Feature Extraction and Classification from Boundary Representation

David Podgorelec and Borut Žalik

Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia

In the paper, an algorithm for explicit feature extraction and classification from boundary representation is presented. It operates in two phases: the topological and the geometrical. While the topological part is just an adaptation of an already known algorithm, the geometrical part represents an original and new solution. In this part, the algorithm manipulates with features filled by material and the empty ones. The algorithm classifies extracted features into eight classes. It successfully and efficiently handles voids, nested features and many cases of mutual feature intersections. The time complexity depends on input data, and never exceeds $O(n^2)$.

Keywords: geometric (form) features, feature recognition, geometric modelling, boundary representation, CAD.

1. Introduction

Feature–based design of parts has been first introduced to facilitate NC–programming and computer–aided process planning (CAPP) while all other representations have not offered satisfactory level of abstraction. Beside to the geometrical and topological data provided by a solid model, some explicit information on geometric features of designed objects are necessary. The geometric feature (form feature, shape–based feature or simply the feature) is a set of geometric and topological entities with some context–dependent functional meaning [1]. A three–dimensional object can be described as a set of geometric features organised in a hierarchical structure. While the final shape of a product is usually not known in detail in the design phase already, automatic procedures for **feature recognition** in the CAD system have to be provided. The feature recognition is executed in two steps:

- **feature extraction** is based on procedures that recognise features and extract them from the model. Concerning user's requests, some algorithms are able to remove the features from the model, but more commonly, the features are being completed with missing elements to valid geometric bodies.
- **feature classification** classifies extracted features into classes, for example: protrusions, depressions, bridges, handles, and through–holes.

Different authors have proposed various methods able to recognise different types of features. Some of them [2] have studied how to transform the CSG model into a presentation convenient for feature recognition, but the majority of authors still project their algorithms on boundary representation. Ferreira and Hinduja [3] have described the method based on convex hulls of particular faces. Most of the form–features on 2.5D components can be recognised, and the method can be extended to handle scenes with non–planar faces, but it does not recognise voids. Sandiford and Hinduja again [4] presented an algorithm, which uses the concavity of faces, edges and vertices to detect features and their intersections. Simple rules have been developed, which allow intersections of quadric surfaces. Intersections of faces adjacent to the feature faces are used to create new edges, and these are employed to create totally new construction faces. These faces are then used to complete the boundary of the feature. An interesting approach was proposed by Meeran and Taib [5]. It recognises isolated,

nested and interacting features from 2D orthographic projections through a two–stage process of profile searching and feature completion. De Floriani [6] paid more attention to topological algorithms. She has described the method for extraction of particular feature types from the relational boundary model called the generalised edge–face graph (GEFG). The algorithm is based on decomposition of GEFG into biconnected and triconnected subgraphs, and it efficiently extracts simple topological shapes that define one or more rings. De Floriani and Bruzzone [7] have also described another topological algorithm based on ring identification and marking biconnected components in so–called symmetric boundary graph. Each shell of object's boundary is described with collections of faces, edges and vertices. Six different relations between these entity types are employed. In this way, later geometrical classification based on concave edges of rings is importantly facilitated.

Falcidieno and Giannini [1] have extended feature recognition with the third step. **Feature organisation** connects extracted features into a hierarchical feature graph. In this way, representation of assemblies of parts is supported. Namely, the majority of engineering problems have to be solved by assemblies rather than by single parts. The representation and manipulation of assemblies involve structural and spatial relationships between individual parts at higher level of abstraction than the representation of single parts. Such a representation must support construction of an assembly from all given parts and its editability: selection of individual parts in the assembly, changes of relative positions of parts, and manipulation of the assembly as a whole [8]. Geometric constraints and their automatic solving offer a solution to this problem [9, 10]. A geometric constraint is a relation among geometric entities that should be satisfied. At the beginning, the constraints were usually employed on geometric elements composing an object: points, line segments, circles and curves, but they can also efficiently describe relations among geometric features i.e. objects composing an assembly. Martini [11] has described a hierarchical approach that employs both, shape constraints to define parts of assembly, and location constraints at higher level of hierarchy to define connections among these parts, and to facilitate their manipulation. The implementation was inspired by the problems in the building design. In contrary, Gui and Mäntylä [12] have stressed advantages of so–called top–down assembly design, where a designer starts with a general mechanical design prototype, and develops an assembly model as an instance of this prototype. Many other authors like Anantha, Kramer and Crawford [8] have also contributed to further integration of geometric features and constraints.

In the presented paper, we propose a solution for explicit feature extraction and classification from the boundary representation. Basic definitions and feature types being recognised are explained in the next chapter. After this, two steps forming the algorithm are highlighted: the topological and the geometrical phase. The first one was inspired by work of De Floriani and Bruzzone [7], but the geometrical part, which divides each of four classes of extracted features into the features filled by material and the empty ones, represents an original and new solution. At the end, the time complexity and applicability of the method are discussed, and our future work in this area is stressed.

2. Geometric Features

To understand the rest of the paper, the reader should be familiar with some basic terms of geometric modelling: a solid (a body, a shape), a surface, a shell, an edge, a vertex, a face, a loop, and a ring. The definitions can be found in [13] for example. The face is **simply connected** if its boundary consists of a single loop. Otherwise, it is **multiple connected**. In this case, one of the loops surrounding all others is said to be the external loop (or simply the loop), and the others are called rings or internal loops. In a boundary representation, a geometric feature is a connected set of faces that form a part of the object's boundary and have some particular function in the design or manufacturing process [7].

According to the geometry, we can distinguish between external and internal features. Former define external shape, and latter define the interior of the object. Most of the algorithms for feature recognition also divide features into explicit and implicit features. **Explicit features** are based on rings, and can be found only on

multiple connected faces. An explicit feature F is fastened to the rest of the solid by a set of rings. We use expression that the feature F defines rings $r_1, \ldots, r_m$ on the object's surface, and each of these rings is said to open the feature F on the surface. According to the number of rings, we distinguish between:

- **DP–features**, which define a single ring. Representatives of this class are **an explicit protrusion** (Fig. 1a) and **an explicit depression** or **a pocket** (Fig. 1b).
- **H–features**, which define two or more rings on one or more faces of the object. These faces can belong to different features of the object. **A handle** (Fig. 1c) is fastened to a single feature, and a **through–hole** (Fig. 1d) tunnels just one feature as well. Two or more features can be connected by **a bridge** (Fig. 1e).

