International Journal of Computer Assisted Radiology and Surgery manuscript No.
(will be inserted by the editor)

Fast and Flexible Distance Measures for Treatment Planning

Ivo Rössling · Christian Cyrus · Lars Dornheim · Andreas Boehm · Bernhard Preim

the date of receipt and acceptance should be inserted later

Abstract *Purpose:* Distance measures are required for diagnoses, therapy decision and documentation. With today's high resolution CT and MR imaging techniques, high quality images have become possible. Yet, manual measurement can be tedious. We present a method for automatically determining different distance-based measures on segmented anatomic structures, like shortest distances, diameters, and wall thicknesses.

Methods: Our method is inspired from computational geometry and based on a surface mesh representation. The computation takes all primitives (points, edges, faces) into account and organizes them efficiently in a spatial tree structure. We followed the generic design paradigm in order to achieve maximum flexibility.

Results: The generic approach allows for a variety of intervention-relevant distance measures to be computed, using only a single type of data structure. For shortest distance, our approach in empirical tests turned out to be more efficient than previous methods from medical application literature. Besides the numerical value, also its defining geometric primitives are determined.

Conclusions: The presented technique is both, fast and flexible. It can be used to interactively derive automatic distance measures for arbitrary mesh-based segmentations. Due to the geometrically exact measurements it is possible to reliably estimate safety margins, assess possible infiltrations and other clinically relevant measures. To exploit this benefit, the method requires precise segmentations as input data.

Keywords treatment planning · automatic measurement · surface meshes

Ivo Rössling · Christian Cyrus · Bernhard Preim
Otto-von-Guericke University
Institute of Simulation and Graphics
Universitätsplatz 2 D-39106 Magdeburg Germany
E-mail: {iroess,bernhard.preim}@ovgu.de, ccyrus@st.ovgu.de

Lars Dornheim
Dornheim Medical Images GmbH, Magdeburg, Germany
E-mail: lars@dornheim-medical-images.de

Andreas Boehm
Universitätsklinikum Leipzig, Germany
E-mail: andreas.boehm@medizin.uni-leipzig.de

1 Motivation

The assessment of anatomical and pathological structures based on imaging techniques plays a vital role in surgical diagnostics and therapy planning. Yet, the visual image (Figure 2) not always allows a reliable evaluation. To support their decisions, medical experts then seek for additional quantitative information on, e. g., texture, shape, volumetry, and spatial relationships.

Specific distance-based measures are crucial for preoperative risk assessment. In the field of tumor diagnosis, for example, maximum extents of tumor and malignant lymph nodes are essential for overall tumor staging (TNM classification), which determine possible therapy decisions. Furthermore, resectability often crucially depends on the distances of tumor tissue to several risk structures (main vessel, etc.).

Quality and resolution of CT and MR imaging have increased considerably in the last years. So it is possible to derive precise measures from this data in principle. However, a purely manual measurement process is tedious and imprecise, not least due to the 3D nature of the data. Medical experts still often measure by hand directly in single 2D slices, occasionally obtaining significantly incorrect measures (Figure 1) due to the fixed cutting plane orientation, but also inexact (since intuitively chosen) endpoints. Manual measurement may be performed directly in 3D, e. g. within some interactive

direct volume rendering environment (cf. Hastreiter et al. [9]). But even if such 3D interaction is available, exploiting the additional dimension exhaustively to find the correct measure may turn into an ambitious task.

For automatic and exact measurements, segmentations are needed. Obviously, to finally obtain accurate measures, corresponding precise segmentations are required. Today, such segmentations are possible, yet not always easy or even fully automatic to generate. However, high resolution segmentations usually consist of a large amount of voxels or triangles. In consequence, measurement methods have to deal with a moderate amount of data, which will become even larger in the future. Reducing the data is not recommended, because accuracy will be lost.

The remainder of this paper is organized as follows: Section 2 starts with reviewing related work on distance computation in medical imaging and other scientific areas. In Section 3 we will first present a generic design that allows for a plenty of different measurement tasks to be treated in a uniform and efficient way. Then we will focus on the concrete application of computing shortest distances between anatomic structures and present corresponding realizations for the individual generic components for accomplishing this particular task. In Section 4, a set of other clinically relevant applications will be discussed. In Section 5, we will return to the problem of shortest distance computation and evaluate our solution in a case study in the field of neck surgery planning. Finally, the last section draws a conclusion of this work.

2 Related Work

Despite its potential, automatic measurement of segmented structures has barely been addressed in pertinent literature on medical imaging. Distance fields, obtained from applying 2D or 3D distance transformations on voxel-based segmentations (see Jones et al. [10] and Fabbri et al. [4] for comprehensive surveys), are used in image analysis and shape recognition, e. g. for extracting skeletons. Measures like object extents or inter-object distances, however, have not been treated with this approach. Distance fields appear simply unsuitable for this kind of tasks for several reasons. Although point-to-object distances are directly at hand, using that object's distance field, there is much more effort required for determining object-to-object distances. Also, a separate distance map needs to be generated for each segmentation, which is both highly time- and space-consuming. Every single distance map needs to cover the whole bounding volume of all relevant objects. In addition, higher image acquisition resolution increases the number of voxels cubically. Finally, since the measure is intrinsically hard-coded into the field, different tasks require distinct distance maps to be build.

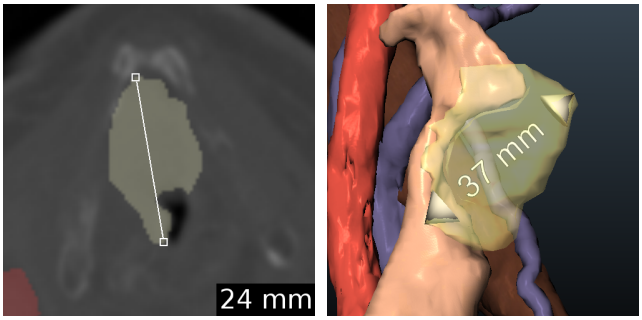

Fig. 1 *Left*: A real clinical case for which the involved medical expert determined a maximum diameter of 24 mm for the tumor. *Right*: In fact, the real tumor extent was 37 mm, which is about 50% more than assessed manually and close to be classified T3.

Even less investigation has been made regarding distance-based measures on segmentations given as surface meshes. In particular, besides Preim et al. [15] no other published approach in medical application could be found, for example, that directly concerns automatic measurement of shortest distances between anatomic structures. This lack of domain-specific approaches calls for considering solutions in other scientific areas. In computational geometry and robotics, for example, this topic has already been investigated more closely in the past. At least in robotics, however, the main interest was more on pure collision detection or approximation. In consequence, the findings can be applied to the present scenario only to a limited extent.

While for a long time approaches were restricted to convex objects (e. g. Bobrow [1], Lin and Canny [14]), more and more the non-convex setting is considered. Thereby, mere analytical approaches (e. g. Gilbert and Johnson [7]) rarely found their way into practice. Voronoi-based approaches (e. g. Kawachi and Suzuki [11] or Sud et al. [18]) were proposed for fast proximity computation. Yet, robust implementation can be problematic in practice (cf. Ledoux [13]). Moreover, as with distance fields, also Voronoi diagrams have their underlying distance measure hard-wired into the representation.

Apart from that, hierarchy-based methods are most commonly applied. The respective hierarchy may be given by different levels of detail (LOD) of the given object (so-called *hierarchical object models*, e.g. Faverjon [6]). Alternatively, the hierarchy may consist of a set of bounding volumes that, in a bottom-up manner, progressively cover more and more volumetric parts of the object (so-called *bounding representations*).

