Reliable Detection and Separation of Components for Solid Objects Defined with Scalar Fields

Oleg Fryazinov, Alexander Pasko
Bournemouth University, UK

Abstract

The detection of the number of disjoint components is a well-known procedure for surface objects. However, this problem has not been solved for solid models defined with scalar fields in the so-called implicit form. In this paper, we present a technique which allows for detection of the number of disjoint components with a predefined tolerance for an object defined with a single scalar function. The core of the technique is a reliable continuation of the spatial enumeration based on the interval methods. We also present several methods for separation of components using set-theoretic operations for further handling these components individually in a solid modelling system dealing with objects defined with scalar fields.

Keywords: component analysis, scalar fields, implicit surfaces, component detection, component separation

1. Introduction

Modern development of CAD/CAM shows the shift from the representation of the objects by its boundary to volumetric representations. This allows users to consider internal structure of the object as well as to define volumetric attributes and properties of the objects in the way closer to the real-life heterogeneous object representation. One of the useful ways to represent the real-life volume objects is using scalar fields. This means that for any point in space a predicate (function) is defined allowing to distinguish points inside the object, outside the object and on the surface of the object. In additional, it gives a measure of some algebraic distance from the given point to the object surface. Such a scalar field is usually considered as a definition of the object geometry in the implicit form.

Scalar fields allow for performing operations on the objects that are very hard to achieve using traditional methods operating with surfaces (Boundary Representation or BRep), such as blending within a certain area or shape metamorphosis with arbitrary changes in topology. On a contrary, some problems that have been already solved for BRep models are yet to be solved for the geometry defined with scalar fields. One of these questions is topological analysis, i.e., detection of the holes, disjoint components and other features for the geometry defined with a scalar field. In this paper, we focus on the analysis of disjoint components, as it is an open issue in modelling with scalar fields. This question is becoming more important with the rapid development of digital fabrication and 3D printing hardware. It is clear that in case of wrongly modelled object the model can break into pieces during the fabrication process and in some extreme cases even break the 3D printing hardware itself.

Traditionally models defined in the implicit form were analysed only if the defining function (scalar field) was simple enough and easy to analyse. In this work, however, we are not restricting the defining function and only assume that the model is bounded and we know the box which encloses the point set belonging to the interior and the surface of the object. In practice, where we are taking into account practical modelling systems that deal with the implicit form, such as BlobTree [1] or HyperFun [2], it can be seen that the defining function can be very complex and far from polynomial.

To date, an analytical solution for topological analysis for the general case has not been found. However, a numerical solution can be found, but it should be reliable, meaning the result should be exact within the given precision. In this paper, we present a technique allowing for detecting the number of disjoint components in the model represented in the implicit form and methods to separate the detected disjoint components. The detection and separation operations result in a continuous and smooth scalar field for each component. These operations are designed such that they can be directly used in a modelling system dealing with the objects represented in the implicit form.

The main contributions of this paper are the following:

1. For a solid model defined with a scalar field, we propose a technique for identifying the number of disjoint components with the pre-defined tolerance without setting any severe restrictions to the tolerance.
2. Various methods for the separation of components are presented.
3. An adaptive spatial continuation is presented for the fast and yet efficient reliable enumeration.

2. Related work

The problem of detection of disjoint components was usually considered in the scope of general topological analysis or as an applied problem. Thus, for the purposes of collision detection, a two dimensional point set was analysed in [3]. For polygonal meshes the analysis of disjoint components was mostly the analysis of the number of shells (structure of connected triangles) within the polygon soup. The question of the number of disjoint components in the polygonal manifold mesh was discussed in [4], where the *corner table* data structure was used to separate disjoint shells.

The problem of detection of the number of disjoint components is related to the null-object detection, for example, when intersecting two objects for collision detection. The collision detection algorithms for BRep typically inspect the boundary components of the intersection to confirm that the boundary is empty [5]. The null object detection was addressed in the Constructive Solid Geometry (CSG) with the primitive redundancy principle allowing for reduction of the CSG-tree to the null tree in the case of the null-object by removing redundant primitives and simplifying the tree [6]. In the case of voxelized objects, the collision detection is addressed by adding a reference to each object to the voxels touched by the object bounding box [7] followed by checking the case when several objects share the same non-empty voxel.

One of the ways to analyse the topology of the model defined implicitly is applying the Morse theory [8]. Thus, in [9] it was used to analyse the topology of the implicit surfaces for the polygonization purposes. However, the Morse theory is hardly applicable for general purposes because of the requirements of C^2-continuity of the defining function and the necessity to derive and to analyse the expressions for the first derivative. Another exact analysis of the topology and geometry of the models defined in the implicit form was done in [10], where the class of models was limited to those defined by the primitive polynomials. Also the analysis of the discrete scalar field by using Reeb graphs along with the Morse theory was presented in [11] Other works on topological analysis of isosurfaces of scalar fields are discussed in [12].

The problem of separation of different components in the object was explicitly set and analysed for two-dimensional geometry defined with BRep in [13] as well as for separating set of connected points by finding the suitable bounding volume for each component in the set [14].

3. Background

3.1. Affine Arithmetic and Revised Affine Arithmetic

Affine Arithmetic was introduced in the early 1990s as an extension to the Interval Arithmetic and is a technique allowing to perform computations over uncertain values[15]. Comparing with classic Interval Arithmetic, Affine Arithmetic is generally considered as the technique that provides tighter bounds for computer quantities.

