Evaluation of Manifold Dual Contouring Algorithms Based on *k*-d tree and Octree Data Structures

Thabang Ramaijane, Hlomani Hlomani, Irina Zlotnikova, and Thabiso Maupong
Department of Computing and Informatics, School of Pure and Applied Sciences, Botswana International University of Science and Technology, Botswana
E-mail: thabang.ramaijane@gmail.com, hlomanihb@biust.ac.bw, zlotnikovai@biust.ac.bw, maupongt@biust.ac.bw

Keywords: dual contouring, isosurface extraction, k-d trees, octrees, adaptive data structures, simplification errors thresholds, quadratic error functions

Received: November 14, 2023

This study evaluated the performance and efficiency of manifold dual contouring algorithms using k-d trees and octrees, therefore, addressing a critical gap in comparative analysis of these data structures. Despite the popularity of k-d trees and octrees in isosurface extraction, their performance has not been empirically compared. This research specifically focuses on visualization quality, performance metrics, and efficiency across various simplification error thresholds. A comprehensive comparative analysis was conducted to identify the conditions under which each data structure is most suitable. Both algorithms employed the manifold vertex clustering scheme for dual contouring of isosurfaces. The methodology involved evaluating visual output, build time, extraction time, and efficiency based on triangle counts during the simplification process. Computational experiments demonstrated that the octree-based algorithm is superior for rendering large models, producing an average of 20% more visual detail due to a higher triangle count. In contrast, the k-d tree-based algorithm showed a 40% reduction in build time and a 35% reduction in extraction time, making it more efficient for processing large implicit models by reducing geometric complexity. These findings provide metrics to assist researchers and practitioners in selecting the most suitable adaptive data structure for achieving optimal simplification results in manifold dual contouring algorithm implementations.

Povzetek: Narejena je primerjava algoritmov za 3D obrobe, ki uporabljajo k-d drevesa in okt-drevesa. Zadnji omogoča več podrobnosti, k-d drevo pa hitrejšo obdelavo in gradnjo.

1 Introduction

An *isosurface* is a group of contour lines (*isolines*) in three dimensions where each of these isolines is made of connected points of a fixed scalar value *(an isovalue)* [1]. The purpose of isosurface extraction is to visualize these contour lines in three dimensions (3D) [1]. Applications of 3D visualization include virtual reality [2], [3], augmented reality [4], and computer-aided architectural design [5]. When visualising a 3D dataset, an isosurface extracting algorithm must handle the processing of the geometry of that particular dataset [6]. Isosurface extraction algorithms are implicit surface representation techniques [7].

The underlying surface geometry of an object must be represented as accurately as possible [8]. One of the first algorithms for the representation of high resolution isosurfaces, *the marching cubes (MC)* algorithm by Lorensen and Cline [9], had demonstrated inconsistencies in the way it generated the final visual depictions of extracted isosurfaces. Numerous attempts have been made up to date to overcome the weaknesses of this foundational algorithm. Schaefer and Warren [10] proposed a method for contouring an implicit function using a grid topologically dual to structured grids such as octrees named *dual marching cubes*. By aligning the vertices of the dual grid with the features of the implicit function, Schaefer and Warren were able to reproduce thin features of the extracted surface without excessive subdivision required by the MC or dual contouring (DC) methods. Lee et al. [11] revised the triangulated cubes of the MC algorithm in order to regularize the connectivity of the isosurface mesh and, therefore, to maximize the valence of six vertices. Jin et al. [12] pointed out that the weaknesses of the MC algorithm came from the use of surface configurations of cubes, including wrong surface production and hole generation. In response to that, Jin et al. proposed an improvement of the MC method by reassigning a value of zero to vertices inside the surface and a value of one to vertices on the surface, and then redefining 15 typical configurations considered in the marching cubes method [12]. Strand and Stelldinger [13] modified the MC algorithm by applying it to the face-centered cubic grid meaning that the local configurations considered when extracting the local surface patches were not cubic. Xu et al. [14] advanced the traditional MC algorithm by (1) replacing the cube edge linear interpolation with the midpoint selection, (2) using the index of the intersection point

to avoid repeated calculation and (3) contracting edge to reduce the number of triangular patches. Vignoles et al. [15] proposed a simplified MC algorithm in which the surface consisted of triangles composed from vertices of the regular 3D grid on which the processed data was defined. Du et al. [16] made suggestions on enhancement of the original MC algorithm focusing mainly on improving efficiency and topological accuracy of the original algorithm. Greß and Klein [17] used the *k*-d trees for the representation of implicit objects. The *k*-d tree grid was built on the assumption that an appropriate grid partitioning criterion for subdividing splitting planes was determined. Construction of the grid took a top-down routine as follows. For every subdivision step, the active grid edges had to be detected, and their related intersection information was computed. To achieve two-manifold and topologically correct representations, the subdivision scheme maintained a list of faces in the cell. This allowed faces contained in the list to be clipped against the splitting plane throughout each subdivision step. These faces were transmitted to the corresponding sub-cells after being clipped. After generating each vertex during the clipping process, a record was made of the axes corresponding to the planes clipped against them. However, like other previous studies, Greß and Klein [17] did not compare the performance of the *k*-d tree-based algorithm with octree-based algorithms. This paper, therefore, discusses the use of the *k*-d tree as the adaptive data structure alternative to the octree for the representation of isosurfaces extracted from implicit models. Although the octree data structures have already been proposed and implemented for achieving topological improvements in several reviewed studies, the focus of this paper is on evaluating the algorithms' performance in accelerating isosurface extraction under various topology simplification thresholds. The evaluation of both algorithms allowed to determine which data structure would be most adaptive depending on the values of simplification error thresholds. The rest of the paper is organized as follows. Section 2 discusses the fundamental concepts and mathematical equations necessary for the understanding of the material presented in this paper. It also provides the detailed analysis of similar studies. Section 3 discusses the proposed approach. Section 4 outlines the methodology for implementation and evaluation of the algorithms. The results are presented in Section 5. In Section 6, the results are discussed and compared with similar reviewed studies. Finally, the conclusions, limitations and future work are considered in Section 7.

2 Literature review

This section lays a foundation for a better understanding of the material presented in this paper by discussing the fundamental concepts and mathematical equations, as well as providing an overview of existing similar studies.

2.1 Background to the study

This subsection defines the fundamental concepts, such as adaptive data structures and quadratic error functions (QEFs), and discusses their role in isosurface simplification strategies.

2.1.1 Adaptive data structures

The adaptive data structures discussed throughout the paper include *k*-d trees and octrees. These structures simplify isosurface extraction on uniform grids, allowing the simplified grid to adapt to the increasing size of the implicit models being processed. The adaptive nature of these data structures ensures that they can efficiently handle large datasets, making them suitable for applications requiring dynamic and scalable isosurface extraction.

A *k*-d tree is a hierarchical multi-dimensional data structure used for subdividing volume datasets and accelerating isosurface extraction [17]. It subdivides the volume one dimension at a time, resulting in binary partitions at each stage. This method allows the *k*-d tree to be applied in any orientation based on the chosen plane, without a predefined traversal sequence, making it particularly useful for adaptive processing and efficient representation of complex volumetric data. The *k*-d tree is a binary tree where every node is a k-dimensional point [18]. The *k*-d tree performs a binary partition in one dimension at a time. Figure 1 illustrates a typical *k*-d tree volume subdivision.

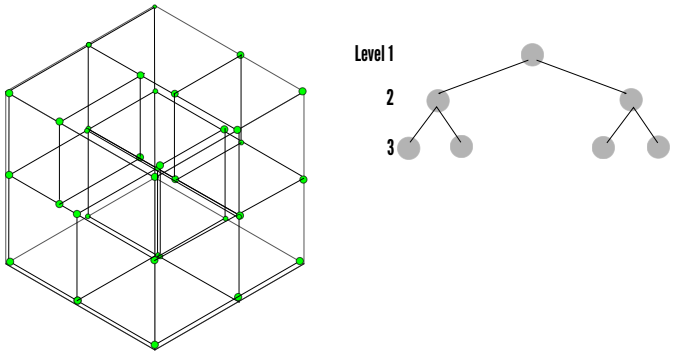

Figure 1: *K*-d tree volume subdivision

An octree is a hierarchical data structure primarily used to represent a decomposed three-dimensional volume, with each node corresponding to a specific sub-volume [19]. Each subdivision produces eight octants, being divided as long as the root node represents the whole volume (Figure 2). An octree simultaneously subdivides a volume in three dimensions, unlike the *k*-d tree which performs a binary partition in one dimension at a time. This data structure is typically employed in isosurface extraction applications to partition uniform and rectilinear grids into subregions, with each leaf node representing a single grid cube. The adaptive nature of the octree allows it to handle large datasets efficiently, as demonstrated by Ju et al.[20].

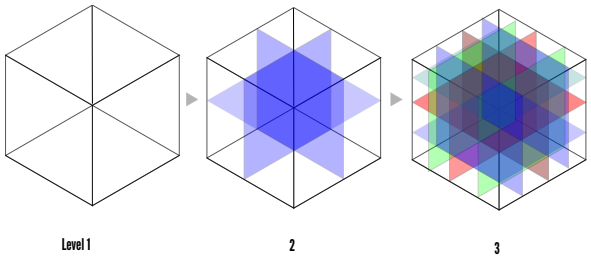

Figure 2: Octree volume subdivision - Levels 1 to 3

2.1.2 Isosurface simplification strategies for adaptive data structures

Simplifying isosurfaces for topologically correct surface representations of volumetric or implicit datasets can be effectively and efficiently achieved using adaptive data structures (i.e., *k*-d trees and octrees). However, the approaches can vary between different implementations.

The three approaches considered in this subsection include (1) error-constrained *k*-d tree grid simplification, (2) octree-based simplification using QEFs, and (3) octree-based simplification through manifold vertex clustering.

The error-constrained *k*-d tree grid simplification (i.e., the simplification of the *k*-d tree via error metrics) is initiated by successively removing minimal partitioning faces from the grid. The simplification process is terminated if the computed error exceeds a user-defined threshold. The intersection points and normals for the *k*-d tree's working grid edges are represented by a set of planes, determining the primitive shape representing the isosurface of the rendered object [10].