Bounding representations consist of two basic parts: A spatial decomposition technique to partition the in-

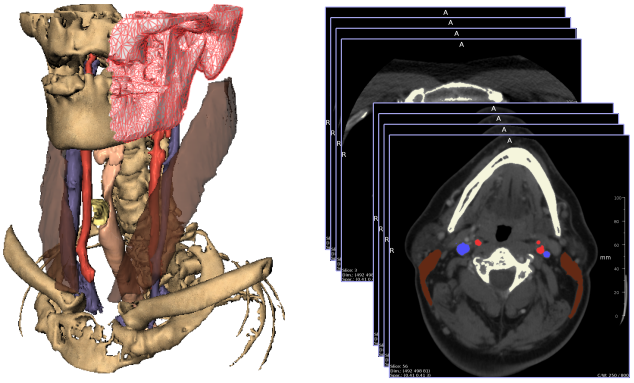

Fig. 2 Two common ways for representing segmentations. *Left*: 3D surface meshes. *Right*: Binary voxel masks as colorized overlays for the CT dataset slices. (*White/Beige*: Bones. *Blue*: Venes. *Red*: Arteries. *Brown*: muscles.)

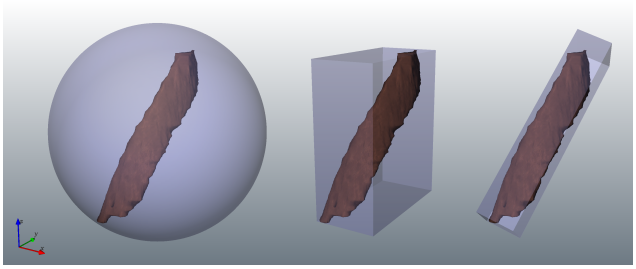

Fig. 3 Bounding volumes for musculus sternocleidomastoideus. *Left*: A bounding sphere is insensitive to object's orientation, but fits elongated objects increasingly worse. Moreover, the unique minimum sphere is generally hard to be determined. *Middle*: The minimum axis-aligned bounding box is extremely easy to compute and compare, but its size is sensitive to object's orientation. *Right*: The minimum object-oriented bounding box fits the object best, but computation and comparison are much more expensive.

put data into suitable subsets, and a geometric shape definition together with a construction rule to create a bounding volume for data to be covered. Gottschalk et al. [8] provide a comprehensive survey on different kinds of bounding volumes and spatial decomposition techniques and discuss the to-be-expected performance of hierarchical methods in terms of a cost equation.

The approach described by Preim et al. [15] for computing shortest distances between anatomic structures is also based on bounding representations. Following Quinlan [16], spheres were used as bounding volumes, which in turn appears to be non-optimal choice with regard to running time. Apart from that the mesh-based segmentations provided as input are initially reduced to their vertex sets. As a result, potentially valuable precision may be lost and, in particular, arbitrary intersections cannot be safely identified anymore.

3 Efficient and Flexible Distance Measurement – A Uniform Approach

In the following, we present a new approach for distance measure computation for surgical diagnostics and treatment planning. Three major objectives shall be met: First of all, the method has to be *efficient*. For effective use in interactive treatment planning, requested measures need to be delivered as immediately as possible. Second, the method must provide some notion of *quality guarantee*. The physician demands to have a justified trust in the computed results. And finally, the method shall be *versatile*. One may come up with a whole set of point solutions to tackle different relevant tasks – a uniform approach, however, helps reducing preprocessing time and overall memory requirements, which is important given today's growing data volumes.

We basically consider segmentations given as 3D surface meshes (Figure 2). They may represent subvoxel accuracy and facilitate particularly efficient, exact and generic distance measurement. This decision is not too much a restriction, as voxel-based segmentations can easily be transformed into equivalent triangle meshes.

Based on the given representation we aim for geometrically correct results. Note that there are indeed inaccuracies in terms of variations during image acquisition and segmentation. However, these issues are already introduced at some earlier point in the pipeline and out of our control. The best one can do is to guarantee that no additional errors arise. Although it may not appear very convincing to compute exact results from inaccurate data, there is nevertheless a clear benefit. Segmentations that were created or at least approved by a medical expert can actually be considered a subjective ground truth. In doing so, we can guarantee exact results for this "clinical finding" that has been expressed by the expert by means of the given segmentations.

The basic strategy of our measurement approach is as follows: In a preprocessing step, a spatial search tree is constructed for each mesh. The identification of a specific characteristic value, such as the shortest distance, then simply amounts to performing a query on these geometric search data structures. The separate choice of the design's exchangeable components thereby allows for computing plenty of different measures with only a *single* kind of data structure. These issues will be discussed more detailed in the following subsections. We will first present the overall generic design and then focus on the concrete application of computing shortest distances between anatomic structures. For this particular task, corresponding realizations for the individual generic components will be given and performance aspects will be discussed.

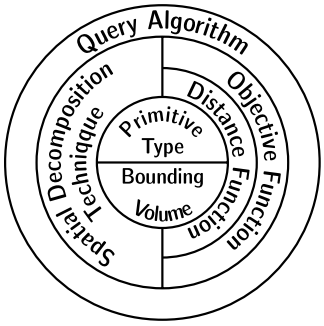

Fig. 4 The generic design of the approach allows the internal components to be replaced in order to fit different measurement tasks or constraints.

3.1 Generic Design

To support a wide range of clinically relevant measures, we seek for maximum flexibility and follow the generic design paradigm. It allows each component to be replaced by any other fulfilling the same requirements and has proven very successful in the design of well-known algorithm libraries, such as the C++ Standard Template Library (STL), the Boost Library, and the Computational Geometry Algorithms Library (CGAL) [5].

The core of the overall design (Figure 4) is made of a small set of permitted ***geometric primitive types***. Input data is assumed to be given in terms of a finite set of elements of these types. For "clean" 3D triangle meshes, it is basically sufficient to provide just triangles. Yet, for also being capable of processing generalized meshes with degenerated faces, we consider all three simplices (i. e. point, edge and triangle). Further types of primitives are possible but rarely used in practice.

The primitives of a given input mesh will be later covered by a hierarchy of ***bounding volumes*** of predefined type(s). The supported shape types can be arbitrary, provided that they are convex and allow for fast distance calculation between any two instances. In general, simple shapes are used (Figure 3) such as spheres, axis-aligned bounding boxes (AABB), object-oriented bounding boxes (OOBB), or discrete oriented polytopes (thereby usually sticking to one type only).

Based on the two sets of allowed primitives and allowed bounding volumes, a ***distance function*** needs to be defined, operating on pairs of instances from each of the two sets. Applying this function on two primitives has to return the exact distance in between, measured in the metric of choice. For two bounding volumes, in contrast, the distance calculation just needs to impose some (depending on the particular context either lower or upper) bound on any possible distance between a primitive contained in the first and another primitive contained in the second bounding volume.

The bounding volumes are used to guide the query through the spatial search tree. This tree is constructed in a top-down or bottom-up manner. Its leaves contain a predefined maximum number of primitives and any node is assigned an associated bounding volume that encloses all primitives covered by this node. The construction process is guided by a ***spatial decomposition technique*** that partitions a given space into subspaces, distributing the contained primitives into corresponding subsets. Depending on the chosen decomposition technique, one ends up with a specific type of spatial search tree, such as an octree or kD-tree (for other techniques see classical text books on computational geometry and geometric data structures [17, 12]).

