Algorithms and Data Structures for Computer Topology

Vladimir Kovalevsky

University of Rostock
Institute of Computer Graphics
Albert-Einstein-Str. 21, 18051 Rostock, Germany
kovalev@tfh-berlin.de

Abstract. The paper presents an introduction to computer topology with applications to image processing and computer graphics. Basic topological notions such as connectivity, frontier, manifolds, surfaces, combinatorial homeomorphism etc. are recalled and adapted for locally finite topological spaces. The paper describes data structures for explicitly representing classical topological spaces in computers and presents some algorithms for computing topological features of sets. Among them are: boundary tracing (n=2,3), filling of interiors (n=2,3,4), labeling of components, computing of skeletons and others.

Introduction: Topology and Computers

Topology plays an important role in computer graphics and image analysis. Connectedness, boundaries and inclusion of regions are topological features which are important for both rendering images and analyzing their contents. Computing these features is one of the tasks of the *computer topology*. We use this term rather than "computational topology" since our approach is analogous to that of digital geometry rather than to that of computational geometry: we are using models of topological spaces explicitly representing each element of a finite topological space as an element of the computer memory, defined by its integer coordinates. The other possible approach would be to think about the Euclidean space, to define objects by equations and inequalities in real coordinates and to approximate real coordinates on a computer by floating point variables.

Computer topology may be of interest both for computer scientists who attempt to apply topological knowledge for analyzing digitized images, and for mathematicians who may use computers to solve complicated topological problems. Thus, for example, essential progress in investigating three-dimensional manifolds has been reached by means of computers (see e.g. [14]).

Topological ideas are becoming increasingly important in modern theoretical physics where attempts to develop a unique theory of gravitation and quantum mechanics have led to the Topological Quantum Field Theory (see e.g. [2,13]), in which topology of multi-dimensional spaces plays a crucial role. This is one more possible application field for computer topology. Thus, computer topology is important both for applications in computer imagery and in basic research in mathematics and physics.

We describe here among others the new data structure called the 3D cell list which allows to economically encode a segmented 3D image or a 3-manifold and gives the possibility to access topological information without searching. Then we describe algorithms on boundary tracing in 2D and 3D, filling of interiors of boundaries in nD (successfully tested up to n=4), component labeling, computing skeletons and computing basic topological relations in nD, $n\leq 4$. All algorithms are based on the topology of abstract cell complexes and are simpler and more economical with respect to time and memory space than traditional algorithms based on grid point models.

1 Basic notions

In this section we recall some notions and definitions from the classical topology, which are necessary for reading the subsequent sections.

1.1 Topology of abstract complexes

There exist topological spaces in which any space element possesses the smallest neighborhood [1]. If the smallest neighborhood is finite then the space is called *locally finite*. Among locally finite spaces abstract cell complexes are those especially well suited for computer applications, as explained below. Abstract complexes are known since 1908 [16]. They are called "abstract" because their elements, the abstract cells, need not to be considered as subsets of the Euclidean space. A historical review may be found in [7, 17].

Definition AC: An *abstract cell complex* (AC complex) $C=(E, B, dim)$ is a set E of abstract elements (cells) provided with an antisymmetric, irreflexive, and transitive binary relation $B \subset E \times E$ called the *bounding relation*, and with a dimension function $dim: E \rightarrow I$ from E into the set I of non-negative integers such that $dim(e') < dim(e'')$ for all pairs $(e',e'')\in B$.

The maximum dimension of the cells of an AC complex is called its dimension. We shall mainly consider complexes of dimensions 2 and 3. Their cells with dimension 0 (0-cells) are called *points*, cells of dimension 1 (1-cells) are called *cracks* (edges), cells of dimension 2 (2-cells) are called *pixels* (faces) and that of dimension 3 are the *voxels*.

If $(e', e'')\in B$ then it is usual to write $e'<e''$ or to say that the cell e' *bounds* the cell e''. Two cells e' and e'' of an AC complex C are called *incident with each other in C* iff either $e'=e''$, or e' bounds e'', or e'' bounds e'. In AC complexes no cell is a subset of another cell, as it is the case in simplicial and Euclidean complexes. Exactly this property of AC complexes make it possible to define a topology on the set of abstract cells independently from any Hausdorff space.

The topology of AC complexes with applications to computer imagery has been described in [9]. We recall now a few most important definitions. In what follows we say "complex" for "AC complex".

Definition SC: A *subcomplex* $S = (E', B', dim')$ of a given complex $C = (E, B, dim)$ is a complex whose set E' is a subset of E and the relation B' is an intersection of B with $E' \times E'$. The dimension dim' is equal to dim for all cells of E'.

Since a subcomplex is uniquely defined by the subset E' it is possible to apply Boolean operations as union, intersection and complement to complexes. We will often say "subset" while meaning "subcomplex".

The *connectivity* in complexes is the *transitive hull of the incidence relation*. It can be shown that the connectivity thus defined corresponds to classical connectivity.

Definition OP: A subset OS of cells of a subcomplex S of a complex C is called *open in S* if it contains all cells of S bounded by cells of OS. An n-cell c^n of an n-dimensional complex C^n is an open subset of C^n since c^n bounds no cells of C^n.

Definition SON: The smallest subset of a set S which contains a given cell c of S and is open in S is called the *smallest (open) neighborhood* of c relative to S and is denoted by SON(c, S).

The word "open" in "smallest open neighborhood" may be dropped since the smallest neighborhood is always open, however, we prefer to retain the notation "SON" since it has been used in many publications by the author.

Definition CL: The smallest subset of a set S which contains a given cell c of S and is closed in S is called the *closure* of c relative to S and is denoted by Cl(c, S).

Definition FR: The *frontier* Fr(S, C) of a subcomplex S of a complex C *relative to* C is the subcomplex of C containing all cells c of C whose SON(c, C) contains both cells of S as well as cells of the complement $C-S$.

Illustrations to AC complexes, SONs and closures of cells of different dimensions may be found in [9, 10, 12].

Definition OF: The *open frontier* Of(S, C) of a subcomplex S of a complex C *relative to* C is the subcomplex of C containing all cells c of C whose closure Cl(c, C) contains both cells of S as well as cells of the complement $C-S$.

Definition BD: The boundary ∂S of an n-dimensional subcomplex S of a complex C is the union of the closures of all $(n-1)$-cells of C each of which bounds exactly one n-cell of S.

Definition TL: A connected one-dimensional complex whose each cell, except two of them, is incident with exactly two other cells, is called a *topological line*.

It is easily seen that it is possible to assign integer numbers to the cells of a topological line in such a way that a cell incident with the cell having the number k has the number $k-1$ or $k+1$. These numbers are called the *topological coordinates* of the cells [10].

Definition CR: A Cartesian (direct) product C^n of n topological lines is called an n-dimensional *Cartesian* complex [8].