The octree simplification using QEFs is a geometry simplification scheme introduced as part of the standard dual contouring algorithm. The QEF for each corresponding leaf is computed as given by Equation (1) (Ju et al. [20]):

$$E[x] = \sum_i (n_i \cdot (x - p_i))^2, \tag{1}$$

where n_i and p_i refer to the resulting intersections (including unit normals) between the contour and edges of the cube. The QEFs are computed at the interior nodes of the octree by combining these QEFs with their corresponding leaves. Nodes whose QEFs have a residual smaller than a given threshold are collapsed into leaves, thereby simplifying the octree.

The octree-based simplification through manifold vertex clustering involves constructing a vertex tree for vertices at the finest level of the octree and determining the QEF for each clustered vertex. To simplify the octree, only vertices topologically related to the surface are clustered together. Each clustered vertex must satisfy an additional topology criterion to ensure it represents a simplified contour that is two-manifold, referred to as *the manifold criterion* [21]:

$$X(S_v) = V(S_v) - E(S_v) + F(S_v), \tag{2}$$

where $X(S_v)$ is the Euler characteristic, $V(S_v)$ is the total number of vertices, $E(S_v)$ is the total number of edges, and $F(S_v)$ is the total number of faces.

2.1.3 Metrics used in topology simplification schemes

Dual contouring based simplification of isosurfaces involves the representation and minimization of a relevant QEF to achieve accurate results. Equation (1) in 2.1.2 is constructed through a collection of intersection points p_i and their normals n_i. The function $E[x]$ can be considered as the inner product $(Ax-b)^T(Ax-b)$, where A is a matrix whose rows are the normals n_i, and b is a vector resulting from $n_i \cdot p_i$. The QEF can be expanded as shown in Equation (3):

$$E[x] = x^T A^T A x - 2x^T A^T b + b^T b, \tag{3}$$

where the matrix $A^T A$ is a symmetric 3×3 matrix, $A^T b$ represents a three-length column vector, and $b^T b$ is a scalar.

Consequently, only the matrices $A^T b$, $A^T A$, and $b^T b$ need to be stored in 10 floats for optimization, rather than storing the matrices A and b. The minimization value $\hat{x}$ required for $E[x]$ can be obtained by solving the normal equation (4):

$$A^T A\hat{x} = A^T b. \tag{4}$$

Another application of QEFs in dual contouring algorithms is for computing error metrics. However, it differs from the method used for *k*-d tree error-constrained simplification of isosurfaces. Although both approaches calculate simplification errors through matrix manipulation of intersection points and Euclidean distances, the computation of the simplification error E in the *k*-d tree method is based on the position of a set of points relative to the simplified object, rather than on a minimization value [22]. The error between the simplified and original representation can be computed by taking the least Euclidean lengths $d_S(v)$ between the set of points $v \in P_{\text{on}}$ on the surface representation of the simplified isosurface, and the lengths $d_I(P_{\text{in}})$ and $d_O(P_{\text{out}})$ from points $p_{\text{in}} \in P_{\text{in}}$ and $p_{\text{out}} \in P_{\text{out}}$ on the exit of the inner and outer of the simplified object. These error metrics can then be categorized as *in-metric*, *on-metric*, and *out-metric*. The on-metric error E_{on} is defined from the distances $d_S(v)$ and measures the distance between the original and simplified isosurface representations. The in-metric E_{in} and out-metric E_{out} errors are established from the relevant distances $d_I(P_{\text{in}})$ and $d_O(P_{\text{out}})$, facilitating the computation of the error through topological modifications. These metrics can be defined for user-specified error thresholds ϵ_{on}, ϵ_{in}, and ϵ_{out} as

$$E_{\text{on}} \leq \epsilon_{\text{on}}, \quad E_{\text{in}} \leq \epsilon_{\text{in}}, and \quad E_{\text{out}} \leq \epsilon_{\text{out}}. \tag{5}$$

Intersection points and normals for the *k*-d tree's working grid edges can be represented by a set of planes S, determining the primitive shape representing the isosurface. To compute the distance of a point relative to the isosurface, the corresponding planes closest to the polygon must

be identified at each completion of the isosurface extraction.

Suppose a plane $s \in S$ consists of a set of points P_s nearest and sampled from the associated polygon P_{on}. The on-metric is defined as

$$E_{\text{on}} = \max_{s \in S} \frac{1}{|P_s|} \sum_{v \in P_s} d_S(v)^2, \tag{6}$$

where $d_S(v)$ represents the distance of a point v from the plane s. Let I denote the exit of the inside of an entity according to the k-d tree grid representation, with O as the exit of the object's exterior. The in-metric and out-metric can be expressed as follows:

$$E_{\text{in}} = \max_{p_{\text{in}} \in P_{\text{in}}} d_I(p_{\text{in}}), \tag{7}$$

$$E_{\text{out}} = \max_{p_{\text{out}} \in P_{\text{out}}} d_O(p_{\text{out}}). \tag{8}$$

Given that the length d_I from p_{in} to the exit of the inside of the object I correlates with the distance to the edge of the surface S when p_{in} and zero otherwise, the equations can be rewritten as follows:

$$E_{\text{in}} = \max_{p_{\text{in}} \in IP_{\text{in}}} d_S(p_{\text{in}}), \tag{9}$$

$$E_{\text{out}} = \max_{p_{\text{out}} \in OP_{\text{out}}} d_S(p_{\text{out}}). \tag{10}$$

In a manner similar to the minimization and representation of QEF, the exact evaluation of finding the value of the on-metric depends on finding the summation of squared distances $E(n, d)$ among points $v_1, \ldots, v_k \in P_{\text{on}}$ and the plane $s \in S$ as defined by a normal n along with a scalar d. This can be performed with an equation similar to the general QEF equation (1):

$$E(n, d) = \frac{1}{k} \sum_{i=1}^{k} (n^T v_i + d)^2. \tag{11}$$

By considering the following coefficients,

$$A = \sum_{i=1}^{k} v_i v_i^T, \quad b = \sum_{i=1}^{k} v_i, \quad c = k, \tag{12}$$

where A represents a symmetric 3×3 matrix, b is a 3-vector, and c is a scalar, Equation (11) can be further expressed as:

$$E(n, d) = \frac{1}{k}(n^T A n - 2dn^T b + d^2 c). \tag{13}$$

Given $P = (A, b, c)$, which is referred to as *the dual quadric*, and assuming the plane passes through the mean of points v_i, then the quantity is $d = -\frac{n^T b}{c}$. The corresponding normal n is associated with the eigenvector of the covariance matrix Z made of points v_i, which corresponds with the least eigenvalue. According to the dual quadric, Z can be computed as follows:

$$Z = \frac{1}{k-1}\left(A - \frac{bb^T}{c}\right). \tag{14}$$

2.2 Related work

Isosurface extraction refers to a broad range of techniques that can be used in the visualization of three-dimensional scalar data [23]. The literature review provides an analysis of two of the most common techniques, namely the *marching cubes* and *dual contouring* algorithms, and their combination, the *dual marching cubes* algorithm. The review also discusses the various motivations behind further development of each technique.

2.2.1 The marching cubes algorithm and its modifications

The foundational marching cubes (MC) algorithm of isosurface extraction for scientific visualization of high resolution 3D volumetric data was developed by Lorensen and Cline in 1987 [9]. The MC algorithm took as input a uniform grid whose vertices were samples of the function *f(x, y, z)* and extracted a surface as the zero-contour. For each cube in the grid, the MC algorithm examined the values at the eight corners of the cube and determined the intersection of the surface with the edges of the cube. A look-up table indexed by the sign configuration at the eight corners that yielded the topology of the surface in side of that cube was provided. Processing each cube in the grid resulted in the complete surface. There have been numerous modifications of this reference algorithm aiming at its further improvement and/or simplification. An extensive survey of the marching cubes algorithms looking at the evolution of the standard MC algorithm from 1987 to 2006 can be found in [24].

Rajon and Bolch [25] considered the original MC algorithm and several of its modifications available at the time of their research, as well as the issues identified in each reviewed MC-based alrorithm. Their specific interest was to generate isosurfaces delineating tissue interfaces from the gray-level medical (trabecular bone) images. Rajon and Bolch indicated that the rectangular shape of image voxels generated voxel effects that altered the outcome of isosurface generation. To minimize voxel effects, Rajon and Bolch proposed an adaptation of the MC algorithm in which a trilinear interpolation of the gray levels was used to generate a hyperboloid surface. According to Rajon and Bolch, the adapted technique was capable of solving the ambiguity problem of the original MC algorithm. It also allowed to reduce the data size inherent to the triangulated surface and provided a simplified algorithm to accurately measure distances within the image. The trilinear interpolation method removed voxel effects and produced chord-length distributions across image regions.

Maple [26] suggested an application of the original MC algorithm and its two-dimensional variation (*marching squares algorithm*) for geometric design and space plan-

ning. The proposed addition to the MC algorithm allowed to approximate the area or volume of the object. This method could also be used in estimation of the area encapsulated between two points on the surface and a line or the volume encapsulated between three points on the surface and a plane.

Research by Andújar et al. [27] focused on improving the efficiency of the MC algorithm. The proposed solution was to select one valid topology that would minimize a desired topological or combinatorial measures (i.e., the total triangle count, the number of connected shells, or the total genus). Since the measures to be minimized in this research were not a□ected by the placement of the vertices of the isosurface mesh, Andújar et al. suggested to position each vertex (i.e., an angular point) of the isosurface at the midpoints of the lattice edges joining inside and outside samples. Moving vertices to more appropriate locations along their lattice edges and, therefore, preserving the topological and combinatorial properties of the isosurface led to a more efficient algorithm.

Lee et al. [11] proposed *the modified marching cubes (MMC)* algorithm by reconsidering the triangulated cubes of the original MC algorithm. The MMC algorithm allowed for regularization of the connectivity of the isosurface mesh. Consequently, a maximum valence of six vertices was achieved in the modified algorithm. The MMC algorithm demonstrated an improvement of the mesh topology and a significant reduction in the connectivity coding cost. The algorithm utilized an approach whereby a simple remark was based on how the isosurface intersected its associated height voxel cube.

Almost all methods based on MC utilize a look-up table to triangulate the isosurface. Renbo et al. [28] presented a variation of the MC algorithm that did not use a conventional look-up table. Instead, it relied on an automated triangulation strategy based on critical points lying on the isosurface in the interior of the cube. These critical points were classified as face shoulder points, body shoulder points, and inflection points. According to Renbo et al., for any case of cube configuration, the improved algorithm was capable of generating a topologically accurate approximation to the isosurface of the trilinear interpolant within the cube. The accuracy of the reconstruction process, as compared to the original MC algorithm, was improved using some extra points located on the characteristic positions of isosurface.