In course of querying the spatial search trees of the involved meshes, the distance function is applied to pairs of bounding volumes or geometric primitives. The issue of merging the different results as well as the sorting order of nodes remaining to be processed, is treated by a given ***objective function***.

The ***query algorithm*** finally combines all components to perform the desired measurement task. In the basic case, two search trees for the structures of interest are passed as input and the requested measure is returned. Yet, input and output parameters can also be extended, e. g. to allow a whole set of objects to be given as input or to provide enhanced output for subsequent use, like the defining geometric primitives corresponding to the computed value(s), etc. Also, further user-specified constraints, such as a region of interest may be taken into account during the search process.

3.2 Components for Computing the Shortest Distance

Our generic framework allows the concrete components to be chosen in a way that optimally suits the specific setting and measurement task. As exemplary medical application we considered the task of determining the shortest distance between two anatomic structures.

We expect the input to be given by surface triangle meshes (cf. Figure 2). To allow also generalized meshes with degenerate faces (e. g. skeletons or centerlines, see Figure 7) to be processed, all three primitive types ***point***, ***edge*** and ***triangle*** were chosen to be supported. For the type of bounding volume, we decided on ***axis-aligned bounding boxes***, for they are fairly easy and efficient to be computed and compared. According to Gottschalk et al. [8], object-oriented bounding boxes yield a better fitting of the data, but are much harder to compute and slower to compare. Bounding spheres, on the other hand, usually fit the data worse than axis-aligned bounding boxes. They also turned out to be more expensive, both in construction and comparison.

As distance function, we implemented the ***Euclidean distance*** for bounding boxes and all pairs of supported primitive types. Since the task was to determine the shortest distances, accordingly the ***minimum*** was used as objective function.

Our spatial decomposition technique is similar to an ***octree***, but based on ***barycenters*** of the covered primitives. An ordinary octree starts with a bounding cube of pre-defined size and position (both usually a power of 2). In each level, the current cube is divided exactly at its middle point into 8 equal sized sub-cubes. For our tree in turn, an overall barycenter (also called "center of mass" or "center of gravity") of the contained primitives is determined, which serves as a pivotal point for dividing the box and partitioning the data. In addition, instead of just using the resulting sub-boxes, we always compute new bounding boxes in order to obtain better bounds. Altogether, the tree is basically as easy to construct as an ordinary octree, but it is much more balanced and thus shows better query time.

Finally, the query algorithm is a ***priority-queue-based search*** on the factor graph induced by the two given search trees. To provide a deeper insight, the remaining three subsections shed some light on the details of the spatial search tree data structure, the shortest distance query algorithm itself, and finally the implementation of distance calculation between primitives.

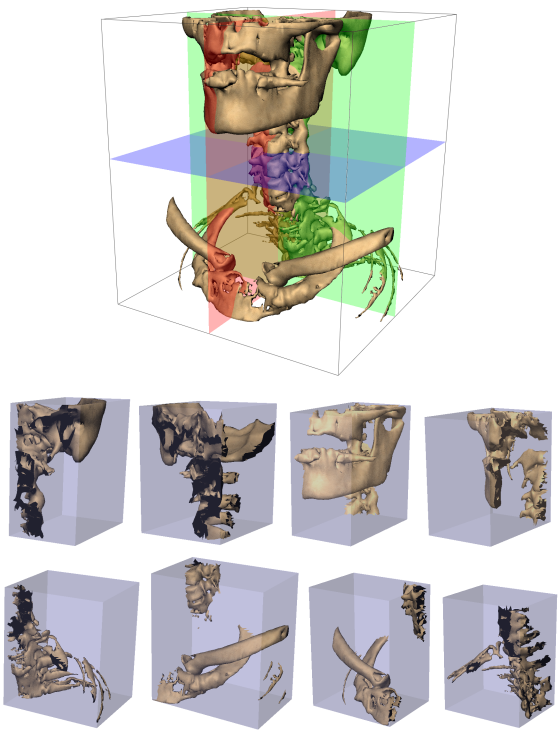

Fig. 5 Recursion step of search tree construction. *Top*: The overall barycenter of all covered primitives determines the position of the splitting planes. *Bottom*: The set of primitives is partitioned into according spatial subsets and corresponding child nodes are created. For each of the subsets, the corresponding AABB and barycenter is thereby computed on-the-fly.

3.3 Construction of a Search Tree Data Structure

For the surface mesh of a given anatomic structure we construct the corresponding search tree recursively in a top-down manner. As a preprocessing step, for each primitive the corresponding barycenter is computed which represents that primitive's center of mass. Each recursion level then consists of two steps (Figure 5). First, a new tree node is created to hold the current set of primitives. In the course of this, the corresponding bounding box and weighted barycenter are determined on-the-fly.

The node's barycenter is simply the weighted sum of all individual barycenters of the covered primitives, using the weights $w = 1, 2, 3$ for points, segments, and triangles, respectively. This point divides the bounding volume into a *left* vs. *right*, *top* vs. *bottom*, and *front*

Algorithm 1 Construct Search Tree

Input: $\mathcal{S} = \{\pi_1 \dots \pi_n\}$: set of input primitives π_i
Output: $\mathcal{T}$: search tree for $\mathcal{S}$

Compute barycenters for the mesh's geometric primitives:
1: **for all** $\pi_i \in S$ **do**
2: compute $bc_i \leftarrow BaryCenter(\pi_i)$

Construct search tree recursively in a top-down manner:
3: $\mathcal{T} \leftarrow CreateTreeNode(\mathcal{S})$
4: $Split(\mathcal{T})$
5: **return** $\mathcal{T}$

Function CreateTreeNode(Set $\mathcal{S}$)
Create new tree node and assign the set's primitives to it, thereby compute bounding box and barycenter on-the-fly:
1: create empty tree node $\mathcal{T}$
2: **for all** $\pi_i \in S$ **do**
3: $\mathcal{T}.primitives \leftarrow \mathcal{T}.primitives \cup \{\pi_i\}$
4: $\mathcal{T}.bbox \leftarrow BoundingBox(\mathcal{T}.bbox, \pi_i)$
5: $\mathcal{T}.weights \leftarrow \mathcal{T}.weights + w_i$
6: $\mathcal{T}.bc \leftarrow \mathcal{T}.bc + w_i \cdot bc_i$
7: **return** $\mathcal{T}$

Function Split(Tree $\mathcal{T}$, bool *recurse*=`true`)
Partition $\mathcal{T}$ into child trees of roughly equal size, corresponding to the 8 octants induced by the barycenter of $\mathcal{T}$:
1: create sets $\mathcal{S}_1 \dots \mathcal{S}_8$ for the 8 octants
2: **for all** $\pi_i \in \mathcal{T}.primitives$ **do**
3: determine $oct \leftarrow Octant(\mathcal{T}.bc, \pi_i)$
4: $\mathcal{S}_{oct} \leftarrow \mathcal{S}_{oct} \cup \{\pi_i\}$
5: **for** $oct \leftarrow 1$ **to** 8 **do**
6: $\mathcal{T}.child_{oct} \leftarrow CreateTreeNode(\mathcal{S}_{oct})$
7: **if** *recurse* **and** $|\mathcal{S}_{oct}| > 1$ **then**
8: $Split(\mathcal{T}.child_{oct})$

vs. *back* side. In the second step, these coordinates are used to partition the node's set of primitives into subsets of roughly similar size. Recursion stops if the node contains no more than some fixed number of elements.

