KYBERNETIKA — VOLUME 55 (2019), NUMBER 5, PAGES 755–781

CLOSE-TO-OPTIMAL ALGORITHM FOR RECTANGULAR DECOMPOSITION OF 3D SHAPES

Cyril Höschl IV and Jan Flusser

In this paper, we propose a novel algorithm for a decomposition of 3D binary shapes to rectangular blocks. The aim is to minimize the number of blocks. Theoretically optimal brute-force algorithm is known to be NP-hard and practically infeasible. We introduce its sub-optimal polynomial heuristic approximation, which transforms the decomposition problem onto a graph-theoretical problem. We compare its performance with the state of the art Octree and Delta methods. We show by extensive experiments that the proposed method outperforms the existing ones in terms of the number of blocks on statistically significant level. We also discuss potential applications of the method in image processing.

Keywords: 3D binary object, voxels, decomposition, rectangular blocks, sub-optimal algorithm, tripartite graph, maximum independent set

Classification: 65D18

1. INTRODUCTION

Binary images, both in 2D and 3D, form a specific class of objects and require dedicated algorithms for their processing and analysis. The major difference from traditional gray-level and color images is that the pixel/voxel matrix representation of binary images (which consists only of zeros and ones) is highly redundant. This has led to many specialized algorithms that employ various loss-less compressive representations for image storage and object description (see, for instance, the books [25, 28]). Such representations result not only in an efficient memory usage but also contribute to fast feature calculation and object recognition.

One of the possible approaches (and probably the most frequently used one) is to *decompose* the object into simple parts which we are able to store and process efficiently (some other approaches, such as object characterization based on its boundary and various kinds of multilevel representations, exist but are beyond the scope of this paper). Having a binary object B (by a binary object we understand a set of all pixels of a binary image whose values equal one), we decompose it into $K \geq 1$ partitions $B_1, B_2, \ldots, B_K$ such that $B_i \cap B_j = \emptyset$ for any $i \neq j$ and $B = \bigcup_{k=1}^{K} B_k$.

The 2D decomposition problem has been studied for decades in computational geometry and some of the methods were later introduced into the image analysis area.

DOI: 10.14736/kyb-2019-5-0755

Although in the continuous domain we may consider various shapes of the partitions (convex, star-shaped, hexagonal, rectangular, etc., see [20]), the decomposition methods in the discrete domain should use only rectilinear rectangular blocks because of a native rectangular structure of the discrete image domain (if other primitives were allowed, we would have to face sampling errors along the boundary).

A commonly accepted measure of the decomposition quality is the number of the resulting blocks K. This is a reasonable criterion, justified by the fact that the complexity of subsequent calculations uses to be $\mathcal{O}(K)$ and compression ratio (if the decomposition is used for compression purposes) also increases as the number of blocks decreases. The time complexity of the decomposition is usually the secondary criterion. Obviously, sophisticated decomposition methods which end up with small number of blocks usually require more time than the simple ones. Since the decomposition is in most tasks performed only once per object and can be done off-line, the time complexity becomes crucial only if it is so high that the method is not feasible in an acceptable time.[1]

Several rectangular 2D decomposition algorithms have been proposed namely in connection with compression and image feature calculation [5, 12, 22, 35, 37, 38, 40, 41] but one may find also other applications in fast spatial filtering, in iterative deconvolution methods [3], in the coded aperture imaging [21], in integrated circuits design [16, 23], and in other areas. The decomposition methods in the above cited papers are simple, intuitive but only suboptimal – they do not guarantee the minimal number of the blocks. In computational geometry, several authors [10, 24, 29] independently proposed basically the same algorithm (later discussed and improved in [9, 17]) which was proved to be optimal since it actually minimizes the number of blocks for an arbitrary input shape. The algorithm has a polynomial time complexity. This algorithm was adapted for image analysis purposes by Suk et al. [39], who also performed a large-scale experimental comparison with other methods. Their experiments proved not only the optimality but also an acceptable time-complexity of the algorithm (the time consumption was of the same order as in the case of the other methods) [39]. In this sense, the 2D decomposition problem has been fully resolved.

During the last decade, 3D image/object analysis has attracted a significant attention due to a dynamic development of 3D imaging devices and technologies. Most of the devices only measure the distance to the object surface, they cannot see "inside" the object.[2] The object is scanned from various sides such that each surface part is visible on at least one scan. The individual scans (views) are combined together and the complete surface of the object is reconstructed. We obtain 3D binary object as a result. All range finders, including the popular X-box sensor Kinect, work in that way. All popular public benchmark databases of 3D objects such as Princeton Shape Benchmark (PSB) and McGill 3D Shape Benchmark [32, 33] contain binary objects. This illustrates there is a great demand of developing efficient algorithms for working with 3D binary objects.

Few papers on 3D shape decomposition can be found in the literature but almost none of them has solved the decomposition into rectangular blocks. Ren et al. dealt with a decomposition into "nearly convex" regions [31], Sivignon and Coeurjolly [34]

[1] The term "acceptable" of course depends on the particular application.

[2] In this paper, we do not consider specialized medical devices such as MRI, CT, SPECT and PET, that actually produce a full 3D voxel cube.

decomposed the object surface into planar patches and several other authors proposed decompositions into geometric primitives and other pre-defined components [18, 42], which requires some kind of "understanding" the object structure.

To our best knowledge, the only paper on 3D shape decomposition into rectangular blocks is by Dielissen and Kaldewaij [6], who proved that decision problem of the optimal 3D decomposition (i. e. that one which minimizes K) is equivalent to a variant of the *Boolean three satisfiability* problem called 3SAT3. This means that the optimal 3D decomposition problem is NP-complete and cannot be efficiently resolved. Nevertheless, 3D decomposition can be accomplished by various sub-optimal methods. Some simple algorithms can be easily designed as an extension of 2D methods. Run-length encoding, the delta-method, and the quadtree decomposition (which turns into octree in 3D) are typical examples.

In this paper, we present a new sub-optimal algorithm. It was inspired by the optimal 2D decomposition algorithm [10, 39] but unlike the optimal 3D algorithm the proposed method is of a polynomial complexity. From this point of view, it can be considered a polynomial approximation[3] for an NP-complete algorithm. As demonstrated experimentally and by statistical tests, the proposed method outperforms both delta-method and octree decomposition significantly.

The method may find applications in 3D shape loss-less compression, in fast calculation of 3D convolution with a binary kernel, in efficient calculation of various integral transformations and integral features of 3D shapes and also in manufacturing and 3D printing of objects.

2. PRESENT STATE OF THE ART

Since 3D decomposition methods are mostly motivated by their 2D ancestors[4], we start our brief review with a description of 2D versions. Then we explain how to extend the methods into 3D.

2.1. Delta method

The Delta method (DM) was under this name originally proposed by Zakaria [41] in the connection with computation of object descriptors but it has been known also in image compression as the run-length encoding (RLE). In the basic version, the "blocks" are continuous row segments for which only the coordinate of the beginning and the length are stored. The method was later slightly improved by Dai [5] and generalized for non-convex shapes by Li [22]. The Delta method is very fast but leads to the number of blocks which uses to be (much) higher than the minimal decomposition. A simple but powerful improvement of the delta method was proposed by Spiliotis and Mertzios [37] and improved later by Flusser [12]. In this "Generalized Delta method" (GDM), the adjacent rows are compared and if there are some with the same beginning and end,

[3] To the best of our knowledge, no approximation that provably bounds the optimal solution up to a small constant factor is known. In this text, by an approximation we mean a polynomial heuristic that provides reasonable good solution compared to the NP-hard optimal algorithm.

[4] We are not aware of any 3D method which does not have a counterpart in 2D

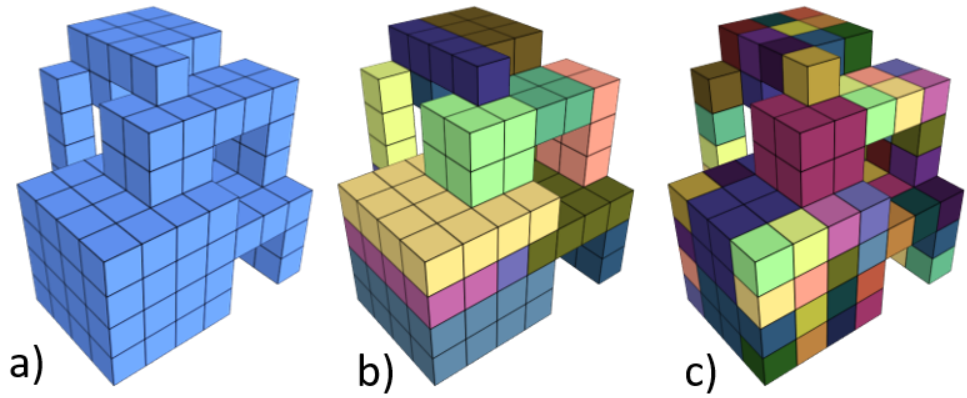

Fig. 1. A sample object (a), its decomposition by the 3GDM into 13 blocks (b), and the octree decomposition into 90 blocks (c).

they are unified to form a rectangle. The GDM is only slightly slower than the basic DM while producing (sometimes significantly) less number of blocks.