Jin et al. [12] identified disadvantages in the original MC algorithm that came from the use of surface configurations of cubes leading to the wrong surface production and hole generation. In response to these weaknesses, the authors proposed an improvement of the MC method by reassigning zero to vertices in the surface and one to vertices on the surface, and then redefining 15 typical configurations considered in the marching cubes method. They reassigned zero to vertices in the surface and one to vertices on the surface, and then improved the marching cubes algorithm by redefining the configurations for the fifteen cases of configurations considered in the original marching cubes method. To evaluate the performance of the proposed improved marching cubes method, Jin et al. reconstructed the surface for MRI volume data using the proposed method along with the original MC method to demonstrate improvement.

Strand and Stelldinger [13] presented three adaptations of the original MC algorithm for preserving the topology of the marching cube producing different local configurations on a face-centered cubic (FCC) grid. The first adaptation presented a combination of the three partitioning schemes employed in earlier MC algorithms (i.e., a Delaunay mesh partitioning [29]). Another adaptation involved tetrahedra mesh partitioning of the FCC grid and possible local configurations resulting in a least simplified instance of topology preserved MC. The third adaptation used rhombic dodecahedra partitioning and demonstrated the best simplification results out of three implementations.

Cui and Liu [30] created *the simplified marching cubes (SpMC)* algorithm as a modification of the standard MC algorithm. They suggested a change in the position of the isosurface extracted vertex at an interpolation point on a cube edge which corresponded to the cube vertex. The modified algorithm required fewer triangulation cases and, in most cases, fewer extracted triangles than the original MC algorithm. The SpMC algorithm did not need interpolation calculations of vertex positions and normal vectors required in the original MC algorithm.

Research by Etiene et al. [31] did not aim at improving the original MC algorithm or any of its variations. Instead, they presented a tool for selection of the most appropriate algorithm among MC variations. Etiene et al. presented techniques for assessment of the behavior of isosurface extraction codes and verification of various visualization algorithms. These techniques were used to distinguish whether anomalies in isosurface features could be attributed to the underlying physical process or to artifacts from the extraction process. Etiene et al. argued the necessity of ”verifiable visualization” - subjecting visualization algorithms (including MC-based ones) to the same verification process as in any other academic discipline. The focus of the verification was on topological properties of isosurface extraction algorithms. Etiene et al. derived formulas for the expected order of accuracy (or convergence rate) of several isosurface features, and compared them to experimentally observed results in the selected codes. According to Etiene et al., results of the verification of various algorithms (i.e., the MC algorithm and its variants) could assist with the selection of the appropriate isosurface extraction technique and visualization algorithm.

Xu et al. [14] suggested three enhancements of the original MC algorithm. The first enhancement was to replace the cube edge linear interpolation with a midpoint selection scheme. The triangular patches generated in this way led to a smoother isosurface in local area thus contributing to the mesh simplification. The second improvement was the use of a three-dimensional array to store the coordinates of the

points of intersection between the cube edge and the isosurface. This helped to avoid repeated calculation of these points. The third enhancement was to employ the edge contraction method to reduce the number of triangular patches. When two vertices satisfied the stated constraint conditions, the two vertices would merge into one, and the edge consisting of the two vertices would contract into a point. If the conditions were not met, the triangle would be kept.

Vignoles et al. [15] suggested a triangulation method leading to the development of *the simplified marching cubes (SMC)* algorithm. The SMC algorithm relied on a mesh built with vertices linked together into triangles of a cuberille grid. Triangles were composed of vertices of the cuberille grid on which the processed data is defined. In the cuberille model, an object is typically represented as a collection of cube shaped voxels [32]. Meshes were *manifold*, i.e., topologically consistent and without holes. The accuracy of the obtained meshes was reported to be lower than in the original MC algorithm, but higher than in the cuberille approximation. The algorithm produced a configuration with a reduced number of triangles as a result of a reduction in the number of vertex points.

The modification by Du et al. [16] focused mainly on improving efficiency and topological accuracy of the original algorithm. To address the issue of efficiency, they altered the way in which an isosurface intersected the hexahedral voxels. The isosurface usually only intersects some voxels. The standard MC algorithm checked and computed all hexahedron voxels in the three dimensional data area. This operation required significant time and resulted in the low efficiency. In the proposed solution, if the hexahedron voxels intersect with the isosurface, the isosurface would be in the continuity along the six surfaces of the hexahedron. If a cube intersects with the isosurface, there are intersection lines on some surfaces of the six surfaces on the cube. The surfaces of the adjacent cube (front, back, upper, lower, left and right) would be extended according to a certain order. Du et al. proved that 90% of the isosurfaces were composed of these six cases. It helped to significantly reduce the time and, therefore, improve efficiency. The issue of topological accuracy was addressed in the following way. In the original MC algorithm, the nodal between the isosurfaces and the voxel boundary was computed on the assumption that the function value changed linearly along the voxel boundary. When the density is high in the three dimensional discrete data area, i.e., when the voxels are very small, this assumption is true. However, if the voxels are large in sparse data area then the isosurfaces could not be accurately obtain. Therefore, the recommendation was to use high data field density and small voxels, so the model would be reconstructed in the high accuracy and exact structure.

Custodio et al. [33] developed *the corrected marching cubes 33 (C-MC33)* algorithm addressing topological issues of the marching cubes 33 algorithm (MC33). The original marching cubes 33 algorithm was, in its turn, a modification of the classical MC algorithm proposed by Tcherniaev [34] and further extended by Lewiner et al. [35]. The MC33 algorithm was one of the first modifications of the MC algorithms aiming to preserve the topology of the trilinear interpolant. The main contribution of Tcherniaev was that the original MC triangulation table was extended to 33 cases – hence, the name of the algorithm. Custodio et al. found several issues in the MC33 algorithm by Tcherniaev and its later extension by Lewiner et al. The identified issues included (1) disambiguation in Cases 10, 12 and 13.5 of the 33 cases considered by Tcherniaev [34], (2) non-manifold surfaces [34], and (3) failure to compute cut plane heights [35]. Custodio et al. presented solutions for the identified issues and implemented them into C-MC33, a more topologically correct version of MC33 [33].

The focus of the study by Chen and Jin [36] was on utilization of a graphics processing unit (GPU) for parallel optimization of polygonized isosurfaces in the MC algorithm. Chen and Jin introduced a GPU-based approach to polygonize and optimize isosurface meshes for implicit surfaces. Specifically, Chen and Jin designed schemes to exploit the parallel features of the GPU hardware by optimising both the geometry (vertex position, vertex distribution, triangle shape, and triangle normal) and the topology (connectivity) aspects of a mesh. According to the authors, this method demonstrated improvement on the resultant mesh quality and acceleration of the isosurface extraction process as compared with CPU-based approach.

Athawale et al. [37] did not propose a modification of the MC algorithm but investigated the impact of data uncertainty on topology and geometry extraction in MC algorithms. They proposed an edge-crossing probability based approach to predict underlying isosurface topology for uncertain data. Athawale et al. pointed out that data uncertainty, characterized by probability distributions, could be propagated through the isosurface extraction process. They derived a probabilistic version of the midpoint decider that, according to the obtained results, resolved ambiguities that arised in identifying topological configurations. Athawale et al. designed a probabilistic techniques for handling uncertainty in cell configurations for isosurface topology etermination. They proposed vertex-based classification and edge-based classification methods to classify vertex signs. The obtained results demonstrated the advantage of nonlocal statistics approach for characterising data uncertainty over locally estimated parametric and non-parametric densities.

Most of the modified MC algorithms used a single 3D scalar field for rendering isosurfaces but did not take into account the characteristics of the scalar field itself. Ronghuan et al. [38] proposed a isosurface extraction method based on multi-resolution scalar field construction and seamless intersecting surface. Research by Ronghuan et al. aimed at improving the efficiency of isosurface extraction and rendering by reducing the isosurface extraction data while maintaining the effect of isosurface rendering. The multi-resolution scalar field was constructed to address the problems of ambiguity and gaps generated between different (low and high) resolution scalar fields of isosurfaces.

A splicing of contours between scalar fields with different resolutions was performed in order to remove the ambiguities and gaps. Thus, the contours of the high-resolution scalar field were kept uniform, and the extraction of low-resolution scalar fields was undertaken with unique triangular patches.

2.2.2 The dual contouring algorithm and its modifications

Dual contouring (DC) was first proposed by Ju et al. in 2002 [20]. Dual contouring is a feature-preserving isosurfacing method that extracts crack-free surfaces from both uniform and adaptive octree grids [20]. Dual contouring aims at reconstructing even sharp features of an object with the help of the gradient of the scalar field. Dual contouring requires that a scalar value and a gradient vector be assigned to each vertex of the octants. For this reason, this algorithm cannot be used for volume fraction data where only a scalar value is assigned to each octant. Dual contouring is a method used for extracting the surface boundary of an implicit volume. The method is *dual* in the sense that vertices generated by DC are topologically dual to faces in the MC algorithm. In dual contouring, a uniform grid is superimposed on the implicit volume. The grid cubes are represented as nodes in an octree data structure. The advantage of DC is that it can reproduce sharp features by inserting vertices anywhere inside the grid cube, as opposed to the marching cubes (MC) algorithm that can insert vertices only on the grid edges.

Greß and Klein[17] proposed the utilization of a k-d tree-based hierarchy for an implicit object representation. They asserted that the k-d tree-based hierarchy was superior to the octree in terms of adapting to the object surface. Greß and Klein obtained more compact implicit representations especially in case of thin object structures. They described a new isosurface extraction algorithm based on k-d-trees instead of octrees in the original dual contouring algorithm by Ju et al. [20]. The process resulted in generation of two-manifold meshes even for k-d trees with cells containing multiple surface components. In addition, a simplification framework was created for the surfaces represented by the k-d tree based on quadric error metrics. The framework allowed controlled topological simplification of the object.

Ju et al. developed a method for contouring a signed grid whose edges were tagged by Hermite data (i.e., exact intersection points and normals) [20]. This method did not require to explicitly identify and process features as in previous Hermite contouring methods. Using a numerically stable representation for QEFs, Ju et al. developed an octree-based contouring method. The DC method imposed no constraints on the octree and did not require crack patching.The important feature of the original DC algorithm by Ju et al. is that it adopted an octree as the data structure to adaptively represent extracted isosurfaces.

