. If $X \subset R^n$ is morphologically closed w. r. t. all edges $F \in \mathscr{F}^1(\mathbb{F})$ and morphologically open w. r. t. all $F \in \mathscr{F}^1(\mathbb{F}_c)$, then*

$$\chi(X) = \chi(X \sqcap \mathbb{F}) \quad \textit{and} \quad \chi(X^c) = \chi(X^c \sqcap \mathbb{F}_c) \, .$$

A proof is given in Ohser *et al.* (2002). Notice that a set X fulfilling the last condition is polyconvex and, hence, its Euler number exists. However, this condition for X is very strong, it depends on $\mathbb{F}$ and, hence, it will not be fulfilled in most applications. Thus, we consider a more natural condition for X. Let B_ε be a (small) ball of radius ε. From Theorem 1 we obtain the following lemma.

Lemma 1 *Let $(\mathbb{F}, \mathbb{F}_c)$ be a pair of complementary adjacency systems on $\mathbb{L}^n$. Then $a\mathbb{F}$ is an adjacency system on $a\mathbb{L}^n$, $a > 0$, and it is*

$$\lim_{a \to 0} \chi(X \sqcap a\mathbb{F}) = \chi(X) \tag{7}$$

for all compact and morphologically regular sets X.

This means that the Euler number is convergent for morphologically regular sets (multigrid convergence). The proof of this lemma follows from the fact that if X is morphologically regular, there exists an $a > 0$ such that $\chi(X \bullet F) = \chi(X)$ for all $F \in a\mathbb{F}$ and $\chi(X \circ F) = \chi(X)$ for $F \in a\mathbb{F}_c$, and choose an $a > 0$ such that $F \subset B_\varepsilon$ for all $F \in a\mathbb{F} \cup a\mathbb{F}_c$.

CONNECTEDNESS

In order to describe a labeling algorithm, it is necessary to introduce the notions of 'connectivity' and 'connected component'. These are provided by topology. Azriel Rosenfeld introduced a digital topology on $\mathbb{L}^2$ (Rosenfeld, 1970). He defined connectedness on lattices and stated a discrete Jordan-Veblen curve theorem. The definition of connectedness can simply be extended to n-dimensional lattices (Lachaud and Montanvert, 2000). Here we introduce connectedness based on adjacency of lattice points.

CONTINUOUS CASE

Firstly we consider the continuous case and introduce connectivity for the Euclidean space $\mathbb{R}^n$. The connected components of a bounded set $X \subset \mathbb{R}^n$ can be considered as the equivalence classes of $X \subseteq \mathbb{R}^n$ w. r. t. an appropriately chosen equivalence relation $\sim$ defined for point pairs in $\mathbb{R}^n$.

Definition 4 *A set $X \subset \mathbb{R}^n$ is said to be connected if for all subsets $X_1, X_2 \subseteq X$ with $X_1 \cup X_2 = X$ it follows that $\overline{X}_1 \cap X_2 \neq \emptyset$ or $X_1 \cap \overline{X}_2 \neq \emptyset$.*

This definition of connectivity is closely related to path-connectivity. A path in $\mathbb{R}^n$ is a continuous mapping $f : [0,1] \mapsto \mathbb{R}^n$. If $f(0) = x$ and $f(1) = y$, $x, y \in \mathbb{R}^n$, then f is called a path from x to y.

Definition 5 *A non-empty set X is called path-connected if for every $x,y \in X$ there exists a path f from x to y such that $f(\cdot) \subseteq X$.*

It is well-known that every path-connected set X is also connected. Furthermore, if X is open and connected, it is also path-connected. Obviously, a connected set X is not necessarily path-connected. For example, the curve of the function $\sin(1/t)$ is connected, but not path-connected. More precisely, the set $X = \{(t, \sin\frac{1}{t}) : t \in \mathbb{R} \setminus \{0\}\} \cup \{(0,t) : t \in [-1,1]\}$ is connected, but not path-connected (Schmitt, 1998).

We write $x \sim y$ for path-connected points $x,y \in \mathbb{R}^n$. It can be shown that the binary relation $\sim$ is an equivalence relation, *i. e.* $\sim$ is reflexive, symmetric, and transitive. The equivalence classes $X_1,\ldots,X_m$ of X under $\sim$ are called path components of X. For more details see, *e.g.*, Armstrong (1997) and Rotman (1993).

DISCRETE CASE

Connectedness in a discretization is closely related to adjacency of lattice points. Hence, we consider a homogeneous lattice $\mathbb{L}^n$ equipped with a pair of complementary adjacency systems $(\mathbb{F},\mathbb{F}_c)$. Let x and y be lattice points, $x,y \in \mathbb{L}^n$. A discrete path from x to y w. r. t. the adjacency system $\mathbb{F}$ is a sequence of lattice points $(x_i)_{i=0}^m \subset \mathbb{L}^n$, $m \in \mathbb{N}$, with $x_0 = x$, $x_m = y$, and $[x_{i-1},x_i] \in \mathbb{F}$, $i = 1,\ldots,m$.

A non-empty discrete set $Y \subseteq \mathbb{L}^n$ is called path-connected w. r. t. $\mathbb{F}$ if $\sharp Y = 1$ or if for all pairs $(x,y) \in Y^2$ with $x \neq y$ there exists a discrete path w. r. t. $\mathbb{F}$ from x to y. Connectedness w. r. t. $\mathbb{F}$ in Y is an equivalence relation.

Definition 6 *Let $\mathbb{L}^n$ be a homogeneous lattice equipped with an adjacency system $\mathbb{F}$, and let $Y \subseteq \mathbb{L}^n$ be a discrete set. The equivalence classes $Y_1,\ldots,Y_m \subseteq Y$, $m \geq 1$, defined through the connectedness w. r. t. $\mathbb{F}$ are called the connected components of Y.*

We will use the notation $Y_{\mathbb{F}} = \{Y_1,\ldots,Y_m\}$ for the set of equivalence classes of Y w. r. t. $\mathbb{F}$. The following Lemma links the equivalence classes of a set $X \subset \mathbb{R}^n$ to the equivalence classes of its digitisation $X \cap \mathbb{L}^n$.