Creating a 3D version of DM is straightforward. In case of the generalized Delta method, the 3D version (denoted as 3GDM) is also clear but we have to test which direction and which order of the segment connecting yields the best result. In 2D we have only two degrees of freedom – we may choose either the vertical or the horizontal direction. As soon as the direction has been chosen, there is no choice in the row connection step. In 3D, we have an option of any of three directions and for each direction we may first connect horizontally adjacent rows and then vertically adjacent "plates" or vice versa.[5] Hence, the 3GDM which we implemented and used in the tests in this paper checks all these six options and selects that one yielding the minimum number of blocks or decides randomly if there is a multiple minimum (see Figure 1 b for an example of the 3GDM decomposition).

2.2. Quadtree and octree decompositions

In 2D, the Quadtree decomposition (QTD) is a popular hierarchical decomposition scheme used in several image processing areas including representation and compression [19], spatial transforms [11] and feature calculation [40]. In its basic version, the QTD works with square images of a size of a power of two. If this is not the case, the image is zero-padded to the nearest such size. The image is iteratively divided into four quadrants. Homogeneity of each quadrant is checked and if the whole quadrant lies either in the object or in the background it is not further divided. If it contains both

[5]The terms "horizontal" and "vertical" are here used relatively to the chosen direction.

object and background pixels, it is divided into quadrants and the process is repeated until all blocks become homogeneous. The decomposition can be efficiently encoded into three-symbol string, where 2 means division, 1 means a part of the object and 0 stands for the background.

The QTD always yields square blocks which may be advantageous for some purpose but usually leads to a much higher number of blocks than necessary. It would be possible to implement a backtracking and to unify the adjacent blocks of the same size into a rectangle but this would increase the complexity. Since the speed is the main advantage of this method, the backtracking is mostly not employed here. A drawback of this decomposition algorithm is that the division scheme is not adapted with respect to the content of the image but it is defined by absolute spatial coordinates. Hence, the decomposition is not translation-invariant and may lead to absurd results when for instance a large single square is uselessly decomposed up to individual pixels.

The QTD can be readily extended into 3D. Instead of square elements we employ cubes and the quadtree scheme is replaced with the octree (OTD). The OTD method keeps all pros and cons of its 2D ancestor (see Figure 1 c for an example of the OTD decomposition).

Octree representation has been commonly used in Minecraft-like computer games and similar voxel-represented scenarios. Popularity of this representation is mainly due to its simple implementation, fast run, and its ability to render the scene in various levels of details. However, it is much less efficient than other methods in terms of the block number.

2.3. Binary space partitioning

The term *Binary space partitioning* (BSP) denotes a wide class of hierarchical decomposition methods, very popular namely in 3D computer graphics [1, 27]. BSP recursively divides the bounding box, which contains the object, by hyperplanes. Individual BSP methods differ from one another by the criteria that select the separating hyperplane. Both QTD and OTD are special cases of BSP, where the hyperplanes are always put in the middle of the box to be decomposed. Other BSP algorithms may put the separator with regard of the object itself, but the separators always cut the entire box into two parts. This leads to object representation by the BSP tree. The BSP algorithms usually perform fast but the number of blocks depends on the chosen criterion.

2.4. Optimal decomposition

As we already pointed out, there exist a 2D decomposition method of polynomial complexity (even in several versions) which guarantees the minimum number of blocks for an arbitrary shape. Here we briefly recall the version proposed in [39].

The method performs hierarchically on two levels. On the first level, we detect all "concave" vertices (i. e. those having the inner angle 270°) of the input object and identify pairs of "cogrid" concave vertices (i. e. those having the same horizontal or vertical coordinates). Then we divide the object into subpolygons by constructing chords which always connect two cogrid concave vertices. As proved in [10] and in other papers,

the optimal choice of the chord set is such that the chords are pair-wise disjoint and their number is maximum possible.

The problem of optimal selection of the chords is equivalent to the problem of finding the maximal set of independent vertices in a graph, where each vertex corresponds to a chord and two vertices are connected by an edge if the two chords have a common point (either a concave vertex or an intersection). Generally, this problem is NP-complete, but our graph is a bipartite one, since any two horizontal (vertical) chords cannot intersect one another. In a bipartite graph, this task can be efficiently resolved. We find a maximal matching, which is a classical problem in graph theory, whose algorithmic solution in a polynomial time has been published in various versions. Some of them are optimized with respect to the number of edges, the others with respect to the number of the vertices (see [8, 14, 15, 17] for some examples of particular algorithms) but all of them are polynomial in both.

As soon as the maximal matching has been constructed, the maximal set of independent vertices can be found much faster than the maximal matching itself – roughly speaking, the maximal independent set contains one vertex of each matching pair plus all isolated vertices plus some other vertices, which are not included in the matching but still independent. As a result, we obtain a set of vertices that is unique in terms of the number of vertices being involved but ambiguous in terms of the particular vertices. However, this ambiguity does not matter – although each set leads to different object partition, the number of the components is always the same. Hence, at the end of the first level, the object is decomposed into subpolygons, which do not contain any cogrid concave vertices (see Figure 2).

The second level is very simple. Each subpolygon arriving from the first level is either a rectangle or a concave polygon. In the latter case, it must be further divided. From each its concave vertex, a single chord is constructed such that this chord terminates either on the boundary of the subpolygon or on the chord that has been constructed earlier. This is a sequential process in which each concave vertex is visited only once. The order of the concave vertices may be chosen arbitrary. Similarly, we may choose randomly between two possible chords offered in each concave vertex. This choice does not influence the final number of blocks. After that, the subpolygon is divided into rectangles, because rectangle is the only polygon having no concave vertices.

The optimal decomposition cannot be readily extended into 3D because it becomes NP-complete, as follows from the analysis presented in the next Section. The method we propose in this paper replaces the NP-complete steps by an approximation of a polynomial complexity.

3. 3D SUBOPTIMAL DECOMPOSITION

When trying to extend the above 2D optimal algorithm into 3D, we discover several substantial differences between the 2D and 3D cases. In 3D, concave edges play the role of concave vertices (see Figure 3). Concave vertices may exist, too, but they have no significance for 3D decomposition. The analogue of the chord is the *separator*, which is the intersection of a plane and the object (see Figure 4). Note that the separator not always splits the entire object into two separate components as it is illustrated in Figure 5. Similarly as in 2D, any concave edge must be contained in a separator to

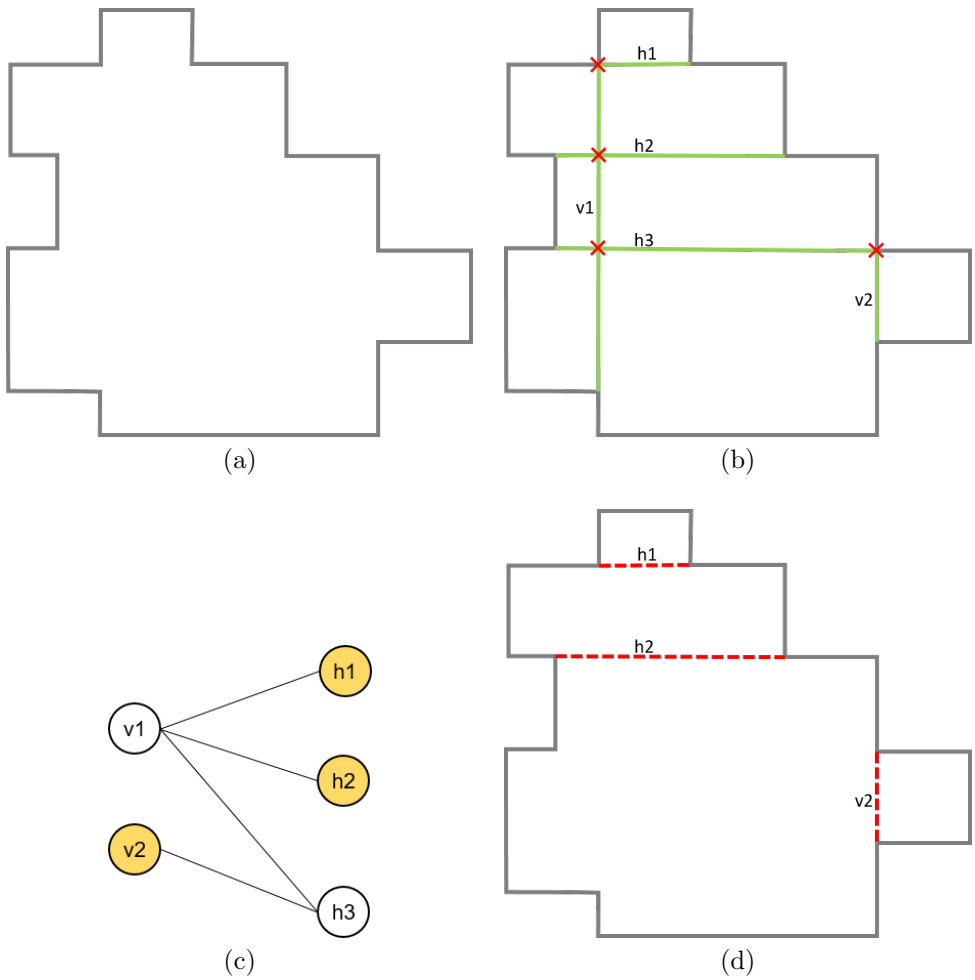

Fig. 2. The first level of the 2D optimal decomposition method. (a) The input object. (b) All possible chords connecting two cogrid concave vertices. The crosses indicate the chord intersections. (c) The corresponding bipartite graph with a maximum independent set of three vertices. Other choices are also possible, such as $\{h_1, h_2, h_3\}$. (d) The first level of the object decomposition.