The second step, which we refer to as *Split*, can actually be initiated at two different points of time. Either, right after constructing the root of the tree, the nodes are repeatedly split in some recursive manner until the whole tree is completely constructed (*Full Split*); or the root node is simply being assigned the whole set of primitives and each query later on then triggers splits *On Demand* for any node it needs to descend to.

3.4 A Shortest Distance Query Algorithm

For the specific task of shortest distance computation, we now briefly describe a concrete query procedure. The algorithm starts with a priority queue that just contains the root nodes of the two search trees to be tested. Until the final distance is found, it iteratively extracts the first element (A, B) from the priority queue, refines the query towards the children of one of the two nodes and reinserts the new pairings back into the queue. In each step, the bounding boxes of the two current nodes impose a lower bound on the distance between any primitive contained in the first bounding box and any primitive contained in the second. This bound is taken as the priority value that determines the position in the ordered priority queue. Each time that node is chosen for refinement that features the larger of the two bounding boxes with respect to the enclosed volume, unless in contains only one primitive.

The first time when two nodes with only one primitive are compared, the first primitive-primitive distance is determined. Since this "real" distance can be larger than the bound given by the two corresponding bounding boxes, further elements of the queue are still to be examined. If better matches are found, the distance is updated accordingly. The search terminates as soon as the head of the queue has a priority that is larger than the distance determined so far.

Algorithm 2 Shortest Distance Query

Input: $\mathcal{S}, \mathcal{T}$: two search trees as described above
Output: $(d, \pi_\mathcal{S}, \pi_\mathcal{T})$: shortest distance between $\mathcal{S}$ and $\mathcal{T}$, and the two defining primitives $\pi_\mathcal{S}, \pi_\mathcal{T}$

Initialize return values and priority queue:
initialize $d \leftarrow \infty$
initialize $\pi_\mathcal{S} \leftarrow$ *invalid*, $\pi_\mathcal{T} \leftarrow$ *invalid*
create empty priority queue Q
if $\mathcal{S}$ is not empty **and** $\mathcal{T}$ is not empty **then**
    add $(\mathcal{S}, \mathcal{T}, dist(\mathcal{S}.bbox, \mathcal{T}.bbox))$ to Q

Keep refining head of Q until shortest distance is found:
while Q not empty **do**
    assign $(\mathcal{A}, \mathcal{B}, \alpha) \leftarrow Head(Q)$
    if $d \leq \alpha$ **then**
        return $(d, \pi_\mathcal{S}, \pi_\mathcal{T})$
    if $\mathcal{A}$ contains only one primitive $\pi_\mathcal{A}$
    and $\mathcal{B}$ contains only one primitive $\pi_\mathcal{B}$ **then**
        update $d \leftarrow \alpha$
        update $\pi_\mathcal{S} \leftarrow \pi_\mathcal{A}$, $\pi_\mathcal{T} \leftarrow \pi_\mathcal{B}$
    else
        RefineHead(Q)
return $(d, \pi_\mathcal{S}, \pi_\mathcal{T})$

Function RefineHead(Priority Queue Q)
Selectively refine the query towards one of the two tree nodes stored at the head of Q:
assign $(\mathcal{A}, \mathcal{B}, \alpha) \leftarrow Head(Q)$
RemoveHead(Q)
if B contains exactly one primitive
or $Vol(\mathcal{B}.bbox) < Vol(\mathcal{A}.bbox)$ **then**
    for all $\mathcal{A}_i \in \mathcal{A}.children$ **do**
        let $\alpha_i \leftarrow dist(\mathcal{A}_i.bbox, \mathcal{B}.bbox)$
        add $(\mathcal{A}_i, \mathcal{B}, \alpha_i)$ to Q
else
    for all $\mathcal{B}_i \in \mathcal{B}.children$ **do**
        let $\alpha_i \leftarrow dist(\mathcal{A}.bbox, \mathcal{B}_i.bbox)$
        add $(\mathcal{A}, \mathcal{B}_i, \alpha_i)$ to Q

3.5 Distance Calculation for Geometric Primitives

For the sake of completeness we will now describe how the distance between two primitives can be computed. However, we will stick to the three primitives *point*, *edge* and *triangle*, since other geometric objects are most uncommon for being used in mesh-based representations. Moreover, we will only discuss the computation of the *shortest Euclidean distance* and simply refer to it as *distance*. The algorithmic design for other measures (e. g. p-norm based distances) may follow a very similar scheme, but need to be crafted individually. (Note however, that providing an implementation for distance computation between all admissible pairs of primitive types and for two bounding volumes is all that it takes to establish the generic approach to support the particular distance measure.)

Regarding the admissible pairs of primitive types, let us start with the most elementary setting: The distance between ***two points*** P ***and*** A is simply the length $|PA|$, i. e., the norm of their difference vector.

The distance between ***a point*** P ***and an edge*** AB depends on where the foot of perpendicular from P onto the line through AB is located. If it is in between A and B, then the desired distance is the height of P over the edge AB. Otherwise, the distance is given by the minimum of $|PA|$ and $|PB|$. Whether or not the mentioned foot of perpendicular is indeed located in between A and B, is easily determined by checking if

the angles $\measuredangle PAB$ and $\measuredangle PBA$ are both less than 90° (which just amounts to evaluating the sign of the two corresponding dot-products).

To determine the distance between ***a point*** P ***and a triangle*** $\triangle ABC$, we first identify whether the foot of perpendicular from P onto the supporting plane of $\triangle ABC$ is actually inside or outside of the triangle. To do so, we initially compute the plane normal (e. g. by cross-product). With each of the triangle's three edges, this normal spans a plane orthogonal to $\triangle ABC$, dividing the space into two half-spaces each. For any such plane, one can easily test whether the remaining third triangle corner and the point P are both part of the same half-space (it just needs to compare the outcome of two 3D orientation tests). If this criterion applies to all three planes, then the foot of perpendicular is included in $\triangle ABC$. In this case, the desired distance is the height of P over the plane through A, B, C (which amounts to computing the value of 4×4-determinant). If for at least one of the three tests P and the third triangle corner happen to be on opposite sides of the orthogonal plane, then the distance between P and that particular triangle edge gives the desired result. This distance can be computed as described before.

For the distance between ***two edges*** PQ ***and*** AB we distinguish three different cases. If the edges happen to intersect, then the distance is obviously 0. Otherwise it is the length of the shortest connector for these two edges, for which there can be two possible settings: Both endpoints of this shortest connector are located in the interior of the edge PQ and AB, respectively. Or at least one endpoint coincides with P, Q, A or B. In the former case, the shortest connector is equal to the two edges' common perpendicular, whose length is then the desired distance. In the latter case, the result is given by the minimum over the four distances between one of the points P, Q, A, B and the other edge.

The distance between ***an edge*** PQ ***and a triangle*** $\triangle ABC$ is 0 in case of a real intersection. Otherwise, it is the minimum over the two distances of P to the triangle and Q to the triangle. These two distances can be computed as described just before.

Finally, the distance between ***two triangles*** $\triangle PQR$ ***and*** $\triangle ABC$ is simply the minimum over the six individual distances between each of the edges and the other triangle which it does not belong to (see previous item on how these distances are computed).