Algorithms for explicit feature recognition usually complete features with missing faces. In this way, each feature becomes a regular geometric solid. An algorithm should create a new face for each ring, and the observed ring becomes the external loop of the new face. However, in the resulting collection of regular 3D shapes, there are also features, which have not defined any rings in the past. We name them **main shapes**. Only the topological data suffices for the object decomposition and classification of extracted features into main shapes, DP–features and H–features, but such topological algorithms are typically used only as preprocessors to geometric algorithms that further classify features from particular classes into external and internal. This geometic classification is usually done by determining convexity of edges forming the rings [7]. External features are protrusions, handles and bridges, and internal are pockets and through–holes. An internal equivalent to the bridge is called **a bridge between internal features** (Fig. 1g and Fig. 1h). The main shape can also be either external or internal. The latter is called **a void** (Fig. 1f). The meaning of the hierarchical structures in Fig. 1a–h will be described later.

The second category consists of **implicit features**. They are not based on rings, but on concave edges of external loops. Algorithms for their recognition are projected on geometrical data. The task of testing convexity of edges can be transformed into the problem of face orientation (determining the exterior and the interior side of the face). An implicit depression is

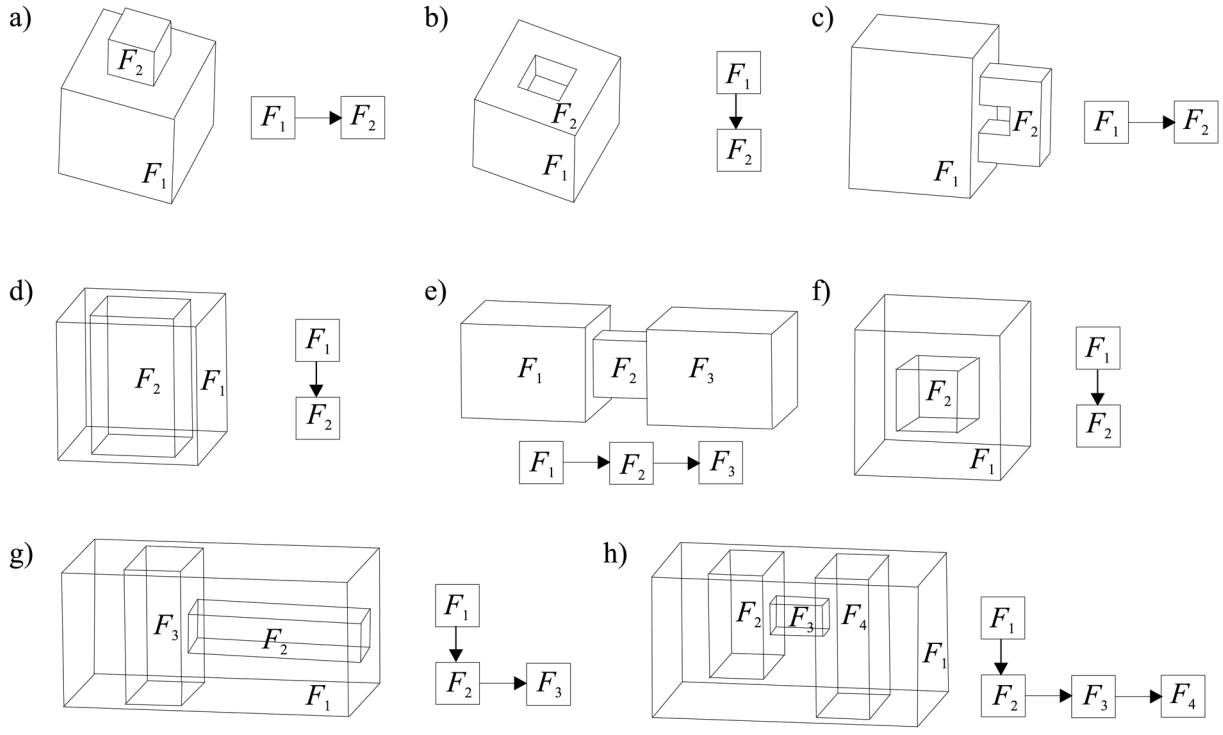

Fig. 1. Simple combinations of explicit features and corresponding hierarchical structures.

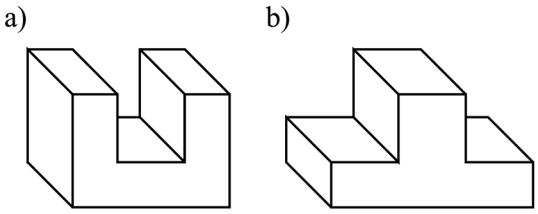

Fig. 2. A shape with an implicit a) depression, b) protrusion.

shown in Fig. 2a, and an implicit protrusion can be seen in Fig. 2b.

3. Algorithm for Recognition of Explicit Geometric Features

The basic idea of the presented algorithm can be employed on any geometric shape, but the current implementation of some supporting functions enables only feature recognition in scenes with planar faces. Explicit features being recognised can be opened by arbitrary number of rings, and these can belong to different features. But a ring is allowed to be defined on a single face of a feature only. This feature is said to be external to the feature opened by the ring. The limitation means that features expanding through more faces like a protrusion on an edge cannot be recognised.

The method is not based on usual division to external and internal features. The features are rather classified as **filled** (by material) and **empty** features. This proves sensible when nested features are present. Namely, it sounds strange and confusing that some external feature could be included in an internal one. The method operates in two phases:

- **a topological part** completes the model with new faces encircled by rings, divides the scene into simpler valid geometric bodies (features), and classifies these features into four classes: main shapes, DP–features, H–features attached to a single feature, and H–features connecting more features. Only topological data are employed.
- **a geometrical part** classifies extracted features into filled and empty ones. Each of four classes is split into two new classes. The classification is not being executed by testing convexity of edges in rings, but rather by determining eventual mutual containment of features. The problem of penetration is partially handled, too.

Besides the geometrical and topological data, the employed boundary model includes fields that characterise features, and some additional information for the program control. A solid is presented by faces, loops, edges and vertices as shown in Fig. 3. Arrows indicate accessibility of entities of a particular type. Three lists of geometric elements are directly accessible through the solid object: faces, edges and vertices. However, loops cannot be reached directly but through faces (the relation face–loop). Multiple connected faces are encircled by several loops. In the data structure, they are organised in a double–connected list. The first list element presents the external loop. It can be followed by several elements presenting eventual rings. After the decomposition, the face–loop relation becomes one–to–one, and we do not have to distinguish between both entity types any more. The inverse relation loop–face enables us to access both elements simultaneously. Loops are also accessible through edges. The relation edge–loop could seem a bit strange or at least redundant to someone, but it speeds up some parts of the program considerably.