The set of cells of C^n is the Cartesian product of n sets of cells of the topological lines which are the *coordinate axes* of the n-dimensional space C^n. They will be denoted by A_i, i=1,2,...,n. A cell of C^n is an n-tuple (a_1, a_2,..., a_n) of cells a_i of the corresponding axes: $a_i \in A_i$. The bounding relation of C^n is defined as follows: the n-tuple (a_1, a_2,..., a_n) is bounding another distinct n-tuple (b_1, b_2,..., b_n) iff for all i=1,2,...n the cell a_i is incident with b_i in A_i and $dim(a_i) \le dim(b_i)$ in A_i.

The dimension of a product cell is defined as the sum of dimensions of the factor cells in their one-dimensional spaces. Topological coordinates of a product cell are defined by the vector whose components are the coordinates of the factor cells in their axes.

Fig. 1a shows four cells in a two-dimensional Cartesian complex: P is a 0-cell (point), C_1 and C_2 are 1-cells (a horizontal and a vertical crack), F is a 2-cell (pixel).

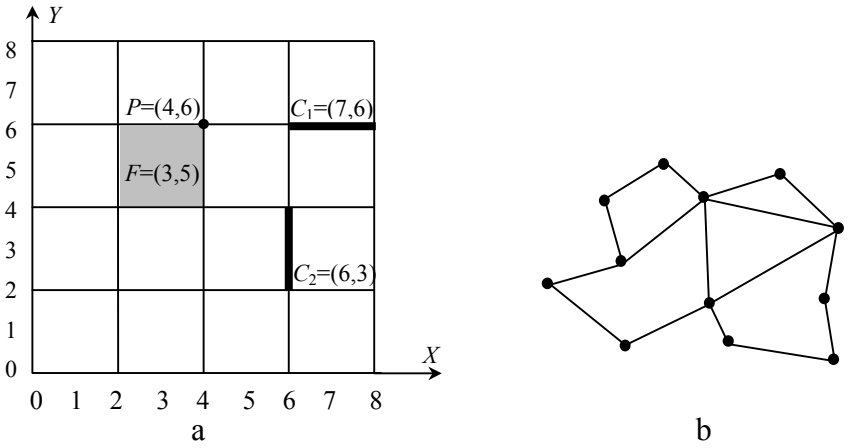

Fig. 1. Example of a two-dimensional Cartesian (a) and non-Cartesian (b) complexes

If we assign even numbers to the 0-cells and odd ones to the 1-cells of the axes then the dimension of a cell in a Cartesian complex is equal to the number of its odd coordinates.

1.2 Combinatorial homeomorphism

The notion of the homeomorphism of two sets is a fundamental notion of topology: two sets are called homeomorphic or topologically equivalent if one of them can be mapped onto the other by a one-to-one continuous function while the inverse map is also continuous. There is another classical way to define the homeomorphism, which way is directly applicable to complexes and may be extended to other locally finite spaces. It is called the *combinatorial homeomorphism* and is based on the notion of *elementary subdivisions*.

The concept of an AC complex is too general: it is e.g. possible to define an AC complex where a 1-cell is bounded by more than two 0-cells. To avoid such situations elementary subdivisions have been defined in classical topology (see e.g. a modern survey in [17]) on the base of the topology of the Euclidean space. We give in what follows an independent, purely combinatorial definition. It is a recursive definition: we make the necessary definitions primarily for 1-cells and then for cells of still greater dimensions.

A 1-cell which is bounded by exactly two 0-cells is called *proper*.

1. An *elementary subdivision* of a proper 1-cell c^1, which is bounded by the 0-cells c_1^0 and c_2^0, replaces the complex $C'=(c_1^0< c^1>c_2^0)$ by the 1-complex C'' with two 1-cells c_1^1, c_2^1 and a new 0-cell c_3^0: $C''=(c_1^0< c_1^1>c_3^0< c_2^1>c_2^0)$. One or both of the 0-cells c_1^0 and c_2^0 can be missing.

2. The following Definitions should be used recursively: first for m=1, then for m=2 etc.

An m-complex arising through N (N≥0) elementary subdivisions of a single proper m-cell is called an *open combinatorial m-ball*. When m=1 then it is a sequence of pairwise incident 1- and 0-cells, starting and ending with a 1-cell. A single 1-cell is also an open combinatorial 1-ball.

The boundary of an open m-ball is called a *combinatorial* (m−1)*-sphere*. When m=1 then it consists of exactly two 0-cells. The closure of an m-ball is called the *closed m-ball.* The union of two closed m-balls with identical boundaries is called a *combinatorial m-sphere.*

An m-cell c^m, m>1 is called *proper* if its boundary ∂c^m is an combinatorial (m−1)-sphere.

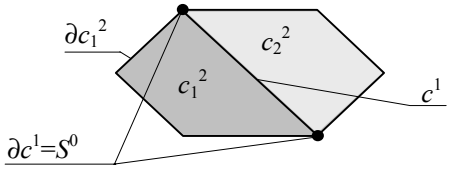

Fig. 2. Example of the elementary subdivision of a 2-cell

An *elementary subdivision* in an n-complex replaces a proper m-cell c^m, 1<m≤n, with two proper m-cells $c_1{}^m$, $c_2{}^m$ and one new proper (m−1)-cell $c^{(m-1)}$ bounding both m-cells $c_1{}^m$ and $c_2{}^m$ while the boundary $\partial c^{(m-1)}$ is an (m−2)-sphere $S^{(m-2)}\subset\partial c^2$, $\partial(c_1{}^m\cup c^{(m-1)}\cup c_2{}^m)=\partial c^m$ and $c^{(m-1)}\notin\partial c^m$.

An AC complex is called *proper* if all its cells are proper.

Two proper AC complexes are called *combinatorially homeomorphic* if they possess isomorphic subdivisions. Fig. 3 shows an example.

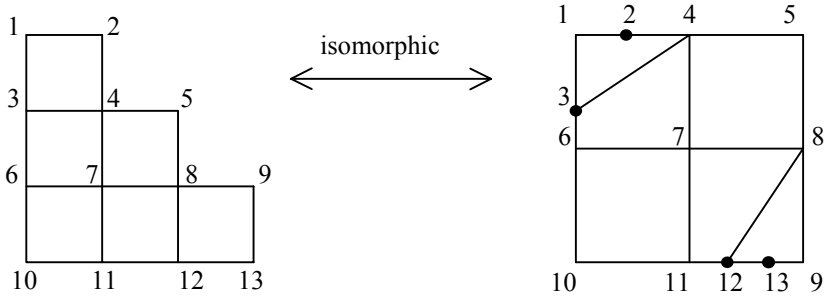

Fig. 3. A subdivision of a digitized square, which is isomorphic to a digitized triangle; black circles are new points introduced during the subdivision

1.3 Manifolds and surfaces

Among the variety of Hausdorff spaces there are spaces possessing certain relatively simple and important topological properties. They are called manifolds. It is known that manifolds of dimension not greater than three may be triangulated. This means that there exists a simplicial complex homeomorphic to a given manifold.