Lemma 2 *Let $(\mathbb{F},\mathbb{F}_c)$ be a pair of complementary adjacency systems on $\mathbb{L}^n$, and let X be a compact and morphologically regular subset of $\mathbb{R}^n$ with the set of equivalence classes $\{X_1,\ldots,X_m\}$ under $\sim$. Then there is a constant $b > 0$ such that*

$$(X \cap a\mathbb{L}^n)_{a\mathbb{F}} = \{X_1 \cap a\mathbb{L}^n,\ldots,X_m \cap a\mathbb{L}^n\} \qquad (8)$$

for all a with $0 < a < b$.

Proof. If X is morphologically closed w. r. t. B_ε, $\varepsilon > 0$, then $X_i \cap X_j = \emptyset$ implies that

$$\inf\{\|x_i - x_j\| : x_i \in X_i,\ x_j \in X_j\} > \varepsilon.$$

Now we choose b such that $\frac{b}{2}(C \oplus \check{C}) \subset B_\varepsilon$, where C is the unit cell of $\mathbb{L}^n$. Then $X_i \sqcap a\mathbb{F}$ and $X_j \sqcap a\mathbb{F}$ are disjoint and, thus, there is no discrete path w. r. t. $\mathbb{F}$ connecting $X_i \cap a\mathbb{L}^n$ and $X_j \cap a\mathbb{L}^n$.

On the other hand, since X_i is morphologically open w. r. t. B_ε, then for each path f in X_i exists a path g with $f \subset g \oplus B_\varepsilon \subseteq X_i$. Since $(g \oplus B_\varepsilon) \cap a\mathbb{L}^n$ is path-connected w. r. t. $a\mathbb{F}$, it follows that $X_i \cap a\mathbb{L}^n$ is path-connected w. r. t. $a\mathbb{F}$, too. □

From Lemma 2 it follows that for sufficiently high lateral lattice resolution the equivalence classes of $(X \cap a\mathbb{L}^n)$ are independent of the choice of the adjacency system. However, this holds only for sets X with sufficiently smooth surface. In general, the equivalence classes of $X \cap \mathbb{L}^n$ depend on $(\mathbb{F},\mathbb{F}_c)$.

LABELING WITH CHOOSABLE ADJACENCY

In this section, we consider an implementation of a general and customizable labeling algorithm. Essential for this and its performance is the exploitation of the fact that runs are naturally connected and as such part of the same component, *cf.* the inclusion Eq. 3. In particular, this complies with all local adjacency systems given above for the case $n = 3$. We begin with a short description of the labeling procedure. Then we explain the necessary variables and data structures and present the procedures as easy-to-translate C-style pseudo-codes.

BASICS

We anticipate that an image $\mathbb{I} \subset \mathbb{L}^n$ is a finite discrete set of lattice points (pixels) with an associated binary-valued color mapping $v : \mathbb{I} \mapsto \{\text{BG\_COL}, \text{FG\_COL}\}$, where BG_COL ≤ 0 stands for the value of a background color, and FG_COL is the value of a foreground color. We simply speak about foreground and background pixels (of $\mathbb{I}$) and by that mean those elements x of $\mathbb{I}$ for which $v(x) = \text{FG\_COL}$ and $v(x) = \text{BG\_COL}$, respectively. Let $Y = \{x \in \mathbb{I} : v(x) = \text{FG\_COL}\}$ denote the set of foreground pixels.

Preceding the labeling, our algorithm does a run length encoding of the foreground pixels of $\mathbb{I}$ to identify Y beforehand and to allow a compact representation. As a *run* we consider a set of

consecutive pixels in a certain direction of $\mathbb{I}$. All admissible directions are indicated here by the vectors $u_1,\ldots,u_n$ of the underlying lattice $\mathbb{L}^n$, see Eq. 1. The run length encoding assigns either a single placeholder label to all runs or a unique preliminary label to each run. This depends on whether, in a subsequent labeling, labels should be propagated to adjacent runs or not. During the labeling, these labels are systematically overwritten in a way which allows to find the connected components in Y.

Using the notation introduced above, in formal terms a labeling w. r. t. an adjacency system $\mathbb{F}$ is a mapping $\mathscr{L}_{\mathbb{F}} : \mathbb{I} \mapsto \mathbb{N}_0$ defined by

$$\mathscr{L}_{\mathbb{F}}(x) = \begin{cases} f(i) & \text{for } x \in Y_i \,, \\ 0 & \text{otherwise, } \textit{i. e. } x \in \mathbb{I}\backslash Y \,, \end{cases}$$

where $f : \{1,\ldots,m\} \mapsto \mathbb{N}$ is a fixed function with $f(i) \neq f(j)$ for $i \neq j$. For simplicity, we let $f = id$ in the following.

To 'cluster' the runs which belong to the same connected component, equivalences between their preliminary labels with regard to $\mathbb{F}$ (also called 'correspondences') have to be registered and processed such that finally all runs making up the i-th connected component are assigned the unique integer $f(i) = i$ as a final label. The central factor in finding all correspondences in an efficient way is the relation of Eq. 3, which implies that the foreground neighbors of only the start and endpoint of each run have to be examined.