Uncertain values in Affine Arithmetic are represented by the affine forms, i.e., polynomials as follows:

$$\hat{x} = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + ... + x_n\varepsilon_n \tag{1}$$

where x_i are known real coefficients and ε_i are noise symbols, i.e., symbolic variables with the values assumed to lie in the interval $\varepsilon_i \in [-1, 1]$.

In Affine Arithmetic, the formula evaluation is performed by replacing operations on real quantities by their affine forms. Similarly to Interval Arithmetic, the inclusion property is applied to Affine Arithmetic, i.e., for any operation $\otimes$:

$$A \otimes B \supset \{a \otimes b, a \in A, b \in B\}$$

where a and b are real values and A and B are uncertain values in the affine form.

All the operations on the affine forms can be divided into affine (exact) and non-affine (approximate) operations. An affine operation is a function that can be represented by the linear combination of the noise symbols of its arguments. Non-affine operations can not be performed over the linear combination of the noise symbols. In this case, an approximate affine function is used and a new noise symbol is added to the affine form to represent the difference between the non-affine function and its approximation. Additional details regarding the construction of both affine and non-affine operations can be found in the literature related to Affine Arithmetic [15] [16].

One of the obvious drawbacks of the Affine Arithmetic is that each non-affine operation introduces one extra noise symbol and therefore for the functions that contain large number of non-linear operators including multiplication the amount of the data for the affine forms can be unbearable. In [17] it was shown that Revised Affine Arithmetic is more suitable for the interval computations for large models defined in the implicit form, as it keeps the number of noise symbols constant still providing tight bounds for the interval of the function.

The revised affine form for the purposes of space partitioning is presented as the following

$$\hat{x} = x_0 + \sum_{i=1}^{3} x_i\varepsilon_i + e_x[-1, 1], e_x \geq 0 \tag{2}$$

Here we have three independent uncertain values, one for each coordinate.

3.2. Affine Arithmetic-driven space partitioning

The inclusion property of the Interval Arithmetic as well as its successors, including Affine Arithmetic, allows for partitioning the space into cells that are inside, outside and potentially intersect the surface of the object. The main idea behind the subdivision is to test each cell for

inclusion of the zero set of the function, to reject those cells that do not contain zero-value points and subdivide the cell into eight otherwise.

The inclusion property for Affine Arithmetic means that, if the affine form for the expression does not contain the zero value, then the actual function range for the given input interval does not contain the zero value as well. In terms of Affine Arithmetic formally it can be represented as follows. For the input box $\square_i$ with the corners $(x_{min}, y_{min}, z_{min})$ and $(x_{max}, y_{max}, z_{max})$, the affine form for the coordinate values can be represented as:

$$\hat{x} = \frac{x_{min} + x_{max}}{2} + \frac{x_{max} - x_{min}}{2}\varepsilon_1$$

$$\hat{y} = \frac{y_{min} + y_{max}}{2} + \frac{y_{max} - y_{min}}{2}\varepsilon_2$$

$$\hat{z} = \frac{z_{min} + z_{max}}{2} + \frac{z_{max} - z_{min}}{2}\varepsilon_3$$

Applying Revised Affine Arithmetic by replacing the functions over the real variables with their affine forms, we get

$$\hat{f} = \hat{f}(\hat{x}, \hat{y}, \hat{z}) = f_0 + f_x\varepsilon_1 + f_y\varepsilon_2 + f_z\varepsilon_3 + e[-1, 1]$$

Here the real coefficients f_0 ... f_3 denote the values obtained after calculation of the affine form of the given function over the affine forms of its arguments (see equation 1).

The function does not contain zero if either the lower bound of the resulting affine form is positive and hence the whole range is positive:

$$f_0 - f_x - f_y - f_z - e > 0$$

or the upper bound is negative and the whole range is negative:

$$f_0 + f_x + f_y + f_z + e < 0$$

The adaptive space partitioning based on the affine arithmetic is discussed in detail in [18] and the algorithm of the space partitioning based on the revised affine arithmetic is shown in [17].

3.3. Enumeration methods

By enumeration we understand the subdivision of the space Ω into a number of small cells and identifying the cells which contain the surface of the model inside. The size of the cell depends on the accuracy of the enumeration and usually is set by the user.

Traditional methods for enumeration are based on the scalar field sampling on a regular grid defined for the whole space Ω where the domain is given by an axis-aligned bounding box [19]. It is obvious that smaller size of the cells leads to higher computational costs, as the value of the defining function should be calculated in a large number of points. Adaptive techniques can be applied for enumeration, however they either are not robust and can miss small features of the model, for example, the one presented in [19] or rely upon pre-defined information obtained for the scalar field, such as [20]. One way to guarantee robustness of the spatial subdivision is the adaptive enumeration based on the Interval Arithmetic [21] and its successors [18, 17]. The robustness of such subdivisions is guaranteed by the inclusion property.

Continuation methods allow to find zero-level set of the scalar field near other already found solutions. For computer graphics applications dealing with scalar fields mostly simplicial continuation methods are used. The main idea of such methods is to construct the simplicial complex that approximates the zero level set of the given scalar field. The process consists of moving from one simplex to the adjacent one through a common facet. Different methods for generating the surface approximation by using continuation are discussed in [22]. Generally it can be seen that we need to find the initial zeros of the surface in order to generate an approximation of a connected piece of the approximated surface. For each next component new initial zero should be found. We extend this idea to the octree construction in our method presented below.