An AC complex which is combinatorially homeomorphic to the triangulation of a manifold may represent the topological properties of a manifold (of dimension up to three) in the same way as the triangulation does. This fact opens the possibility to model manifolds in computers for investigating them.

Definition MA: An n-dimensional *combinatorial manifold* (n-manifold) without boundary is an n-dimensional complex M in which the boundary of the SON(P, M) of each 0-cell P is homeomorphic to an $(n-1)$-sphere. In a *manifold with boundary* the SON(P, M) of some 0-cell P may have a boundary homeomorphic to a "half-sphere", i.e. to an $(n-1)$-ball.

Surfaces in a 3D space are frontiers of 3D subsets of the space. Under rather general conditions surfaces are 2-manifolds. Conditions under which the frontier of a 3D subset is a 2-manifold as well as surfaces which are no manifolds but rather "quasi-manifolds" are considered in [11].

2 Data structures

In this Section we consider some well-known and also some new data structures useful for representing topological information in two- and three-dimensional digitized images.

2.1 The standard raster

Two- and three-dimensional images are usually stored on a computer in arrays of the corresponding dimension. Each element of the array contains either a gray value, or a color, or a density. This data structure is not designed for topological calculations, nevertheless, it is possible to perform topological calculations without changing the data structure. For example, it is possible to trace and encode the boundary of a region in a two-dimensional image in spite of the apparent difficulty that the boundary consists of 0- and 1-cells, however, the raster contains only pixels which *must* be interpreted as 2-cells. The reason is that a pixel is mostly a carrier of an optical feature which is proportional to certain elementary area. Thus pixels must correspond to elementary areas which are the 2-cells rather than 0- or 1-cells whose area is zero. On the same reason voxels must correspond to 3-cells.

The tracing in the standard raster is possible because the concept of an AC complex is the *way of thinking* about topological properties of digitized images rather than a way of encoding them. Let us explain this idea for the case of tracing boundaries.

We think of a two-dimensional (2D) image as of a 2D Cartesian complex containing cells of dimensions form 0 to 2. The 2-cells (pixels) have integer coordinates which, unlike to topological coordinates of a pixel being always odd (compare Section 1.1 and Fig. 1), may take in the standard raster *any integer values*, both odd or even. Pixels are explicitly represented in the raster, whereas the 0- and 1-cells are present implicitly.

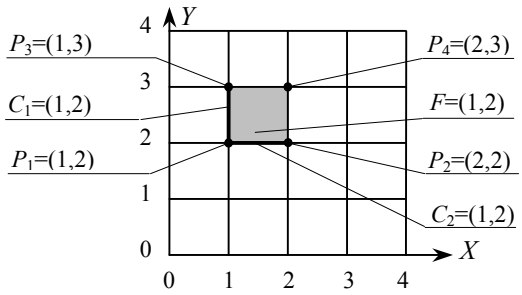

Fig. 4. Non-topological coordinates of cells of lower dimensions

Coordinate Assignment Rule: Each pixel F of a 2D image gets one 0-cell assigned to it as its "own" cell. This is the 0-cell lying in the corner of F which is the nearest to the origin of the coordinates (P_1 in Fig. 4). Also two 1-cells incident with F and with P are declared to be own cells of F (C_1 and C_2 in Fig. 4). Thus each pixel gets three own cells of lower dimensions. *All own cells of F get the same coordinates as F*.

In the three-dimensional case each voxel gets seven own cells of lower dimensions which are arranged similarly. These seven cells get the same coordinates as the corresponding voxel.

Unfortunately, some cells in the boundary of the raster remain without an "owner". In most applications this is of no importance. Otherwise the raster must be correspondingly enlarged.

According to the above rule, it is not difficult to calculate the coordinates of all pixels incident with a given point and to get the gray values of the pixels from the array. Depending on these gray values the tracing point P moves to its next position. Details of the tracing algorithm are described in Section 3.1.

The majority of low level topological problems in image processing may be solved in a similar way, i.e. without representing cells of lower dimension as elements of some multidimensional array. A typical exception is the problem of filling the interior of a region defined by its boundary. The solution is simpler when using two array elements per pixel: one for the pixel itself and one more for its own vertical crack (1-cell). The solution consists in reading the description of the boundary (e.g. its crack code), labeling all vertical cracks of the boundary in the array, counting the labeled cracks in each row (starting with 0), and filling the pixels between the crack with an even count $2 \cdot i$ and the next crack (with the count $2 \cdot i+1$). The details of this algorithm are described in Section 3.3.

Even more complicated topological problems may be solved by means of the standard raster. For example, when tracing surfaces (Section 3.2) or producing skeletons (Section 3.5) simple pixels must be recognized. A pixel is simple relative to a given region if the intersection of its boundary with the boundary of the region is connected. It is easier to correctly recognize all simple pixels if *cells of all dimensions* of the region are labeled. To perform this in a standard raster, it is possible to assign a bit of a raster element representing the pixel F to each own cell of F. For example, suppose that one byte of a two-dimensional array is assigned to each pixel of the image shown in Fig. 4. Consider the pixel F with coordinates (1, 2) and the byte assigned to it. The bit 0 of the byte may be assigned to the 0-cell P_1, the bit 1 to the 1-cell C_1, the bit 2 to the 1-cell C_2. The remaining bits may be assigned to F itself. Similar assignments are also possible in the 3D case.

As we see, there is no necessity to allocate memory space for each cell of a complex, which would demand four times more memory space than that needed for pixels only, or eight times more than that needed for the voxels in the 3D case.

The most data structures commonly used in the processing of 2D images may be used together with the standard raster. These are primarily the run length code and the crack code. The latter differs from the widely used Freeman code in that it contains only four directions rather than eight of the Freeman code. This is due to the properties of a 2D Cartesian complex whose oriented 1-cells have exactly four different directions.

2.2 The topological raster

Complexes used in topological investigations by means of a computer often have a relatively small number of cells. The direct access to cells of all dimensions and the possibility to use more than two different labels for cells of lower dimensions is then more important than the possibility to save memory space. This is the case, e.g. when investigating 3-manifolds represented as boundaries of subsets in a four-dimensional space while the space is represented as a four-dimensional array. In such cases a topological raster is more suitable than a standard one.

In a topological raster each coordinate axis is a topological line (see Definition TL in Section 1.1). The 0-cells of the axis have even coordinates, the 1-cells have odd coordinates. The dimension and the orientation (if defined) of any cell may be calculated from its topological coordinates which in this case coincide with the indices of the corresponding array element. The dimension of any cell is the *number* of its odd coordinates, the orientation is specified by indicating *which* of the coordinates are odd. For example, the cell C_1 in Fig. 1 above has *one* odd coordinate and this is its X-coordinate. Thus it is a *one*-dimensional cell oriented along the X-axis. The 2-cell F has two and the 0-cell P has no odd coordinates. In a three-dimensional complex the orientation of the 2-cells may be specified in a similar way: if the ith coordinate of a 2-cell F is the only even one then the normal to F is parallel to the ith coordinate axis.