Now, we consider some basic mathematical objects to handle images. Let $X \subset \mathbb{R}^n$ be as in Lemma 2, $M \subseteq \bar{M} = \{1,\ldots,n\}$ and U_M be a matrix consisting of the column vectors u_j with $j \in M$. In particular, it holds $U\mathbb{Z}^n = U_{\bar{M}}\mathbb{Z}^n$. For a vector v, let v_j denote its j-th element. We define

the window: $W = UR^n$ with $R^n = \{r \in \mathbb{R}^n : 0 \leq r_{(j)} \leq d_{(j)} \text{ for } 1 \leq j \leq n\}$ for a fixed $d \in \mathbb{N}^n$,

the image: $\mathbb{I} = \mathbb{L}^n \cap W = U(\mathbb{Z}^n \cap R^n)$ where $\mathbb{L}^n = U\mathbb{Z}^n$,

the color mapping: $v : \mathbb{I} \mapsto \{\text{BG\_COL}, \text{FG\_COL}\}$ with

$$v(x) = \begin{cases} \text{FG\_COL} & \text{for } x \in \mathbb{I} \cap X \\ \text{BG\_COL} & \text{for } x \in \mathbb{I}\backslash X \end{cases},$$

the set of foreground pixels: $Y = \mathbb{I} \cap X$,

projections: $P_M(\mathbb{L}^n) = U_M\mathbb{Z}^{\sharp M} \subseteq \mathbb{L}^n$, and

a subimage: $S_{M,x} = (x + P_M(\mathbb{L}^n)) \cap W \subseteq \mathbb{I}$ for $x \in \mathbb{I}$.

Here, $\sharp M$ denotes the number of elements in M. Associated with any set $A \subseteq \mathbb{I}$ is the color mapping $v|_A$. As special cases, we mention that a *slice* is a subimage $S_{M,x}$ with $\sharp M = 2$, and a *line* is a subimage $S_{M,x}$ with $\sharp M = 1$. Clearly, every line is the intersection of two slices $S_{M_1,x}$ and $S_{M_2,x}$ with $\sharp(M_1 \cap M_2) = 1$. Note that $S_{M,x} = S_{M,y}$ for $x - y \in P_M(\mathbb{L}^n)$.

The following aspects will be important:

- For a n-dimensional image $\mathbb{I}$ there are $2^n n!$ possibilities to scan through $\mathbb{I}$ along its n lattice directions. The factorial in this term originates from the scanning order of the direction indices, and the factor 2^n accounts for the orientations of passing the directions. From now on, we assume that these orientations coincide with those of the basis vectors $u_1,\ldots,u_n$ such that $n!$ possibilities remain. The scanning order is given by the *ranks* of the directions, where the first scanning direction has rank 0, the second one rank 1, *etc*.
- Runs of pixels are detected and coded in the rank 0-direction. For an anisotropic set $X \cap W$ (with discrete analogon Y), the processing speed of our labeling algorithm depends on the choice of the rank 0-direction since it determines the number of runs.

Now we describe the main data structures which we use in our pseudo-codes for the run length encoding and the labeling. To simplify understanding the code, scalar variables begin with a small letter and data structures including vectors begin with a capital letter. All vectors are assumed to provide the methods `add()` to add an element, `at()` to access the element with the index given by the argument (starting from 0), `init()` to set all elements to the value of the argument, `resize()` to resize the vector to the length given by the argument and `size()` to query the number of elements. The $i+1$-th element can be accessed alternatively by the `[i]`-operator, following the name of the vector.

The main vectors, arrays and structures used in our run length encoding and labeling procedures are `Image`, `ImgSize`, `Rank`, `Index`, `RleRun`, `RleLine` and `Neighbors`.

By `Image` we denote the pixel array for the image data $\{(x, v(x)) : x \in \mathbb{I}\}$, precisely the entry for the pixel $x \in \mathbb{I}$ has the value $v(x)$. We assume that the entry is accessed by `Image`$[x_1]\ldots[x_n]$ where here $(x_1,\ldots,x_n)$ is the coordinate vector for x w. r. t. the basis $\{u_1,\ldots,u_n\}$, *i. e.* $x = x_1u_1 + \ldots + x_nu_n$. In an implementation of an algorithm capable to handle images of an arbitrary dimensionality n which is unknown initially, the addressing of pixels is a bit

more complex since the above style is inappropriate for unknown n. However, this is not an issue of interest here, so for illustration we choose the simple form. The vector `ImgSize` stores the dimensions of `Image`. It is the pendant of the vector d in the definition of the window from the model. `Rank` is a vector of length n and stores the ranks of the lattice directions. The value `Rank[i]` with $i \in \{0,\ldots,n-1\}$ is the rank of the direction with index i (indicated by u_i). The scanning order is `Rank[0]`,...,`Rank[n-1]`. `Index` is the complementary vector (of length n) of `Rank`, which maps the rank of a direction to its index, *i.e.* `Index[Rank[i]] = i` and `Rank[Index[r]] = r` for $0 \leq i,r < n$. The structure `RleRun`, holding the data of a single run, consists of the attribute `label` and the two-element vector `Pos`. The value `Pos[0]` is the coordinate of the start pixel of the run in the rank 0-direction, and `Pos[1]` is the corresponding coordinate of its end pixel. The procedure for the run length encoding writes a preliminary label into `label`. Finally, `RleLine` is a vector of `RleRun`s and represents a run length encoded line $S_{\{i_0\},x}$ of the image $\mathbb{I}$. Here, i_0 denotes the index of the rank 0-direction. The size of `RleLine` depends on the contents of the image, the index i_0 and the line with representative $x \in \mathbb{I}$. The array `Neighbors` represents a vector of relative coordinate vectors for the neighbors of a pixel according to the governing notion of connectedness. This means that the neighbors of a pixel $x \in \mathbb{I}$ have coordinate vectors of the form $(x_1,\ldots,x_n)+$ `Neighbors[j]`, where $j \in \{0,\ldots,\kappa-1\}$, κ is the number of neighbors and `Neighbors[j]` is a coordinate vector of length n with `Neighbors[j][k]` $\in \{-1,0,1\}$ for $k \in \{0,\ldots,n-1\}$. The order in which the neighbors are stored in `Neighbors` does not matter. This array represents the neighborhood graph of an adjacency system accordinge to Definition 1. Due to the Eq. 3, `Neighbors` is expected to contain the $2n$ vectors $(0,\ldots,v_k,\ldots,0)$ with $k \in \{1,\ldots,n\}$ and $v_k \in \{-1,1\}$ being the k-th element.

```
void DoEncoding()
{
   bool IND;
   int i, r, totNumLin, prelLabel;
   vector<int> CurPixCs(n), LinePos(n-1);
   RleRun Run;
   RleLine *pLine;

   // set variables
   CurPixCs.init(0);
   LinePos.init(0);
   totNumLin = 1;
   Index[Rank[0]] = 0;
   FOR i FROM 1 TO n-1 DO
   {
      Index[Rank[i]] = i;
```