The distance between two axis-parallel bounding-boxes A and B is also easily computed. In fact, the three dimensions can be treated separately. For each dimension $u \in \{x, y, z\}$ we just need to test the intervals $[u^A_{\min}, u^A_{\max}]$ and $[u^B_{\min}, u^B_{\max}]$ for overlapping. If they do, the corresponding sub-distance for that dimension is 0. Otherwise it is given by the size of the gap in between the two intervals: $\Delta u = \max\{u^B_{\min} - u^A_{\max}, u^A_{\min} - u^B_{\max}\}$. The desired distance between the two bounding boxes is then simply $dist(A, B) = \|(\Delta x, \Delta y, \Delta z)\|_2$.

Note that the previous paragraphs only describe the basic scheme of how distances between pairs of primitives are computed. In fact, the final implementation is somewhat slightly different to what has just been presented. This algorithmic difference yet only serves optimization purposes. In particular, some elementary calculation are performed only once, for they are simply redundant. Moreover, expensive tests for rather unlikely settings (like coincidence, collinearity, or coplanarity) are deferred to the end, while fast computations for more likely settings are handled first.

3.6 Performance Aspects

Conducting a pure theoretically-driven worst-case runtime analysis on the described approach will lead to rather pessimistic results. In fact, it is possible to construct extremely degenerated synthetic input that forces the algorithm to perform poorly. This kind of analysis, however, is not appropriate to characterize realistic behavior for the setting to be expected in practice.

Due to the organic origin of the data, segmentation boundaries are naturally curved, leading to more or less fine-grained surface meshes. Disproportionally large primitives are hence not to be expected and unfavorable overlapping of bounding volumes will be small. Moreover, our barycenter-based spatial decomposition partitions the given set of primitives at every level into at least two subsets of roughly equal size. The resulting search tree will not be perfectly balanced, but of fairly logarithmic height.

As the spatial search trees are basically well-formed, the expected running time mainly depends on the degree to which the input data itself is degenerated w r. t. the specific measurement task. A mesh representing a perfect sphere would lead to a large number of candidate pairs of primitives to be tested for determining the object's diameter. This extreme setting, however, is not to be expected for real human anatomic structures. Natural variations can be expected that bring out prominent candidate regions for the maximum diameter and early reduce search space.

For determining shortest distances, the worst-case setting would be two objects featuring a large surface patch with nearly the same distance niveau very close to the minimum value. Again, this would lead to a large number of primitive pairs to be tested. Unfortunately, this setting appears somewhat possible in practice, e. g. in terms of two anatomic structures that are

anatomically-dependent on each other. Imagine the skin of human head, for example. Across the superior area of cranium a quite uniform distance between skin and underlying skull can be expected. Although the minimum distance between these two structures may not be of too much clinical relevance, this setting is yet all the more perfect in providing a real world worst-case example. Consequently, we integrated this scenario as challenging input into our test suite. The resulting running times give reason to assume that real-world data feature variations of sufficient magnitude thanks to which most of the "almost-matches" implicitly get sorted out. In fact, for our test data the query time never exceeded a justifiable value.

In conclusion, data obtained from human anatomy may be expected to be not exceedingly critical in terms of the distance computation suffering from running time explosion. Of course, an absolute guarantee cannot be given. However, the issue may easily be dealt with by incorporating some small bound ε of fractional amount (e. g. 0.1 mm) on the minimum difference between two distances values in order to be considered different. This allows the query algorithm to stop, as soon as a preliminary result has been found for which the computed distance is not more than ε away from the priority stated by the head of the queue.

4 Clinically Relevant Applications

In the following, we will discuss which different kinds of medically inspired problems can be treated by our approach. Each of these tasks requires only minimal adjustments to some of the design's exchangeable components. Note that although spatial decomposition technique and bounding volume type may be specifically chosen to optimally suit the application – the particular solution prestented in Section 3.2 ff. should work well for most of the problems. In particular, if different applications do not require dissimilar representation of the input data, one and the same data structure can be used for all of them, which allows sharing data memory and renders multiple preprocessing superfluous.

Augmented shortest distances

The method described so far allows for computing the ***shortest distance*** between two segmented anatomic structures (Figure 7). A comparison of this distance against some fixed positive value or 0 directly indicates a possible ***safety distance violation*** or ***infiltration condition***, respectively. This question can be very crucial, as a tumor that is too close to (or even infiltrates) the basis cranii, for example, is considered inoperable.

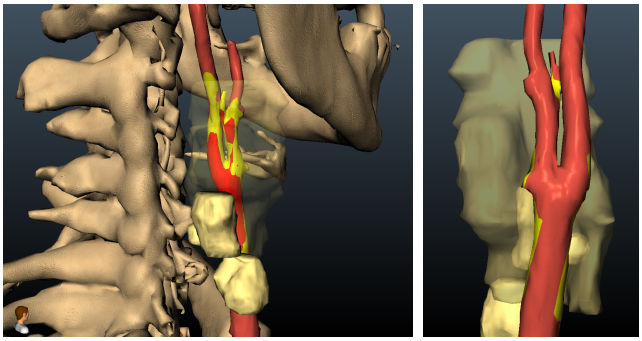

Fig. 6 Color-coding of safety distance violation for arteria carotis (light red) with respect to the contiguous tumor (transparent) and two violation levels (yellow: 3 mm or closer, dark red: 1 mm or closer). *Left*: The color-coding can be used to display contact areas that are usually occluded. *Right*: As only yellow patches of the colorization are visible, this perspective reveals that the tumor obviously does not encase the arteria by more than 50%.

However, we can do more than simply checking for a distance violation condition itself. Since the priority queue provides its front elements in sorted order, we can extract a whole sequence of pairs of primitives with similar distance. In doing so we may determine the ***outline of a contact area*** or ***infiltration boundary*** as well as ***distance isolines***. The examination of such outlines is important in some situations (Figure 6): While tumor tissue that just touches arteria carotis interna is still operable, a resection becomes impossible as soon as the artery is encased by more than 50%.

Single Structure Measures

The meshes which are to be compared do not necessarily have to reflect surfaces of different anatomic structures. Instead, if we were to test a single anatomic structure against its own skeleton, the minimal distance between surface and skeleton can be determined, i. e. the ***minimal inner radius***. This allows not only localization of blood vessel stenosis, for example, but also assessment of respiratory tract constrictions with regard to minimum clearance (Figure 7). Similarly, if for a specific anatomic structure separate segmentations were given, one for the inner and one for the outer surface, one can efficiently determine position and extent of ***minimal wall thickness***.

Clearly, one of the two meshes can also simply be a point in 3-space. In this case, we can determine the minimum distance from that unique point to the given mesh. This allows, for example, to set one anatomic structure fixed (e. g. a tumor or malignant lymph node) and interactively explore distances to arbitrary points on the surface of the other structure (e. g. a blood vessel). This can be much more substantial than the single shortest distance between the two structures alone

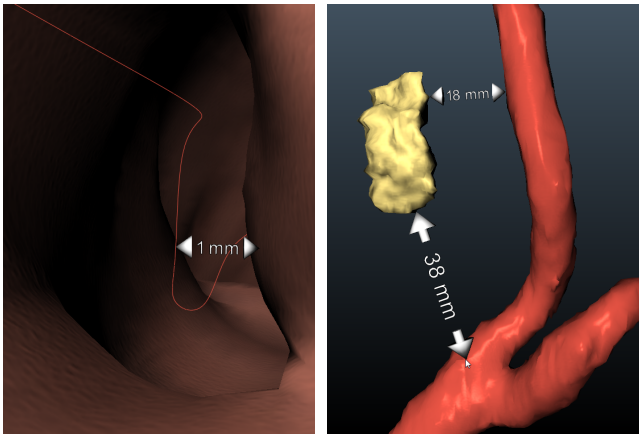

Fig. 7 *Left*: Assessment of a pharynx constriction. Based on the centerline, position and extent of minimum clearance can be computed. *Right*: While the single shortest distance between tumor and arteria carotis is 18 mm, one can also fix the tumor and interactively discover the blood vessel's surface. The perpendicular to the selected "structure of interest" is computed in real-time.

