Computer Science • 17 (4) 2016 http://dx.doi.org/10.7494/csci.2016.17.4.541

Tomasz Jurczyk
Barbara Głut

# TREE STRUCTURES FOR ADAPTIVE CONTROL SPACE IN 3D MESHING

**Abstract** *The article presents a comparison of several octree- and kd-tree-based structures used for the construction of control space in the process of anisotropic mesh generation and adaptation. The adaptive control space utilized by the authors supervises the construction of meshes by providing the required metric information regarding the desired shape and size of elements of the mesh at each point of the modeled domain. Comparative tests of these auxiliary structures were carried out based on different versions of the tree structures with respect to computational and memory complexity as well as the quality of the generated mesh. Analysis of the results shows that kd-trees (not present in the meshing literature in this role) offer good performance and may become a reasonable alternative to octree structures.*

**Keywords** control space, kd-tree, octree, anisotropic metric, mesh generation and adaptation

**Citation** Computer Science 17 (4) 2016: 541–560

## 1. Introduction

The generation and adaptation of meshes for 3D models play an important role in a number of areas, such as numerical simulations, computational geometry, computer graphics, the visualization of objects, surface reconstruction, and many others.

The mesh should meet a number of requirements related to the geometry of the object and its application. Regarding the geometry of the object, it is necessary to take into account the curvature of the boundaries, sharp edges, proximity to other elements of the object, boundaries between the different sub-domains, or the specific requirements of the user. In many cases, it is preferable to create a mesh consisting of anisotropic elements. In order to generate a mesh fulfilling the specified quality criteria, it is necessary to assign the desired size and element shape to each point of the modeled object. A common way to achieve such a result is to use the Riemannian metric, changing it locally in various sub-areas of a three-dimensional object [1, 2, 4, 11]. The sources of the metric may be of a various nature, relating to the characteristics of the domain and specific application of the mesh. The task of the generator is to create an isotropic mesh in a prescribed Riemannian space, which will be appropriately adapted (and sometimes anisotropic) in Euclidean space.

The process of meshing often requires the creation of a structure in which information can be stored and updated, and from which it is possible to retrieve information about the metric at any given point in the domain. In general, prescribing the size and shape of the elements is closely integrated with the algorithm of mesh generation and adaptation. With respect to code efficiency, the key issues are the access time of retrieving this data and the memory required for storage of this structure. The structure design, together with the algorithms implementing its crucial functionality, may also affect the quality of the generated or adapted mesh.

In the literature, propositions of different forms of this structure can be found (as well as its various names: sizing map, cartesian mesh, background mesh), with the octree structure being the most-typical choice. Pirzadeh[16] introduced a uniform Cartesian mesh, where the sizing field is smoothed globally. A major drawback here is the memory needed to store such a structure. Aubry et al. [3] and Deister et al. [6] locally adapted the field in an adaptive Cartesian mesh. Another is to use the quadtree/octree structure (currently, the most-common approach that reduces the need for storage). Various approaches may be applied for the refinement of these structures, which take into consideration the geometry of the domain or gradient-size function [13, 15, 18]. There are also methods in which the creation of an octree is performed using a medial axis [17]. In the area of mesh adaptation for computation, a background mesh identical to the mesh from the previous calculation step is also quite often used [14]. While there are plenty of articles relating to methods of metric application to generate meshes, relatively little attention is paid to issues like the impact of the form of this auxiliary structure on generation time and the quality of the produced mesh.

### 1.1. Metric and control space in meshing

In an automated process of mesh generation and adaptation, it is a common approach to take advantage of a metric defined in any point of the domain. This metric is applied as an operator determining the desired size and shape of the elements. The metric itself may come from different sources (both continuous and discrete) and have different representations [10]. The metric data gathered from various sources should be properly adjusted in each point of the domain. For this purpose, several elementary operations on metric have been developed (i.e., interpolation, comparison, intersection and gradation control). More detailed information about the technical details of these operations can be found in [9]. In the created mesh generator, an auxiliary structure covering the modeled domain was introduced – an adaptive control space (ACS) – responsible for gathering and adjusting metric data as well as supervising the construction and adaptation of the mesh [8]. ACS stores the metric information in discrete points. At the step of gathering and processing metric data from different sources, the ACS was equipped with a number of operations, such as initializing all control vertices based on available metric sources, setting a continuous metric source in the whole domain, inserting a discrete metric source at some point, and adjusting metric gradation according to the prescribed maximum metric gradation ratio. In the process of mesh generation, the most-elementary task of ACS is to return a metric tensor at any point of the modeled domain by means of metric interpolation from the discrete control vertices of ACS. The applied interpolation procedures depend on the nature of the control space and metric representation.

Two families of such control structures are presented in this article – those based on octree and kd-tree. The main task described in this article is preparing discrete stucture $\mathcal{K}(P)$ approximating an analytical (continuous) sizing field (isotropic or anisotropic) $\mathcal{F}(P)$ at each point $P$ in the domain $\mathcal{D}$ of $\mathcal{F}$. Such a conversion has a number of advantages:

- storing data in a unified discrete structure allows us to further adjust the sizing information; for example, smoothing the sizing field or setting an appropriate anisotropy ratio,
- when the cost of computing the original sizing information is too high, conversion to a computationally efficient structure may be advantageous,
- information stored in a discrete way can be combined with other sizing sources, both discrete and continuous.

The price of conversion is an additional computational cost associated with the creation of a proper discrete structure approximating the given sizing field with the prescribed precision.

## 2. Kd-tree and octree structures

The structure of a discrete control space is based on a three-dimensional tree. In order to be memory efficient, a distinction is made between the internal nodes and the leaves

of the tree. The proposed kd-tree structures were created based on a general form of kd-tree[5] with modifications introduced by the authors (with respect to the kd-tree application as a control space structure for mesh generation and adaptation).

The kd-tree structure is created with an adaptation procedure that recursively performs the following steps (starting from an initial tree containing a single node – the root of the tree):

- evaluate the approximation error $\delta_n(\mathcal{F}, \mathcal{K})$ for the given leaf;
- if the approximation error is higher than the given threshold and the maximum depth of the tree has not yet been reached:
  - select a dividing hyperplane,
  - split the leaf with the selected hyperplane,
  - assign metric values for the newly created two leaves (using $\mathcal{F}$),
  - call this procedure recursively for both leaves,

For octree structures, the adaptation is similar (with the exception of splitting, since nodes in an octree are always split into eight children of equal dimensions).

## 2.1. Error estimation for tree nodes

For each kd-tree (or octree) node where the approximation error needs to be evaluated, the maximum difference between the approximation and function values for a regular grid of points within the cell is computed:

$$\delta_n = \max_{P_i \in S_p} \delta_{\mathcal{M}}(\mathcal{F}(P_i), \mathcal{K}(P_i)) \ , \tag{1}$$

where $S_p$ is a set of points within a node arranged in a regular grid of size $N_s \times N_s \times N_s$ and $\delta_{\mathcal{M}}$ is a measure of metric non-conformity[11].