4. Method overview

4.1. Adaptive spatial continuation

In our method of enumeration, we combine the Revised Affine Arithmetic-driven space partitioning with continuation. Note that methods that combine exhaustive spatial enumeration with continuation are known, some were discussed in [22]. However only non-interval methods were used to date.

We consider the object defined in the implicit form as

$$S = (x, y, z) \in \Omega \subseteq \Re^3 : f(x, y, z) = 0$$

with its affine form $\hat{f}(\hat{x}, \hat{y}, \hat{z})$. In our work we use the following classes of objects:

- The objects where the symbolic representation of the defining function is known. This class is rather large and includes a lot of geometric primitives including sphere, box, skeletal convolution surfaces, as well as the large set of transformations of the primitives including affine transformations, deformations such as twisting and bending, and other operations including set-theoretic operations, metamorphosis and blending. The interval extension can be constructed using Interval Arithmetic and its successors. We found that for complex symbolic objects the most efficient technique is to use Revised Affine Arithmetic, which was recently shown as a fast and reliable interval extension [17].

- The objects where the field values can be bounded within the given domain. One of the examples of the objects within this class are the objects represented

as a signed distance field where the field value is the Euclidean distance to the surface. The interval extension can be found by sampling the value in the centre of the domain and extending the interval with the extent of the domain as presented in [23]

In practice, we are working with large procedural trees that contain the objects of different types. For simplicity, we use the term "affine form" to refer to the extension of the object using Revised Affine Arithmetic with objects converted from Interval Arithmetic where it is needed.

For each cell we can calculate two types of the range of the defining function:

1. The "affine" range resulting from using the affine form of the defining function for the given bounding box;
2. The "control" range resulting from calculating the value of the defining function in the corners of the bounding box. We suppose that the calculation of the defining function at a random point in space is computationally less expensive that the calculation of the affine form for a random bounding box.

For the affine range, the lower and upper bounds are found by using the affine form as follows. For the cell i with the bounding box $\square_i$, the affine form $\hat{f}(\hat{x}, \hat{y}, \hat{z})$ is calculated. The interval $[\hat{f}]$ is corresponding to the interval extension of the given affine form. Note that the inclusion property means that if $[\hat{f}] > 0$ or $[\hat{f}] < 0$, the function values inside the box are either all positive or all negative and therefore the cell does not intersect the surface. On the other hand, if $0 \in [\hat{f}]$ the range of the function inside the box might not include the zero value because of the overestimation.

For the control range, we calculate the interval $[\underline{f_c}, \overline{f_c}]$, where

$$\begin{aligned} \underline{f_c} &= \min_{corner=1}^{6} f_{corner}(x, y, z) \\ \overline{f_c} &= \max_{corner=1}^{6} f_{corner}(x, y, z) \end{aligned} \tag{3}$$

Here f_{corner} denotes the value of the defining function in a corner of the current cell.

While the affine range cannot guarantee that the zero-set intersects the cell in the case of $0 \in [\hat{x}]$, the control range can provide information on whether the zero-set intersects the cell or not. Indeed, if the actual range of the function is $[f]$ then $[\underline{f_c}, \overline{f_c}] \in [f]$ and $0 \in [\underline{f_c}, \overline{f_c}] \implies 0 \in [f]$.

If the control range of the cell contains zero, then we switch from adaptive enumeration to continuation and find the cells that are adjacent to the edge of the current cell with the sign change as shown in Algorithm 1 outlining the entire enumeration process. Note that continuation allows us to find adjacent cells, but it is not guaranteed that all the cells that intersect the zero-set are found. Therefore after continuation takes place, further analysis of the cells larger than the predefined tolerance whose affine range contains zero is done.

Algorithm 1 Adaptive spatial continuation

Procedure: enumerate($\hat{x}$, $\hat{y}$, $\hat{z}$)
  Calculate the value of the defining function in the corners of the cell: $f(\underline{\hat{x}}, \underline{\hat{y}}, \underline{\hat{z}})$, $f(\underline{\hat{x}}, \underline{\hat{y}}, \overline{\hat{z}})$ ... $f(\overline{\hat{x}}, \overline{\hat{y}}, \overline{\hat{z}})$
  Calculate $\underline{f_c}$ and $\overline{f_c}$ as a minimum and maximum value of all values in the corners
  if the range of the function includes a 0 value **then**
    Find the edge(s) $e = (\mathbf{v}_1, \mathbf{v}_2)$ for which $sign(f(\mathbf{v}_1) \neq sign(f(\mathbf{v}_2)$
    Find the non-enumerated neighbour cells
    if The non-enumerated neighbour cell j exists **then**
      Find bounds for the cell j: $\hat{x}_j$, $\hat{y}_j$, $\hat{z}_j$
      Propagate the information about function inclusion up the octree for the cell j
      enumerate($\hat{x}_j$, $\hat{y}_j$, $\hat{z}_j$)
    end if
  end if
  if the size of the cell is smaller than some predefined threshold **then**
    return (no intersection with zero-level set at given resolution)
  end if
  if the range of the function does not include a 0 value **then**
    Calculate the affine form with the arguments [$\hat{x}$, $\hat{y}$, $\hat{z}$]
    Get the range of the function from the affine form
    if the range of the function does not include a 0 value **then**
      return (no intersection with zero-level set);
    end if
  end if
  Subdivide the cell into 8 sub-cells
  For sub-cell i with the range($\hat{x}_i$, $\hat{y}_i$, $\hat{z}_i$), enumerate($\hat{x}_i$, $\hat{y}_i$, $\hat{z}_i$)
  return

Despite being reliable, all interval-based methods suffer from overestimation. To deal with this, only the control range is used to define if the cell intersects the zero-level set or not at the lowest level, where the size of the cell is less or equal to the predefined tolerance.