THE RUN LENGTH ENCODING ALGORITHM

Besides the data structures from above, the implementation of our run length encoding needs the following. Entities marked as 'global' are available in all routines. The variables `n`, `FG_COL` and `BG_COL` as well as the vectors `Rank` and `Index` are global.

`IND` – Boolean variable which indicates whether a run of foreground pixels is active (`true`) or not (`false`)

`LABEL_PROP` [global] – Boolean variable which indicates whether label propagation is switched on (`true`) or off (`false`)

`totNumLin` – the total number of lines

`prelLabel` – variable for assigning a unique preliminary label to each run if `LABEL_PROP == false`

`CurPixCs` – vector holding the coordinates of the current pixel (w. r. t. the original order of directions), *i.e.* the current pixel is `Image[CurPixCs[n-1]]..[CurPixCs[0]]`.

`LinePos` – vector storing the position of the current line in the image w. r. t. the scanning order, *i.e.* `CurPixCs[Index[r]] = LinePos[r-1]` for $0 < r < n$

`RleData` [global] – a $n-1$-dimensional array of `RleLine`s with dimensions (`ImgSize[Index[n-1]]`,..., `ImgSize[Index[1]]`), stores the run length encoded version of `Image`

We are now prepared to turn to the pseudo-code for the procedure `DoEncoding()`, which performs the run length encoding.

```
      totNumLin *= ImgSize[i];
   }

   // allocate memory for the array RleData
   // The size of the i-th dimension (i = 1,...,n-1) of RleData is the
   // image size in the direction with rank n-i.
   RleData = new RleLine[ImgSize[Index[n-1]]]..[ImgSize[Index[1]]];

   // main loop
   DO
   {
      pLine = &RleData[LinePos[n-2]]..[LinePos[0]];

      // set the coordinates of the current pixel
      FOR r FROM 1 TO n-1 DO
         CurPixCs[Index[r]] = LinePos[r-1];
      CurPixCs[Index[0]] = 0;

      // check whether a run starts at the first pixel in the current line
      IF Image[CurPixCs[n-1]]..[CurPixCs[0]] == FG_COL THEN
      {
         IND = true;
         Run.Pos[0] = 0;
      }
      ELSE
         IND = false;

      FOR i FROM 1 TO ImgSize[Index[0]]-1 DO
      {
         CurPixCs[Index[0]]++;

         IF Image[CurPixCs[n-1]]..[CurPixCs[0]] == FG_COL THEN
         {
            // if IND == true, then current run continues, nothing to do

            // a new run starts
            IF !IND THEN
            {
               IND = true;
               Run.Pos[0] = i;
            }
         }
         ELSE
         {
            // current run ends
            IF IND THEN
            {
               IND = false;
               Run.Pos[1] = i-1;
               prelLabel++;
               Run.label = LABEL_PROP? -999:prelLabel;
               pLine->add(Run);
            }

            // if IND == false, then BG_COL-run continues, nothing to do
         }
      }

      // the end of the current line in the rank 0-direction is reached
      // check whether some run is active and end it
      IF Image[CurPixCs[n-1]]..[CurPixCs[0]] == FG_COL THEN
      {
```

```
         IF !IND THEN
            Run.Pos[0] = ImgSize[Index[0]]-1;
         Run.Pos[1] = ImgSize[Index[0]]-1;
         prelLabel++;
         Run.label = LABEL_PROP? -999:prelLabel;
         pLine->add(Run);
      }

      // update the line position, according to the scanning order
      r = 0;
      WHILE r < n-1 AND ++LinePos[r] >= ImgSize[Index[r+1]] DO
         LinePos[r++] = 0;
   }
   WHILE LinePos[n-2] < ImgSize[Index[n-1]]
}
```

THE LABELING ALGORITHM

Next, we consider the actual labeling of the image data, which is based on a preceding run length encoding. We have already stated that all correspondences between preliminary labels can be found by testing only the start and the endpoint of runs with their respective neighbors according to the chosen adjacency. The number of correspondences can be kept very small if not a unique preliminary label is assigned to each run beforehand (during the run length encoding), but instead labels are propagated to all adjacent, non-labeled runs (having still the placeholder label -999), and a new preliminary label is given only to a run which has not already received one, while scanning the image in the specified order. If labels are propagated, only correspondences between a new preliminary label and a previously propagated one have to be registered and resolved later on. Otherwise, the number of preliminary labels and the number of arising correspondences usually is much higher. A recursive labeling approach like that from Martín-Herrero (2004) is based on an extremal form of label propagation in the sense that the propagation may emerge from every point of a run and is started again in every line which received a propagated label. In our algorithm, it is restricted to the start and endpoint of a run and stopped at the adjacent runs. The number of preliminary labels determines the size of the vector `LabelMap`, whose entry indices represent the preliminary labels. This vector is processed by means of the correspondences (in the procedures `Associate()` and `ResetLabelMap()`) such that each entry is assigned the value of the associated final label. Subsequently, the labels of the runs are updated using this `LabelMap`. At last, the labeled image can be written as a pixel array by decoding the run length representation `RleData`.

In the pseudo-code for the labeling method `DoLabeling()`, variables which are not described below have the same meaning as in `DoEncoding()`.