## 2.2. Method of node splitting

The nodes of a kd-tree (where adaptation is required) are split along the selected axis and a point defining a splitting hyperplane. The following methods of splitting a hyperplane were implemented and tested:

1. **Longest Axis** ($s_L$). The node is split in the middle along the longest axis.
2. **Maximum Gradient of the Approximated Function** ($s_G$). The node box is split into $N_s$ segments along the axes. Let us introduce split boxes numbering: $s_{i,j,k}$ where $i, j, k \in \{1, 2, \ldots, N_s\}$. Then, we will consider a gradient to be the difference of function values between two faces of the kdtree node box $s_{i,j,k}$; i.e.,

$$\nabla^x_{i,j,k} = \delta_{\mathcal{M}}(\mathcal{F}(x_m + \frac{1}{2}d_x, y_m, z_m), \mathcal{F}(x_m - \frac{1}{2}d_x, y_m, z_m)) \tag{2}$$

$$\nabla^y_{j,i,k} = \delta_{\mathcal{M}}(\mathcal{F}(x_m, y_m + \frac{1}{2}d_y, z_m), \mathcal{F}(x_m, y_m - \frac{1}{2}d_y, z_m)) \tag{3}$$

$$\nabla^z_{k,i,j} = \delta_{\mathcal{M}}(\mathcal{F}(x_m, y_m, z_m + \frac{1}{2}d_z), \mathcal{F}(x_m, y_m, z_m - \frac{1}{2}d_z)) \tag{4}$$

where $d_x, d_y, d_z$ are the sizes of the box along the x, y, and z axes, respectively, and $x_m, y_m, z_m$ are the coordinates of the box's $s_{i,j,k}$ center.
The axis with the largest gradient is then chosen to be split along. The point of the split is the middle $(x_m, y_m, z_m)$ of the node box with the largest gradient.

3. **Maximum Sum of Gradient of the Approximated Function** ($s_S$). Let us introduce the numbering of split boxes and the gradient as in the method $s_G$. For each dimension ($d \in \{x, y, z\}$), the sums of gradients are computed as

$$\Sigma_d = \sum_{i=1}^{N_s} \sum_{j=1}^{N_s} \sum_{k=1}^{N_s} \nabla_{i,j,k}^d \ . \tag{5}$$

The axis $d^*$ with the largest sum of gradients is then chosen to be split along. In order to determine the point of a split, the value of $\tau$ is obtained

$$\tau = \min \left\{ t \mid \sum_{i=1}^{t} \sum_{j=1}^{N_s} \sum_{k=1}^{N_s} \nabla_{i,j,k}^{d^*} > \frac{1}{2} \Sigma_{d^*} \right\} \ . \tag{6}$$

The node is split at the position

$$\begin{cases} x_m \text{ of the box } s_{\tau,j,k} & \text{if } d^* = x, \\ y_m \text{ of the box } s_{j,\tau,k} & \text{if } d^* = y, \\ z_m \text{ of the box } s_{j,k,\tau} & \text{if } d^* = z, \end{cases} \tag{7}$$

where $j, k \in \{1, 2, \ldots, N_s\}$.

## 2.3. Kd-tree-L

### 2.3.1. Structure description

The kd-tree structure $\mathcal{K}_L$ approximates the source function $\mathcal{F}$ with values stored exclusively in its leaves (one value per leaf). The function value is constant within each leaf.

The main kd-tree structure (Fig. 1) stores the bounding box and a reference to the tree root. Internal nodes store the selection of a split axis, coordinates of a split point, and two references to child nodes. Leaves of this kd-tree store only single-function values (in the form of a metric tensor).

### 2.3.2. Adaptation procedure

The adaptation procedure follows the general scheme described earlier. The recursive procedure takes as parameters source function $\mathcal{F}$, approximation threshold $\delta_\tau$, the current tree level, and the bounding box of the current tree node. Instead of storing the bounding box locally for each tree node, the bounding box from the main kd-tree structure is passed as an initial value; after each adaptation split, this bounding box is accordingly adjusted, and the updated versions are passed to both child nodes for further adaptation. The function values in the child nodes are calculated from source function $\mathcal{F}$ in the middle point of each of these nodes.

```
struct KdTreeL
{
  Box3d box;
  KdNode *top;
};
```

```
struct KdNode
{
   bool isLeaf;
   union KdData
   {
     struct KdInternal
     {
        Axis axis;
        double coordinate ;
        KdNode *children[2];
     } split;
     struct KdLeaf
     {
       T value;
     } leaf;
   } data;
 }
```

**Figure 1.** Structure declaration for kd-tree $\mathcal{K}_L$.

### 2.3.3. Retrieving data

In order to retrieve value $\mathcal{K}_L(P)$ of point $P$ within the bounding box of the tree, the containing leaf is found using information about split axes and coordinates in the internal nodes. After the containing leaf is found, the value stored within it is returned directly.

## 2.4. Kd-tree-V

### 2.4.1. Structure description

This version of kd-tree structure ($\mathcal{K}_V$) approximating source function $\mathcal{F}$ has values stored in the vertices of the tree nodes. The function value within each leaf is calculated from the vertices using linear shape functions.

The main kd-tree structure (Fig. 2) stores the bounding box and a reference to the tree root – plus a container of function values referenced in the vertices of the kd-tree. Some vertices are shared between several tree nodes (neighboring and/or descending); thus, actual values are stored in a single container on a main level, and the nodes of a tree store only references.

Internal nodes store the same data as $\mathcal{K}_L$ (split axis, split coordinate, and two references to child nodes). Leaves of this kd-tree store an array of eight references to function values corresponding to its eight vertices.

### 2.4.2. Adaptation procedure

The adaptation procedure is similar to $\mathcal{K}_L$. The main difference is the initialization of the new nodes, where arrays of references have to be appropriately prepared for both child nodes. The child nodes inherit half of their vertices from the parent node; the other half is common for both child nodes and has to be calculated from source function $\mathcal{F}$ at the intersections of the parent node and the splitting hyperplane.

```
struct KdTreeV
{
  Box3d box;
  KdNode *top;
  List<T> values;
};
```

```
struct KdNode
{
  bool isLeaf;
  union KdData
  {
    struct KdInternal
    {
       Axis axis;
       double coordinate ;
       KdNode *children[2];
    } split;
    struct KdLeaf
    {
      T* values[8];
    } leaf;
  } data;
}
```

**Figure 2.** Structure declaration for kd-tree $\mathcal{K}_V$.

### 2.4.3. Retrieving data

In order to retrieve value $\mathcal{K}_V(P)$ of point $P$, the containing leaf is found as in $\mathcal{K}_L$. However, during descent through the internal nodes, the local bounding box needs to be updated (as in the adaptation procedure). Then, after the containing leaf is found, local coordinates for shape functions are calculated, and the returned result is computed as a weighted sum of values from the vertices of this leaf.

## 2.5. Kd-tree-Li

### 2.5.1. Structure description

Another proposed version of kd-tree structure ($\mathcal{K}_{Li}$) approximating source function $\mathcal{F}$ has values stored in the leaves only (as in $\mathcal{K}_L$). However, the value within each leaf is calculated using not only the value stored therein but also values stored in the neighboring leaves.