We shall show in Section 3 how topological relations between cells, like the bounding or incidence relations, may be calculated from their topological coordinates.

2.3 Data structures using lists of space elements

A data structure designed to *efficiently* represent topological information must satisfy the following two demands:

1. The structure must contain *complete* topological information sufficient to get knowledge about topological relations among the parts of the image or of a 3D scene *without a search*. To the topological relations belong primarily the incidence and the adjacency relations (two distinct subsets are adjacent if there is a space element incident with both of them).
2. The structure must be able to correctly represent non-proper complexes which are often used in topological investigations because they contain much less elements than the corresponding proper complexes.

The notions of proper and non-proper complexes have been introduced by the author in [12]. We give here only an example and the necessary short explanation. The surface of a torus may be represented as a complex consisting of one 0-cell, two 1-cells and one 2-cell (Fig. 5a). This representation has the advantage of being very simple.

However, if one would try to interpret this representation as an AC complex, difficulties would occur since e.g. the AC complexes corresponding to Fig. 5a and Fig. 5b are the same: the same sets of four cells, the same bounding relation and the same dimensions of the cells. The difference between these two complexes is that each of the 1-cells L_1 and L_2 in Fig. 5a bounds the 2-cell *two times*, on both sides. This may be seen, if one considers the embedding of the complex in a Euclidean space: a neighborhood of a point on the 1-cell contains two half-disks each of which lies in one and the same 2-cell. However, there is no possibility to describe this relation in the language of complexes.

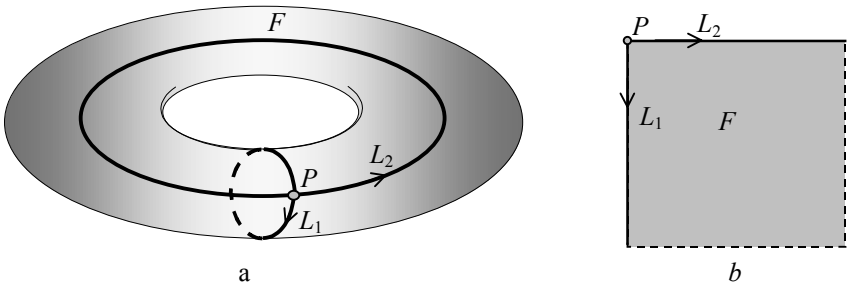

Fig. 5. Representations of the surface of a torus (a) and of a simple complex (b)

Since one of our aims is to consider a purely combinatorial approach with no relation to a Euclidean space we consider the possibility to overcome this difficulty by introducing the notion of an incidence structure [12] as explained below.

Thus when considering Fig. 5a as a representation of a complex then it is not a proper one (see Section 1.2): though each k-cell with $k>0$ is homeomorphic to an open k-ball the boundaries of the cells are not homeomorphic to $(k-1)$-spheres.

Data structures known from the literature do not fulfil the above demands 1 and 2. The classical incidence matrix (see e.g. [15]) enables one to encode any proper cell

complex. It contains complete topological information. However, it is not suitable to encode non-proper complexes, as explained above. Besides that, it is not economical: it contains in the case of an *n*-dimensional complex

$$\sum_{k=1}^{n} N_{k-1} \cdot N_k$$

elements where N_k is the number *k*-dimensional cells. This number is in practically relevant cases too large. Because of this reasons data structures using "linear" rather than "quadratic" lists of space elements are preferable.

Most 2D data structures of this kind can be hardly generalized for the 3D case. So the structures using the notion of "half-edges", e.g. the FTG [5], or the n-G-map [3] would need in the 3D case the introduction of "half-faces". In this case each edge would occur in so many copies as the number of faces bounded by it. The structure would be no more a graph as this is the case for the FTG: a complete FTG structure would be needed for each 3D region, which is not economical. No suggestion for a 3D version of the FTG structure is known to the author.

In computer graphics and geometric modeling 3D list data structures are known since many years. One of the most popular is the "boundary representation" [4]. This structure enables one to easily trace the boundary of a 2D face of a body. However, to find which bodies in a 3D scene are adjacent to each other demands an exhaustive search through the descriptions of *all vertices of all bodies in the scene*. Even simpler questions, as e.g. which edges are incident with a given vertex, demand an exhaustive search to be answered. This is true for all 3D data structures known to the author.

As far as we know, the possibility to represent non-proper complexes was not discussed in the literature before the author's publication [12].

2.4 The two-dimensional cell list

A 2D data structure satisfying the above mentioned demands, called the *cell list*, has been suggested by the author [9]. The peculiarity of the cell list is that the topological information, namely that of the incidence, is *explicitly* represented in it: it is possible to directly get the information about the boundaries of regions and the endpoints of lines. Information about adjacencies is available with a *restricted local search* since "adjacent" means "incident with an incident element".

The data structure of the cell list is based on the topological notion of a block complex [15] which we have adapted to AC complexes [9].

Definition BC: Consider a partition *M* of a complex *A* into subsets S_i^k. Subsets with k=0 are 0-cells of *A*; each of the subsets with k>0 is combinatorially homeomorphic to an open *k*-dimensional ball. There are a bounding relation *BR* and a dimension function *Dim* defined on *M* in the natural way. The triple

$$B(A)=(M, BR, Dim)$$

is called a *block complex of A*, the subsets S_i^k are called *k-dimensional blocks* or *k-blocks*.

Examples of two-dimensional block complexes and cell lists may be found in [9, 10].

2.5 The three-dimensional cell list

We call the subcomplex composed of all cells incident with a given *proper* cell c the *incidence structure* of c. In a 2D space the incidence structure of a cell consists either of a cyclic sequence or of two pairs of cells. The cyclic sequences are B-isomorphic to one-dimensional complexes (B-isomorphism [11] is a one-to-one map retaining the bounding relation but not the dimensions of cells). They can be represented in the computer as chained lists. In the 3D case the incidence structures of 0- and 3-blocks are B-isomorphic to two-dimensional complexes.

The author has shown [12] that in the case when the space is a 3-manifold the incidence structures are B-isomorphic to two-dimensional spheres. A finite 2-sphere is isomorphic to the surface of a convex polyhedron and therefore may be represented as a list of polygonal faces. This is the theoretical base of the 3D cell list. This data structure is appropriate to describe topological features of 3-manifolds and of 3D scenes containing *many bodies* which may have *common faces, edges or vertices*.

Let us consider a simple example of a topological 3D cell list with only two bodies where five faces of each cube are considered as a single face.

