Efficient Geometry-based Similarity Search of 3D Spatial Databases

Daniel A. Keim
University of Halle-Wittenberg,
Kurt-Mothes-Str. 1, D-06120 Halle, Germany
keim@informatik.uni-halle.de

Abstract

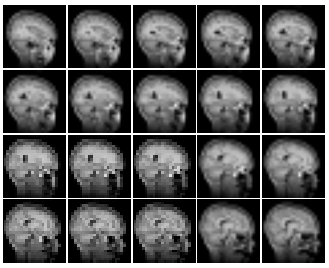
Searching a database of 3D-volume objects for objects which are similar to a given 3D search object is an important problem which arises in number of database applications — for example, in Medicine and CAD. In this paper, we present a new geometry-based solution to the problem of searching for similar 3D-volume objects. The problem is motivated from a real application in the medical domain where volume similarity is used as a basis for surgery decisions. Our solution for an efficient similarity search on large databases of 3D volume objects is based on a new geometric index structure. The basic idea of our new approach is to use the concept of hierarchical approximations of the 3D objects to speed up the search process. We formally show the correctness of our new approach and introduce two instantiations of our general idea, which are based on cuboid and octree approximations. We finally provide a performance evaluation of our new index structure revealing significant performance improvements over existing approaches.

1 Introduction

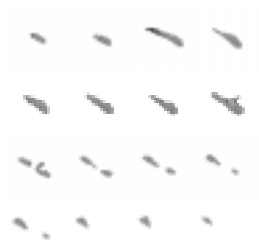
Searching a database of 3D objects for objects which are similar to a given 3D search object is an important problem. The problem arises in a number of applications such as CAD and Medicine. Since our motivation for the work presented in this paper comes from a cooperation with a radiological clinic, in the following we briefly describe the background. The specific medical application of our partners in medicine is epilepsy of children. The current medical theory of epilepsy of children assumes that an irregular development of a specific portion of the brain called the hippocampus is the reason for epilepsy. In several studies, it has been observed that epilepsy only occurs with children whose hippocampi are significantly larger than the average hippocampus of healthy children. On the other hand, it has been found that there also exists a significant number of children which have large hippocampi but do not develop epilepsy.

The current medical practice is to use the available magnetic resonance imaging (MRI) and computer tomography (CT) data to determine the volume of the patient's hippocampus. The data resulting from MRI or CT scanning of the patient are multiple layers of images (cf. Figure 1a) which can be segmented (cf. Figure 1b) and then combined into a voxel-based 3D representation (cf. Figure 2). Note that the position of the patient is fixed in taking the MRI or CT images and therefore, no significant translation or rotation may occur. Depending on the determined volume, the hippocampus is then completely removed by brain surgery. Although this procedure is the best medical doctors are able to provide for the time being, there is a serious need for a more thorough analysis of the hippocampus and the volume defects that cause epilepsy. The observation of children with a large hippocampus but no epilepsy leads to the hypothesis that the shape of the deformation may indicate the defect. In first studies, initial support for this hypothesis has been collected. For a thorough analysis, however, a large number of hippocampi has to be examined and their shapes have to be compared. Using a database of hippocampi, the deformations that lead to epilepsy can be better understood and the surgery can hopefully be improved, e.g., by only removing the affected portions of the hippocampus. A more immediate goal, however, is to use the database to search for similar cases and make the surgery decision based on the outcome of this search. This already would largely improve the decision process and help to avoid unnecessary surgeries. Although our work is motivated by a rather specific medical application, the problem of finding all objects from a database of 3D objects which are similar to a given 3D object is a general problem which arises in many application areas such as CAD, pattern recognition, and others.

It is widely recognized that 3D similarity search is a difficult problem — by far more difficult than the 2D similarity search. Database technology does no yet support geometry-based similarity search of 3D-objects. In comparison to the available systems that support 2D spatial data, the 3D data is much more complex. The currently most widely used techniques for accessing database of complex objects are feature-based approaches (e.g., [Fal 94, MG 95]) which are mainly used as a simple filter to restrict the search space. In case of our application, a useful feature would be, for example, the volume of the objects or a feature vector containing the volumes of a pattern spectrum of the objects [Kor 96]. Although filtering approaches can be very effective, in our application they do not provide good



a. MRI Images



b. Seg. Hippocampus

Figure 1: Image Series of MRI Images and the Segmented Hippocampus

results since the volume of the patients with epilepsy do not differ significantly. This means that a filtering based on the volume would not reduce the search space significantly. Instead, it may even prune objects with interesting shapes away. A second approach which comes from the area of pattern recognition is the similarity search of 3D objects based on their 2D projections. Although this is a very interesting approach which has been used successfully in pattern recognition [NLH 88, SFA 90], it is not very helpful for finding the differences of objects which are generally similar as in our application.

Our new idea to solve the problem is to develop an index structure which supports an efficient geometry-based similarity search on large databases of 3D volume objects. The new index structure uses the actual geometry of the data objects to support an efficient similarity search of the objects. A problem of using the actual 3D geometry is the complexity of the 3D objects, which is by far too complex to be directly stored in any index structure. A solution which has also proven useful in the case of indexing extended 2D objects is to use approximations of the objects in the index to support an efficient pruning of irrelevant objects. In the 2D case, complex polygonal objects are approximated, for example, by minimal bounding rectangles and stored in an R-tree [Gut 84] or its variants such as the R+-tree [SRF 87] or the R*-tree [BKSS 90]. The R-tree and its variants may be seen as a generalization of the B-tree for the two-dimensional case. Our new geometry-based similarity search tree (GSS-tree) may also be seen as a generalization of the B-tree and R-tree for the three-dimensional case. To effectively support similarity queries, the GSS-tree however uses more accurate approximations. Instead of the minimum bounding rectangles of the R-tree, our geometry-based similarity search tree therefore uses the Minimum Surrounding Volume (MSV) and the Maximum Included Volume (MIV) of the data objects as approximations. Since the approximations themselves may be rather complex, instead of using a single approximation in the GSS-tree we use sets of hierarchical approximations which approximate the real data object with increasing accuracy.

The rest of the paper is organized as follows. In section 2, we formally introduce the geometry-based similarity search tree and as a prerequisite, the concept of hierarchical approximations. In section 3, we then introduce two instantiations of our general idea using cuboids and octrees as approximations — the Cuboid Similarity Search tree (cf. section 3.1) and the Octree Similarity Search tree (cf. section 3.2). In section 4, we provide a performance evaluation of the GSS-tree and section 5 provides conclusions and directions for future research.

Figure 2: 3D Representation of the Segmented Hippocampus (Multiple Voxels are combined into Cuboids)

2 The Geometry-based Similarity Search Tree

Before defining the similarity search problem on 3D volume data, in this subsection we first introduce our voxel-based definition of a 3D volume and the volume-based similarity search problem.

2.1 Problem Definition

Definition 1: *(Voxel-based 3D Volume)*
In a three-dimensional discrete space with extension x_{max}, y_{max}, z_{max}, a 3D volume Vol is defined as the set of voxels belonging to the volume

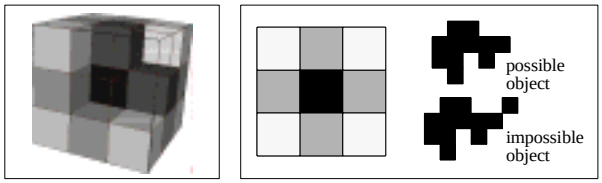
$$Vol = \{(x, y, z)\} \text{ where}$$
$$\forall (x, y, z) \in Vol:\ 0 \le x < x_{max}, 0 \le y < y_{max}, 0 \le z < z_{max} \text{ and}$$
$$\forall (x_1, y_1, z_1), (x_2, y_2, z_2) \in Vol:\ (x_1, y_1, z_1) \sim_N^* (x_2, y_2, z_2).$$

The relation $\sim_N^*$ is the transitive closure of the neighborhood relation of voxels, which may be defined differently depending on the application.