The main kd-tree structure (Fig. 3) stores the bounding box and a reference to the tree root. Internal nodes store the same data as $\mathcal{K}_L$ (split axis, split coordinate, and two references to child nodes). In the leaves of $\mathcal{K}_{Li}$, some additional data is stored besides the single scalar or metric value (for the sake of adaptation and/or the retrieving procedure). The array of references to neighboring nodes is necessary for computing results in the retrieving procedure. Other data (level, local bounding box, and the flag isAdapted) are required for the adaptation procedure due to its breadth-first characteristics.

### 2.5.2. Adaptation procedure

The adaptation procedure for $\mathcal{K}_{Li}$ is a bit more complicated because, in order to estimate the approximation error in any leaf, its neighbors should already be adapted, since they influence the function value retrieved from this leaf. In order to overcome

this obstacle, the adaptation scheme utilizes a breadth-first approach. Before deciding whether to split the current leaf, an additional step is taken, ensuring that all existing neighbors of this leaf either have finished their adaptation or their adaptation has reached at least the same level as the current leaf. If any neighboring leaf does not fulfill these conditions, it is adapted up to the level where these requirements are met. Implementing these modifications requires some additional data to be stored in the leaf nodes of the tree: local bounding box, local level, and a flag marking the completion of adaptation for a given leaf.

```
struct KdTreeLi
{
  Box3d box;
  KdNode *top;
};
```

```
struct KdNode
{
  bool isLeaf;
  union KdData
  {
    struct KdInternal
    {
       Axis axis;
       double coordinate;
       KdNode *children[2];
    } split;
    struct KdLeaf
    {
      T value;
      KdNode* neighbours[6];
      int level;
      Box3d box;
      isAdapted;
    } leaf
  } data;
}
```

**Figure 3.** Structure declaration for kd-tree $\mathcal{K}_{Li}$.

The adaptation procedure follows the general scheme described earlier. The recursive procedure takes as parameters source function $\mathcal{F}$, approximation threshold $\delta_\tau$, the current tree level, and the bounding box of the current tree node. The function values in the child nodes are calculated from the source function at the middle point of each of these nodes.

### 2.5.3. Retrieving data

In order to retrieve value $\mathcal{K}_{Li}(P)$ of point $P$, the containing leaf is found, along with all neighboring leaves closest to point $P$ (starting from the information about neighboring nodes stored in the containing leaf).

Let $v_c$ be the value stored in the containing leaf and $v_{x0}, v_{x1}, v_{y0}, v_{y1}, v_{z0}, v_{z1}$ be the values stored in the neighboring leaves. If any neighboring node is missing, $v_c$ is used instead.

The result value is calculated using 27-node quadratic hexahedral shape functions for local coordinates of point $P$ within the containing leaf with the following vertex values:

- the inner vertex of the node box has value $v_c$,
- the mid-face vertices of the node box have values set as the average of $v_c$ and the value stored in the leaf adjacent through the given face (if any neighboring node is missing, $v_c$ is used in its place),
- the mid-edge vertices of the node box have values set as the average of the mid-face vertices from the two adjacent faces,
- the corner vertices of the node box have values set as the average of the mid-face vertices from the three adjacent faces.

## 2.6. Octree structures

Several versions of octree structures were also implemented following the typical approaches used in the area of meshing [4, 12, 18] as reference for comparison with the proposed kd-tree structures.

Although efficient, an octree structure for control space in meshing also has some drawbacks:

- Cartesian system alignment of the tree, which may degrade the approximation quality for non-axis-aligned models and reduce its capability to capture other symmetry (e.g., polar or spherical) of mesh density,
- global characteristics, which can spatially combine metric information for topologically independent (or distant) parts of the model, often resulting in the unnecessary refinement of the mesh size.

These problems are also valid for the presented versions of kd-tree structures, and their correction requires separate research.

### 2.6.1. Octree-L

$\mathcal{K}o_L$ stores values in the leaves only. Each internal node contains coordinates of the middle node and an array of eight references to the child nodes. Leaves have only single values of metric tensors.

### 2.6.2. Octree-LB

The $\mathcal{K}o_{LB}$ structure also stores metric values only in the leaves. However, in order to obtain better regularity, a local balancing of nodes is introduced (with a maximum depth difference between children set to 1). Internal nodes contain the same data as in the $\mathcal{K}o_L$ structure. For the leaves, additional data is introduced besides the value of the metric tensor: current tree level, coordinates of node box, and references to neighboring nodes — in order to facilitate maintaining the local balancing condition.

### 2.6.3. Octree-V

$\mathcal{K}o_V$ is an octree structure without balancing, where metric values are stored in the vertices of the leaves and the metric within an octree leaf is calculated using linear shape functions. Internal nodes are similar to $\mathcal{K}o_L$. Leaves contain the current tree level, references to neighboring nodes, and references to leaf vertices – which are stored in a separate container.

## 3. Tests

The tests were designed to measure the quality of the created trees, performance of the creation, and the quality of the meshes produced by using a tree-based control space.

### 3.1. Test functions

All test functions were defined to produce a metric (isotropic of anistropic) defined by

$$\mathcal{M}_s = \begin{pmatrix} h_1^{-2}(x,y,z) & 0 & 0 \\ 0 & h_2^{-2}(x,y,z) & 0 \\ 0 & 0 & h_3^{-2}(x,y,z) \end{pmatrix} \tag{8}$$

where $h_1, h_2, h_3$ are the lengths of elements along the main directions.

The tests were performed for five different test functions. Functions $\mathcal{F}_1$, $\mathcal{F}_2$, and $\mathcal{F}3$ were selected as formulas emulating typical definitions of metric sources [9, 17] defined in some vicinity of a point ($\mathcal{F}_1$) and mesh spacing correlated with the distance to some linear boundary ($\mathcal{F}_2$) or discrete model features ($\mathcal{F}_3$), and function $\mathcal{F}_4$ and $\mathcal{F}_5$ were based on the numerical examples presented in [7]:

1. **3DGaussian**

$$\mathcal{F}_1 : \mathcal{D}_1 \ni (x,y,z) \to h = e^{\frac{x^2+y^2+z^2}{2 \cdot 0.1^2}} \tag{9}$$

   where $\mathcal{D}_1 = [-1,1] \times [-1,1] \times [-1,1]$ and $h_1 = h_2 = h_3 = h$,

2. **Squared distance to line**

$$\mathcal{F}_2 : \mathcal{D}_1 \ni (x,y,z) \to h = \max(d_L^2, \epsilon) \tag{10}$$

   where $d_L$ is the distance to the reference line passing through points $(-1,-1,-0.5)$ and $(0.5,1,0)$, $h_1 = h_2 = h_3 = h$ and $\epsilon = 10^{-10}$.

3. **Linear distance from 3 points**

$$\mathcal{F}_3 : \mathcal{D}_1 \ni (x,y,z) \to h = \max(\overline{d}_P, \epsilon) \tag{11}$$

   where $\overline{d}_P$ is the mean of distances from points $P_1 = (0.5, 0.5, 0.5)$, $P_2 = (0.5, 0.5, -0.5)$ and $P_3 = (0,0,0)$, and $h_1 = h_2 = h_3 = h$.