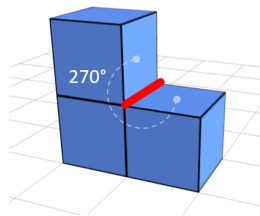

Fig. 3. A concave edge formed by two voxel faces of a 270° angle in between.

get the decomposition into blocks. Separators containing no concave edges are possible but useless. Unlike 2D, where a chord can contain two concave vertices at maximum, a separator can contain an *arbitrary high number* of concave cogrid edges (the edges laying in a plane which is perpendicular to an axis are called *cogrid edges*). From this we can see that the 3D version of the optimal algorithm (if it exists) cannot work in two levels but rather in m levels, where m depends on (but not necessarily equals to) the maximum number of the existing cogrid concave edges. Another difference from 2D is that a separator may split a perpendicular concave edge into two separate concave edges. In this way, placing a separator eliminates some concave edges but may at the same time induce new ones, which is impossible in 2D (see Figure 6). The most significant difference is, however, the following one. Even if we place the separators in order given by the number of the concave edges they eliminate, we do not end up with the minimum number of blocks. Placing a separator which eliminates the maximum possible number of the concave edges at the particular moment may not be globally optimal since it may prevent placing some separator(s) which would finally lead to a better decomposition (see Figure 8 for illustration of such simple situation). Before we fix the separator, we should check the complete subtree of all other alternatives. This makes the task NP-complete.[6]

3.1. The basic version

The basic version of a sub-optimal algorithm is a heuristics which basically follows from the above thoughts. It works iteratively in a greedy manner. In each iteration, we place proper separators and cut the object by them. This eliminates all edges connected with the chosen separators. We repeat this step until all concave edges have disappeared.

The tricky part in each iteration is how to choose the proper separators. We have already shown that the optimal brute force approach is NP-complete. To overcome this, we choose the separators according their *weight*. The weight w_s of separator s is a function which estimates how significant (i. e. how useful) is the particular separator for the decomposition. Intuitively, it should reflect the number of the concave edges the separator eliminates and should be easy to evaluate (preferably in a polynomial time). Two possible particular choices of w_s will be discussed later.

[6]Note that the NP-completeness was formally proven in [6] by transforming the decomposition problem onto a 3SAT3 problem.

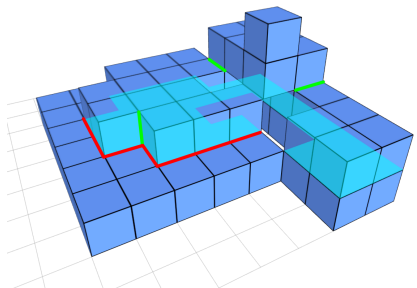

Fig. 4. A separator is a cross-section of a plane and the object. Meaningful separators eliminate some concave edges of the object. In this example, the red edges have been eliminated by a cyan separator.

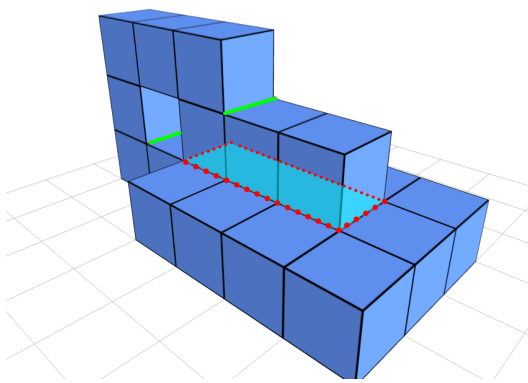

Fig. 5. Example of a separator which is meaningful (it eliminates red dotted edges) even if it does not split the object into separated parts.

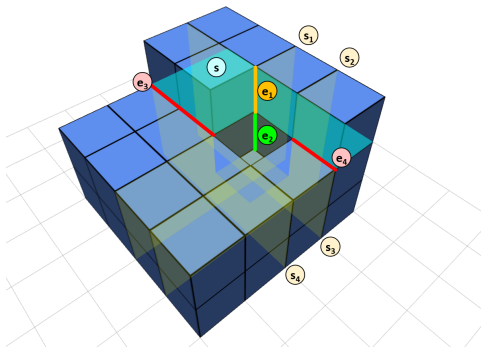

Fig. 6. An example of a separator that splits perpendicular concave edges and intersects other separators. Cyan-highlighted separator s eliminates edges e_3, e_4 and intersects one vertical edge that has been divided into e_1 and e_2. It also intersects other separators s_1 and s_2 and is adjacent to separators s_3 and s_4

In each iteration, the algorithm finds all possible separators and calculates their weights. Let us denote the highest weight as α and the set of all separators with this weight as M. Now the method tries to place as many separators from M as possible but at the same time it must avoid all mutually intersecting separators because they are redundant (by "intersecting" we understand also adjacent separators, i. e. those which share an edge), see Figs. 6 and 7. In other words, we are looking for a maximum subset of M of non-intersecting separators.

This task can be reformulated as a task of finding the maximum independent set in a tripartite graph, which is a well-known problem in graph theory. We refer to the maximum independent set in graph G as $MaxIS(G)$ or $MaxIS$ for short.

We construct graph $G = (V, E')$ whose vertices are the separators from the set M. Vertices u and v form edge $(u, v) \in E'$ iff the corresponding separators intersect each other. Graph G is tripartite because parallel separators (along axes x, y and z) are always disjoint. Example of a graph construction is shown in Figure 7. The maximum independent set of vertices $MaxIS$ gives us the largest set of disjoint separators of weight α. We split the object by these separators and proceed to the next iteration. Note that the set of the separators may not be unique because the graph may contain more than one $MaxIS$ set of the same cardinality. In such a case we chose the separator set randomly.

We repeat the iterations until all concave edges have been eliminated. Depending on the particular choice of w_s, α may not monotonically decrease during the iteration process. At the end, the object has been decomposed into rectangular blocks. The partitioning may sometimes produce adjacent blocks that share one side and therefore they can be merged into a single block. As soon as the iterations have been completed, we find and merge these adjacent blocks. Since the optimal merging of the blocks is an

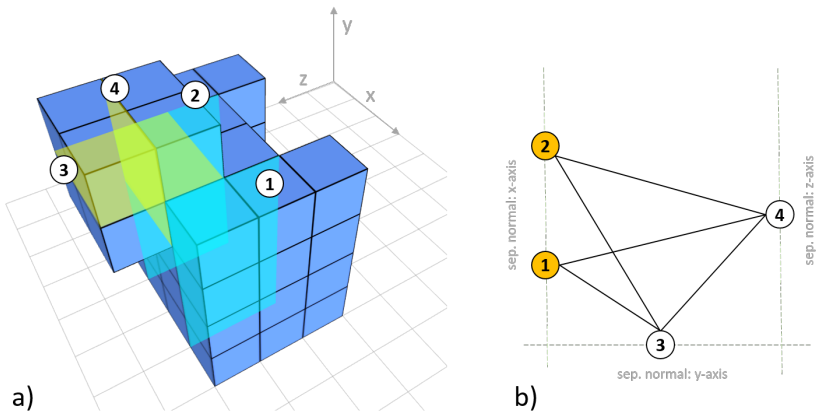

Fig. 7. An example of a graph construction: On the left, there are four separators of the same weight. On the right, we can see the corresponding tripartite graph. The graph vertices are associated with the separators and the graph edges reflect their mutual intersections (or adjacency). In this example, highlighted vertices {1, 2} form the $MaxIS$ (compare with corresponding disjoint separators 1 and 2).

NP complete problem, we applied a simplified linear heuristics that merges the adjacent blocks.

3.2. The weight function

As we already explained, the choice of the weight function w_s determines the sub-optimal approximation of the full-search technique. It should describe the significance of the separator for the decomposition. High weights should be given to separators, the early placement of which leads to a low number of blocks. At the same time, the evaluation of the weight of each separator should be fast enough. This is why we limit ourselves to two weight functions, both of which can be evaluated directly on the current level and do not require any recursive hierarchical calculations.