(cf. Figure 7). Since this kind of "one-sided" shortest distance query can be computed extremely fast, it is even suitable for real-time usage. In fact, it is already successfully used for collision detection in interactive virtual endoscopy.

If the minimum determination as part of the query procedure's distance and objective function is replaced everywhere by the maximum, the ***largest distance*** between two anatomic structures will be computed instead. Invoking this altered query with only one single anatomic structure for both arguments, we can determine the ***maximum extent*** of this structure, also called its diameter (Figure 1). This measure is of particular interest in the scope of tumor assessment (TNM classification). A malignant tumor of hypopharynx or oropharynx, for example, is to be classified at least as T2, if its maximum extent is larger than 2 cm, and T3, if it is also larger than 4 cm. Similarly, lymph node affection in this area has to be classified at least N2, if it is solitaire and with maximum extent of more than 3 cm or if it is multiple, and N3 for a malignant lymph node of more than 6 cm diameter. Leaving potential infiltration aside, the tumor in Figure 1, for example, is clearly to be classified as T2 based on the medical expert's measurement, but is close to T3 when considering the automatic measurement.

Combined Input/Output Measures

Input as well as output of the algorithm can also be further combined or enhanced. The shortest distance computation, for example, is not restricted to only two meshes. Instead, by simply adding additional pairs of anatomic structures during initialization of the queue, one may determine which inter-object distances are actually to be considered. This way it is possible to compute ***distances between groups*** of anatomic structures. Hence, one may directly ask for the shortest distance between a group of malign objects (e. g. tumor and affected lymph nodes) and a group of risk structures (e. g. arteria carotis, thyroid cartilage, straight muscles of neck, etc.), instead of checking all relevant pairs of structures individually.

Similarly, also the final output can be post-processed further, thereby using the same method yet another time. By taking the resulting infiltration boundary from above as new input again, one may compute the ***diameter of surface-surface intersections***, for example. In the same way, one may use the maximum extent determined for a given anatomic structure to compute a projection of the structure's mesh to an orthogonal plane. Taking this projection as input again allows computing the ***second direction of maximum extent***. The third can be computed in the very same manner. Note though, that these last approaches involve construction of new geometric primitives, which can be a non-trivial task to be done exactly.

Constrained Measures

All the tasks described so far can also very easily be performed with respect to a specific given orthogonal range, i. e. some axis-parallel bounding box. By additionally comparing every node's associated bounding volume against this user-defined area, one can easily determine ***constrained measures within regions of interest***. This may particularly be relevant for ***local vessel stenosis assessment*** and ***local wall thickness determination***. Also other shape types can be imagined to serve as a region of interest, as for instance spheres, ellipsoids or arbitrary polytopes. Yet, the more complex the corresponding shape, the more expensive it is to perform the required intersection test for a given node of the search tree.

Apart from the spatial location, the measurement can also be constrained with respect to a possible direction. In fact, the distance function can be adapted to measure only with respect to some arbitrary predefined linear subspace. This allows measures like a ***projective extent*** or ***projective distance*** to be computed.

Beyond Euclidean Distances on 3D Meshes

Basically all examples described so far were based on the standard Euclidean metric for 3-space. However, as with the other components of the generic approach, also the integral distance function can be adapted to

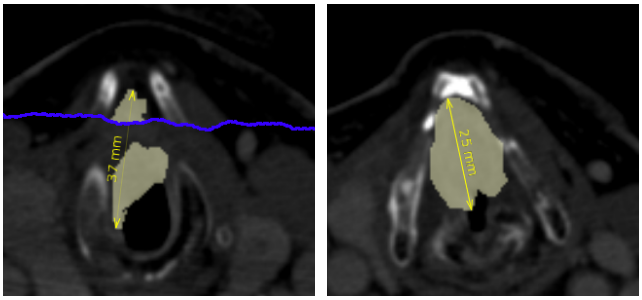

Fig. 8 Automatic measures computed on voxel-based segmentations. *Left*: Maximum extent for a tumor in 3D. The lower image part shows the slice for the starting point, the upper accordingly the slice for the ending point. *Right*: Maximum extent for the same tumor, but constrained to the current selected single slice.

meet new applications. As for instance, we generally spoke about distances *between* two objects, but never explicitly mentioned symmetry as a requirement. And indeed, this criterion is not necessary. One may just as well apply asymmetric distance functions (i. e. *from* one object *to* another) when querying the spatial search tree, as long as the upper/lower bound criterion holds for the bounding volumes. So it is possible to obtain ***asymmetric or weighted measures***.

Another interesting example is the possibility to define distance functions for spaces other than 3D. In particular, one may incorporate the dimension *time*, contained in some medical data. This way, ***time-dependent measures*** for 2D+t or 3D+t can be obtained, as for instance the ***dynamic minimal wall thickness*** or ***dynamic minimum clearance***. Again, these measures can be constrained to a specific region of interest.

Let us give a last note on the initial decision for using mesh-based segmentations. Due to the generic design of the approach, it is actually not necessary to first convert a given voxel-based segmentation into a mesh-based one. In fact, by simply interpreting the coordinates of each voxel as a 3D vertex, one may build the spatial search trees directly on top of the segmentations, thus allowing for a ***voxel-based minimum distance or maximum diameter computation***. It is not even necessary to compute original real-world coordinates, if the distance function used for querying the search trees simply incorporates the dataset's spacing – origin and orientation can be completely neglected.

5 Evaluation

5.1 Data and Experiments

To evaluate our approach, a small test suite was compiled. Two experiments were performed, one using synthetic data and one using real world medical data.

For the first experiment, two sphere meshes were generated by recursively subdividing the triangle faces of an icosahedron. Both spheres were tessellated to the same level $l = 4..8$ and given a common radius of $r = 5$. One sphere was centered at the origin, while the other was displaced by $\Delta x = 11$ along the x-axis. Hence, the two meshes had a pre-defined distance of $d = 1$.

For the second experiment, segmentations of neck structures in terms of surface triangle meshes of different complexity were used as input, which have been provided by our clinical partner. They were created using a medical research tool for manual and semi-automatic segmentation for CT datasets [2,3] and accredited by an authorized medical expert. For selected pairs of these meshes, the shortest distance has been computed.

According to our clinical partner, most relevant for surgical planning are spatial relations between anatomic vs. pathologic structures. This amounts to computing the distance between a structure of moderate size (blood vessel, respiratory tract, bones) and a second structure of quite small size (tumor, lymph node). Since running times for this kind of setting turned out barely measurable, a different selection was made in consultation with our clinical partner. Segmentations of moderate or large mesh size were chosen to show the suitability of the approach for more complex input data. Moreover, with bones vs. skin, a test setting was added that can be considered a kind of worst case scenario regarding the computational cost to be expected. Again, this setting was investigated to verify that our approach is also applicable in difficult problem instances.

For each pair of meshes in both experiments, the shortest distance was computed five times altogether. First, the original implementation of Preim et al. [15] was used, obtaining reference timings. Then, our approach was used to compute the shortest distance again in four different ways. Two variants are due to the way how input data is interpreted: One may reduce each input mesh to its set of vertices like it was done by Preim et al. [15], or one may process the mesh indeed as a set of triangles. The remaining two variants result from the splitting strategy being either *Full Split* or *On Demand*.