4. **Test-iso**

$$\mathcal{F}_4 : \mathcal{D}_2 \ni (x,y,z) \to h = \begin{cases} 1 - 19y/40 & \text{if } y \in [0,2] \\ 20^{(2y-9)/5} & \text{if } y \in (2,4.5] \\ 5^{(9-2y)/5} & \text{if } y \in (4.5,7] \\ \frac{1}{5} + \frac{4}{5}\left(\frac{y-7}{2}\right)^4 & \text{if } y \in (7,9] \end{cases} \tag{12}$$

where $\mathcal{D}_2 = [0,9] \times [0,9] \times [0,9]$ and $h_1 = h_2 = h_3 = h$,

5. **Test-aniso**

$$\begin{aligned} \mathcal{F}_5 : \quad & \mathcal{D}_3 \ni (x,y,z) \to \\ & h_1 = \begin{cases} 1 - 19x/40 & \text{if } x \in [0,2] \\ 20^{(2x-7)/3} & \text{if } x \in (2,3.5] \\ 5^{(7-2x)/3} & \text{if } x \in (3.5,5] \\ \frac{1}{5} + \frac{4}{5}\left(\frac{x-5}{2}\right)^4 & \text{if } x \in (5,7] \end{cases} \\ & h_2 = \begin{cases} 1 - 19y/40 & \text{if } y \in [0,2] \\ 20^{(2y-9)/5} & \text{if } y \in (2,4.5] \\ 5^{(9-2y)/5} & \text{if } y \in (4.5,7] \\ \frac{1}{5} + \frac{4}{5}\left(\frac{y-7}{2}\right)^4 & \text{if } y \in (7,9] \end{cases} \\ & h_3 = \begin{cases} 1 - 19z/40 & \text{if } z \in [0,2] \\ 20^{(2z-11)/7} & \text{if } z \in (2,5.5] \\ 5^{(11-2z)/7} & \text{if } z \in (5.5,9] \\ \frac{1}{5} + \frac{4}{5}\left(\frac{z-9}{2}\right)^4 & \text{if } z \in (9,11] \end{cases} \end{aligned} \tag{13}$$

where $\mathcal{D}_3 = [0,7] \times [0,9] \times [0,11]$.

## 3.2. Test Design

The building process of all of the presented tree structures was analyzed for each of the defined functions. Each function was tested with a number of accuracies (i.e., $\{4.0, 2.0, 1.0, 0.5, 0.3, 0.1, 0.08, 0.06\}$), meaning the maximal difference between the kd-tree's output value and the actual output of the function (error). Three methods of splitting the node boxes were tested: longest axis, maximum gradient, and maximum sum of gradients. For each case, the following quantities were measured:

- average access time ($\bar{t}_a$) – a uniform grid of points in $\mathcal{D}$ was created; for each point, the metric value was retrieved from the tree structure, then the average value was calculated,
- creation time ($t_c$) – total time required for adaptation of the tree structure to given function $\mathcal{F}$,
- approximation error ($\delta_D$) – maximum value of difference $\delta_{\mathcal{M}}$ between the kd-trees /octree's approximation and the actual value of function $\mathcal{F}$ (checked for a uniform grid of points in the $\mathcal{D}$ of $\mathcal{F}$,
- tree size – number of tree nodes, number of metric values stored in the tree, and the total memory usage of the tree structure ($m_u$),
- tree balancing – maximum depth of the tree and maximum local balance.

Additionally, for test functions $\mathcal{F}_4$ and $\mathcal{F}_5$, each created control structure was used to generate a tetrahedral mesh with the test-tree structure fulfilling the role of control space. In order to evaluate the quality of the created meshes, the quality criteria for edge lengths and size and shape of the mesh elements were calculated – based on the reference control space (using base function $\mathcal{F}$ directly). The created meshes were compared using the following criteria:

- size of the mesh – represented as number of tetrahedra in the mesh (NT);
- length of edges (in metric space):
  $\overline{L}$ – mean value,
  $L_\sigma$ – standard deviation,
  $L_R$ – number of edges with length sufficiently close to optimal (for an ideal mesh, each edge should have a metric length equal to 1), where $L_R$ denotes the ratio of number of edges with metric lengths in the range of $[0.8, 1.25]$ to the total number of edges in the mesh;
- quality of elements – metric non-conformity coefficient $\delta_{\mathcal{M}}$ was used to evaluate both the size and shape of the elements in the mesh:
  $\overline{\delta}_{\mathcal{M}}$ – mean value,
  $\delta^{\sigma}_{\mathcal{M}}$ – standard deviation,
  $\delta^{R}_{\mathcal{M}}$ – number of elements with the value of the quality coefficient sufficiently close to optimal (for an ideal mesh, all elements should have the value of the non-conformity coefficient equal to zero), where $\delta^{R}_{\mathcal{M}}$ denotes the ratio of number of elements with $\delta_{\mathcal{M}} \in [0, 2]$ to the total number of elements in the mesh.

## 4. Analysis of results

A precise evaluation and presentation of the results is difficult, due to the high number of parameters affecting the particular results and the large number of performed tests. Because of this, only selected results – the most important – are presented. The analysis was carried out with an emphasis on the time and memory efficiency of various versions of the kd-tree structures as well as the accuracy of approximation and its impact on the quality of the generated meshes.[1]

### 4.1. Time and memory efficiency

As can be seen in Tables 1, 2, 3, 4, 5, and 6, the total size of the trees depends on approximated function $\mathcal{F}$ and the given approximation accuracy threshold $\delta_\tau$.

For $\mathcal{F}_1$, the created tree structures have the largest sizes; in this case, the splitting method has no significant influence. For $\mathcal{F}_4$ and $\mathcal{F}_5$, the effect of selecting the splitting method is clearly visible. Method $s_L$ produces structures with sizes considerably different than the other techniques (for all tested kd-tree versions). For functions $\mathcal{F}_4$

[1]The tests were performed using an Intel Core i7-3520M 2.9 GHz computer with 16 GB memory.

and $\mathcal{F}_5$, it gives much larger structures; but, for $\mathcal{F}_2$ and $\mathcal{F}_3$, the structures created by splitting with $s_L$ are actually smaller than for the other splitting methods. Clearly, the selection of a splitting method should take into account the nature of variation of the metric in the given domain. The sizes of octree structures were, in general, larger than the kd-trees (especially for smaller approximation accuracy thresholds).