The first one simply counts the number of the concave edges which the separator eliminates when placed

$$w_s^{(1)} = |\{e \mid e \in E \wedge e \subset s\}|. \tag{1}$$

In the example in Figure 5, the highlighted separator has the weight $w_s^{(1)} = 4$ since it contains four concave edges.

A more sophisticated choice (but also slightly more time-consuming to evaluate) which reflects the fact that the separator may also generate some new concave edges is

$$w_s^{(2)} = |\{e \mid e \subset s\}| - |\{e \mid e \perp s\}|, e \in E \tag{2}$$

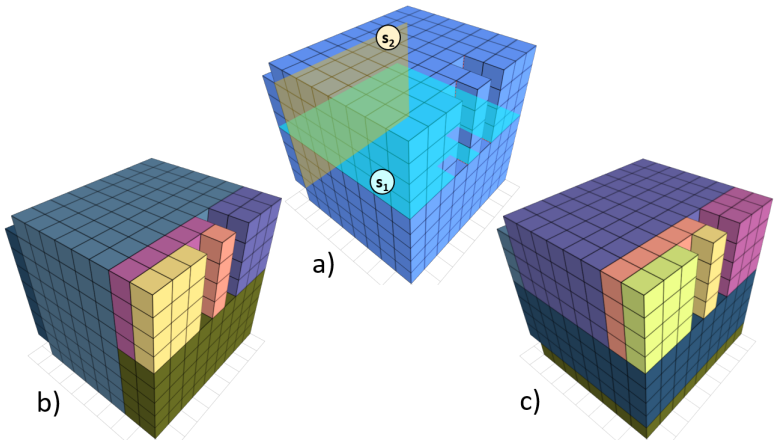

Fig. 8. An illustration that neither $w_s^{(1)}$ nor $w_s^{(2)}$ select the optimal separators. At the first step, both weight functions prefer s_1 to s_2 because $w_{s_1}^{(1)} = 6$ and $w_{s_1}^{(2)} = 4$ while $w_{s_2}^{(1)} = w_{s_2}^{(1)} = 2$. However, placing s_1 leads to 8 blocks (see c)), while using s_2 yields the optimal decomposition into 7 blocks.

which is in fact $w_s^{(1)}$ minus the number of the concave edges perpendicular to and intersected by the separator.

In the experimental section we will compare the performance of $w_s^{(1)}$ and $w_s^{(2)}$, among others.

3.3. Implementation

In the following pseudocode, we describe the algorithm more formally. Placing the separator in the object is implemented in a way that the separator becomes "final" and forms a "wall" that cannot be divided any further. We first search for all concave edges and all separators that contain them. Then we iteratively choose maximum sets of disjoint separators with the highest weight and move them to the set of walls. The concave edges, eliminated by these separators, are removed from the list and new edges (if any) are added. As soon as the iteration process has been completed, the blocks are formed by original object surface and/or by "final inner walls" created by the separators. For each block we store the coordinates of its upper left front voxel and three block dimensions. The last step – block merging – is accomplished by lexicographic sorting the blocks w.r.t. x, y, z, identifying adjacent blocks of the same side size and updating the data in the block list. The complexity of the merging is only $\mathcal{O}(K' \log K')$ where K' is the number of blocks produced by the iterative part of the algorithm. (Note that the block merging step is due to the sub-optimality of the algorithm. If the decomposition was optimal, no merging would be possible and this step could be removed from the algorithm.)

The most time-consuming part is finding the $MaxIS$ on the line 9 of the algorithm. For general graphs, this problem is NP-hard. Although the graph we work with is a tripartite one, which is much simpler than a general graph, finding the maximum independent set is still NP-hard w.r.t. the number of separators of the same weight. Theoretically, this number may be proportional to the number of all surface voxels and it also depends on the shape of the particular object. However, in average, there is a high correlation ($r = 0.82$) of the object volume and the average number of separators among all iteration steps as it is illustrated in Figure 10. It should be noted that the number of separators in majority of the iteration steps is usually much lower namely for high α, but may become so high for low α that the algorithm may not be feasible.

This is another substantial difference from the 2D case – finding the maximum independent set in a bipartite graph is of a polynomial complexity [39]. In the next Section, we propose an approximation of a polynomial time complexity, which we use in our implementation.

We implemented the decomposition for Node.js framework. In addition to standard node.js package written in Javascript, we created an online visual tool that runs in any modern browser that supports WebGL (such as the latest versions of Google Chrome). In this tool, the user can interactively create a custom object or upload an external object, run different decomposition methods (the proposed algorithm can be even executed step-by-step for better understanding of the process). The visualization helps in understanding the behavior of the decomposition algorithms. The tool was designed mainly for educational purposes, it is not meant for routine work. It is available online at `http://goo.gl/hAEuCg`, a sample screenshot can be seen in Figure 9.

Algorithm 1 Sub-optimal 3D decomposition

$E \leftarrow$ find concave edges
$S \leftarrow$ find separators
$W \leftarrow \emptyset$ ▷ the set of final walls
while $|S| > 0$ **do**
  $w_s \leftarrow$weight(s), $\forall s \in S$ ▷ calc. weight for each sep.
  $\alpha \leftarrow \max_{s \in S}(w_s)$ ▷ calc. the max. weight
  $M \leftarrow \{s \mid w_s = \alpha \wedge s \in S\}$ ▷ separators of max. weight
  $G = (V, E') \leftarrow v_s \in V \Leftrightarrow s \in M, (v_s, v_p) \in E' \Leftrightarrow s \perp p$ ▷ create graph
  $I \leftarrow MaxIS(G)$ ▷ find max. indep. set of vertices
  $F \leftarrow \{s \mid v_s \in I\}$ ▷ seps. chosen in MaxIS become final
  $W \leftarrow W \cup F$ ▷ move final seps. to the set of walls
  $C \leftarrow \{c \mid c \perp s \wedge c \in S \wedge s \in F\}$ ▷ intersecting separators
  $N \leftarrow$ new divided separators that replace C
  $S \leftarrow (S \cap C \cap F) \cup N$ ▷ remove final seps., add divided seps.
  divide all $e \in E$ that intersect any $s \in F$ ▷ split edges that inters. walls
end while
convert voxels bounded by walls $w \in W$ into rectangular blocks
merge adjacent blocks

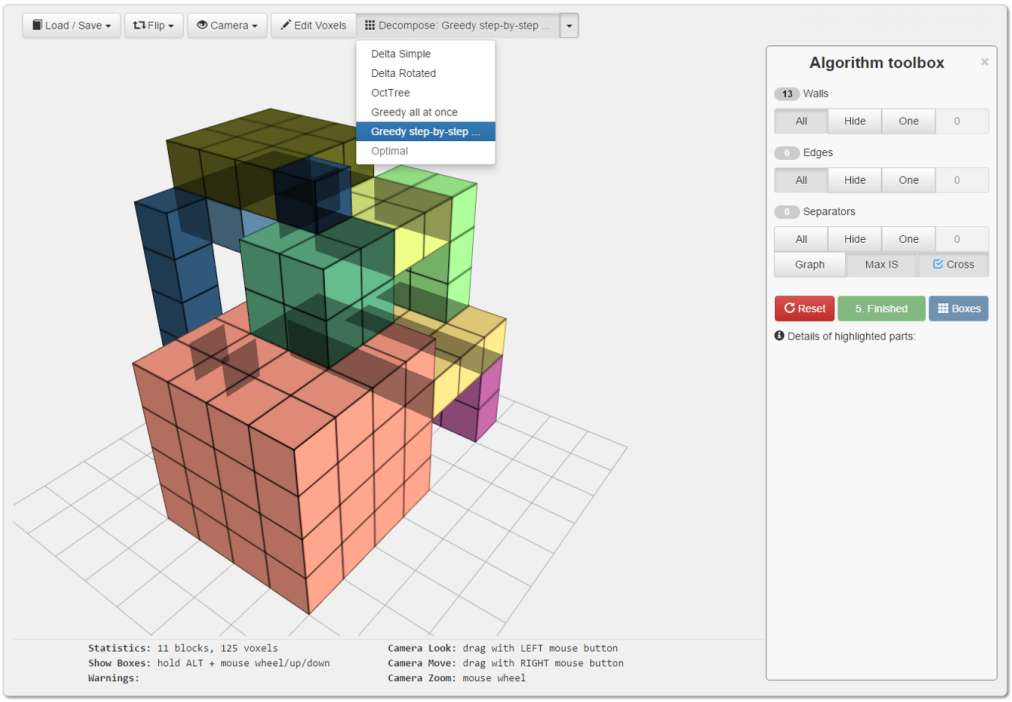

Fig. 9. A sample screen shot of our online decomposition tool.