`NB_VALID` – Boolean variable which indicates whether a neighbor pixel lies inside the image (`true`) or not (`false`)

`newPrelLabel` – variable for assigning a unique preliminary label to each non-labeled run (with placeholder label) if `LABEL_PROP == true`

`prelLabel` – stores the preliminary label of a run; if `LABEL_PROP == false`, then `prelLabel` has the same meaning as in `DoEncoding()`, otherwise `prelLabel` does not necessarily increase while passing through the image but might hold a propagated label

`CurNbCs` – vector holding the coordinates of the current neighbor pixel (w. r. t. the original order of directions)

`LabelMap` [global] – vector mapping the preliminary labels (as the indices of the entries) to the associated final labels (as their values)

The labeling procedure `DoLabeling()` returns the number of objects in `Image` corresponding to the adjacency deducible from the predefined array `Neighbors`. In the following pseudo-code, code passages enclosed by '###' should be regarded as present only if the condition in the ###-header is fulfilled.

```
int DoLabeling()
{
   bool NB_VALID;
   int i, j, k, m, r, newPrelLabel, prelLabel, objects;
   vector<int> CurPixCs(n), CurNbCs(n), LinePos(n-1), LabelMap;
   RleLine *pLine;
```

```
RleRun *pRun, *pNbRun;

// set variables
newPrelLabel = 0;
CurPixCs.init(0);
LinePos.init(0);

IF LABEL_PROP THEN
   LabelMap.resize(1);
ELSE
   LabelMap.resize(prelLabel+1);
LabelMap.init(BG_COL);

// main loop
DO
{
   pLine = &RleData[LinePos[n-2]]..[LinePos[0]];

   // set the coordinates of the current pixel
   FOR r FROM 1 TO n-1 DO
      CurPixCs[Index[r]] = LinePos[r-1];

   // test all runs in the current line addressed by pLine
   FOR i FROM 0 TO pLine->size()-1 DO
   {
      pRun = &(pLine->at(i));

      ### LABEL_PROP == true
      // check whether run has yet received a propagated label
      // otherwise assign a new preliminary label and register it
      IF pRun->label == -999 THEN
      {
         pRun->label = ++newPrelLabel;
         LabelMap.add(BG_COL);
      }
      ###

      prelLabel = pRun->label;

      // examine the start and endpoint of the current run
      FOR j FROM 0 TO 1 DO
      {
         IF j == 0 OR (j == 1 AND pRun->Pos[0] != pRun->Pos[1]) THEN
         {
            CurPixCs[Index[0]] = pRun->Pos[j];

            // test the valid neighbors for label correspondences
            FOR k FROM 0 TO Neighbors.size()-1 DO
            {
               NB_VALID = true;

               FOR m FROM 0 TO n-1 WHILE NB_VALID DO
               {
                  CurNbCs[m] = CurPixCs[m] + Neighbors[k][m];

                  IF CurNbCs[m] < 0 OR CurNbCs[m] >= ImgSize[m] THEN
                     NB_VALID = false;
               }

               // neighbor is valid, i.e. lies inside the image
               IF NB_VALID THEN
               {
```

```
                    // query the address of the current neighbor
                    // this is NULL if the neighbor is a background pixel
                    pNbRun = QueryPtr(CurNbCs);

                    IF pNbRun != NULL AND pNbRun->label != prelLabel THEN
                    {
                       ### LABEL_PROP == true
                       IF pNbRun->label != -999 THEN
                          Associate(prelLabel, pNbRun->label, LabelMap);
                       ELSE
                          pNbRun->label = prelLabel;
                       ###

                       ### LABEL_PROP == false
                       Associate(prelLabel, pNbRun->label, LabelMap);
                       ###
                    }
                 }
              }
           }
        }

        // update the line position, according to the scanning order
        r = 0;
        WHILE r < n-1 AND ++LinePos[r] >= ImgSize[Index[r+1]] DO
           LinePos[r++] = 0;
     }
     WHILE LinePos[n-2] < ImgSize[Index[n-1]]

     // setup the final LabelMap
     objects = ResetLabelMap(LabelMap);

     // update the run labels
     LinePos.init(0);

     DO
     {
        pLine = &RleData[LinePos[n-2]]..[LinePos[0]];

        FOR i FROM 0 TO pLine->size()-1 DO
           pLine->at(i).label = LabelMap.at(pLine->at(i).label);

        r = 0;
        WHILE r < n-1 AND ++LinePos[r] >= ImgSize[Index[r+1]] DO
           LinePos[r++] = 0;
     }
     WHILE LinePos[n-2] < ImgSize[Index[n-1]]

     // return the number of objects in the image
     return objects;
}
```

The array `RleData` now contains all information on the labeled image. The next routine decompresses this information and writes an image in pixel format.

```
void WriteLabeledImage()
{
   int i, r;
   vector<int> CurPixCs(n), LinePos(n-1);
   RleLine *pLine;
   RleRun *pRun;
```

```
   LinePos.init(0);

   DO
   {
      pLine = &RleData[LinePos[n-2]]..[LinePos[0]];

      FOR r FROM 1 TO n-1 DO
         CurPixCs[Index[r]] = LinePos[r-1];

      FOR i FROM 0 TO pLine->size()-1 DO
      {
         pRun = &(pLine->at(i));

         FOR CurPixCs[Index[0]] FROM pRun->Pos[0] TO pRun->Pos[1] DO
            Image[CurPixCs[0]]..[CurPixCs[n-1]] = pRun->label;
      }

      r = 0;
      WHILE r < n-1 AND ++LinePos[r] >= ImgSize[Index[r+1]] DO
         LinePos[r++] = 0;
   }
   WHILE LinePos[n-2] < ImgSize[Index[n-1]]
}
```

The overall calling sequence is `DoEncoding() - DoLabeling() - WriteLabeledImage()`. We now explain shortly the auxiliary routines `QueryPtr()`, `Associate()`, as well as `ResetLabelMap()`.