For each such combination, CPU time was measured averaged over 20 passes, excluding the time for reading data into memory. The source code was compiled with GCC 4.2.1 at optimization level O2. The experiments were finally performed on a 32-bit Linux system with an Intel Pentium 4 at 3.2 GHz and 1GB RAM.

5.2 Results

The results for the first experiment on synthetic data are given in Table 1. It can be directly extracted that

Table 1 Running times in milliseconds for computing the shortest distance between two tessellated spheres with radius $r = 10$ and a displacement of $\Delta x = 11$ (T_C=*Construction Time*, T_Q=*Query Time*). The number of vertices and triangles refers to a single sphere, while construction time is cumulative for both.

Algorithm →			Reference [15]		Vertices				Triangles			
					Full-Split		On-Demand		Full-Split		On-Demand	
Level	No. of Vertices	No. of Triangles	T_C (ms)	T_Q (ms)	T_C (ms)	T_Q (ms)	T_C (ms)	T_Q (ms)	T_C (ms)	T_Q (ms)	T_C (ms)	T_Q (ms)
4	2562	5120	4.2	3.2	18.0	<0.1	4.9	2.4	37.2	0.2	9.7	9.8
5	10242	20480	12.0	36.4	119.6	<0.1	18.3	16.2	163.1	0.2	32.7	50.0
6	40962	81920	41.6	245.8	572.1	<0.1	91.3	88.0	684.5	0.3	135.3	236.4
7	163842	327680	156.6	12679.4	2639.7	0.2	386.6	394.6	3261.5	0.5	557.9	1036.7
8	655362	1310720	622.0	70085.8	9207.3	0.2	1142.7	1224.1	13468.8	0.8	2288.2	3578.2

Table 2 Running times in milliseconds for computing shortest distances between pairs of anatomic structures (J.=*Jugularis*, C.=*Carotis*, l/r=*left*/*right*), once using the original mesh (upper half of tableau) and once a reduced variant (lower tableau half), based on two separate phases (T_C=*Construction Time*, T_Q=*Query Time*)

Algorithm →			Reference [15]		Vertices				Triangles			
					Full-Split		On-Demand		Full-Split		On-Demand	
Object	No. of Vertices	No. of Triangles	T_C (ms)	T_Q (ms)	T_C (ms)	T_Q (ms)	T_C (ms)	T_Q (ms)	T_C (ms)	T_Q (ms)	T_C (ms)	T_Q (ms)
C. (l)	3620	7232	2.0	27.8	9.6	0.2	1.6	7.2	55.8	11.2	9.2	33.8
C. (r)	3120	6210			7.8		1.6		26.0		7.6	
C. (l)	3620	7232	2.0	29.0	9.6	0.4	2.0	11.4	55.6	8.0	8.6	33.8
J. (l)	3142	6276			6.6		1.4		29.0		4.4	
C. (l)	3620	7232	60.2	2670.6	9.2	1.0	1.6	448.0	56.0	64.4	15.8	875.9
Bones	222079	445348			1579.3		275.8		2240.0		553.0	
C. (l)	1807	3616	1.0	0.8	3.6	<0.1	1.0	3.2	23.2	9.8	3.0	16.2
C. (r)	1558	3104			3.4		0.8		14.2		3.0	
C. (l)	1807	3616	0.8	10.8	4.0	0.8	0.8	5.0	25.4	58.6	3.1	76.4
J. (l)	1567	3137			3.0		0.4		12.6		2.8	
C. (l)	1807	3616	29.4	704.4	4.0	1.0	1.0	229.8	18.0	244.8	3.2	705.4
Bones	110452	222674			741.2		137.8		1250.7		214.0	
Skin	129099	258348	60.0	78600.1	878.5	44.6	159.8	1241.9	3351.0	142.0	507.8	2066.8
Bones	110452	222674			725.2		129.6		1260.1		227.4	

Full Split has high construction time for the benefit of low query time, while *On Demand Split* shows just the opposite behavior.

Regarding the use of vertices vs. triangles, one should expect the latter to be always significantly slower than former one, since computing distances between triangles is much more expensive than between points (cf. Section 3.5). Yet, the *Full Split* column for triangles shows just slightly larger query times than the same column for vertices. This tells us that only very few triangle pairs finally had to be compared. In most of the cases subtrees have successfully been ruled out based on their associated bounding boxes. Moreover, since even for level 8 with hundreds of thousands of vertices/triangles the query never exceeded 2 ms, large parts of the search space must have been discarded at some early hierarchy level already. (Otherwise there were a lot of bounding box pairs to be tested, which in summa would have taken noticeably more time.) Thus, the query times for *On Demand Split* must have almost completely been caused by on-demand-construction of the missing nodes on the search paths in both trees.

This behavior stands in contrast to that of Preim et al.'s approach [15]. While construction time there was always the smallest among all five competitors, the query time grew disproportionally faster with increasing mesh size than for all the other instances. For level 7 with about 16 000 vertices it was already five orders of magnitude slower than its *Full Split* counterpart based on vertices or even on triangles. For level 8 the approach finally turned unusable.

Table 2 provides the results for the second experiment. The upper half of the tableau shows the running times obtained for original data as input. The lower

half in turn shows the results for a reduced variant of the input meshes. Again, *On Demand Split* showed faster construction time, while *Full Split* was significantly faster in performing the query. This time, the results showed the expected behavior that the two triangle-based variants were noticeably slower than their vertex-based two counterparts. Compared to the previous experiment this tells us that in this new setting much more pairs of triangles needed to be tested.

Similar to the first experiment, running times were hardly measurable as long as the number of vertices for each segmentation stayed in about the single-digit thousands. Regarding such input sizes, all five variants can therefore be regarded as efficient. However, for meshes of about 10^5 and more vertices Preim et al.'s approach [15] again showed disproportionally large running times in relation to the remaining variants. In particular, in case of the largest triangle meshes (last table row) all four variants of our approach were significantly faster with respect to cumulated running time (and even more when only query time is considered). Although the ratio is much smaller than in experiment 1, there is still a factor of about 1700 in the query time between our *Full Split* variant for vertices and the approach due to Preim et al. [15]. Even for the triangle *Full Split* we can report our approach to be more than 500 times faster in query time, which underlines the efficiency of this method particularly in presence of large input data.

Both experiments indicate a significant benefit that additional expenditure in the preprocessing phase can offer. For the synthetic data of experiment 1 an almost 2 000 fold speedup was measured, comparing the query time for *On Demand Split* and *Full Split* based on vertices. For triangles, a ratio of up to 500 could be observed. The reason is that *On Demand Split* performs partitioning at the time it is required as the query walks town the search trees. Thereby, each and every primitive already needs to be touched at least once in order to get the (initially unsplit) root node split. In consequence, query time for *On Demand Split* increases at least linear in the size of the input. Indeed, while the number of primitives grows by factor 4 from one level to the next, the corresponding query time grows with a factor of about 4-5 for vertices as well as triangles.

For *Full Split* in contrast, the whole hierarchy has already been built completely before it comes to the first query to be processed. In the best case, only logarithmic many pairs of nodes will be on the search path that need to be processed. This most optimal setting, however, is not guaranteed. Instead, it may turn out that there are quite some more pairs of nodes and primitives to be tested. The larger this number of pairs, the more they will contribute to the overall query time. At the same time, however, the effective ratio between the two types of query time will shrink, since the query times for *On Demand Split* and *Full Split* only differ by the time required for splitting the nodes.