4.2. Analysis of components by using enumeration

As it can be seen, after the enumeration we can isolate three types of cells: cells that are inside the object, cells that contain the zero-level set and cells outside the object. For component analysis we assume that with the given resolution, the number of disjoint components for an object defined in the implicit form coincides with a number of disjoint components in the enumeration where the cells that lie outside the surface are not taken into an account. The enumeration properties supporting this statement are the following.

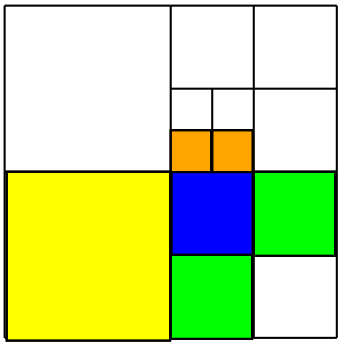

Figure 1: Example of neighbour cells for propagation in the case of a quadtree: initial cell is denoted as blue, same level neighbour cells are denoted as green, level-up neighbour cell is denoted as yellow and level-down neighbour cells are denoted as orange.

First, the number of disjoint components is not less than the number of components in the enumeration. As the enumeration is based on the interval methods and encloses the zero-level set, it includes each disjoint component. Secondly, the number of disjoint components in the model is not greater than the number of components in the enumeration. As the last level in the enumeration contains only the cells which intersect the zero-level set of the defining function, not its interval extension, two components can be in the same cells or in neighbouring cells only if the distance between these components is less than twice the size of the cell. The size of the cell is half of the resolution, therefore for the given resolution, no space between components is missed. As the number is not less and not greater, it is equal.

To detect the number of components in the enumeration we use the marking with propagation technique. For propagation we build the adjacency relation between cells by using the following simple rules:

- For the cells with the same depth within the octree, we mark as adjacent the cells that have common sides with the given one (denoted by green in Figure 1)
- If neighbour cells at the same depth have child cells, we mark as adjacent these cells which share the common side with the given one (denoted by orange in Figure 1)
- If neighbour cells have lower depth level, we mark the level-up cell (denoted by yellow in Figure 1)

The number of components for the object is the number of non-connected clusters in the octree. The number of clusters can be easily determined by propagation as follows:

1. While there are non-marked cells, increase the number of the components by 1 and mark the first non-marked cell with the current components number
2. Recursively mark all the adjacent cells for the current one

At the end of this procedure, we have the number of components and the octree where each subcluster is marked by the number which represents the index of the component this cluster belongs to. The resulting octree is used for component separation which we discuss below.

4.3. Component separation

At the analysis step, we obtain the cells that lie inside the implicitly defined object or intersect the zero level set. A cluster of cells corresponding to a specific component can be directly used for this component separation, for example, by using set-theoretic intersection of the cell cluster with the original object. Thus, we introduce separation approaches where the clusters of internal and border cells in the octree are used in the function definition for separated components.

Note that for the binary point membership classification purposes (CSG-type separation) we can use the cells of the octree directly, as in this case we can identify if the point is inside of any cell in the set or outside and therefore define a membership for the particular component. However in the case the components are going to be used for further operations in the modelling system dealing with scalar fields, not only the point membership classification, but actual evaluation of the component defining function at any point in space is needed. Therefore, we first need to introduce a scalar field representing the octree for the purposes of the component separation.

Various methods can be used to represent a subcluster of the octree cells as a single volume by a scalar field. We can roughly distinguish these methods into two categories: in the first category we consider an octree subcluster as a volume enclosed by a surface and we construct a scalar field based on this surface, in the second category we treat the subcluster as a whole volume with the corresponding volume function. Below we discuss these methods in more detail.

4.3.1. Surface approaches

The distance field to the octree can be found directly similar to the method presented in [24]. To query the value of the scalar field, we recursively find the closest feature in the octree which lies on the boundary. Thus, we find distance to the closest border cell of the octree, i.e. the leaf cell that does not have neighbours on some of the sides. Then for the closest node we find the distance to the closest feature: the vertex, edge or face of the node providing that the feature itself does not belong to the inner cells of the octree. In case of several closest nodes, the one which results in smaller distance to the closest features is taken into account. Note that inner features of the cells are not involved in the result and effectively this method can be seen as calculating the distance to the surface of the octree. In some cases it is more efficient to