The method `QueryPtr()` returns a pointer to the run which contains the pixel with coordinate vector `Cs` if this is not a background pixel. Otherwise, the pixel is not contained in any run, and the `NULL`-pointer is returned.

```
RleRun* QueryPtr(vector<int>& Cs)
{
   int i, pos;
   vector<int> LinePos(n-1);
   RleLine *pLine;

   pos = Cs[Index[0]];
   FOR i FROM 1 TO n-1 DO
      LinePos[i-1] = Cs[Index[i]];

   pLine =
      &RleData[LinePos[n-2]]..[LinePos[0]];

   FOR i FROM 0 TO pLine->size()-1 DO
   {
      IF pLine->at(i).Pos[1] >= pos THEN
      {
         IF pLine->at(i).Pos[0] <= pos THEN
            return &(pLine->at(i));
         ELSE
            return NULL;
      }
   }
   return NULL;
}
```

The recursive method `Associate()` relates equivalent preliminary labels `a` and `b` with the smallest equivalent label in `LabelMap`. After calling this routine, traversing `LabelMap` via repeated substitution of `index` by `LabelMap[index]` as long as the latter is non-zero, starting from either `index = a` or `index = b`, leads to the smallest equivalent label as the last `index`.

```
void Associate(int a, int b,
   vector<int>& LabelMap)
{
   int c;

   IF a < b THEN
   { c = a; a = b; b = c; }

  IF LabelMap[a] == BG_COL THEN
    LabelMap[a] = b;
  ELSE IF LabelMap[a] != b THEN
    Associate(LabelMap[a], b, LabelMap);
}
```

Finally, `ResetLabelMap()` assigns unique and minimal final labels to the preliminary ones which have no smaller counterparts and resolves the connections made up by `Associate()`. This means that afterwards each preliminary label `i` is mapped to the smallest integer `j = LabelMap[i]` which preserves the equivalence relations. The return value of `ResetLabelMap()` is the number of objects in `Image`.

```
int ResetLabelMap(vector<int>& LabelMap)
{
   int i, objects;
```

```
   objects = 0;

   FOR i FROM 0 TO LabelMap.size()-1 DO
   {
      IF LabelMap[i] == BG_COL THEN
         LabelMap[i] = ++objects;
      ELSE
         LabelMap[i] = LabelMap[LabelMap[i]];
   }

   return objects;
}
```

The latter both routines have been proposed and discussed in Martín-Herrero and Peón-Fernández (2000). The algorithm presented in this section is the basis for the current labeling function in the commercial software MAVI (Fraunhofer ITWM, Department of Image Processing, 2005).

EXAMPLE AND DISCUSSION

In this final section, we look at an exemplary application of our algorithm as well as of the alternative recursive algorithm by Martín-Herrero, *cf.* Martín-Herrero (2004). The test data are two binarized 3D microtomography images of materials. Their visualizations are shown in Figs. 1 and 2.

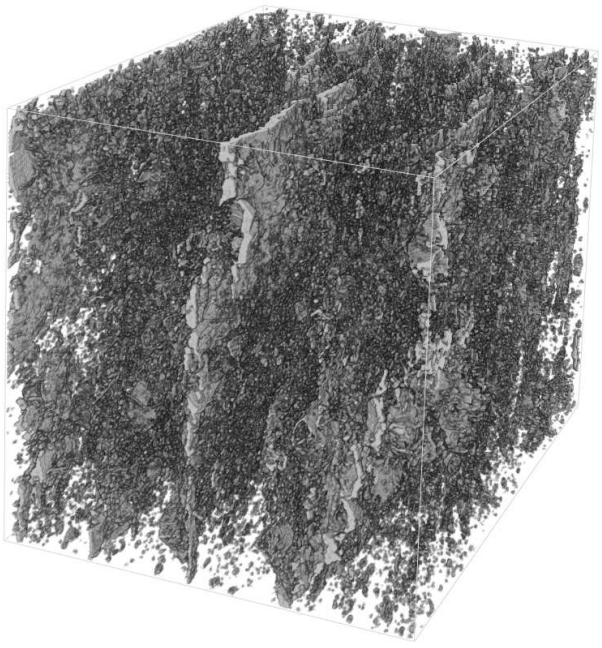

Fig. 1. *3D image (660 × 660 × 660 pixels, 0.7 µm pixel size) showing a specimen of a metallic foam in early extension stage.*

In Fig. 1 the metallic matrix is transparent while the pore space appears opaque. In Fig. 2 the solid matter is opaque and the pore space is transparent. Both images are at hand in RAW format with 8 Bit per pixel. The first image has a size of 274 MB and 5.7 % foreground pixels. The second one needs 26 MB and has 86.4 % foreground pixels. For details about the materials and imaging methods see Helfen *et al.* (2002; 2003).

Our test system is a PC with AMD Athlon(tm) 64 Processor 3800+, 1 GB RAM, running SuSE Linux 10.0 (kernel v2.6.13). We compiled the source codes with the GCC v4.0.2, using the optimization switch '-O3'. The performance data are listed in the tables below ('S./O.' denotes our algorithm, 'M.-H.' the one from Martín-Herrero).

Fig. 2. *3D image (299 × 300 × 300 pixels, 0.75 µm pixel size) showing a specimen of the Fontainebleau sand stone.*

At first, we want to make clear what the times listed in Tables 1 and 2 refer to. For our algorithm, we note that after calling the procedures `DoEncoding()` and `DoLabeling()` the run length array `RleData` with the correct run labels is obtained. It might be sufficient or even beneficial to do any further processing of the image on this structure. Otherwise, it remains to write the labeled image as a pixel array using `WriteLabeledImage()`. The processing time for this has not been considered in Tables 1 and 2. Since the algorithm by Martín-Herrero identifies non-labeled foreground pixels by a negative value, it usually necessitates a preprocessing of the source pixel array `Image` to change the foreground label `FG_COL` to this value (if the values in `Image` are non-negative). In principle, the identification value can be any value which does not coincide with the

background color `BG_COL` (mostly chosen to be 0) and any valid final label. However, the number of objects in `Image` is not predicted, and we require the final labels to be consecutive integers starting from 1. If one agrees to drop this restriction, a substitution of `FG_COL` can be avoided. Anyway, the time possibly needed for this is not recorded above. We also want to emphasize that the times have been obtained with variants of both algorithms optimized for 3D data and a fixed scanning order.