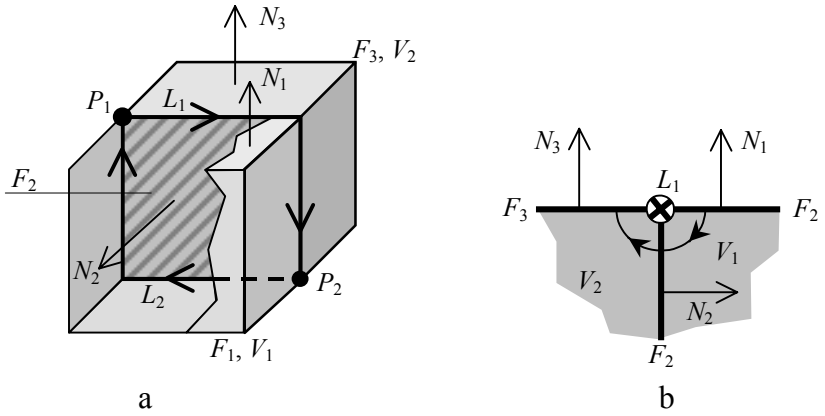

Fig. 6. A simple 3D block complex (a) and its cross section perpendicular to L_1 (b)

The 3D cell list of the 3D image of Fig. 6 is shown in the following tables.

List of branch points (0-blocks)

Label	$\boldsymbol{N}_{\mathrm{SON}}$	Lines
P_1	2	$-L_1, +L_2$
P_2	2	$+L_1, -L_2$

The partial list of the 0-blocks indicates for each 0-block P_i the number N_{SON} of all 1-blocks (lines) incident with P_i and their labels. The negative sign of a line's label in the row of P_i indicates that this line goes away from P_i. We have skipped here the geometric information, i.e. the coordinates.

In the list of 1-blocks (lines) N_{SON} denotes the number of blocks in the SON of the line L_i, i.e. the number of blocks bounded by L_i. The pointer Z_k points to the chained list containing the indices of these blocks as shown in the last column. The order of

the sequence corresponds to a right-handed rotation around L_i. A negative sign before a label of a face indicates that its normal is oriented against the direction of rotation.

List of lines (1-blocks)

Label	Starting point	End point	N_{SON}	Pointer	Chained list
L_1	P_1	P_2	5	Z_1	$-F_1 \rightarrow V_1 \rightarrow -F_2 \rightarrow V_2 \rightarrow +F_3 \rightarrow 0$
L_2	P_2	P_1	5	Z_2	$-F_1 \rightarrow V_1 \rightarrow -F_2 \rightarrow V_2 \rightarrow +F_3 \rightarrow 0$

List of faces (2-blocks)

Label	+Vol	–Vol	N_{Cl}	Pointer	Chained list
F_1	–	V_1	4	Z_3	$P_1 \rightarrow -L_2 \rightarrow P_2 \rightarrow -L_1 \rightarrow P_1$
F_2	V_1	V_2	4	Z_4	$P_1 \rightarrow -L_2 \rightarrow P_2 \rightarrow -L_1 \rightarrow P_1$
F_2	–	V_2	4	Z_5	$P_1 \rightarrow +L_1 \rightarrow P_2 \rightarrow +L_2 \rightarrow P_1$

The list of faces contains for each current face F_i the labels of two volumes bounded by F_i. The volume denoted by "+Vol" lies in the direction of the normal of F_i. N_{Cl} denotes the number of blocks in $Cl(F_i)$ which is shown as the chained list in the last column. The order of the sequence corresponds to a right-handed rotation around the normal of F_i. A negative sign before a label of a line indicates that the line is oriented against the direction of the rotation.

List of volumes (3-blocks)

Label	N_{Cl}	Faces
V_1	2	$+F_1, -F_2$
V_2	2	$+F_2, +F_3$

The partial list of volumes contains for each volume V_i the number N_{Cl} of the incident faces which are listed in the last column. The negative sign before the label of a face in the row of V_i indicates that the normal to the face is pointing away from V_i.

3 Algorithms

We describe here some algorithms for computing topological features of subsets in 2D and 3D digitized images. Since the programming languages of the C-family are now more popular than that of the PASCAL-family, we use here a pseudo-code which resembles the C-language.

3.1 Boundary tracing in 2D images

Boundary tracing becomes extremely simple when thinking of a 2D image as of a 2D complex. The main idea of the algorithm consists in the following: at each boundary point find the next boundary crack and make a step along the crack to the next

boundary point. Repeat this procedure until the starting point is reached again. Starting points of all boundary components must be found during an exhaustive search through the whole image. The following subroutine `Trace()` is called each time when a not yet visited boundary point of a region is found.

To avoid calling `Trace()` more than once for one region, vertical cracks must be labeled (e.g. in a bit of `Image[]`) as "already visited". `Trace()` follows the boundary of one foreground region while starting and stopping at the given point `(x,y)`. Points and cracks are present only implicitly as explained above in Section 2.1. `Trace()` starts always in the direction of the positive Y-axis. After each move along a boundary crack *C* the values of *only two pixels R* and *L* of SON(*P*) of the end point *P* of *C* must be tested since the values of the other two pixels of SON(*P*) have been already tested during the previous move. For a detailed description of this algorithm see [10].

The pseudo-code of `Trace()`

`Image[NX, NY]` is a 2D array (standard raster) whose elements contain gray values or colors. The variables `P, R, L` and the elements of the arrays `right[4]`, `left[4]` and `step[4]` are structures each representing a 2D vector with integer coordinates, e.g. `P.X` and `P.Y`. The operation "+" stands for vector addition. Text after // is a comment.

```
void Trace(int x, int y, char image[])
{ P.X=x; P.Y=y; direction=1;
  do
  { R=P+right[direction];  // the "right" pixel
    L=P+left[direction];    // the "left" pixel
    if (image[R]==foreground)
      direction=(direction+1) MOD 4; // right turn
    else
      if (image[L]==background)
        direction=(direction+3) MOD 4; // left turn
    P=P+step[direction]; //a move in the new direction
  } while( P.X!=x || P.Y!=y);
} // end Trace
```

3.2 Tracing of surfaces in 3D

To trace surfaces of bodies in a 3D standard raster the method by Gordon and Udupa [6] uses the 2D technique in 2D slices. The code of a single closed surface is disconnected, i.e. it consists of isolated codes of the slices; 33% of the pixels are visited twice, which is not economical, and a body whose parts have only common cracks but no common faces are considered as disconnected.

A more efficient method producing a single connected sequence of code elements for each closed surface is that of [11]. According to the method the program chooses an arbitrary pixel of the surface *S* as the starting pixel and labels its closure. Then it traces the open frontier Of(*L*, *S*) (see Section 1.1) of the set *L* of labeled cells, encodes the pixels of Of(*L*, *S*) (1 byte per pixel), and labels the closures of simple pixels (Section 2.1). This ensures that *L* remains homeomorphic to a closed 2-ball. It has been proved that if the surface *S* is homeomorphic to a sphere then the traced

sequence is a Hamilton path: each pixel is visited exactly once. Otherwise there remain a few non-simple pixels which are visited twice. Their code elements are attached to the end of the sequence of simple pixels. Thus the code sequence is always connected.