Definition 2: *(Neighborhood Relation of degree d $\sim_{N_d}$)*
The neighborhood relation $\sim_{N_d}$ of degree d ($d \in \{0, 1, 2\}$) may be defined as

$$(x_1, y_1, z_1) \sim_{N_d} (x_2, y_2, z_2) :\Leftrightarrow$$
$$|x_2 - x_1| \le 1 \ \wedge\ |y_2 - y_1| \le 1 \ \wedge\ |z_2 - z_1| \le 1$$
$$\wedge\ |x_2 - x_1| + |y_2 - y_1| + |z_2 - z_1| \le d.$$

The degree allows to specify different notions of neighborhood relations and denotes whether the voxels must be direct neighbors (degree 0), one-level diagonal neighbors (degree 1) or two-level diagonal neighbors (degree 2). In Figure 3a, the dark grey voxels are the voxels which are in neighborhood relation $\sim_{N_0}$, the medium grey voxels are in



a. Neighborhood Relation $\sim_{N_d}$ b. Neighborhood Relations $\sim_{N_0}$

Figure 3: Neighborhood Relation

neighborhood relation $\sim_{N_1}$, and the light grey voxels are in neighborhood relation $\sim_{N_2}$. In Figure 3b, the dark grey squares are in neighborhood relation $\sim_{N_0}$ of the black middle square. Figure 3b also shows two example objects — the upper fulfills definition 2 using the neighborhood relation $\sim_{N_0}$ while the lower object would not be allowed using the degree zero neighborhood relation. Due to the volume properties of our medical volume objects, in our application, it is appropriate to use a $\sim_{N_0}$ neighborhood relation.

To define the similarity search task on 3D-volume objects, we first need to define a similarity measure $\delta: Vol \times Vol \rightarrow \Re$ which determines the volume difference between two volumes.

Definition 3: *(Similarity Measure Volume Difference δ_{VD})*
The similarity measure $\delta_{VD}: Vol \times Vol \rightarrow \Re$ of two 3D volumes vol_1 and vol_2 may be defined as

$$\delta_{VD}(vol_1, vol_2) = 1 - \frac{\|vol_1 \cap vol_2\|}{\|vol_1 \cup vol_2\|}$$

where $\|Vol\|$ denotes the volume covered by *Vol*.

Note that the denominator of the fraction is only necessary for normalizing the resulting volume difference with respect to the overall volume of vol_1 and vol_2. Instead of $\|vol_1 \cup vol_2\|$, other normalization factors such as $max(\|vol_1\|, \|vol_2\|)$ or $0{,}5 \cdot (\|vol_1\| + \|vol_2\|)$ may be used.

Definition 4: *(Congruence, ε-Similarity, NN-Similarity)*
Two 3D volumes vol_1 are called $congruent$ $vol_1 = vol_2$ iff $\delta_{VD}(vol_1, vol_2) = 0$.
Two 3D volumes vol_1 and vol_2 are called $\varepsilon - similar$ with respect to δ_{VD} iff $\delta_{VD}(vol_1, vol_2) \leq \varepsilon$.
A 3D volume vol_1 is called $NN - similar$ to a given 3D volume vol_2 with respect to δ_{VD} and a database of volumes *DB* iff

$$\forall vol \in DB: \ \delta_{VD}(vol_2, vol_1) \leq \delta_{VD}(vol_2, vol).$$

The similarity measure δ_{VD} satisfies the properties of a metric, which is expressed by the following lemma.

Lemma 1: *(Properties of the Similarity Measure δ_{VD})*

(1) identity-preservation: $\forall vol: \ \delta_{VD}(vol, vol) = 0$
(2) commutativity: $\forall vol_1 \forall vol_2:$
$$\delta_{VD}(vol_1, vol_2) = \delta_{VD}(vol_2, vol_1)$$
(3) triangle inequation: $\forall vol_1 \forall vol_2 \forall vol_3:$
$$\delta_{VD}(vol_1, vol_2) + \delta_{VD}(vol_2, vol_3) \geq \delta_{VD}(vol_1, vol_3)$$

Up to now, we have only defined the similarity of two volumes. The next step is to define a similarity query which is searching a database of volumes for similar volumes. Given a database *DB* of *n* volumes vol_i, the similarity query on volumes may be defined as follows.

Definition 5: *(ε-Similarity Query, NN-Similarity Query)*
Given a query volume *s*, find all volumes *vol* from the database *DB* which are $\varepsilon - similar$ to *s* with respect to δ_{VD}, i.e. determine $\{vol \in DB \mid \delta_{VD}(s, vol) < \varepsilon\}$.

Given a query volume *s*, find the volumes vol from the database *DB* which is $NN - similar$ to *s* with respect to δ_{VD}, i.e. determine

$$\{\overline{vol} \in DB \mid \forall vol \in DB, vol \neq \overline{vol}: \ \delta_{VD}(s, \overline{vol}) \leq \delta_{VD}(s, vol)\}$$

Note that finding all congruent volumes is a subtask of solving the ε-similarity or *NN*-similarity query task. This means that any algorithm which solves the similarity query must at least find all congruent volumes.

2.2 Hierarchical Approximations

To support an efficient pruning in the GSS-tree, we use two types of approximations — the Minimum Included Volume (MIV) approximation and the Maximum Surrounding Volume (MSV) approximation. Using both types of approximations has several advantages: The most important advantage is that the pruning of irrelevant branches of the tree becomes more effective. This is important not only in the search process, but also in inserting new data objects. Using both approximations further provides an additional criterion in guiding the search process, allowing the algorithms to follow the more promising branches first. Note that MIV and MSV approximations are generalizations of conservative and progressive approximations which have been successfully used for an efficient query processing in geographical databases [BKS 93] and for similarity queries on databases containing high-dimensional point data [ABKS 98].

Definition 6: *(Maximum Included Volume - MIV, Minimum Surrounding Volume - MSV)*
An approximation of a 3D volume *vol* is called Maximum Included Volume $(MIV(vol))$ iff

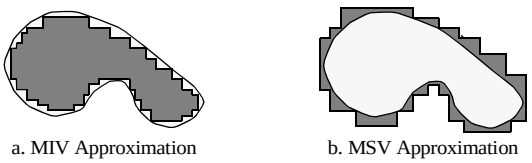
$MIV(vol) \subseteq vol$ and $MIV(vol)$ is maximal for a given type of approximation.

An approximation of a 3D volume *vol* is called Minimum Surrounding Volume $(MSV(vol))$ iff

$vol \subseteq MSV(vol)$ and $MSV(vol)$ is minimal for a given type of approximation.

The maximality of the MIV and the minimality of the MSV approximations depends on the type of approximation used. Examples are provided in the description of the cuboid and octree instantiations (cf. section 3).

In many cases, even a coarse approximation of a volume which may be efficiently stored and processed may already allow a substantial pruning of the search space. In order to make the similarity search more efficient, we therefore use the concept of hierarchical approximations. The basic idea of hierarchical approximations is to use sets



a. MIV Approximation b. MSV Approximation

Figure 4: Example of MIV and MSV Approximation

of approximations which differ in their accuracy. In the search process, we always use the least accurate approximation which provides enough differentiation between the volume objects, thereby reducing the storage overhead and allowing fast comparisons due to the low complexity of the volume objects involved.

Definition 7: *(Hierarchical Approximation)*
A sequence of MIV approximations $MIV_1(vol)$, $MIV_2(vol)$, …, $MIV_k(vol)$ of a volume *vol* is called a hierarchical approximation iff

$$\forall i = 1 \ldots k: \; MIV_i(vol) \subseteq MIV_{i+1}(vol).$$

A sequence of MSV approximations $MSV_1(vol)$, $MSV_2(vol)$, …, $MSV_k(vol)$ of a volume *vol* is called a hierarchical approximation iff

$$\forall i = 1 \ldots k: \; MSV_{i+1}(vol) \subseteq MSV_i(vol).$$

The definition of a hierarchical approximation implies for a sequence of MIV approximations that

$$\|MIV_1(vol)\| \leq \|MIV_2(vol)\| \leq \ldots \leq \|MIV_k(vol)\| \leq \|vol\|$$

and for a sequence of MSV approximations that

$$\|MSV_1(vol)\| \geq \|MSV_2(vol)\| \geq \ldots \geq \|MSV_k(vol)\| \geq \|vol\|.$$

Hierarchical approximations have some important properties which are used for pruning the search space in the GSS-tree. The most important property directly follows from the above observation and is summarized in the following lemma.