In comparison with the second image, the first one has only few foreground pixels and a pixels per object ratio of 59. By contrast, the second image contains mainly foreground pixels, which are multiply connected, and has 704 926 pixels per object in average. On the first dataset, the recursive algorithm by Martín-Herrero is more efficient than ours and takes clear advantage of extremal label propagation. Nevertheless, it appears that our method with activated label propagation can serve as an alternative. Since the algorithms rely on different image representations (pixel array vs. RLE structure), it is not appropriate to compare the times for the labeling only. In this respect, we remark that it could be valuable for both algorithms to inspect how they can be adapted to different image representations. For this question, the fundamental work Dillencourt *et al.* (1992) should be consulted.

On the dataset for Fig. 2, our approach combined with label propagation shows some superiority compared to the recursive one. For the latter, we note here the relatively high recursion depth along with the fact that many pixels are queried which have already received a propagated label. The label propagation in our method significantly reduces the number of preliminary labels for this image and makes the labeling procedure including the computation of `LabelMap` very fast. For the evaluation and comparison, it is also important that the runs for the second image contain 39 pixels in average, while for the first one this number is 4. Roughly speaking, it appears that a setting in which many pixels are captured in few runs overlapping in adjacent lines is more amenable to our method than to the other. Also in view of the above times, it should be carefully observed whether a preprocessing such as RLE leads to an overall improvement in the processing needs (the processing time, in particular). Although for the worst case of a 'salt and pepper' image (with a large amount of small objects) also the recursive and probably almost every other labeling algorithm can not fully deploy its potential, a run length encoding surely will spoil the total performance of the procedure. Martín-Herrero's algorithm from Martín-Herrero (2004) and the enhanced Hoshen-Kopelman algorithm from Hoshen (1998) offer an on-the-fly analysis of objects. This feature could be added to our algorithm by collecting information during the run length encoding and processing it subject to the label correspondences (and propagations) during the labeling.

Two other important impact factors for the labeling are the adjacency system and the scanning order. The above results concern the labeling w. r. t. the 6-adjacency. Since neighbors in the direction of the (quasi-)run length encoding need not to be checked for correspondences, a κ-adjacency requires a test of only $\kappa - 2$ neighbors in `DoLabeling()` and the recursion part of the algorithm from Martín-Herrero (2004), respectively. When switching to *e. g.* the 14.x-adjacency, in our method the ratio $(14-2)/(6-2) = 3$ of tested neighbors is roughly reflected in the times needed for the labeling. Although described only for the 6-adjacency in Martín-Herrero (2007), the recursive method can be straightforwardly extended to other adjacencies. A test and comparison for these has not yet been done.

Table 1. *Performance data for Fig. 1 and the 6-adjacency, 275 536 objects.*

algorithm	S./O.: ♯ preliminary labels M.-H.: max. recursion depth	execution time		
		RLE	Labeling	total
S./O., `LABEL_PROP = false`	3 856 027	2.73 s	5.93 s	8.66 s
S./O., `LABEL_PROP = true`	738 601	2.72 s	3.70 s	6.42 s
M.-H.	504 706	—	4.37 s	4.37 s

Table 2. *Performance data for Fig. 2 and the 6-adjacency, 33 objects.*

algorithm	S./O.: ♯ preliminary labels M.-H.: max. recursion depth	execution time		
		RLE	Labeling	total
S./O., `LABEL_PROP = false`	590 753	0.34 s	2.96 s	3.3 s
S./O., `LABEL_PROP = true`	28 176	0.33 s	0.58 s	0.91 s
M.-H.	590 649	—	1.83 s	1.83 s

Regarding the theoretical foundation from section "Connectedness", we want to remark the following. Both algorithms clearly work for arbitrary adjacency systems including the 18-adjacency. However, one should be aware that further processing and analysis of images labeled w. r. t. the 18-adjacency can lead to conflicts. Since there does not exist an adjacency system for the background which is complementary to the 18-adjacency according to Definition 3, the topology of the foreground pixels does not fit the topology of the background pixels. In particular, the Euler number of the background differs from the sum of the Euler numbers of the obtained equivalence classes. Thus, whenever further processing and analysis of the label image is necessary, we recommend a labeling w. r. t. an adjacency system $\mathbb{F}$ with existing $\mathbb{F}_c$ according to Definition 3.

A change in the scanning order can have significant impact on the processing time, depending on the anisotropy of the image. Using the general implementations of the algorithms (not optimized for a specific scanning order), the change in processing time for the 6-adjacency is as follows: For the algorithm from Martín-Herrero (2004), it is up to 11 % for the first image and up to 13 % for the second one. For our algorithm with label propagation, it is up to 19 % for the first image and up to 7 % for the second one. However, one should regard the short times and a possible amplification of marginal variations. Moreover, the second image is nearly isotropic. The impact of the choice of the scanning order is expected to become obvious for strongly anisotropic images. For the datasets used in this benchmark and further details see the webpage at http://www.itwm.fhg.de/bv/projects/BA/RLE.

CONCLUSION

In the first part of our paper, we discussed the discretization and labeling of an *n*-dimensional set w. r. t. to a given adjacency system. We suggest here to define the complementarity of adjacency systems by a relation for the Euler number instead of by the Jordan-Veblen and Jordan-Brouwer theorem in 2D and 3D, respectively. This appears natural from a practical point of view and is relevant for the consistency of the labeling results. In the second part, we proposed a new and flexible labeling algorithm based on a preceding run length encoding of the input image. We shortly address here the applications of this preprocessing and a few possible extensions of the algorithm.