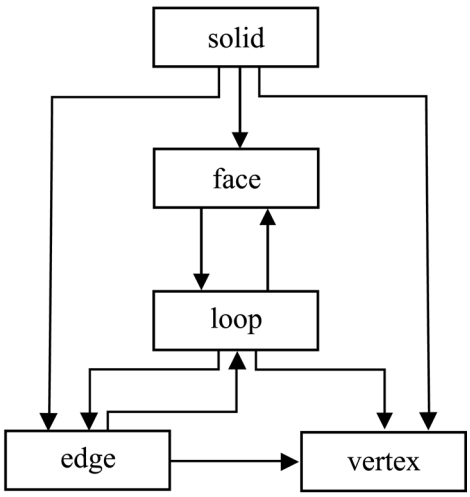

Fig. 3. Object's boundary representation.

4. Topological Part of the Algorithm

In the first part of the algorithm, the feature extraction and the first step of the feature classification are done. The employed topological algorithm is just an adaptation of the algorithm described by De Floriani and Bruzzone [7] with some modifications of the data structures. While only the topological data are employed, this step works successfully with objects with non–planar faces as well. The feature extraction is carried out in two steps:

Creation of new faces – new face is created for each ring, and the ring becomes its external loop. At the same time, it is removed from the list of loops of previously multiple connected face. This face is said to be external to the newly created face. After the removal of all rings, we obtain the set of simply connected faces.

Object decomposition – the scene is decomposed into features with regard to adjacency of edges and vertices only. The features are simpler bodies bounded by simply connected faces only. In the first part of the decomposition, each face is tested against all faces situated in the face list before it. The test searches for faces that share common edges and therefore belong to the same feature. Such faces are called combinable. The same term is used for the feature candidates that should be joined. Although all necessary information are accessible directly from the boundary representation, some additional fields in the data structure speed up the process considerably. The description of the face is extended by the fields feature_ID and edge_set. Two arrays of integers: combinable and correction are also employed in this step. They refer to feature candidates, and the current number of them is stored in number_of_features, which may not exceed the predefined constant max_features. The edge_set has to be initialised with indices of all edges of the external loop of the corresponding face i.e. with their relative positions in the edge list, and all fields feature_ID have to be set to some value larger than the real number of features (max_features + 1). Three different situations can occur during testing a face against preceding members of the face list:

(i) The face currently being tested is not combinable with any other face. New feature candidate is identified. The number_of_features is incremented by one, and the fields combinable[number_of_features] and correction [number_of_features] are employed then. While such a face and a corresponding feature are not combinable with any other identified since that moment, the value of combinable is set to its index.

(ii) The tested face f_t is combinable with a face f_p belonging to the feature indexed i, and f_t is already combinable with some other face belonging to the feature j, where $j < i$. The following actions have to be performed:

```
f[t].edge_set = ⋃(f[t].edge_set, f[p].edge_set);

f[p].edge_set = f[t].edge_set

combinable[f[p].feature_ID] = f[t].feature_ID;
```

(iii) The face currently being tested (f_t) is combinable with a face f_p belonging to the feature i, and f_t has not been combinable with any face yet, or it is eventually combinable with some face belonging to the feature j, where $j > i$. The following actions have to be performed:

```
f[t].edge_set= ⋃(f[t].edge_set, f[p].edge_set);

f[p].edge_set = f[t].edge_set

f[t].feature_ID = combinable[f[p].feature_ID];
```

It is obvious that the values combinable are forced to become as low as possible. They are initialised with their indices and later, they can be only decreased. We are especially interested in those fields that keep their values unchanged after all comparisons. These fields actually present real features, and will be called feature–introducing elements (FIEs). Currently, each face is assigned to one of number_of_features feature candidates, but we want them all assign to the FIEs only. The number_of_features should present number of FIEs, and indices of FIEs have to be transformed into the range [1, number_of_features].

In the continuation, we first calculate corrections of indices of FIEs and the correct number_of_features. Then we find the FIE and repair value of feature_ID for each face. The procedures

are given below. Finally, we are ready to perform the real object decomposition by splitting common lists of faces, edges and vertices into partial ones describing particular features. At this stage, we can already initialise some auxiliary data refering to the features: coordinates of the bounding box needed for the minimax test in the geometrical part of the algorithm, the number of newly created faces of the feature (number_of_rings), and the number of features that contain the rings describing these newly created faces (number_of_parents).

```
Algorithm FindFIE (face, fie)
begin
  fie = combinable[face.feature_ID];
  loop
    if (fie = combinable[fie]) then exit loop;
      fie = combinable[fie];
  forever;
end.

...
(* calculate correct feature_ID of face f *)
FindFIE (f, fie);
f.feature_ID = fie correction[fie];
...
```

Feature extraction is followed by **a topological part of the feature classification**. In the feature's data structure there is also the integer variable feature_class, which is set to value between 1 and 4:

1 – main shape
(number_of_rings = number_of_parents = 0),
2 – DP–feature
(number_of_rings = 1),
3 – H–feature on one feature
(number_of_rings $>$ 1, number_of_parents = 1),
4 – bridge
(number_of_parents $>$ 1).

An example of feature extraction and topological classification is given in an appendix.