Lemma 2: *(Monotonicity of Hierarchical Approximations)*
The minimum overlap / union of a search volume vol_s and hierarchical MIV approximations is monotonously increasing when going to more exact approximation levels, i.e.

$$\forall vol \;\; \forall i = 1 \ldots k-1: \quad \|MIV_i(vol) \cap / \cup vol_s\| \leq \|MIV_{i+1}(vol) \cap / \cup vol_s\|.$$

The maximum overlap / union of a search volume vol_s and hierarchical MSV approximations is monotonously decreasing when going to more exact approximation levels, i.e.

$$\forall vol \;\; \forall i = 1 \ldots k-1: \quad \|MSV_i(vol) \cap / \cup vol_s\| \geq \|MSV_{i+1}(vol) \cap / \cup vol_s\|.$$

Proof:
The lemma directly follows from the monotonicity property of $\cup$ and $\cap$ as well as $\| \; \|$ and the definition of hierarchical approximations. ❑

A second property which is important for using the MIV and MSV approximations of sets of volumes in the directory nodes is the following:

Lemma 3: *(Monotonicity of Union and Intersection of Hierarchical Approximations)*
The intersection of the hierarchical MIV approximations of a set of volumes $VS_m = \{vol_1, \ldots, vol_m\}$ is monotonously increasing when going to a more exact approximation level, i.e.

$$\forall m \; \forall VS_m \;\; \forall i = 1 \ldots k-1: \; \bigcap_{j=1}^{m} MIV_i(vol_j) \subseteq \bigcap_{j=1}^{m} MIV_{i+1}(vol_j)$$

The union of the hierarchical MSV approximations of a set of volumes $VS_m = \{vol_1, \ldots, vol_m\}$ is monotonously decreasing when going to a more exact approximation level, i.e.

$$\forall m \; \forall VS_m \;\; \forall i = 1 \ldots k-1: \; \bigcup_{j=1}^{m} MSV_{i+1}(vol_j) \subseteq \bigcup_{j=1}^{m} MSV_i(vol_j)$$

Idea of the Proof:
The lemma can be shown by induction over *m*, the number of volumes in *VS*. Details can be found in [Kei 97]. ❑

Examples of hierarchical approximations of volume data are octree approximations and approximations by sets of cuboids (cf. section 3).

2.3 Structure of the GSS-Tree

The idea of the geometry-based similarity search tree is to cluster similar objects (i.e., objects with a high volume overlap) in data pages and store the MSV and MIV approximations of the objects in the directory pages. In the directory, the MSVs and MIVs are combined (by union or intersection) and stored on the next higher directory level. The accuracy of the approximations stored in the directory nodes is chosen as low as possible for the objects on the next lower level to be discernible. This guarantees that only the smallest amount of information possible is stored in each of the directory nodes and that the rather expensive volume comparisons can be efficiently approximated. A result, however, is that the accuracy of the approximation may vary between different directory levels, and even on the same directory level, nodes with a different accuracy of the approximations are possible.

The GSS-tree is a completely dynamic height-balanced tree similar to the B-tree [BM 72] and the R-tree [Gut 84]. The leaf nodes of the GSS-tree are all on the same level and contain (MIV, MSV)-pairs together with pointers to the actual volume objects. The directory nodes contain (MIV, MSV)-pairs together with child pointers. Since MIV and MSV approximations of different accuracy levels are used, the fanout of a node is allowed to vary within certain ranges. Also, the node size is not fixed but corresponds to one or multiple adjacent disk pages. This is necessary to prevent the tree from degenerating in case large approximations are necessary to distinguish the data objects.

To simplify the description of the GSS-tree, we use the following notations: The terms MIV_i^l and MSV_i^l are used to denote the hierarchical MIV and MSV approximation of accuracy *i* on tree level *l*, where tree level *l=1* is the root level. (The tree level *l* is only mentioned when necessary.) To access the different parts of an node entry $e = (MIV_i, MSV_j, ptr)$, we write $e.MIV_i$, $e.MSV_j$, and $e.ptr$. The MIV accuracy level AL_{MIV} of the node *e* is $AL_{MIV}(e) = i$ and the MSV accuracy level is $AL_{MSV}(e) = j$. The GSS-tree may be formally defined as:

Definition 8: *(Geometry-based Similarity Search Tree)*
The Geometry-based Similarity Search tree (GSS-tree) is a tree consisting of data and directory nodes. The data nodes contain entries of the form $(MIV_i, MSV_j, obj\text{-}ptr)$ and the directory nodes contain entries of the form $(MIV_i, MSV_j, child\text{-}node)$. The GSS-tree satisfies the following properties:

1. Every data and directory node of a GSS-tree of order m contains between m and $2 \cdot m$ entries unless it is the root. The root contains between 1 and $2 \cdot m$ entries.

2. The node size is variable and depends on the accuracy level AL_{MIV} and AL_{MSV} of the node. Let $b(MIV_i)$ denote the number of bytes necessary to store an MIV approximation of accuracy level i, $b(MSV_j)$ the number of bytes for an MSV approximation of accuracy level j, and $b(ptr)$ the number of bytes for the pointer ptr. For a node storing (MIV_i, MSV_j, ptr)-tuples and a page size of p, the node size is

$$X = \left\lceil \frac{2 \cdot m \cdot (b(MIV_i) + b(MSV_j) + b(ptr))}{p} \right\rceil$$

times the normal page size p.

3. For each entry $(MIV_{\hat{i}}, MSV_{\hat{j}}, obj\text{-}ptr)$ in a leaf node LN, $MIV_{\hat{i}}$ and $MSV_{\hat{j}}$ are the hierarchical approximations of the data object, *obj-ptr* is pointing to. The accuracy level $\hat{i}$ of the MIV approximations in a leaf node LN is chosen such that

$$\hat{i} := \underset{i = 1 \ldots k}{MIN} \{i \mid \forall e_1, e_2 \in LN: e_1.MIV_i \neq e_2.MIV_i\}$$

and the accuracy level j of the MSV approximations is chosen such that

$$\hat{j} := \underset{j = 1 \ldots k}{MIN} \{j \mid \forall e_1, e_2 \in LN: e_1.MSV_j \neq e_2.MSV_j\}.$$

4. For each entry $e = (MIV_{\hat{i}}^l, MSV_{\hat{j}}^l, child\text{-}node^l)$ in a directory node DN, MIV_i^l is defined as

$$e.MIV_i^l := \bigcap_{el \in e.child\text{-}node^l} el.MIV_i^{l+1}$$

where $\hat{i}$ is determined as

$$\hat{i} := \underset{i = 1 \ldots MinAL_{MIV}}{MIN} \{i \mid \forall e_1, e_2 \in DN: e_1.MIV_i^l \neq e_2.MIV_i^l\}$$

and MSV_j^l is defined as

$$e.MSV_j^l := \bigcup_{el \in e.child\text{-}node^l} el.MSV_j^{l+1}$$

where $\hat{j}$ is determined as

$$\hat{j} := \underset{j = 1 \ldots MinAL_{MSV}}{MIN} \{j \mid \forall e_1, e_2 \in DN: e_1.MSV_j^l \neq e_2.MSV_j^l\}$$

$MinAL_{MIV}$ is the minimum MIV accuracy level in DN

$$MinAL_{MIV} := \underset{e \in DN}{MIN} \{AL_{MIV}(e)\}$$

and $MinAL_{MSV}$ is the minimum MSV accuracy level in DN

$$MinAL_{MSV} := \underset{e \in DN}{MIN} \{AL_{MSV}(e)\}.$$

5. All leaves appear on the same level of the tree. ❑

The five properties of Definition 8 define the GSS-tree. In analogy to the B-tree and R-tree, property one defines the fanout of the tree which is between m and $2 \cdot m$ for all nodes except the root node. Since the objects stored in the nodes are hierarchical approximations, the node size is allowed to be a multiple of the normal page size, which is defined by property two. Property three and four define the MIV and MSV approximations used as keys in the tree. An MIV approximation on level l is the intersection of all MIV approximations in the corresponding childnode on level *(l+1)* and the MSV approximation is the union of all MSV approximation in the corresponding childnode on level *(l+1)*. The accuracy of the approximation is chosen as low as possible as long as the approximations stored in the node remain different, but it is never increasing when going to a higher level in the tree [level *(l+1)* to *l*]. The last property states in analogy to the B-tree and R-tree that the tree is height-balanced, which means that all leaves are on the same level of the tree. Since the fanout of all nodes except the root node is between m and $2 \cdot m$, the height of the tree is limited by

$$\lceil \log_{2m+1}(N+1) \rceil \leq H_{ASS}(N) \leq \left\lceil \log_{m+1}\left(\frac{N+1}{2}\right) \right\rceil + 1$$

if N is the number of volumes in the index. This means that the length of one path in the tree is logarithmic in the number of data objects.