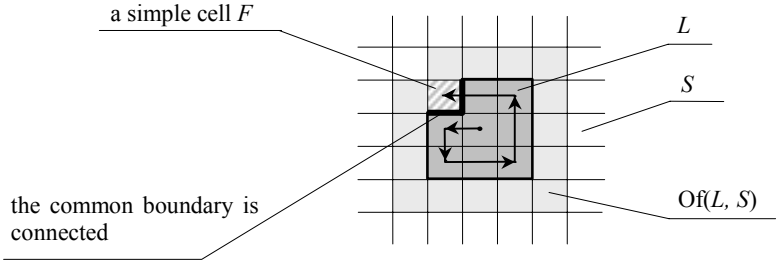

Fig. 7. The moves at the beginning of the tracing

A verbal description of the algorithm follows. A detailed description and the related proofs may be found in [11].

The algorithm

Notations: S is the surface to be traced. It must be a 2-quasi-manifold. $L \subset S$ is the subset of labeled cells; it is homeomorphic to a closed 2-ball. The "rest sequence" is the set of non-simple pixels at the stage when all simple pixels of S are already labeled. The rest sequence is empty if the genus of S is zero.

1. Take any pixel of S as the starting pixel F_0, label its closure and save its coordinates as the starting coordinates of the code. This is the seed of L. Denote any one crack of the boundary of Fr(F_0,S) as C_{old} and find the pixel F of S which is incident with C_{old} and adjacent to F_0. Set F_{old} equal to F_0 and the logical variable *REST* to FALSE. *REST* indicates that the tracing of the rest sequence is running.
2. (Start of the main loop) Find the crack C_{new} as the first unlabeled crack of Fr(F,S) encountered during the scanning of Fr(F,S) clockwise while starting with the end point of C_{old} which is in Fr(L,S). If there is no such crack and F is labeled stop the algorithm: the encoding of S is finished.
3. If F is simple label its closure.
4. Put the direction of the movement from F_{old} to C_{old} and that of the movement from C_{old} to F into the next byte of the code. If the pixel F is non-simple set the corresponding bit in the code (to recognize codes of non simple pixels in the ultimate sequence).
5. If *REST* is TRUE check, whether F is equal to F_{stop} and C_{new} is equal to C_{stop}. (These variables were defined in item 6 of the previous loop). If this is the case stop the algorithm and analyze the rest sequence to specify the genus of S as explained in [11]. Delete multiple occurrences of pixels from the rest sequence.
6. If F is simple set *REST* equal to FALSE; else set F_{stop} equal to F, C_{stop} equal to C_{new} and *REST* equal to TRUE.
7. Set F_{old} equal to F. Find the pixel F_{new} of S incident with C_{new} and adjacent to F. Set F equal to F_{new} and C_{old} equal to C_{new}. Go to item 2. **End of the algorithm**.

3.3 Filling the interiors of boundaries in multi-dimensional images

To test whether an n-cell P of an n-space lies in the interior of a given closed boundary it is necessary to count the intersections of the boundary with a ray from P to any point outside the space; however, it is difficult to distinguish between intersection and tangency (Fig. 8 a and b). The solution becomes easy if the boundary is given as one or many $(n-1)$-dimensional manifolds in a Cartesian AC complex and the "ray" is a sequence of alternating n- and $(n-1)$-cells all lying in one row of the raster (Fig. 8c).

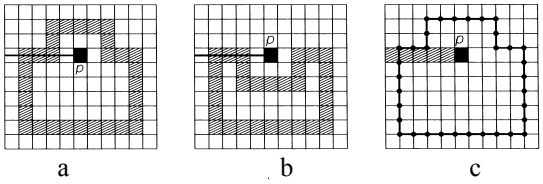

Fig. 8. Intersection (a) and tangency (b) are difficult to distinguish in "thick" boundaries; this is easy at boundaries in complexes (c)

In a 2D image the boundary must be a closed sequence of cracks and points (Fig. 8c). Then intersections are only possible at vertical cracks and the problem of distinguishing between intersections and points of tangency does not occur. The method has been successfully implemented for dimensions n=2, 3, 4.

The pseudo-code

Denote by `F` the current n-cell of the n-dimensional standard raster. Choose a coordinate axis A of the Cartesian space (e.g. A=X in the 2D case). Denote by `C(F)` the own $(n-1)$-cell of `F`, whose normal is parallel to A (e.g. the vertical crack of `F` in the 2D case). Label all $(n-1)$-cells of the given boundary whose normal is parallel to A. In the 2D case when A=X these are the vertical cracks of the given boundary.

```
for each row R parallel to A do
{ BOOLEAN fill=FALSE;
  for each n-cell F in the row R do
  { if C(F) is labeled then fill=1−fill; // inverting fill
    if fill is TRUE then F=foreground;
    else                 F=background;
  }
}
```

3.4 Component labeling in an n-dimensional space

We consider here the simplest case of a 2D binary image in a standard raster while the algorithm is applicable also to multi-valued and multi-dimensional images in a topological raster. It is expedient to consider a multi-dimensional image as a one-dimensional array $Image[N]$. For example, in the 2D case the pixel with coordinates (x, y) may be accessed as $Image[y \cdot NX+x]$ where NX is the number of pixels in a row.

In a standard raster a function must be given which specifies which raster elements are adjacent to each other and thus are connected if they have the same color. In our simple 2D example we use the well-known "8-adjacency" of the foreground pixels and the "4-adjacency" of the background pixels. In the general case the adjacency of the *n*-cells of an *n*-dimensional complex must be specified by rules specifying the membership of cells of lower dimensions [9] since an adjacency of *n*-cell depending on their "color" is not applicable for multi-valued images.

In a topological raster the connectivity of two cells is defined by their incidence which in turn is defined by their topological coordinates (see below Condition 3.6.3 in Section 3.6).

The algorithm

Given is a binary array `Image[]` of `N` elements and the functions `NumberNeighb(color)` and `Neighb(i,k)`: the first one returns the number of adjacent pixels depending on the color of a given pixel; the second one returns the index of the `k`th neighbor of the `i`th pixel. As the result of the labeling each pixel gets additionally (in another array `Label[]`) the label of the connected component which it belongs to.