**Table 1**

Selected results for tree structures created for function $\mathcal{F}_1$.

| | | tree size ($m_u$) [kB] | | | access time ($\bar{t}_a$) [$\mu s$] | | |
|---|---|---|---|---|---|---|---|
| tree | split | $\delta_\tau = 2$ | $\delta_\tau = 0.5$ | $\delta_\tau = 0.06$ | $\delta_\tau = 2$ | $\delta_\tau = 0.5$ | $\delta_\tau = 0.06$ |
| $\mathcal{K}_L$ | $s_L$ | 348.3 | 2337.1 | 50132.3 | 31.8 | 40.4 | 69.5 |
| | $s_G$ | 296.1 | 2271.7 | 29683.7 | 34.7 | 44.5 | 65.7 |
| | $s_S$ | 237.5 | 1780.1 | 39054.6 | 32.7 | 42.0 | 67.4 |
| $\mathcal{K}_V$ | $s_L$ | 131.5 | 399.2 | 1871.8 | 63.9 | 66.2 | 71.6 |
| | $s_G$ | 100.0 | 299.1 | 2204.9 | 70.6 | 75.0 | 83.3 |
| | $s_S$ | 95.4 | 285.0 | 2563.9 | 69.4 | 73.2 | 81.5 |
| $\mathcal{K}_{Li}$ | $s_L$ | 932.8 | 4267.2 | 76370.8 | 326.4 | 338.0 | 384.0 |
| | $s_G$ | 900.5 | 3438.8 | 52316.6 | 343.3 | 356.1 | 403.7 |
| | $s_S$ | 604.2 | 2840.6 | 48540.8 | 327.3 | 341.0 | 378.5 |
| $\mathcal{K}o_L$ | | 714.9 | 4500.9 | 115446.9 | 18.5 | 21.7 | 36.5 |
| $\mathcal{K}o_{LB}$ | | 1251.0 | 7877.9 | 202037.3 | 18.6 | 21.8 | 37.5 |
| $\mathcal{K}o_V$ | | 813.6 | 11874.2 | 139598.2 | 84.3 | 91.4 | 113.4 |

**Table 2**

Selected results for tree structures created for function $\mathcal{F}_2$.

| | | tree size ($m_u$) [kB] | | | access time ($\bar{t}_a$) [$\mu s$] | | |
|---|---|---|---|---|---|---|---|
| tree | split | $\delta_\tau = 2$ | $\delta_\tau = 0.5$ | $\delta_\tau = 0.06$ | $\delta_\tau = 2$ | $\delta_\tau = 0.5$ | $\delta_\tau = 0.06$ |
| $\mathcal{K}_L$ | $s_L$ | 333.7 | 956.8 | 6209.3 | 21.6 | 26.9 | 40.9 |
| | $s_G$ | 544.0 | 1502.1 | 7420.0 | 22.2 | 29.0 | 43.1 |
| | $s_S$ | 605.4 | 1469.2 | 7595.6 | 20.9 | 25.9 | 39.5 |
| $\mathcal{K}_V$ | $s_L$ | 455.3 | 690.6 | 1625.4 | 57.6 | 59.1 | 77.4 |
| | $s_G$ | 987.9 | 1841.3 | 4511.3 | 60.9 | 64.9 | 72.2 |
| | $s_S$ | 1143.7 | 1935.5 | 4985.8 | 59.3 | 62.1 | 67.7 |
| $\mathcal{K}_{Li}$ | $s_L$ | 791.2 | 1865.8 | 10064.9 | 299.7 | 308.8 | 331.9 |
| | $s_G$ | 1541.3 | 2932.1 | 11560.9 | 306.0 | 344.3 | 350.8 |
| | $s_S$ | 1774.3 | 2627.9 | 10510.7 | 299.8 | 308.1 | 333.3 |
| $\mathcal{K}o_L$ | | 735.9 | 2095.7 | 13413.9 | 15.1 | 17.2 | 22.2 |
| $\mathcal{K}o_{LB}$ | | 1423.0 | 3671.3 | 23474.3 | 15.5 | 17.4 | 23.0 |
| $\mathcal{K}o_V$ | | 1051.6 | 3260.7 | 22490.4 | 77.4 | 80.4 | 90.8 |

Access time is more advantageous for structures where no interpolation of a metric is required ($\mathcal{K}_L$, $\mathcal{K}o_L$, $\mathcal{K}o_{BL}$). The highest values of access time were noted for $\mathcal{K}_{Li}$, where the most-complex interpolation scheme was used. Despite the fact that the tree sizes for octree structures are, in general, larger than in the case of kd-trees, the time cost of retrieving the results from leaf-based octree structures (avoiding interpolation of metrics) is still quite advantageous.

**Table 3**

Selected results for tree structures created for function $\mathcal{F}_3$.

| | | tree size ($m_u$) [kB] | | | access time ($\bar{t}_a$) [$\mu s$] | | |
|---|---|---|---|---|---|---|---|
| tree | split | $\delta_\tau = 2$ | $\delta_\tau = 0.5$ | $\delta_\tau = 0.06$ | $\delta_\tau = 2$ | $\delta_\tau = 0.5$ | $\delta_\tau = 0.06$ |
| $\mathcal{K}_L$ | $s_L$ | 0.2 | 0.9 | 30.0 | 11.1 | 14.0 | 24.4 |
| | $s_G$ | 0.2 | 1.0 | 16.0 | 11.4 | 15.9 | 25.1 |
| | $s_S$ | 0.2 | 1.3 | 17.5 | 11.0 | 16.2 | 24.8 |
| $\mathcal{K}_V$ | $s_L$ | 0.9 | 2.8 | 10.7 | 47.7 | 54.1 | 56.7 |
| | $s_G$ | 1.2 | 4.5 | 31.1 | 52.9 | 57.6 | 66.8 |
| | $s_S$ | 1.2 | 4.2 | 29.8 | 51.0 | 56.8 | 64.2 |
| $\mathcal{K}_{Li}$ | $s_L$ | 0.6 | 4.0 | 47.0 | 273.8 | 284.3 | 301.1 |
| | $s_G$ | 0.6 | 2.0 | 28.1 | 277.3 | 285.7 | 308.7 |
| | $s_S$ | 0.6 | 4.0 | 31.2 | 276.6 | 287.2 | 307.2 |
| $\mathcal{K}o_L$ | | 0.9 | 3.9 | 54.9 | 12.3 | 13.0 | 16.1 |
| $\mathcal{K}o_{LB}$ | | 1.5 | 6.8 | 96.0 | 12.4 | 13.2 | 16.1 |
| $\mathcal{K}o_V$ | | 2.5 | 2.5 | 215.9 | 64.5 | 65.0 | 72.6 |

**Table 4**

Selected results for tree structures created for function $\mathcal{F}_4$.

| | | tree size ($m_u$) [kB] | | | access time ($\bar{t}_a$) [$\mu s$] | | |
|---|---|---|---|---|---|---|---|
| tree | split | $\delta_\tau = 2$ | $\delta_\tau = 0.5$ | $\delta_\tau = 0.06$ | $\delta_\tau = 2$ | $\delta_\tau = 0.5$ | $\delta_\tau = 0.06$ |
| $\mathcal{K}_L$ | $s_L$ | 34.1 | 175.0 | 3335.5 | 21.0 | 27.5 | 38.4 |
| | $s_G$ | 0.2 | 1.1 | 3.7 | 11.1 | 15.4 | 18.9 |
| | $s_S$ | 0.6 | 1.1 | 2.9 | 13.0 | 14.6 | 17.3 |
| $\mathcal{K}_V$ | $s_L$ | 29.1 | 433.3 | 6552.2 | 59.6 | 62.8 | 67.5 |
| | $s_G$ | 2.5 | 2.8 | 5.1 | 54.0 | 54.6 | 56.0 |
| | $s_S$ | 1.9 | 2.8 | 5.1 | 51.0 | 52.8 | 54.4 |
| $\mathcal{K}_{Li}$ | $s_L$ | 145.6 | 626.9 | 6232.4 | 307.1 | 317.8 | 338.3 |
| | $s_G$ | 0.9 | 3.0 | 9.5 | 279.9 | 285.6 | 288.4 |
| | $s_S$ | 2.0 | 3.3 | 7.5 | 278.1 | 281.0 | 284.6 |
| $\mathcal{K}o_L$ | | 66.9 | 342.9 | 6534.9 | 15.0 | 17.0 | 20.6 |
| $\mathcal{K}o_{LB}$ | | 138.0 | 705.0 | 13068.8 | 15.6 | 17.4 | 21.3 |
| $\mathcal{K}o_V$ | | 210.4 | 2665.9 | 14538.6 | 78.9 | 87.1 | 92.0 |