In searching the GSS-tree for a given search object, we need two variations of our similarity measures which allow us to traverse the tree and efficiently prune the search space. The idea is to define two similarity measures δ_{VD}^{min} and δ_{VD}^{max} which provide a lower and upper bound for δ_{VD} of a search object and all objects stored in some branch of the tree. Let $OS(e)$ refer to the set of objects stored in branch e of the tree. δ_{VD}^{min} and δ_{VD}^{max} can be defined as:

Definition 9: *(Minimum and Maximum Volume Difference δ_{VD}^{min} and δ_{VD}^{max})*[1]
The minimum volume difference $\delta_{VD}^{min}: Vol \times Node \to \Re$ of a 3D search volume vol_s and the set of 3D volumes $OS(e)$ stored in the subtree e may be defined as

$$\delta_{VD}^{min}(vol_s, e) = 1 - \frac{\|vol_s \cap e.MSV\|}{\|vol_s \cup e.MIV\|}.$$

The maximum volume difference $\delta_{VD}^{max}: Vol \times Node \to \Re$ of a 3D search volume vol_s and the set of 3D volumes $OS(e)$ stored in the subtree e may be defined as

1. Note that δ_{VD}^{min} and δ_{VD}^{max} can also be seen as an interval estimate of δ_{VD} in the sense of interval arithmetic. Theorem 1 then corresponds to the fundamental invariant of interval arithmetic.

$$\delta_{VD}^{max}(vol_s, e) = 1 - \frac{\|vol_s \cap e.MIV\|}{\|vol_s \cup e.MSV\|}.$$

In the following lemma, we show the monotonicity of the δ_{VD}^{min} and δ_{VD}^{max} similarity measures. Lemma 4 together with Lemma 2 and Lemma 3 are then the basis for Theorem 1 which shows the important search tree property of the GSS-tree. As we explain later, Theorem 1 is the basis for the correctness of our search algorithm.

Lemma 4: *(Monotonicity of δ_{VD}^{min} and δ_{VD}^{max})*

For δ_{VD}^{min} and δ_{VD}^{max} as defined by Definition 9 and a GSS-tree as defined by Definition 8, the following monotonicity properties hold:

1. Monotonicity of δ_{VD}^{min}:

$$\forall vol_s\ \forall DN\ \ \forall e \in DN\ \ \forall el \in e.child\text{-}node:$$

$$\delta_{VD}^{min}(vol_s, e) \le \delta_{VD}^{min}(vol_s, el).$$

2. Monotonicity of δ_{VD}^{max}:

$$\forall vol_s\ \forall DN\ \ \forall e \in DN\ \ \forall el \in e.child\text{-}node:$$

$$\delta_{VD}^{max}(vol_s, e) \ge \delta_{VD}^{max}(vol_s, el).$$

Proof: see [Kei97]

Now we are able to show the important search tree property of the GSS-tree which is expressed by the following theorem.

Theorem 1: *(Search Tree Property of the GSS-Tree)*
For δ_{VD}^{min} and δ_{VD}^{max} as defined by Definition 9 and a GSS-tree as defined by Definition 8, the following search tree property holds:

$$\forall vol_s\ \forall DN\ \ \forall e \in DN\ \ \forall vol \in OS(e):$$

$$\delta_{VD}^{min}(vol_s, e) \le \delta_{VD}(vol_s, vol) \le \delta_{VD}^{max}(vol_s, e).$$

Proof: see [Kei97]

Theorem 1 is of high relevance for the correctness of our search algorithm. Theorem 1 implies that in the top-down traversal of a path in the GSS-tree, the minimal and maximal similarity of the search object vol_s and the objects in a subtree $OS(e)$ converge against the actual similarity of the search object vol_s and the objects in the subtree $(vol \in OS(e))$. This means that in following a path of the tree, the search space may be restricted, which allows us to reduce the number of potentially relevant objects.

The high storage requirements seem to be a major drawback of the GSS-tree since storing all hierarchical approximations seems to be prohibitively expensive. The storage requirements, however, may be reduced considerably by only storing the additional information of the MIV and MSV approximations in going to a more accurate approximation level. This can be done since

$$MIV_{i_1}^{l} \subseteq MIV_{i_2}^{l+1} \ \wedge\ MSV_{j_1}^{l} \supseteq MSV_{j_2}^{l+1}$$

which holds due to property four of Definition 8, Lemma 3, and due to the observation already mentioned in the proof of Lemma 4, namely that

$$\forall DN\ \ \forall e_1, e_2 \in DN:\ Level(e_1) \ge Level(e_2) \Rightarrow AL_{MIV}(e_1) \ge AL_{MIV}(e_2)$$

$$\forall DN\ \ \forall e_1, e_2 \in DN:\ Level(e_1) \ge Level(e_2) \Rightarrow AL_{MSV}(e_1) \ge AL_{MSV}(e_2)$$

Note that the idea of storing only the incremental changes of the approximations rather than the full approximations is similar to the idea of prefix-trees. The benefit of applying the prefix idea in the GSS-tree, however, is much higher due to the high storage requirements of the 3D approximations. Applying the prefix idea is possible because of the properties of hierarchical approximations and the properties of the GSS-tree.

Note also that the definition of the GSS-tree is independent of the type of approximation used. Any hierarchical MIV and MSV approximation may be used as long it fulfills the requirements of Definition 6 and Definition 7. Two specific instantiations of the GSS-tree which use cuboids and octrees as approximations are the Cuboid Similarity Search tree and the Octree Similarity Search tree, which are described in section 3.

2.4 Search Algorithm

In searching for all data objects which are similar to a given search volume vol_s, according to Definition 4 we have to distinguish between ε-similarity and *NN*-similarity. In case of ε-similarity, the basic idea of the search algorithm is to use the minimum and the maximum volume difference ($\delta_{VD}^{min}(vol_s, e)$ and $\delta_{VD}^{max}(vol_s, e)$) to prune all branches of the tree, of which the minimum volume difference is higher than the allowed volume difference (ε). The search starts in the root node and tries to prune as many branches as possible. The branches of the tree which cannot be pruned since their minimum volume difference is smaller than the allowed volume difference are put into a list of nodes to be searched in the remaining search process. The objects belonging to nodes of which the maximum volume difference is smaller than the allowed volume difference are added to the result list.