Fig. 9. Illustration to the algorithm of component labeling

The pseudo-code

Allocate the array `Label[N]` of the same size as `Image[N]`. Each element of "Label" must have at least $\log_2 N$ bits, where `N` is the number of elements in `Image`. In the first loop each element of `Label` gets its own index as its value:

```
for (i=1; i<N; i++) Label[i]=i;
for (i=1; i < N; i++)
{ color=Image[i];
  for (j=0; j<NumberNeighb(color); j++)
  { k=Neighb(i, j); //the index of the jth neighbor of i
    if (Image[k]==color) SetEquivalent(i,k,Label);
  }
} // end of the first run
SecondRun(Label,N); // end of the algorithm
```

The subroutine `SetEquivalent()` makes the preparation for labeling the pixels having the indices `i` and `k` as belonging to one and the same component. For this purpose one of the pixels gets the index of the "root" of the other pixel. The function

`Root()` returns the last value in the sequence of indices where the first index `k` is that of the given pixel, the next one is the value of `Label[k]` etc. until `Label[k]` becomes equal to `k`. The subroutine `SecondRun()` replaces the value of `Label[k]` by the value of a component counter or by the root of `k` depending on whether `Label[k]` is equal to `k` or not.

Pseudo-codes of the subroutines

```
subroutine SetEquivalent(i,k,Label)
{ if (Root(i,Label)<Root(k,Label))
                  Label[Root(k,Label)]=Root(i,Label);
  else Label[Root(i,Label)]=Root(k,Label);
} // end of SetEquivalent

int Root(k, Label)
{ do
  { if (Label[k]==k) return k;
    k=Label[k];
  } while(1);
} // end of Root

subroutine SecondRun(Label,N)
{ count=1;
  for (i=0; i<N; i++)
  { value=Label[i];
    if (value==i)
    { Label[i]=count;
      count=count+1;
    }
    else Label[i]=Label[value];
  }
} // end of SecondRun
```

3.5 Skeleton of a set in 2D

Definition SK: The skeleton of a given set T in a two-dimensional image I is a subset $S \subset T$ with the properties:

a) S has the same number of connected components as T;
b) The number of connected components of $I–S$ is the same as that of $I–T$;
c) Certain singularities of T are retained in S.

Singularities may be defined e.g. as the "end points" in a 2D image or "borders of layers" in a 3D image etc.

A well-known difficulty in calculating skeletons is that it is impossible to remove all simple pixels simultaneously without violating the above conditions. However, representing an image as a complex C makes it possible to calculate the skeleton by a

procedure which may be either sequential or parallel. It is based on the notion of the *open frontier* (s. Section 1.1 above). The procedure consist in removing simple non-singular cells of T alternatively from the frontier Fr(T, C) and from the open frontier Of(T, C). A cell c of the frontier Fr(T, C) (respectively of Of(T, C)) is *simple* if the intersection of SON(c)−{c} (respectively Cl(c)−{c}) with both T and its complement C−T is connected. We present a simple version for a 2D topological raster.

The algorithm

Let C[NX, NY] be a 2D array with topological coordinates. The subset T is given by labeling cells of all dimensions of T: $C[x, y] > 0$ iff the cell $(x, y) \in T$. To delete a cell means to set its label $C[x, y]$ to zero. A 0- or 2-cell c is considered as *singular* iff it is incident with exactly one labeled cell other than c.

To calculate the skeleton of T run the following loop:

```
do  { Scan C and delete all simple and non-singular cells of T∩Fr(T, C);
      CountClose = number of cells deleted during this scan;
      Scan C and delete all simple and non-singular cells of T∩Of(T, C);
      CountOpen = number of cells deleted during this scan;
    } while (CountClose+CountOpen > 0);
    // end of Algorithm
```

Fig. 10 shows an example.

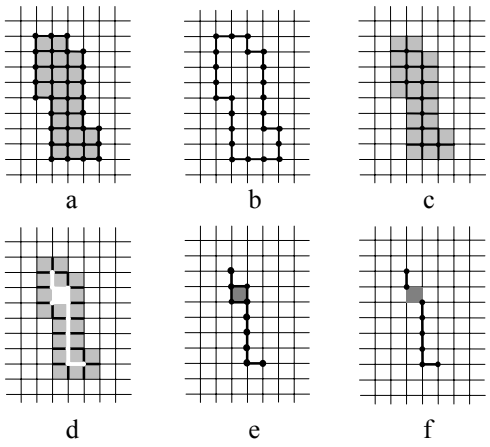

Fig. 10. a) a given 2D subcomplex T; b) its frontier Fr; c) the set T–Fr: the simple cells of the frontier deleted; d) the open frontier Of of the set T–Fr; e) the set T–Fr–Of: the simple cells of the open frontier deleted; f) the skeleton

The result may be, if desired, easily transformed either to a sequence of pixels or to 1-complex containing only points and cracks.

3.6 Algorithms for topological investigations

Topological computations are particularly simple in a Cartesian complex with topological coordinates. We present in the following sections some basic algorithms.

3.6.1 Computing the dimension of a cell in an *n*-dimensional space

If $\boldsymbol{X}=(X_1, X_2, ..., X_n)$ is a cell of an n-dimensional Cartesian complex then

$$Dimension(X) = \sum_{i=1}^{n} X_i MOD\ 2.$$

3.6.2 Condition of bounding in an *n*-complex: the cell *A* bounds the cell *B*

A_i is the ith coordinate of the cell A; $\dim(A_i)$ is the dimension of A_i in the ith coordinate axis ($\dim(A_i)$ is either 0 or 1).

The condition: $\forall i = 1...n; \dim(A_i) \le \dim(B_i) \wedge MaxDif = 1;$

where $\dim(A_i) = A_i \text{ MOD } 2$; and $MaxDif = \max |A_i - B_i|$; $i=1...n$.

3.6.3 Condition of incidence: the cell *A* is incident with the cell *B*

A bounds B OR B bounds A OR $A = B$;

3.6.4 Computing the SON of a *k*-cell *A* in an *n*-dimensional space

To explain the idea we first show as an example all cells of the SON of a 1-cell A in a 3D space. Let k=1 and A=(2, 3, 6). A has *two even coordinates*. It is possible to change *one or both of them* by ±1 to get a cell bounded by A.

The number N of cells bounded by A: $N = 2 \cdot C_2^1 + 2^2 \cdot C_2^2 = 2 \cdot 2 + 4 \cdot 1 = 8$; where C_k^i denotes the number of combinations of i elements from k.

The cells bounded by A are:

(1, 3, 6)
(3, 3, 6)
(2, 3, 5)
(2, 3, 7)
(1, 3, 5)
(1, 3, 7)
(3, 3, 5)
(3, 3, 7)

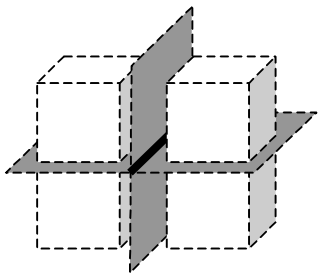

Fig. 11. The SON of a 1-cell in a 3D space

Now we present the pseudo-code of a function computing the SON of a cell "`Cell`" in a 4-dimensional space. The coordinates of all cells $C \in$ SON(*Cell*) will be saved in the array `SON[ ][4]`.