During the phase of structure creation, an important factor (besides the final size of the structure) is the overhead associated with the calculation of the criterion used to decide how and whether to split the nodes of the tree. This requires a number of calls of function $\mathcal{F}$ and structure $\mathcal{K}$ being built. Thus, the structures using metric interpolation will have creation times accordingly longer. A comparison of the structure creation times for all test functions $\mathcal{F}$ and for the selected approximation accuracy ($\delta_\tau = 0.5$) are shown in Table 6.

## 4.2. Approximation accuracy

Figures 4 and 5 present approximation error $\delta_d$ (for splitting method $s_G$ and functions $\mathcal{F}_4$ and $\mathcal{F}_5$) as a function of tree size. These examples illustrate the efficiency of these trees. For these cases, the kd-tree structures had much smaller sizes, and the requested error threshold could be met with a significantly lower memory cost.

**Table 5**

Selected results for tree structures created for function $\mathcal{F}_5$.

| | | tree size ($m_u$) [kB] | | | access time ($\bar{t}_a$) [$\mu s$] | | |
|---|---|---|---|---|---|---|---|
| tree | split | $\delta_\tau = 2$ | $\delta_\tau = 0.5$ | $\delta_\tau = 0.06$ | $\delta_\tau = 2$ | $\delta_\tau = 0.5$ | $\delta_\tau = 0.06$ |
| $\mathcal{K}_L$ | $s_L$ | 29.1 | 565.3 | 4716.3 | 22.5 | 30.7 | 40.3 |
| | $s_G$ | 1.3 | 10.5 | 94.8 | 16.4 | 22.7 | 32.8 |
| | $s_S$ | 1.3 | 10.8 | 73.7 | 15.6 | 22.4 | 30.0 |
| $\mathcal{K}_V$ | $s_L$ | 29.1 | 829.7 | 5197.7 | 58.6 | 65.1 | 69.0 |
| | $s_G$ | 8.8 | 24.8 | 120.6 | 60.5 | 64.0 | 69.2 |
| | $s_S$ | 8.1 | 24.5 | 105.2 | 59.1 | 64.1 | 67.5 |
| $\mathcal{K}_{Li}$ | $s_L$ | 63.5 | 506.9 | 12176.5 | 303.6 | 317.1 | 345.7 |
| | $s_G$ | 8.8 | 26.7 | 198.2 | 293.9 | 300.4 | 316.0 |
| | $s_S$ | 10.5 | 28.1 | 154.6 | 292.6 | 300.8 | 308.4 |
| $\mathcal{K}o_L$ | | 84.9 | 1614.9 | 11934.9 | 15.8 | 18.6 | 22.4 |
| $\mathcal{K}o_{LB}$ | | 159.0 | 3214.5 | 23705.3 | 16.0 | 18.9 | 22.9 |
| $\mathcal{K}o_V$ | | 465.1 | 3679.1 | 54338.9 | 80.2 | 90.1 | 101.6 |

**Table 6**

Selected results for tree structures for all functions created for accuracy $\delta_\tau = 0.5$.

| | | tree size ($m_u$) [kB] | | | | | creation time ($t_c$) [$ms$] | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| tree | split | $\mathcal{F}_1$ | $\mathcal{F}_2$ | $\mathcal{F}_3$ | $\mathcal{F}_4$ | $\mathcal{F}_5$ | $\mathcal{F}_1$ | $\mathcal{F}_2$ | $\mathcal{F}_3$ | $\mathcal{F}_4$ | $\mathcal{F}_5$ |
| $\mathcal{K}_L$ | $s_L$ | 2337.1 | 956.8 | 0.9 | 175.0 | 565.3 | 294 | 86 | 1 | 21 | 119 |
| | $s_G$ | 2271.7 | 1502.1 | 1.0 | 1.1 | 10.5 | 841 | 328 | 1 | 1 | 6 |
| | $s_S$ | 1780.1 | 1469.2 | 1.3 | 1.1 | 10.8 | 594 | 438 | 1 | 1 | 8 |
| $\mathcal{K}_V$ | $s_L$ | 399.2 | 690.6 | 2.8 | 433.3 | 829.7 | 27 | 38 | 1 | 34 | 89 |
| | $s_G$ | 299.1 | 1841.3 | 4.5 | 2.8 | 24.8 | 37 | 209 | 1 | 1 | 4 |
| | $s_S$ | 285.0 | 1935.5 | 4.2 | 2.8 | 24.5 | 41 | 225 | 1 | 1 | 3 |
| $\mathcal{K}_{Li}$ | $s_L$ | 4267.2 | 1865.8 | 4.0 | 626.9 | 506.9 | 932 | 262 | 1 | 148 | 162 |
| | $s_G$ | 3438.8 | 2932.1 | 2.0 | 3.0 | 26.7 | 1100 | 642 | 1 | 1 | 9 |
| | $s_S$ | 2840.6 | 2627.9 | 4.0 | 3.3 | 28.1 | 958 | 654 | 2 | 1 | 8 |
| $\mathcal{K}o_L$ | | 4500.9 | 2095.7 | 3.9 | 342.9 | 1614.9 | 341 | 88 | 1 | 36 | 287 |
| $\mathcal{K}o_{LB}$ | | 7877.9 | 3671.3 | 6.8 | 705.0 | 3214.5 | 363 | 81 | 1 | 39 | 311 |
| $\mathcal{K}o_V$ | | 11874.2 | 3260.7 | 2.5 | 2665.9 | 3679.1 | 556 | 96 | 1 | 154 | 435 |

Even though tree $\mathcal{K}_V$ usually gives better approximation accuracy and a lower number of tree nodes for the given approximation threshold, the additional memory cost of storing metric values in the vertices of tree nodes and the time cost of metric interpolation results in it being outperformed by $\mathcal{K}_L$, a simpler version of kd-tree (in terms of approximation accuracy and access time per used memory). Kd-tree $\mathcal{K}_{Li}$ also does not show results that are sufficiently good enough to outweigh its higher complexity and metric interpolation cost.

## 4.3. Impact on Created Meshes

Figure 6 presents the reference meshes created with control spaces directly using the analytical form of test functions $\mathcal{F}_4$ (for domain $\mathcal{D}_2$) and $\mathcal{F}_5$ (for domain $\mathcal{D}_3$).