5. Geometrical Part of the Algorithm

The topological algorithm represents a preprocessor for geometric classification. We cannot imagine an application that would treat a depression or protrusion, for example, in the same way. If the manufacturing machine obtains information to process a DP–feature, it "does not know" whether to remove the interior or the exterior of the feature. Most common geometrical classification distinguishes between external and internal geometric features. The classification usually tests concavity of the edges forming rings, and meet the complex problem of face orientation [14]. As long as main shapes are supposed to be filled by material, the problem is solved easily, but it becomes a hard one if voids are allowed. A protrusion on void does not define a concave, but a convex edge. Before performing any tests of concavity of edges, the differentiation of filled and empty main shapes has to be done. In addition to this, we allow features nested to arbitrary level to increase generality and applicability of the method. Therefore, a hierarchical structure connecting main shapes according to their eventual mutual containment is employed. Features not contained in any other feature are placed at the root level, and at the level n, we meet the features directly contained in the features of the level $n-1$. Features placed at odd levels are identified as filled, and those at even level are empty features. However, voids can be contained in any filled feature and not only in main shapes. For this reason, all the features should be arranged in the hierarchical structure, and therefore, they can also be classified according to odd or even level of appearance. Before describing the details, the main procedure of the geometrical part of our algorithm is listed.

```
Algorithm ClassifyFilledEmpty()
begin
  elt = CreateNode(1);
  rootNode = elt;
  j = 2;
  loop
    if (j > number_of_features) then exit loop;
    elt = CreateNode(j);
    rootNode = InsertNodeToStructure(elt, rootNode);
    j = j + 1;
  forever;
  ClassificationOddEven(rootNode, 1);
end.
```

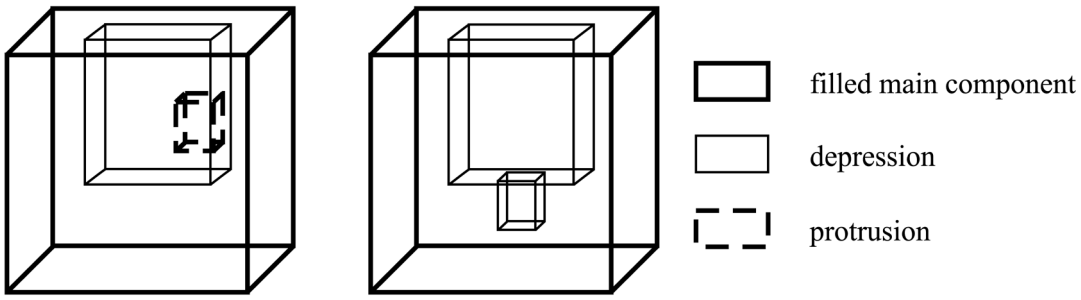

Fig. 4. Left: depression in depression is protrusion. Right: nested depression.

5.1. Relation of Containment

Function InsertNodeToStructure presents the only point where the algorithm meets geometrical data. Namely, determination of eventual feature intersections and the containment test are hidden inside its body. We suppose that only an empty feature can be directly contained in the filled one and the opposite, the filled feature always occupies some empty space and not the space occupied by some other filled feature. In this way, we obtain the desired result: empty features are at even levels in the hierarchical structure, and filled features are at odd levels. A depression in a depression has to be recognised as protrusion, and a nested depression is still a depression. Both examples are shown in Fig. 4.

The containment test has to identify whether the tested feature is situated inside another feature or not. We use a 3D extension of simple containment test point–in–polygon to find out whether the point is inside the body or not [15]. While the test is performed after handling possible feature intersections, it suffices to test a single point of the tested feature against all faces of the other feature. Unnecessary calculations can be avoided by performing a simple minimax test.

5.2. Auxiliary Hierarchical Structure

Geometrical classification is facilitated by employing the hierarchical structure connecting nodes (features) according to their mutual containment or intersections. Two nodes at the same level are not contained in each other, and will be called **siblings**. The feature F_i contained in some other feature F_j is recursively inserted into the substructure rooted in F_j, among the **descendants** of F_j. Direct descendants are called **sons**. Analogously, nodes that contain some other node (F_i) are called **ancestors** of F_i. Direct ancestor is **father**. Each node, except the ones at the root level, has exactly one father. Nodes are accessible through the ancestors, and the inverse connections are not defined. The number of sons is arbitrary, but in the structure, only a pointer to one of them is used. All its siblings are listed behind it in the connected list. Each list is connected in one direction only, and therefore, we shall distinguish between left and right siblings. When we reach a particular node, its left siblings have been visited already, and the right siblings and the descendants of the node form the substructure that should be visited next. Each node contains the pointer to the first node in the list of its sons (first_son), the pointer to the next right sibling (first_sibling) and the field feature_ID that establishes a unique relation between the node and the corresponding feature obtained in the topological part.

In Fig. 1 already, all types of features recognised by our algorithm and the corresponding hierarchical structures are given. Horizontal connections present siblings, and the vertical are used for father–son relations. Unfortunately, this simple organisation is useful only while feature intersections are not allowed. When we meet a feature F_i that is partially contained inside some other feature F_j, and the rest of F_i is outside F_j, then we cannot decide whether to insert the corresponding node F_i between the siblings of F_j or between its descendants. In such case we use two or more copies of the node. In the next section, we propose a solution to this problem.

5.3. The Problem of Feature Intersections

When we talk about general and really usable algorithm, we cannot avoid the problem of feature intersections. Possible appearance of intersections of solid bodies presents one of the main problems in computer graphics and geometric modelling. In our algorithm, only the real scenes are expected. Filled feature can be directly (partially) contained only in an empty feature, and similarly, empty feature can be directly partially contained in a filled one only. We rather use the term **partial containment** instead of feature intersection to show that one feature is pierced by another in such situation. The node of feature F_i piercing the feature F_j is cloned, and the original is inserted among the sons of F_j, and the copy is inserted into the list of right siblings of F_j. The first copy is presenting a part of F_i that is contained in F_j, and the second copy is used for the part of F_i outside F_j. After study of both examples in Fig. 5, the reader will understand why two copies of the protrusion F_3 are necessary. In case a), the protrusion is placed among descendants of the main shape F_1 and the depression F_2 only, and the void F_4, which is situated outside the main shape F_1, is incorrectly classified as the filled feature. In case b), the protrusion F_3 is inserted to the list of siblings of the main shape F_1, and the void F_4 between the descendants of the feature F_1. The void F_4 is again incorrectly recognised as filled feature.

Possible intersections are determined by searching for the intersections of the faces belonging to the first feature and those belonging to the second one. If two faces are intersecting, the procedure can be released. While the polygons presenting faces can be concave as well, it does not suffice to test only two line segments, but to find all intervals on line obtained as the intersection of two planes. It is wise to employ the test minimax. If the intersection is not identified after testing all pairs of intervals, the next pair of faces is being tested. If no intersections are identified after all possible tests, the containment test can be employed on the same pair of features.