3.4. Approximating the maximum independent set

The vertices of graph $G = (V, E')$ can be clustered into three disjoint subsets P_x, P_y, P_z according to the axis that the corresponding separators are perpendicular to

$$G(V, E') = G(P_x \cup P_y \cup P_z, E'). \tag{3}$$

Clearly, each subset is composed of parallel separators which cannot intersect each other and hence G is a tripartite graph because there are no graph edges inside the individual subsets.

The complexity of finding the maximum independent set of vertices of G is exponential w.r.t. the number of the vertices and edges, which varies in individual iterations. If the number of the vertices is low, which is a typical situation at the beginning of the algorithm when the maximum weight α is high, the exponential time may be acceptable. Finding the $MaxIS$ is equivalent to finding the maximal clique in the complement graph[7], so we adopted the popular Bron-Kerbosch algorithm [2] for the clique problem to find the $MaxIS$. This is, however, not feasible for large graphs, typically arriving in

[7]Complement graph H to the given graph G consists of the same set of vertices and complementary set of edges, i. e. two distinct vertices of H are adjacent if and only if they are not adjacent in G.

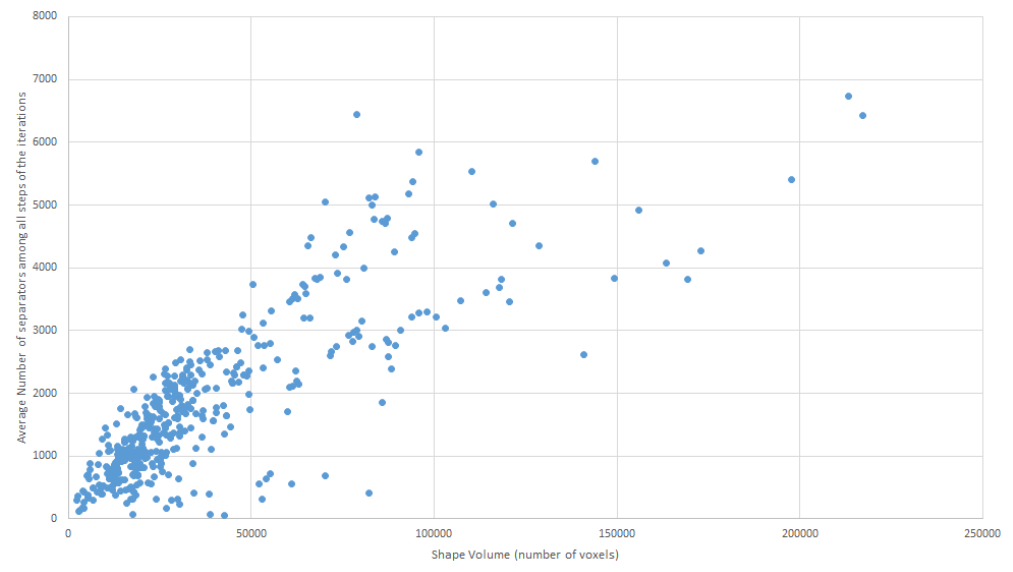

Fig. 10. Correlation ($r = 0.82$) between the object volume and the number of the separators.

case of complex objects with many concave edges at the iteration levels when the weight approaches one.

Very simple approximation for $MaxIS$ is just to take the largest subset among P_x, P_y, P_z instead (let us denote it as $IS^{(1)}$). The independence is guaranteed but for most (especially large) graphs this approximation is far from the actual $MaxIS$.

A better approach is to treat this problem in tripartite graphs as an extension of the bipartite graph problem. We choose two largest vertex subsets among P_x, P_y, P_z and consider a subgraph of G (let us denote it as G_2), which is a bipartite graph. On G_2 we find the exact maximum independent set $I_2 = MaxIS(G_2)$. This is solvable in a polynomial time thanks to the König's theorem [4]. We implemented this step by means of the maximum network flow algorithm by Edmonds and Karp [8] of a time-complexity $\mathcal{O}(VE'^2)$ and also alternatively by the Dinic's algorithm [7] with the complexity $\mathcal{O}(V^2E')$. Since the $G(V, E')$ depends on the shape of each object, we first construct the graph and then we choose the E.K. alg. if $|V||E'|^2 < |V|^2|E'|$, or the Dinic's alg. otherwise. Finally, we unify I_2 with those vertices from the remaining third part of the graph which are not adjacent to any vertex of the selected independent set. We denote this final independent set as $IS^{(2)}$ (see Figure 11).

The choice of how to calculate/approximate the $MaxIS$ can be in our implementation done by the user. It is always a trade-off between the time efficiency and the size of the independent set (which consequently influences the number of blocks). The optimal solution provides the correct maximum set, but it is NP-hard and thus for complicated

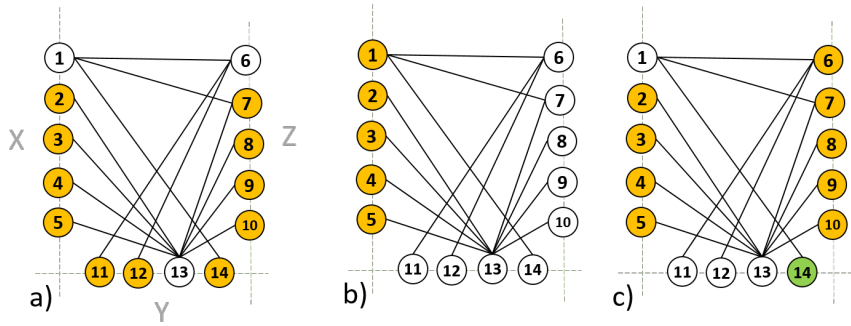

Fig. 11. Finding maximum independent set in a tripartite graph. (a) optimal solution $MaxIS$, (b) an approximative heuristic $IS^{(1)}$ where only the largest part is taken, (c) better approximative heuristic $IS^{(2)}$ as an extension of bipartite subgraph $MaxIS$ with the vertex No. 14 appended from the third part of the graph.

objects it can run unacceptably long time. $IS^{(1)}$ is retrieved very quickly, but the set is much smaller and thus leads to more blocks in the final decomposition. $IS^{(2)}$ provides a very good compromise, as demonstrated in the next Section by experiments.

4. EXPERIMENTS

The main goal of this Section is to compare the proposed decomposition method with the state of the art Octree algorithm [26], the BSP algorithm where the separator was chosen such that it minimizes the number of voxels it passes through, and namely with the generalized delta method, which is known from 2D to be very powerful. The second goal is to study the performance of various modifications of the proposed algorithm. We verify that the polynomial heuristic functions $IS^{(1)}$ and $IS^{(2)}$ provide good approximation of the optimal NP-hard solution of the $MaxIS$. Additionally, we compare the two different weight functions that evaluate the separators' significance and verify that the enhanced $w^{(2)}$ performs significantly better than $w^{(1)}$.

4.1. Models from the McGill database

The first round of experiments was run on a database of 416 voxelized models from the McGill 3D Shape Benchmark[8] [33] (we denote this set as "MDB"). All models have been inscribed into a $128 \times 128 \times 128$ cube. Each object has a different volume, but together the whole MDB contains more than 13.5 millions of voxels.

We decomposed all shapes by the OTD, BSP, 3GDM, and by the proposed method with various settings (see Figure 12 for some examples). The test results are summarized in Table 1. In the fourth and fifth rows, we used $MaxIS$ algorithm with the separator weights $w^{(1)}$ and $w^{(2)}$, respectively. Comparison of the performance of these two weights

[8]Some of these objects can be found in the Princeton Shape Benchmark [32], too.

Fig. 12. Examples of the decomposed models from the McGill database [33]. The decomposition was performed by the proposed method.

Method	Graph IS method	Weight function	Total No. of blocks $[10^3]$	Mean time per object [s]
Octree	N/A	N/A	3413	1.3
BSP	N/A	N/A	512	4.6
3GDM	N/A	N/A	458	0.7
Proposed	$MaxIS$	$w^{(1)}$	448	88.6
Proposed	$MaxIS$	$w^{(2)}$	445	107.5
Proposed	$IS^{(1)}$	$w^{(2)}$	450	67.4
Proposed	$IS^{(2)}$	$w^{(2)}$	445	83.3

Tab. 1. The number of the decomposition blocks for the entire MDB achieved by various methods.

was done by Wilcoxon test. The null hypothesis was that there is no significant difference between these two sample decompositions. The null hypothesis was rejected with p-value < 0.001, which led us to the conclusion that $w^{(2)}$ performs significantly better. On the last two rows of the table we can see the most important results of the experiment - decomposition achieved by heuristics $IS^{(1)}$ and $IS^{(2)}$. In both cases, solely the better weight $w^{(2)}$ was used. A surprising result is that the polynomial heuristic $IS^{(2)}$ yields almost the same number of blocks as the optimal NP-hard algorithm $MaxIS$. This was confirmed by the Wilcoxon test – the null hypothesis was accepted with the p-value > 0.1. This result proves the efficiency of $IS^{(2)}$ algorithm. When applying $IS^{(1)}$ heuristic, the decomposition works faster but in average it yields slightly higher number of blocks. Since the differences are more or less consistently spread over the whole database, the Wilcoxon test rejected the null hypothesis with p-value < 0.001. A comparison with the 3GDM is clearly in favor of the proposed method. For both $IS^{(1)}$ and $IS^{(2)}$ (and of course also for $MaxIS$), the Wilcoxon test confirmed that 3GDM performed significantly worse on this database.