The original DC algorithm over the years underwent various modifications aiming to improve its performance. For example, Zhang et al. proposed a modified dual contouring algorithm DC with topology-preserving simplification [39]. The aim of the suggested modification was to preserve the disconnected surface components in cells during isosurface simplification. Zhang et al. represented isosurface components in a form of *enhanced cells*. In an enhanced cell, each surface component was represented by a vertex and its connectivity information. A topology-preserving vertex clustering algorithm was applied to build a vertex octree. An enhanced dual contouring algorithm was employed to extract error-bounded multi-resolution isosurfaces from the vertex octree while preserving the finest resolution isosurface topology. A connectivity-guided vertex clustering algorithm was used to simplify the isosurface components. After building a hierarchically clustered vertex octree, topology-preserved isosurfaces could be extracted under various error bounds by the enhanced dual contouring algorithm. Dual contouring with topology-preserving simplification using enhanced cell representation.

Schaefer et al. [40] modified their original DC algorithm in order to guarantee that the mesh generated was *manifold* even under adaptive simplification. They extended the original DC algorithm by complementing it with an octree-based topology-preserving vertex-clustering algorithm for adaptive contouring. The contoured surface generated by the extended method contained only manifold vertices and edges, preserved sharp features, and possessed better adaptivity than the original algorithm.

Zhang and Qian [41] developed a modification of the octree-based dual contouring (DC) method to construct surface and volumetric meshes for complicated domains. After considering all possible topology configurations, they developed an extension of the standard DC algorithm which, according to them, guaranteed the correct topology. The process of constructing surface and volumetric meshes included the following steps. First, one base mesh was generated using the original DC method. Then all the octree leaf cells were considered and categorized into 31 topology groups. In order to discriminate between these cells, the values of their face and body saddle points were computed based on a trilinear representation inside the cells. Then Zhang and Qian modified the base mesh and introduced more minimizer points within the same cell. With these minimizer points the mesh connectivities were updated to preserve the correct topology. This method was further extended to 3D tetrahedral mesh generation via an advancing front technique. A Laplacian smoothing technique was applied to improve the mesh quality; for tetrahedral mesh a combination of edge-contraction, smoothing and optimization was also applied. In a more recent work, Liang and Zhang [42] extended initial research by introducing the modification of the DC algorithm for adaptive triangular or tetrahedral mesh generation to guarantee a better angle range. The algorithm was based on a quadtree or octree structure, and could generate interior and exterior meshes with conformal boundary. The results demonstrated that the improved octree-based dual contouring method was ca-

pable of generating guaranteed-quality meshes.

Peixoto and de Moura [43] complemented the original DC algorithm with two discretization methods – *the non-compact dual simplification (NDS)* and *the sewing octree*. These discretization methods were developed to operate on polygonal surfaces specifically generated by the octree-based adaptive dual contouring algorithm. The NDS method focused on preserving the simplified topology of non-compact surfaces. However, this method was also applicable to compact surfaces if their polygonalization regions did not have any regions of non-compact surfaces. The sewing octree scheme provided a way of combining two or more octrees that shared faces or edges and contained portions of the surface polygonalized with dual contouring. These methods could be employed either independently or coupled by (1) dividing the original cube in two or more cubes, (2) carrying out the polygonalization, (3) simplifying these regions with NDS, if necessary, and (4) glueing the resulting surfaces with the sewing octree. This procedure, as authors stated, guaranteed that the resulting surface would have the same topology as the original surface.

Rashid et al. [44] suggested an improvement of the original DC method to produce watertight and two-manifold surface meshes. They observed that the original DC method produced only one vertex for each grid cube and, therefore, was unable to generate watertight and non-manifold meshes. The solution was to decompose an ambiguous grid cube into a maximum of 12 tetrahedral cells. In addition to that, Rashid et al. also introduced the polygon generation rules. The improved algorithm resulted in the production of watertight and two-manifold surface meshes.

Varadhan et al. [45] presented two algorithms for accurate polygonization of implicit surfaces from volumetric data, namely, feature-sensitive adaptive subdivision and isosurface reconstruction. Isosurface reconstruction used directed distances, i.e., distance along a direction, to perform an exact edge intersection test. This edge intersection test was used to detect intersections of the edge with a surface. This test was combined with the original dual contouring algorithm to obtain an improved reconstruction algorithm which Varadhan et al. called *extended dual contouring*. It was capable of reconstructing thin features while avoiding creation of additional handles. The algorithm tool into account the characteristics of the grid and considered complex edges. It enumerated all the intersections along the edges, separated them into components and reconstructed the isosurface locally within each cell.

2.2.3 Dual marching cubes

Dual marching cubes (DMC) is a combination of the MC and DC methods, and can be considered as a modification of both. However, due to its significant impact on the visualization discipline, it is considered separately from other MC and DC modifications. Schaefer and Warren [10] presented a method for contouring an implicit function using a grid topologically dual to structured grids such as octrees. First, this algorithm computed an octree from the input scalar field and then generated an appropriate dual grid based on the scalar field. After the generation of the dual grid, mesh vertices are computed in the same manner as in the original MC algorithm. Because the dual marching cubes algorithm extracted a surface from a grid structure whose cells were assigned values, it was applicable to volume fraction data on an octree. By aligning the vertices of the dual grid with the features of the implicit function, the proposed algorithm was able to reproduce thin features of the extracted surface without the excessive sub-division required by the original MC and DC methods. The DMC method led to a crack-free, adaptive polygonalization of the surface that reproduced sharp features.

However, one of the main issues with the DMC method by Schaefer and Warren was related to the topology of the extracted surfaces. A data structure other than the uniform grids had to be employed for efficient and accurate surface representations of the generated implicit models. The uniform grids could not provide grid sizes with enough space for extracting implicit models. Another limitation of the DMC method is the lack of sharp features from the environments generated by the previous implementations of the MC algorithms. This problem was attributed to improper contouring as large flat regions would often get tiled with small polygons resulting into the generated meshes.

In the same year, but a few months later than Schaefer and Warren [10], Nielson [46] proposed a similar algorithm coincidentally named *dual marching cubes*. The dual marching cubes algorithm presented by Nielson was, however, a different strategy to reconstruct an isosurface from the volume data. The intersection of the isosurface with the cell was approximated by a polygon on the cell faces. Nielson's work included the definition and computational algorithms for a new class of surfaces which were *dual* to the isosurface produced by the MC algorithm. These isosurfaces had the same separating properties as the MC surfaces but were comprised of quad patches eliminating the common negative aspect of poorly shaped triangles of the MC isosurfaces. Based upon the concept of *the dual operator*, Nielson proposed an iterative scheme for generating smooth separating surfaces for binary, enumerated volumes often produced by segmentation algorithms. Importantly, the DMC method by Nielson was not based on QEF or octrees. The primary distinction between the DMC algorithm by Schaefer and Warren [10] and the one by Nielson [46] lied in how the extracted vertex locations were determined. Schaefer and Warren defined the vertex locations as the minimizer of a QEF, whereas Nielson specified them based on a geometric function of the primal mesh geometry, such as the face centroid.

A series of further modifications of the two original DMC methods were carried out over the years. The DMC algorithm by Schaefer and Warren was modified by Kim et al. [47] by introducing an interpolation scheme on the octree volume fraction data. The algorithm was similar to the

original dual marching cubes except for the computation of the mesh vertex. The mesh vertices were linked together to create the surface mesh. An approximation based on the shape of the octants of a sphere was proposed to achieve an accurate vertex computation of the octree.

Greß and Klein [17] used a different data structure to represent implicit objects – the k-d tree. The k-d tree grid was built on the assumption that an appropriate grid partitioning criterion for subdividing splitting planes was determined. Construction of the grid took a top-down routine as follows. For every subdivision step, the active grid edges had to be detected and their related intersection information was computed. However, to achieve two-manifold and topologically correct representations, the subdivision scheme maintained a list of faces in the cell. This allowed faces contained in the list to be clipped against the splitting plane throughout each subdivision step, and these faces were handed over to the corresponding sub-cells after being clipped. After generating each vertex during the clipping process, a record was made of the axes corresponding to the planes clipped against them. Even though the k-d tree was selected in [17] to efficiently replace the octree, the study did not compare the efficiency of these structures across various simplification error thresholds. This gap in comparison motivated the work presented in this paper.

Grosso and Zint [48] suggested a modification of the Nielson's DMC algorithm. The proposed method aimed at reconstructing surfaces from volume data using a dual marching cubes approach without look-up tables (similar to the Nielson's method). The method generated quad-only meshes which were consistent across cell borders, i.e., they were manifold and watertight. Vertices were positioned exactly on the reconstructed surface leading to high accuracy. A half-edge data structure was used for storing the meshes for further processing. The method processed elements in parallel and, therefore, ran efficiently on GPU. Due to the transition between layers in volume data, meshes had numerous vertices with valence of three. Grosso and Zint used simplification patterns for eliminating quads containing these vertices reducing the number of elements and increasing quality.

He et al. [49] considered both DMC algorithms but opted for modification of Nielson's DMC algorithm by adding degrees of freedom to flexibly position each extracted vertex within its dual cell. This work proposed gradient-based mesh optimization of 3D surface mesh by representing it as the isosurface of a scalar field. According to He et al., the existing implementations were designed to extract meshes from fixed, known fields, and in the optimization setting; therefore, they lacked the degrees of freedom to represent high-quality feature-preserving meshes, or demonstrated numerical instabilities. He at al. proposed an isosurface representation aiming at optimization of an unknown mesh with respect to geometric, visual, or even physical objectives. The authors introduced additional parameters into the representation allowing for local adjustments to the extracted mesh geometry and connectivity. These parameters were updated along with the underlying scalar field via automatic differentiation when optimising for a downstream task. The extraction scheme presented extensions to optionally generate tetrahedral and hierarchically-adaptive meshes.

2.3 Literature review summary

A critical review of the marching cubes, dual contouring and dual marching cubes algorithms and their modifications prepared us to the implementation of an approach for evaluating the efficiency of the k-d trees and octrees in the simplification of isosurfaces in a dual contouring algorithm for a specific or given error. The problem addressed in this research is the lack of comparative tools for the developers to guide the selection of adaptive data structures (i.e., k-d trees and octrees) during implementations.