In case of *NN*-similarity, instead of the maximum volume difference ε, the smallest volume difference found so far in the search is used for pruning. The smallest volume difference which is already found is denoted either by the volume difference of the search volume vol_s and some volume vol in the database ($\delta_{VD}(vol_s, vol)$) or by the smallest maximum volume difference ($\delta_{VD}^{max}(vol_s, e)$) of the search volume vol_s and some node e. Again, all nodes which cannot be pruned are put into a list of nodes to be searched and heuristics are used to determine the most promising node to be examined next. After choosing a node, the list of nodes to be searched is pruned again and the same process is repeated until the leaf level is reached. In case of *NN*-similarity, the volume difference of data ob-

```
EpsilonSimilaritySearch (GSSTree *rootnode, VOL *search_vol, float epsilon,
                         SearchStrategy strategy, VolList *result)
{  ListOfTriples *SearchList;
   float VolDiffMin, VolDiffMax, VolDiff;

   SearchList = ∅; SearchList->append(rootnode, 0, Volume (DS));
   for (n = SearchList->GetFirst(); n != NULL; n = SearchList->GetFirst())
   {   SearchList->RemoveFirst();
       for (e = n->GetFirstChild(); e != NULL; e = n->GetNextChild())
       {   if (e == leafnode)
           {   VolDiff = 1- (Volume(e->obj-ptr ∩ search_vol)/
                             Volume(e->obj-ptr ∪ search_vol));
               if (VolDiff <= epsilon)
                   result->append(e);
           } else {VolDiffMin = 1- ((Volume(e->MSV ∩ search_vol)/
                                     Volume(e->MIV ∪ search_vol));
               VolDiffMax = 1- ((Volume(e->MIV ∩ search_vol)/
                                 Volume(e->MSV ∪ search_vol));
               if (VolDiffMin <= epsilon)
                   SearchList->append(e, VolDiffMin, VolDiffMax);
               if (VolDiffMax <= epsilon)
                   for (o = e->GetFirstChild(); o != NULL; i = o->GetNextChild())
                       result->append(o);
           }
       }
       Sort(SearchList, strategy);
   }
}
```

Figure 5: ε -Similarity Search Algorithm

jects is also used for pruning the list. The search process ends if all nodes which possibly contain similar data objects are removed from the search list and the most similar data object is found. If more than one nearest neighbor is searched for, the search ends after the desired number of nearest neighbors has been found. The implementation details of the algorithm and other important algorithms of the GSS-tree (such as the insertion algorithm) can be found in [Kei 97].

3 Instantiations of the GSS-Tree

The GSS-tree is a generic geometry-based index structure which can be instantiated using different hierarchical approximations. In the following, we present two different instantiations — the cuboid similarity search tree and the octree similarity search tree.

3.1 The Cuboid Similarity Search Tree

An instantiation of the GSS-tree which may be seen as an extension of the rectangular approximation of 2D objects in the R-tree is the usage of cuboid approximations of the 3D volume objects. The idea of the Cuboid Similarity Search tree (CSS-tree) is to define the hierarchical MIV and MSV approximations by sets of cuboids. The MIV approximations are defined as sets of additive cuboids and MSV approximations are defined as sets of subtractive cuboids. *Additive* means that the MIV approximations are defined as the union of a set of adjacent non-overlapping cuboids; and *subtractive* means that the MSV approximations are defined as the data space minus a set of non-overlapping cuboids. For a more accurate MIV approximation, additional cuboids are used in the union; for a more accurate MSV approximation, additional cuboids are subtracted from the data space.

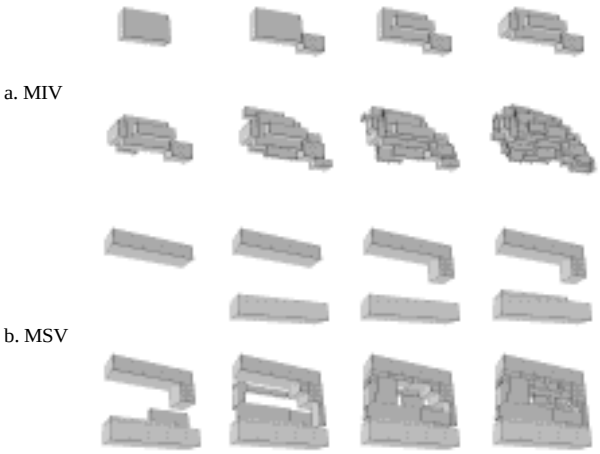


Figure 6: Hierarchical Cuboid MIV and MSV Apprs

Definition 10: *(Cuboid MIV and MSV Approximations)* The hierarchical cuboid **MIV** approximations of a 3D volume Vol are defined as sets of cuboid volumes. The MIV_i approximation of level i is defined inductively by i cuboid volumes $\{C_1, \ldots, C_i\}$:

$$MIV_1 := C_1$$
$$\text{where} \quad Vol \cap C_1 = C_1 \ \wedge \ \forall C \neq C_1: \ \|C_1\| \geq \|C\|$$
$$MIV_i := MIV_{i-1} \cup C_i$$
$$\text{where } \forall (x_1, y_1, z_1), (x_2, y_2, z_2) \in MIV_i: \ (x_1, y_1, z_1) \sim_N^* (x_2, y_2, z_2)$$
$$\wedge \ (Vol - MIV_{i-1}) \cap C_i = C_i$$
$$\wedge \ \forall C \neq C_i: \ \|C_i\| \geq \|C\|$$

The hierarchical cuboid **MSV** approximations of a 3D volume Vol are defined as sets of cuboid volumes. The MSV_i approximation of level i is defined inductively by the data space DS minus i cuboid volumes $\{C_1, \ldots, C_i\}$:

$$MSV_1 := DS - C_1$$
$$\text{where} \quad Vol \cap C_1 = \emptyset \ \wedge \ \forall C \neq C_1: \ \|C_1\| \geq \|C\|$$
$$MSV_i := MSV_{i-1} - C_i$$
$$\text{where} \quad (MSV_{i-1} - Vol) \cap C_i = C_i$$
$$\wedge \ \forall C \neq C_i: \ \|C_i\| \geq \|C\| \ .$$

In Figure 6, multiple levels of hierarchical cuboid MIV and MSV approximations are presented. In the following, we have to show that the approximations defined in Definition 10 are hierarchical MIV and MSV approximations according to Definition 6 and Definition 7, since the correctness of the GSS search tree (cf. Theorem 1) requires the approximations to fulfill these properties.

Lemma 5: *(Cuboid MIV and MSV Apprs are Hierarchical Appr)* The cuboid approximations are hierarchical MIV and MSV approximations according to Definition 6 and Definition 7.

Idea of the Proof:

According to Definition 10, $\forall j$: $C_j \subseteq Vol$ from which follows that $\forall i$

$$\bigcup_{j=1}^{i} C_j \subseteq Vol \quad \Rightarrow \quad MIV_i \subseteq Vol .$$

Also according to Definition 10, $\forall j$: $C_j \cap Vol = \varnothing$ from which follows that $\forall i$

$$Vol \subseteq DS - \bigcup_{j=1}^{i} C_j \quad \Rightarrow \quad Vol \subseteq MSV_i .$$

The *MIV* approximations are hierarchical since the union operation can only provide larger *MIV*s (for $i \rightarrow i+1$) and the *MSV* approximations are hierarchical since the minus operations can only provide smaller *MSV*s (for $i \rightarrow i+1$). ❑

Lemma 5 is important since together with Theorem 1 and the search algorithm presented in subsection 2.4, it guarantees the correctness of the CSS-tree, i.e. that no false dismissals occur in the search process. Note that for the proof and therefore also for the correctness of our approach, we do not need the minimality and adjacency requirement of Definition 10. The minimality requirement is important to guarantee the best possible performance, and the adjacency requirement is important to guarantee that the MIV approximations are still 3D volumes according to Definition 1. Both requirements, however, may be relaxed without losing the correctness.

There are a number of algorithms which are necessary to implement the CSS-tree. For the generic insert and search algorithms presented in 2.4, we need efficient implementations for determining the hierarchical cuboid MIV and MSV approximations and for calculating the union and intersection of sets of cuboids. For determining the hierarchical MIV and MSV approximations, the basic idea is to recursively determine the voxel of maximal intersection and the portion of the volume which is 'reachable' from this point — called the quasi-convex hull. The quasi-convex hull is then the basis for determining the maximal cuboid that fits into the remaining volume. The complexity of the heuristics-based algorithm is $O(|vol|)$. Since the algorithms work on the voxel-based representations of the volumes and are therefore more graphics-related, for details the reader is referred to [Kei 97].