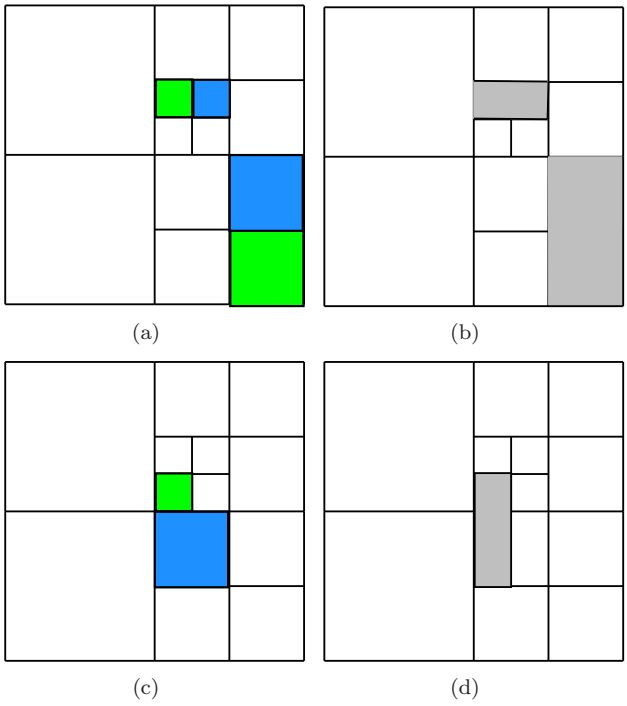

Figure 2: Cell operations example

extract the surface from the octree as a quad mesh and use this mesh for signed distance queries. Multiple methods for the distance evaluation to the mesh exist, for example, as presented in [25]. The sign of the distance field can be calculated with the point membership classification for the octree. Signed distance fields are efficient and fast, but they result in C^1 discontinuities that can be unacceptable for further operations on the results of the components separation.

Another method that allows to avoid C^1-discontinuities, but significantly slower is to extract the surface from the border nodes of the octree and create a BSP-field [26]. This field is constructed based on the equations for the half-spaces of the faces and set-theoretic operations over these half-spaces. In the case of octrees we can benefit from the fact that all the faces in the surface of the octree are axis-aligned and therefore the halfspace equations will look either like $f_{BSP_x}(x,y,z) = a_x x + b_x$, $f_{BSP_y}(x,y,z) = a_y y + b_y$ or $f_{BSP_z}(x,y,z) = a_z z + b_z$ and in the of subdividing the face by a half-space the result is also a quad with axis-aligned edges.

4.3.2. Set-theoretic approach

The obvious way to represent the octree as a single volume with a scalar field is to represent each cell in the octree by a function and then union all these cells together. As we use axis-aligned bounding box as the shape for the cell in the octree, the function for the cell whose corner coordinates are $(x_{min}, y_{min}, z_{min})$ and $(x_{max}, y_{max}, z_{max})$ looks pretty easy:

$$\begin{aligned} f_{cell}(x,y,z) = &((x - x_{min}) \cdot (x_{max} - x)) \wedge \\ &((y - y_{min}) \cdot (y_{max} - y)) \wedge \\ &((z - z_{min}) \cdot (z_{max} - z)) \end{aligned} \tag{4}$$

The function for the octree cluster consisting of n cells can be represented as

$$f_{octree} = \vee_{i=1}^{n} f_{cell_i} \tag{5}$$

Here by $\wedge$ and $\vee$ we denote functions implementing set-theoretic intersection and union respectively.

Traditionally set-theoretic operations on the objects represented with scalar fields are done with R-functions which are non-regularised. Therefore the union of all cells in the octree cluster will result in internal zeroes that will appear in the result of the intersection of the original object with the cluster. To avoid this we propose to make modifications to the cells:

- Replace two adjacent cells lying on the same level of the octree by union of these cells (see Figure 2 a, b)
- In the case the adjacent cell lies on the level above, replace the cell by the extended cell to the opposite side of the adjacent cell (see Figure 2 c, d)
- In the case the adjacent cell lies on the level below, leave the cell untouched
- If we modify the cell more than one time because of various adjacent cells, use set-theoretic union for all the modified cells

4.4. Adaptive component analysis and separation

As we discussed above, the enumeration we use for analysis of the components is later used for separation. However this re-use affects the performance of the modelling system where the operation of the components separation is introduced. Obviously, with discrete enumeration the smaller tolerance, the more precisely we can get the result. On the other hand, for a very small tolerance, the enumeration can be very expensive from the octree storage point of view and also can lead to the computationally expensive function evaluation of the octree for set-theoretic intersection we use for the separation. Instead of keeping the octree with the highest precision, the number of components we found with it can be used to decrease the size of the enumeration octree and to make its function evaluation less computationally expensive.

The algorithm of the analysis and separation becomes as shown on the Algorithm 2.

It is clear that building the enumeration octree becomes more time-consuming in this case. However, it can be done at preprocessing stage only once.

	Octree depth	Enumeration, our method	Component detection	Enumeration, [17]
Simple object (Fig. 3)	6	1.043	0.28	1.751
Metamorphosis (Fig. 5)	7	7.427	0.683	26.861
Collision detection (Fig. 6de)	8	1.542	0.179	15.845

Table 1: Timings for adaptive spatial enumeration and component detection. The depth of the octree estimated as the minimal resulting in the correct number of the components.