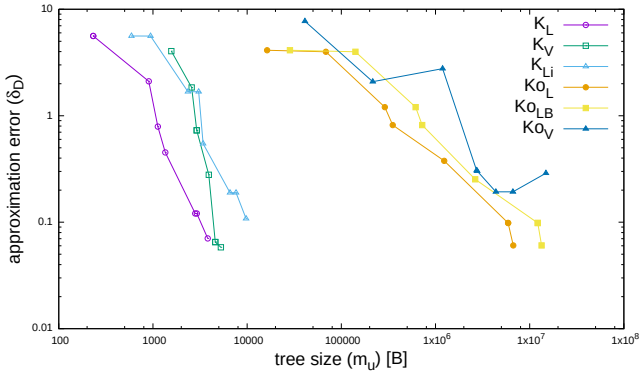


**Figure 4.** Approximation error of control tree structures created (with splitting method $s_G$) for function $\mathcal{F}_4$.

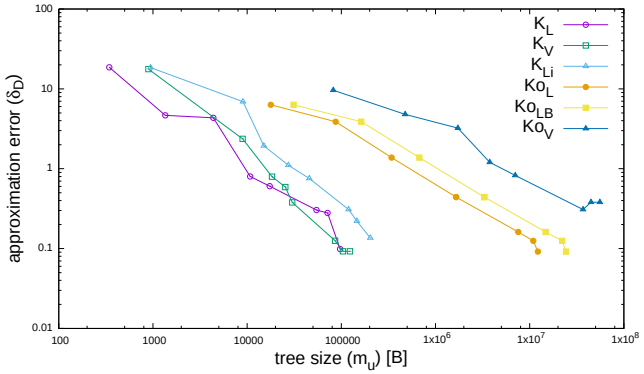


**Figure 5.** Approximation error of control tree structures created (with splitting method $s_G$) for function $\mathcal{F}_5$.

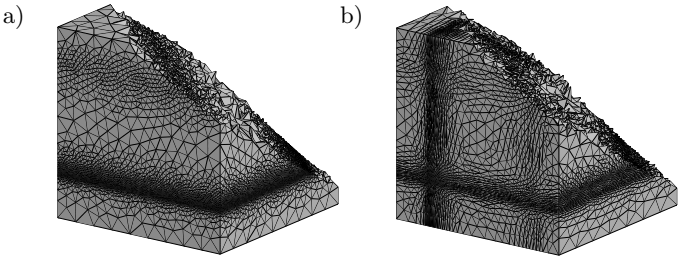


**Figure 6.** Tetrahedral mesh generated for control space using the analytical function directly: a) $\mathcal{F}_4$ (isotropic) – NT=2208833, b) $\mathcal{F}_5$ (anisotropic) – NT=652056.

Figure 7 illustrates the influence of the error threshold used to control the adaptation process on the quality of the produced meshes. Decreasing threshold $\delta_\tau$ has the effect of producing meshes more like the reference meshes (i.e., generated using the control space based directly on functions $\mathcal{F}$).

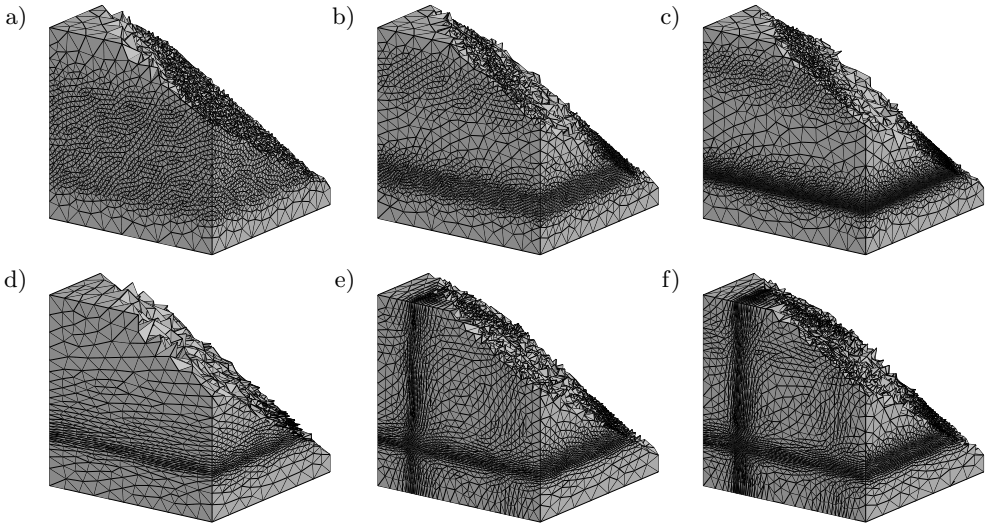


**Figure 7.** Tetrahedral mesh generated for $\mathcal{K}_V$ tree structure adapted with split method $s_G$: a) $\mathcal{F}_4$, $\delta_\tau = 4$, NT=319136, b) $\mathcal{F}_4$, $\delta_\tau = 0.5$, NT=909652, c) $\mathcal{F}_4$, $\delta_\tau = 0.06$, NT=2028032, d) $\mathcal{F}_5$, $\delta_\tau = 4$, NT=61365, e) $\mathcal{F}_5$, $\delta_\tau = 0.5$, NT=424594, f) $\mathcal{F}_5$, $\delta_\tau = 0.06$, NT=576523.

The quality of meshes generated for control spaces based on tree-based structures adapted with a decreasing approximation error threshold is illustrated in Figures 8 and 9. The quality is represented here by the $L_R$ criterion, providing the ratio of number of edges with metric length sufficiently close to optimal in the whole mesh. It can be seen that sufficient quality of the mesh can be reached with reasonably low threshold $\delta_\tau$. If threshold value $\delta_\tau$ is set too high, the characteristics of the generated mesh may be distorted in some subareas of the domain. The results are consistent with the analysis of quality of the kd-tree structures presented in Figures 4 and 5.

## 4.4. Conclusions

Based on our experiments, the presented kd-trees seem to be a good alternative for octree structures as a control space structure for mesh generation and adaptation. They provide greater flexibility for the adaptation procedure, which (in most cases) allows us to create more-efficient structures with smaller memory overhead.

Both kd-tree and octree structures are sensitive to the orientation of the model. Their efficiency and quality of approximation may be worse in the case where model orientation is not aligned with the principal axes of the created tree structure. In both types of trees, some improvements in this respect would be desirable (e.g., associated

with the analysis of the skeleton and symmetry of the model). Similarly, in the case of models with complex and/or concave boundaries, it would be beneficial to consider the introduction of some additional data into the tree structure, or make a domain decomposition of the control space.