5.4. Creating the Hierarchical Structure

Two functions are employed to create the hierarchical structure. The simple one named CreateNode(int j) creates the node presenting the feature F_j, and the second one InsertNodeToStructure(elt, rootNode) recursively inserts the node elt into the substructure with root rootNode. The hierarchical structure should be organised in the way that all the copies presenting the same feature have common sons, but different siblings. Six different situations can occur while inserting elt in the structure:

1. Substructure rootNode is empty. The node elt becomes the root of the substructure.
2. Feature elt is piercing the feature rootNode. Node elt is recursively inserted into the substructure of the descendants of rootNode. If elt is not a sibling of the rootNode yet (the

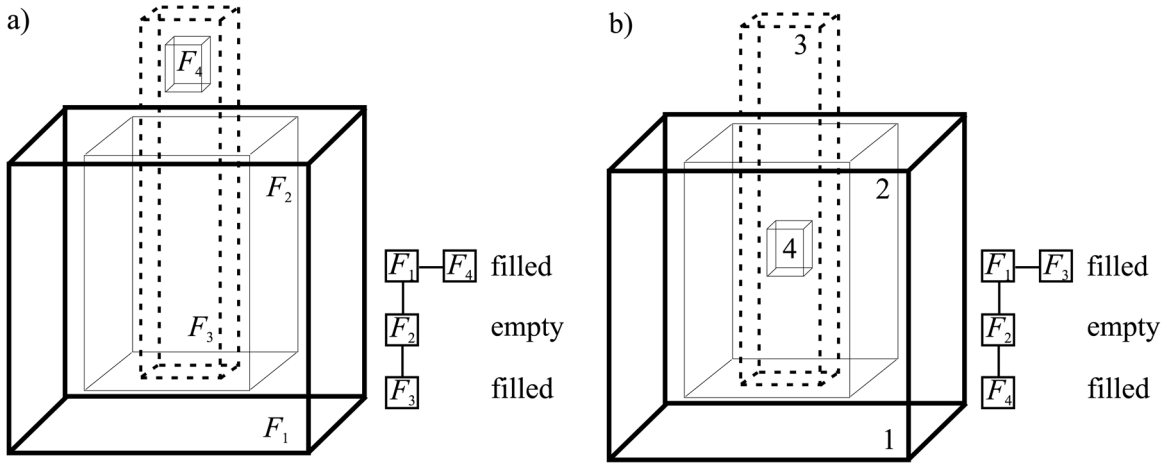

Fig. 5. The piercing feature is not only a descendant of the pierced one (a) and not only a neighbour (b) - the void F_4 is incorrectly recognised as filled feature.

Boolean field visited is employed for this task), a copy of elt is inserted into the substructure of right siblings of rootNode.

3. The node rootNode is piercing the new node elt. The rootNode is cloned, and a copy is inserted in the structure of descendants of elt. The node elt occupies the place of the rootNode in the structure, another copy of the rootNode is inserted in the structure of right siblings of elt, and all the previous right siblings of the rootNode are recursively inserted into the substructure with the new root node elt.
4. New node elt is contained in the root rootNode. The node elt is recursively inserted into the structure of descendants of rootNode.
5. The node rootNode is contained in new node elt. The rootNode is cloned, and a copy is inserted in the structure of descendants of elt. The node elt occupies the place of the original rootNode, and all the previous right siblings of the rootNode are recursively inserted into the substructure with the new root node elt.
6. New node elt and the root of the structure rootNode do not intersect and are not contained in each other as well. The node elt is recursively inserted into the structure of right siblings of rootNode.

5.5. Geometrical Classification to Filled and Empty Features

The nodes at odd levels are presenting filled features and the nodes at even levels correspond to empty features. Each node contains the integer attribute feature_ID that enables direct access to the data structure presenting the feature. We remember that the topological part has set the value feature_class in this structure to the integer number between 1 and 4. The geometrical classification only changes the sign of the feature_class for all features with the corresponding nodes at even levels of the hierarchical structure. In this way, we split each of the four classes into two new classes, and therefore, obtain eight feature classes. But the problem appears because multiple copies presenting the same feature can be located at both, odd and even levels. Without mathematical proof, we shall declare that in the depth–first search of the structure, the first copy (the left–most copy) of the node met is correctly nested. For both examples from Fig. 5, we obtain the structure shown in Fig. 6. The left most copy of the protrusion F_3 is correctly nested at the third (odd) level. Right copies $F_{3,1}$ and $F_{3,2}$ are not tested, and therefore, the incorrect even level of the node $F_{3,1}$ does not have any influence to the solution. The common descendant of all three copies (void F_4) is met after visiting the left most copy of its fathers, and it is again correctly set at the fourth (even) level.

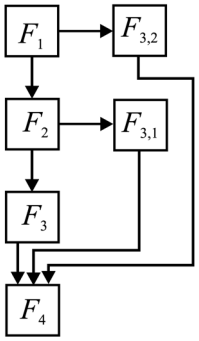

Fig. 6. Incorrect level of the right copy ($F_{3,1}$) does not affect the solution.

In Fig. 7, two more complex examples are given. In both cases, we meet the same problem: the existance of features that are pairly intersecting each other, and we cannot specify which one is the piercing feature and which is the pierced one. While the algorithm can only handle this type of feature intersections, it simply chooses one of the intersecting features to be the piercing one and another to be the pierced feature. The choice depends on indices of features and also on ordering of faces of both features. However, we have tested all possible orders of features for both examples, and the obtained results were correct in all cases: the left most copies of solid features are situated at odd levels, and the left most copies of empty features can be met at even levels only.

In the example a), we have two solid main shapes and each of them contains a depression. Four features can be ordered in $4! = 24$ different ways, but because of the symmetry, only 12 of them have to be observed. But in all cases, there are two possibilities: the solid feature with lower index is piercing the solid feature with