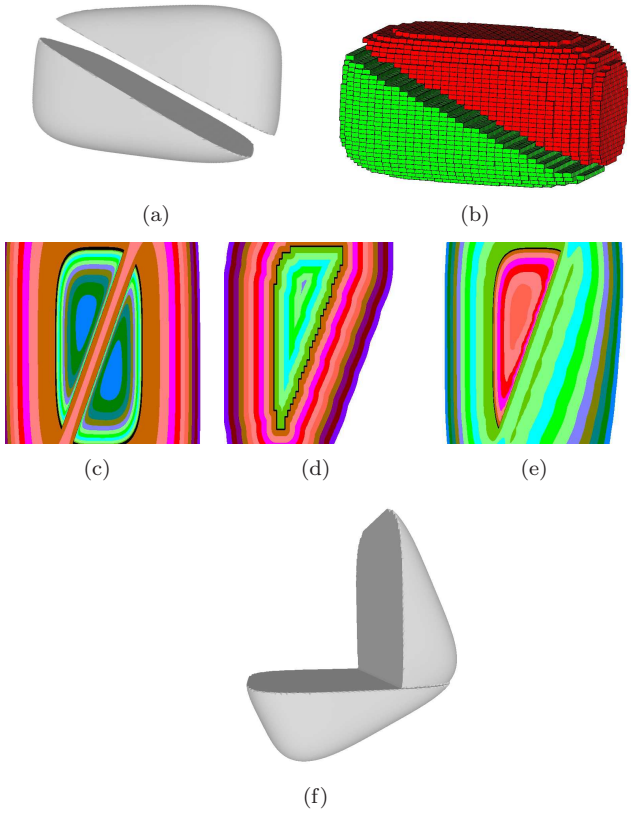

Figure 3: Analysis and separation of components in a simple object defined in the implicit form: a) Initial object (a thin box subtracted from a superellipsoid), b) The octree of the object with two components detected, c) The field for an initial object, d) The signed distance field for the first cluster of the octree, e) The first component of the object separated by set-theoretic intersection of the initial object with the signed distance field for the first cluster of the octree, f) Further operations on the components: the second component is transformed and the set-theoretic union is applied.

Algorithm 2 Adaptive component analysis and separation

Procedure: AdaptiveEnumeration($level_{max}$)
Build the octree with the precision depending on the $level_{max}$
Calculate the number of components N for the given octree
$n = N$, $level_{current} = level_{max}$
while $N = n$ **do**
Save the current octree, if $level_{current} \neq level_{max}$ replace the last saved
Remove the cells of the level $level_{current}$
Calculate the number of components n for the given octree
$level_{current} = level_{current} - 1$
end while
Use the saved octree for the separation algorithm
return

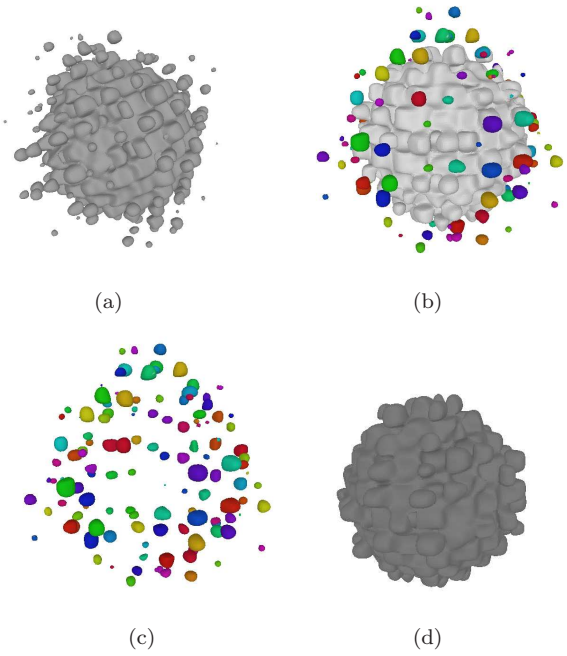

Figure 4: Solid noise example: a) Initial object, b) Components detected, c) The component with the largest volume is removed by using set-theoretic subtraction, d) The component with the largest volume is isolated by using set-theoretic intersection.

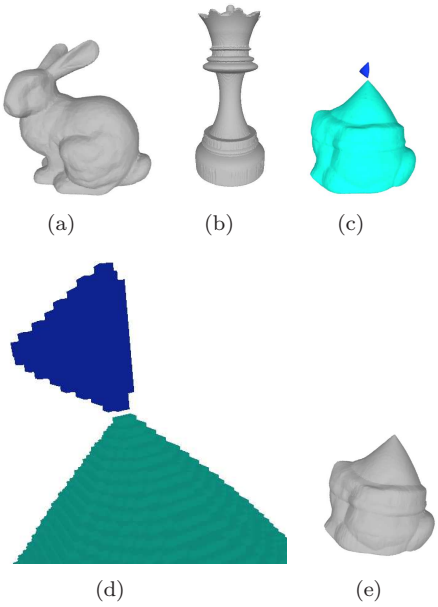

Figure 5: Metamorphosis example: a) Source model b) Target model, c) Intermediate shape after metamorphosis operation with components detected, d) Zoom in of the segmented octree, e) Separated largest component as the resulting model.

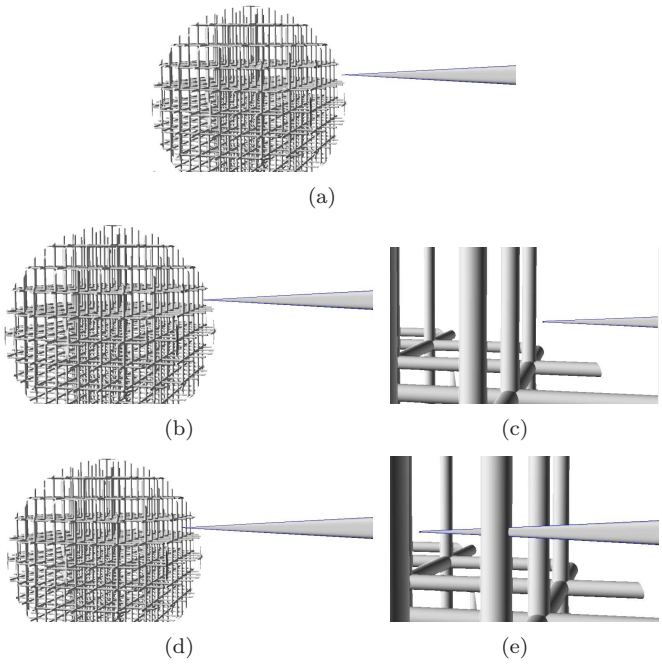

Figure 6: Collision detection between a needle and a microstructure: a) The needle outside the bounding sphere, no collision, b) The needle inside the bounding sphere, but no collision, c) Zoom-in of case b, d) Collision detected, e) Zoom-in of case d