Table 1 and Table 2 presents a summary of the key findings of the reviewed literature given in a chronological order.

3 Proposed approach to implementation of the two algorithms

The k-d tree- and octree-based manifold dual contouring algorithms for isosurface extraction were implemented with the objective of comparing their efficiency. The implementation of both algorithms was performed in two phases: construction of the underlying rectilinear grid (Phase 1) and isosurface extraction of implicit models through error-controlled simplification (Phase 2). Within each phase there were several sub-phases. The construction of the underlying rectilinear grid included the following sub-phases: formulation of QEFs (Sub-phase 1.1), generation of the scalar field (Sub-phase 1.2), composition of vertex data with Hermite data (Sub-phase 1.3), preparation of the manifold criterion topology preserving scheme for contouring the scalar field (Sub-phase 1.4), and construction of the general mesh in accordance with the topology preserving scheme (Sub-phase 1.5). The isosurface extraction of implicit models through error-controlled simplification included the following sub-phases: generation of the scalar field of the implicit surface representation (Sub-phase 2.1), preparation of the computation of topology preserving vertex clustering using QEFs (Sub-phase 2.2), and QEF controlled isosurface simplification on implicit k-d tree- or octree-based adaptive grids (Sub-phase 2.3). A detailed flowchart representing each phase and sub-phase in the implementation of the k-d tree algorithm is depicted in Figure 3.

The flowchart for the original octree algorithm is not included in this paper as it was implemented following prior research by Schaefer et al. [40]. The implementation involved the standard process of building an octree hierarchy

Table 1: The summary of the reviewed literature (in the chronological order) – Part I

Techniques	Key Results	Identified Gaps
Associative *k*-d trees [18]	Multi-dimensional binary search trees	No comparison with octrees
Octree generation [19]	Efficient volume decomposition	Limited to octree structures
Marching cubes [9]	High-resolution 3D surface construction	Inconsistencies in visual depictions
Dual contouring of Hermite data [20]	Crack-free, gradient-based surfaces	Requires scalar and gradient data
Adaptive isosurface *k*-d Trees [17]	Efficient representation of thin objects	Lack of comparative analysis with octrees
Trilinear interpolation adaptation [25]	Minimized voxel effects	Voxel shape impacts outcome
Feature-sensitive subdivision and isosurface reconstruction [45]	Reconstructed thin features, avoided additional handles	Limited to specific grid structures
Dual marching cubes (based on QEFs) [10]	Preserved sharp features, adaptive contouring	Topological inaccuracies
Dual marching cubes (based on geometric centroid) [46]	Smoother, quad-dominant meshes without look-up tables	Did not use QEF minimization, limited empirical comparison
Optimizing topological and combinatorial complexity [27]	Efficient isosurface extraction	Limited focus on topology
Regularized marching cubes [11]	Improved mesh topology, maximum vertex valence of six	Did not address volumetric data complexity
Improved marching cubes [12]	Addressed wrong surface production	Hole generation in original method
Topology preserving marching cubes [13]	Preserved topology on face-centered cubic grid	Limited to specific grid structures
Improved marching cubes [14]	Smoother isosurfaces, reduced triangle patches	No comprehensive performance evaluation
Simplified marching cubes [15]	Efficient discretization for deposition/ablation simulations	Lower accuracy compared to original MC algorithm
Efficient improved marching cubes [16]	Increased efficiency and topological accuracy	Focused on specific efficiency improvements

and performing dual contouring with manifold vertex clustering to ensure topological correctness and detail preservation. The readers are advised to refer to the original publication [40] for a more detailed description of the algorithmic steps for the key phases. Both implemented manifold dual contouring algorithms used rectilinear grids to represent the extraction of implicit models. Implementation by Greß and Klein [17] employed the combination of uniform grids with the topology preserving scheme for manifold extracted isosurfaces with sharp features. This implementation resulted in the generation of too many polygons. This was not desirable, particularly during extraction of relatively flat surfaces. Numerous polygons would make it difficult to render large models made of flat surfaces.

3.1 Phase 1: Construction of the underlying rectilinear grid

Phase 1 of the implementation of the two algorithms involved constructing the underlying rectilinear grid. The advantages of rectilinear grids are a negligible memory footprint and readily support smooth data reconstruction, though with reduced geometric flexibility [50]. Construction of the rectilinear grid was required for working with implicit surface representations of extracted isosurfaces.

Sub-phase 1.1 involved the formulation of the QEFs. This study adopted the QEF formulation from Ju et al. [20] as given by Equations (1) and (3) in Subsection 2.1.2. These equations allowed us to determine the corresponding vertex locations and the error associated with those locations. In a situation where the accuracy of positioning the vertices is more important than the basic representation of the QEF, the relationship between the orthogonal matrix Q of the QR

Table 2: The summary of the reviewed literature (in the chronological order) – Part II

Techniques	Key Results	Identified Gaps
Corrected marching cubes 33 [33]	Addressed topological issues in MC33	Hole generation in original method
GPU-based polygonization and optimization [36]	Improved mesh quality and extraction speed	Limited to GPU-based methods
Edge-crossing probability for uncertainty handling [37]	Resolved ambiguities in topology	Complexity in handling uncertainty
Gradient-based mesh optimization [49]	Flexible vertex positioning, high-quality meshes	Numerical instabilities, limited flexibility
Improved dual contouring [44]	Watertight and two-manifold meshes	Complexity in implementation
Interpolation scheme on octree volume fraction data [47]	Accurate vertex computation, improved mesh quality	Limited to octree structures
Multi-resolution isosurface extraction [38]	Efficient data reduction while maintaining rendering	Difficulty in accurately determining the isosurface location, holes that may occur when merging or transitioning between isosurfaces
Parallel dual marching cubes without look-up tables [48]	High accuracy, efficient on GPU	Transition issues between volume data layers

decomposition can be used to satisfy the relation $Q^TQ = I$, leading to the error equation $E[x]$ to be determined through Equation (15):

$$\begin{aligned}(Ax-b)^T(Ax-b) &= (Ax-b)^TQ^TQ(Ax-b)\\ &= (QAx-Qb)^T(QAx-Qb)\\ &= (A\hat{x}-\hat{b})^T(A\hat{x}-\hat{b})+r^2.\end{aligned} \quad (15)$$

In this form, the evaluation of $E[x]$ becomes attainable by computing the squared vector product of the vector $\hat{A}x-b$, before adding r^2. For a non-singular $\hat{A}$ case, minimization of x can be calculated through solving the equation $\hat{A}\hat{x} = \hat{b}$ by performing back substitution. $\hat{A}$ is calculated from the noisy normals tending to always be coplanar. The same matrix $\hat{A}$ becomes approximately singular. Minimizing the $\hat{x}$ value results in setting in the far outside of the outlining cube. Solving this problem means that the singular value decomposition (SVD) of $\hat{A}$ must be computed, and its pseudo-inverse must be formed by cutting down its trivial singular values:

$$\hat{A} = U\Sigma V^T, \quad (16)$$

where Σ is a diagonal matrix with singular values. The pseudo-inverse $\hat{A}^+$ of $\hat{A}$ is given by Equation (17)

$$\hat{A}^+ = V\Sigma^+U^T, \quad (17)$$

where Σ^+ contains reciprocals of non-zero singular values. Using SVD, the minimizer x is computed as follows:

$$x = V\Sigma^+U^Tb. \quad (18)$$

Sub-phase 1.2 involved the generation of the scalar field. A scalar field refers to a function ϕ which specifies a constant or scalar value to every point in three-dimensions. A scalar field is a function associating a single number to every point in a space. The focus of this implementation was on isosurface extraction of implicit surface representations, and the scalar fields were generated from implicit functions.

The formulation of QEFs and generation of the scalar field in Sub-phases 1.1 and 1.2 provided the inputs for Subphase 1.3, namely, composition of vertex data with Hermite data. Inputs included information regarding the positioning of the vertices and their related attributes. A scalar field represents every angular point in the three-dimensional space referred to as a vertex. Each of these vertices contains attributes such as a vertex position and normal. This information was required in the extraction of the isosurface for the algorithm to determine what type of surface was extracted at that particular vertex. The approach used to process vertices tagged with Hermite data was similar to the one introduced in [20].

Sub-phase 1.4 involved the preparation of the manifold criterion topology preserving scheme. For the implementation involving isosurface simplification using octrees, the manifold vertex clustering technique was adopted from Schaefer et al. [40]. For the implementation using *k*-d trees in the manifold vertex clustering technique by Schaefer et al., the octree isosurface simplification was replaced with the *k*-d tree adaptive isosurface simplification. For both implementations the simplification error thresholds were the same.