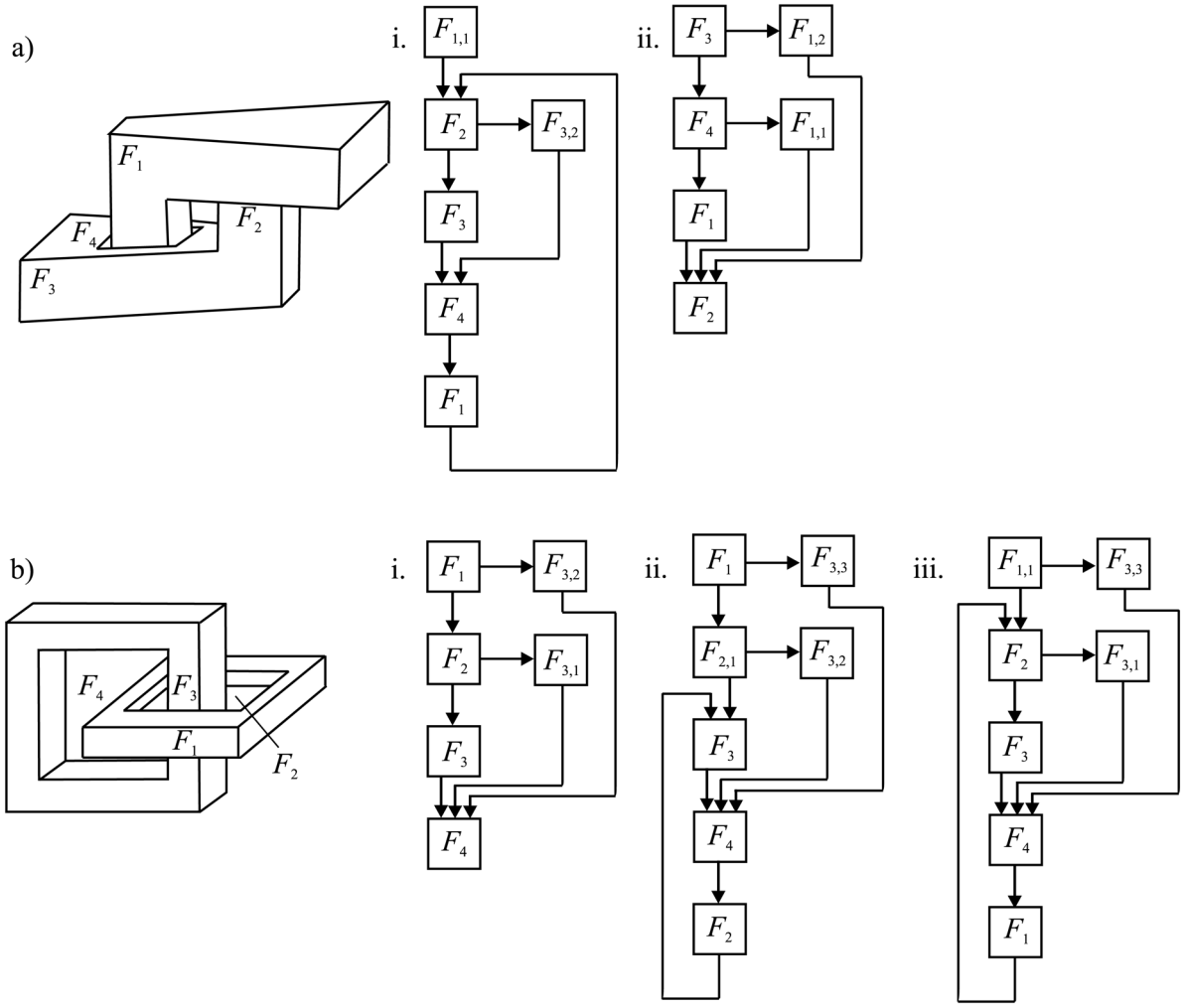

Fig. 7. Two more examples of pairs of components piercing each other.

higher index, and the opposite. Therefore, 24 situations have to be observed. Two of them corresponding to indexing, applied in the figure, are presented. The structure i) corresponds to the case when the solid F_3 is the piercing one, and the structure ii) was obtained when the solid F_1 was piercing the solid F_3 and the pocket F_4.

The example b) is even more complex. We have two solid main shapes, and each of them contains a through–hole. Again, we have 24 different ways of feature indexing, but because of the symmetry, only 12 of them are interesting. But this time, both, the solid feature and its nested through–hole are simultaneously piercing another solid feature and a through–hole, and the opposite. Altogether, we had to test $12 \cdot 2^4 = 192$ situations and we always obtained correct result.

In all hierarchical structures, the original nodes are labeled F_1, F_2, F_3, F_4, and copies are equipped with additional indices regarding the time of their creation (the copy created earlier has lower index). By observing the hierachical structures in Fig. 7 carefully, a conclusion can be made that the nodes can be perturbed several times during the creation of the hierarchical structure. In the case b.iii), the node $F_{1,1}$ which was created as the right copy, even became the root node, and therefore, the left most copy of the node F_1. The main reason for these perturbations is hidden in the situations 2 and 5 from the section 5.4. Because of the perturbations, many pairs of features that were already tested against each other are met for testing again and again. To avoid repeating of tests already done, the adjecency matrix is employed. Each element $a_{i,j}$ stores one of the following values:

0 – the features F_i and F_j have not been tested yet,
1 – the feature F_i contains the feature F_j,
2 – the feature F_j contains the feature F_i,
3 – the feature F_i is piercing the feature F_j,
4 – the feature F_j is piercing the feature F_i,
5 – features F_i and F_j do not contain each other and do not intersect.

Of course, only the pairs of features, where both corresponding matrix elements are set to 0, have to be tested, what accelerates the program considerably.

6. The Time Complexity

The time complexity of the algorithm varies from step to step and does not exceed $O(n^2)$. In the topological part, a new face is created for each ring of multiple connected faces at the beginning. We obtain linear time complexity $O(n)$, where n is the number of rings. In the object decomposition step, each face is tested against all faces situated in the face list before it. We have $n(n-1)/2$ comparisons, and this gives the time complexity $O(n^2)$, where n is the number of faces. Corrections of indices of FIEs are then calculated in $O(n)$, where n is the number of feature candidates. After this, the FIE for each face should be determined. Let us suppose that we have a single feature, and an inconvenient order of its faces causes that each face is attached to different feature candidate. Fortunately, this is not possible, and the time complexity of this step cannot exceed $O(n^2)$. At the end of the object decomposition step, each face, edge and vertex is assigned to the corresponding feature. Each list element is visited once, and we obtain the time complexity $O(n)$. The topological classification is also executed in $O(n)$, where n is the number of faces (the feature's attributes depend on attributes of the feature's faces). The time complexity of the geometrical part depends a lot on the branching of the hierarchical structure. In the worst case, we have to test each feature being inserted against all other features in the structure, and we obtain $n(n-1)/2$ comparisons.

7. Conclusions

In the paper, the algorithm for explicit feature extraction and classification from boundary representation is presented. It operates in two phases: the topological and the geometrical, and classifies extracted features into eight classes. The topological part is not limited to the bodies with planar faces, but in the geometrical part, the current implementation of some tasks requires this limitation. It can be omitted by approximation of curved faces by sets of planar polygons and by performing the containment test and the feature intersection test on these polygons.