Summarizing, the most important result of the test is the following. Polynomial heuristics $IS^{(2)}$ with the weight $w^{(2)}$ is statistically equivalent (in terms of the block number) to NP-hard $MaxIS$ algorithm and is at the same time significantly better than all other tested methods. This is expressed visually in Figure 13.

4.2. Random cubes

The aim of this experiment was to compare the performance of the 3GDM and the proposed method on the set of irregular, less compact objects. We used one hundred $32 \times 32 \times 32$ cubes inside which we randomly "carved" holes of an average density 50%. We employed these random cubes because they do not exhibit any dominant edge direction, unlike most of the real objects. Hence, the comparison is not biased by object anisotropy. The proposed method was applied in both configurations, i. e. using $MaxIS$ and $IS^{(2)}$ algorithms. A sample cube along with its decompositions can be seen in Figure 14. The individual number of blocks are shown "cube by cube" in Figure 15 and the summary results are in Table 2. Similarly as on the MDB objects, we can

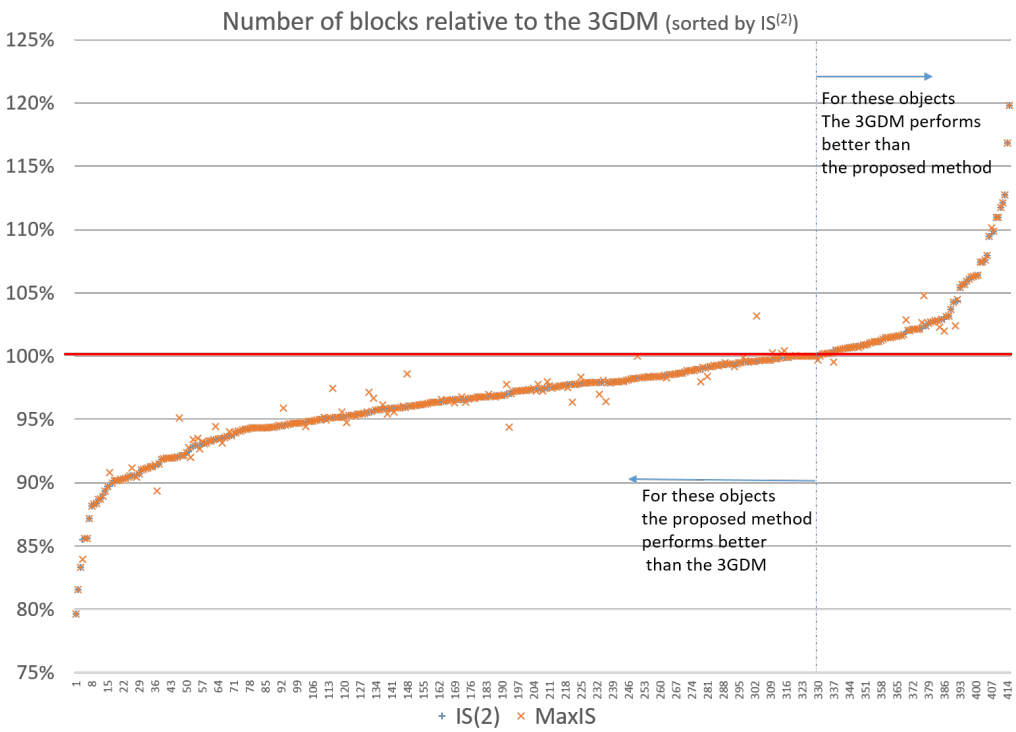

Fig. 13. Comparison of 3GDM and the proposed method in terms of the block number. Red solid line corresponds to the 3GDM which is taken as an etalon. The objects have been sorted according to the differences between 3GDM and $IS^{(2)}$ (denoted as blue "+"). The differences between 3GDM and $MaxIS$ are denoted as orange "×".

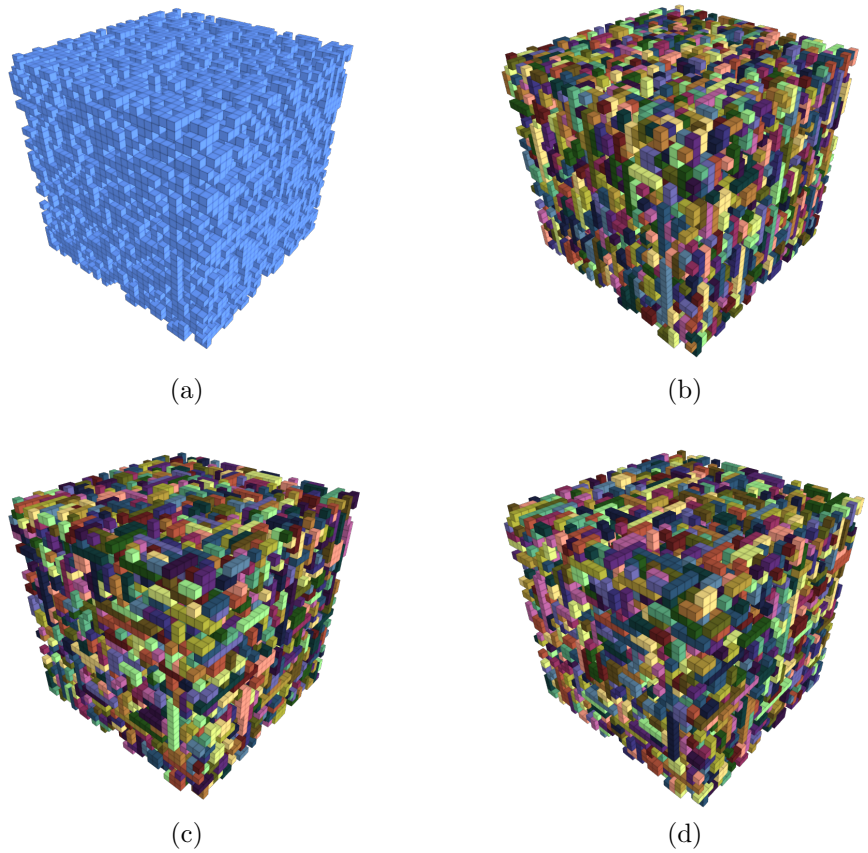

Fig. 14. A sample random cube (16449 voxels) (a), its 3GDM decomposition (7174 blocks) (b), $MaxIS$ decomposition (6458 blocks) (c), and $IS^{(2)}$ decomposition (6521 blocks) (d).

clearly see the insignificant difference between $MaxIS$ and $IS^{(2)}$, and significantly worse performing 3GDM and BSP (these conclusions were again confirmed by the Wilcoxon test).

5. APPLICATIONS

Potential applications of the proposed decomposition method can be found in all areas where decomposition of 3D shapes is required and where the number of the blocks is the main issue. This is typically if the decomposition is performed off-line, if it is then used many times in subsequent calculations, and if the number of blocks influences substantially the time and/or the cost of a subsequent processing. If, on the other hand, realtime or close-to-realtime decomposition is required as a primary criterion, then the 3GDM method provides the best solution.

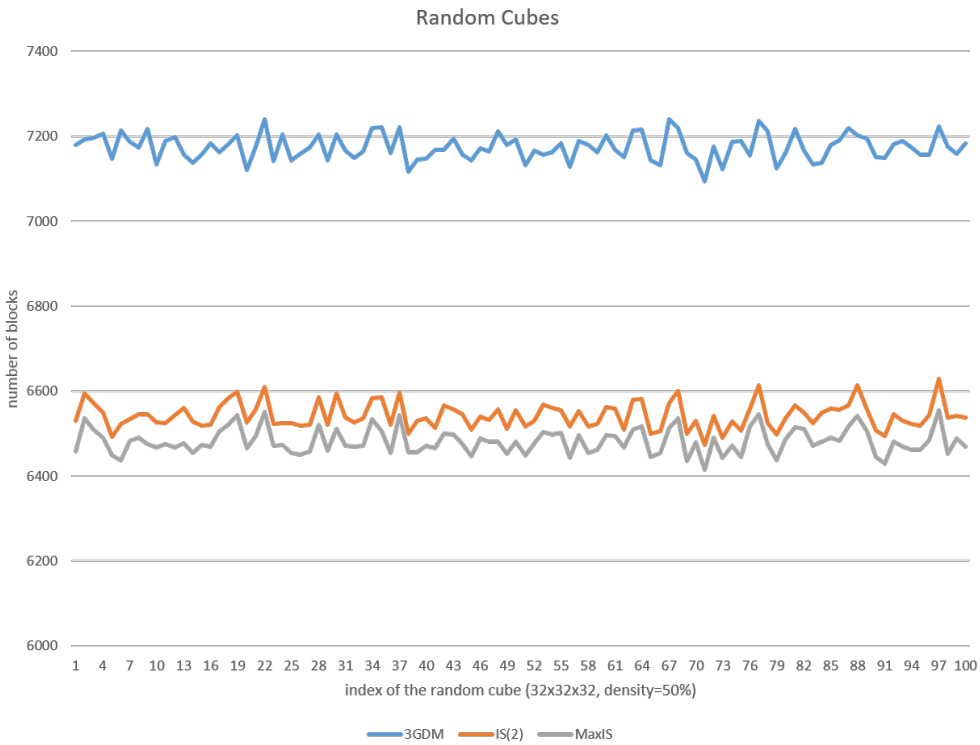

Fig. 15. The number of blocks of the decomposition of 100 random cubes. Blue line (top) – 3GDM, orange line (middle) – $IS^{(2)}$, gray line (bottom) – $MaxIS$.