This described behavior could be observed for the real medical data. While in some cases an almost 450 fold speedup could be observed for vertices, in other cases this factor is only 6-7 fold. For triangles, the ratio is generally somewhat smaller, ranging from about 15 to only 2-3. Only once the factor was smaller than 1.5.

An interesting effect can be noticed in the *Full Split* column for triangles. One should expect that running times for original data should always be larger than (or at least as large as) the corresponding running times for the reduced version of the same data. For almost all pairings of structures and algorithms this has indeed been the case. However, when measuring the distance for *Carotis versus Jugularis* as well as *Carotis versus Bones*, using *Full Split* on triangles, the query time was surprisingly smaller for the larger of the two mesh versions. The reason can be found by analyzing the consequence of mesh reduction that was performed to obtain the smaller mesh variant. The positive effect was that the search trees were reduced in their height – however, only by less than one level on average. The adverse effect in turn was that in course of the reduction the data has also smoothed. In consequence, after the reduction step the shortest distance was somewhat less prominent than it was before. In addition, the reduction also resulted in the meshes to consist of larger triangles on average. In consequence, the primitives' corresponding bounding boxes were in the same way larger. As a result, some pairs of primitive could not be ruled out already based on their bounding volumes. These two issues finally led to a larger number of triangle pairs that needed to be tested.

For both experiments, the implementation due to Preim et al. [15] showed disproportionally large query time on sufficiently big input data, compared to the four variants for our method. This performance behavior is mainly caused by two reasons. On the one hand, the splitting strategy which was empirically determined by Preim et al. [15] does not appear to scale well for large input data. This leads to a very large number of element pairs that need to be compared. On the other hand, spheres appear to be a rather inadequate choice for bounding volumes. First, they obviously fit the data less well (see also Figure 3). And second, for distance computation between spheres expensive square root operations are inevitable. For our method, in contrast, we can use squared distances throughout the whole search and just perform a single square root operation after the minimum or maximum has been found.

6 Conclusion and Outlook

We presented a general approach for computing clinically relevant distance measures based on surface triangle meshes. With a variety of examples (Section 4), we demonstrated its wide applicability in diagnosis and treatment planning. The different characteristics can thereby be computed using only one and the same integral spatial data structure.

Concerning the *shortest distance* measure, our approach in empirical tests showed to be more efficient than previous methods from medical application literature. The fast query times also allow for real-time collision detection with simple reference structures. In doing so, the method has already been used successfully as integral part of a software tool in the field of virtual endoscopy.

Apart from that, geometrically exact results are computed. All geometric primitives available in the input are incorporated, instead of following the common practice of simply confining to vertices and disregarding their connectivity. This also allows to identify real penetrations and infiltrations and to measure them. Furthermore, returning not only the simple measured value, but also its defining primitives, improves means for high quality distance visualization, particularly in close-up range. As a prerequisite, however, relevant structures have to be segmented.

In the future, the following points may be addressed. The spatial decomposition technique that has been implemented for shortest distances is fairly simple. The same holds for the choice of using AABB as bounding volumes. The advantage is that the resulting data structure can also be used for all the other applications discussed in Section 4. However, more advanced decomposition techniques and data structures may be implemented to yield better running times, in general or for particular applications only.

Moreover, one may consider a notion of *maximum allowed error* (in contrast to the *geometric exactness* presented), providing means for an early return. The query procedure presented in Section 3.4 stops just when the queue's head shows a priority larger than the minimum effective distance computed so far. Yet, if the result is only demanded to be correct up to some maximum allowed error, the algorithm may stop much earlier, possibly on the level of bounding boxes already. The permitted error may thereby be provided individually by the user prior to measurement, or it may be set to some generally approved global or context-specific fixed value.

References

1. James E. Bobrow. A Direct Minimization Approach for Obtaining the Distance between Convex Polyhedra. *Int. J. Robot. Res.*, 8(3):65–76, 1989.
2. Jana Dornheim, Lars Dornheim, Bernhard Preim, and Gero Strauß. Modellbasierte Segmentierung von Weichgewebestrukturen in CT-Datensätzen de Halses. In Dirk Bartz, Stefan Bohn, and Jürgen Hoffmann, editors, *curac.08 Tagungsband*, pages 197–200, September 2008.
3. Jana Dornheim, Christian Tietjen, Bernhard Preim, Ilka Hertel, and Gero Strauß. Image Analysis and Visualization for the Preoperative Planning of Neck Dissections. In *5th International Conference on Computer Aided Surgery around the Head*, 2008.
4. Ricardo Fabbri, Luciano da Fontoura Costa, Julio C. Torelli, and Odemir Martinez Bruno. 2D Euclidean distance transform algorithms: A comparative survey. *ACM Comput. Surv.*, 40(1), 2008.
5. Andreas Fabri, Geert-Jan Giezeman, Lutz Kettner, Stefan Schirra, and Sven Schönherr. On the design of CGAL, a computational geometry algorithms library. *Softw. Pract. Exper.*, 30(11):1167–1202, 2000.
6. B. Faverjon. Hierarchical Object Models for Efficient Anti-Collision Algorithms. In *Proc. IEEE ICRA*, volume 1, pages 333–340, 1989.
7. E. G. Gilbert and D. W. Johnson. A Fast Procedure for Computing the Distance Between Complex Objects in Three Space. *IEEE J. Robot. Autom.*, 4(2):193–203, April 1988.
8. S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. In *SIGGRAPH '96*, pages 171–180, 1996.
9. Peter Hastreiter, Christof Rezk-Salama, Bernd Tomandl, et al. Fast Analysis of Intracranial Aneurysms Based on Interactive Direct Volume Rendering and CTA. In *MICCAI '98: Proceedings of the First International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 660–669, 1998.
10. Mark W. Jones, J. Andreas Baerentzen, and Milos Sramek. 3D Distance Fields: A Survey of Techniques and Applications. *IEEE Trans. Vis. Comput. Graph.*, 12(4):581–599, 2006.
11. Katsuaki Kawachi and Hiromasa Suzuki. Distance computation between non-convex polyhedra at short range based on discrete voronoi regions. In *In Proceedings of Geometric Modeling and Procesing*, pages 123–128, 2000.
12. Elmar Langetepe and Gabriel Zachmann. *Geometric Data Structures for Computer Graphics.* 2006.
13. Hugo Ledoux. Computing the 3d voronoi diagram robustly: An easy explanation. In *ISVD '07: Proceedings of the 4th International Symposium on Voronoi Diagrams in Science and Engineering*, pages 117–129, 2007.
14. Ming C. Lin and John F. Canny. A Fast Algorithm for Incremental Distance Calculation. In *Proc. IEEE ICRA*, volume 2, pages 1008–1014, 1991.
15. Bernhard Preim and Dirk Bartz. *Visualization in Medicine: Theory, Algorithms, and Applications*, chapter 13 – Measurements in Medical Visualization, pages 313–339. 2007.
16. Sean Quinlan. Efficient distance computation between non-convex objects. In *Proc. IEEE ICRA*, volume 4, pages 3324–3329, 1994.
17. Jörg-Rüdiger Sack and Jorge Urrutia. *Handbook of computational geometry.* 2000.
18. Avneesh Sud, Naga Govindaraju, Russell Gayle, Ilknur Kabul, and Dinesh Manocha. Fast proximity computation among deformable models using discrete Voronoi diagrams. *ACM Trans. Graph.*, 25(3):1144–1153, 2006.