Sub-phase 1.5 pertained to the construction of the gen-

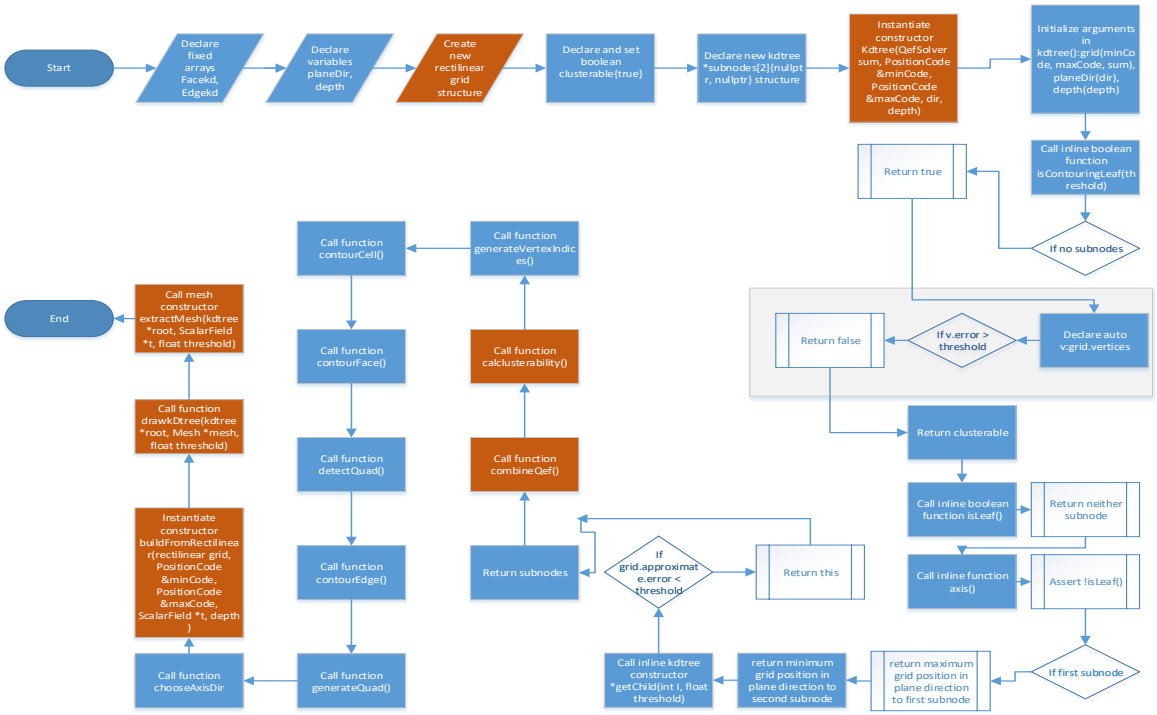

Figure 3: Flowchart for the implementation of the k-d tree algorithm

eral mesh according to the topology preserving scheme. The previous sub-phases 1.1-1.4 supplied the information required for the construction of the embedded polygonal mesh as part of the underlying rectilinear grid. The first step in this sub-phase was to draw an axis aligned bounding box (AABB) enclosing the whole mesh including all vertices and triangles of the polygons. Each vertex contained attributes to be used by the algorithm for positioning and clustering of that particular vertex. The attributes comprised a vertex's index, position and normal in relation with the mesh and the overall rectilinear grid. The algorithm added vertices into the grid according to the associated scalar field representation of the surface. This was performed through QEF based computation of each vertex location with respect to edges of the AABB. The flat normals were generated for each vertex according to its position of each. After that, to ensure that the mesh was topologically correct for watertight and two-manifold and surfaces, the vertices were clustered through the manifold vertex clustering scheme.

3.2 Phase 2: Isosurface extraction through error controlled simplification

The first two sub-phases of isosurface extraction through error controlled simplification were generation of the scalar field of the implicit surface representation (Sub-phase 2.1), and preparation of the computation of topology preserving vertex clustering using QEFs (Sub-phase 2.2). Those standard sub-phases were adopted from Schaefer et al. [40]. The next sub-phase (2.3) involved QEF-controlled isosurface simplification on implicit k-d tree or octree based adaptive grids. Our main focus was on utilization of the k-d tree for simplification of the rectilinear grid surface representation in order to adaptively contour implicit surfaces of any dimensions. The simplification scheme was adopted from [40] with the exception of a k-d tree being utilized instead of an octree. The algorithm performed traversal of the k-d tree in a top-down manner. For every leafless k-d tree cell, two offspring nodes were considered. Two offspring nodes had faces internal to their associated parent cell along each Euclidean axis. Similar to the previous phase (Sub-phase 1.4), vertices were clustered with consideration to the manifold criterion. The difference was that this time vertices were topologically linked together by their edges twofold to the internal faces of the parent cell. The vertex edges were identified through recursion. In each set, vertices were clustered together through a combination of their QEFs and minimising error functions to evaluate the new vertex's location and its associated error. If the error for evaluating and locating the new vertex was less than the given threshold, then this vertex was marked as collapsible. The subdivision process involved storing a list of faces within the cell and of each individual subdivision step. The faces within the cell were clipped to the partitioning plane in accordance with the manifold criterion. Each of these clipped faces was moved into the associated sub-cell. Information obtained from the clipping process included the orientation of the planes the faces were clipped against. This information was used in detection of vertices associated with the inter-

section points found on the edges of the grid.

4 Methodology

The *k*-d tree and octree algorithms were implemented on a Windows 10 system with an Intel Core i5 CPU, NVIDIA GeForce MX150 GPU, and 4 GB RAM. The *k*-d tree construction involved recursive subdivision along one dimension, using QEFs for vertex placement and clustering based on manifold criteria. The octree construction used recursive space subdivision into eight octants, applying QEF similarly and ensuring crack-free, manifold surfaces through dual contouring. Both algorithms employed predefined error thresholds to control the level of detail. Performance metrics like build time, extraction time, and triangle count were used for comparative analysis.

To ensure that the evaluation tool would work for implicit surface representations of extracted isosurfaces, implicit models were prepared using an implicit function with a scalar field quantity as an input. An implicit function is a function defined for differentiation of functions containing the variables, which cannot be easily expressed in the form of $y = f(x)$ [51]. An implicit function is a mathematical function taking in more than one related variable at a time.

The implicit function was programmed in C++ with its inputs being the scalar field quantity and the depth of the field. The time (seconds) for extracting an implicit surface representation from the generated implicit model was determined for each of the two dual manifold contouring algorithms, the *k*-d tree- and octree-based, as part of the comparative evaluation process. The extraction of an implicit surface representation was performed in the CPU using the OpenGL API. The rendering of graphics on the screen was carried out in the GPU, also using OpenGL API.

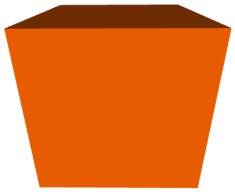

Figure 4: A cube

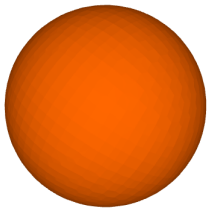

Figure 5: A sphere

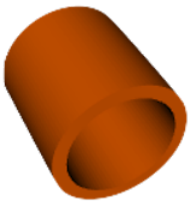

Figure 6: A cylinder

5 Results

5.1 Implicit surface representations for selected objects

Both compared algorithms were run on a variety of implicit models depicted in Figures 4 (a cube), 5 (a sphere), 6 (a cylinder), 7 (a torus), and 8 (a random surface).

The implicit surface representations from the extracted isosurface meshes displayed the solid objects commonly studied or modeled, for example, in computer aided design (CAD) and constructive solid geometry (CSG). The solid volumes were shaded through to reflect the nature of the represented real-life physical objects. The assumption was that the materials depicted in the surface representations were solid in nature, although in reality this might not always be the case. The implicit surface representations considered in this research were computer-generated solid objects extracted from models of implicit functions, and not from volumetric datasets of actual physical objects. The

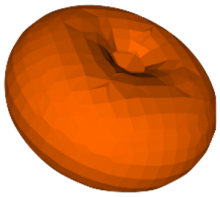

Figure 7: A torus

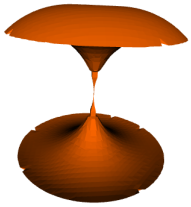

Figure 8: A random surface

manifold dual contouring algorithm was used to extract the embedded mesh and simplify the surface representation of the underlying rectilinear grid for a given error threshold.

Figure 9 shows a wire frame of axis-aligned bounding box (AABB) and its enclosed sphere. This diagram shows how clustered vertices were connected into polygons of the resulting extracted isosurface.

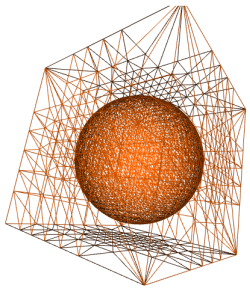

Figure 9: A sphere enclosed inside an AABB

5.2 Results of evaluation of the two algorithms

The two algorithms for isosurface extraction, (k-d tree and octree-based, were evaluated and compared using three parameters, namely, *visualization*, *performance*, and *efficiency*.

In regard to *visualization*, no noticeable difference was observed in the visual graphics extracted by the k-d tree- and octree-based algorithms. Both algorithms utilized the manifold vertex clustering scheme for dual contouring of isosurfaces. The images in Figures 4-9 could have been produced by any of the two algorithms. It could, however, be expected that there might be a difference in the rendering of large models in favor of the octree-based algorithm. An octree-based algorithm would generate more visual details by using a larger number of triangles in its simplification process as compared to the k-d tree algorithm as shown in Tables 3 and 4 (triangle count) for various simplification error thresholds (denoted as Simpl. Err. Threshold). The visual difference also could have been observed if one of the two algorithms was based on an entirely different isosurface extraction approach, e.g., the original marching cubes algorithm.

The performance of each algorithm was measured using two metrics, (1) the time to build the adaptive hierarchy of the data structure (Build Time) and (2) the time taken to extract the implicit surface representations (Extract Time). The values of these metrics for various simplification error thresholds (Simpl. Err. Threshold) are shown in Tables 3 and 4. The comparison between the two algorithms indicated that for building and extraction of an identical implicit surface representation under the same simplification error threshold the k-d three-based algorithm performed better than the octree-based algorithm. For example, in the case of a cylinder model with the simplification error threshold

Table 3: Results for the *k*-d tree-based manifold dual contouring algorithm

Object Modeled	Simpl. Err. Threshold	Build Time (sec)	Extract Time (sec)	Triangle Count
Cylinder	$1e^{-3}$	0.3349	0.015	2266
	$1e^{-5}$	0.015	0.029	2244
	$1e^{-7}$	0.347	0.015	2280
	$1e^{-9}$	0.351	0.015	2336
Sphere	$1e^{-3}$	0.326	0.036	9190
	$1e^{-5}$	0.333	0.035	9206
	$1e^{-7}$	0.348	0.037	9206
	$1e^{-9}$	0.337	0.036	9206
Torus	$1e^{-3}$	0.095	0.01	2080
	$1e^{-5}$	0.09	0.01	2080
	$1e^{-7}$	0.088	0.01	2080
	$1e^{-9}$	0.089	0.01	2080
Random surface	$1e^{-3}$	0.284	0.033	7626
	$1e^{-5}$	0.284	0.033	7550
	$1e^{-7}$	0.278	0.033	7558
	$1e^{-9}$	0.279	0.033	7558

Table 4: Results for the octree-based manifold dual contouring algorithm

Object Modeled	Simpl. Err. Threshold	Build Time (sec)	Extract Time (sec)	Triangle Count
Cylinder	$1e^{-3}$	1.545	0.028	14303
	$1e^{-5}$	1.588	0.029	14192
	$1e^{-7}$	1.598	0.028	14304
	$1e^{-9}$	1.583	0.028	14400
Sphere	$1e^{-3}$	1.557	0.02	9472
	$1e^{-5}$	1.568	0.02	9464
	$1e^{-7}$	1.558	0.019	9464
	$1e^{-9}$	1.542	0.019	9464
Torus	$1e^{-3}$	1.693	0.007	2080
	$1e^{-5}$	1.712	0.007	2080
	$1e^{-7}$	1.708	0.006	2080
	$1e^{-9}$	1.924	0.007	2080
Random surface	$1e^{-3}$	1.78	0.025	7928
	$1e^{-5}$	1.75	0.02	7928
	$1e^{-7}$	1.754	0.019	7928
	$1e^{-9}$	1.741	0.018	7928

of $1e^{-3}$, a *k*-d tree-based manifold dual contouring algorithm achieved the 78% reduction in build time and 46% reduction in extract time as compared to the octree-based algorithm. The average reduction of the build and extract times, across all modeled objects and all simplification error thresholds, was 83.4% and 61.2%, respectively, in favor of the *k*-d tree-based algorithm.