For a better understanding of the CSS-tree, in Figure 7 we show a simple two-dimensional CSS-tree which results from inserting eight objects. The order of the tree is $m = 1$ and each node of the tree is completely filled with $2 \cdot m$ objects. In Figure 7, the MIV and MSV approximations of the two entries in each node are shown. Note that in case of the MSV approximations, only the cuboids are shown; the actual MSV approximations are defined as *DS* minus the shown cuboids. The accuracy level of the nodes which corresponds to the number of cuboids used to approximate the objects in the considered subtree is increasing in top-down direction of the tree. Note that different accuracy levels may be used for different nodes on the same tree

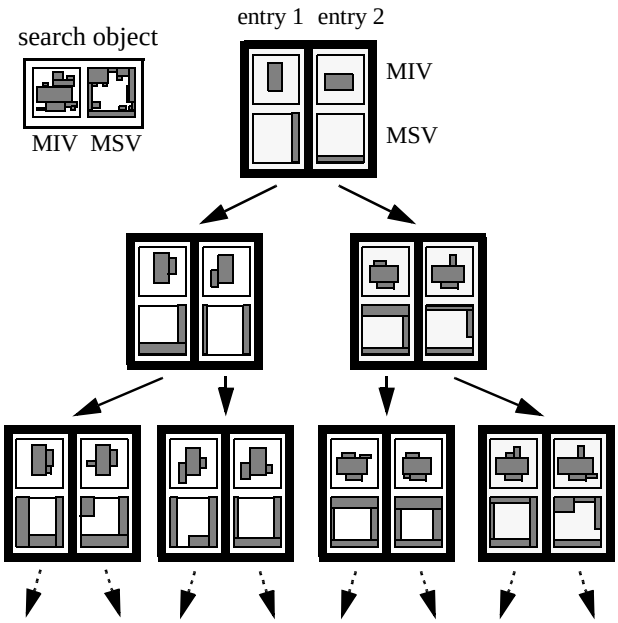


Figure 7: Example of the CSS-Tree

level. On the first level of the tree, for example, the accuracy level of the left node is two whereas the accuracy level of the right node is three. In searching for all objects which are volume-similar to the given search object, the δ_{VD}^{min} and δ_{VD}^{max} of the search object and the hierarchical approximations stored in the nodes have to be calculated. In the example, only the nodes with the grey background (right-most path of the tree) have to be visited.

3.2 The Octree Similarity Search Tree

A second instantiation of the GSS-tree is the Octree Similarity Search Tree (OSS-tree). The OSS-tree uses octrees as hierarchical MIV and MSV approximations and can therefore be seen as a combination of the concepts of the R-tree and the Octree. The octree provides an efficient representation of volume data and is therefore widely used for storing and processing volume data (a detailed description can be found in [Sam 90a] and [Sam 90b]). The octree is based on the principle of a recursive decomposition of space. The basic idea is to successively subdivide a three-dimensional bounded voxel array into eight equal-sized octants. If the volume does not cover an entire octant, the octant is divided into suboctants until blocks are obtained that are entirely covered by the volume or by empty space. For our purpose of using the octree as hierarchical approximations in the OSS-tree, we use the standard octree which is also referred to as region octree.

For defining the hierarchical octree approximations, we first need to introduce a formal definition of the octree representation of a volume.

Definition 11: *(Octree of a Volume)*

A (standard) octree representation of a volume *vol* is a non-balanced tree. A node *n* of the octree is defined by

- the tree level (denoted by $n.level \in \{0 \ldots k\}$),
- the portion of space represented by n (denoted by *n.oct*), and
- the node type (denoted by $n.type \in \{b, w, g\}$),
- for nodes of type *g*, the eight son nodes (denoted by *n.s[i], i = 1..8*).

Let $Nodes^l$ be the set of all nodes on level *l*. The octree is defined by

1. the rootnode:

$$rootnode.level = 0$$

$$rootnode.oct = DS$$

$$rootnode.type = \begin{cases} b & if\ vol = DS \\ w & if\ vol = \emptyset \\ g & else \end{cases}$$

2. the other nodes:

$$\forall n \in Nodes^l : n.type = g \quad \Rightarrow \quad n.s[i] = node_i \quad (i = 1 \ldots 8, l \geq 0)$$

where $node_i.level = l + 1$

$$node_i.oct = Octant(n.oct, i)$$

$$node_i.type = \begin{cases} b & if\ (node_i.oct \subseteq vol) \\ w & if\ (vol \cap node_i.oct = \emptyset) \\ g & else \end{cases}.$$

Note that the octree may be seen as a variable resolution data structure. If we want a lower resolution representation of the considered volume, we may simply use only the lower levels of the octree. According to a theorem from [Sam 90a], the size of an octree is proportional to the sum of the resolution (res) and the size of the boundary of the object (vol^b), i.e. both, the storage complexity as well as the complexity of several algorithms is $O(vol^b + res)$. The advantages of the octree are therefore twofold: The storage and processing requirements in using an octree representation of a volume are only proportional to the 2D surface of the object instead of the 3D volume object itself, and second, the octree provides a variable resolution representation of the volume without inducing additional storage or processing costs.

The first property is important to obtain efficient algorithms for the union and intersection of two volumes and their approximations. The second property is important for our definition of the hierarchical MIV and MSV approximations. The basic idea of the MIV approximation is to modify the octree such that the nodes with type *g* are set to type *w*. This may be applied to an octree of an arbitrary resolution and thereby we obtain a sequence of MIV approximation. For defining the hierarchical MSV approximation, we use a similar idea. In this case, however, nodes of type *g* are replaced by nodes of type *b*.

Definition 12: *(Octree MIV and MSV Approximations)*
The hierarchical octree **MIV** approximations of a 3D volume Vol are defined as a set of octree approximations of varying resolution where the nodes of type *g* are replaced by nodes of type *w*. The MIV_i approximation of level *i* is defined as:

$$MIV_i := \bigcup_{n \in \bigcup_{l=0 \ldots i} Nodes^l} \{n.oct \mid n.type = b\}$$

The hierarchical octree **MSV** approximations of a 3D volume Vol are defined as a set of octree approximations of varying resolution where the nodes of type *g* are replaced by nodes of type *b*. The MSV_i approximation of level *i* is defined as

$$MSV_i := \bigcup_{n \in \bigcup_{l=0 \ldots i} Nodes^l} \{n.oct \mid n.type = b \vee (n.level = i \wedge n.type = g)\}$$

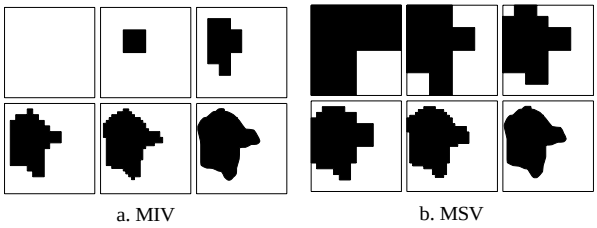

a. MIV b. MSV

Figure 8: Hierarchical Quadtree MIV and MSV Apprs

In Figure 8, we present a two-dimensional example of multiple levels of hierarchical quadtree MIV and MSV approximations (quadtrees are the 2D equivalent of the three-dimensional octrees). In the following, we have to show that the approximations defined in Definition 12 are hierarchical MIV and MSV approximations according to Definition 6 and Definition 7, since the correctness of the GSS search tree (cf. Theorem 1) requires the approximations to fulfill these properties.

Lemma 6: *(Octree MIV and MSV Apprs are Hierarchical Appr)*
The octree MIV and MSV approximations are hierarchical MIV and MSV approximations according to Definition 6 and Definition 7.

Idea of the Proof:
The proof that $\forall j$: $MIV_j \subseteq Vol \wedge Vol \subseteq MSV_j$ can be done based on Definition 12 by induction over *j*. The proof that the MIV and MSV approximations are hierarchical (i.e., $\forall i$: $MIV_i \subseteq MIV_{i+1} \wedge MSV_{i+1} \subseteq MSV_i$) can be shown by induction over *i*. A formal proof is provided in [Kei 97]. ❑

Lemma 6 is important since together with Theorem 1 and the search algorithm presented in subsection 2.4, it guarantees the correctness of the OSS-tree, i.e. that no false dismissals occur in the searching process.