The last version of the algorithm is implemented in C++ and is going to be integrated with the geometric constraint solver into an efficient tool for manipulation of mechanical parts. Let us suppose that we have a protrusion placed exactly in the middle of the top face of the block (main shape). If we change the dimensions of the block, it is hard to believe that the protrusion is still placed in the middle. The geometrical data describing the protrusion have to be manually updated to satisfy this requirement. This is just a simple example, but in practise, hundreds or thousands of calculations should be performed to enable such modifications of the geometry preserving additional requirements of the relative positions of the features. This is the ideal task for employing geometric constraints. But remember that first we have to obtain geometric features from the original boundary representation in some way, and this is the function of the algorithm described in the paper.

Beside to generalisation of the algorithm to handle objects with non–planar faces, and integration of the feature–based and constraint–based design, our future work is intended to introduce new explicit feature types containg rings opened on two or more neighbouring faces.

APPENDIX

Let us highlight the object decomposition and topological classification (Section 4) on a simple example described in Fig. 8. Left and right vertical faces are denoted $f_1 - f_4$, front and back faces are $f_5 - f_8$, and bottom and top faces are

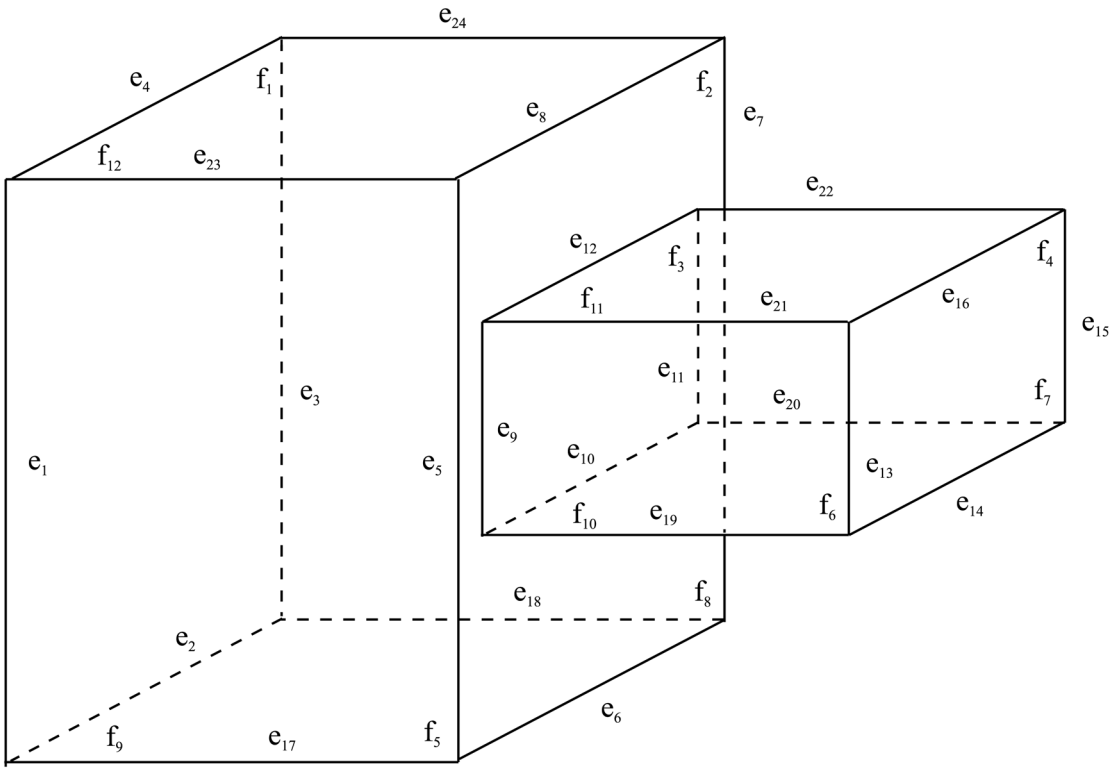

Fig. 8. Example illustrating object decomposition.

$f_9 - f_{12}$. The faces are indexed in order of their appearance in the face list. The initial data are shown in Tab. 1.

face	edge_set	feature_ID
1	1–4	1001
2	5–8	1001
3	9–12	1001
4	13–16	1001
5	1, 5, 17, 23	1001
6	9, 13, 19, 21	1001
7	11, 15, 20, 22	1001
8	3, 7, 18, 24	1001
9	2, 6, 17, 18	1001
10	10, 14, 19, 20	1001
11	12, 16, 21, 22	1001
12	4, 8, 23, 24	1001

Tab. 1. The initial data for the object decomposition example.

While the faces $f_1 - f_4$ do not share any common edges, they are assigned to four different features (Tab. 2 and Tab. 3).

face	edge_set	feature_ID
1	1–4	1
2	5–8	2
3	9–12	3
4	13–16	4

Tab. 2. Changes after testing faces $f_1 - f_4$.

feature	combinable
1	1
2	2
3	3
4	4

Tab. 3. Feature candidates after testing faces $f_1 - f_4$.

The face f_5 shares the edge e_1 with the face f_1, and therefore belongs to the feature F_1 (situation iii). While it also shares common edge e_5 with face f_2, the feature F_2 is identified combinable with the feature F_1 (see Tab. 4 and Tab. 5).

face	edge_set	feature_ID
1	1–5, 17, 23	1
2	1–8, 17, 23	2
5	1–8, 17, 23	1

Tab. 4. Changes after testing face f_5.

feature	combinable
1	1
2	1

Tab. 5. Feature candidates after testing face f_5.

Similarly, face f_6 shares common edge e_9 with the face f_3, and therefore belongs to the feature F_3 (situation iii). It also shares edge e_{13} with face f_4, and the feature F_4 is identified combinable with the feature F_3 (Tab. 6 and Tab. 7).

face	edge_set	feature_ID
3	9–13, 19, 21	3
4	9–16, 19, 21	4
6	9–16, 19, 21	3

Tab. 6. Changes after testing face f_6.

feature	combinable
3	3
4	3

Tab. 7. Feature candidates after testing face f_6.