5. Applications and results

Our method of component analysis and separation can be used in a modelling system dealing with models defined with scalar fields. It can be applied to models with different complexity as well as with different procedures for the scalar field evaluation. Consider a model created with a set-theoretic operation on a superellipsoid and a thin block represented with a constructive tree as and example of analysis procedurally defined CSG models (see Figure 3), and also a more complex shape defined with procedural solid noise (see Figure 4).

The table 5 shows timings for some models we use in this work. The tests were performed on a laptop with i3 2.27Ghz processor and 4GB RAM. Neither hardware acceleration nor multithreading were used, however the results show that the analysis can be done in few seconds. Comparing with the existing methods for building a reliable enumeration of the scalar fields, such as [17], it can be seen that adaptive spatial continuation is much faster. Note that the detection time depends on the octree and therefore it is equivalent for different enumeration methods as soon as they produce equivalent octrees.

One of the applications of the method is separation of the biggest component for operations where disjoint components typically appear. For example, consider a metamorphosis operation. This operation is very easy to implement for the models defined with scalar fields, as in the simplest case, the function for the intermediate shape can be defined by a linear interpolation between defining functions for the source and the target model. However the result can contain disjoint components as the method does not preserve neither the topology nor the number of components. From the user's point of view, it can be suggested that the number of components should be kept constant, so our technique can be used. At the intermediate stages, the geometry is analysed, the enumeration is built and the number of components is counted. If the number of components us greater than the number of components in the source and the target models, then the size of the components is analysed by calculating the volume of the corresponding enumeration cluster, and the components with smaller size are subtracted from the model. The results are presented in Figure 5.

Another interesting application of our technique is the long-standing problem of collision detection for models defined in the implicit form and the related problem of the null-set detection. It can be seen that if set-theoretic intersection of two models results in the empty set, then two models do not collide. This also means that the number of components for the intersection is zero in the case two objects do not collide. In the Figure 6 we show an example of collision detection using our technique.

It should be noted that in most of our examples the quality of the field was not really the issue, as the component analysis and separation were the latest stage in the workflow. Because of that we mostly use surface approaches (i.e., use the signed distance field to the octree) rather than the set-theoretic approach because of the complexity and the longer evaluation of the latter.

6. Conclusion and discussion

Modelling with scalar fields is the area where many problems that are solved for other representation are yet to be solved. The topological analysis is one of these problems, which was tackled for some object subclasses, but not for the general case. In this paper, we presented a numerical solution which allows to detect the number of the disjoint components in the model defined in the implicit form. The data structure that is used for the detection is used for separation by using scalar field created from the enumeration and further set-theoretic operations.

The presented technique allows to analyse the large number of the classes of the models defined in the implicit form, however it still suffers from being numerical. This means that there is a connection between efficiency and accuracy, because the more exact we want the result to be, the deeper the enumeration should be built. This in its turn means that the function value and the interval extension should be calculated at a very large number of points which can take a while even for a relatively simple defining function.

One interesting application of the presented technique can be a separation of the components when we explicitly know the number of components and want only to separate them. While we discussed one of the probable ways to handle this situation, more research can be done to find the enumeration which is small enough to be efficiently evaluated and bound the components well.

The component analysis is one of the problems in the scope of the general topological analysis including the evaluation of the genus and the number of disjoint components, critical points detection [12] and the Reeb graph construction [11]. While our method supports detection of the number of disjoint components, it is not directly applicable to other topological analysis problems. The extension of our technique to the more complete analysis of object topology is one the directions for future research, for example, extending the method to the computation of homology (Betti number from 0 to 2 for 3D objects defined with scalar fields).

Appendix A. An example of calculating the ranges of the cell for the given model

The model presented in Figure 4 has the following defining function:

$$\begin{aligned} f(x,y,z) =& 81 - 100x^2 - 100y^2 - 100z^2 + \\ & (3.8\sin(15x) + 1.6\sin(11.1x + 1.1\sin(15x))) \cdot \\ & (3.8\sin(15y) + 1.3\sin(11.1x + 1.1\sin(15x))) \cdot \\ & (3.8\sin(15z) + 2.6\sin(11.1x + 1.1\sin(15x))) \end{aligned} \tag{A.1}$$

Consider the cell $\square_i$ with coordinates x_{min} = (-0.26, -7.04, 5.28), x_{max} = (1.62, -5.8, 6.38). The function value in the corners takes a value of 0.3388, 0.3346, 0.6187, 0.6036, 0.3482, 0.3146, 0.6112 and 0.5058. The control range is $\underline{f_c} = 0.3146$, $\overline{f_c} = 0.6187$. As it can be seen, the control range does not include zero. On the next step of the algorithm we calculate the affine range. The affine forms are calculated as following: $\hat{x} = 0.68 + 0.94\varepsilon_1$, $\hat{y} = -6.42 + 0.62\varepsilon_2$, $\hat{z} = 5.83 + 0.55\varepsilon_3$. The value of the affine function is $\hat{f} = 0.2484 - 0.0034\varepsilon_1 - 0.0042\varepsilon_3 + 0.3841e_x$, the affine range is $[\hat{f}] = [-0.1434, 0.6402]$. The affine range shows that the zero level set potentially crosses the cell, so further subdivision is done.