There are a number of algorithms which are necessary to implement the OSS-tree. For the generic insert and search algorithms presented in subsection 2.4, we need efficient implementations for determining the hierarchical octree MIV and MSV approximations and for calculating the

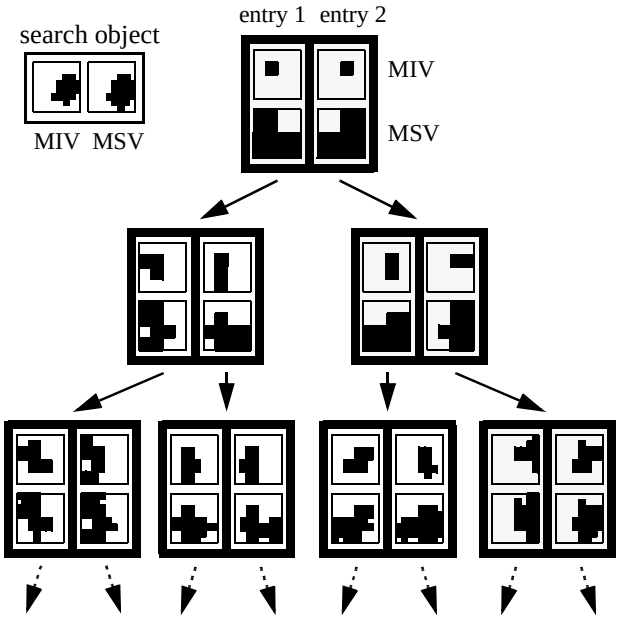


Figure 9: Example of the OSS-Tree

union and intersection of octree MIV and MSV approximations. The algorithms for determining an octree representation of a volume as well as the union and intersection algorithms are similar to those presented in the literature (e.g., [Sam 90a]) and are therefore not discussed here.

For a better understanding of the OSS, in Figure 9 we show a simple two-dimensional OSS-tree which results from inserting eight objects. For simplicity, in the example all nodes have the same accuracy level. In searching for all objects which are volume-similar to the given search object, the δ_{VD}^{min} and δ_{VD}^{max} of the search object and the hierarchical approximations stored in the nodes have to be calculated. In the example, only the nodes with the grey background (right-most path of the tree) have to be visited.

4 Experimental Evaluation

To show the practical relevance of our method, we performed an experimental evaluation the GSS-tree and its two instantiations (CSS-tree and OSS-tree). We also compared the performance results to the currently used method which is a direct volume-based search. All experimental results presented in this section are computed on a 64-bit HPC160 workstation with a few hundred MBytes of main memory and several GBytes of secondary storage. The prototype of the GSS-tree has been implemented in C++ as templates to support different hierarchical MIV and MSV approximations of the data objects. The implementation details can be found in [Kei 97].

For our experiments, we used a realistic data set consisting of real medical hippocampi volumes obtained from our medical partners. The data objects have a resolution of $64 \times 64 \times 16$ voxels, which is given by the medical image generation and segmentation process. Due to the problems

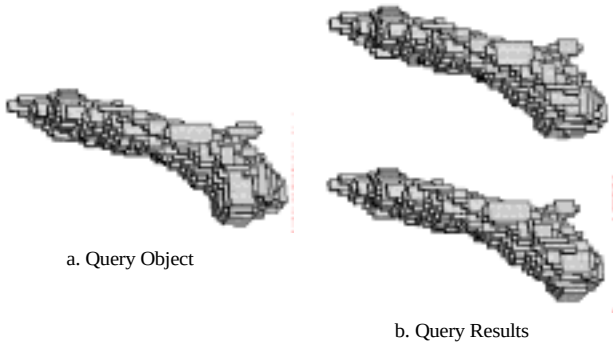


Figure 10: NN-Similarity Query and Results (Data Set 1)

with an automatic segmentation of medical images, however, the data set was originally rather small. Since the advances in medical imaging and (semi-)automatic segmentation will soon produce much larger data sets, for a more realistic performance evaluation and to obtain a variable size database (needed for the experiments depending on the size of the database), we extended the real data set by modifying the data objects as realistically as possible and added the modified data objects to the data set. The modification has been done by adding sphere-shaped regions at randomly generated boundary voxels of the original volume. As a result, we obtain a data base with between 10 and 800 volume objects.

4.1 Evaluation of the Effectiveness

In contrast to feature-based approaches to similarity search, the effectiveness of our geometry-based approach with respect to the given similarity measure can be fully guaranteed (cf. Theorem 1 in section 2.3). This means that our similarity search tree guarantees to find exactly the data objects which fulfill the given volume similarity measure. Nevertheless, for the medical scientists it is interesting to compare the results of our similarity search to their expectations. This is useful to validate and improve the similarity measure. In Figure 10, we show one search object together with two NN-similar result objects which are determined by the CSS-tree and OSS-tree. In Figure 11, we present a query objects together with four ε -similar result objects. The representation shows the objects in their cuboid representation and uses two light sources to enable a limited form of a 3D view. Due to the limits of

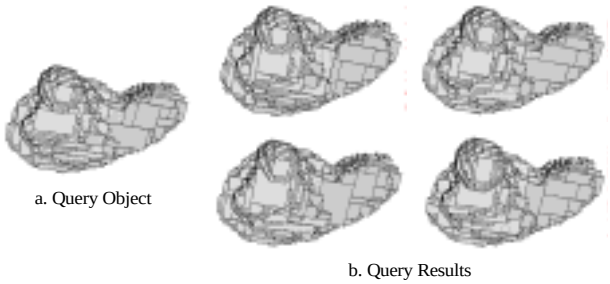


Figure 11: ε -Similarity Query and Results (Data Set 2)

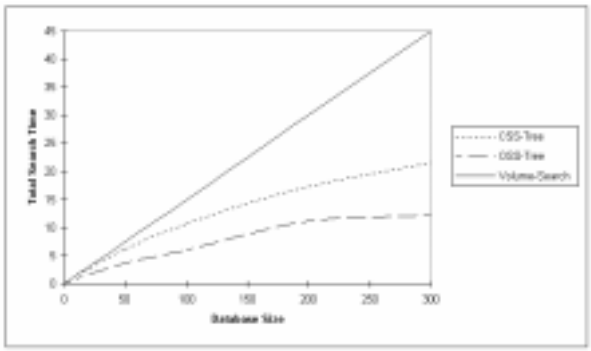


Figure 12: Total Search Time of *NN*-Similarity Queries

the 2D representation, the 3D shapes of the objects are still difficult to discern. The jaggedness of the representation is due to the limitations of the original data which comes from the voxel-based representation of limited resolution.

4.2 Evaluation of the Efficiency

From a database perspective more important than the effectiveness is the efficiency of our approach. In Figure 12, we show the total search time (in seconds) for NN-similarity queries depending on the number of data objects. As expected, for both the CSS-tree and OSS-tree, the search time is sublinear in the number of data objects. In Figure 12, we also show the total search time of the volume-based filtering approach. The basic idea of volume-based filtering is to store the total volume of all data objects in a one-dimensional data structure (e.g., a B-tree) and use the total volume of the search object to restrict the search to a certain volume range resulting in a set of potentially relevant objects. This set is then scanned linearly and in the refinement step, each object is intersected with the search object. The total search time for the volume-based filtering approach is dominated by the time for the time-consuming intersection tests in the refinement step. If the filtering selectivity is constant, the number of objects to be tested in the refinement step increases linearly with the number of data objects in the database (cf. Figure 12). For some applications such as the one described in [Kor 96], a volume-based filtering approach provides good results since the