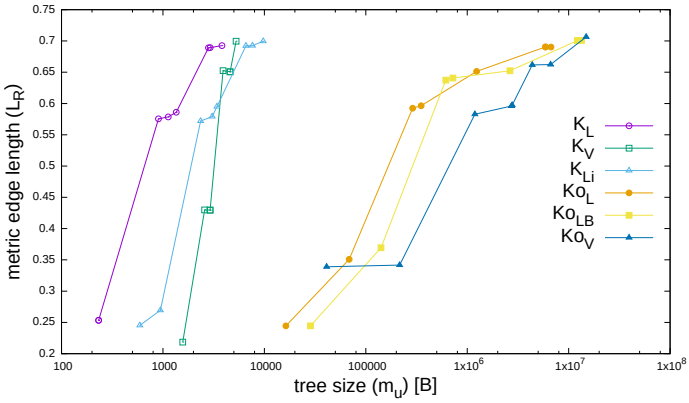


**Figure 8.** Quality of meshes generated for control tree structures created with splitting method $s_G$ for function $\mathcal{F}_4$.

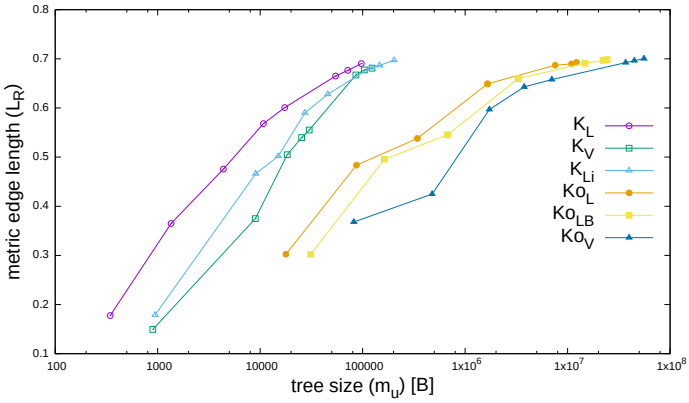


**Figure 9.** Quality of meshes generated for control tree structures created with splitting method $s_G$ for function $\mathcal{F}_5$.

With respect to our selection of the recomended kd-tree version, it seems that the relationship of quality and efficiency to the size of the structure in the typical problems is most advantageous for the simplest kd-tree structure ($\mathcal{K}_L$). The kd-tree structures with interpolation of metric value within the leaves ($\mathcal{K}_V$ and $\mathcal{K}_{Li}$) generally yield better results for the given threshold accuracy of adaptation; however, it is burdened

with additional cost of execution time (both during the adaptation and subsequent use of the structure). It should be noted that trees with an interpolation of metric value within leaves ($\mathcal{K}_V$ and $\mathcal{K}_{Li}$) provide higher continuity of the approximated size function, which may be relevant depending on the application (e.g., the algorithm for mesh generation or adaptation).

## 5. Summary

This article presents the implementation details and comparison of several kd-tree or octree structures for facilitating automated 3d anisotropic mesh generation and adaptation. Time and memory efficiency were inspected in a number of practical tests, together with the approximation quality of the created structures and the quality of the volume meshes generated using control space based on the tested structures. The analysis of results show that the tested kd-tree structures offer an attractive alternative to the octree structure commonly used as a control space structure in the area of mesh generation. The selection of the optimal version of kd-tree structure may, however, depend on the nature of the approximated sizing field.

### Acknowledgements

*The research presented in this paper was partially supported by the AGH grant 11.11.230.124.*

## References

[1] Alauzet F.: Size Gradation Control of Anisotropic Meshes. *Elements in Analysis and Design*, vol. 46(1–2), pp. 181–202, 2010.

[2] Alauzet F., Loseille A., Dervieux A., Frey P.: *Multi-Dimensional Continuous Metric for Mesh Adaptation*, pp. 191–214. Springer, Berlin, Heidelberg, 2006.

[3] Aubry R., Karamete K., Mestreau E., Dey S., Löhner R.: Linear Sources for Mesh Generation. vol. 35(2), pp. A886–A907, 2013.

[4] de Berg M., Cheong O., van Kreveld M., Overmars M.: *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2008.

[5] Borouchaki H., George P.L., Hecht F., Laug P., Saltel E.: Delaunay mesh generation governed by metric specifications. Part I. Algorithms. *Finite Elements in Analysis and Design*, vol. 25, pp. 61–83, 1997.

[6] Deister F., Tremel U., Hassan O., Weatherill N.P.: Fully automatic and fast mesh size specification for unstructured mesh generation. *Eng. Comput. (Lond.)*, vol. 20, pp. 237–248, 2004.

[7] George P., Borouchaki H.: *Delaunay Triangulation and Meshing: Application to Finite Elements*. Butterworth–Heinemann, 1998.

[8] Jurczyk T., Głut B.: Adaptive Control Space Structure for Anisotropic Mesh Generation. In: *Proceedings of ECCOMAS CFD 2006 European Conference on Computational Fluid Dynamics*, Egmond aan Zee, The Netherlands, 2006.

[9] Jurczyk T., Głut B.: The Insertion of Metric Sources for Three-dimensional Mesh Generation. In: *Proceedings* $13^{th}$ *International Conference on Civil, Structural and Environmental Engineering Computing*, Chania, Crete, Greece, 2011, paper 116.

[10] Jurczyk T., Głut B.: Preparation of the Sizing Field for Volume Mesh Generation. In: *Proceedings* $13^{th}$ *International Conference on Civil, Structural and Environmental Engineering Computing*, Chania, Crete, Greece, 2011, paper 115.

[11] Labbé P., Dompierre J., Vallet M.G., Guibault F., Trépanier J.Y.: A universal measure of the conformity of a mesh with respect to an anisotropic metric field. *International Journal Numerical Methematical in Engineering*, vol. 61, pp. 2675–2695, 2004.

[12] Lo D.S.: *Finite Element Mesh Generation*. CRC Press, 2015.

[13] Miranda A.C.O., Martha L.F.: Mesh generation on high-curvature surfaces based on a background quadtree structure. In: *Proceedings 11th International Meshing Roundtable*, pp. 333–342, 2002.

[14] Owen S.J., Saigal S.: Surface mesh sizing control. *International Journal for Numerical Methods in Engineering*, vol. 47(1–3), pp. 497–511, 2000.

[15] Persson P.O., Staten M.L., Xiao Z., Chen J., Zheng Y., Zeng L., Zheng J.: 23rd International Meshing Roundtable (IMR23) Automatic Unstructured Element-sizing Specification Algorithm for Surface Mesh Generation. *Procedia Engineering*, vol. 82, pp. 240–252, 2014.

[16] Pirzadeh S.Z.: Structured Background Grids for Generation of Unstructured Grids by Advancing-Front Method. *AIAA Journal*, vol. 31(2), pp. 257–265, 1993.

[17] Quadros W.R., Vyas V., Brewer M., Owen S.J., Shimada K.: A computational framework for automating generation of sizing function in assembly meshing via disconnected skeletons. *Engineering with Computers*, vol. 26(3), pp. 231–247, 2010.

[18] Zhu J., Blacker T., Smith R.: Background Overlay Grid Size Functions. In: *Proceedings* $11^{th}$ *International Meshing Roundtable*, pp. 65–74, Sandia National Laboratories, Ithaca, NY, 2002.

## Affiliations

**Tomasz Jurczyk**
AGH University of Science and Technology, jurczyk@agh.edu.pl

**Barbara Głut**
AGH University of Science and Technology, glut@agh.edu.pl

**Received**: 20.08.2016
**Revised**: 14.09.2016
**Accepted**: 14.09.2016