If the image or its objects are to be further processed after the labeling, then a compact and easy-manageable representation of the image as given in a simple form by a run length array can be desirable. We only mention the morphological filtering of objects, noise removal and geometric transforms as operations which can take advantage of this. For further uses of the run length encoding we refer to the papers Di Zenzo *et al.* (1996) and Messom *et al.* (2002). Since the recursive method of Martín-Herrero and ours follow in a way complementary ideas, a combination of both methods could be worth considering.

Moreover, the tasks of run length encoding and labeling are well-suited for parallelization. A simple step towards this would be to separate big images into blocks and process first their interiors independently and then gather correspondences in the border areas. Using label offsets after the block computations to avoid a false coincidence of labels, a label map for the whole image can be computed.

ACKNOWLEDGMENTS

The authors very much appreciate the comments and suggestions by the referees. We also thank Björn Wagner and Michael Godehardt from the Fraunhofer Institute for Industrial Mathematics (ITWM) for software support and fruitful discussions about implementation of algorithms. The research of the second author was supported by the FH^3-programme of the German Federal Ministry of Education and Research under project grant 1711B06.

REFERENCES

Aguilera A, Rodríguez J, Ayala D (2002). Fast connected component labeling algorithm: A non voxel-based approach. Tech. Rep. LSI-02-26-R, Universitat Politècnica, Catalunya. http://www.lsi.upc.edu/dept/techreps/.

Armstrong MA (1997). Basic Topology. Berlin: Springer.

Di Zenzo S, Cinque L, Levialdi S (1996). Run-based algorithms for binary image analysis and processing. IEEE Trans Pattern Anal 18:83–9.

Dillencourt MB, Samet H, Tamminen M (1992). A general approach to connected-component labeling for arbitrary image representations. J ACM 39:253–80.

Fraunhofer ITWM, Department of Image Processing (2005). MAVI – Modular Algorithms for Volume Images. http://www.itwm.fhg.de/mab/projects/MAVI/.

Helfen L, Baumbach T, Schladitz K, Ohser J (2003). Determination of structural properties of light materials by three-dimensional synchrotron-radiation imaging. Imaging Microsc 5:55–7.

Helfen L, Baumbach T, Stanzick H, Banhart J, Elmoutaouakkil A, Cloetens P (2002). Viewing the early stage of metal foam formation by computed

tomography using synchrotron radiation. Adv Eng Mater 4:808–13.

Hopcroft J, Tarjan RE (1973). Efficient algorithms for graph manipulation. Comm ACM 16(6):372–8.

Hoshen J (1998). On the application of the enhanced Hoshen-Kopelman algorithm for image analysis. Pattern Recogn Lett 19:575–84.

Hoshen J, Koppelman R (1976). Percolation and cluster distribution. I. cluster multiple labeling technique and critical concentration algorithm. Phys Rev B 14:575–84.

Klette R, Rosenfeld A (2004). Digital Geometry. Amsterdam: Morgan & Kaufman Publ.

Lachaud JO, Montanvert A (2000). Continuous analogs of digital boundaries: A topological approach to iso-surfaces. Graphical Models 62:129–64.

Martín-Herrero M (2004). Hybrid cluster identification. J Phys A Math Gen 37:9377–86.

Martín-Herrero M (2007). Hybrid object labelling in digital images. Machine Vision Appl 18:1–15.

Martín-Herrero M, Peón-Fernández J (2000). Alternative techniques for cluster labelling on percolation theory. J Phys A Math Gen 33:1827–40.

Messom CH, Demidenko S, Subramaniam K, Gupta GS (2002). Size/position identification in real-time image processing using run length encoding. Proc 19th IEEE Instrum Meas Tech Conf 2:1055–9 vol. 2.

Nagel W, Ohser J, Pischang K (2000). An integral-geometric approach for the Euler-Poincaré characteristic of spatial images. J Microsc 198:54–62.

Ohser J, Nagel W, Schladitz K (2002). The Euler number of discretized sets – on the choice of adjacency in homogeneous lattices. In: Mecke KR, Stoyan D, eds., Morphology of Condensed Matter, Lect Notes Phys. Berlin: Springer.

Ohser J, Nagel W, Schladitz K (2003). The Euler number of discretised sets – surprising results in three dimensions. Image Anal Stereol 22:11–9.

Ritter GX, Wilson JN (2001). Handbook of computer vision algorithms in image algebra, chap. 6, Connected Component Algorithms. Boca Raton, Florida: CRC Press, 173–86.

Rosenfeld A (1970). Digital topology. Amer Math Monthly 86:621–30.

Rosenfeld A, Pfaltz JL (1966). Sequential operations in digital picture processing. J ACM 13:471–94.

Rotman JJ (1993). An Introduction to Algebraic Topology. Berlin: Springer-Verlag.

Schladitz K, Ohser J, Nagel W (2006). Measurement of intrinsic volumes of sets observed on lattices. In: Kuba A, Nyul LG, Palagyi K, eds., Proc 13th Int Conf Discrete Geom Comput Imagery, LNCS. DGCI, Szeged, Hungary, Berlin: Springer.

Schmitt M (1998). Digitization and connectivity. In: Heijmans HJAM, Roerdnik JBTM, eds., Mathematical morphology and its application to image and signal processing. Dortrecht: Kluwer Academic Publishers.

Schneider R (1993). Convex Bodies: The Brunn-Minkowski Theory. Cambridge: Encyclopedia of Mathematics and Its Application Vol. 44, Cambridge University Press.

Thurfjell L, Bengtsson E, Nordin B (1992). A new three-dimensional connected components labeling algorithm with simultaneous object feature extraction capability. Comput Vision Graph 54:357–64.