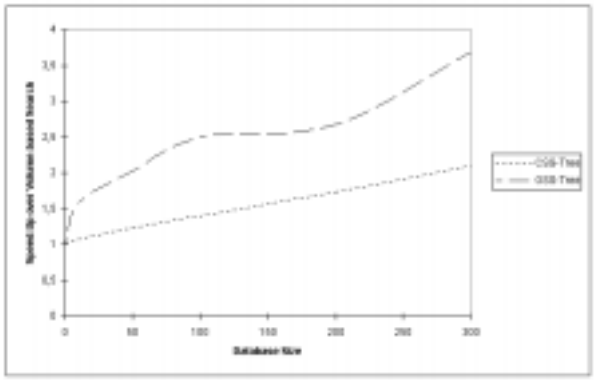


Figure 13: Speed-Up over Volume-based Similarity Search

a. Page Accesses

b. CPU-Time

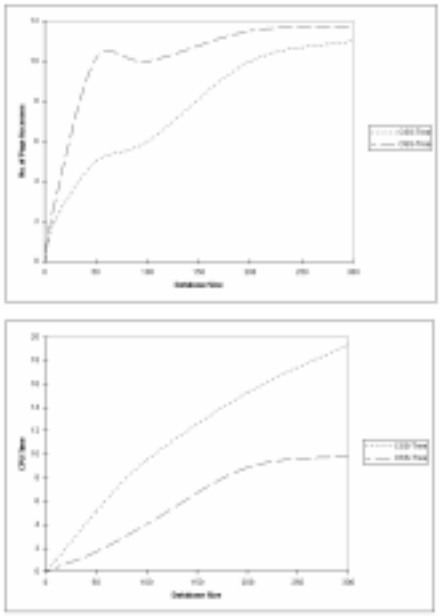


Figure 14: Comparison of Pages Accesses and CPU-Time

filtering selectivity is high. If the volume of all data objects in the database is in a rather small range, however, the selectivity of volume-based filtering is poor since in the refinement step almost the whole database has to be scanned and the time-consuming intersection test has to be performed for each database object. In our database of hippocampi, the objects are all pretty similar and their volume is in a small range since the objects are normalized. The performance of the volume-based filtering is therefore rather bad compared to both — the CSS-tree and the OSS-tree. In Figure 13, we show the speed-up of the CSS-tree and the OSS-tree over the volume-based filtering approach. Since volume-based filtering is almost linear in *N* and CSS-tree and OSS-tree are sublinear in *N*, the speed-up increases with increasing *N*.

In Figure 14, we provide a more detailed analysis of the CSS-tree and the OSS-tree. We compare the number of page accesses[1] and the CPU-time (in seconds). As expected, the CPU-time of the CSS-tree is much higher since

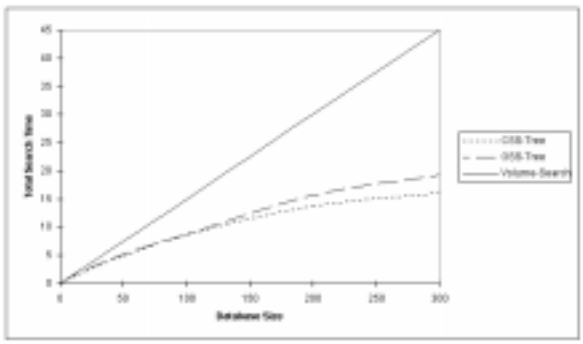


Figure 15: Comparison of CSS- and OSS-tree (Data Set 2)

the intersection and union of sets of cuboids is much more time-consuming than the intersection and union of MIV and MSV octree approximations. On the other hand, the precision of the cuboid approximations is better, which leads to a better pruning of nodes early in the search process and therefore a lower number of page accesses. Figure 14 clearly shows the advantages and disadvantages of CSS-tree and OSS-tree: The CSS-tree provides a better filtering and therefore less node accesses, but the intersection and union operations on sets of cuboids are more time-consuming. In contrast, the OSS-tree provides fast intersection and union operations, but due to the fixed space partitioning of the octree the approximations are of less precision and therefore more node accesses are necessary. The usefulness of the octree partitioning scheme, however, is data-dependent and therefore, a general statement, which of the instantiations (CSS-tree or OSS-tree) provides a better performance, is not possible. To show the advantage of the CSS-tree over the OSS-tree, we generated a second slightly different data set. In data set two, the likelihood that the partitioning scheme of the octree provides a suboptimal partitioning is higher. In Figure 15, we present the performance result of the CSS-tree and OSS-tree. In this case, the performance of the CSS-tree is better than the performance of the OSS-tree.

5 Conclusions

The main contribution of the paper is a new geometry-based index structure which generalizes the well-known R-tree approach for an efficient volume-based similarity search on 3D volume objects. Our solution is based on the general concept of using both, progressive (MIV) and conservative (MSV) approximations, and the concept of using a hierarchy of approximations. The approximations are used to define a minimum and maximum volume difference measure which is formally shown to be correct and allows an efficient pruning of the search space. We developed two instantiations of our geometry-based index structure, which are based on cuboid and octree approximations. The practical relevance and feasibility of our approach is shown by applying our new techniques to the real data from our medical applications. Our experimental evaluation of the two variants of the GSS-tree reveals significant performance improvements over existing approaches. Although the GSS-tree has been developed with our medical application in mind, it is generally applicable in a wide range of other applications.

There are a number of open research directions and a lot of future work to do: Other MIV and MSV approximations need to be explored and more experience needs to be collected by applying the GSS-tree in other application contexts, some of which may require an extension of the GSS-tree to allow a similarity search under other invariances.

1. Note that for a fair comparison, the number of page accesses is determined from the number of node accesses by weighting each node access with the size of the node. The weighting corresponds to the time needed to load the node.

References

[ABKS 98] Ankerst M., Braunmüller B., Kriegel H.-P., Seidl T.: *Improving Adaptable Similarity Query Processing by Using Approximations*, Proc. 24th Int. Conf. on Very Large Data Bases, New York, 1998, pp. 206-217.

[BKS 93] Brinkhoff T., Kriegel H.-P., Schneider R.: *'Comparison of Approximations of Complex Objects Used for Approximation-based Query Processing in Spatial Database Systems'*, Proc. 9th Int. Conf. on Data Engineering, Vienna, Austria, 1993, pp.40-49.

[BKSS 90] Beckmann N., Kriegel H.-P., Schneider R., Seeger B.: *'The R*-tree: An Efficient and Robust Access Method for Points and Rectangles'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, 1990, pp. 322-331.

[BM 72] Bayer R., McCreight E. M.: *'Organization and Maintenance of Large Ordered Indices'*, Acta Informatica, Vol. 1, No. 3, pp. 173-189.

[Fal 94] Faloutsos C., Barber R., Flickner M., Hafner J., Niblack W., Petkovic D.: *'Efficient and Effective Querying by Image Content'*, Journal of Intelligent Information Systems, 1994, Vol. 3, pp. 231-262.

[Gut 84] Guttman A.: *'R-trees: A Dynamic Index Structure for Spatial Searching'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Boston, MA, 1984, pp. 47-57.

[Kei 97] Keim D. A.: 'Efficient Support of Similarity Search in Spatial Data Bases', Habilitation thesis, University of Munich, 1997.

[Kor 96] Korn F., Sidiropoulos N., Faloutsos C., Siegel E., Protopapas Z.: *'Fast Nearest Neighbor Search in Medical Image Databases'*, Proc. 22nd Int. Conf. on Very Large Data Bases, Mumbai, India, 1996, pp. 215-226.

[MG 93] Mehrotra R., Gary J. E.: *'Feature-Based Retrieval of Similar Shapes'*, Proc. 9th Int. Conf. on Data Engineering, Vienna, Austria, 1993, pp. 108-115.

[NLH 88] Noborio H., Liang P., Hackwood S.: *'Construction of the Octree Approximating Three-Dimensional Objects Using Multiple Views'*, IEEE Trans. on Pattern Analysis and Machine Learning, Vol. 10, 1988, pp. 769-782.

[Sam 90a] Samet H.: *'Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS'*, Addison-Wesley, 1990.

[Sam 90b] Samet H.: *'The Design and Analysis of Spatial Data Structures'*, Addison- Wesley, 1990.

[SFA 90] Srinivasan P., Fukusa S., Azimoto S.: *'Computational Geometric Methods in Volumetric Intersection for 3D Reconstruction'*, Pattern Recognition, Vol. 23, 1990, pp. 843-857.

[SRF 87] Sellis T., Roussopoulos N., Faloutsos C.: *'The R^+-Tree: A Dynamic Index for Multi-Dimensional Objects'*, Proc. 13th Int. Conf. on Very Large Databases, Brighton, England, 1987, pp 507-518.