References

[1] B. Wyvill, A. Guy, E. Galin, Extending the CSG Tree. Warping, blending and Boolean operations in an implicit surface modeling system, Computer Graphics Forum 18 (2) (1999) 149–158.

[2] V. Adzhiev, R. Cartwright, E. Fausett, A. Ossipov, A. Pasko, V. Savchenko, HyperFun project: a framework for collaborative multidimensional F-Rep modelling, in: Proc Implicit Surfaces '99, Eurographics/ACM SIGGRAPH Workshop, J. Hughes and C. Schlick (Eds.), 1999, pp. 59–69.

[3] D. M. Mount, Intersection detection and separators for simple polygons, in: Proceedings of the eighth annual symposium on Computational geometry, SCG '92, ACM, New York, NY, USA, 1992, pp. 303–311.

[4] J. R. Rossignac, Solid and Physical Modeling, John Wiley & Sons, Inc., 2007.

[5] J. W. Boyse, Interference detection among solids and surfaces, Commun. ACM 22 (1) (1979) 3–9.

[6] R. B. Tilove, A null-object detection algorithm for constructive solid geometry, Commun. ACM 27 (7) (1984) 684–694.

[7] S. Bandi, D. Thalmann, An adaptive spatial subdivision of the object space for fast collision detection of animated rigid bodies, Computer Graphics Forum 14 (3) (1995) 259–270.

[8] J. C. Hart, Morse theory for implicit surface modeling, in: H.-C. Hege, K. Polthier (Eds.), Mathematical Visualization, Springer Berlin Heidelberg, 1998, pp. 257–268.

[9] B. T. Stander, J. C. Hart, Guaranteeing the topology of an implicit surface polygonization for interactive modeling, in: Proceedings of the 24th annual conference on Computer graphics and interactive techniques, SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1997, pp. 279–286.

[10] E. Berberich, M. Kerber, M. Sagraloff, Exact geometric-topological analysis of algebraic surfaces, in: Proceedings of the twenty-fourth annual symposium on Computational geometry, SCG '08, ACM, New York, NY, USA, 2008, pp. 164–173.

[11] W. Harvey, O. Rbel, V. Pascucci, P.-T. Bremer, Y. Wang, Enhanced topology-sensitive clustering by Reeb graph shattering, in: R. Peikert, H. Hauser, H. Carr, R. Fuchs (Eds.), Topological Methods in Data Analysis and Visualization II, Mathematics and Visualization, Springer Berlin Heidelberg, 2012, pp. 77–90.

[12] T. Nieda, A. Pasko, T. L. Kunii, Detection and classification of topological evolution for linear metamorphosis, The Visual Computer 22 (5) (2006) 346–356.

[13] J. S. Mitchell, S. Suri, Separation and approximation of polyhedral objects, Computational Geometry 5 (2) (1995) 95 – 114.

[14] J.-D. Boissonnat, J. Czyzowicz, O. Devillers, J. Urrutia, M. Yvinec, Computing largest circles separating two sets of segments, in: Proceedings of the 8th Canadian Conference on Computational Geometry, Carleton University Press, 1996, pp. 173–178.

[15] L. H. de Figueiredo, J. Stolfi, Self-Validated Numerical Methods and Applications, Brazilian Mathematics Colloquium monographs, IMPA/CNPq, Rio de Janeiro, Brazil, 1997.

[16] L. H. de Figueiredo, J. Stolfi, Affine arithmetic: Concepts and applications, Numerical Algorithms 37 (2004) 147–158.

[17] O. Fryazinov, A. Pasko, P. Comninos, Fast reliable interrogation of procedurally defined implicit surfaces using extended revised affine arithmetic, Comput. Graph. 34 (6) (2010) 708–718.
[18] L. H. D. Figueiredo, J. Stolfi, Adaptive enumeration of implicit surfaces with affine arithmetic, Computer Graphics Forum 15 (1996) 287–296.
[19] J. Bloomenthal et al. (Ed.), Introduction to Implicit Surfaces, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.
[20] D. Kalra, A. H. Barr, Guaranteed ray intersections with implicit surfaces, SIGGRAPH Comput. Graph. 23 (3) (1989) 297–306.
[21] J. M. Snyder, Interval analysis for computer graphics, SIGGRAPH Comput. Graph. 26 (2) (1992) 121–130.
[22] A. Gomes, I. Voiculescu, J. Jorge, B. Wyvill, C. Galbraith, Implicit Curves and Surfaces: Mathematics, Data Structures and Algorithms, 1st Edition, Springer Publishing Company, Incorporated, 2009.
[23] D. Storti, C. Finley, M. Ganter, Interval extensions of signed distance functions: iSDF-reps and reliable membership classification, Journal of Computing and Information Science in Engineering 10 (2) (2010) 1–8.
[24] S. F. Frisken, R. N. Perry, A. P. Rockwood, T. R. Jones, Adaptively sampled distance fields: a general representation of shape for computer graphics, in: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000, pp. 249–254.
[25] C. Ericson, Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3D Technology), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
[26] O. Fryazinov, A. A. Pasko, V. Adzhiev, BSP-fields: An exact representation of polygonal objects by differentiable scalar fields based on binary space partitioning, Computer-Aided Design 43 (3) (2011) 265–277.