The evaluation of *efficiency* involved determining the numbers of triangles for both algorithms under various simplification error thresholds and for various modeled objects. There is a direct link between efficiency and the triangle counts as the less is the triangle count, the higher is the simplification efficiency. The average improvement in simplification efficiency across all modeled objects and simplification error thresholds for the *k*-d tree-based algorithm was calculated at 17.7% as compared to the octree-based algorithm. In case of the cylinder model, across all simplification error thresholds the improvement in efficiency was the highest (84.5%) for the *k*-d tree-based algorithm. However, for the torus model the triangle count across all simplification error thresholds for simplifying the representation of the torus model was the same for both algorithms. This can be explained by the model size because a smaller model with fewer triangles could not be simplified any further by either algorithm. The evaluation results demonstrated that the octree-based algorithm is more effective in preserving the original geometric details of an implicit model. On average, the octree-based algorithm generated a higher number of rectangles, making it the more suitable choice for applications requiring detailed geometric accuracy.

6 Discussion

The comparative analysis conducted in this study revealed several insights into the performance and efficiency of the *k*-d tree- and octree-based manifold dual contouring algorithms in relation to the state of the art. This discussion compares the results of our work with those listed in Table 1, focusing on performance metrics, visual and efficiency outcomes, potential reasons for observed differences, and novel contributions and advancements over the similar existing work.

6.1 Differences in performance metrics

The performance metrics evaluated in this study included build time, extract time, and triangle count. The *k*-d tree-based algorithm demonstrated superior performance in terms of build and extract times compared to the octree-based algorithm. Specifically, the *k*-d tree-based algorithm achieved an average reduction of 83.4% in build time and 61.2% in extract time across all modeled objects and simplification error thresholds. In contrast, previous studies, for example, by Greß and Klein [17], did not provide a direct comparative analysis between *k*-d trees and octrees, thus lacking a performance benchmark for *k*-d tree efficiency.

6.2 Visual and efficiency outcomes

In terms of visual outcomes, no significant differences were observed between the *k*-d tree- and octree-based algorithms when evaluated visually. Both algorithms utilized the manifold vertex clustering scheme, ensuring high-quality isosurface representations. However, the octree-based algorithm excelled in preserving geometric details, producing an average of 20% more triangles than the *k*-d tree-based algorithm. This aligns with findings by Schaefer and Warren [10] and Kim et al. [47], who emphasized the superior visual detail retention of octree-based methods.

Regarding efficiency, the *k*-d tree-based algorithm showed an improvement in simplification efficiency, with an average reduction in triangle count by 17.7% compared to the octree-based algorithm. This efficiency gain is particularly relevant for large implicit models, where reducing geometric complexity is crucial. Previous studies, such as those by Jin et al. [12] and Vignoles et al. [15], highlighted improvements in specific aspects of the marching cubes algorithm but did not achieve the overall efficiency gains demonstrated by the *k*-d tree-based approach in our research.

6.3 Potential reasons for observed differences

The observed differences in performance and efficiency can be attributed to the inherent characteristics of *k*-d trees and octrees. *K*-d trees partition the space adaptively along one dimension at a time, which allows for more flexible and efficient handling of large datasets with varying geometric complexities. This adaptive partitioning reduces the overall geometric complexity more effectively than the simultaneous three-dimensional subdivision used by octrees.

Additionally, the *k*-d tree's ability to minimize build and extract times stems from its binary partitioning scheme, which simplifies the computational process. In contrast, the octree's approach to maintaining high visual detail results in higher triangle counts and longer processing times.

6.4 Study contributions

The contributions of this study to the body of knowledge in computer science, specifically in the areas of computer graphics and geometric modeling, are as follows.

First, unlike previous studies that primarily focused on individual algorithm improvements, this study provides a comprehensive empirical comparison between *k*-d tree- and octree-based manifold dual contouring algorithms. For example, Schaefer et al. [40] focused on octree-based dual contouring methods without juxtaposing them against *k*-d tree methodologies, while others whose research is listed in Table 1 have only improved aspects of individual algorithms.

Second, by evaluating build time, extract time, and triangle count across various simplification error thresholds, this study presents a holistic view of algorithm efficiency. This multi-metric evaluation approach advances the understanding of performance trade-offs in isosurface extraction.

Third, the findings will assist researchers and practitioners in selecting suitable adaptive data structures for different application scenarios. The demonstrated efficiency of the *k*-d tree-based algorithm for large implicit models offers a practical alternative to octree-based methods.

Fourth, the *k*-d tree-based algorithm's superior performance in reducing geometric complexity without compromising visual quality represents a significant advancement in isosurface extraction techniques.

Lastly, the comparative analysis undertaken in this study demonstrated the strengths and weaknesses of *k*-d tree- and octree-based manifold dual contouring algorithms, providing insights for future research and practical implementations in the field of isosurface extraction.

7 Conclusions, limitations and future work

7.1 Conclusions

This paper presented an evaluation of the two manifold dual contouring algorithms using *k*-d trees and octrees across various simplification error thresholds. The first manifold dual contouring octree-based algorithm was adopted from Schaefer et al. [40]. The second manifold dual contouring algorithm was adopted from the same work by Schaefer et al. [40] but a *k*-d tree was employed instead of the octree

for simplification as proposed by Greß and Klein in [17]. A rectilinear grid was developed as a prerequisite to capture the initial representation of the underlying mesh, and the QEF was formulated, resulting in the construction of the general mesh according to the topology preserving scheme. The isosurface extraction of implicit models was carried out through the error controlled *k*-d tree and octree simplification.

The following conclusions were drawn from the evaluation of visualization, performance and efficiency of the compared algorithms. Scaling implicit environments can impact the performance of a manifold dual contouring algorithm due to the processing of excessive geometric details. The *k*-d tree-based algorithm was found to be more suitable, as this adaptive data structure significantly reduces the amount of geometry in large implicit models. For a scenario requiring more of the geometric detail of the simplified implicit model to be retained, the octree would be considered more suitable because of this data structure's ability to preserve geometric detail as demonstrated by the higher triangle count of the simplification results.

The performance of each algorithm was measured using two metrics, (1) the time taken to build the adaptive hierarchy of the data structure and (2) the time taken to extract the implicit surface representations. The comparison of the two algorithms demonstrated that for building and extraction of a implicit surface representation under the same simplification error threshold the *k*-d tree-based algorithm performed better than the octree-based algorithm. The average reduction of the build and extract times, across all modeled objects and all simplification error thresholds, was 83.4% and 61.2%, respectively, in favor of the *k*-d tree-based algorithm. The evaluation of efficiency for the two algorithms involved determining the numbers of triangles for both algorithms under various simplification error thresholds for various modeled objects. The average improvement in simplification efficiency across all modeled objects and simplification error thresholds by the *k*-d tree-based algorithm was calculated at 17.7% as compared to the octree-based algorithm. However, if geometry had to be preserved as much as possible in order to capture the original geometric details of an implicit model, the octree would be more applicable as, on average, it generated more rectangles.

Research results presented in this paper could be useful for both researchers and practitioners in the area of visualization facing a choice of a suitable adaptive data structure (i.e., a *k*-d tree or an octree) for better simplification results in implementations of the manifold dual contouring algorithm.

7.2 Study limitations

The study had several limitations that need to be addressed in future research. Firstly, the research scope was restricted to a specific set of implicit models (cube, sphere, cylinder, torus, and random surface). Including additional models with varying complexities could provide a broader evaluation. Secondly, experiments were conducted on a single PC configuration due to resource constraints, which might affect performance results on different hardware setups. Thirdly, the focus was on specific implementations of *k*-d tree- and octree-based algorithms; however, exploring other variations might offer new insights. Fourthly, as this research used predefined simplification error thresholds, varying these could affect performance metrics and visual quality. Fifthly, further experiments are needed to validate the scalability of both algorithms with larger datasets and more complex implicit models. Lastly, the study did not include visual comparisons of the extracted isosurfaces under different simplification thresholds, which would help illustrate the practical implications of the quantitative results.

7.3 Future work

Future research could include a more diverse set of implicit models to enhance the generalizability of the findings. Evaluating the performance on distributed computing systems or different hardware configurations could provide deeper insights into the scalability and efficiency of the algorithms. Investigating other variations or hybrid approaches of *k*-d tree and octree-based algorithms might uncover new performance benefits. Implementing adaptive simplification thresholds based on the model's complexity and application requirements could further optimize the algorithms' performance and efficiency. Applying these algorithms to real-world datasets, such as medical imaging or geological surveys, could demonstrate their practical utility and reveal additional areas for improvement. Including visual comparisons of the extracted isosurfaces from both algorithms under different simplification thresholds will help in illustrating the practical implications of the quantitative results presented.

References

[1] C. Hansen, C. Johnson, The visualization handbook, Cambridge, MA: Academic Press, 2004.

[2] W. Wu, L. Ye, Research on real scene robot 3d visualization of historical architectural heritage based on big data objects, Informatica (Slovenia) 48 (2024) 93–102. doi:10.31449/inf.v48i8.5776.

[3] D. Cheng, Animation vr scene stitching modeling based on genetic algorithm, Informatica (Slovenia) 48 (2024) 83–96. doi:10.31449/inf.v48i5.5364.

[4] S. H. Al-Dhaimesh, N. Taib, Review paper: investigation of augmented reality – bim benefits in design process in the aec industry, Informatica (Slovenia) 47 (2023) 111–126. doi:10.31449/inf.v47i5.4671.

[5] H. Fan, B. Goyal, K. Z. Ghafoor, Computer-aided architectural design optimization based on bim tech-

nology, Informatica (Slovenia) 46 (2022) 323–332. doi:10.31449/inf.v46i3.3935.