The face f_7 is combinable with faces f_3, f_4 and f_6 as well, while we rewrite sets of edges with unions of two sets. Similarly, the face f_8 is combinable with faces f_1, f_2 and f_5. The situation in Tab. 8 and Tab. 9 is obtained.

face	edge_set	feature_ID
1	1–5, 7, 17, 18, 23, 24	1
2	1–8, 17, 18, 23, 24	2
3	9–13, 15, 19–22	3
4	9–16, 19–22	4
5	1–8, 17, 18, 23, 24	1
6	9–16, 19–22	3
7	9–16, 19–22	3
8	1–8, 17, 18, 23, 24	1

Tab. 8. Changes after testing faces f_7 and f_8.

feature	combinable
1	1
2	1
3	3
4	3

Tab. 9. Feature candidates after testing faces f_7 and f_8.

After testing all horizontal faces $f_9 - f_{12}$ we finally obtain the situation in Tab. 10 and Tab. 11.

face	edge_set	feature_ID
1	1–8, 17, 18, 23, 24	1
2	1–8, 17, 18, 23, 24	2
3	9–16, 19–22	3
4	9–16, 19–22	4
5	1–8, 17, 18, 23, 24	1
6	9–16, 19–22	3
7	9–16, 19–22	3
8	1–8, 17, 18, 23, 24	1
9	1–8, 17, 18, 23, 24	1
10	9–16, 19–22	3
11	9–16, 19–22	3
12	1–8, 17, 18, 23, 24	1

Tab. 10. Situation after testing all faces f_1–f_{12}.

feature	combinable
1	1
2	1
3	3
4	3

Tab. 11. Feature candidates after testing all faces f_1–f_{12}.

Faces are assigned to four different features, but only two of them are FIEs: the feature F_1 and F_3 have values combinable equal to their indices. The former does not need any corrections, but for the feature F_3, the correction is set to 1 while it is preceded by one feature (F_2) combinable with some other. The number_of_features is set to 2. Now, we find the FIE for each face. Finally, we calculate the correct values of feature_ID for each face, and physically divide lists of faces, edges and vertices.

Faces f_1, f_2, f_5, f_8, f_9, f_{12} and the corresponding edges and vertices are assigned to the feature F_1, and all the rest to the feature F_2. The number_of_rings for the feature F_2 is set to 1 while the face f_3 was created from the ring $e_9 - e_{10} - e_{11} - e_{12}$, and the number_of_parents is set to 1 as well. For the feature F_1, both values are zero. The feature F_1 is therefore recognised as a main shape, and the feature F_2 is a DP–feature.

References

[1] B. FALCIDIENO AND F. GIANNINI, Automatic Recognition and Representation of Shape–Based Features in a Geometric Modeling System, *Computer Vision, Graphics and Image Processing*, No. 48 (1989), pp. 93–123.

[2] J. R. WOODWARK, Some speculations on feature recognition, *Computer–Aided Design*, Vol. 20, No. 4, 1988, pp. 189–196.

[3] J. C. E. FERREIRA AND S. HINDUJA, Convex hull–based feature–recognition method for 2.5D components, *Computer–Aided Design*, Vol. 22, No. 1, 1990, pp. 41–48.

[4] D. SANDIFORD AND S. HINDUJA, Construction of feature volumes using intersection of adjacent surfaces, *Computer–Aided Design*, Vol. 33, 2001, pp. 455–473.

[5] S. MEERAN, J. M. TAIB, A generic approach for recognising isolated, nested and interacting features from 2D drawings, *Computer–Aided Design*, Vol. 31, No. 14, 1999, pp. 891–910.

[6] L. DE FLORIANI, Feature Extraction from Boundary Models of Three–Dimensional Objects, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 8, 1989, pp. 785–798.

[7] L. DE FLORIANI, E. BRUZZONE, Building a feature–based object description from a boundary model, *Computer–Aided Design*, Vol. 21, No. 10, 1989, pp. 602–610.

[8] R. ANANTHA, G. A. KRAMMER, AND R. H. CRAWFORD, Assembly modelling by geometric constraint satisfaction, *Computer–Aided Design*, Vol. 28, No. 9, 1996, pp. 707–722.

[9] B. ŽALIK, N. GUID, An approach to applying constraints in geometric modelling, *Journal of Computing and Information Technology*, Vol. 3, No. 4, 1995, pp. 229–244.

[10] D. PODGORELEC, A new constructive approach to constraint–based geometric design, *Computer–Aided Design*, Vol. 34, No. 11, 2002, pp. 769–785.

[11] K. MARTINI, Hierarchical geometric constraints for building design, *Computer–Aided Design*, Vol. 27, No. 3, 1995, pp. 181–191.

[12] J. K. GUI AND M. MÄNTYLÄ, New concepts for complete product assembly modeling, ACM digital library, *Proceedings of the Second symposium on solid modeling and applications*, 1993, pp. 397–406.

[13] M. E. MORTENSON, *Geometric Modeling*, John Wiley, New York, 1985, 763 pp.

[14] P. GAVANKAR, M. R. HENDERSON, Graph–based extraction of protrusions and depressions from boundary representations, *Computer–Aided Design*, Vol. 22, No. 7, 1990, pp. 442–450.

[15] J. D. FOLEY, A. VAN DAM, S. K. FEINER, J. F. HUGHES, *Computer Graphics — Principles and Practise*, 2nd ed., Addison–Wesley, Reading, 1990, 1174 pp.

Received: June, 2001
Revised: October, 2002
Accepted: December, 2002

Contact address:
David Podgorelec
Faculty of Electrical Engineering & Computer Science
University of Maribor
Smetanova 17, 2000 Maribor, Slovenia.
e-mail: david.podgorelec@uni-mb.si

DAVID PODGORELEC received the BSc degree, MSc degree and PhD in computer science from the University of Maribor, Slovenia, in 1993, 2000 and 2002, respectively. He is a lecture assistant in the department of Computer Science, Faculty of Electrical Engineering & Computer Science (EE&CS), University of Maribor, Slovenia. His research interests include constraint–based and feature–based geometric design, computational geometry, visualisation of medical data, and multimedia applications.

BORUT ŽALIK is currently an associate professor at the Faculty of EE&CS at the University of Maribor, Slovenia. He also has a position of a senior research fellow at the Montfort University, U. K. He received his BSc in electrical engineering in 1985, MSc and PhD in computer science, both from the University of Maribor in 1989 and 1993, respectively. His research interests include computational geometry, geometric modelling, scientific visualisation, GIS applications, multimedia applications.