Method	Total No. of blocks $[10^3]$
BSP	838
3GDM	717
$MaxIS$	648
$IS^{(2)}$	654

Tab. 2. The total number of blocks of the decomposition of 100 random cubes by various methods.

5.1. Compression

Our method can be used in 3D shape encoding/compression, both loss-less and lossy ones. In a loss-less compression, we store the position and the size of each block. To optimize the compression ratio, we order the blocks according to their size such that the blocks of the same size form a substring. Then we store only the positions of the blocks while the size is stored only once for each substring. In a lossy compression, we throw away the smallest blocks, typically from $1 \times 1 \times 1$ to a certain limit. This significantly improves the compression ratio but many shape details may disappear when the object has been reconstructed.

5.2. Feature calculation

Many features, which have been proposed for 3D shape description and recognition, are of the form of an integral transformation

$$M_{\mathbf{p}}^{(f)} = \int_{\Omega} \pi_{\mathbf{p}}(\mathbf{x}) f(\mathbf{x}) \mathrm{d}\mathbf{x} \ . \tag{4}$$

where $\mathbf{p}$ is a 3D multi-index, $\{\pi_{\mathbf{p}}(\mathbf{x})\}$ is a set of basis functions of the image space (transformation kernels), $f(\mathbf{x})$ is characteristic function of the shape, and Ω is a bounded subset of R^3. Fourier coefficients, wavelet coefficients, and image moments are few popular examples [13]. If we decompose the object into K disjoint blocks B_k, Eq. (4) can be rewritten as

$$M_{\mathbf{p}}^{(f)} = \sum_{k=1}^{K} \int_{B_k} \pi_{\mathbf{p}}(\mathbf{x}) \mathrm{d}\mathbf{x} \ . \tag{5}$$

If the basis functions $\pi_{\mathbf{p}}(\mathbf{x})$ can be integrated on a rectangular region by means of primitive functions and Newton-Leibnitz theorem in $\mathcal{O}(1)$ time (which is the case of all polynomial and harmonic bases), then the evaluation of $M_{\mathbf{p}}^{(f)}$ from Eq. (5) is of $\mathcal{O}(K)$ complexity while the direct calculation from Eq. (4) is proportional to the total number of the object voxels.

The object features are typically computed for a large set of the basis functions and used repeatedly, so the time benefit of the decomposition may be really huge even if the decomposition itself might be relatively slow.

5.3. Fast convolution

When we calculate a convolution of 3D image (graylevel or color) f with a binary kernel h, we can benefit from the decomposition as well. If we decompose the support of h into disjoint blocks, then we have

$$f * h = f * \sum_{k=1}^{K} h_k = \sum_{k=1}^{K} f * h_k, \tag{6}$$

where h_k is a characteristic function of block B_k.

The evaluation of convolution of arbitrary f with rectangular B_k in a single voxel can be accomplished in $\mathcal{O}(1)$ time regardless of the block size. We precompute partial sums of f in all three dimensions, so we create an auxiliary array g such that

$$g(L, M, N) = \sum_{\ell=1}^{L} \sum_{m=1}^{M} \sum_{n=1}^{N} f(\ell, m, n).$$

This array is precomputed only once for the given f and may be used for any h. The convolution $f * h_k$ in a certain voxel is then evaluated just from the values of g which correspond to the corners of properly shifted block B_k. This approach is significantly faster than calculating convolution from the definition, which is proportional to the kernel size, and also than using FFT; especially when multiple convolutions with the same kernel are to be calculated. Such a situation arises for instance in deblurring/sharpening of CT images (and other data of a similar nature) when the blurring kernel is known since it characterizes the imaging device. Iterative deconvolution methods, such as popular Richardson-Lucy algorithm, iteratively estimate the input image, evaluate many times convolutions with the same kernel, and check in each iteration the similarity to the blurred acquired image.

The kernel decomposition is particularly efficient if the kernel matrix has a full or close-to-full rank because in that case we cannot effectively use any fast convolution algorithm based on kernel factorization.

5.4. Manufacturing

Manufacturing of 3D structures is often done by assembling them from simple components. If these components are rectangular blocks, then our algorithm can be advantageously applied because the production cost and time are proportional to the number of blocks, while durability of the product uses to be inversely proportional to it. The time of decomposition, which is performed on a computer model of the product, is negligible comparing to the total production time.

6. CONCLUDING DISCUSSION

In this paper, we presented an original method of block-wise decomposition in 3D. The method is a double approximation of the optimal algorithm, which is NP-complete and practically infeasible. We proposed the criterion for the separator selection in the first heuristic approximation. In the second approximation, the maximum independent set in a tripartite graph, the finding of which is again NP-complete (and to our knowledge is also hard to approximate), is replaced by a polynomial sub-optimal solution. We proved by large-scale experiments that the proposed method is statistically better than the 3GDM, which is the best one among the existing methods in terms of the number of blocks. This determines that potential applications of the proposed method can be found namely in the tasks where it is more desirable to keep the number of blocks as low as possible rather than to minimize the decomposition runtime.

It is interesting that even though the proposed algorithm is generally sub-optimal, it is optimal on specific shapes. A trivial example of such class are objects, which do not

have any cogrid concave edges. Another class where our method performs optimally, is formed by objects all concave edges of which are parallel.[9] This is not true for the main competitor, 3GDM, which is still suboptimal on these objects.

Although in the paper we have been dealing with binary shapes only, all methods can be theoretically used for graylevel and color 3D images as well. A graylevel image can be expressed as a sum of several binary 3D images multiplied by a proper constant, which can be obtained either as intensity slices or bit slices (these techniques were proposed for 2D images in [30, 36] and can be adapted to 3D case readily). We can decompose each slice independently by means of our algorithm. However, these binary "images" use to be highly fragmented (especially low bit planes resemble a "random chessboard") and their decomposition typically yields high number of blocks. Hence, this kind of decomposition is of a little practical importance for graylevel/color 3D images.

For the sake of completeness, we should mention that the format of binary images used in some databases is not based on a voxel representation. A common representation of a shape is by triangular patches (facets) of the surface. In that case, only the vertices of the triangles are stored. Such a representation may be often used directly for compression, feature calculation, and other tasks. This is, however, not true in general. Calculating convolution and integral transformations in triangular representation may be very tricky or even impossible. If we want to find a rectangular decomposition, we have to convert the triangular representation to the voxel one first. Although several conversion algorithms exist (see, for instance, [13] for one of the simplest methods), this approach is not very efficient and the triangular representation is mostly used without any conversion to voxels.

ACKNOWLEDGEMENT

This work was supported by the Czech Science Foundation (Grant No. GA18-07247S) and by the *Praemium Academiae*, awarded by the Czech Academy of Sciences.

(Received December 20, 2018)

REFERENCES

[1] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars: Computational Geometry: Algorithms and Applications. 3rd Edition. Springer-Verlag TELOS, Santa Clara 2008. DOI:10.1007/978-3-540-77974-2

[2] C. Bron and J. Kerbosch: Algorithm 457: Finding all cliques of an undirected graph. Commun. ACM *16* (1973), 9, 575–577. DOI:10.1145/362342.362367

[3] P. Campisi and K. Egiazarian: Blind Image Deconvolution: Theory and Applications. CRC, 2007. DOI:10.1201/9781420007299

[9] This property can be proven formally. First, we prove that in this case all concave edges must have their beginning in the same object face. The same is true for the edge ends. This means that such object is an "elongation" of a 2D shape, which forms the object face, into the third dimension. Our algorithm is equivalent to 2D optimal decomposition of the base and "elongation" of the rectangles into blocks. Hence, our method must be optimal, too.