[6] I. Bankman, Handbook of medical image processing and analysis, Cambridge, MA: Academic Press, 2009.

[7] J. O'Brien, G. Turk, Modelling with implicit surfaces that interpolate, ACM Transactions on Graphics 21 (2002) 855–873. doi:10.1145/571647.571650.

[8] M. Edmunds, R. Laramee, G. Chen, N. Max, E. Zhang, C. Ware, Surface-based flow visualization, Computers Graphics 36 (2012) 974–990. doi:10.1016/j.cag.2012.07.006.

[9] W. Lorensen, H. Cline, Marching cubes: A high resolution 3d surface construction algorithm, ACM SIGGRAPH Computer Graphics 21 (1987) 163–169.

[10] S. Schaefer, J. Warren, Dual marching cubes: Primal contouring of dual grids, in: Proceedings of the Pacific Conference on Computer Graphics and Applications, 2004, pp. 70–76. doi:10.1109/pccga.2004.1348336.

[11] S.-W. Lee, H.-Y. Jung, R. Prost, A. Senot, Regularized marching cubes mesh, in: Proceedings of the IEEE International Conference on Image Processing, Vol. 3, 2005, pp. 788–791. doi:10.1109/icip.2005.1530510.

[12] J. Jin, Q. Wang, Y. Shen, J. Hao, An improved marching cubes method for surface reconstruction of volume data, in: Proceedings of the World Congress on Intelligent Control and Automation, Vol. 2, 2006, pp. 10454–10457. doi:10.1109/wcica.2006.1714052.

[13] R. Strand, P. Stelldinger, Topology preserving marching cubes-like algorithms on the face-centered cubic grid, in: Proceedings of the 14th International Conference on Image Analysis and Processing, 2007, pp. 781–788. doi:10.1109/iciap.2007.4362871.

[14] Z. Xu, C. Xiao, X. Xu, An improved marching cubes algorithm based on edge contraction, in: Proceedings of the International Conference on Signal Processing Proceedings, 2010. doi:10.1109/icosp.2010.5655719.

[15] G. Vignoles, M. Donias, C. Mulat, C. Germain, J.-F. Delesse, Simplified marching cubes: an efficient discretization scheme for simulations of deposition/ablation in complex media, Computational Materials Science 50 (2011) 893–902. doi:10.1016/j.commatsci.2010.10.027.

[16] Q. Du, J. Zhao, L. Shi, L. Wang, Efficient improved marching cubes algorithm, in: Proceedings of the 2nd International Conference on Computer Science and Network Technology, 2012, pp. 416–419. doi:10.1109/iccsnt.2012.6525967.

[17] A. Greß, R. Klein, Efficient representation and extraction of 2-manifold isosurfaces using kd-trees, in: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications, 2003, pp. 370–397. doi:10.1109/pccga.2003.1238278.

[18] J. Bentley, Multidimensional binary search trees used for associative searching, Communications of the ACM 18 (1975) 509–517. doi:10.1145/361002.361007.

[19] D. Meagher, Octree generation, analysis and manipulation, Ph.D. thesis, Rensselaar Polytechnic Institute, Troy New York (1982).

[20] T. Ju, F. Losasso, S. Schaefer, J. Warren, Dual contouring of hermite data, ACM Transactions on Graphics 21 (2002) 339–346. doi:10.1145/566654.566586.

[21] J.-D. Boissonnat, S. Oudot, Provably good sampling and meshing of surfaces, Graphical Models 67 (2005) 405–451. doi:10.1016/j.gmod.2005.01.004.

[22] M. Garland, P. Heckbert, Surface simplification using quadric error metrics, in: Proceedings of the ACM SIGGRAPH Conference on Computer Graphics, 1997, p. 209–216. doi:10.1145/258734.258849.

[23] Y. Livnat, C. Hansen, C. Johnson, Isosurface extraction for large-scale data sets, in: Proceedings of the Conference on Data Visualization: The State of the Art, 2003, pp. 77–94. doi:10.1007/978-1-4615-1177-9_6.

[24] T. Newman, H. Yi, A survey of the marching cubes algorithm, Computers Graphics 30 (2006) 854–879. doi:10.1016/j.cag.2006.07.021.

[25] D. Rajon, W. Bolch, Marching cube algorithm: review and trilinear interpolation adaptation for image-based dosimetric models, Computerized Medical Imaging and Graphics 27 (2003) 411–435. doi:10.1016/s0895-6111(03)00032-6.

[26] C. Maple, Geometric design and space planning using the marching squares, in: Proceedings of the International Conference on Geometric Modeling and Graphics, 2003, pp. 90–95. doi:10.1109/gmag.2003.1219671.

[27] C. Andujar, P. Brunet, A. Chica Calaf, J. Rossignac, A. Vinacua, Optimizing the topological and combinatorial complexity of isosurfaces, Computer-Aided Design 37 (2005) 847–857. doi:10.1016/j.cad.2004.09.013.

[28] X. Renbo, L. Weijun, Y. Wang, A robust and topological correct marching cube algorithm without look-up

table, in: Proceedings of the 5th International Conference on Computer and Information Technology, 2005, pp. 565–569. doi:10.1109/cit.2005.44.

[29] N. Chrisochoides, D. Nave, Simultaneous mesh generation and partitioning for delaunay meshes, Mathematics and Computers in Simulation 54 (1999) 321–339. doi:10.1016/s0378-4754(00)00192-0.

[30] S. H. Cui, J. Liu, Simplified patterns for extracting the isosurfaces of solid objects, Image and Vision Computing 26 (2008) 339–346. doi:10.1016/j.imavis.2007.02.003.

[31] T. Etiene, C. Scheidegger, L. Nonato, R. Kirby, C. Silva, Verifiable visualization for isosurface extraction, IEEE Transactions on Visualization and Computer Graphics 15 (2009) 1227–1234. doi:10.1109/tvcg.2009.194.

[32] Q. Xiaoqing, W. Davis, An extended cuberille model for identification and display of 3d objects from 3d gray value data, in: Proceedings of the Conference on Graphics Interface, 1992, pp. 70–77.

[33] L. Custodio, T. Etiene, S. Pesco, C. Silva, Practical considerations on marching cubes 33 topological correctness, Computers Graphics 37 (2013) 840–850. doi:10.1016/j.cag.2013.04.004.

[34] E. Tcherniaev, Marching cubes 33: Construction of topologically correct isosurfaces, Tech. rep., Institute for High Energy Physics (1996).

[35] T. Lewiner, H. Lopes, A. Vieira, G. Tavares, Efficient implementation of marching cubes' cases with topological guarantees, Journal of Graphics Tools 8 (2012) 1–15. doi:10.1080/10867651.2003.10487582.

[36] J. Chen, X. Jin, Gpu-based polygonization and optimization for implicit surfaces, The Visual Computer 31 (2014) 119–130. doi:10.1007/s00371-014-0924-7.

[37] T. Athawale, E. Sakhaee, E. Alireza, Isosurface visualization of data with nonparametric models for uncertainty, IEEE Transactions on Visualization and Computer Graphics 22 (2015) 777–786. doi:10.1109/tvcg.2015.2467958.

[38] Y. Ronghuan, X. Wei, W. Lingda, H. Hongxing, Research on multi-resolution isosurface extraction method for 3d scalar field, in: Proceedings of the 2nd IEEE International Conference on Data Science in Cyberspace, 2017, pp. 359–362. doi:10.1109/DSC.2017.79.

[39] N. Zhang, W. Hong, A. Kaufman, Dual contouring with topology-preserving simplification using enhanced cell representation, in: Proceedings of the IEEE Visualization Conference, 2004, pp. 505–512. doi:10.1109/visual.2004.27.

[40] S. Schaefer, T. Ju, J. Warren, Manifold dual contouring, IEEE Transactions on Visualization and Computer Graphics 13 (2007) 610–619. doi:10.1109/tvcg.2007.1012.

[41] Y. Zhang, J. Qian, Dual contouring for domains with topology ambiguity, Computer Methods in Applied Mechanics and Engineering s 217–220 (2012) 34–45. doi:10.1016/j.cma.2012.01.004.

[42] X. Liang, Y. Zhang, An octree-based dual contouring method for triangular and tetrahedral mesh generation with guaranteed angle range, Engineering with Computers 30 (2014) 211–222. doi:10.1007/s00366-013-0328-8.

[43] A. Peixoto, C. Moura, Topology preserving algorithms for implicit surfaces simplifying and sewing, in: Proceedings of the International Conference on Computational Science and Its Applications, 2014, pp. 352–367. doi:10.1007/978-3-319-09129-7_27.

[44] T. Rashid, S. Sharmin, M. Audette, Watertight and 2-manifold surface meshes using dual contouring with tetrahedral decomposition of grid cubes, Procedia Engineering 163 (2016) 136–148. doi:10.1016/j.proeng.2016.11.037.

[45] G. Varadhan, S. Krishnan, Y. Kim, D. Manocha, Feature-sensitive subdivision and isosurface reconstruction, in: Proceedings of the IEEE Visualization Conference, 2003, pp. 99–106. doi:10.1109/visual.2003.1250360.

[46] G. Nielson, Dual marching cubes, in: Proceedings of the IEEE Visualization Conference, 2004, pp. 489–496. doi:10.1109/visual.2004.28.

[47] S. Kim, Y. Ohtake, Y. Nagai, H. Suzuki, A novel interpolation scheme for dual marching cubes on octree volume fraction data, Computers and Graphics 66 (2017) 169–178. doi:10.1016/j.cag.2017.05.021.

[48] R. Grosso, D. Zint, A parallel dual marching cubes approach to quad only surface reconstruction, The Visual Computer 38 (2022) 1301–1316. doi:10.1007/s00371-021-02139-w.

[49] Y. He, M. Mirzargar, S. Hudson, M. Kirby, R. Whitaker, An uncertainty visualization technique using possibility theory: Possibilistic marching cubes, International Journal for Uncertainty Quantification 5 (2015) 433–451. doi:10.1615/int.j.uncertaintyquantification.2015013730.

[50] D. El-Rushaidat, R. Yeh, X. Tricoche, Accurate parallel reconstruction of unstructured datasets on rectilinear grids, Journal of Visualization 24 (2021) 787–806. doi:10.1007/s12650-020-00740-0.

[51] J. Stewart, Calculus early transcendentals, Cengage Learning, 2015.