The pseudo-code

```
void SaveSON(int Cell[4], int SON[ ][4])
{ int C[4], step[4];
  //"step" contains the increments of the coordinates: 1 for even and 0 for odd ones
  for (k=0; k<4; k++)
  { if ((Cell[k] MOD 2)==0) step[k]=1;
    else step[k]=0;
  }
  i=0;
  // four nested loops; C runs through all cells bounded by Cell:
  for (C[3]=Cell[3]-step[3]; C[3]≤Cell[3]+step[3];C[3]++)
  for (C[2]=Cell[2]−step[2]; C[2]≤Cell[2]+step[2];C[2]++)
  for (C[1]=Cell[1]−step[1]; C[1]≤Cell[1]+step[1];C[1]++)
  for (C[0]=Cell[0]−step[0]; C[0]≤Cell[0]+step[0];C[0]++)
  { for (k=0; k<4; k++) SON[i][k]=C[k];
    i=i+1;
  }
} // end of SaveSON
```

A similar algorithm for computing the closure of a cell in a 4-dimensional space:

```
void SaveClosure(int Cell[4], int Cl[ ][4])
{ int C[4], step[4];
  //"step" contains the increments of the coordinates: 0 for even and 1 for odd ones
  for (k=0; k<4; k++)
  { if ((Cell[k] MOD 2)==1) step[k]=1;
    else step[k]=0;
  }
  i=0;
  // four nested loops; C runs through all cells bounding Cell:
  for (C[3]=Cell[3]−step[3]; C[3]≤Cell[3]+step[3];C[3]++)
  for (C[2]=Cell[2]−step[2]; C[2]≤Cell[2]+step[2];C[2]++)
  for (C[1]=Cell[1]−step[1]; C[1]≤Cell[1]+step[1];C[1]++)
  for (C[0]=Cell[0]−step[0]; C[0]≤Cell[0]+step[0];C[0]++)
  { for (k=0; k<4; k++) Cl[i][k]=C[k];
    i=i+1;
  }
} // end of SaveClosure
```

Conclusion

We have demonstrated that abstract cell complexes may be successfully used for modeling locally finite topological spaces satisfying the classical axioms on a computer and for solving topological problems. We have suggested a new data structure, the 3D cell list, for encoding 3D segmented images or 3-manifolds in such a way that topological relations between subsets may be immediately extracted from the structure without searching. Another important property of the cell list is the possibility to consistently describe the so-called non-proper complexes having the advantage of being very simple although not representable by classical means since in a non-proper complex a cell may *multiply* bound another cell.

We have presented descriptions and/or pseudo-codes of seven basic algorithms for computing topological features of subsets of an abstract cell complex represented on a computer as an n-dimensional array, $n \leq 4$. Some of the described algorithms are the necessary tools when implementing high-level topological algorithms, namely:

- Automatic calculation the cell list of a segmented (labeled) n-dimensional image (n=2 or 3);
- Handle decomposition of a 3-manifold in a 4D space, which is useful for comparing two manifolds with each other.
- Identification of faces of a polyhedron represented by a cell list and producing the cell list of a 3-manifold with the aim of comparing two manifolds with each other.
- Modeling of linked spheres in an n-dimensional space (n=3, 4, 5) with the aim to experimentally investigate their topological properties.

The letter algorithms have been developed and successfully tested by the author, however, they cannot be described here because of paper size limitations.

An important open problem is, whether the described approach, which yields very simple descriptions of 3-manifolds, may contribute to the problem of their classification.

References

1. Alexandroff, P.: Diskrete Räume. Mat. Sbornik. 2 (1937) 501-518

2. Barett, J.: Quantum Gravity as Topological Quantum Field Theory. Jour. Math. Phys. 36 (1995) 6161-6179

3. Bertrand, Y., Fiorio, Ch., Pennaneach, Y.: Border Map: A Topological Representation for nD Image Analysis. In: Bertrand, G., Couprie, M., Perroton, L. (eds.): Discrete Geometry for Computer Imagery. Lecture Notes in Computer Science, Vol. 1568, Springer-Verlag, Berlin Heidelberg New York (1999) 242-257.

4. Encarnacao, J., Strassler, W., Klein, R.: Computer Graphics. R. Oldenbourg Verlag, Munich (1997)

5. Fiorio, Ch.: A Topologically Consistent Representation for Image Analysis: The Frontier Topological Graph. In: Miguet, S., Montanvert, A., Ubéda, S. (eds.): Discrete Geometry for Computer Imagery. Lecture Notes in Computer Science, Vol. 1176, Springer-Verlag, Berlin Heidelberg New York (1996) 151-162

6. Gordon, D., Udupa, J.K.: Fast Surface Tracking in Three-Dimensional Binary Images. Computer Vision, Graphics and Image Processing 45 (1989) 196-214,

7. Klette, R.: Cell Complexes through Time. University of Auckland, CITR-TR-60, June 2000.

8. Kovalevsky, V.: On the Topology of Digital Spaces. In: Proceedings of the Seminar "Digital Image Processing",.Technical University of Dresden (1986) 1-16

9. Kovalevsky, V.: Finite Topology as Applied to Image Analysis. Computer Vision, Graphics and Image Processing 45 (1989) 141-161

10. Kovalevsky, V.: Finite Topology and Image Analysis. In: Hawkes, P. (ed.): Advances in Electronics and Electron Physics, Vol. 84. Academic Press (1992) 197-259

11. Kovalevsky, V.: A Topological Method of Surface Representation. In: Bertrand, G., Couprie, M., Perroton, L. (eds.): Discrete Geometry for Computer Imagery. Lecture Notes in Computer Science, Vol. 1568. Springer-Verlag, Berlin Heidelberg New York (1999), 118-135

12. Kovalevsky, V.: A New Means for Investigating 3-Manifolds. In: Borgefors, G., Nyström, I., Sanniti di Baja, G. (eds.): Discrete Geometry for Computer Imagery. Lecture Notes in Computer Science, Vol. 1953. Springer-Verlag, Berlin Heidelberg New York (2000) 57-68

13. Lawrence, R.: Triangulation, Categories and Extended Field Theories. In: Baadhio, R., Kauffman, L. (eds.): Quantum Topology. World Scientific, Singapore (1993) 191-208

14. Matveev, S.: Computer Classification of 3-Manifolds. Russian Journal of Mathematical Physics 7 (2000) 319-329

15. Rinow, W.: Textbook of Topology. VEB Deutscher Verlag der Wissenschaft, Berlin (1975)

16. Steinitz, E.:Beitraege zur Analysis Situs, Sitzungsbericht Berliner Mathematischer Gesellschaft, Vol.7. (1908) 29-49

17. Stillwell, J.: Classical Topology and Combinatorial Group Theory. Springer-Verlag, Berlin Heidelberg New York (1995)