[4] W. Cook, W. Cunningham, W. Pulleyblank, and A. Schrijver: Combinatorial Optimization. Wiley Series in Discrete Mathematics and Optimization, Wiley 2011. DOI:10.1002/9781118033142.scard

[5] M. Dai, P. Baylou, and M. Najim: An efficient algorithm for computation of shape moments from run-length codes or chain codes. Pattern Recognition *25* (1992), 10, 1119–1128. DOI:10.1016/0031-3203(92)90015-b

[6] V. J. Dielissen and A. Kaldewaij: Rectangular partition is polynomial in two dimensions but np-complete in three. Inform. Process. Lett. *38* (1991), 1, 1–6. DOI:10.1016/0020-0190(91)90207-x

[7] E. A. Dinic: Algorithm for solution of a problem of maximum flow in a network with power estimation. Soviet Math. Doklady *11* (1970), 1277–1280.

[8] J. Edmonds and R. M. Karp: Theoretical improvements in algorithmic efficiency for network flow problems. J. Assoc. Comput. Machinery *19* (1972) 2, 248–264. DOI:10.1145/321694.321699

[9] D. Eppstein: Graph-theoretic solutions to computational geometry problems. In: 35th International Workshop on Graph-Theoretic Concepts in Computer Science WG'09, Vol. LNCS 5911, Springer, 2009, pp. 1–16. DOI:10.1007/978-3-642-11409-0_1

[10] L. Ferrari, P. V. Sankar, and J. Sklansky: Minimal rectangular partitions of digitized blobs. Computer Vision, Graphics, Image Process. *28* (1984), 1, 58–71. DOI:10.1016/0734-189x(84)90139-7

[11] J. Flusser: An adaptive method for image registration. Pattern Recognition *25* (1992), 1, 45–54. DOI:10.1016/0031-3203(92)90005-4

[12] J. Flusser: Refined moment calculation using image block representation. IEEE Trans. Image Process. *9* (2000), 11, 1977–1978. DOI:10.1109/83.877219

[13] J. Flusser, T. Suk, and B. Zitová: 2D and 3D Image Analysis by Moments. Wiley, 2016. DOI:10.1002/9781119039402

[14] A. V. Goldberg and S. Rao: Beyond the flow decomposition barrier. J. Assoc. Comput. Machinery *45* (1998), 5, 783–797. DOI:10.1145/290179.290181

[15] A. V. Goldberg and R. E. Tarjan: A new approach to the maximum-flow problem. J. Assoc. Comput. Machinery *35* (1988), 4, 921–940. DOI:10.1145/48014.61051

[16] K. D. Gourley and D. M. Green: A polygon-to-rectangle conversion algorithm. IEEE Computer Graphics Appl. *3* (1983) 1, 31–36. DOI:10.1109/mcg.1983.262975

[17] H. Imai and T. Asano: Efficient algorithms for geometric graph search problems. SIAM J. Comput. *15* (1986), 2, 478–494. DOI:10.1137/0215033

[18] A. Jain, T. Thormählen, T. Ritschel, and H.-P. Seidel: Exploring shape variations by 3d-model decomposition and part-based recombination. Computer Graphics Forum *31* (2012), (2pt3), 631–640. DOI:10.1111/j.1467-8659.2012.03042.x

[19] E. Kawaguchi and T. Endo: On a method of binary-picture representation and its application to data compression. IEEE Trans. Pattern Analysis Machine Intell. *2* (1980), 1, 27–35. DOI:10.1109/tpami.1980.4766967

[20] J. M. Keil: Polygon decomposition. In: Handbook of Computational Geometry, Elsevier, 2000, pp. 491–518. DOI:10.1016/b978-044482537-7/50012-7

[21] A. Levin, R. Fergus, F. Durand, and W. T. Freeman: Image and depth from a conventional camera with a coded aperture. In: Special Interest Group on Computer Graphics and Interactive Techniques Conference SIGGRAPH'07, ACM, New York 2007. DOI:10.1145/1275808.1276464

[22] B. C. Li: A new computation of geometric moments. Pattern Recognition *26* (1993), 1, 109–113. DOI:10.1016/0031-3203(93)90092-b

[23] W. Liou, J. Tan, and R. Lee: Minimum partitioning simple rectilinear polygons in $O(n \log \log n)-$time. In: Proc. Fifth Annual ACM symposium on Computational Geometry SoCG'89, ACM, New York 1989, pp. 344–353. DOI:10.1145/73833.73871

[24] W. Lipski Jr., E. Lodi, F. Luccio, C. Mugnai, and L. Pagli: On two-dimensional data organization II. In: Fundamenta Informaticae, Vol. II of Annales Societatis Mathematicae Polonae, Series IV, 1979, pp. 245–260. DOI:10.1007/978-1-4613-9323-8_18

[25] S. Marchand-Maillet and Y. M. Sharaiha: Binary Digital Image Processing: A Discrete Approach. Academic Press, 1999. DOI:10.1016/b978-012470505-0/50007-1

[26] D. Meagher: Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-d Objects by Computer. Tech. Rep. IPL-TR-80-111, Rensselaer Polytechnic Institute 1980.

[27] T. M. Murali, P. K. Agarwal, and J. S. Vitter: Constructing binary space partitions for orthogonal rectangles in practice. In: Proc. 6th Annual European Symposium on Algorithms, ESA '98, Springer-Verlag, London 1998, pp. 211–222. DOI:10.1007/3-540-68530-8_18

[28] F. B. Neal and J. C. Russ: Measuring Shape. CRC Pres, 2012. DOI:10.1201/b12092

[29] T. Ohtsuki: Minimum dissection of rectilinear regions. In: Proc. IEEE International Conference on Circuits and Systems ISCAS'82, IEEE, 1982, pp. 1210–1213.

[30] G. A. Papakostas, E. G. Karakasis, and D. E. Koulouriotis: Efficient and accurate computation of geometric moments on gray–scale images. Pattern Recognition *41* (2008), 6, 1895–1904. DOI:10.1016/j.patcog.2007.11.015

[31] Z. Ren, J. Yuan, and W. Liu: Minimum near-convex shape decomposition. IEEE Trans. on Pattern Analysis and Machine Intelligence *35* (2013), 2546–2552. DOI:10.1109/tpami.2013.67

[32] P. Shilane, P. Min, M. Kazhdan, and T. Funkhouser: The Princeton Shape Benchmark. In: Proc. Shape Modelling Applications, 2004. DOI:10.1109/smi.2004.1314504

[33] K. Siddiqi, J. Zhang, D. Macrini, A. Shokoufandeh, S. Bouix, and S. Dickinson: Retrieving articulated 3-d models using medial surfaces. Mach. Vision Appl. *19* (2008), 4, 261–275. DOI:10.1007/s00138-007-0097-8

[34] I. Sivignon and D. Coeurjolly: Minimum decomposition of a digital surface into digital plane segments is NP-hard. Discrete Appl. Math. *157* (2009), 3, 558–570. DOI:10.1016/j.dam.2008.05.028

[35] J. H. Sossa-Azuela, C. Yáñez-Márquez, and J. L. Díaz de León Santiago: Computing geometric moments using morphological erosion. Pattern Recognition *34* (2001), 2, 271–276. DOI:10.1016/s0031-3203(99)00213-7

[36] I. M. Spiliotis and Y. S. Boutalis: Parameterized real-time moment computation on gray images using block techniques. J. Real-Time Image Process. *6* (2011), 2, 81–91. DOI:10.1007/s11554-009-0142-0

[37] I. M. Spiliotis and B. G. Mertzios: Real-time computation of two-dimensional moments on binary images using image block representation. IEEE Trans. Image Process. *7* (1998), 11, 1609–1615. DOI:10.1109/83.725368

[38] T. Suk and J. Flusser: Refined morphological methods of moment computation. In: 20th International Conference on Pattern Recognition ICPR'10, IEEE Computer Society, 2010, pp. 966–970. DOI:10.1109/icpr.2010.242

[39] T. Suk, C. Höschl IV, and J. Flusser: Decomposition of binary images – A survey and comparison. Pattern Recognition *45* (2012), 12, 4279–4291. DOI:10.1016/j.patcog.2012.05.012

[40] C.-H. Wu, S.-J. Horng, and P.-Z. Lee: A new computation of shape moments via quadtree decomposition. Pattern Recognition *34* (2001), 7, 1319–1330. DOI:10.1016/s0031-3203(00)00100-x

[41] M. F. Zakaria, L. J. Vroomen, P. Zsombor-Murray, and J. M. van Kessel: Fast algorithm for the computation of moment invariants. Pattern Recognition *20* (1987), 6, 639–643. DOI:10.1016/0031-3203(87)90033-1

[42] Y. Zhou, K. Yin, H. Huang, H. Zhang, M. Gong, and D. Cohen-Or: Generalized cylinder decomposition. ACM Trans. Graph. *34* (2015) 6, 171:1–171:14. DOI:10.1145/2816795.2818074

Cyril Höschl IV, Czech Academy of Sciences, Institute of Information Theory and Automation, Pod Vodárenskou věží 4, 182 08 Praha 8. Czech Republic.
e-mail: hoschl@utia.cas.cz

Jan Flusser, Czech Academy of Sciences, Institute of Information Theory and Automation, Pod Vodárenskou věží 4, 182 08 Praha 8. Czech Republic.
e-mail: flusser@utia.cas.cz